Vypracované otázky k MSZ pro rok 2022

Specializace NNET

20. dubna 2022

Vladimír Dušek, xdusek27

Specializace Počítačové sítě – NNET

1. Architektura superskalárních procesorů a algoritmy zpracování instrukcí mimo pořadí, predikce skoků.
2. Paměťová konzistence a předbíhání operací čtení a zápisu, podpora virtuálního adresového prostoru.
3. Datový paralelismus SIMD, HW implementace a SW podpora.
4. Architektury se sdílenou pamětí UMA a NUMA, zajištění lokality dat.
5. Problém koherence pamětí cache na systémech se sdílenou pamětí, protokol MSI.
6. Paralelní zpracování v OpenMP: Smyčky, sekce a tasky a synchronizační prostředky.
7. Pravděpodobnost, podmíněná pravděpodobnost, nezávislost.
8. Náhodná proměnná, typy náhodné proměnná, funkční a číselné charakteristiky, významná rozdělení pravděpodobnosti.
9. Bodové a intervalové odhady parametrů, testování hypotéz o parametrech.
10. Vícevýběrové testy, testy o rozdělení, testy dobré shody.
11. Regresní analýza.
12. Markovské řetězce a základní techniky pro jejich analýzu.
13. Randomizované algoritmy (Monte Carlo a Las Vegas algoritmy).
14. Problém generalizace strojového učení a přístup k jeho řešení (trénovací, validační a testovací sada, regularizace, předtrénování, multi-task learning, augmentace dat, dropout, ...)
15. Generativní modely a diskriminativní přístup ke klasifikaci (gaussovský klasifikátor, logistická regrese, ...)
16. Neuronové sítě a jejich trénování (metoda gradientního sestupu, účelová (loss) funkce, výpočetní graf, aktivační funkce, zápis pomocí maticového násobení, ...)
17. Neuronové sítě pro strukturovaná data (konvoluční a rekurentní sítě, motivace, základní vlastnosti, použití)
18. Prohledávání stavového prostoru (informované a neinformované metody, lokální prohledávání, prohledávání v nejistém prostředí, hraní her, CSP úlohy)
19. Klasifikace formálních jazyků (Chomského hierarchie), vlastnosti formálních jazyků a jejich rozhodnutelnost.
20. Konečné automaty (jazyky přijímané KA, varianty KA, minimalizace KA, Mihill-Nerodova věta).
21. Regulární množiny, regulární výrazy a rovnice nad regulárními výrazy.
22. Zásobníkové automaty (jazyky přijímané ZA, varianty ZA).
23. Turingovy stroje (jazyky přijímané TS, varianty TS, lineárně omezené automaty, vyčíslitelné funkce).
24. Nerozhodnutelnost (problém zastavení TS, princip diagonalizace a redukce, Postův korespondenční problém).
25. Časová a paměťová složitost (třídy složitosti, úplnost, SAT problém).
26. Postrelační a rozšířené relační databáze (objektový a objektově relační databázový model – struktura a operace; podpora práce s XML a JSON dokumenty v databázích).
27. NoSQL databáze (porovnání relačních a NoSQL; CAP věta a ACID/BASE principy; typy NoSQL databází; dotazování v NoSQL databázích; agregace dat pomocí Map-Reduce a agregační pipeline).
28. Získávání znalostí z dat (pojem znalost; typické zdroje dat; základní úlohy získávání znalostí; analytické projekty a proces získávání znalostí z dat).

