FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Výuková aplikace pro CPM metodu
Student:	Bc. Tomáš Doležálek
Vedoucí:	Ing. Petra Pavlíčková, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2021/22

Pokyny pro vypracování

Výuková aplikace pro matematické modelování na základě metody CPM. Cílem této aplikace je vylepšení výuky matematického modelování a to přímo metody CPM.
1. Určete procesy, které bude aplikace podporovat.
2. Namapujte jednotlivé procesy na funkční požadavky aplikace.
3. Navrhněte grafické rozhraní aplikace.
4. Určete technologie, kterými bude aplikace vytvořena.
5. Připravte scénáře pro základní průchody aplikací.
6. Proveďte implementaci aplikace.
7. Otestujte aplikaci dle testovacích scénářů a opravte případné chyby.
8. Vytvořte uživatelskou a instalační příručku.
9. Proveďte ekonomické zhodnocení navržené aplikace.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 28. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Výuková aplikace pro CPM metodu

Bc. Tomáš Doležálek

Katedra softwarového inženýrství)
Vedoucí práce: Ing. Petra Pavlíčková Ph.D.

27. května 2020

Poděkování

Tímto bych rád poděkoval vedoucí mé diplomové práce Ing. Petře Pavlíčkové Ph.D. za cenné rady a pomoc při zpracování této závěrečné práce. Dále bych chtěl také poděkovat Ing. Pavlu Čejkovi za pomoc při gramatické kontrole práce. V neposlední řadě patří toto poděkování také mé rodině a přítelkyni, za jejich podporu při mém studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 27. května 2020 .....

Odkaz na tuto práci

Doležálek, Tomáš. *Výuková aplikace pro CPM metodu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

V této diplomové práci se čtenář seznámí s jednou ze základních metod časové analýzy projektu a to s metodou kritické cesty. Pro pochopení této metody jsou v diplomové práci rozebrány potřebné základy teorie grafů, které jsou důležité pro pochopení vlastností a tvorby síťového grafu. Metoda kritické cesty je založená na výpočtech v síťovém grafu aktivit projektu. V teoretické části práce je rozebrána metoda kritické cesty včetně příkladů výpočtu a to pro hranově i uzlově definované síťové grafy. V praktické části této diplomové práce se věnuji implementaci webové aplikace pro podporu výuku metody kritické cesty. Praktická část je rozdělena na návrh a analýzu a pak na samotnou implementaci a testování. V návrhové části jsou popsány podporované procesy, analýza funkčních a nefunkčních požadavků a případy užití. Následuje návrh uživatelského rozhraní a volba architektury pro implementaci. V implementační části jsou popsány algoritmy pro průchod síťovým grafem, jednotlivé části aplikace a jejich třídy. Testování zahrnuje heuristickou analýzu pro zhodnocení použitelnosti uživatelského rozhraní a popis základních testovacích scénářů. Nakonec je v práci provedeno ekonomické zhodnocení nákladů na analýzu a vývoj výukové aplikace.

Klíčová slova metoda kritiké cesty, síťový graf, projektové řízení, výuková aplikace, případy užití, návrh uživatelského rozhran

Abstract

This final thesis focuses on critical path method which is one of the basic methods for time analysis of project activities based on its activity network. Theoretical part of this thesis explains essential terms from graph theory which are required to understand the characteristics of activity network diagram and how to construct it. Reader is then introduced to critical path method whose theoretical background is described in detail and the method itself is presented on examples, both arrow diagramming and precedence diagramming methods are shown. Practical section of this thesis focuses on design, analysis, implementation and testing of educationl application that focuses on critical path method. Supported processes are described in the design part. Functional and non functional requirements are gathered and described and a list of use cases is created as well. Prototype of user interface follows using a wireframe form. Lastly the architecture of the application is chosen and described. In the implementation part, the focus is on description of selected parts of application as well as explanation of some graph traversal algorithms which were used. Testing follows with user interface usability analysis using a heuristic analysis. Functionality of application is tested using a set of created test cases. Thesis closes with an economic evaluation of costs related to application design and development.

Keywords critical path method, activity diagram, project management, educational application, use cases, user intefrace design

Obsah

Úvod **1**

1 Cíl práce **3**

2 Teoretická část **5**
2.1 Projektové řízení 5
2.2 Teorie grafů 6
2.3 Síťový graf 7
2.3.1 Konstrukce síťového grafu 8
2.4 CPM - metoda kritické cesty 13
2.4.1 1. fáze - postup vpřed 15
2.4.2 2. fáze - zpětný průchod 15
2.4.3 3. fáze - výpočet časových rezerv 16
2.4.4 Výpočet CPM - hranově definovaný graf 17
2.4.5 Výpočet CPM - uzlově definovaný graf 20
2.5 Výuková aplikace 25

3 Praktická část **27**
3.1 Analýza a návrh aplikace 27
3.1.1 Podporované procesy 27
3.1.2 Funkční požadavky 28
3.1.2.1 Síťový graf 28
3.1.2.2 Výpočet metody CPM 29
3.1.2.3 Tutoriály 29
3.1.3 Nefunkční požadavky 30
3.1.3.1 Aplikace běžící ve webovém prohlížeči 30
3.1.4 Případy užití 31
3.1.4.1 UC1: Vytvoření nového síťového grafu 31
3.1.4.2 UC2: Přidání uzlu do síťového grafu 31
3.1.4.3 UC3: Editace uzlu síťového grafu 32

3.1.4.4 UC4: Odebrání uzlu ze síťového grafu 33
3.1.4.5 UC5: Přidání hrany do síťového grafu 34
3.1.4.6 UC6: Editace hrany síťového grafu 35
3.1.4.7 UC7: Odebrání hrany síťového grafu 36
3.1.4.8 UC8: Vyznačení kritického prvku grafu 37
3.1.4.9 UC9: Zrušení vyznačení kritického prvku grafu 38
3.1.4.10 UC10: Uložení síťového grafu 39
3.1.4.11 UC11: Načtení síťového grafu 39
3.1.4.12 UC12: Spuštění tutoriálu 40
3.1.4.13 UC13: Procházení tutoriálu 41
3.1.4.14 UC14: Ukončení tutoriálu 41
3.1.5 Návrh uživatelského rozhraní 42
3.1.5.1 Task list 42
3.1.5.2 Úvodní obrazovka 44
3.1.5.3 Obrazovka Tutoriály 44
3.1.5.4 Obrazovka Editoru síťových grafů 46
3.2 Architektura 48

4 Implementace **51**
4.1 Zvolené technologie 51
4.2 Implementace 52
4.2.1 Třídy View, Controller a Model 53
4.2.2 Prohlížeč tutoriálů - třídy TutorialVIew, TutorialController a TutorialModel 53
4.2.3 Editor grafů - třídy GraphView, GraphController a GraphModel 55
4.2.4 Validace síťového grafu 56
4.2.5 Výpočty v síťovém grafu 56
4.2.6 Animace průchodu výpočtu metodou CPM v síťovém grafu 57
4.2.7 Obsah tutoriálů 59

5 Testování **61**
5.1 Nielsenova heuristická analýza 61
5.2 Testování funkcionality 63
5.2.0.1 TC1: Spuštění a načtení aplikace 64
5.2.0.2 TC2: Spuštění tutoriálu 64
5.2.0.3 TC3: Procházení stránek tutoriálu 65
5.2.0.4 TC4: Spuštění editoru grafů 66
5.2.0.5 TC5: Přidání a odebrání uzlu síťového grafu . 67
5.2.0.6 TC6: Editace uzlu síťového grafu 68
5.2.0.7 TC7: Přidání a odebrání hrany síťového grafu 69
5.2.0.8 TC8: Editace hodnot hrany síťového grafu . . 71

5.2.0.9 TC9: Označení kritických a nekritických prvků v grafu 73
5.2.0.10 TC10: Uložení a načtení síťového grafu 74
5.2.0.11 Testovací zařízení 77
5.2.0.12 Webové prohlížeče 78
5.3 Ekonomické zhodnocení 78

Závěr **81**

Literatura **83**

A Seznam použitých zkratek **85**

B Ukázky aplikace **87**

C Obsah přiloženého CD **91**

Seznam obrázků

2.1 Activity-on-node diagram aktivit z tabulky 2.1 9
2.2 Modelování vztahů mezi aktivitami (zleva): start-to-start, finish-to-finish, start-to-finish 10
2.3 Znázornění reálné a fiktivní aktivity 10
2.4 Použití fiktivní aktivity pro modelování souběžných činností 11
2.5 Použití fiktivní aktivity pro normalizaci síťového grafu 11
2.6 Použití fiktivní aktivity pro oddělení nezávislých činností 12
2.7 Activity-on-arrow diagram aktivit z tabulky 2.1 12
2.8 Activity-on-arrow diagram aktivit z tabulky 2.1 po zjednodušení a očíslování uzlů 13
2.9 Uzel i v Activity-on-arrow diagramu 17
2.10 Activity-on-arrow diagram pro příklad výpočtu CPM 17
2.11 Dopředný průchod: výpočet nejdřívějšího možného počátku provádění aktivit 18
2.12 Zpětný průchod: výpočet nejzazšího přípustného konce provádění aktivit 19
2.13 Síťový graf aktivit doplněný o časové rezervy (levé číslo v rámečku představuje celkovou časovou rezervu, pravé volnou) a s červeně vyznečnou kritickou cestou 20
2.14 Uzel v activity-on-node diagramu 20
2.15 Activity-on-node diagram pro příklad výpočtu CPM 21
2.16 Výsledné hodnoty EF a ES aktivit po dopředném průchodu 23
2.17 Výsledné hodnoty LF a LS aktivit po zpětném průchodu 24
2.18 Síťový graf s vypočtenými časovými rezervami jednotlivých aktivit. Tyto rezervy jsou uvedeny v rámečku pod jednotlivými aktivitami, číslo vlevo představuje celkovou a vpravo volnou časovou rezervu. Kritická cesta je zvýrazněna červeně. 25

3.1 Úvodní obrazovka aplikace 45
3.2 Podnabídka sekce tutoriály 45

3.3 Obrazovka se stránkou tutoriálu 46
3.4 Obrazovka s editorem síťového grafu 47
3.5 Kontextové menu při kliknutí na uzel 47
3.6 Formulář pro editaci hodnot v uzlu 48

B.1 Ukázka aplikace - stránka tutoriálu 87
B.2 Ukázka aplikace - editor grafů 88
B.3 Ukázka aplikace - ilustrace výpočtu 89

Seznam tabulek

2.1 Tabulka aktivit 8

5.1 Test case 1 - Spuštění a načtení aplikace 64
5.2 Test case 2 - Spuštění tutoriálu 64
5.3 Test case 3 - procházení tutoriálu. 65
5.4 Test case 4 - Spuštění editoru grafů 66
5.5 Test case 5 - Přidání a odebrání uzlu síťového grafu 67
5.6 Test case 6 - Editace uzlu grafu 68
5.7 Test case 7 - Přidání hrany do uzlu grafu 70
5.8 Test case 8 - Přidání hrany do uzlu grafu 71
5.9 Test case 9 - Označení kritičnosti a odebrání označení prvků . . . 73
5.10 Test case 10 - Uložení a načtení síťového grafu 75

Úvod

Metoda kritické cesty představuje jednu ze základních metod síťové analýzy v projektovém řízení. Znalost této metody je tedy důležitá pro všechny studenty vysokých škol, kteří studují obory zabývající se projektovým řízením. Pro podporu výuky metody kritické cesty bude v rámci této závěrečné práce vytvořena výuková aplikace, která umožní studentům lépe pochopit tuto metodu a rovnou si ji vyzkoušet při tvorbě síťových grafů a výpočtech v nich. Čtenář této závěrečné práce se nejdříve seznámí s teoretickými základy, které jsou potřeba pro pochopení metody kritické cesty. Bude se jednat o základy projektového řízení a vysvětlení důležitých pojmů z teorie grafů nutných pro pochopení vlastnotí síťového grafu. Dále bude v práci vysvětleno, jakým způsobem lze síťový graf aktivit projektu vytvořit. Nakonec budou položeny teoretické základy pro metodu kritické cesty a bude vysvětlena její důležitost v projektovém řízení. Poté bude metoda představena na příkladu, a to v obou hlavních formách modelování síťových grafů: hranově definované a uzlově definované.

Výuková aplikace bude vytvořena formou webové aplikace, která poběží ve webovém prohlížeči přímo na koncových zařízeních. Tvorba této aplikace bude popsána v praktické části diplomové práce. Budou zde rozebrány jednotlivé části vývoje aplikace, od návrhu a analýzy, přes volbu architektury, popis samotné implementace, zvolených technologií a knihoven, a nakonec testování. Návrhová část bude obsahovat popis jednotlivých podporovaných procesů, sběr požadavků, návrh uživatelského rozhraní a popis případů užití. K otestování aplikace vytvořím a použiji testovací scénáře, zaměřím se také na otestování použitelnosti uživatelského rozhraní. Jelikož jde o aplikaci určenou pro běh ve webovém prohlížeči, bude nutné ověřit funkčnost aplikace v různých prohlížečích.

Vytvořená aplikace podpoří výuku metody kritické cesty. Součástí aplikace budou tutoriály na nichž si student osvojí znalosti nutné pro pochopení metody kritické cesty, síťových grafů a nutné teorie grafů. Aplikace také umožní

studentům vytvářet si vlastní síťové grafy a bude provádět validaci hodnot vypočtených metodou kritické cesty. Uživatelé tak získají nástroj, který jim umožní pochopit a procvičit si metodu kritické cesty.

Kapitola 1

Cíl práce

Hlavním cílem této závěrečné práce je navrhnout a implementovat výukovou aplikaci, která umožním posluchačům bakalářského studia vyzkoušet si a lépe pochopit metodu kritické cesty. Aplikace by měla umožnit studentům pochopit principy metody kritické cesty prostřednictvím příkladů a umožnit aplikovat tuto metodu na příkladech vlastních. Metoda kritické cesty (CPM) se používá pro výpočet doby trvání projektu. Dílčími cíli této diplomové práce jsou seznámení čtenáře s teoretickými základy pro pochopení metody CPM, zejména potřebnou teorií grafů a vysvětlení síťového grafu, popis a vysvětlení metody kritické cesty a její aplikování na zvolených příkladech. Dalším cílem pak je provést analýzu výukové aplikace, implementaci s použitím dohodnutých technologií a otestování této aplikace. Nakonec bude provedeno zhodnocení této aplikace.

Kapitola 2

Teoretická část

V teoretické části této diplomové práce se pokusím čtenáři přiblížit a vysvětlit metodu kritické cesty. Čtenář bude seznámen s potřebnými základy teorie grafů a s vysvětlením pojmu síťový graf. Na příkladech bude ilustrováno, jakým způsobem lze síťový graf vytvořit a jakou úlohu hraje v projektovém řízení. Po zavedení potřebných pojmů z teorie grafů a seznámení se síťovým grafem bude čtenáři vysvětlena podstata metody kritické cesty a ilustrováno její použití na příkladu.

2.1 Projektové řízení

Projekt je činnost, kterou zpravidla vykonává více lidí s cílem dosáhnout určeného výsledku. Projekt pomáhá lidem vložit tuto činnost do nějakého rámce a ten pak řídit. Jde o uzavřenou posloupnost kroků, neboli činností, které představují základní prvek, z nichž je projekt složen. Je to jedinečný proces řízených a koordinovaných činností s jasně danými daty zahájení a ukončení. Tyto činnosti jsou prováděny k dosažení cíle projektu. Cíl, případně rozsah projektu musí mít jasně specifikované a definované požadavky a měl by přinést zainteresovaným stranám užitek. Projekt jako takový je pak omezen těmito prostředky: časem, zdroji a náklady[1]. Metoda CPM se zaměřuje na odhad a výpočet doby trvání projektu, a také na kontrolu, že tato odhadnutá doba bude splněna. Z těchto důvodů je důležité správně odhadnout dobu trvání jednotlivých činností v projektu. Takové odhady lze provést vícero způsoby; například může jít o odhad expertem, zkušenosti z předchozích projektů a jejich vyhodnocení, parametrický odhad (u softwaru například podle počtu obrazovek) nebo lze využít různých norem a standardů.

Jelikož je projekt unikátním a často i poměrně komplikovaným procesem, je potřeba jej řídit, čímž se zabývá podobor managementu zvaný projektové řízení. Životní cyklus projektu se skládá ze tří částí: předprojektová fáze, projektová fáze a postprojektová fáze. Projektové řízení je s projektem spjato ve všech těchto fázích, nicméně v každé má trochu jiný úkol.

Před samotným spuštěním projektu je provedena tzv. předprojektová fáze. Ta je zaměřena na vyhodnocení proveditelnosti projektu, cílů a přínosů. Projekty vždy existují v rámci nějakého kontextu a spotřebovávají limitované zdroje, a proto je nutné v této fázi provést vyhodnocení projektu, na jehož základě se lze rozhodnout, který projekt bude skutečně zahájen a kdy. K provedení tohoto vyhodnocení poskytuje projektové řízení řadu nástrojů a postupů. Jde například o SWOT analýzu, jež nahlíží na projekt jak z vnitřního prostředí organizace, tak i z pohledu vnějších faktorů, které mohou mít vliv na úspěšnost projektu.[2]

V projektové fázi se zabýváme realizací projektu. Tato fáze v sobě zahrnuje přípravy projektu, plánování, realizaci projektu a ukončení. V této fázi se zaměřujeme hlavně na kontrolu včasného provádění a dokončení činností, kontrolu dodržení harmonogramu, sledování a předcházení rizik. V této fázi projektu se také mohou objevit změnové požadavky, které musíme správně řídit, abychom neohrozii úspěch projektu. Cílem této fáze je uhlídat náklady a rozsah projektu, aby mohl být dokončen v potřebném termínu. Někdy je také nutné slevit z požadované funkcionality, případně odložit práce na ní až po dokončení projektu, aby důležitější a klíčové výstupy projektu byly dokončeny před dokončením projektu.

Poprojektová fáze je spjata především s analýzou projektu a jeho vyhodnocením. V rámci ukončení projektu dochází k předání projektu a jeho akceptaci, kdežto při vyhodnocení analyzujeme, zda bylo dosaženo všech cílů, jestli splněné cíle projektu přinesly očekávaný užitek, analyzujeme ekonomické ukazatele projektu, apod. V rámci tohoto vyhodnocení také sepíšeme tzv. ponaučení (lessons learned), které by mělo pomoci při řízení budoucích projektů. Toto ponaučení obsahuje například řešení problémů, které se při projektu vyskytly, proč byly způsobeny a jak by bylo možné se jim vyhnout. Lze ho také použít pro zlepšení odhadů obdobných budoucích projektů. Metoda kritické cesty vstupuje do projektového řízení ve fázi přípravy a plánování, kde ji lze využít pro odhad délky doby trvání projektu a pak dále při realizaci projektu ke kontrole, zda tato délka bude dodržena. V postprojektové fázi se již metoda kritické cesty neuplatňuje.

2.2 Teorie grafů

Metoda kritické cesty je úzce spjata se síťovým grafem jednotlivých aktivit v projektu. Z tohoto důvodu je vhodné zavést a vysvětlit několik pojmů z teorie grafů. Graf lze obecně definovat jako neprázdnou množinu uzlů a množinu hran, které vždy spojují pouze dva uzly. Matematicky je pak graf definován následovně[3]:

$$G = (V, E); \forall e \in E : e = (a, b); a, b \in V$$

Kde V představuje množinu uzlů a E množinu hran, které jsou vzájemně disjunktní. Dvojice uzlů $(a, b) \in E(G)$ je v grafu spojena hranou, tato hrana

tedy vede z uzlu a do uzlu b a zároveň také z b do a. Naproti tomu *orientovaný graf* je speciálním případem grafu, kde hrany grafu nazýváme orientované hrany nebo jednoduše šipky. Orientovaná hrana $(a, b) \in E(G)$ orientovaného grafu G pak určuje odkud kam hrana vede, v tomto případě z uzlu a do uzlu b.

Pro pochopení metody kritické cesty je rovněž vhodné znát pojmy pro cestování v grafech. Začneme sledem, *sled* představuje konečnou posloupnost uzlů a hran v grafu, kde uzly a hrany na sebe vzájemně navazují. Pokud se ve sledu neopakují žádné hrany, nazýváme ho *tahem*. Nyní můžeme definovat pojem *cesta*, jde totiž o tah, kde se neopakují žádné uzly. Nakonec zavedeme pojem cyklus, neboli kružnice. Jde o tah, který má stejný počáteční a koncový uzel. Tyto pojmy lze také definovat pro orientovaný graf. Například cesta v orientovaném grafu se nazývá *orientovaná cesty* a jde o cestu grafem, která při pohybu mezi uzly respektuje orientaci hran.

Acyklický graf, je takový graf, který neobsahuje žádný cyklus. Jelikož už umíme cestovat grafem, můžeme si také vysvětlit souvislost grafu. Souvislost grafu v podstatě říká, jestli je graf roztržený nebo ne. *Graf je souvislý*, pokud pro každé dva vrcholy a a b existuje v tomto grafu také cesta z a do b.[4]

V rámci metody CPM pracujeme s tzv. ohodnoceným grafem. Ohodnocený graf je takový graf, u kterého provedeme přiřazení hodnoty buď hranám a nebo uzlům grafu. Hranám a uzlům, lze také přiřadit podmínky[4]. Pro potřeby této práce se spokojíme s číselným ohodnocením.

2.3 Síťový graf

Síťový graf představuje matematický model projektu a znázorňuje jednotlivé činnosti a návaznosti mezi nimi[5]. Jedná se o orientovaný, acyklický a souvislý graf. Při jeho sestavování musíme dodržet následující pravidla[5]:

- Síťový graf má právě jeden koncový a jeden počáteční uzel
- Do každého uzlu s výjimkou počátečního musí vést alespoň jedna hrana
- Z každého uzlu s výjimkou koncového následuje aspoň jedna hrana
- Libovolné dva uzly v grafu spojuje nejvýše jedna hrana (nejedná se tedy o multigraf)

Souvislost síťového grafu může být problematická pokud uvažujeme definici souvislého grafu z předchozí kapitoly. Síťový graf lze chápat jako orientovaný strom s jedním kořenovým uzlem (počáteční uzel) a jedním listem (koncový uzel). Takový graf je souvislý pokud do každého uzlu vede orientovaná cesta z počátečního uzlu a z každého uzlu vede orientovaná cesta do koncového uzlu.

Rozlišujeme dva druhy síťových grafu, podle toho jakým způsobem reprezentuje aktivity (činnosti) v projektu. Síťový graf může být hranově definovaný nebo uzlově definovaný. U hranově definovaného grafu představují hrany jednotlivé aktivity, uzly pak představují milníky v projektu - dokončení předchozích a počátek nových aktivit. V případě uzlově definovaného grafu představují uzly jednotlivé aktivity v projektu. Hrany pak v tomto případě pouze představují přechod od dokončení jedné aktivity k zahájení aktivity další. V obou případech platí, že práce na následujících aktivitách mohou být započaty až poté, kdy jsou všechny předcházející aktivity dokončeny. To je v grafu znázorněno tím, že do uzlů může vést více hran.