29. Porozumění datům (důvod a cíl; popisné charakteristiky dat a vizualizační techniky; korelační analýza).
30. Prostorové DB (problematika mapování prostoru, ukládání, indexace; využití).
31. Indexace (nejen) v prostorových DB (kD-Tree a Grid File (a jejich varianty), R-Tree).
32. Lambda kalkul (definice všech pojmů, operací...).
33. Práce v lambda kalkulu (demonstrace reprezentace čísel a pravdivostních hodnot a operací nad nimi).
34. Haskell – lazy evaluation (typy v jazyce včetně akcí, uživatelské typy, význam typových tříd, demonstrace lazy evaluation).
35. Prolog – způsob vyhodnocení (základní princip, unifikace, chování vestavěných predikátů, operátor řezu – vhodné a nevhodné užití).
36. Prolog – změna DB/programu za běhu (demonstrace na prohledávání stavového prostoru, práce se seznamy).
37. Model PRAM, suma prefixů a její aplikace.
38. Distribuované a paralelní algoritmy – algoritmy nad seznamy, stromy a grafy.
39. Interakce mezi procesy a typické problémy paralelismu (synchronizační a komunikační mechanismy).
40. Distribuované a paralelní algoritmy – předávání zpráv a knihovny pro paralelní zpracování (MPI).
41. Distribuovaný broadcast, synchronizace v distribuovaných systémech.
42. Klasifikace a vlastnosti paralelních a distribuovaných architektur, základní typy jejich topologií.
43. Distribuované a paralelní algoritmy – algoritmy řazení, select, algoritmy vyhledávání.
44. Bezdrátové lokální sítě (Wifi, Bluetooth).
45. Hledání minimální kostry obyčejného grafu (pojmy, stromy a kostry, Kruskalův algoritmus, Primův algoritmus).
46. Hledání nejkratších cest ze zdrojového uzlu do všech ostatních uzlů grafu (Bellman-Fordův algoritmus, Dijkstrův algoritmus).
47. Klasifikace algoritmů volby koordinátora, algoritmus Bully a jeho složitost.
48. Podmínky konsistentního globálního stavu distribuovaného systému.
49. Principy distribuovaného zpracování MapReduce, průběh a jednotlivé operace distribuovaného výpočtu pomocí MapReduce, jeho implementace v Apache Hadoop a Apache Spark.
50. Symetrická kryptografie. Vlastnosti, vlastnosti bezpečného algoritmu, délka klíče, útok silou, příklady symetrických algoritmů, Feistelovy šifry, DES, režimy činnosti, proudové šifry.
51. Asymetrická kryptografie, vlastnosti, způsoby použití, poskytované bezpečnostní funkce, elektronický podpis a jeho vlastnosti, hybridní kryptografie, algoritmus RSA, generování klíčů, šifrování, dešifrování.
52. Hašovací funkce, klíčovaný haš a MAC a jejich použití a vlastnosti.
53. Správa klíčů v asymetrické kryptografii (certifikáty X.509).
54. Základní architektury přepínačů, algoritmy pro plánování, řešení blokování, vícestupňové přepínací sítě.
55. Základní funkce směrovače, zpracování paketů ve směrovači, typy přepínání a architektur.
56. Metody pro výpočet směrování v sítích (Bellman-Ford, Dijkstra, Path vector, DUAL).
57. Řízení toku dat (flow-control) a prevence zahlcení (congestion-control) na transportní vrstvě (MP-TCP, QUIC, SCTP, DCCP).
58. Metody detekce síťových incidentů (signatury, statistické metody) a nástroje (IDS/IPS).
59. Sítě Peer-to-Peer: vlastnosti, chování, způsoby směrování. Strukturované a nestrukturované sítě.

60. Události v JavaScriptu (smyčka událostí, asynchronní programování, klientské události, obsluha událostí)
61. Přenos a distribuce webových dat (URI, protokol HTTP, proudy HTTP, CDN, XHR)
62. Bezpečnost webových aplikací (SOP, XSS, CSRF, bezpečnostní hlavičky HTTP)

Obsah

Kapitola 1

Hledání minimální kostry obyčejného grafu (pojmy, stromy a kostry, Kruskalův algoritmus, Primův algoritmus).

1.1 Metadata

- Předmět: Grafové algoritmy (GAL)
- Přednáška:
  - 5) Stromy, minimální kostry, Jarníkův a Borůvkův algoritmus.
  - 6) Růst minimální kostry, algoritmy Kruskala a Prima.
- Záznam:
  - 2020-10-22
  - 2020-10-29

1.2 Úvod a kontext

Orientovaný graf Orientovaný graf je dvojice $G = (V, E)$, kde V je konečná množina uzlů a $E \subseteq V \times V$ je množina hran.

Neorientovaný graf Neorientovaný graf je dvojice $G = (V, E)$, kde V je konečná množina uzlů a $E \subseteq \binom{V}{2}$ je množina hran. (Hrana je tedy dvouprvková množina, avšak běžně se držíme stejného značení jako u orientovaných grafů a používáme dvojici.)

Ohodnocený graf Ohodnocený graf je takový graf, jehož každá hrana má přiřazenou nějakou hodnotu, typicky definovanou pomocí váhové funkce $w : E \mapsto \mathbb{R}$.

Podgraf Graf $G' = (V', E')$ je podgraf grafu $G = (V, E)$ jestliže $V' \subseteq V$ a $E' \subseteq E$.

Sled Posloupnost uzlů $\langle v_0, v_1, \ldots, v_k \rangle$, kde $(v_{i-1}, v_i) \in E$ pro $i = 1, \ldots, k$ se nazývá sled délky k z v_0 do v_k.

Uzavřený sled Sled $\langle v_0, v_1, \ldots, v_k \rangle$ se nazývá uzavřený, pokud existuje hrana (v_0, v_k).

Dosažitelnost Pokud existuje sled s z uzlu u do uzlu v, říkáme, že v je dosažitelný z u sledem s, značeno $u \overset{s}{\Rightarrow} v$.

Tah Tah je sled ve kterém se neopakují hrany.

Cesta Cesta je sled ve kterém se neopakují uzly.

Souvislý graf Neorientovaný graf se nazývá souvislý, pokud mezi libovolnými dvěma uzly existuje cesta.

Kružnice Uzavřená cesta se nazývá kružnice.

Cyklus Orientovaná kružnice se nazývá cyklus (první a poslední uzel je shodný).

Prostý graf Orientovaný graf bez cyklů se nazývá prostý.

Acyklický graf Graf je bez cyklů, resp. kružnic, se nazývá acyklický.

Strom Graf, který je souvislý a acyklický, se nazývá strom.

Kostra Strom, který tvoří podgraf souvislého grafu na množině všech jeho vrcholů, se nazývá kostra (*spanning tree*).

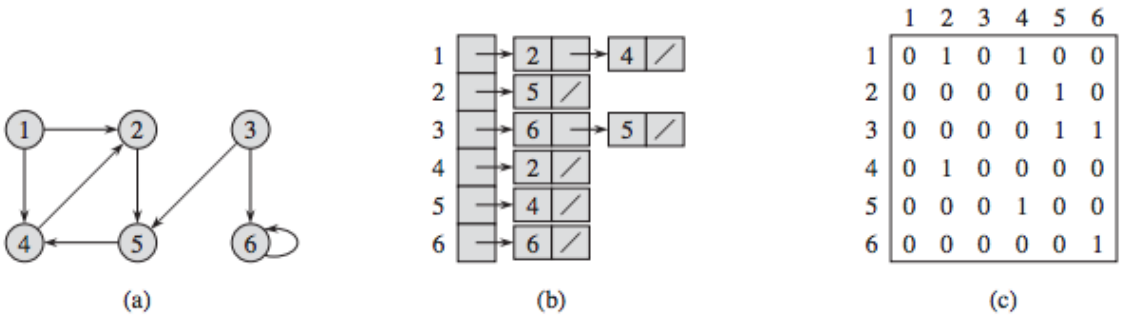
Minimální kostra Nechť $G = (V, E)$ je souvislý neorientovaný graf s váhovou funkcí $w : E \mapsto \mathbb{R}$. Minimální kostra (*MST, minimum spanning tree*) je strom $G' = (V, E')$, kde $E' \subseteq E$ a

$$w(E') = \sum_{(u,v) \in T} w(u, v)$$

je minimální ze všech možných alternativních koster.

Seznam sousedů Seznam sousedů (Adj, *adjacency list*) je reprezentace grafu v paměti. Jde o preferovanou variantu pro řídké grafy – kde $m << n^2$. Pro každý uzel máme definovaný seznam jeho sousedů.

Matice sousednosti Matice sousednosti (*adjacency matrix*) je reprezentace grafu v paměti. Jde o preferovanou variantu pro husté grafy – kde m je skoro n^2.



Obrázek 1.1: Příklad reprezentace grafu pomocí seznamu sousedů a matice sousednosti.

1.3 Generický algoritmus

Hledání minimální kostry je problém, který lze řešit algoritmy, které spadají do kategorie tzv. hladových (*greedy*) deterministických algoritmů. Spočívají v tom, že průběžně odhadují kostru přidáváním dalších hran a nikdy se nemusejí vracet (neprovádí se *backtracking*).

Generický algoritmus tvoří jakousi základní kostru pro další, už konkrétní, algoritmy.