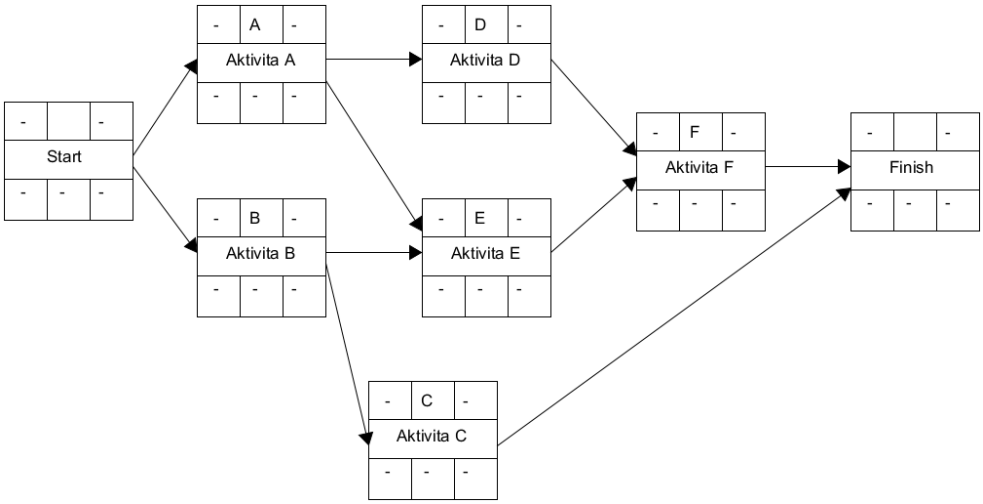
2.3.1 Konstrukce síťového grafu

Jak už bylo zmíněno, při konstrukci síťového grafu máme možnost zvolit si hranově definovaný graf (activity-on-arrow) nebo uzlově definovaný graf (activity-on-node). Nyní si ukážeme, jak takový síťový graf sestrojit. Pro sestrojení síťového grafu potřebujeme seznam aktivit a jejich návazností, který můžeme získat například pomocí WBS - work breakdown structure. Uvažujme následující tabulku aktivit:

Tabulka 2.1: Tabulka aktivit

Aktivita	Předchůdce
A	-
B	-
C	B
D	A
E	A,B
F	D,E

Uzlově definovaný (activity-on-node) síťový graf těchto aktivit (neboli činností) můžeme vytvořit celkem intuitivně. Z tabulky 2.1 vidíme, že projekt může začít provedením aktivit A a B. Tyto aktivity jsou na sobě nezávislé, lze je zpracovat paralelně a mohlo by se tedy jednat o počáteční uzly výsledného grafu. Jelikož však víme, že síťový graf nemůže mít více než jeden počáteční uzel, pomůžeme si vložením fiktivního uzlu, který nepředstavuje aktivitu v projektu a tedy nespotřebuje žádný čas. Z tohoto fiktivního uzlu povedou orientované hrany do aktivit A a B. Podle tabulky postupně přidáváme uzly reprezentující jednotlivé aktivity a hrany představující závislosti mezi aktivitami. Nakonec vložíme další fiktivní uzel reprezentující dokončení projektu, do něhož povedou orientované hrany z aktivit, které nemají žádné následovníky, v tomto případě jde o aktivity C a F. Výsledný diagram je zobrazen na následujícím obrázku 2.1

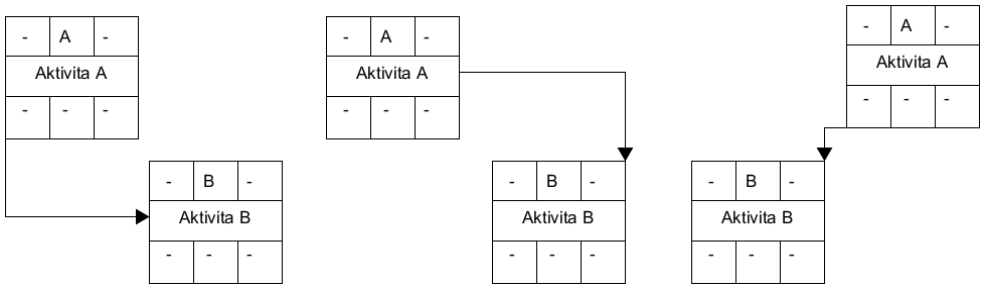


Obrázek 2.1: Activity-on-node diagram aktivit z tabulky 2.1

Všechny aktivity na grafu zobrazeném na obrázku 2.1 mohou být zahájeny až když předchozí aktivity v grafu byly dokončeny, můžeme se však setkat i s dalšími způsoby modelování vztahů mezi aktivitami [6]:

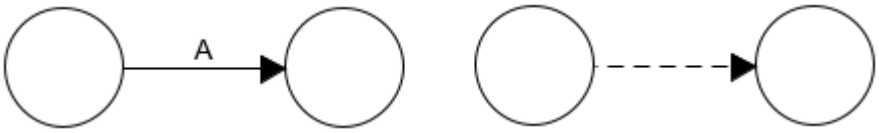
- Finish-to-start (zahájení aktivity čeká na dokončení předchozí aktivity)
- Finish-to-finish (aktivita nemůže být dokončena dříve než předchozí aktivita)
- Start-to-start (aktivita nemůže být zahájena dříve než je zahájena aktivita předchozí)
- Start-to-finish (aktivita nemůže být dokončena dříve než je zahájena aktivita předchozí)

V síťovém grafu na obrázku 2.1 jsou všechny aktivity typu finish-to-start. Pokud bychom měli u jednotlivých aktivit přiřazeny doby trvání aktivit, jednoduše bychom je vepsali do jednotlivých uzlů grafu. To v tomto příkladu nebylo znázorněno, jelikož v rámci metody CPM se do uzlů vpisují i další hodnoty, což bude ukázáno v kapitole věnující se CPM. U tohoto diagramu lze také provést ohodnocení hran tzv. lagem. Lag představuje čas, který musí uplynout, než je dokončen pohyb po hraně. Pokud máme hranu typu finish-to-start mezi dvěma aktivitami ohodnocenou lagem ve výši pěti dnů, nemůže následující aktivita začít dříve než pět dnů po skončení aktivity předchozí.



Obrázek 2.2: Modelování vztahů mezi aktivitami (zleva): start-to-start, finish-to-finish, start-to-finish

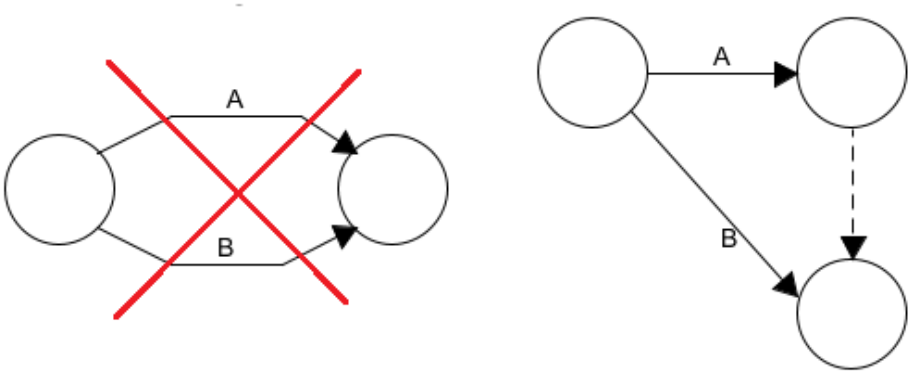
Na rozdíl od uzlově definovaného síťového grafu jsou aktivity u hranově definovaného grafu představovány právě hranami diagramu. Uzly pak představují body, ve kterých jsou předchozí aktivity již dokončeny a práce na následujících aktivitách zahájeny. To poněkud komplikuje vytváření síťového grafu zejména co se týče zachycení všech závislostí. V grafu se budou nově nacházet dva typy hran: hrany představující reálné aktivity projektu a hrany představující fiktivní aktivity. Tyto fiktivní aktivity se používají právě pro zachycení závislostí mezi jednotlivými aktivitami. Fiktivní aktivity typicky nespotřebovávají žádný čas a prostředky, lze je také použít pro znázornění čekacích aktivit, v takovém případě fiktivní aktivita čas spotřebovává.



Obrázek 2.3: Znázornění reálné a fiktivní aktivity

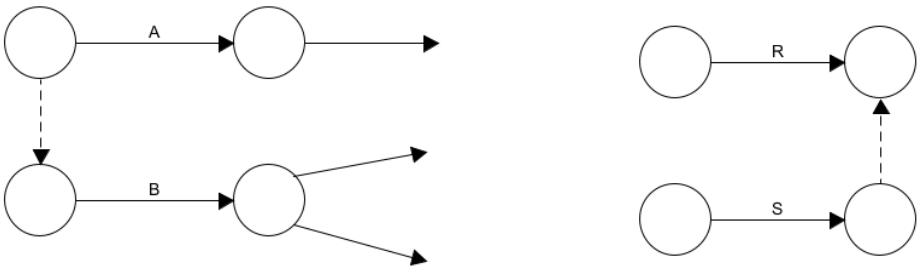
V následujících příkladech se budu věnovat použití fiktivních činnosti při konstrukci hranově definovaného síťového grafu [5]:

- Fiktivní aktivitu lze použít pro oddělení souběžných činností. Případ vlevo (obr. 2.4) by vedl na multigraf, což je v rozporu s definicí síťového grafu.



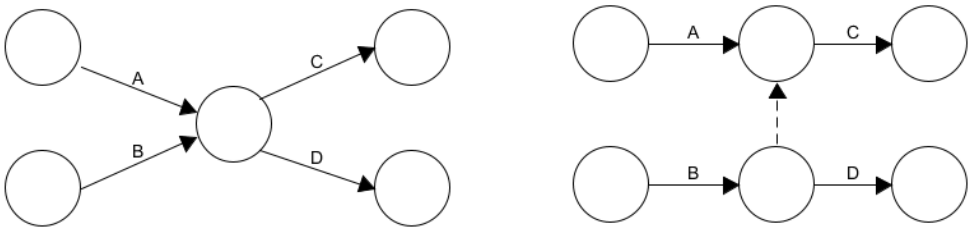
Obrázek 2.4: Použití fiktivní aktivity pro modelování souběžných činností

- Obdobně můžeme použít fiktivní aktivitu pro zajištění existence pouze jednoho počátečního a koncového uzlu.



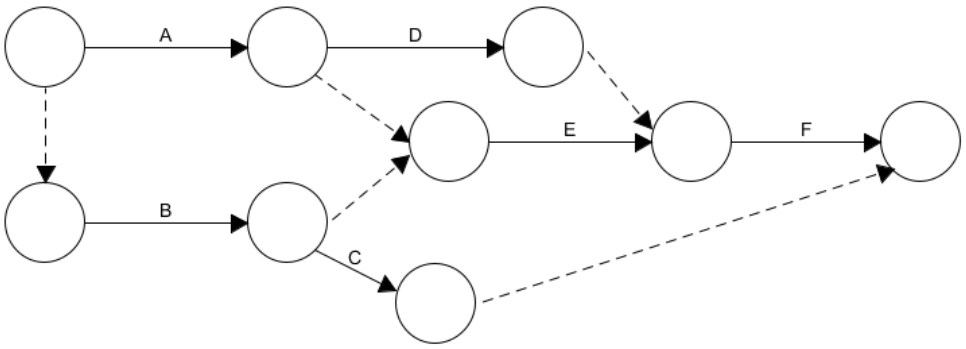
Obrázek 2.5: Použití fiktivní aktivity pro normalizaci síťového grafu

- V následujícím příkladu použijeme fiktivní aktivitu pro oddělení závislých a nezávislých aktivit. V grafu vlevo jsou aktivity C a D závislé na dokončení aktivit A a B. Vpravo jsme použili fiktivní aktivitu k oddělení aktivity D, která je tak závislá pouze na dokončení aktivity B.



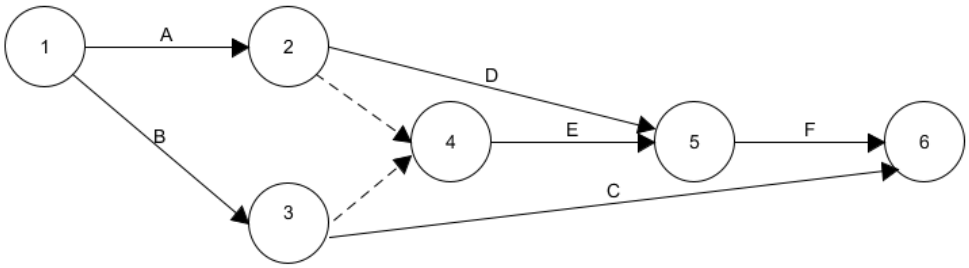
Obrázek 2.6: Použití fiktivní aktivity pro oddělení nezávislých činností

Nyní můžeme přistoupit k vytvoření síťového diagramu aktivit z tabulky 2.1. Každá aktivita v tomto diagramu je jednoznačně dána jejím počátečním a koncovým uzlem. Při vytváření hranově definovaného síťového grafu můžeme začít těmi aktivitami, které nemají žádné předchůdce. Tyto aktivity povedou z počátečního uzlu síťového grafu. Pokud budeme dále postupovat způsobem, že pro každou aktivitu načrtneme zároveň koncový a počáteční uzel, mohl by síťový graf vypadat následovně jako na obrázku 2.7. V tomto grafu byla rovněž provedena i normalizace koncových a počátečních uzlů pomocí fiktivních aktivit.



Obrázek 2.7: Activity-on-arrow diagram aktivit z tabulky 2.1

Vytvořený graf na obrázku 2.7 sice reprezentuje dané aktivity a splňuje požadavky na síťový graf, nicméně se v něm nachází nadbytečné fiktivní aktivity, což by mohlo případného uživatele takového grafu mást a komplikovat práci s ním. V následujícím kroku provedeme zjednodušení síťového grafu eliminací nadbytečných fiktivních aktivit.



Obrázek 2.8: Activity-on-arrow diagram aktivit z tabulky 2.1 po zjednodušení a očíslování uzlů

Obrázek 2.8 představuje zjednodušení diagramu vyobrazeném na obrázku 2.7. Dále bylo v tomto diagramu provedeno očíslování jednotlivých uzlů, které ale nepředstavuje dobu trvání aktivit. Očíslování je provedeno tak, že první uzel je ohodnocen číslem jedna. Další uzly očíslujeme pouze tehdy, jsou-li všechny předchozí uzly daného uzlu očíslovány. Očíslování uzlů lze použít pro kontrolu grafu, kde hrany musí vždy vést od uzlů s nižším číslem do uzlů s číslem vyšším. Tím se zároveň odhalí existence cyků v grafu, cyklu totiž vždy bude existovat orientovaná hrana vedoucí z uzlu s vyšším pořadovým číslem do uzlu s číslem nižším. Narozdíl od tabulky 2.1 se můžeme setkat se zadáním činností právě pomocí čísel uzlů, mezi kterými povedou hrany představující dané aktivity.

2.4 CPM - metoda kritické cesty

Metoda kritické cesty byla vyvinuta v roce 1957 firmou E.I. du Pont de Numours, ve které se předchůdce CPM používal již během projektu Manhattan. Výzkumníci této americké firmy vyvinuli metodu CPM s využitím activity-on-arrow diagramů a poprvé byla metoda kritické cesty vyzkoušena při uzavírání provozů například z důvodu údržby. Ve stejném období byla vyvinuta podobná metoda i v Evropě, kde britská CEGB - státní instituce odpovědná za výrobu a distribuci elektřiny - vyvinula metodu pro zjištění tzv. "nejdelší neredukovatelné sekvenci událostí". S využitím této metody se CEGB podařilo ušetřit 42 % času při údržbě elektrárny [7].

Metoda kritické cesty, z anglického *critical path method*, je jednou ze základních deterministických metod síťové analýzy projektu. Používá se k určení doby trvání projektu na základě doby trvání jednotlivých aktivit a jejich návaznostech. Cílem metody CPM je v síťovém grafu aktivit projektu nalézt tzv. *kritickou cestu*. Kritická cesta je taková cesta od počátečního uzlu až do koncového uzlu grafu, jejíž doba trvání je ze všech takovýchto cest nejdelší. Síťový graf projektu vždy obsahuje jednu či více kritických cest. Význam kritické cesty spočívá v tom, že projekt nemůže být dokončen dříve, než

je doba trvání kritické cesty. Je tedy zjevné, že pro využití této metody je důležité dobře odhadnout délku trvání jednotlivých aktivit. To tuto metodu zvýhodňuje v případě, že je možné dobu trvání jednotlivých aktivit určit deterministicky nebo s vysokou přesností. Zpoždění aktivit na kritické cestě vždy povede k prodloužení doby trvání projektu, obdobně zrychlené provedení aktivit na kritické cestě může vést ke dřívějšímu dokončení projektu, zde ale záleží na konkrétním sítovém diagramu. Dobu trvání projektu neovlivňují pouze aktivity na kritické cestě, ale i aktivity nekritické neboli dílčí. Právě vhodným naplánováním dílčích aktivit lze docílit zkrácení doby projektu. Síťovou analýzu projektu lze tedy použít pro výpočet doby trvání projektu, pro sestavení plánu provádění jednotlivých aktivit a pro kontrolu dodržování tohoto časového harmonogramu. Z tohoto popisu metody CPM je zjevné, že se zaměřuje hlavně časovou analýzu projektu. V reálných projektech se ale setkáváme i s omezením ostatních zdrojů, která musíme respektovat při vytváření prováděcího plánu aktivit.

Při síťové analýze projektu je potřeba projít následujícími kroky:

- Vytvoření seznamu aktivit (činností), které budou v rámci projektu provedeny.
- Časový odhad doby provedení jednotlivých činností.
- Vytvoření síťového grafu na základě závislostí jednotlivých činností.
- Ohodnocení síťového grafu o doby trvání činností.
- Výpočet dílčích termínů činností (počátek, konec, rezerva atd.).
- Nalezení a analýza kritické cesty.
- Výpočet a rozmístění časových rezerv.

Pro každou činnost v projektu metoda CPM odvozuje následující časové charakteristiky ([8],s. 191):

- Nejdřívější možný začátek provádění činnosti: Aktivita nemůže začít dříve, než je tento bod v čase od začátku projektu, je závislá na dokončení předchozích aktivit.
- Nejdříve možné dokončení provádění činnosti: Aktivita nemůže být dokončena dříve, než je tento bod v čase od začátku projektu.
- Nejzazší možný začátek činnosti: Aktivita nesmí začít později, než je tento termín, jinak dojde ke zpoždění provádění následujících činností.
- Nejzazší přípustná doba dokončení: Aktivita nesmí být dokončena později, než je tento termín, jinak dojde ke zpoždění provádění následujících činností.

Samotnou metodu CPM pak můžeme rozdělit na 3 fáze ([8] s. 192 - 199). Pro následující definice uvažujme hranově orientovaný síťový graf.

2.4.1 1. fáze - postup vpřed

1. fáze je zaměřena na tzv. dopředný průchod síťovým grafem. V této fázi je proveden výpočet nejdřívějšího možného začátku a dokončení provádění jednotlivých aktivit. Dopředný průchod tedy začíná v počátečním uzlu síťového grafu. Uvažujme uzel hranově orientovaného grafu na obrázku 2.9. Symbol t_i^0 představuje nejdřívější možný začátek provádění všech činností vycházejících z uzlu i. Výpočet této hodnoty pro uzel j je dán následujícím vzorcem ([8], s. 192):

$$t_j^0 = \max_i(t_i^0 + y_{ij}) \tag{2.1}$$

Symbol y_{ij} představuje dobu provádění aktivity na hraně mezi uzly i a j. Tento vztah lze pochopit tak, že pro určení nejdříve možného počátku aktivit začínajících v uzlu j se použije maximum z nejdříve možného dokončení činností, které do tohoto uzlu vedou. Výpočet v 1. fázi započneme v počátečním uzlu síťového grafu kterému přiřadíme $t_1^0 = 0$. Postupujeme dále po grafu a vypočteme nejdřívější možný začátek činností vycházejících z těchto uzlů, dokud se nedostaneme do uzlu koncového. Pokud si poslední uzel v síťovém grafu označíme indexem n, pak hodnota t_n^0 představuje nejkratší možný čas, za který je možné projekt dokončit. To zjevně vyplývá z podstaty samotného algoritmu CPM, kde v posledním uzlu grafu vybíráme maximum z nejdřívěho možného dokončení všech cest v grafu. Hodnota t_n^0 představuje délku *kritické cesty.*

2.4.2 2. fáze - zpětný průchod

2. fáze se týká výpočtu nejzazších přípustných začátků a konců provádění činností. V této fázi metody CPM provedeme zpětných průchod síťovým grafem, který začneme v jeho posledním uzlu. Pohybujeme se proti směru orientovaných hran v grafu. Nejpozději přípustný konec prováděnní činností t_i^1 je dán následujícím vzorcem ([8], s. 193):

$$t_i^1 = \min_i(t_j^1 - y_{ij}) \tag{2.2}$$

Ze vzorce tedy vyplývá, že pro určení nejzazšího konce aktivit, které vedou do tohoto uzlu a jsou tedy v tomto bodě síťového grafu již dokončeny, se vybere minimum z nejzazšího začátku provádění aktivit, které v tomto uzlu začínají.

Při samotném výpočtu začneme v posledním uzlu grafu u_n, kde musíme určit hodnotu t_n^1. Tato hodnota představuje nejzazší přípustný konec všech aktivit v projektu a tedy i nejpozdější přípustný termín konce všech aktivit na projektu. Hodnotu t_n^1 tedy můžeme zvolit dle potřeb projektu, musí ale platit, že $t_n^1 \geq t_n^0$, přičemž při výpočtech se často setkáváme s tím, že jsou

si obě hodnoty rovny. Dále postupujeme grafem proti směru orientovaných hran s využitím vzorce 2.2. Nakonec lze provést částečnou kontrolu výpočtu v počátečním uzlu grafu, kde by se hodnota t_1^1 měla rovnat rozdílu $t_n^1 - t_n^0$.

2.4.3 3. fáze - výpočet časových rezerv

3. fáze se pak zabývá výpočtem rezerv pro jednotlivé aktivity. Časová rezerva představuje dobu o kterou může být daná aktivita opožděna, aniž by došlo ke zpoždění celkového dokončení projektu. Rezervy rozdělujeme na dva druhy:

- celková časová rezerva
- volná časová rezerva

Rozdíl je v tom, že celková časová rezerva dané aktivity je čas, po který se může opozdit dokončení této aktivity aniž by došlo ke zpoždění celého projektu. Oproti tomu volná časová rezerva představuje čas, o který se může aktivita zpozdit, aniž by došlo ke zpoždění nejdřívějšího možného počátku provádění navazujících aktivit. V této fázi metody CPM máme jednotlivé aktivity představované hranami h_{ij} určené časovým oknem pro jejich realizaci. Toto okno je dáno nejdříve možným počátkem zahájení činností aktivity t_i^0 vypočteném v uzlu, ze kterého hrana vychází a nejzazší přípustnou dobou dokončení aktivity t_j^1, která je vypočtená v uzlu, do kterého hrana vede. Jelikož známe dobu trvání jednotlivých aktivit y_{ij}, můžeme provést výpočet časových rezerv. *Celkovou časovou rezervu* CR_{ij} můžeme vypočíst pomocí následujícího vzorce:

$$CR_{ij} = t_j^1 - t_i^0 - y_{ij} \tag{2.3}$$

Význam celkové časové rezervy spočívá v tom, že se díky ní dají určit kritické činnosti v projektu. Pokud si označíme nejdříve možný začátek provedení činností koncového uzlu síťového grafu jako $T = t_n^0$ a plánovanou dobu projektu $T_{pl} = t_n^1$, pak kritické aktivity jsou právě ty aktivity, jejichž časová rezerva se rovná rozdílu $T_{pl} - T$. Pokud jsme si ve 2. fázi metody CPM zvolili $T = T_{pl}$ (tj. $t_n^0 = t_n^1$), tedy délku kritické cesty jako plánovanou délku projektu, jsou kritické aktivity ty aktivity, jejichž celková časová rezerva je rovna nule.

Volnou časovou rezervu lze vypočíst jako rozdíl mezi nejdříve možnou dobou zahájení následujících činností a nejdříve možnou dobou dokončení provádění činnosti. Tato časová rezerva se může u činností v projektu objevit pouze tehdy, pokud více činností má stejného následovníka. Výpočet volné časové rezervy aktivity VR_{ij} je dán následujícím vzorcem:

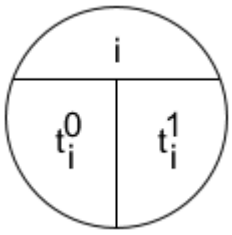
$$VR_{ij} = t_j^0 - (t_i^0 + y_{ij}) \tag{2.4}$$

Pro časové rezervy vždy platí následující vztah: $CR_{ij} \geq VR_{ij}$, volná časová rezerva aktivity nikdy nemůže být delší než její celková časová rezerva. Časové

rezervy můžeme chápat také jako cenný zdroj při projektovém řízení. Dají se použít například pro vykrytí rizik a neočekávaných událostí.