Řez Nechť $G = (V, E)$ je graf. Řez grafu G je dvojice $(S, V - S)$, kde $\emptyset \subseteq S \subseteq V$.

Křížení Hrana $(u, v) \in E$ kříží řez $(S, V - S)$, pokud jeden její konec je v S a druhý v $V - S$.

Respektování Nechť $A \subseteq E$ je množina hran. Řez $(S, V - S)$ respektuje množinu hran A, pokud žádná hrana v A nekříží řez $(S, V - S)$.

Lehkost Nechť $(S, V - S)$ je řez a B je množina hran, která ho kříží. Hrana z množiny B s nejmenší hodnotou se nazývá lehká.

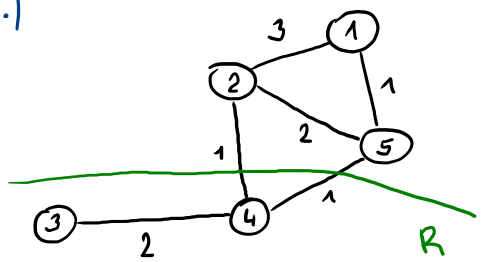
Bezpečnost Nechť $G = (V, E)$ je souvislý neorientovaný graf s reálnou váhovou funkcí w. Nechť $A \subseteq E$ je součástí nějaké minimální kostry G. Nechť $(S, V - S)$ je řez, který respektuje A. Nechť (u, v) je lehká hrana křížící $(S, V - S)$. Pak hrana (u, v) je bezpečná pro A.

```
1 def generic_mst(G):
2     # G je graf
3     # A je mnozina hran rozpracovane minimalni kostry
4     A = {}
5     while netvori_kostru(A, G):
6         for hrana in G.E:
7             if je_bezpecna(A, hrana):
8                 A += {hrana}
9     return A
```

Výpis 1.1: Generický algoritmus. Před každou iterací algoritmu je množina A podmnožinou nějaké minimální kostry. Hrana $(u, v) \in E$ je bezpečná pro A, pokud $A \cup \{(u, v)\}$ je podmnožinou nějaké minimální kostry.

Př. $G = (V, E)$ $W : E \to \mathbb{R}$ (definované obvyklým)

1.)



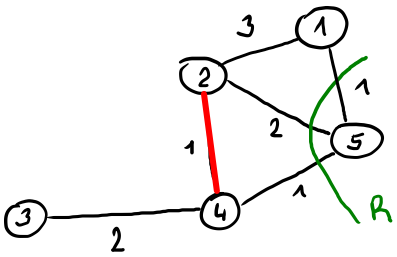
$A = \{\}$

$R = (\{1,2,5\}, \{3,4\})$

$LH = \{(2,4), (4,5)\}$

$A \leftarrow A \cup \{(2,4)\}$

2.)



$A = \{(2,4)\}$

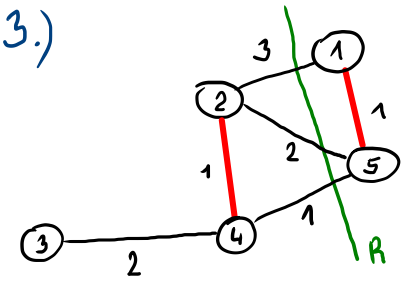
$R = (\{1,2,3,4\}, \{5\})$

$LH = \{(1,5), (4,5)\}$

$A \leftarrow A \cup \{(1,5)\}$

Obrázek 1.2: Příklad, část 1.

3.)



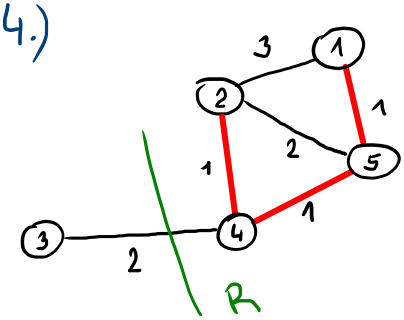
$A = \{(1,5), (2,4)\}$

$R = (\{1,5\}, \{2,3,4\})$

$LH = \{(4,5)\}$

$A \leftarrow A \cup \{(4,5)\}$

4.)