2.4.4 Výpočet CPM - hranově definovaný graf

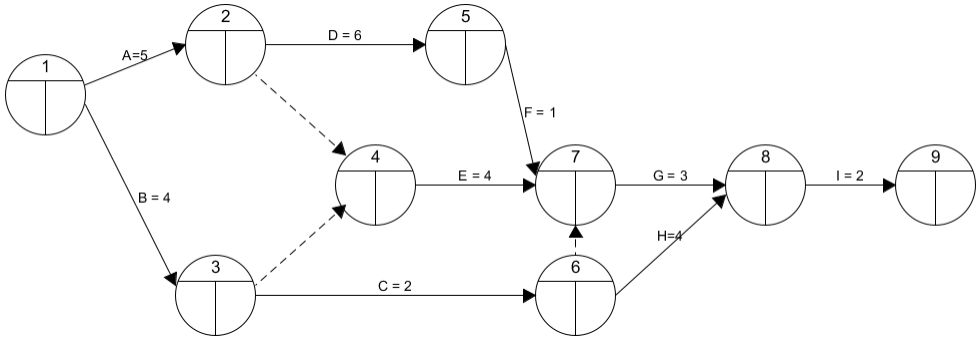
V tomto příkladu bude předveden výpočet metody CPM na hranově definovaném grafu (activity-on-arrow). Pro použití metody CPM budeme při výpočtu uvažovat následující podobu uzlu v síťovém grafu 2.9.



- i představuje index uzlu
- t_i^0 nejdříve možný začátek činností vycházejících z tohoto uzlu
- t_i^1 nejzazší možný konec provádění činností vedoucích do tohoto uzlu

Obrázek 2.9: Uzel i v Activity-on-arrow diagramu

Na následujícím obrázku 2.10 je zobrazen síťový graf, který bude použit pro příklad výpočtu metody CPM. V grafu jsou očíslovány jednotlivé uzly a proběhlo také ohodnocení hran o délky trvání jednotlivých aktivit. Pro větší názornost jsou aktivity označeny písmeny.

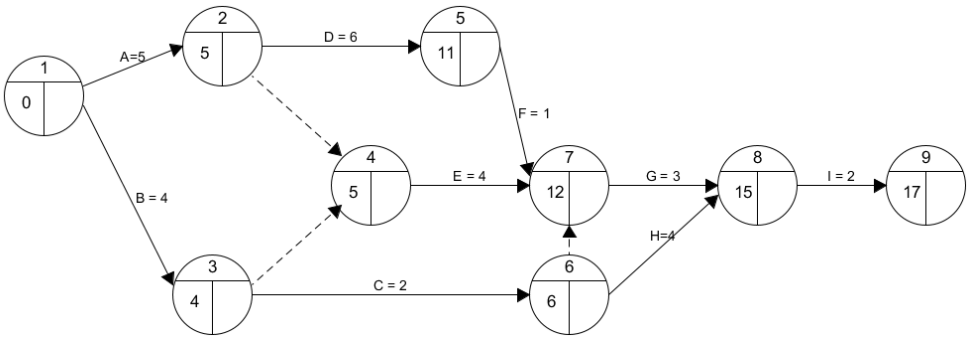


Obrázek 2.10: Activity-on-arrow diagram pro příklad výpočtu CPM

1. fáze - postup vpřed V této fázi provádíme dopředný výpočet v síťovém grafu a vypočítáváme nejdříve možné počátky aktivit pro jednotlivé uzly (hodnoty t_i^0). Postup tedy začneme v prvním uzlu grafu dosazením hodnoty $t_1^0 = 0$. Pokračujeme průchod grafem za využití vztahu 2.1:

- 1. uzel: $t_1^0 = 0$
- 2. uzel: $t_2^0 = t_1^0 + y_{10} = 0 + 5$
- 3. uzel: $t_3^0 = t_1^0 + y_{12} = 0 + 4$
- 4. uzel: $t_4^0 = \max(t_2^0 + 0, t_3^0 + 0) = \max(5, 4) = 5$ - do tohoto uzlu vstupují dvě hrany fiktivních aktivit, které mají nulovou délku trvání
- ... (obdobným způsobem projdeme zbytek grafu)
- 7. uzel: $t_7^0 = \max(t_4^0 + y_{47}, t_5^0 + y_{57}, t_6^0 + 0) = 12$

Síťový graf 2.10 bude po dopředném průchodu vyplněn následovně dle obrázku 2.11:



Obrázek 2.11: Dopředný průchod: výpočet nejdřívějšího možného počátku provádění aktivit

Hodnota nejdříve možného začátku zahájení činností v poslední uzlu představuje i nejkratší dobu, za kterou lze provést všechny aktivity v projektu. Aktivity na tomto projektu lze tedy dokončit nejdříve za 17 dní.

2. fáze - postup vzad Po dokončení první fáze je možné přejít k výpočtu nejzazšího přípustného dokončení provádění aktivit. Vypočítáváme tedy hodnoty t_i^1 pro každý uzel v grafu. Postup začínáme v koncovém uzlu grafu dosazením hodnoty $t_9^1 = t_9^0$, pak procházíme graf proti směru orientovaných hran a pro výpočet využíváme vztahu 2.2:

- 9. (konečný) uzel: $t_9^1 = t_9^0$

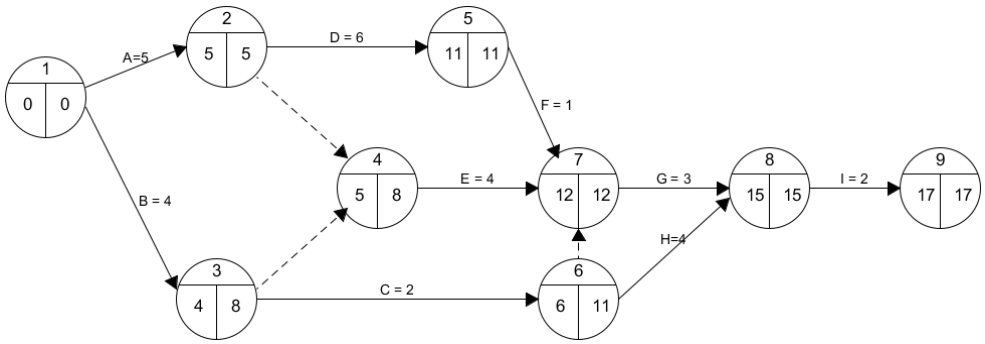
- 8. uzel: $t_8^1 = t_9^1 - y_{89} = 17 - 2 = 15$, do 8. uzlu vedou dvě orientované hrany z uzlů 6. a 7., musíme nejdřív provést výpočet t_7^1 pro 7. uzel grafu, protože tento výsledek potřebujeme pro výpočet t_6^1 v 6. uzlu grafu, pokračujeme tedy následovně dle obrázku 2.12:

- 7. uzel: $t_7^1 = t_8^1 - y_{78} = 12$

- 6. uzel: $t_6^1 = \min(t_7^1 - 0, t_8^1 - y_{68}) = \min(12, 15 - 4) = 11$

- ... (obdobným způsobem projdeme zbytek grafu)

Nakonec provedeme kontrolu $t_1^1 - t_1^0 = t_9^1 - t_9^0$, pokud by tato rovnost neplatila, vznikla při výpočtu chyba. Pokud rovnost platí, nutně to neznamená, že byl výpočet proveden správně!



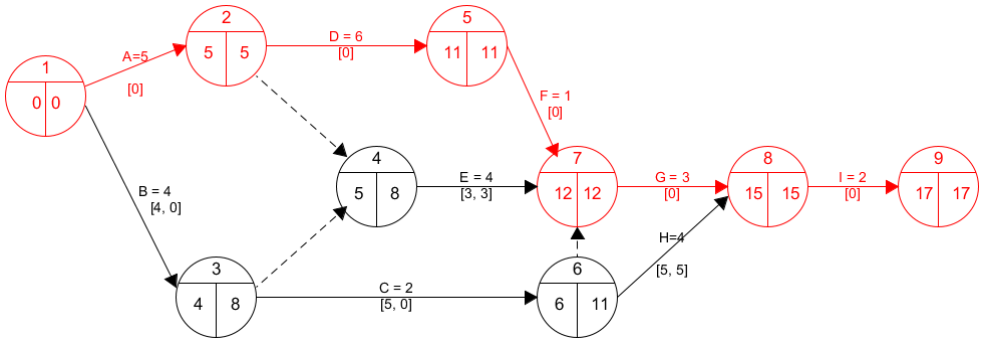
Obrázek 2.12: Zpětný průchod: výpočet nejzazšího přípustného konce provádění aktivit

3. fáze - výpočet rezerv Nyní přejdeme k výpočtu časových rezerv a určení kritické cesty. Jelikož jsme u koncového uzlu grafu zvolili $t_9^1 = t_9^0$, budou kritické aktivity mít nulovou časovou rezervu. Pro výpočet použijeme vzorce 2.3 a 2.4:

- Aktivita A (y_{12}): $CR_{12} = t_2^1 - t_1^1 - y_{12} = 5 - 0 - 5 = 0$, aktivita A je tedy kritická aktivita

- Aktivita B (y_{13}): $CR_{13} = t_3^1 - t_1^1 - y_{13} = 8 - 0 - 4 = 4$, aktivita B není kritickou aktivitou a její zpoždění až o 4 dny nepovede na zpoždění celého projektu. Můžeme spočítat i volnou časovou rezervu $V_{13} = t_3^0 - t_1^0 - y_{ij} = 4 - 4 = 0$, aktivita B nemá žádnou volnou časovou rezervu, její zpoždění by tedy zpozdilo nejdříve možný počátek provádění návazných činností.

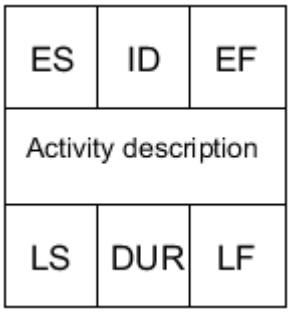
- ... (stejným způsobem provedeme výpočet pro zbylé aktivity)



Obrázek 2.13: Síťový graf aktivit doplněný o časové rezervy (levé číslo v rámečku představuje celkovou časovou rezervu, pravé volnou) a s červeně vyznečnou kritickou cestou

2.4.5 Výpočet CPM - uzlově definovaný graf

V předchozí části této závěrečné práce bylo ukázáno, jak lze aplikovat metody CPM na hranově definovaný síťový graf. Nyní bude výpočet předveden na uzlově definovaném grafu (activity-on-node), k výpočtu použijeme síťový graf ekvivalentní k hranově definovanému grafu 2.10, který byl použit v předchozí části. Seznámíme se také s anglickými názvy používanými pro označení jednotlivých vypočtených hodnot. Připomeňme si, že v uzlově definovaném grafu jsou aktivity znázorněny uzly grafu. Pro konstrukci síťového grafu byl použit uzel s následujícím rozmístěním hodnot:



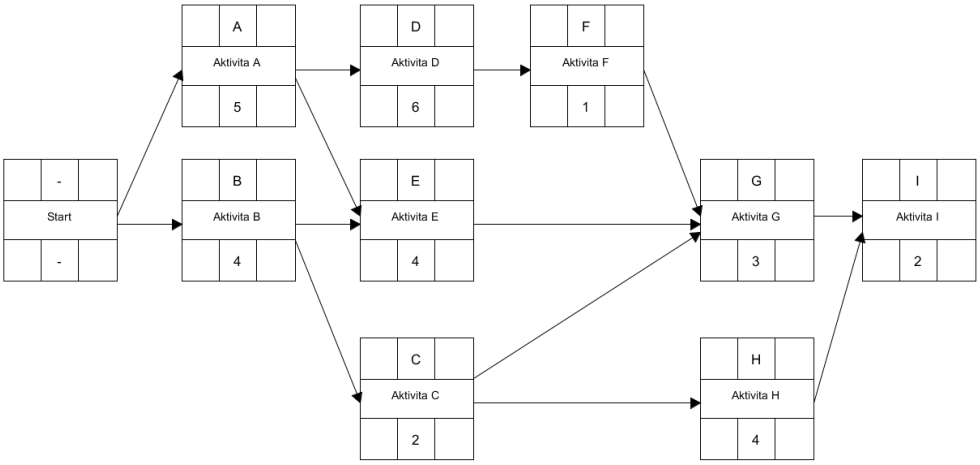
Obrázek 2.14: Uzel v activity-on-node diagramu

Než přejdeme k samotnému výpočtu, je třeba si vysvětlit jednotlivé zkratky použité v tomto uzlu 2.14:

- ES (early start): nejdřívější možný začátek provádění činnosti
- EF (early finish): nejdříve možné dokončení provádění činnosti

- LS (late start): nejzazší možný začátek činnosti
- LF (late finish): nejzazší přípustná doba dokončení činnosti
- DUR (duration): doba potřebná k provedení činnosti
- ID: identifikátor činnosti
- Activity description: popis činnosti

V activity-on-node grafech se také můžeme setkat s uzly, které v sobě mají zaintegrovány i časové rezervy. Všechna čísla, kterými ohodnotíme uzel síťového grafu, se pojí pouze s konkrétní aktivitiou, kterou uzel představuje.



Obrázek 2.15: Activity-on-node diagram pro příklad výpočtu CPM

1. fáze - dopředný průchod V této fázi vypočteme hodnoty ES a EF pro všechny uzly v grafu. Zde při výpočtech záleží jestli jako hodnotu ES u prvních aktivit v síťovém grafu zvolíme $ES = 1$ nebo $ES = 0$. Pokud zvolíme druhou možnost (tzv. nulová metoda), hodnotu EF_a aktivity a vypočteme jednoduše jako součet nejdříve možného začátku provádění činnosti a doby trvání této činnosti, platí tedy následující vztah: $EF_a = ES_a + DUR_a$. Hodnotu ES_b uzlu b vypočteme jako maximum z hodnot EF přímých předchůdců uzlu:

$$ES_b = \max_a(EF_a); (ab) \in E(G) \tag{2.5}$$

Výše uvedený vztah 2.5 lze použít pouze tehdy, je-li hodnota ES u počátečních aktivit rovna nule. Pokud by byla hodnota ES u počátečního uzlu nastavena na $ES = 1$, pak pro hodnotu nejdříve možného dokončení aktivit EF je potřeba výpočet upravit následovně: $EF_a = ES_a - 1 + DUR_a$. To lze pochopit tak, že

při zahájení prací na počátečních aktivitách v projektu, bylo aktivitami v projektu spotřebováno nula dnů. Hodnota ES uzlu v tomto případě představuje, kolik souhrně času spotřebovaly předchozí aktivity. Pokud bychom zvolili jako hodnotu $ES = 1$ u počátečních aktivit, bylo by potřeba pro výpočet hodnoty ES následujících aktivit přidat k maximu EF předchozích aktivit právě tuto jedničku, vztah by tedy vypadal následovně:

$$ES_b = \max_a(EF_a) + 1; (ab) \in E(G). \tag{2.6}$$

V tomto případě hodnota ES v uzlu aktivity představuje souhrný čas spotřebovaný předchozími aktivitami a zároveň ještě den navíc, který spotřebovala aktivita samotná.

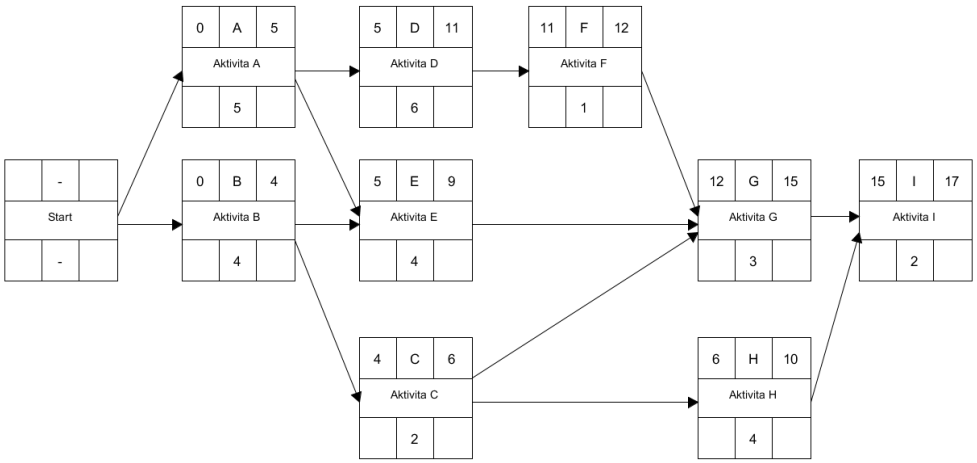
Při dopředném průchodu uzlově definovaného grafu postupujeme podobně jako u hranově definovaného grafu. Začneme tedy ve startovním uzlu grafu. Zde nemusíme nic vyplňovat, takže přejdeme rovnou do počátečních aktivit. V tomto případě jde o aktivity A a B, zde zvolíme jako ES hodnotu nula. Pro každý uzel v grafu vypočítáme hodnoty ES a EF:

- aktivita A: $ES_A = 0$, $EF_A = ES_A + DUR_A = 5$
- aktivita B: $ES_B = 0$, $EF_B = ES_B + DUR_B = 4$
- aktivita C: $ES_C = EF_B = 4$, $EF_C = ES_C + DUR_C = 4 + 2 = 6$, pro hodnotu ES uzlu C jsme použili hodnotu EF uzlu B, aktivita B je přímým předchůdcem aktivity C
- aktivita E: $ES_E = \max(EF_A, EF_B) = 5$, $EF_E = ES_E + DUR_E = 5 + 4 = 9$, pro hodnotu ES uzlu E jsme použili maximum z hodnot EF přímých předchůdců této aktivity, tedy aktivit A a B
- ... (obdobným způsobem projdeme zbytek grafu)

Dopředný průchod uzlově definovaným grafem je znázorněn na obrázku 2.16. Hodnota EF_I poslední aktivity v grafu rovněž představuje celkovou dobu, za kterou lze nejdříve dokončit práce na všech aktivitách v projektu.

2. fáze - postup vzad Ve 2. fázi výpočtu provedeme zpětný postup síťovým grafem. Začneme tedy v koncovém uzlu grafu, což je uzel aktivity I. Zde je potřeba určit hodnotu nejzazšího přípustného dokončení aktivity LF. Obdobně jako u příkladu s hranově definovaným grafem jsem zvolil hodnotu $LF = EF$. Pro výpočet hodnoty nejzazšího možného počátku provádění aktivity EF použijeme následující vztah: $EF_u = LF_u - DUR_u$. Při pohybu proti směru orientovaných hran se přenáší hodnota LS z následujících aktivit do hodnoty LF předchůdců. Zde opět musíme zohlednit volbu hodnoty ES v 1. fázi, pokud bylo zvoleno ES rovno nule u počátečních aktivit, pak platí následující vztah:

$$LF_a = \min_b(LS_b); (ab) \in E(G) \tag{2.7}$$

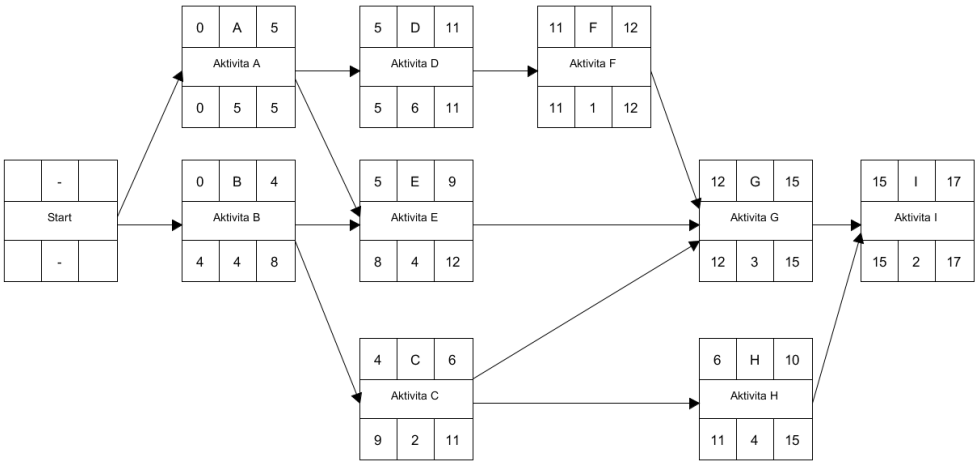


Obrázek 2.16: Výsledné hodnoty EF a ES aktivit po dopředném průchodu

Pokud jsme při dopředném průchodu použili tzv. jedničkovou metodu ($ES = 1$ u počátečních aktivit), je potřeba tuto jedničku také přičíst a upravit vztah pro výpočet nejzazšího možného počátku provádění aktivity $EF_a = LF_a - DUR + 1$. Obdobným způsobem také upravíme vztah pro výpočet nejzazšího přípustného dokončení předcházejících aktivit: $LF_a = \min_b(LS_b) - 1; (ab) \in E(G)$, zde jsme jedničku museli naopak odečíst. Zpětný průchod síťovým grafem zahájíme v aktivitě I:

- aktivita I: $LF_I = EF_I = 17$, $LS_I = LF_I - DUR_I = 15$
- aktivita H: $LF_H = LS_I = 15$, $LS_H = LF_H - DUR_H = 11$
- aktivita G: $LF_G = LS_I = 15$, $LS_G = LF_G - DUR_G = 12$
- aktivita C: $LF_C = \min(LS_G, LS_H) = \min(12, 11) = 11$, $LS_C = LF_C - DUR_C = 9$, pro výpočet hodnot aktivit C jsme potřebovali znát hodnoty vypočtené pro obě následovnické aktivity H a G. Výpočet nemůže pokračovat aktivitou B, protože nebyla ještě zpracována aktivita E.
- ... (obdobným způsobem projdeme zbytek grafu)

Výsledek po dokončení druhé fáze metody CPM v activity-on-node diagramu ilustruje následující obrázek 2.17. Nyní bychom mohli jednoduše určit kritické aktivity, v případě uzlově definovaného grafu jde o aktivity, pro které platí $LF = EF$.



Obrázek 2.17: Výsledné hodnoty LF a LS aktivit po zpětném průchodu

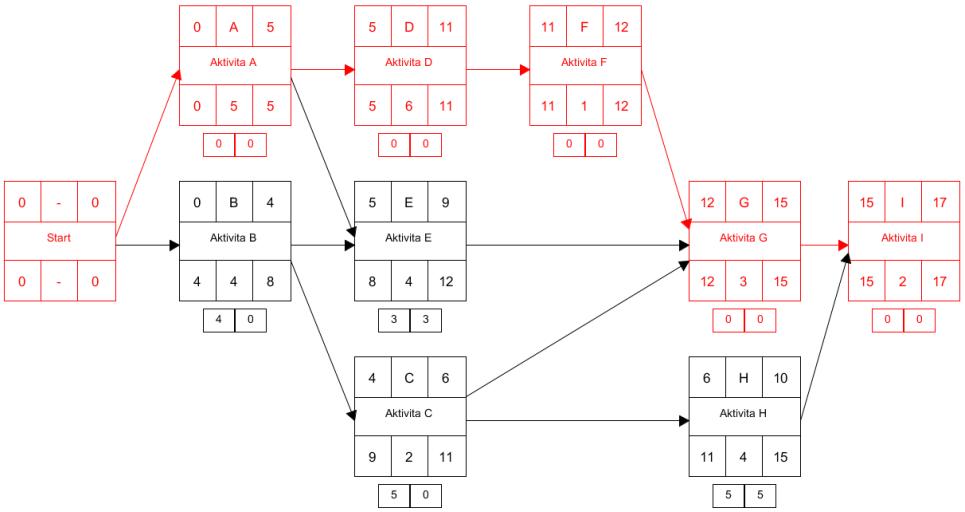
3. fáze - výpočet rezerv (float) V této fázi provedeme výpočet časových rezerv a vyznačení kritických aktivit do síťového grafu. Časové rezervy budeme v tomto případě označovat jako *float* (jde o běžně používaný termín v anglicky psaných textech, někdy také bývá označován jako *slack*), celková časová rezerva ponese označení TF (total float) a volná časová rezerva FF (free float). Výpočet *celkové časové rezervy* je v tomto případě přímočarý:

$$TF_a = LF_a - EF_a \tag{2.8}$$

Oproti tomu výpočet *volné časové rezervy* se poněkud komplikuje. Jak již bylo zmíněno v definici, volná časová rezerva představuje nejvyšší možné zpoždění aktivity, při kterém bude dodržen nejdříve možný počátek provádění aktivit následujících. Při výpočtu tedy musíme brát na zřetel hodnoty ES následovníků aktivity a zvolit minimum z těchto hodnot. Výpočet tedy vypadá následovně:

$$FF_a = \min_b(ES_b) - LS_a; ab \in E(G) \tag{2.9}$$

Opět si musíme dát pozor na to, jakou hodnotu ES jsme použili u počátečních aktivit na projektu, výše uvedený vztah platí pro hodnotu $ES = 0$, pro hodnotu $ES = 1$ vztah upravíme stejně jako v předchozích případech, tedy $FF_a = \min_b(ES_b) - 1 - LS_a; ab \in E(G)$.



Obrázek 2.18: Síťový graf s vypočtenými časovými rezervami jednotlivých aktivit. Tyto rezervy jsou uvedeny v rámečku pod jednotlivými aktivitami, číslo vlevo představuje celkovou a vpravo volnou časovou rezervu. Kritická cesta je zvýrazněna červeně.

2.5 Výuková aplikace

Výukové aplikace jsou součástí moderního přístupu k vyučování nazývaného interaktivní výuka. Interaktivní výuka opouští běžný model vyučování, kde většinou učitel přednáší látku a žák ji pasivně přijímá. V interaktivním modelu výuku mají větší zastoupení informační technologie a elektronická zařízení. Příkladem mohou být například interaktivní tabule, elektronické učebnice, výukové aplikace apod. Cílem této metody je nabídnout studentům vzdělání zajímavější formou, při které jsou studenti aktivněji zapojeni do výuky, a tím je více motivovat ke studiu. Díky využití možností informačních technologií lze výuku pojmout mnohem názorněji, studentům může být probíraná látka vysvětlena a znázorněna pomocí video nebo audio nahrávek nebo případně pomocí animací, na kterých si student může krok po kroku vyzkoušet například nově probraný postup řešení úlohy. Výuková aplikace vytvořená v této diplomové práci bude využívána pro podporu vyučování projektového řízení na vysoké škole. Studenti se prostřednictvím této aplikace seznámí s metodou CPM prostřednictvím tutoriálů v aplikaci a budou si moci metodu CPM vyzkoušet na předpřipravených příkladech a nebo vytvářet příklady vlastní.

Kapitola 3

Praktická část

3.1 Analýza a návrh aplikace

Před samotnou implementací je potřeba provést sběr a analýzu požadavků na připravovanou aplikaci. Sběr požadavků provádíme z toho důvodu, abychom určili a zjistili, co má připravovaná aplikace umět, jaké má podporovat procesy a jak se má chovat. Analýza sesbíraných požadavků se provádí pro upřesnění těchto požadavků a pro řešení nesrovnalostí. Cílem těchto činností je tedy lepší pochopení zadání ze strany programátora a shoda zainteresovaných stran na tom, jaké problémy aplikace řeší a co bude dělat.

3.1.1 Podporované procesy

Jelikož se jedná o výukovou aplikaci cílící na vysvětlení metody CPM, její pochopení a práci s ní, identifikoval jsem následující procesy, které by tato aplikace měla podporovat:

- Práce se síťovým grafem: aplikace musí umožnit uživatel práci se síťovým grafem, jelikož představuje základ pro výpočet metody CPM. Uživatel bude moci grafy vytvářet a editovat, tedy přidávat a odebírat z grafů uzly a hrany a provádět ohodnocení uzlů a hran. Síťové grafy bude také možné ukládat a načítat.

- Výpočet metody CPM: aplikace graficky znázorní na zadaném síťovém grafu výpočet pomocí metody CPM. Uživateli bude zobrazen dopředný i zpětný průchod síťovým grafem a aplikace zobrazí, jakým způsobem probíhá výpočet jednotlivých hodnot.

- Validace uživatelských řešení: na uživatelem zadaném a ohodnoceném síťovém grafu provede aplikace validaci jednotlivých hodnot a upozorní uživatele na případné chyby.

- Tutoriály: aplikace bude obsahovat tutorial, na kterém bude znázorněna, vysvětlena a popsána metoda kritické cesty, síťový graf a potřebná teorie grafů.

3.1.2 Funkční požadavky

Funkční požadavky specifikují, co by měla výsledná aplikace dělat a umět. Mezi funkční požadavky můžeme zařadit například manipulaci a zpracování dat, technické detaily, výpočty apod.

3.1.2.1 Síťový graf

Práce se síťovým grafem je důležitou součástí vytvářené aplikace. Jak již bylo nastíněno v popisu podporovaných procesů, aplikace musí umět podporovat tvorbu a validaci síťových grafů, umožnit jejich ohodnocení a editaci. V rámci pohodlnější práce s aplikací a snížením strachu z vytváření chyb by aplikace měla při práci se síťovým grafem podporovat tzv. undo a redo.

Síťový graf - seznam funkčních požadavků

1. Tvorba síťového grafu, předpokládá se forma activity-on-arrow.
   a) Přidávat uzly
   b) Odebírat uzly
   c) Přidávat hrany mezi uzly
   d) Odebírat hrany mezi uzly
   e) Ohodnocovat uzly: nejzazší doba ukončení provádění aktivit a nejdříve možný počátek provádění aktivit, číselné označení uzlu
   f) Undo (krok zpět)
2. Ohodnocovat hrany: doba trvání aktivity, krátký popis aktivity
3. Odlišné zvýraznění pro hrany fiktivních a čekacích aktivit
4. Ukládat vytvořené síťové grafy
5. Načítat vytvořené síťové grafy
6. Export celého grafu ve formě obrázku
7. Aplikace si bude pamatovat stav síťového grafu i když uživatel přejde do jiné části aplikace (například do tutoriálu)

3.1.2.2 Výpočet metody CPM

Aplikace je zaměřena na výuku metody CPM a její výpočet. Musí tedy být schopna tento výpočet provést nad zadaným síťovým grafem. Pro lepší pochopení této metody je také vhodné, aby aplikace uměla zvalidovat uživatelem provedený výpočet a upozornila ho na chyby.

1. Výpočet metody CPM nad zadaným síťovým grafem
    a) Dopředný průchod a výpočet nejdřívěji možných počátku provádění činností
    b) Zpětný průchod a výpočet nejzazších možných počátků provádění činností
    c) Výpočet celkové časové rezervy
    d) Výpočet volné časové rezervy
    e) Zvýraznění kritické cesty
2. Validace hodnot v síťovém grafu
    a) Zvaliduje uživatelem zadané hodnoty a vyznačení kritické cesty
    b) Zvýrazní prvky grafu (uzly a hrany) u nichž jsou špatně vyplněné údaje
    c) Nebude označovat chybící nebo chybně označené prvky na kritické cestě (došlo by tím k prozrazení řešení a uživateli by nebylo umožněno přijít na něj sám)
3. Grafické znázornění výpočtu
    a) Aplikace zobrazí dopředný a zpětný průchodu metody CPM krok po kroku
    b) V každém kroku zvýrazní, které prvky grafu se podílejí na výpočtu údajů a tento výpočet bude ukázán

3.1.2.3 Tutoriály

Tutoriály budou ve výukové aplikaci sloužit pro seznámení uživatele s danou problematikou potřebnou pro pochopení metody CPM a dále pro vysvětlení výpočtu doby trvání projektu pomocí metody kritické cesty.

1. Zobrazení tutoriálů
2. Procházení tutoriálů
3. Aplikace si bude pamatovat stav procházení tutoriálu dokud uživatel nepřepne na jiný tutoriál

3.1.3 Nefunkční požadavky

Na rozdíl od funkčních požadavků nefunkční požadavky neříkají, co by systém měl umět, ale co by měl splňovat a jak by se měl chovat. Jde tedy o popis a zhodnocení kritérií a vlastností aplikace, které musí splňovat pro pohodlné používání. Mezi nefunkční požadavky můžeme například zařadit dobu odezvy na uživatelské vstupy, dostupnost, podporované platformy a další.

3.1.3.1 Aplikace běžící ve webovém prohlížeči

Jde o základní požadavek na výukovou aplikaci. Tato aplikace by měla běžet jako offline webová aplikace. To znamená, že poběží lokálně ve webovém prohlížeči. Tím se zajistí jednoduchost instalace bez potřeby webového serveru a snadná přenositelnost na zařízení studentů, kterým k běhu bude stačit pouze moderní webový prohlížeč. Vzhledem k povaze aplikace se nepředpokládá běh na mobilních zařízeních s malou obrazovkou, jakými jsou například chytré telefony.

1. Aplikace realizovaná pomocí webových technologií
2. Bez nutnosti instalace
3. Offline běh na koncových zařízení
4. Podpora pouze pro zařízení s velkou obrazovkou (počítače, notebooky)
5. Síťové grafy budou hranově definované (activy-on-arrow)
6. Aplikace bude ukládat síťové grafy ve formátu JSON
7. Aplikace provede export celého síťového grafu ve formátu PNG
8. Tutoriály budou uloženy ve formátu JSON, aplikace je automaticky načte při spuštění
9. Obsah tutoriálu bude formátovaný hypertextem (html)

Rychlost reakce a odezva Rychlost reakcí aplikace hraje důležitou roli pro komfort jejího používání, i když je v tomto případě závislá od použitého webového prohlížeče.

1. Spuštění aplikace: do 5s.
2. Výpočet kritické cesty v síťovém grafu: do 5s.
3. Přidání uzlu nebo hrany do grafu: méně než 1s.
4. Přechod mezi jednotlivými kroky tutorialu: méně než 1s.

3.1.4 Případy užití

Případy užití (use case) popisují funkcionalitu vyvíjené aplikace z pohledu uživatele. Jde tedy o pohled na chování systému z pozice uživatele. V rámci případů užití rozdělujeme uživatele na jednotlivé role (tzv. actor), tyto role pak mohou iniciovat jednotlivé případy užití a také s používají k omezení toho, které případy užití mohou iniciovat. Vztah mezi jednotlivými rolemi a případy užití znázorňuje diagram případů užití (use case diagram). Tento diagram však nepopisuje, jakým způsobem se jednotlivé případy užití budou provádět. K tomu slouží až specifikace případů užití. Každý případ užití má svou specifikaci, ve které je vysvětleno, k čemu daný případ užití slouží, jsou popsány jednotliví účastnicí, kteří se ho účastní a podmínky pro spuštění. Samotné vykonání případu užití je v rámci jeho specifikace popsáno hlavním a alternativními scénáři.

3.1.4.1 UC1: Vytvoření nového síťového grafu

Popis scénáře Případ užití popisuje postup pro vytvoření nového síťového grafu.

Aktoři

- Uživatel
- Aplikace

Hlavní scénář

1. Uživatel vybere v menu položku *Síťový graf*.
2. Uživatel vybere v podmenu položku *Nový*.

Podmínky úspěchu Uživateli se zobrazí obrazovka pro práci se síťovým grafem.

Výjimky Uživateli se zobrazí chybová hláška: ”Nelze vytvořit síťový graf”.

3.1.4.2 UC2: Přidání uzlu do síťového grafu

Popis scénáře Případ užití popisuje postup pro přidání nového uzlu do síťového grafu.

Aktoři

- Uživatel
- Aplikace

Předpoklady Uživatel je na obrazovce pro vytváření síťového grafu.

Hlavní scénář

1. Uživatel klikne na tlačítko *Nový uzel*.
2. Na pracovní ploše se zobrazí nový uzel.
3. Číslo uzlu bude automaticky určeno aplikací.
4. Kliknutím a držením levého tlačítka myši uživatel přemístí uzel na pracovní ploše.

Alternativní scénář

1. Uživatel klikne pravým tlačítkem myši na prázdné místo pracovní plochy.
2. Z kontextového menu zvolí položku *Nový uzel*.
3. Nový uzel bude vytvořen v místě kliknutí.
4. Dále pokračuje bodem 3. hlavního scénáře.

Podmínky úspěchu Uzel byl přidán na obrazovku.

Výjimky Pokud se uzel nepodaří přidat, zobrazí se chybová hláška.

3.1.4.3 UC3: Editace uzlu síťového grafu

Popis scénáře Případ užití popisuje postup pro editaci uzlu - změnu ohodnocení.

Aktoři

- Uživatel
- Aplikace

Předpoklady Uživatel je na obrazovce pro vytváření síťového grafu. Na této obrazovce existuje uzel pro editaci.

Hlavní scénář

1. Uživatel klikne pravým tlačítkem na vybraný uzel.
2. Ve zobrazeném kontextovém menu vybere položku *Editovat uzel*.
3. Zobrazí se formulář pro editaci hodnot uzlu zobrazující stávající hodnoty:
   - Nejdříve možný počátek provádění aktivit (hodnota t_0^i).
   - Nejzazší možný konec provádění aktivit (hodnota t_1^i).
4. Uživatel přepíše stávající hodnoty novými.
5. Kliknutím na tlačítko potvrzení se nové hodnoty uloží do uzlu.

Alternativní scénář

1. Může nastat během editace hodnot ve formuláři pro editaci uzlu.
2. Kliknutím na tlačítko *Zrušit* se editace zruší a zůstávají původní hodnoty.

Podmínky úspěchu Hodnoty v uzlu byly změněny.

Výjimky Pokud se uzel nepodaří editovat, je zobrazena chybová hláška.

3.1.4.4 UC4: Odebrání uzlu ze síťového grafu

Popis scénáře Případ užití popisuje postup pro odstranění uzlu ze síťového grafu.

Aktoři

- Uživatel
- Aplikace

Předpoklady Uživatel je na obrazovce pro vytváření síťového grafu.

Hlavní scénář

1. Uživatel klikne pravým tlačítkem na uzel, jehož hodnoty chce editovat.
2. Z menu zvolí položku *Odstranit uzel*.
3. Uzel je odstraněn z grafu.
4. Aplikací jsou odstraněny také hrany vedoucí do uzlu a vedoucí z uzlu.

Alternativní scénář

1. Uživatel vybere levým tlačítkem uzel.
2. Uživatel smaže uzel stiskem klávesy *delete*.
3. Scénář pokračuje bodem 2) hlavního scénáře.

Alternativní scénář - odstranění více uzlů najednou

1. Uživatel označí uzly k odstranění stiskem a držením klávesy *L-CTRL* a kliknutím levého tlačítka myši na vybrané uzly.
2. Stiskem tlačítka *delete* jsou uzly odstraněny z grafu.

Podmínky úspěchu Uzel byl odebrán ze síťového grafu.

Výjimky Pokud se uzel nepodaří odebrat, zobrazí se chybová hláška.

3.1.4.5 UC5: Přidání hrany do síťového grafu

Popis scénáře Případ užití popisuje postup pro přidání hrany do síťového grafu.

Aktoři

- Uživatel
- Aplikace

Předpoklady Uživatel je na obrazovce pro vytváření síťového grafu. V grafu existují minimálně dva uzly.

Hlavní scénář

1. Uživatel levým tlačítkem myši vybere uzel.
2. Zobrazí se tlačítko pro přidání hrany.
3. Uživatel klikne na toto tlačítko a drží levé tlačítko myši.
4. Uživatel táhne hranu ke zvolenému cílovému uzlu.
5. Cílový uzel je aplikací zvýrazněn.
6. Uživatel pustí levé tlačítko myši.
7. Hrana je přidána mezi vybraný a vyznačený uzel.

Alternativní scénář

1. Tlačítko pro přidání uzlu se zobrazí při najetí kurzorem myši nad uzel.
2. Uživatel klikne na toto tlačítko a drží levé tlačítko myši.
3. Scénář pokračuje bodem 4) hlavního scénáře.

Alternativní scénář - zrušení akce

- Pokud není hrana dotažena ke koncovému uzlu, je přidání zrušeno.
- Pokud mezi zvolenými uzly již vede hrana, je přidání zrušeno aplikací.

Podmínky úspěchu Hrana je přidána mezi zvolené uzly.

Výjimky Pokud se hranu nepodaří přidat, zobrazí se chybová hláška.

3.1.4.6 UC6: Editace hrany síťového grafu

Popis scénáře Případ užití popisuje postup pro editaci hrany síťového grafu - změnu ohodnocení hrany.

Aktoři

- Uživatel
- Aplikace

Předpoklady Uživatel je na obrazovce pro vytváření síťového grafu. V grafu existuje minimálně jedna hrana.

Hlavní scénář

1. Uživatel klikne pravým tlačítkem na hranu, jejíž hodnoty chce editovat.
2. Ve zobrazeném kontextovém menu vybere položku *Editovat hranu*.
3. Zobrazí se formulář s položkami pro editaci se stávajícími hodnotami.
    - typ aktivity (drop down)
    - popis aktivity
    - doba trvání aktivy
    - volná časová rezerva
    - celková časová rezerva

4. Uživatel vyplní nové hodnoty.
5. Uživatel klikne na tlačítko pro potvrzení editace.
6. Hodnoty v hraně jsou změněny.

Alternativní scénář

1. Může nastat kdykoliv během hlavního scénáře.
2. Kliknutím na tlačítko *Zrušit* se editace zruší a zůstávají původní hodnoty.

Alternativní scénář - editace fiktivní aktivity

1. Nastane v bodě 3. hlavního scénáře při editaci fiktivní aktivity.
2. Zobrazí se formulář s položkami pro editaci se stávajícími hodnotami:
   - typ aktivity (drop down)
   - popis aktivity
   - doba trvání aktivity
3. Scénář pokračuje bodem 4) hlavního scénáře.

Podmínky úspěchu Hodnoty v hraně jsou změněny. Pokud je změněn i typ hrany, je vykreslena dle nového typu.

Výjimky Pokud se hranu nepodaří editovat, vypíše se chybová hláška.

3.1.4.7 UC7: Odebrání hrany síťového grafu

Popis scénáře Případ užití popisuje postup pro odstranění hrany ze síťového grafu.

Aktoři

- Uživatel
- Aplikace

Předpoklady Uživatel je na obrazovce pro vytváření síťového grafu. V grafu existuje minimálně jedna hrana.

Hlavní scénář

1. Uživatel klikne pravým tlačítkem na zvolenou hranu.
2. Uživatel v kontextovém menu zvolí položku *Odstranit*.
3. Hrana je odstraněna z grafu.

Alternativní scénář

1. Uživatel označí hranu levým tlačítkem myši.
2. Stiskem tlačítka *delete* je hrana odstraněna z grafu.

Alternativní scénář - odstranění více hran najednou

1. Uživatel označí hrany k odstranění stiskem a držením klávesy *L-CTRL* a kliknutím levého tlačítka myši na dané hrany.
2. Stiskem tlačítka *delete* jsou hrany odstraněny z grafu.

Podmínky úspěchu Hrana je odstraněna z grafu.

3.1.4.8 UC8: Vyznačení kritického prvku grafu

Popis scénáře Případ užití popisuje postup pro označení prvku grafu jako kritického (uzel nebo hrana).

Aktoři

- Uživatel
- Aplikace

Předpoklady Uživatel je na obrazovce pro vytváření síťového grafu. V grafu existuje minimálně jedna hrana.

Hlavní scénář

1. Uživatel klikne pravým tlačítkem na prvek grafu, který chce označit jako kritický (uzel nebo hrana).
2. Ve zobrazeném kontextovém menu vybere položku *Označit kritický*.
3. Prvek síťového grafu je zvýrazněn fialovou barvou.

Alternativní scénář - označení více prvků najednou

1. Uživatel označí jednotlivé prvky držením klávesy *L-CTRL* a tažením levým tlačítkem myši nebo klikáním levým tlačítkem myši na zvolené prvky.
2. Uživatel klikne pravým tlačítkem myši na jeden z vybraných prvků grafu a v kontextovém menu zvolí položku *Označit kritický* .
3. Prvky síťového grafu je zvýrazněn fialovou barvou.

Podmínky úspěchu Zvolené prvky jsou zvýrazněny fialovou bravou.

3.1.4.9 UC9: Zrušení vyznačení kritického prvku grafu

Popis scénáře Případ užití popisuje postup pro odebrání označení prvku grafu jako kritického (uzel nebo hrana).

Aktoři

- Uživatel
- Aplikace

Předpoklady Uživatel je na obrazovce pro vytváření síťového grafu. V grafu existuje minimálně jedna hrana označena jako kritická a uzly mezi kterými vede, které jsou označeny jako kritické.

Hlavní scénář

1. Uživatel klikne pravým tlačítkem na kritický prvek grafu, u kterého chce zrušit kritické označení.
2. Ve zobrazeném kontextovém menu vybere položku *Odebrat kritický*.
3. Zvýraznění kritického prvku je zrušeno.

Alternativní scénář - označení více prvků najednou

1. Uživatel označí jednotlivé prvky držením klávesy *L-CTRL* a tažením levým tlačítkem myši nebo klikáním levým tlačítkem myši na zvolené prvky.
2. Uživatel klikne pravým tlačítkem myši na jeden z vybraných prvků grafu a v kontextovém menu zvolí položku *Odebrat kritický*.
3. Zvýraznění kritických prvků je zrušeno.

Podmínky úspěchu Zvolené prvky nejsou dále označeny jako kritické.

3.1.4.10 UC10: Uložení síťového grafu

Popis scénáře Případ užití popisuje postup pro uložení síťového grafu.

Aktoři

- Uživatel
- Aplikace

Předpoklady Uživatel je na obrazovce pro editaci síťového grafu.

Hlavní scénář

1. Uživatel stiskne tlačítko *Uložit*.
2. Zobrazí se okno pro uložení souboru s grafem .
3. Uživatel zvolí jméno souboru pro uložení.
4. Kliknutím na tlačíko pro potvrzení je síťový graf uložen.

Alternativní scénář

1. Může nastat kdykoliv během hlavního scénáře.
2. Kliknutím na tlačítko *Zrušit* se uložení souboru zruší.

Podmínky úspěchu Síťový graf je uložen do souboru.

Výjimky Pokud se uložení nepodaří, je zobrazena chybová hláška popisující důvod neúspěchu.

3.1.4.11 UC11: Načtení síťového grafu

Popis scénáře Případ užití popisuje postup pro načtení síťového grafu

Aktoři

- Uživatel
- Aplikace

Předpoklady Uživatel má k dispozici soubor s uloženým grafem.

Hlavní scénář

1. Uživatel najede kurzorem myši na položku menu *Síťový graf*.
2. V zobrazené podnabídce klikne na položku *Načíst graf*.
3. Uživatel vybere soubor s grafem.
4. Uživatel zvolí umístění souboru pro uložení.
5. Aplikace odstraní stávající prvky grafu a načte ze souboru nové.

Alternativní scénář

1. Může nastat kdykoliv během hlavního scénáře.
2. Kliknutím na tlačítko *Zrušit* se nahrátí ze souboru zruší.

Podmínky úspěchu Síťový graf je načten ze souboru.

Výjimky Poku

3.1.4.12 UC12: Spuštění tutoriálu

Popis scénáře Případ užití popisuje postup pro spuštění tutorialu.

Aktoři

- Uživatel
- Aplikace

Předpoklady Uživatel je má spuštěnou aplikaci.

Hlavní scénář

1. Uživatel najede kurzorem myši na položku v menu *Tutoriály*.
2. Uživateli se zobrazí podnabídka s názvy tutoriálů.
3. Uživatel klikne na název znoleného tutoriálu.
4. Uživateli je zobrazena úvodní stránka zvoleného tutoriálu.

Podmínky úspěchu Uživateli je zobrazena úvodní stránka tutoriálu včetně ovládacích prvků tutoriálu.

3.1.4.13 UC13: Procházení tutoriálu

Popis scénáře Případ užití popisuje postup pro procházení tutoriálu.

Aktoři

- Uživatel
- Aplikace

Předpoklady Uživatel má spuštěný tutoriál. Tutoriál zablokuje tlačítka dopředu/dozadu pokud neexistuje další stránka v daném směru procházení.

Hlavní scénář

1. Uživateli jsou v rámci tutoriálu zobrazena tlačítka pro procházení tutoriálu *Tutoriály*.
2. Uživatel klikne na tlačítko dopředu.
3. Tutoriál načte další stránku.
4. Uživatel klikne na tlačítko dozadu.
5. Tutoriál načte předchozí stránku.

Alternativní scénář Nastane v případě, že uživatel bude ovládat tutoriál pomocí kurzorových šipek.

1. Uživatel stiskne kurzorovou šipku vpravo.
2. Tutoriál načte další stránku.
3. Uživatel stiskne kurzorovou šipku vlevo.
4. Tutoriál načte předchozí stránku.

Podmínky úspěchu Uživateli je načtena další stránka tutoriálu ve směru procházení.

3.1.4.14 UC14: Ukončení tutoriálu

Popis scénáře Případ užití popisuje postup pro opuštění tutoriálu.

Aktoři

- Uživatel
- Aplikace

Předpoklady Uživatel má spuštěný tutoriál.

Hlavní scénář

1. Uživatel klikne na tlačítko jiné sekce aplikace.
2. Uživateli je zobrazena zvolená sekce.
3. Aplikace si zapamatuje stav procházení tutoriálu.

Podmínky úspěchu Uživateli je načtena zvolená sekce.