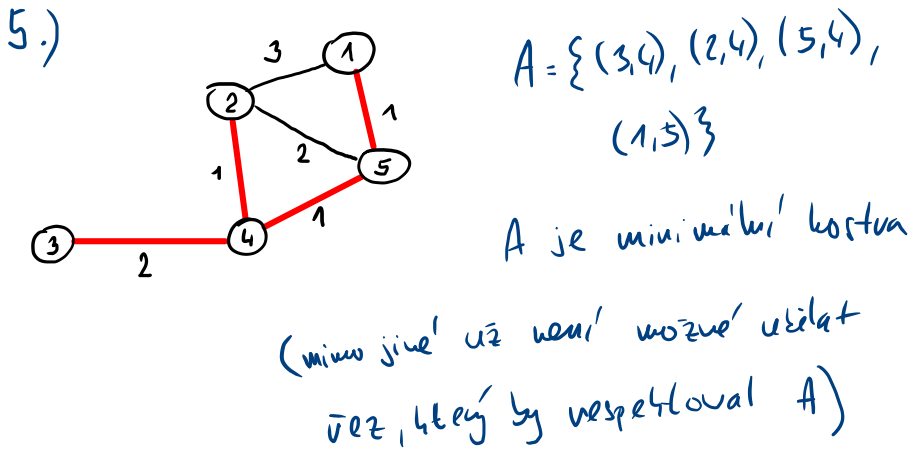
$A = \{(1,5), (2,4), (4,5)\}$

$R = (\{1,2,4,5\}, \{3\})$

$LH = \{(3,4)\}$

$A \leftarrow A \cup \{(3,4)\}$

Obrázek 1.3: Příklad, část 2.



Obrázek 1.4: Příklad, část 3.

1.4 Kruskalův algoritmus

Kruskalův a Primův algoritmus se liší v tom, jakým způsobem vybírají bezpečnou hranu. Kruskalův algoritmus nahlíží na A jako na les a hledá hranu s nejmenším ohodnocením, která spojuje stromy v lese. Na konci je A jeden strom.

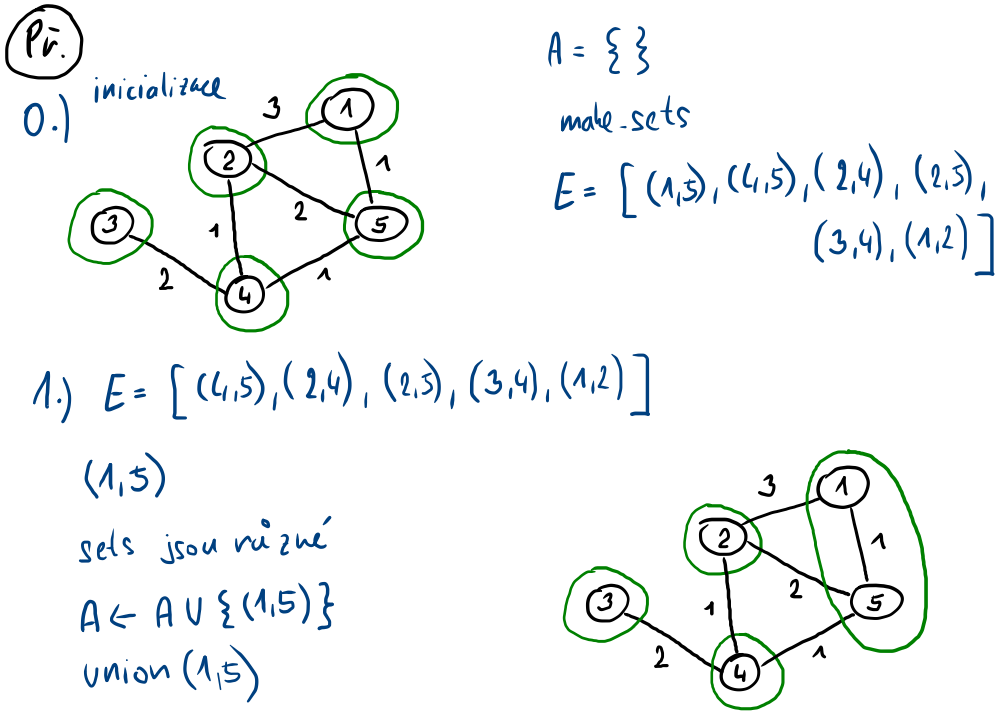
```
1  def kruskal_mst(G):
2      # G je graf
3      # A je mnozina hran rozpracovane minimalni kostry
4
5      # inicializace, kazdy uzel je ve sve mnozine
6      A = {}
7      for v in G.V:
8          make_set(v)
9
10     # seradit vzestupne podle w
11     E = sort(G.E, G.w)
12
13     for (u, v) in E:
14         if find_set(u) != find_set(v):
15             A += {(u, v)}
16             union(u, v)
17
18     return A
```

Výpis 1.2: Kruskalův algoritmus. Funkce `make_set(v)` vytvoří množinu obsahující v, `find_set` vrátí reprezentanta množiny v, `union(u ,v)` sjednotí dvě množiny obsahující u a v.