3.1.5 Návrh uživatelského rozhraní

Jako uživatelské rozhraní jsou označovány prostředky k vzájemné komunikaci uživatele s aplikací. Uživatel pomocí uživatelského rozhraní ovládá aplikaci, aplikace na tyto vstupy reaguje a provádí požadované operace a zobrazuje uživateli výstupy. Práce s uživatelským rozhraním by měla být pro uživatele příjemná a efektivní. Návrh rozhraní by tedy měl vést k tomu, aby uživatel mohl s rozhraním snadno manipulovat, aby rozhraní bylo přehledné a poskytovalo dostatečnou zpětnou vazbu.

V rámci vývoje výukové aplikace jsem vytvořil wireframe (lo-fi prototyp) uživatelského rozhraní aplikace. Výhodou prototypování je, že prototyp uživatelského rozhraní lze vytvořit rychleji a s nižšími náklady, než kdyby se uživatelské rozhraní vytvářelo přímo s použitím cílových technologií. Prototyp je také možno jednodušeji upravit dle měnících se požadavků. Takovéto protypy uživatelského rozhraní lze vytvářet různými způsoby, často stačí i obyčejná tužka a čtverečkovaný papír. Já jsem pro potřeby této práce zvolil prototyp formou wireframu, protože je možné jednotlivé obrazovky ve wireframu pospojovat odkazy, díky čemuž jsem získal možnost, vyzkoušet si částečně uživatelské rozhraní ještě před samotnou implementací.

3.1.5.1 Task list

Task list neboli seznam úloh je souhrn úloh a akcí, které může uživatel v aplikaci provádět a se kterými se může při používání aplikace setkat. Seznam úloh se používá při základním návrhu uživatelského rozhraní. Pro vytvoření tohoto seznamu je potřeba mít sepsané a ujasněné případy užití, které budou v aplikaci prováděny. Z případů užití následně vyderivujeme jednotlivé úlohy, ze kterých se skládájí a které musí uživatel nebo aplikace v rámci připadu užití vykonat. Z těchto úloh bychom měli následně vyderivovat i jednotlivé atomické úlohy. Takto vytvořený seznam úloh ještě podrobíme potřebné analýze, vhodné je například seřadit jednotlivé úlohy podle důležitosti nebo podle obrazovek, na kterých je možné vykonat. V rámci návrhu uživatelského rozhraní jsem vytvořil následující seznam úloh v aplikaci, jednotlivé úlohy jsou seskupené podle jednotlivých obrazovek.

- Úvodní obrazovka
  - Zobrazit uvítací zprávu
  - Spustit nový tutoriál
  - Načíst síťový graf
  - Vytvořit nový síťový graf
  - Přejít na poslední otevřený tutoriál od spuštění aplikace
  - Přejít na poslední síťový graf tutoriál od spuštění aplikace
- Prohlížeč tutoriálů
  - Načíst nový tutoriál
  - Přejít na další stránku tutoriálu
  - Přejít na předchozí stránku tutoriálu
  - Zobrazit stránku tutoriálu
  - Návrat na úvodní obrazovku
  - Přejít na obrazovku s editorem grafů
  - Přejít na poslední otevřený tutoriál od spuštění aplikace
  - Přejít na poslední síťový graf tutoriál od spuštění aplikace
- Editor síťových grafů
  - Vytvořit nový síťový graf
  - Načíst síťový graf
  - Přidat uzel do grafu
  - Přidat hranu mezi uzly
  - Odebrat uzel nebo hranu z grafu
  - Editovat ohodnocení uzlu
  - Editovat ohodnocení hrany
  - Posouvat prvky grafu po pracovní ploše
  - Posouvat celou pracovní plochu
  - Přibližovat a oddalovat síťový graf
  - Provádět výpočty nad síťovým grafem (jednotlivé fáze metody kritické cesty)
  - Zvalidovat síťový graf z hlediska teogire grafů
  - Zvalidovat uživatelem zadané hodnoty z hlediska metody CPM
  - Zobrazit výsledek validace

- Zobrazit výsledek operace
- Uložit síťový graf
- Uložit síťový graf jako obrázek
- Animovat průchod výpočtu metody CPM síťovým grafem
- Přejít na poslední otevřený toturiál od spuštění aplikace
- Přejít na poslední síťový graf toturiál od spuštění aplikace

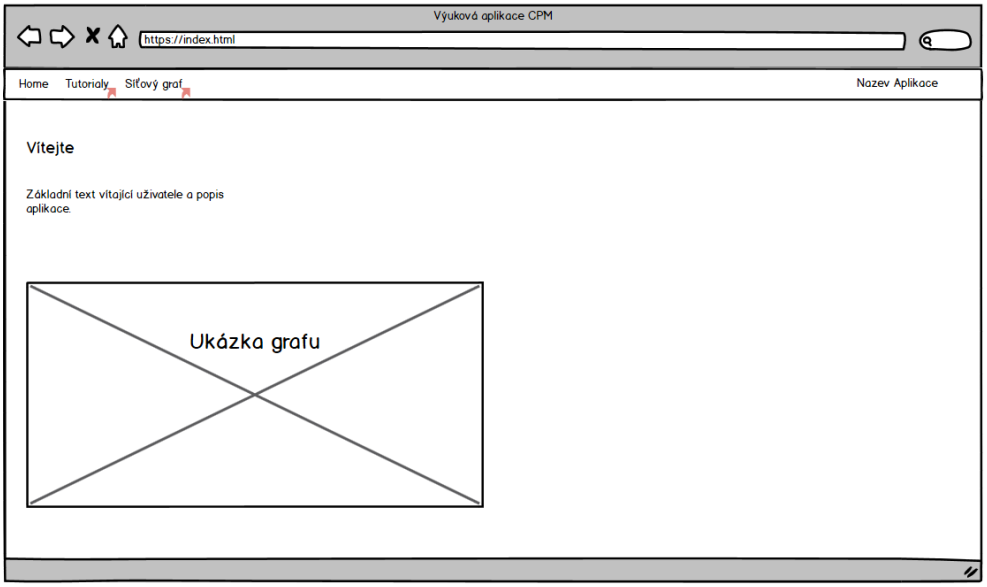
Analýzou seznamu úloh jsem si udělal představu o tom, jaké ovládací prvky budou potřeba na jednotlivých obrazovkách. Menu aplikace se bude skládat ze tří částí: tlačítka pro návrat na hlavní obrazovku, tlačítka pro návrat k tutoriálu a tlačítka pro návrat k síťovému grafu. Další ovládací prvky, například pro spuštění nového tutoriálu nebo pro načtení síťového grafu se zobrazí jako rozbalovací podmenu daných částí hlavního menu. Prohlížeč tutoriálů si vystačí s tlačítky pro přepínání mezi jednotlivými stránkami tutoriálu. Oproti tomu editor síťových grafů bude obsahovat větší množství ovládacích prvků a umístění všech těchto prvků formou tlačítka na obrazovce by pro uživatele mohlo být matoucí a nepřehledné. Z tohoto důvodu budou některé ovládací prvky editoru grafů dostupné pouze z kontextového menu, ačkoliv je pro uživatele výhodnější mít ovládací prvky vždy zobrazené, protože si nemusí pamatovat kde jsou, čímž se také urychluje doba nutná pro naučení ovládání apliakce.

3.1.5.2 Úvodní obrazovka

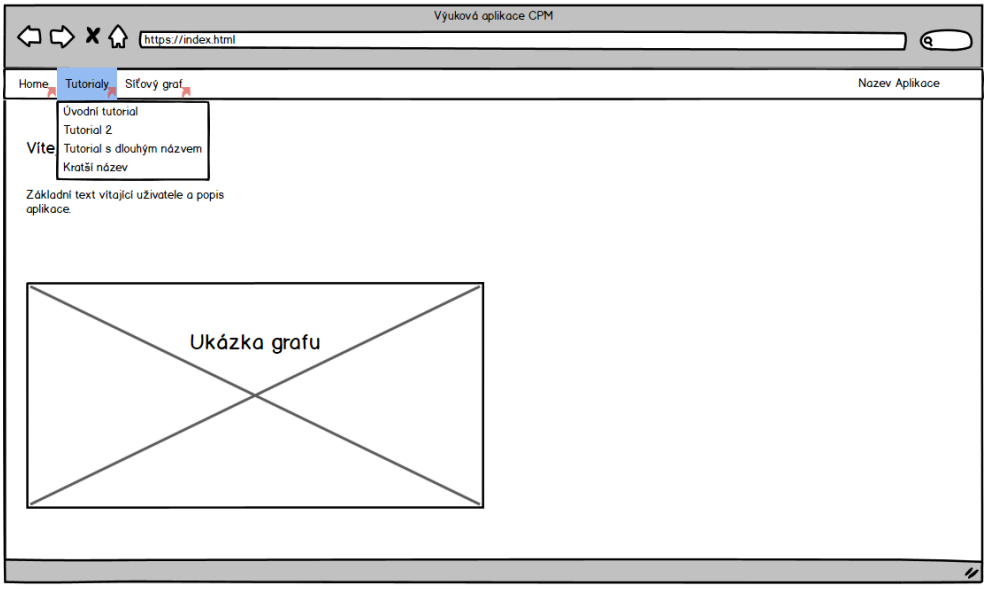
Na obrázku 3.1 je zobrazen návrh úvodní obrazovky aplikace, která se uživateli zobrazí po jejím spuštění. Na úvodní obrazovce bude zobrazena uvítací zpráva a krátký popis aplikace. Také zde bude umístěn ukázkový síťový graf, který lze v aplikaci vytvořit. Navigační menu je umístěno v horní části obrazovky, což je časté umístění u webových aplikací a uživateli by takováto navigace v aplikaci neměla dělat problém. Uživatel má skrze navigační menu přístup do ostatních částí aplikace. Pokud najede kurzorem myši na vybranou sekci, zobrazí se rozbalovací pod menu dané sekce. Toto chování je ilustrováno na obrázku 3.2. Pro rychlé přepínání mezi sekcemi budou využity přímo jejich názvy, uživatel tak bude moci rychle přejít například mezi rozpracovaným síťovým grafem a rozečteným tutoriálem.

3.1.5.3 Obrazovka Tutoriály

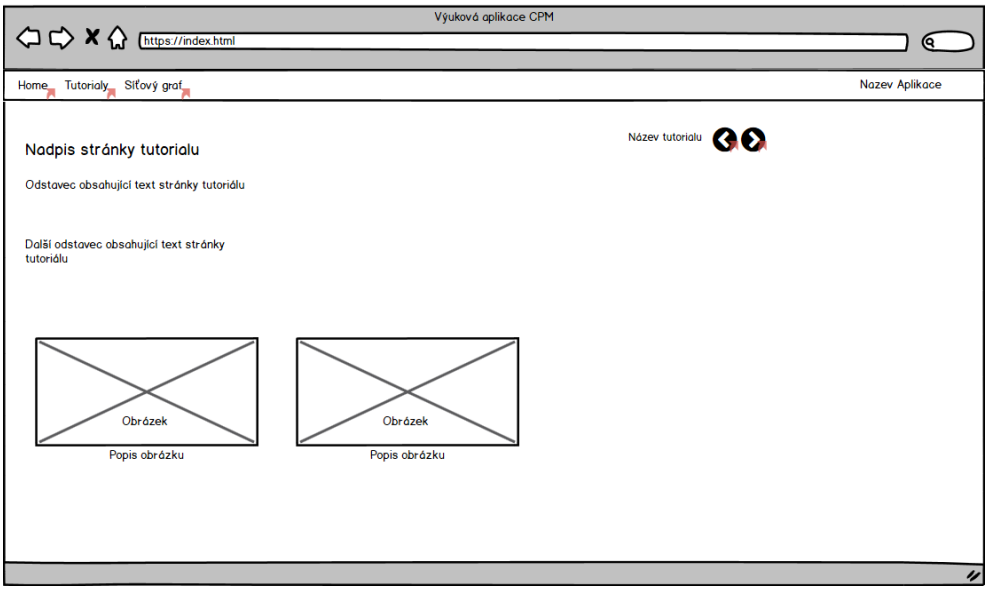
Obrazovka tutoriály 3.3 se od úvodní obrazovky moc neliší, aplikace si tak zachovává jednotný vzhled. Na obrazovce je umístěný ovládací prvek pro procházení mezi jednotlivými stránkami tutoriálu. Jde o dvě navigační šipky v pravém horním rohu. Takovéto umístění má výhodu v tom, že nijak neomezuje prostor pro vlastní obsah tutoriálu. Nevýhodou je, že tlačítka jsou



Obrázek 3.1: Úvodní obrazovka aplikace



Obrázek 3.2: Podnabídka sekce tutoriály

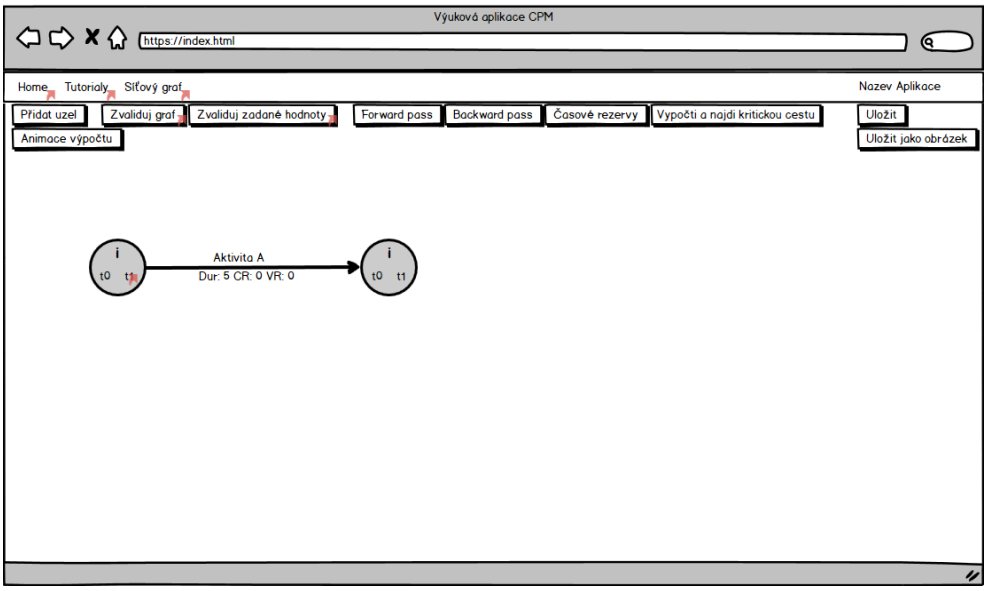


Obrázek 3.3: Obrazovka se stránkou tutoriálu

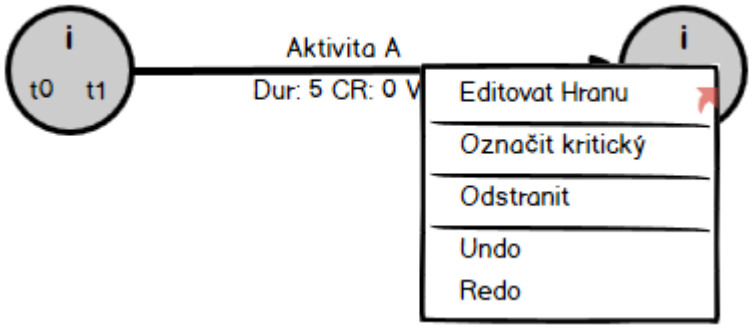
umístěna dále od ostatních ovládacích prvků na obrazovce, zejména tedy od menu aplikace. Tuto nevýhodu do určité míry zmírní možnost porcházet mezi stránkami tutoriálu pomocí kurzorových šipek na klávesnic. Zvolený způsob ovládání mi příjde přívětivý zejména pro uživatele notebooků, kde je touchpad velmi blízko kurzorovým šipkám. Vlastní obsah tutoriálu může být velice různorodý, počítá se i s možností zobrazit editor síťových grafů přímo na jednotlivých stránkách tutoriálu, v tom případě se ale uživatel nejspíše nevyhne vertikálnímu posunu.

3.1.5.4 Obrazovka Editoru síťových grafů

Jde o nejkomplikovanější obrazovku celé aplikace, zejména co se týče počtu ovládacích prvků. Tlačítka pro ovládání editoru grafů jsem se rozhodl umístit do horní části obrazovky. Síťové grafy se typicky kreslí zleva doprava a přišlo mi výhodné, ušetřit místo v horizontálním směru umístěním ovládacích prvků k hornímu okraji. Takto zbyde uživateli k práci velká a ničím nerušená pracovní plocha. Vzhledem ke specifické funkcionalitě větší části tlačítek se jeví jako uživatelsky více přívětivé zachovat jejich textový popis a nenahrazovat ho pouze ikonami. Příklad obrazovky s jednoduchým síťovým grafem je zobrazen na obrázku 3.4. Kromě tlačítek bude mít uživatel na této obrazovce možnost vyvolat kontextové menu. Toto kontextové menu bude dostupné při kliknutí pravým tlačítkem myši na volnou plochu grafu nebo na některý z prvků v grafu 3.5 a položky v tomto menu se podle toho budou měnit. Z kontextového menu

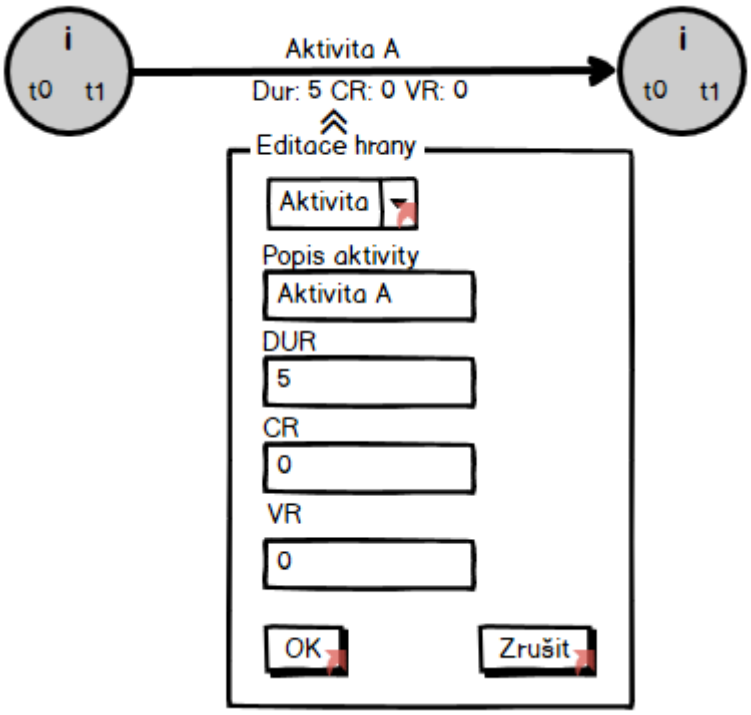


Obrázek 3.4: Obrazovka s editorem síťového grafu



Obrázek 3.5: Kontextové menu při kliknutí na uzel

bude možné vyvolat formulář pro editaci prvku v grafu a tento formulář bude zobrazen v blízkosti editovaného prvku. Předpokládám, že při práci s kontextovým menu se uživatel bude soustředit právě na tuto oblast obrazovky, a proto mi příjde logické do této oblasti umístit formuláře pro editaci. Samotný formulář 3.6 pak bude mít popisky (labely) umístěné nad vstupními poli, protože takotévo umístění zrychluje práci s formulářem [10].



Obrázek 3.6: Formulář pro editaci hodnot v uzlu

3.2 Architektura

Pro implementaci aplikace jsem zvolil návrhový vzor model-view-controller (MVC). Aplikace vytvořená tímto návrhovým vzorem je rozdělená na tři komponenty, které spolu komunikují:

- View - tato část se stará o manipulaci s uživatelským rozhraním a také zajišťuje zobrazení modelu.
- Controller - reaguje na podněty ostatních komponent a zajišťuje jejich provázání. Controller také manipuluje s ostatními komponentami, může například podle stavu aplikace nařídit view, které ovládací prvky má zobrazovat. Taktéž provádí na základě uživatelských vstupů manipulaci s modelem.
- Model - tato část udržuje stav aplikace (data a informace) a také obsahuje aplikační logiku. Podle požadavků controlleru provádí změny ve stavu aplikace a manipulaci s daty, dle aplikační logiky.

Jednotlivé komponenty v této architektuře jsou vzájemně propojeny rozhraním. Důsledkem toho je menší provázanost těchto komponent, při vhodné imple-

mentaci je například možné vyměnit view, aniž by se muselo zasahovat do controlleru nebo modelu. Architektura také umožňuje oddělit odpovědnosti jednotlivých částí. Nevýhodou této architektury je obtížnější trasování požadavků, které musejí procházet jednotlivými komponentami. Myslím si, že jde o vhodnou volbu architektury pro implementaci výukové aplikace, jelikož bych chtěl aplikaci vytvořit formou jediné webové stránky, kde se bude view měnit podle toho, jakou část aplikace si uživatel otevře. Například u prohlížeče tutoriálu se tato architektura přímo nabízí, jelikož ovládací prvky zůstanou na každé stránce tutoriálu stejné a bude se měnit pouze obsah. Zvolená architektura také umožní v případě potřeby integrovat editor grafů přímo do jednotlivých stránek tutoriálu a uživatel si tak bude moci rovnou vyzkoušet tutoriálem popisovanou problematiku.

Kapitola 4

Implementace

V této kapitole se budu zabývat popisem implementace výukové aplikace. Popíši jaké technologie jsem zvolil a použil při implementaci a následně přejdu k popisu samotné implementace. Součástí kapitoly bude popis vybraných algoritmů a částí kódu aplikace.

4.1 Zvolené technologie

HTML, javascript a css Z nefunkčních požadavků popsaných v předchozí kapitole této závěrečné práce vyplynulo, že výuková aplikace bude implementována formou webové aplikace. Důležitost je kladena zejména na jednoduchost instalace a používání. Základ aplikace tvoří HTML, javascript a css. To jsou obvyklé technologie používáné při vývoji webových aplikací. Jelikož má aplikace běžet pouze na zařízeních koncových uživatelů a jedním z požadavků byl běh bez použití web serveru, nebyla ani použita žádná serverová technologie. HTML je zkratka pro hypertext markup language, jde o značkovací jazyk, který se používá pro strukturování webových dokomentů. Výsledný dokument je repezentován tzv. document object modelem (DOM). Každý element webové stránky, kterými jsou například tlačítka, odstavce, odkazy, obrázky, atd., je součástí DOMu. Javascript může pomocí DOM přistupovat k jednotlivým elementům webové stránky, či aplikace, a tím vlastně získáváme API, se kterým můžeme při implementaci pracovat. Pro implementaci jsem použil javascript ve verzi ES6. CSS je zkratka pro cascading style sheets, pomocí css se jednotlivým elementům v html přidávají styly, které určují jejich výsledný vzhled a tím i vzhled celé webové stránky. CSS v podstatě popisuje, jakým způsobem má webový prohlížeč jednotlivé elementy zobrazit.

W3.CSS Pro grafický návrh aplikace a definici vzhledu jednotlivých HTML elementů jsem použil framework W3.CSS. Původně jsem také zvažoval užití knihovny Bootstrap 4, ale nakonec jsem došel k názoru, že pro mé využití je

W3.CSS naprosto dostačující. W3.CSS framework byl vytvořen s cílem poskytnout menší, rychlejší a k použití jednodušší framework [11]. Narozdíl od Bootstrapu nepotřebuje pro svůj běh javascript. K frameworku je dostupný přehledný tutoriál se spoustou příkladů, které usnadňují jeho použití.

Cytoscape.js Jde o javascriptovou knihovnu zaměřenou na teorii grafů. Tuto knihovnu jsem použil pro vizualizaci síťových grafů. Samotná knihovna je rozdělena na dvě části: jádro (core) a kolekci elementů, která obsahuje uzly a hrany grafu. Knihovna poskytuje API jak pro manipulaci s jádrem, tak i pro manipulaci s kolekcemi. Cytoscape.js také podporuje kaskádové styly (css), čehož jsem využil zejména pro vytvoření vlastního druhu uzlu nebo pro zvýrazňování jednotlových prvků v grafu. Výhodou knihovny je také množství rozšíření, které jsou pro ní dostupné. Z těch jsem využil například rozšíření pro tvorbu kontextového menu nebo pro tažení hran mezi uzly. Samotná knihovna je implementována v javascriptu a nevyžaduje žádné další knihovny, nicméně některá rozšíření vyžadovala použití jquery, a proto je tato knihovna přidána do aplikace. Knihovna cytoscape.js také obsahuje některé algoritmy pro průchod grafem, které jsem ale v aplikaci nevyužil. [12]

Tippy.js a Popper.js Popper.js je malá javascriptová knihovna pro pozicování elementů, která se dá použít pro umísťování tooltipů pro jednotlivé elementy v DOM. Knihovna je také jednoduše integrovatelná do cytoscape.js. Zatímco Popper představuje engine pro pozicování tooltipů, Tippy.js jsem použil pro jejich vytváření a stylování [13]. Tyto tooltipy jsem pak použil hlavně v editoru grafů pro přidání popisků k jednotlivým tlačítkům, ale také třeba i pro potřeby animace metody CPM.

4.2 Implementace

Webová aplikace je tvořena souborem index.html, který obsahuje definici hlavního menu, základní strukturu aplikace a tlačítka pro ovládání aplikace. Jediný obsah v tomto souboru je uvítací zpráva, která se zobrazí uživatelům po spuštěním aplikace. Tento soubor je pak manipulován javascriptem v zásivlosti na tom, které údaje a ovládací prvky jsou uživateli potřeba zobrazit. Aplikace se také spouští otevřením tohoto souboru ve vhodném webovém prohlížeči. Součástí výukové aplikace nejsou žádné další html soubory a veškerá navigace a vykreslování obsahu se odehrává v rámci spouštěcího index.html souboru. Aplikace je dále tvořena kódem napsaným v javascriptu, který je umístěn ve složce controller, daty pro tutoriály a ikonami pro tlačítka ve složce data a použitými knihovnami, které jsou umístěny ve složce lib. Ovládání aplikace představuje pro web typickou event driven architekturu, kde uživatel použitím ovládacího prvku, například tlačítka, vyvolá event (událost), který je odchycen

aplikací a ta pak provede požadovanou akci. Samotné eventy nejsou součástí html souboru, ale javascriptu, konkrétně ve třídách typu controller.

4.2.1 Třídy View, Controller a Model

Hlavní část aplikace, která se stará o zobrazování a přepínání mezi jednotlivými částmi - prohlížečem tutoriálů, úvodní obrazovkou a editorem grafů, je implementována ve třídách Model, View a Controller umístěných v souboru main.js. Při spuštění aplikace jsou vytvořeny instance právě těchto tří tříd.

Třída model v sobě drží referenci na controller tutoriálů a na controller editoru grafů. Třída model v sobě také obsahuje metodu pro načtení tutoriálů, které jsou uloženy ve formátu JSON v souboru data/tutorial_data.js. Tento soubor v sobě obsahuje jednotlivé tutoriály, které jsou uloženy ve formátu JSON do proměnných. Tyto proměnné jsou pak na konci souboru namapovány na tlačítka, kterými se spouští jednotlivé tutoriály. Tímto způsobem lze do aplikace přidávat nové tutoriály nebo tutoriály editovat, aniž by se musel upravit stávající kód aplikace.

Zvolený formát JSON pro mapování tutoriálů 4.1 je jednoduchý na pochopení, pořadí tutoriálu je určeno pozicí tutoriálu od začátku JSONu, čísla mají pouze informativní charakter. Myslím si, že by i pro uživatele neznalého javascriptu nemělo představovat problém přidat do aplikace nový tutoriál.

```
    var tutorialButtonMapping = {
     "0" : {buttonText : "Úvod do aplikace",
        tutorial_data : tut_user_guide},
     "1" : {buttonText : "Teorie grafů", tutorial_data :
        tut_graph_theory},
     "2" : {buttonText : "Konstrukce síťového grafu",
        tutorial_data : tut_aoa},
     "3" : {buttonText : "Metoda kritické cesty",
        tutorial_data : tut_cpm},
     "4" : {buttonText : "Editor grafů", tutorial_data :
        tut_editor},
}
```

Kód 4.1: Mapování tutoriálů na tlačítka

4.2.2 Prohlížeč tutoriálů - třídy TutorialVIew, TutorialController a TutorialModel

Prohlížeč tutoriálů je tvořen třídami TutorialView, TutorialController, TutorialModel a TutorialPage. Nejjednoduší třídu představuje TutorialPage, která reprezentuje jednu stránku tutoriálu. TutorialPage v sobě obsahuje jak hyper-

text s tělem tutoriálu, tak případný síťový graf, který se má načíst na dané stránce tutoriálu.

Třída TutorialModel při načtení tutoriálu vytvoří z jeho JSON definice kolekci instancí třídy TutorialPage. Model dále obsahuje metody pro procházení tutoriálem a načtení jiného tutoriálu, aby se při změně tutoriálu nemusela vytvářet nová instance třídy TutorialModel, nakonec je zde ještě metoda pro uložení stavu síťového grafu. Tutoriál si ukládá změny provedené v jednotlivých síťových grafech na stránkách tutoriálu, aby mohl uživatel mezi nimi nerušeně procházet. Tyto změny jsou uloženy až do doby, než je uživatelem spuštěn jiný tutoriál.

```
var tutorial = {
    title: "Titulek tutoriálu",
    pages : [
        {
            pageId: 0,
            type: "text_only",
            text_content: `<h1>Nadpis stránky</h1>
            <div>
                <p>
                    Příklad tutoriálu
                </p>
            </div> `
        },
        {
            pageId: 1,
            type: "text_graph",
                        graph: {
                "nodes": [],
                "edges": []
            } ,
            text_content: `<h1>Gratuluji</h1>
            <div>
                <p>
                    Nyní byste měli zvládnout základní
                       navigaci v~této aplikaci.
                </p>
            </div> `
        },
    ]
}
```

Kód 4.2: Stránky tutoriálu ve formátu JSON uložené do proměnné tutorial

Třída TutorialController obsahuje metody, které reagují na události vyvolané v TutorialView. Existence třídy TutorialView je možná překvapivá, protože by její funkcionalitu klidně mohla zastat i třída View, která je popsána výše. Rozhodl jsem se ale funkcionalitu specifickou pro tutoriál vložit právě do třídy TutorialView.

Samotné stránky tutorialu jsou uloženy v souboru data/tutorial_data.js. Zde má každý tutorial vytvořenou proměnnou do které je uložena definice tutoriálu ve formátu JSON (viz. ukázka 4.2), tutoriál je pak přidán pomocí mappingu, jak již bylo popsáno v odstavci věnujícímu se třídě Model (viz. 4.1). Oddělení tutoriálů od samotné aplikační logiky má tu výhody, že by je jednoduchou úpravou odkazu bylo možné distribuovat například ze školního serveru. Samotný tutoriál se skládá z titulku (title) a stránek (pages). Obsah stránky je definován pod klíčem *text_content*. Stránka tutoriálu může také obsahovat editor síťových grafů s přednačteným grafem. Graf se vkládá pod klíč *graph* a to buď přímo jako JSON nebo jako proměnná obsahující tento JSON. JSON grafu je možné získat uložením síťového grafu vytvořeném v editoru grafů.

4.2.3 Editor grafů - třídy GraphView, GraphController a GraphModel

Editor síťových grafů je tvořen třemi třídami: GraphView, GraphController a GraphModel. Třída GraphView is udržuje reference na tlačítka jimiž se ovládá editor síťových grafů, další součástí této třídy jsou metody pro vypisování hlášek uživateli a metody pro zobrazování, schovávání a vypínání něterých tlačítek. GraphController má v sobě definované jednotlivé metody, které obsluhují události vyvolané v GraphView. V této tříde je zároveň definované kontextové menu. Komunikace mezi GraphModelem a GraphControllerem je oboustraná, model například notifikuje kontrolér o ukončení animace, ten pak notifikuje GraphView, které upraví svůj stav.

Aplikačí logika editoru grafů je schována do třídy GraphModel, v této třídě je také vytvořena instance cytoscape.js. To svým způsobem porušuje architekturu MVC, jelikož elementy grafu jsou vytvářeny a zobrazovány přímo v modelu grafu. Dalším takovým prohřeškem je editace uzlů a hran, kde je definice formulářů přímo součástí modelu. Metody pro vytvoření těchto formulářů jsem umístil do GraphModelu proto, že mi nepřišlo vhodné data pro naplnění formulářů výchozími hodnotami předávat do GraphView, ale hlavně také proto, že se tyto formuláře dle návrhu uživatelského rozhraní mají zobrazovat přímo u editovaných prvků grafu a informace o jejich pozici má v sobě uložen model v instanci cytoscape.js. Event handlery pro tyto formuláře jsou součástí GraphControlleru. Třída GraphModel také obsahuje metody pro provádění výpočtu jednotlivých fází metody CPM.

4.2.4 Validace síťového grafu

Validace síťového grafu je implementována v metodě validateGraph() třídy GraphModel. Tato metoda provádí validaci síťového grafu s ohledem na teorii grafů a vlastnosti síťového grafu. Kontroluje se existence pouze jednoho počátečního a koncového uzlu v síťovém grafu a neexistence cyklu. Pro kontrolu počtu počátečních a koncových uzlů jsem použil metody knihovny cytoscape.js, které přímo vrací uzly bez potomků a uzly bez rodičů.

Pro detekci cyklů jsem použil variantu algoritmu prohledávání do hloubky (depth first search - DFS) rozšířenou o obarvování uzlů. Bíle označíme uzly na začátku zpracování, tyto uzly ještě nebyly algoritmem navštíveny. Šedivou barvou označíme ty uzly, které algoritmus již navštívil, ale ještě neprošel všechny jejich potomky. Černě pak označíme ty uzly, kde algoritmus prošel všemi jejich potomky. Pokud algoritmus při procházení grafem narazí na uzel označený šedivou barvou, znamená to, že objevil cyklus. To je dáno tím, že algoritmus prochází síťovým grafem do hloubky, potomek by tedy v podstatě narazil na svého rodiče. Pseudokód je vyobrazen v ukázce kódu 4.3. Takto napsaný algoritmus pro detekci grafů projde v acyklickém grafu všechny uzly a hrany.

```
//Všechny uzly jsou ve výchozím stavu považovány za bílé
isAcyclic(node){
 if(node.color == grey){
        return false;
 }
 node.color = grey;
 forEach(child : node.childNodes){
   if(child.color != black){
         if(isAcyclic(child) == false){
           return false;
         }
   }
 }
 node.color = black;
 return true;
}
```

Kód 4.3: Algoritmus pro detekci cyklů v grafu

4.2.5 Výpočty v síťovém grafu

Pro výpočet dopředné a zpětné fáze (forward a backward pass) metody kritické cesty je potřeba projít všechny cesty v grafu v daném směru procházení. To by bylo samozřejmě neefektivní, a proto jsem do algoritmu pro výpočet zahrnul i prořezávání procházených cest. Algoritmus nepokračuje v procházení cesty,

pokud hodnota procházené cesty nemůže být použita pro zbytek výpočtu. Uvažujme například dopředný průchod síťovým grafem. Do uzlu může vést více hran a tedy i více cest, pro hodnotu nejdříve možného počátku aktivit tohoto uzlu se ale použije pouze maximum z doby trvání jednotlivých cest. Pokud tedy algoritmus zpracovává cestu, která má kratší dobu trvání než je momentální hodnota v uzlu, může výpočet zastavit viz. 4.4.

```
forwardPassDFS(parentNode, activity, node){
  newEarlyStart = parentNode.earlyStart + activity.
     duration;
  if(node.earlyStart == undefined){
        node.earlyStart = newEarlyStart;
  }
  else if( node.earlyStart < newEarlyStart){
        node.earlyStart = newEarlyStart
  } else {
        return;
  }

  forEach( child : node.childNodes ){
        activity = node.getActivityTo(child);
        forwardPassDFS(node, activity, child);
  }
}
```

Kód 4.4: Pseudokód výpočtu dopředné fáze metody CPM

Během výpočtu časových rezerv jsem využil možnosti přímého přístupu k hranám grafu. Jelikož všechny potřebné hodnoty jsou již vypočítány při dopředném a zpětném průchodu, stačí pro každou hranu navštívit její koncový a počáteční uzel. Knihovna cytoscape.js obsahuje potřebné metody, a proto jsem nemusel procházet síťový graf. Obdobným způsobem jsem postupoval i při vyhledávání kritické cesty v síťovém grafu. Stačilo pouze projít kolekci hran a z ní vybrat ty hrany, které měly časové rezervy rovny nule.

4.2.6 Animace průchodu výpočtu metodou CPM v síťovém grafu

Na rozdíl od předchozích výpočtů jsou na samotnou animaci průchodu metody kritické cesty kladeny vyšší nároky. Nestačí totiž pouze provést výpočet, ale také ukázat posloupnost kroků, která se přiblíží způsobu, jakým by výpočet mohl provést i uživatel aplikace. Z tohoto důvodu je potřeba provést nejdříve topologické seřazení jednotlivých uzlů grafu. Topologické uspořádání seřadí uzly grafu takovým způsobem, že rodiče jsou vždy seřazeni před svými potomky. Pokud projdeme takto seřazené uzly, přiblížíme se způsobu, jakým by

výpočet provedl uživatel. Ten určitě nebude procházet postupně jednotlivé cesty v grafu a přepisovat hodnoty, ale spíše pokud narazí na uzel, kde mu pro rozhodnutí o hodnotě chybí vypočtené nějaké cesty v grafu, provede nejdříve jejich výpočet a pak se k problémovému uzlu vrátí. Pro implementaci topologickéhe seřazení uzlů jsem zvolil průchod grafem formou prohledávání do šířky (breadth-first search - BFS).

Algoritmus 4.5 svůj průdchod začíná v počátečním uzlu síťového grafu, jemuž nastaví délku cesty na nulu a označí ho jako navštívený. Uzel je pak vložen do fronty (datový typ FIFO - první kdo je do fronty vložen, je z ní také první vybrán). Fronta je procházena v cyklu, dokud jsou v ní nějaké prvky. Ve smyčce cyklu je z fronty vybrán uzel. Pro každého potomka uzlu algoritmus ověří, jestli byl již navštíven. Pokud ano, algoritmus také zkontroluje jestli cesta, kterou se do potomka dostal, je delší než stávající nejdelší cesta. Tím dochází k prořezávání procházených cest. Pokud je cesta delší, je potomek vložen do fronty a délka jeho cesty aktualizována. V případě, že potomek nebyl ještě algoritmem navštíven, je také vložen do fronty a jeho cesta je nastavena. Výstupem algoritmu jsou jednotlivé uzly grafu a délka nejdelší cesty, která do nich vede. Délkou je v tomto případě myšlen počet hran, nikoliv souhrná doba trvání aktivit na této cestě. Nakonec je nutné provést seřazení uzlů podle délky cesty, pro animaci dopředného průchodu vzestupně a pro zpětný průchod sestupně.

```
BFS_sort(start_node) {
  queue.push(start_node);
  start_node.visited = true;
  start_node.path_length = 0;
  while (queue.length > 0) {
    v_node = queue.front();
        forEach( child : v_node.children ) {
          if (child.visited) {
                if (child.path_length < v_node.
                   path_length+1) {
                      child.path_length = v_node.
                         path_length+1;
                      queue.push(child);
                }
          } else {
               child.path_length = v_node.path_length+1;
               child.visited = true;
               queue.push(child);
          }
        }
  }     }
```

Kód 4.5: Pseudokód topologického seřazení uzlů

4.2.7 Obsah tutoriálů

Součástí aplikace je pět tutoriálů s názvy Úvod do aplikace, Teorie grafů, Konstrukce síťového grafu, Metoda kritické cesty a Editor grafů. První a poslední tutoriál se zabývá samotnou výukovou aplikací. V tutoriálu Úvod do aplikace se uživatel seznámí se základním rozložením aplikace, tutoriál Editor grafů se pak zabývá práci s editorem síťových grafů. Uživatel se v tomto tutoriálu seznámí s ovládáním editoru grafů a prováděním základních operací, mezi které patří například vytváření a odebírání uzlů, vytváření a odebírání hran, editace prvků v grafu, provádění výpočtů a validace grafu, načítání a ukládání grafu.

Zbývající tutoriály jsou zaměřeny na metodu kritické cesty a teorii nutnou k jejímu pochopení. Tutoriál Teorie grafů je zaměřen na základní pojmy z teorie grafů, které je nutné znát pro pochopení síťového grafu. Navazující tutoriál Konstrukce síťového grafu se pak hlouběji zabývá síťovým grafem a na vhodných příkladech ukazuje, jakým způsobem je možné sestrojit hranově definovaný síťový graf. Poslední tutoriál s názvem Metoda kritické cesty je zaměřen na vysvětlení této metody. Uživatel je nejdříve seznámem s významem metody CPM a jednotlivých charakteristik, které se pro aktivity při použití této metody vypočítávají. Následně jsou jednotlivé fáze metody kritické cesty demonstrovány na příkladu, včetně vysvětlení jednotlivých výpočtů.

Kapitola 5

Testování

5.1 Nielsenova heuristická analýza

Heuristická analýza představuje formu testování uživatelského rozhraní nebo samotné aplikace. Heuristická analýza porovnává aplikaci vůči předem daným pravidlům, výstupem analýzy je zpráva obsahující seznam zjištěných nedostatků, které porušují některá z pravidel heuristické analýzy. Tato analýza je prováděna experty, ideální počet představuje pět expertů, kteří by tím měli odhalit okolo tří čtvrtin všech problémů [14]. Jde tedy o formu testování bez účasti uživatelů. Jelikož jde o jednu ze základních metod pro otestování použitelnosti, rozhodl jsem se provést Nilsenovu heuristickou analýzu na výukové aplikaci k ohdalení možných problémů při použití aplikace. Nielsenova heuristická analýza [15] se skládá z následujících pravidel:

- Viditelnost stavu systému - systém by měl vždy informovat uživatele o tom, co dělá. Uživatel by neměl mít pocit, že systém zamrzá nebo nereaguje na jeho povely.
- Shoda mezi systémem a realitou - systém by měl s uživatelem komunikovat srozumitelným způsobem a zachovávat konvence reálného světa. V tomto ohledu může být dnes sporné použití ikony s disketou pro ukládání, i když je to zažitá konvence.
- Minimální zodpovědnost - uživatel by při používání aplikace neměl mít strach z toho, že se něco pokazí a měl by mít možnost vrátit provedené změny. Uživatel by také měl být upozorněn, pokud operace provede nevratné změny a mělo by od něj v tomot případě být vyžadováno potvrzení.
- Shoda s použitou platformou - aplikace nebo systém by měly dodržovat konvence dané použitou platformou. Například aplikace v operačním systému Windows mají typicky tlačítka pro uzavření a minimalizaci

okna umístěná v pravém rohu. Jejich přemístění by uživatele zmátlo a nabouralo jejich zažité postupy při používání takových aplikací.

- Prevence chyb - systém by svým návrhem měl předcházet vzniku chyb, například tím, že zamezí zadání neplatného vstupu.
- Rozpoznávání místo vzpomínání - důležité prvky by měli být viditelné a snadno dosažitelné. Uživatel nesmí být nucen a nechce si pamatovat velké množství informací. Informace potřebné pro použití systému by měl mít vždy v danou chvíli zobrazené.
- Flexibilita a efektivita - systém by měl být navržen s ohledem na potřeby běžných i pokročilých uživatelů. Typickým příkladem jsou klávesové zkratky, ty se naučí používat uživatel, který provádí dané akce často.
- Minimalita - vše co je momentálně zobrazeno soutěží o pozornost uživatele a může odvádět jeho pozornost od důležitých prvků.
- Smysluplné chybové hlášky - chybové hlášky by měly být v běžném jazyce, měly by uživateli popsat, proč k chybě došlo a doporučit mu, jak situaci vyřešit. Zejména u výuková aplikace by vhodné chybové hlášky měly mít i poučný charakter.
- Nápověda a dokumentace - Systém by měl být použitelný i bez nápovědy, nicméně i tak by vždy měl nějakou formu nápovědy obsahovat.

Pro jednotlivé body heurisitké analýzy jsem nalezl následující nedostatky:

Viditelnost stavu systému V aplikaci chyběla upozornění na dokončení některých akcí. Například pokud uživatel po aplikaci požadoval vypočtení a nalezení kritické cesty, aplikace ji vyznačila, ale poté neinformovala uživatele o dokončení výpočtu. Na podobný problém jsem narazil i u výpočtu časových rezerv.

Shoda mezi systémem a realitou V tomto bodě aplikace obstála. Jednotlivé názvy sekcí a tlačítek jsou srozumitelné a je jasné, co který ovládací prvek znamená a jaké akce spouští. Použité ikony usnadňují navigaci v aplikaci a odpovídají funkcionalitě, kterou reprezentují. Nahradit tlačítka pouze ikonami by nebylo vhodné. Zjednodušení uživatelského rozhraní by nevyvážilo nutnost uživatele pamatovat si, co která ikona znamená.

Minimální zodpovědnost Tento bod se týká pouze editoru grafů. Přidávání a odebírání prvků v grafu je ošetřeno pomocí undo a redo. Aplikace by měla uživatele upozornit na to, že při načtení nového síťového grafu bude stávající graf přepsán bez možnosti návratu. Kromě tohoto problému zbytek aplikace splňuje požadavky tohoto bodu.

Shoda s použitou platformou Chování výukové aplikace odpovídá tomu, co by uživatel od takové aplikace očekával. Webové stránky a webové aplikace se často v základním návrhu od sebe moc neliší, což usnadňuje práci uživatele. S aplikací se dá pracovat jako s běžnou webovou stránkou. Problematické se zdá pouze neošetření tlačítek *zpět* a *dopředu* webového prohlížeče, která nelze používat pro procházení uvnitř aplikace.

Prevence chyb V aplikaci je zajištěno, aby uživatel při editaci uzlů a hran nemohl zadat nesmyslné hodnoty, a je také validováno, aby mezi dvěma uzly nevzniklo více hran. Tyto kontroly by šly dále rozšířit automatickou detekcí cyklů při přidávání hran a kontrolou, aby hrany nevedly z uzlů s vyšším pořadovým číslem do uzlů s nižším, což by automaticky zabránilo i vzniku cyklů.

Rozpoznávání místo vzpomínání Aplikace by mohla zobrazovat číslo stránky tutoriálu, na které se uživatel v dané chvíli nachází a také celkový počet stránek v tutoriálu. V editoru síťových grafů jsou vyznačeny chybné prvky grafu, ale chybová hláška se zobrazí pouze jednou a pak zmizí. Možná by bylo vhodné chybovou hlášku nechat zobrazenou až do příští validace.

Flexibilita a efektivita Součástí aplikace jsou pouze základní klávesové zkratky pro undo a redo a neobsahuje žádný mód pro pokročilé uživatele. Nemyslím si, že by to byl vzhledem k plánovanému využití problém.

Minimalita Myslím, že tento bod heuristické analýzy je aplikací splněn.

Smysluplné chybové hlášky Uživatel je upozorněn na případné chyby v přirozeném jazyce i s vysvětlením, jak k chybě došlo. Načítání síťových grafů by mohlo zobrazovat hlášku, pokud soubor neodpovídá očekávánému formátu.

Nápověda a dokumentace Aplikace obsahuje tutoriál, který uživatele popíše základní navigaci v aplikaci a také tutoriál věnující se funkcionalitě editoru síťových grafů. Tlačítka v editoru síťových grafů jsou opatřena titulky, nicméně i tak v aplikací chybí nápověda.

5.2 Testování funkcionality

Pro otestování funkcionality vytvořené výukové aplikace jsem vytvořil testovací scénáře, podle kterých jsem otestoval základní funkcionalitu vyvinuté aplikace. Testovací scénáře jsou namapované na jednotlivé případy užití, aby došlo k ověření, že všechny případy užití byly implementovány.

5.2.0.1 TC1: Spuštění a načtení aplikace

Tento scénář se zabývá spuštěním výukové aplikace v podporovaném webovém prohlížeči. Uživatel spustí aplikaci a ověří, že byly načteny všechny prvky aplikace a zejména jednotlivé tutoriály.

Tabulka 5.1: Test case 1 - Spuštění a načtení aplikace

TC1: Spuštění a načtení aplikace		
Popis scénáře	Testovací případ slouží k otestování spuštění výukové aplikace	
Use case		
Předpoklady	Uživatel spouští aplikaci v podporovaném prohlížeči.	
ID	**Krok testu**	**Očekávaný výsledek**
1.	Uživatel spustí aplikaci otevřením souboru index.html.	Aplikace zobrazí úvodní obrazovku
2.	Uživatel přejde kurzorem myši na tlačítko *Tutoriály* v menu aplikace	Zobrazí se rolovací podmenu s tlačítky pro spuštění všech tutoriálů.
3.	Uživatel přejde kurzorem myši na tlačítko *Síťový graf* v menu aplikace	Zobrazí se rolovací podmenu s tlačítky. *Nový graf* a *Síťový graf*

5.2.0.2 TC2: Spuštění tutoriálu

V tomto scénáři je ověřeno, že aplikace je schopna korektně spustit tutoriál. po spuštění tutoriálu by se měla zobrazit úvodní obrazovka daného tutoriálu a ovládací prvky tutoriálu.