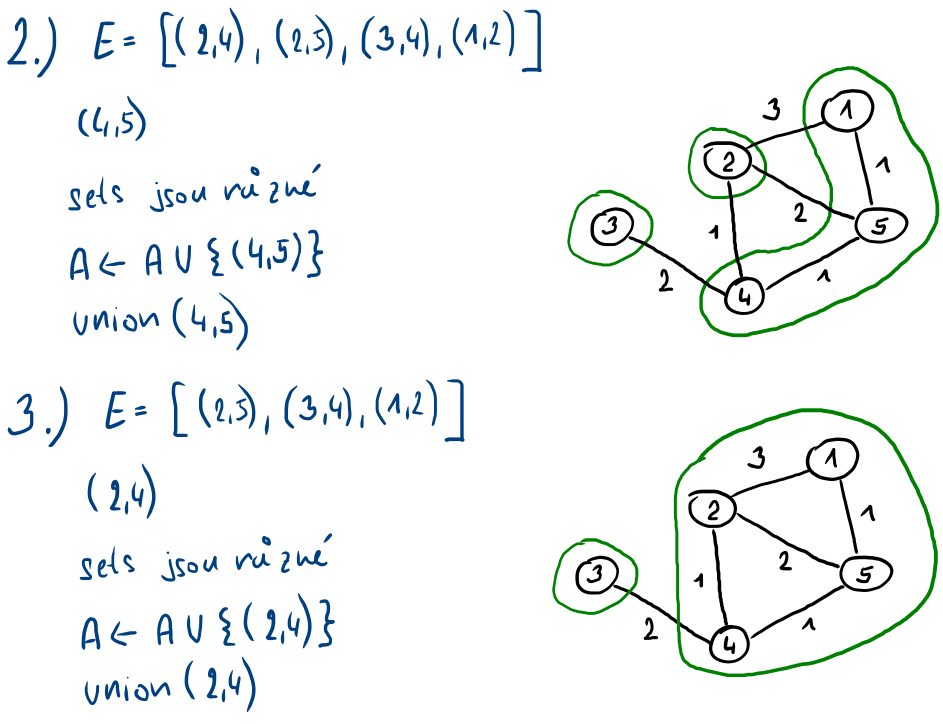
1.4.1 Složitost

- Řádek 3 – $O(1)$
- Řádek 6-7 – n-krát složitost *make_set* (n je počet uzlů).
- Řádek 10 – $O(m \cdot \log(m))$ (m je počet hran).
- Řádky 12-15 – Závisí na implementaci *find_set* a *union*.
  - Při implementaci seznamem s heuristickou celkem: $O(m + n \cdot \log(n))$.
  - Při stromové implementaci s váhami a zkratkami celkem: $O((m + n) \cdot \alpha(n))$. Kde α je velmi pomalu rostoucí funkce ($\alpha \leq 4$).
- Pro souvislý graf platí $m > n$. Proto množinové operace stojí $O(m \cdot \alpha(n))$. Jelikož $\alpha(n) = O(\log(n)) = O(\log(m))$, tak celková složitost je $O(m \cdot \log(m))$.
- Dále platí $m < n^2$, pak $\log(m) = O(\log(n))$, proto celkem: $O(m \cdot \log(n))$.

1.4.2 Příklad



Obrázek 1.5: Příklad, část 1.

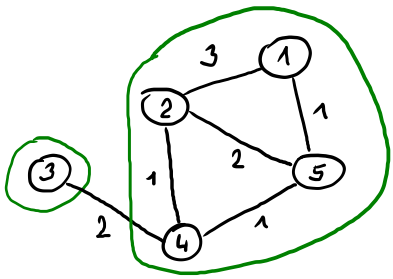


Obrázek 1.6: Příklad, část 2.