Tabulka 5.2: Test case 2 - Spuštění tutoriálu

TC2: Spuštění tutoriálu		
Popis scénáře	Testovací případ slouží k otestování spuštění nového tutoriálu ve výukové aplikaci	
Use case	UC12	
Předpoklady	Aplikace je spuštěna a tutoriály byly při spuštění načteny.	
ID	**Krok testu**	**Očekávaný výsledek**
1.	Uživatel přejde kurzorem myši na tlačítko *Tutoriály* v menu aplikace.	Zobrazí se rolovací podmenu s tlačítky pro spuštění všech tutoriálů.
Test case pokračuje na další stránce		

Tabulka 5.2 – Test case 2 pokračování

ID	Krok testu	Očekávaný výsledek
2.	Uživatel zvolí položku *Úvod do aplikace*	Zobrazí se první stránka tutoriálu s nadpisem *Úvod*, zobrazí se text úvodní stránky tutoriálu a tlačítka pro procházení tutoriálu. Vedle tlačítek pro procházení tutoriálu je zobrazen název tutoriálu, číslo stránky a celkový počet stránek.

5.2.0.3 TC3: Procházení stránek tutoriálu

V tomto testovacím scénáři je ověřeno správné chování prohlížeče tutoriálů. Po spuštění tutoriálu je ověřeno chování tlačítek pro procházení mezi jednotlivými stránkami tutoriálu a překreslování obsahu dle jednotlivých stránek.

Tabulka 5.3: Test case 3 - procházení tutoriálu.

TC3: Procházení tutoriálu		
Popis scénáře	Testovací případ slouží k otestování procházení spuštěného tutoriálu ve výukové aplikaci	
Use case	UC12, UC13	
Předpoklady	Aplikace je spuštěna a tutoriály byly při spuštění načteny.	
ID	**Krok testu**	**Očekávaný výsledek**
1.	Uživatel přejde kurzorem myši na tlačítko *Tutoriály* v menu aplikace.	Zobrazí se rolovací podmenu s tlačítky pro spuštění všech tutoriálů.
2.	Uživatel zvolí položku *Úvod do aplikace*.	Zobrazí se první stránka tutoriálu včetně nadpisu a textu. Název tutoriálu je zobrazen vedle tlačítek pro procházení tutoriálem, číslo stránky je nastaveno na jedna. Zobrazí se tlačítka pro procházení tutoriálem, tlačítko pro přepnutí na předchozí stránku je vypnuté.
3.	Uživatel stiskne tlačítko pro přechod na další stránku tutoriálu.	Aplikace načte následující stránku tutoriálu: nadpis stránky a obsah. Počítadlo stran se zvýší o jedna. Tlačítko pro přechod na předchozí stránku se stane aktivní.
Test case pokračuje na další stránce		

Tabulka 5.3 – Test case 3 pokračování

ID	Krok testu	Očekávaný výsledek
4.	Uživatel stiskne pravou kurzorovou šipku pro přechod na další stránku tutoriálu.	Aplikace načte následující stránku tutoriálu: nadpis stránky a obsah. Počítadlo stran se zvýší o jedna. Tlačítko pro přechod na předchozí stránku se stane aktivní.
5.	Uživatel přejde na poslední stránku tutoriálu.	Aplikace načte následující stránku tutoriálu: nadpis stránky a obsah. Číslo stránky na počítadlu stran souhlasí s celkovým počtem stran. Tlačítko pro přechod na následující stránku bude vypnuté.
6.	Uživatel stiskne tlačítko pro přechod na předešlou stránku tutoriálu.	Aplikace načte předcházející stránku tutoriálu: nadpis stránky a obsah. Počítadlo stran se sníží o jedna. Tlačítko pro přechod na další stránku tutoriálu se stane aktivní.
7.	Uživatel stiskne levou kurzorovou šipku pro přechod na předešlou stránku tutoriálu.	Aplikace načte předcházející stránku tutoriálu: nadpis stránky a obsah. Počítadlo stran se sníží o jedna.

5.2.0.4 TC4: Spuštění editoru grafů

Tento scénář se zabývá spuštěním editoru grafů ve výukové aplikaci. Po spuštění editoru grafů by se uživateli měly zobrazit a zpřístupnit ovládací prvky, mezi které patří tlačítka a kontextové menu. Samotnou funkcionalitu pak ověřují jiné testovací scénáře.

Tabulka 5.4: Test case 4 - Spuštění editoru grafů

TC4: Spuštění tutoriálu		
Popis scénáře	Testovací případ slouží k otestování spuštění editoru grafů ve výukové aplikaci	
Use case	UC1	
Předpoklady	Aplikace je spuštěna a uživatel je na domovské obrazovce.	
ID	**Krok testu**	**Očekávaný výsledek**
Test case pokračuje na další stránce		

Tabulka 5.4 – Test case 4 pokračování

ID	Krok testu	Očekávaný výsledek
1.	Uživatel přejde kurzorem myši na tlačítko *Síťový graf* v menu aplikace.	Zobrazí se rolovací podmenu s tlačítky *Nový graf* a *Načíst graf*
2.	Uživatel zvolí položku *Nový graf*	Aplikace zobrazí editor grafů včetně tlačítek pro jeho ovládání.
3.	Uživatel klikne pravým tlačítkem myši na volnou plochu editoru grafů.	Zobrazí se kontextové menu s následujícími položkami: undo, redo, přidat uzel, odebrat vybrané.

5.2.0.5 TC5: Přidání a odebrání uzlu síťového grafu

Tento scénář se zabývá přidáváním uzlů do síťového grafu a odebíráním uzlů ze síťového grafu. Přidávání a odebírání uzlů představuje základní funckionalitu editoru síťových grafů. Do grafu lze uzly přidávat buď pomocí tlačítka pro přidání uzlu nebo pomocí položky v kontextovém menu. Je také potřeba ověřit, že se přidávané uzly řádně očíslují a to i v případě, že je uzel odstaněn. Editor grafů by měl vždy přidat uzel s takovým číslem, které v posloupnosti chybí, tak aby nevznikaly v očíslování mezery.

Tabulka 5.5: Test case 5 - Přidání a odebrání uzlu síťového grafu

TC5: Přidání a odebrání uzlu síťového grafu	
Popis scénáře	Testovací případ slouží k otestování přidávání uzlů a odebírání uzlů síťového grafu
Use case	UC2, UC4
Předpoklady	Aplikace je spuštěna a uživatel má otevřený prázdný editor síťových grafů.

ID	Krok testu	Očekávaný výsledek
1.	Uživatel stiskne tlačítko *Nový uzel*.	Na plochu editoru je přidán nový uzel s pořadovým číslem 1.
2.	Uživatel stiskne pravé tlačítko myši na prázdné ploše editoru grafů	Zobrazí se kontextové menu s položkami: undo, redo, přidat uzel, odstranit vybrané.
3.	Uživatel klikne levým tlačítkem myši na polozku *Přidat uzel*.	V místě vyvolání kontextového menu z bodu 2) je vytvořen nový uzel s pořadovým číslem 2
Test case pokračuje na další stránce		

Tabulka 5.5 – Test case 5 pokračování

ID	Krok testu	Očekávaný výsledek
4.	Uživatel klikne pravým tlačítkem myši na uzel s pořadovým číslem 1.	V místě kliknutí se zobrazí kontextové menu s položkami: editovat uzel, označit kritický, odstranit, undo, redo.
5.	Uživatel klikne levým tlačítkem myši na položku *Odstranit*.	Uzel je odstraněn z grafu.
6.	Uživatel stiskne tlačítko *Nový uzel*.	Na plochu editoru je přidán nový uzel s pořadovým číslem 1.
7.	Uživatel klikne levým tlačítek myši na uzel s pořadovým číslem 2.	Aplikace provede zvýraznění uzlu změnou barvy okraje uzlu a zdvojením okrajové čáry.
8.	Uživatel stiskne klávesu *Delete*.	Uzel je odstraněn z grafu.
9.	Uživatel stiskne tlačítko *Nový uzel*.	Na plochu editoru je přidán nový uzel s pořadovým číslem 2.

5.2.0.6 TC6: Editace uzlu síťového grafu

Tento scénář se zabývá změnou ohodnocení (editací hodnot) uzlu v síťovém grafu. V uzlu lze editovat hodnotu nejdřívějšího možného počátku aktivit t_0^i a hodnotu nejzazšího přípustného dokončení aktivit t_1^i. Je tedy potřeba ověřit, že tyto hodnoty lze řádně změnit: musí se jednat o celé kladné číslo nebo o prázdný řetězec pro smazání hodnoty. Formulář pro editaci uzlu by také měl ukazovat stávající hodnoty, jež jsou v uzlu uložené.

Tabulka 5.6: Test case 6 - Editace uzlu grafu

TC6: Editace uzlu síťového grafu		
Popis scénáře	Testovací případ slouží k otestování editace uzlů síťového grafu	
Use case	UC2, UC4	
Předpoklady	Aplikace je spuštěna a uživatel má otevřený prázdný editor síťových grafů.	
ID	**Krok testu**	**Očekávaný výsledek**
1.	Uživatel stiskne tlačítko *Nový uzel*.	Na plochu editoru je přidán nový uzel s pořadovým číslem 1.
2.	Uživatel klikne pravým tlačítkem myši na uzel s pořadovým číslem 1.	V místě kliknutí se zobrazí kontextové menu s položkami: editovat uzel, označit kritický, odstranit, undo, redo.
Test case pokračuje na další stránce		

Tabulka 5.6 – Test case 6 pokračování

ID	Krok testu	Očekávaný výsledek
3.	Uživatel zvolí položku *Editovat uzel*.	Pod uzlem se zobrazí formulář pro zadání hodnot t_0^1 a t_1^1. Editovaný uzel je zvýrazněný dvojitým okrajem.
4.	Uživatel vyplní hodnoty $t_0^1 = 0$ a $t_1^1 = 1$ a poté stiskne tlačítko *OK*.	Formulář se zavře a v uzlu se objeví zadané hodnoty, v levém poli hodnota t_0^1 a v pravém t_1^1.
5.	Uživatel zopakuje body 2) a 3) tohoto testovacího scénáře.	Zobrazí se formulář pro editaci uzlu s předvyplněnými hodnotami $t_0^1 = 0$ a $t_1^1 = 2$.
6.	Uživatel změní hodnotu $t_0^1 = 2$ a $t_0^1 = 3$ a poté stiskne tlačítko *OK*.	Formulář se zavře a v uzlu se objeví nově zadané hodnoty.
7.	Uživatel znovu otevře formulář pro editaci uzlu zopakováním kroků 2) a 3) tohoto testovacího scénáře.	Zobrazí se formulář pro editaci uzlu s předvyplněnými hodnotami $t_0^1 = 2$ a $t_1^1 = 3$.
8.	Uživatel změní hodnotu $t_0^1 = 3$ a poté stiskne tlačítko *Zrušit*.	Formulář se zavře a hodnoty se nezmění $t_0^1 = 2$ a $t_1^1 = 3$.
9.	Uživatel znovu otevře formulář pro editaci uzlu zopakováním kroků 2) a 3) tohoto testovacího scénáře.	Zobrazí se formulář pro editaci uzlu s předvyplněnými hodnotami $t_0^1 = 2$ a $t_1^1 = 3$.
10.	Uživatel smaže hodnoty t_0^1 a t_1^1 a stiskne tlačítko *OK*.	Formulář se zavře a hodnoty t_0^1 a t_1^1 budou smazané.

5.2.0.7 TC7: Přidání a odebrání hrany síťového grafu

Tento testovací scénář se zabývá přidáváním a odebíráním hran v síťovém grafu. Hrany lze přidávat pouze mezi existující uzly. Editor síťových grafů validuje vznik nových hran, tak aby mezi dvěma uzly nemohlo být více hran a také nepodporuje smyčky vedoucí a končící v témže uzlu. Pokročilejší validace vzniku cyklů je provedena až při uživatelem požadované validaci grafu nebo při některých výpočtech.

Tabulka 5.7: Test case 7 - Přidání hrany do uzlu grafu

TC7: Přidání a odebrání hrany síťového grafu		
Popis scénáře	Testovací případ slouží k otestování přidávání a odebírání hran síťového grafu	
Use case	UC5, UC7	
Předpoklady	Aplikace je spuštěna a uživatel má otevřený prázdný editor síťových grafů.	
ID	**Krok testu**	**Očekávaný výsledek**
1.	Uživatel stiskne tlačítko *Nový uzel*.	Na plochu editoru je přidán nový uzel s pořadovým číslem 1.
2.	Uživatel stiskne tlačítko *Nový uzel*.	Na plochu editoru je přidán nový uzel s pořadovým číslem 2.
3.	Uživatel posune uzel s pořadovým číslem 2, tak aby se uzly nepřekrývaly.	Uzel číslo 2 byl přesunut po pracovní ploše editoru grafů.
4.	Uživatel přejede kurzorem myši na uzel číslo 1.	Na uzlu s pořadovým číslem 1 se zobrazí tlačítko pro vytvoření hrany.
5.	Uživatel stiskne tlačítko pro přidání hrany levým tlačítkem myši a začne táhnout hranu.	Počáteční uzel (s pořadovým číslem 1) je zvýrazněn a hrana je vykreslena.
6.	Uživatel táhne hranu k uzlu s číslem 2.	Jakmile je tažená hrana dost blízko od uzlu s pořadovým číslem 2, je hrana přichycena k uzlu a koncový uzel je zvýrazněn.
7.	Uživatel pustí levé tlačítko myši.	Hrana je vytvořena mezi uzly s pořadovými čísly 1 a 2. Hrana je tvořena nepřerušovanou linkou se šipkou u koncového uzlu č. 2.
8.	Uživatel zkusí vytvořit novou hranu mezi uzly s čísly 1 a 2 zopakováním bodů 4) - 6).	Nová hrana není vytvořena a uživateli se zobrazí chybová hláška varující před vznikem multigrafu.
9.	Uživatel klikne pravým tlačítkem myši na vytvořenou hranu.	Zobrazí se kontextová nabídka s následujícími položkami: editovat hranu, označit kritický, odstranit, undo, redo
10.	Uživatel zvolí položku *Odstranit*.	Hrana je odstraněna ze síťového grafu.
Test case pokračuje na další stránce		

Tabulka 5.7 – Test case 7 pokračování

ID	Krok testu	Očekávaný výsledek
11.	Uživatel vytvoří novou hranu mezi uzly s čísly 1 a 2, obdobně jako v krocích 4) - 7)	Hrana je vytvořena mezi uzly s pořadovými čísly 1 a 2. Hrana je tvořena nepřerušovanou linkou se šipkou u koncového uzlu č. 2.
12.	Uživatel klikne levým tlačítkem myši na vytvořenou hranu.	Hrana je zvýrazněna modrou barvou.
13.	Uživatel stiskne tlačítko *Delete*.	Zvýrazněná hrana je odstraněna ze síťového grafu.
14.	Uživatel vytvoří novou hranu mezi uzly s čísly 1 a 2, obdobně jako v krocích 4) - 7)	Hrana je vytvořena mezi uzly s pořadovými čísly 1 a 2. Hrana je tvořena nepřerušovanou linkou se šipkou u koncového uzlu č. 2.
15.	Uživatel klikne levým tlačítkem myši na uzel č. 1 a stiskne klávesu *delete*.	Uzel č. 1 je odstraněn, hrana vedoucí mezi uzly č. 1. a 2. je také odstraněna.

5.2.0.8 TC8: Editace hodnot hrany síťového grafu

Tento testovací scénář se zabývá změnou ohodnocení hran v editoru síťových grafů. U hran lze editovat popis hrany, dobu trvání aktivy, volnou a časovou rezervu. V rámci editace těchto parametrů je také validována správnost zadaných hodnot. Pro dobu trvání a rezervy to znamená buď kladné celé číslo nebo prázdný řetězec pro smazání. V rámci editace hrany lze také změnit její typ z aktivity na fiktivní hranu pomocí přepínače. Pokud je hrana takto změněna, zobrazuje se u ní pouze popisek a doba trvání a je také změněn její vzhled. Formulář pro editaci hran by měl zobrazovat stávající hodnoty, u fiktivní hrany lze změnit pouze popis a dobu trvání.

Tabulka 5.8: Test case 8 - Přidání hrany do uzlu grafu

TC8: Editace hodnot hrany síťového grafu		
Popis scénáře	Testovací případ slouží k otestování editace hran v síťovém grafu	
Use case	UC6	
Předpoklady	Aplikace je spuštěna a uživatel má otevřený prázdný editor síťových grafů. V editoru grafů byly vytvořeny dva uzly (č. 1 a č. 2) a hrana obdobně jako v TC7.	
ID	**Krok testu**	**Očekávaný výsledek**
Test case pokračuje na další stránce		

Tabulka 5.8 – Test case 8 pokračování

ID	Krok testu	Očekávaný výsledek
1.	Uživatel klikne pravým tlačítkem myši na vytvořenou hranu.	Zobrazí se kontextová nabídka s následujícími položkami: editovat hranu, označit kritický, odstranit, undo, redo.
2.	Uživatel zvolí položku *Editovat hranu*.	Hrana je zvýrazněna, je zobrazen formulář s poli pro editaci hodnot hrany: typ (aktivita), description, duration, VR a CR.
3.	Uživatel vyplní formulář: description (aktivita), duration (10), VR (0), CR (1) a stiskne tlačítko *OK*.	Zadané hodnoty jsou zobrazeny na hraně. Hodnota description (aktivita) zobrazena nad hodnotami DUR: 10, CR: 1 a VR: 0.
4.	Uživatel znovu otevře formulář pro editaci hrany zopakováním bodů 1) - 2).	Zobrazí se formulář pro editaci hran s předvyplněnými hodnotami: typ (aktivita), description (aktivita), DUR (10), CR (1) a VR (0).
5.	Uživatel změní hodnoty ve formuláři a stiskne tlačítko *Zrušit*.	Hodnoty zobrazené na hraně nebyly změněny: typ (aktivita), description (aktivita), DUR (10), CR (1) a VR (0).
6.	Uživatel znovu otevře formulář pro editaci hrany zopakováním bodů 1) - 2), změní hodnotu VR na 3 a stiskne tlačítko *OK*.	Hodnota VR hrany je změněna, zbytek hodnot zůstal stejný: typ (aktivita), description (aktivita), DUR (10), CR (1) a VR (3).
7.	Uživatel znovu otevře formulář pro editaci hrany zopakováním bodů 1) - 2), změní typ aktivity na *fiktivní* a stiskne tlačítko *OK*.	Hrana je změněna na fiktivní a je zobrazena přerušovanou linkou, na hraně se zobrazuje pouze její popis - description (aktivita) a hodnota DUR (10).
8.	Uživatel znovu otevře formulář pro editaci hrany zopakováním bodů 1) - 2), změní typ aktivity na *aktivita* a stiskne tlačítko *OK*.	Hrana je změněna na aktivitu a je zobrazena nepřerušovanou linkou, na hraně jsou zobrazeny všechny hodnoty: description (aktivita), DUR (10), CR (1) a VR (3).
9.	Uživatel znovu otevře formulář pro editaci hrany zopakováním bodů 1) - 2) a smaže hodnotu DUR.	Hodnota hrany je smazána, zbytek hodnot zůstal nezměněn: description (aktivita), DUR (), CR (1) a VR (3).

5.2.0.9 TC9: Označení kritických a nekritických prvků v grafu

Tento test case se zabývá označením prkvů síťového grafu jako kritických a odebráním tohoto označení. Kritické prvky jsou označeny k tomu určenou fialovou barvou, červená barva se používá pro vyznačení chybných prvků v grafu. U uzlů má označení formu fialového okraje u hran je pak celá hrana vyvedena ve fialové barvě.

Tabulka 5.9: Test case 9 - Označení kritičnosti a odebrání označení prvků

TC9: Označení kritičnosti a odebrání označení prvků	
Popis scénáře	Testovací případ slouží k otestování označování a zrušení označení kritických prvků v grafu (uzly a hrany).
Use case	UC8, UC9
Předpoklady	Aplikace je spuštěna a uživatel má otevřený prázdný editor síťových grafů. V editoru grafů byly vytvořeny dva uzly (č. 1 a č. 2) a hrana obdobně jako v TC7.

ID	**Krok testu**	**Očekávaný výsledek**
1.	Uživatel klikne pravým tlačítkem myši na hranu v síťovém grafu.	Zobrazí se kontextová nabídka s následujícími položkami: editovat hranu, označit kritický, odstranit, undo, redo.
2.	Uživatel zvolí položku *Označit kritický*.	Hrana je zvýrazněna fialovou barvou, uzly nejsou nijak zvýrazněny.
3.	Uživatel klikne pravým tlačítkem na uzel s číslem 1 v síťovém grafu.	Zobrazí se kontextová nabídka s následujícími položkami: editovat uzel, označit kritický, odstranit, undo, redo.
4.	Uživatel zvolí položku *Označit kritický*.	Uzel číslo 1 je zvýrazněn fialovou barvou.
6.	Uživatel klikne pravým tlačitkem na hranu vedoucí mezi uzly č. 1 a 2.	Zobrazí se kontextová nabídka s následujícími položkami: editovat hranu, odebrat kritický, odstranit, undo, redo.
7.	Uživatel vybere položku *Odebrat kritický*.	Hrana není zvýrazněna fialovou barvou.
Test case pokračuje na další stránce		

Tabulka 5.9 – Test case 9 pokračování

ID	Krok testu	Očekávaný výsledek
6.	Uživatel klikne pravým tlačítkem na uzel číslo 1.	Zobrazí se kontextová nabídka s následujícími položkami: editovat uzel, odebrat kritický, odstranit, undo, redo.
7.	Uživatel vybere položku *Odebrat kritický*.	Uzel není zvýrazněn fialovou barvou.
8.	Uživatel vybere všechny prvky v grafu (uzly č. 1 a 2 a hranu, která vede mezi nimi), a to tak, že drží klávesu *L-CTRL* a držením levého tlačítka myši označí všechny prvky v grafu.	Označené prvky jsou zvýrazněny: uzly dvojitým okrajem a modrou bravou, hrana modrou barvou.
9.	Uživatel klikne pravým tlačítkem myši na libovolný prvek grafu a v kontextovém menu zvolí položku *Označit kritický*.	Uzly č. 1 a 2 jsou vyznačeny fialovou bravou a dvojitým okrajem, šipka hrany je vyznačena fialovou barvou.
10.	Uživatel klikne na prázdnou plochu grafu, čímž zruší výběr prvků.	Barva linky hrany se změní z modré na fialovou, uzly nejsou označeny dvojitým okrajem.
11.	Uživatel označí prvky v grafu obdobně jako v bodě 8).	Uzly v grafu jsou vyznačeny dvojitým okrajem, hrana modrou linkou a fialovou šipkou.
12.	Uživatel klikne pravým tlačítkem myši na libovolný prvek grafu a v kontextovém menu zvolí položku *Odebrat kritický*.	Uzly č. 1 a 2 jsou vyznačeny modrou barvou s dvojitým okrajem, hrana je vyznačena modrou barvou.
13.	Uživatel klikne na prázdnou plochu grafu, čímž zruší výběr prvků	Barevné vyznačení prvků je zrušeno a uzly nejsou označeny dvojitým okrajem.

5.2.0.10 TC10: Uložení a načtení síťového grafu

Tento test case se zabývá uložením síťového grafu do souboru a načtením grafu ze souboru. Uložení grafu uloží všechny prvky v grafu včetně jejich pozice, ohodnocení a barevného zvýraznění. Validace samotného uloženého souboru by byla obtížná, jde totiž o JSON soubor, který obsahuje informace potřebné k rekonstrukci grafu. Kromě dat jde například i o informace o pozici nebo o zvýraznění prvku v grafu. Validace by tedy vyžadovala znalost tohoto souboru a také by ruční kontrola testerem byla náchylnější na chyby. Z tohoto důvodu je ověření uložení provedeno načtením souboru v aplikaci a vizuální kontrolou.

Tabulka 5.10: Test case 10 - Uložení a načtení síťového grafu

TC10: Označení kritičnosti a odebrání označení prvků	
Popis scénáře	Testovací případ slouží k otestování označování a zrušení označení kritických prvků v grafu (uzly a hrany).
Use case	UC10, UC11
Předpoklady	Aplikace je spuštěna a uživatel má otevřený prázdný editor síťových grafů. V editoru grafů byly vytvořeny dva uzly (č. 1 a č. 2) a hrana obdobně jako v TC7.

ID	**Krok testu**	**Očekávaný výsledek**
1.	Uživatel klikne na tlačítko *Uložit* v editoru síťových grafů.	Zobrazí se nabídka pro vložení názvu souboru.
2.	Uživatel ponechá výchozí název *myGraph.json* a stiskne tlačítko *OK*.	Webový prohlížeč, ve kterém je aplikace spuštěna, provede stažení souboru.
3.	Uživatel najede kurzorem myši na položku *Síťový graf* v hlavním menu aplikace. Ve zobrazeném podmenu zvolí položku *Nový graf*	Všechny prvky stávajícího grafu jsou smazány.
4.	Uživatel stiskne tlačítko *Nový uzel*.	Do grafu je vložen uzel s pořadovým číslem 1.
5.	Uživatel najede kurzorem myši na položku *Síťový graf* v hlavním menu aplikace. Ve zobrazeném podmenu zvolí položku *Načíst graf*.	Zobrazí se dialog pro vybrání souboru.
6.	Uživatel vybere v úložišti soubor uložený v bodě 2) tohoto testovacího scénáře.	Stávající uzel číslo 1 je smazán a nahrazen prvky načtenými ze rouboru: uzly s pořadovými čísly 1 a 2 a hrana vedoucí z uzlu č.1 do uzlu č. 2. se šipkou u uzlu č.2, graf je vycentrován na načtené prvky. Prvky grafu jsou do něj pozicovány tak, jak v něm byly původně vytvořeny.
7.	Uživatel stiskne tlačítko *Nový uzel*.	Do grafu je vložen uzel s pořadovým číslem 3.
8.	Uživatel vytvoří novou hranu mezi uzly číslo 2 a 3, obdobně jako v bodech 5) – 7) v testovacím scénáři TC7 5.7.	Nová hrana byla přidána mezi uzly číslo 2 a 3.

Test case pokračuje na další stránce

Tabulka 5.11 – Test case 10 pokračování

ID	Krok testu	Očekávaný výsledek
9.	Uživatel uloží graf stiskem tlačítka *Uložit* a zvolí název souboru *graf2.json*.	Webový prohlížeč, ve kterém je aplikace spuštěna, provede stažení souboru.
10.	Uživatel najede kurzorem myši na položku *Síťový graf* v hlavním menu aplikace. Ve zobrazeném podmenu zvolí položku *Načíst graf*.	Zobrazí se dialog pro vybrání souboru.
11.	Uživatel stiskne tlačítko *Zrušit*, čímž zruší načtení souboru.	Jednotlivé prvky grafu zůstavají nezměněny: v grafu existují uzly č. 1, 2 a 3, hrana mezi uzly 1 a 2 a hrana mezi uzly 2 a 3.
12.	Uživatel najede kurzorem myši na položku *Síťový graf* v hlavním menu aplikace. Ve zobrazeném podmenu zvolí položku *Načíst graf* v dialogovém okně vybere soubor uložený v bodě 2) tohoto testovacího scénáře.	Uložený graf je načten: uzly č. 1 a 2 a hrana mezi nimi se šipkou v uzlu číslo 2.
13.	Uživatel provede editaci uzlu č. 1: $t_0^1 = 1$, $t_1^1 = 2$, editaci uzlu č. 2: $t_0^2 = 2$, $t_1^2 = 3$ a hrany vedoucí mezi uzly 1 a 2: description (aktivita), DUR (1), CR(1), VR(0).	Ohodnocení obou uzlů a hrany je změněno.
14.	Uživatel označí uzel č. 2 jako kritický.	Uzel číslo 2 je vyznačen fialovou barvou.
15.	Uživatel uloží graf stiskem tlačítka *Uložit* a zvolí název souboru *graf3.json*.	Webový prohlížeč, ve kterém je aplikace spuštěna, provede stažení souboru.
16.	Uživatel provede načtení souboru *graf2.json* uloženém v bodě 9) tohoto testovacího scénáře, načtení provede obdobně jako v bodech 5) a 6).	Graf je zrestaurován do stavu v bodě 9). V grafu jsou neohodnocené uzly č. 1, 2 a 3, v grafu jsou také neohodnocené hrany vedoucí z uzlu č. 1 do uzlu č. 2 a z ulzu č. 1 do uzlu č. 3.
Test case pokračuje na další stránce		

Tabulka 5.11 – Test case 10 pokračování

ID	Krok testu	Očekávaný výsledek
17.	Uživatel provede načtení souboru *graf3.json* uloženém v bodě 14) tohoto testovacího scénáře, načtení provede obdobně jako v bodech 5) a 6).	Graf je zrestaurován do stavu v bodě 14). V grafu jsou ohodnocené uzly č. 1 a 2. s hodnotami $t_0^1 = 1$, $t_1^1 = 2$, $t_0^2 = 2$, $t_1^2 = 3$. Mezi tyto uzly je také načtena ohodnocená hrana s následujícími hodnotami: description (aktivita), DUR (1), CR(1), VR(0). Uzel číslo 2 je vyznačen fialovou barvou jako kritický.

Uvedené testovací scénáře byly použity pro otestování základní funkcionality výukové aplikace. Dále jsem ještě provedl otestování jednotlivých funkcí síťového grafu. Jelikož toto testování obsahovalo různé pozitivní a negativní testy, byl by popis a množství jendotlivých testovacích scénářů rozsáhlý, a proto jsem ho v této práci neuvedl.

5.2.0.11 Testovací zařízení

Pro otestování rychlosti odezvy aplikace jsem zvolil následující přenosné počítače:

- Asus GL502VM: i5-6300HQ, 16 GB RAM, SSD, displej o úhlopříčce 15,6”a rozlišení 1920x1080
- Dell Latitude 5480: i5-7300U, 16 GB RAM, SSD, displej o úhlopříčce 14”a rozlišení 1920x1080

Nejde o nikterak výkonné stroje. Asus je sice vybaven čtyřjádrovým procesorem, ale má maximální frekvenci pouze 2,8 GHz při vytížení všech čtyř jader. Oproti tomu současný procesor stejné kategorie i5-9300H dosahuje při vytížení všech jader frekvence až 4 GHz a je navíc schopen zpracovat 8 vláken najednou. Dell s pouze dvoujádrovým procesorem i5-7300U svým výkonem patří již do kategorie levných a méně výkonných počítačů, navíc jde o notebook vybavený antivirem a šifrováním disku, kde procesor často i při nečinnosti dosahuje vysokého zatížení. Aplikace v obou případech reagovala na povely uživatele svižně a dostála požadavkům na plynulost a odezvu. Ani menší displej Dellu se neukázal jako problém při jejím používání, čitelnost textu i na tomto displeji byla v pořádku.

5.2.0.12 Webové prohlížeče

Webovou aplikaci jsem otestoval v následujících prohlížečích webových stránek. Tyto prohlížeče lze instalovat na různé operační systémy, takže by neměl být problém aplikaci používat na různých platformách.

- Google Chrome 81.0.4044.129
- Microsoft Edge 44.18362.449.0 (verze používající EdgeHTML)
- Microsoft Edge 81.0.416.72 (nový prohlížeč postavený na Chromium)
- Firefox 76.0.1

Nejvíce problematický se ukázal Microsoft Edge se starším vykreslovacím jádrem EdgeHTML. Tento prohlížeč byl součástí Windows 10 předtím, než byl v lednu 2020 vydán v nové verzi s vykreslovacím jádrem Chromium. Výuková aplikace nefungovala ve starší verzi MS Edge založené na jádře EdgeHTML, protože v této verzi nefungovaly některé prvky javascriptu ES6. Problém byl zejména v definici třídních funkcí pomocí *šipkové definice* (arrow function). Původně jsem kvůli tomu kód převedl z javascriptu ES6 do verze ES5 pomocí transpilátoru. Obdobný problém se ukázal i v prohlížeči Safari, který je součástí operačního systému macOS. Nakonec však stačilo problémové funkce přesunout přímo do konstruktorů tříd. Poté se aplikace ve starší verzi MS Edge spustila, ale nefungovala úplně korektně. V tutoriálech se totiž špatně vystylovaly nadpisy. Jde o problém způsobený tím, že tato starší verze prohlížeče Edge nepodporuje selektor *:scope*. Kromě tohoto menšího nedostatku aplikace v prohlížeči MS Edge založeném na vykreslovacím jádře EdgeHTML funguje správně dle očekávání. Nová verze založená na jádře Chromium funguje korektně, a proto bych doporučil použít spíše tuto verzi. U zmíněných verzí Firefoxu ani Chromu jsem na žádné problémy nenarazil.

5.3 Ekonomické zhodnocení

Ve výukové aplikace jsem nepoužil žádné knihovny ani technologie, které by vyžadovaly placenou licenci. V této části se tedy zaměřím na odhad pracnosti vývoje této aplikace. Pro tyto odhady se při vývoji softwaru používá jako jednotka tzv. *člověkoden* - označovaný zkratkou MD z anglického man day. Do vývoje aplikace je také potřeba zahrnout čas na pochopení metody kritické cesty a na sepsání tutoriálů a uživatelské příručky. Pro určení finančních nákladů na jeden MD jsem použil nabídku na vedení cvičení, která činila 400 Kč za 90 minutové cvičení. Z toho vychází hodinová sazba 267 Kč hrubé mzdy. Za jeden MD by pak zaměstnavatel zaplatil $267 * 8 * 1,338 = 2858$ Kč v superhrubé mzdě. Pokud bych uvažoval, že cvičící stráví hodinu svého času přípravou na každé cvičení, klesla by tím jeho hodinová sazba na 160 Kč, což je ale pro vývojáře, byť studenta brigádníka, nejspíše nepřijatelná částka. Jako

ideální kompromis mi příjde částka 230 Kč na hodinu hrubé mzdy, což dává 2461,92 Kč za MD ve mzdě superhrubé.

Pro jednotlivé části vývoje aplikace jsem odhadl pracnost následovně:

- Nastudování problematiky - 2 MD
- Návrh aplikace
  - Funkční požadavky, nefunkční požadavky, use case - 1,5 MD
  - GUI + wireframe - 1,5 MD
- Implementace - 7,5 MD
- Testování - 3 MD
- Sepsání tutoriálů - 2 MD
- Sepsání uživatelské příručky - 0,5 MD

Celková pracnost je 18 MD, což při 2461,92 Kč za MD činí 44 314,56 Kč. Jinými slovy by takovýto projekt zaměstnal vývojáře skoro na jeden pracovní měsíc.

Závěr

Cílem této závěrečné práce bylo navrhnout a vyvinout aplikaci pro podporu výuky metody kritické cesty. Provedl jsem identifikaci procesů, které by tato aplikace měla podporovat. Následovala analýza funkčních a nefunkčních požadavků. Z těchto požadavků jsem vytvořil jednotlivé případy užití. Uvedné případy užití popisují chování aplikace z pohledu uživatele a interakci mezi uživatelem a aplikací. Důležitou součástí návrhu aplikace bylo vytvoření návrhu uživatelského rozhraní prostřednictvím wireframe. Jako pozitivní hodnotím možnost propojit jednotlivé obrazovky wireframe v závislosti na akcích a přechodech mezi nimi a tím částečně simulovat chování aplikace ještě před její implementací. Pro architekturu aplikace jsem zvolil vzor model-view-controller, ve které je aplikace rozdělena na tři části. Komponenta view je zaměřena na práci s uživatelským rozhraním a práci se vstupy, které jdou přes uživatelské rozhraní. Controller komunikuje se zbývajícími částmi a reaguje na uživatelské vstupy, které nejdou přímo z view a nakonec komponenta model, která v sobě udržuje stav aplikace, provádí manipulaci s daty a výpočty.

Implementaci aplikace jsem v souladu s požadavky provedl formou webové aplikace, která bude spustitelná na koncových zařízeních uživatelů a nebude vyžadovat instalaci. Cílem byla jednoduchá distribuce, kdy si uživatel pouze stáhne aplikaci a může jí ihned používat, čímž se předejde zbytečnému plýtvání času na cvičeních. Další výhodou takto koncipované aplikace je mutliplatformnost, která je dána rozšířeností webových prohlížečů ve kterých byla aplikace testována.

Aplikace obsahuje dva tutoriály týkající se práce s aplikací: Úvod do aplikace, ve kterém se uživatel seznámí se základní navigací ve výukové aplikaci, a tutoriál s názvem Editor síťových grafů, který se věnuje jednotlivým funkcím editoru síťových grafů. Součástí aplikace jsou také výukové tutoriály pro jejichž obsah jsem použil znalosti získané v teoretické části této práce. Aplikace obsahuje tutoriál věnující se teorii grafů nutné pro pochopení síťového grafu, dále tutoriál věnující se samotné konstrukci síťového grafu aktivit projektu a tutoriál vysvětlující metodu kritické cesty a způsob, jakým se provádějí jed-

notlivé fáze výpočtů a vyznačení kritické cesty včetně vysvětlení významu kritické cesty v projektu.

V aplikaci jsem naimplementoval editor síťových grafů, který je určen uživatelům pro vytváření sítových grafů a vyzkoušení výpočtů metody kritické cesty. Editor obsahuje funkcionalitu týkající se metody kritické cesty a je schopen provádět výpočty jednotlivých fází této metody včetně vyznačení kritické cesty. Uživatel také může tento editor použít pro kontrolu jím zadaných hodnot. V editoru je také možné spustit animaci, která uživateli ukáže krok po kroku výpočet metody kritické cesty. Uživatel také může vytvořené grafy uložit pro pozdější načtení nebo vygenerovat obrázek grafu.

K otestování funkcionality aplikace jsem použil připravené testovací scénáře. Funkce, které nebyly popsány v těchto scénářích jsem otestoval za využití definic z teoretické části této práce, týká se to otestování provádění výpočtů metody kritické cesty a provádění validací. V rámci testování jsem také provedl heuristickou analýzu, jejímž cílem bylo ověřit použitelnost uživatelského rozhraní a odhalit případné nedostatky v této oblasti. Aplikace byla také otestována a vyzkoušena v různých webových prohlížečích, pro toto otestování jsem vybral Chrome, Firefox a Edge. Nakonec jsem ověřil rychlost odezvy aplikace otestováním na dvou starších přenosných počítačích, kde rychlost odezvy splnila stanovené cíle. V rámci testování jsem narazil na některé nedostatky a chyby, které jsem následně opravil. Aplikace je připravena k použití.

Vyvinutá výuková aplikace splňuje stanovené požadavky a cíle. Studentům přiblíží teorii potřebnou pro pochopení metody kritické cesty a umožňuje si tuto metodu rovnou prostřednictvím aplikace vyzkoušet.

Literatura

[1] Projekt - ManagementMania.com. [online]. Copyright © 2011 [cit. 01.03.2020]. Dostupné z: https://managementmania.com/cs/projekt

[2] Informační systém [online]. Copyright ©3 [cit. 02.03.2020]. Dostupné z: https://is.muni.cz/el/1421/jaro2018/VIKBA22/um/3_projektovy_management/03_Zivotni_cyklus_projektu_a_predprojektova_faze.pdf

[3] Teorie grafů - Vojtěch Hordějčuk. Vojta Hordějčuk aka voho - Software Engineer and Bedroom Music Producer [online]. Copyright © 2008 [cit. 02.03.2020]. Dostupné z: http://voho.eu/wiki/graf/

[4] Základní pojmy teorie grafů - University information system MENDELU [online]. [cit. 02.03.2020]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=19937

[5] Metody plánování projektů. [online]. [cit. 02.03.2020]. Dostupné z: http://promis.econ.muni.cz/lecture/2/2/3/

[6] MUKRI, Ismail. Project management professional (PMP) : the fundamentals. Canada, Ontario: Beacon Associates Publications, 2004. ISBN 09734738359780973473834.

[7] WEAVER, Patrick. THE ORIGINS OF MODERN PROJECT MANAGEMENT [online]. 18. 4. 2007, , 1 - 3 [cit. 2020-03-06]. Dostupné z: https://www.mosaicprojects.com.au/PDF_Papers/P050_Origins_of_Modern_PM.pdf

[8] Operační výzkum. JABLONSKÝ, Josef. Operační výzkum: kvantitativní modely pro ekonomické rozhodování. Třetí vydání. Mikulova 1572, 149 00 Praha 4: Professional Publishing, 2007, 191 - 198. ISBN 978-80-86946-44-3.

[9] Basic CPM CalculationsT [online]. [cit. 2020-03-06]. Dostupné z: https://www.mosaicprojects.com.au/PDF/Schedule_Calculations.pdf

[10] Žikovský, Pavel. Návrh uživatelských rozhraní - X. přednáška Form usability [přednáška]. Praha: ČVUT v Praze, předmět MI-NUR zimní semestr 2019.

[11] W3.CSS Tutorial - w3.schools.com [online]. [cit. 2020-05-10]. Dostupné z https://www.w3schools.com/w3css/default.asp

[12] FRANZ, M, CT LOPES, G HUCK, Y DONG, O SUMER a GD BADER. Cytoscape.js: a graph theory library for visualisation and analysis Bioinformatics (2016) [online]. 28. 9. 2015, 309 - 311, [cit. 2020-05-10]. Dostupné z http://bioinformatics.oxfordjournals.org/content/32/2/309.full.pdf

[13] Tippy.js. Tippy.js [online]. [cit. 2020-05-10]. Dostupné z: https://atomiks.github.io/tippyjs/

[14] Žikovský, Pavel. Návrh uživatelských rozhraní - 3. Testing without users [přednáška]. Praha: ČVUT v Praze, předmět MI-NUR zimní semestr 2019.

[15] 10 Usability Heuristics for User Interface Design - Nielsen Norman Group [online]. [cit. 11.05.2020]. https://www.nngroup.com/articles/ten-usability-heuristics/

Příloha **A**

Seznam použitých zkratek

CPM Critical Path Method

ES Early start time

EF Early finish time

LS Late start time

LF Late finish time

FF Free float

TF Total float

MD Man day

HTML Hypertext markup language

CSS Cascading style sheet

TC Test case

BFS Breadth-first search

DFS Depth-first search

FIFO First in, first out

MVC Model view controller

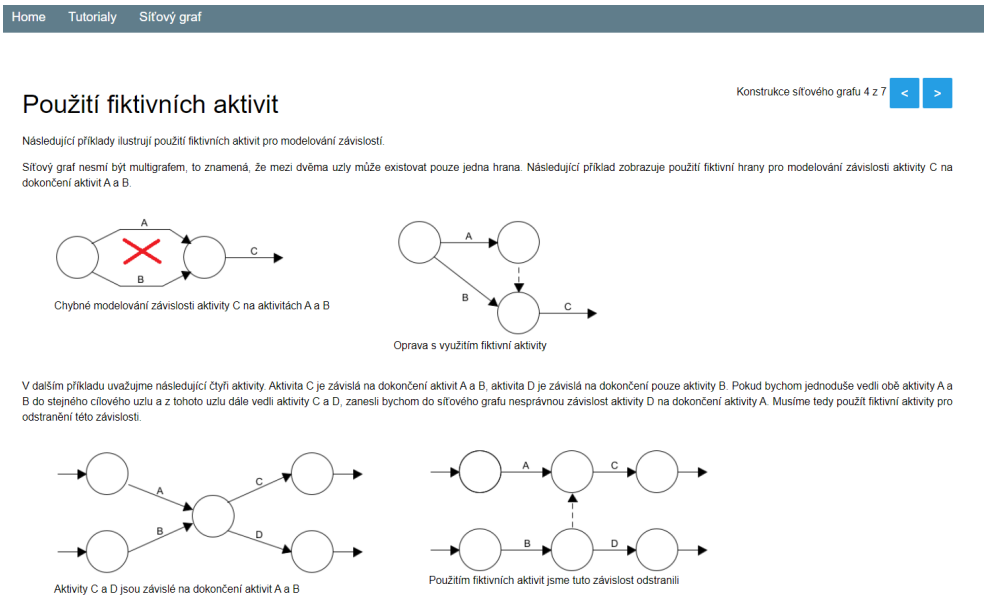
API Application programming interface

DOM Document object model

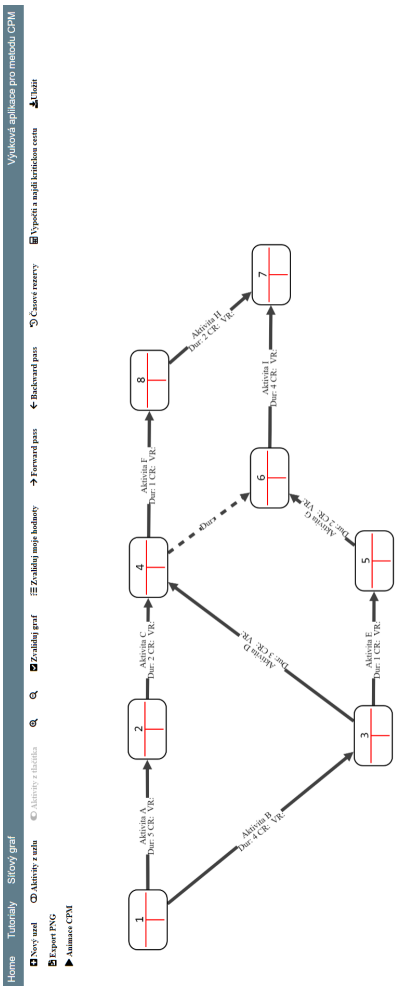
JSON JavaScript object notation

Příloha B

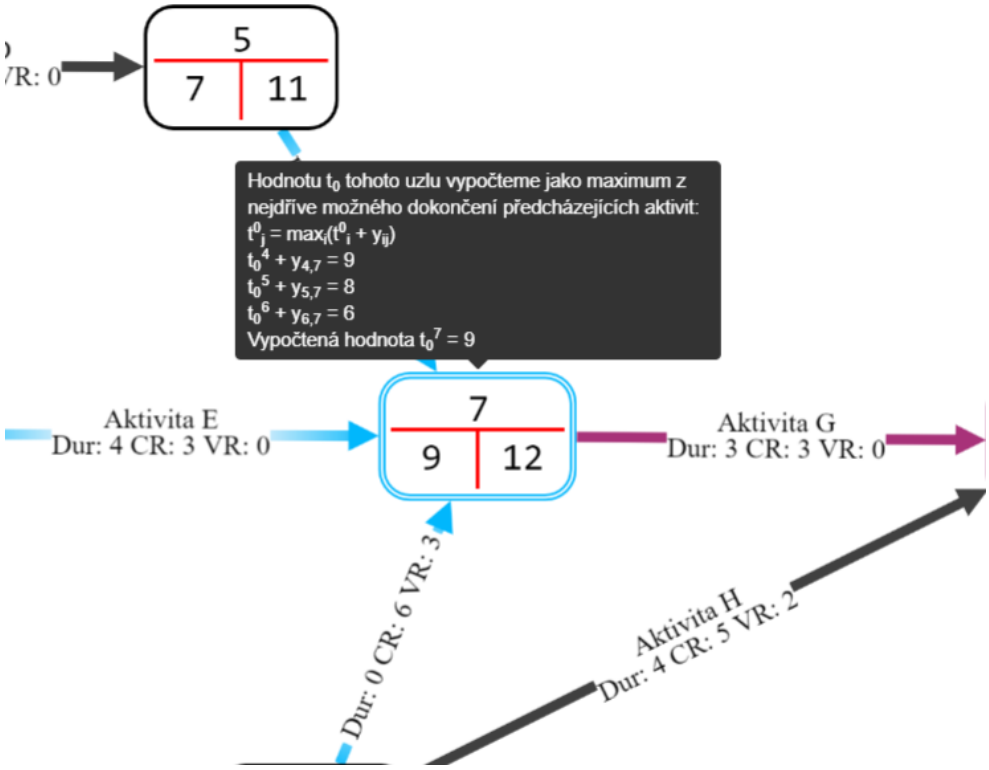
Ukázky aplikace



Obrázek B.1: Ukázka aplikace - stránka tutoriálu



Obrázek B.2: Ukázka aplikace - editor grafů



Obrázek B.3: Ukázka aplikace - ilustrace výpočtu

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `aplikace` ..... zdrojové kódy implementace
  - `index.html` ..... soubor pro spuštění aplikace
- `text` ..... text práce
  - `DP_Dolezalek_Tomas_2020.pdf` ..... text práce ve formátu PDF
- `thesis` ..... zdrojová forma práce ve formátu LaTeX