4.) $E = [(3,4), (1,2)]$

(2,5)

sets nejsou různé



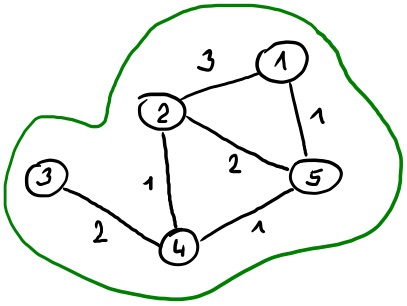
4.) $E = [(1,2)]$

(3,4)

sets jsou různé

$A \leftarrow A \cup \{(3,4)\}$

union(3,4)



5.) $E = [\,], (1,2)$, sets nejsou různé

Obrázek 1.7: Příklad, část 3.

1.5 Primův-Jarníkův algoritmus

Primův algoritmus buduje tzv. *A* strom. Má zadaný určitý uzel, ze kterého hledá nejbližší další uzel, který by připojil. A pak další a další.

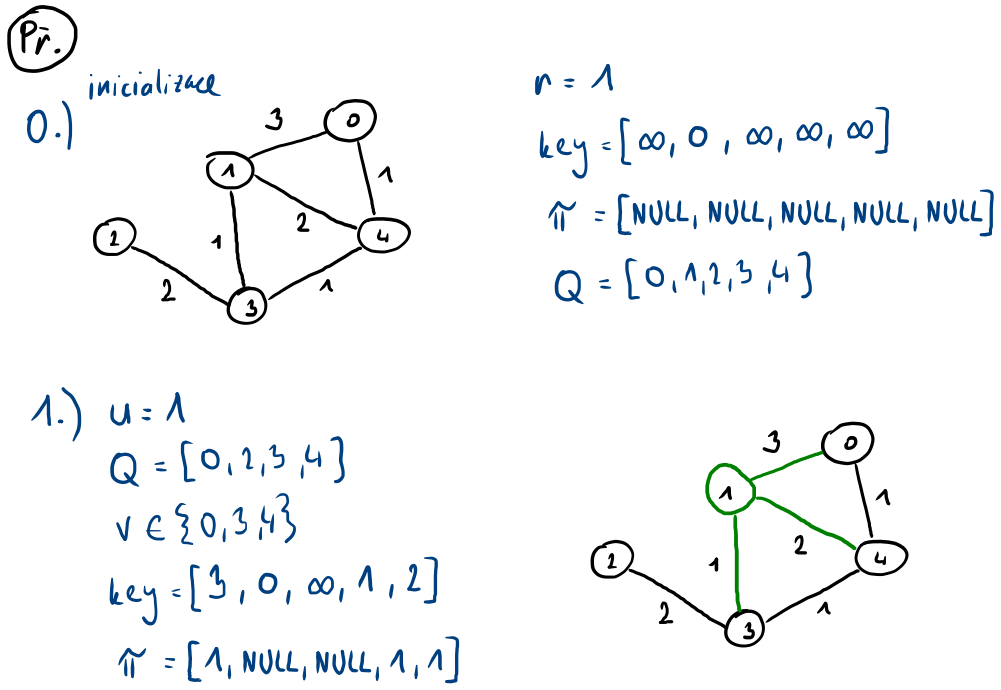
```
1  def prim_mst(G, r):
2      # G je graf
3      # r je vychozi uzel
4
5      for u in G.V:
6          key[u] = INF # pole cen prechodu, kolik stoji prechod do vrcholu na indexu
7          pi[u] = NULL # pole predchudcu, kdo je predchudce vrcholu na indexu
8
9      key[r] = 0
10     Q = Queue(G.V) # prioritni fronta uzlu
11
12     while not Q.empty():
13         u = Q.extract_min(key) # vrati prvek z Q s nejmensi hodnotou v key
14
15         # pro vsechny sousedy uzlu u (Adj je seznam sousedu)
16         for v in Adj[u]:
17             # pokud je levnejsi cesta a jeste to neni prozkoumany uzel
18             if v in Q and w(u, v) < key(v):
19                 pi[v] = u
20                 key[v] = w(u, v)
21                 Q.decrease_key(key) # aktualizace prioritni fronty
22
23     return pi
```

Výpis 1.3: Primův algoritmus.

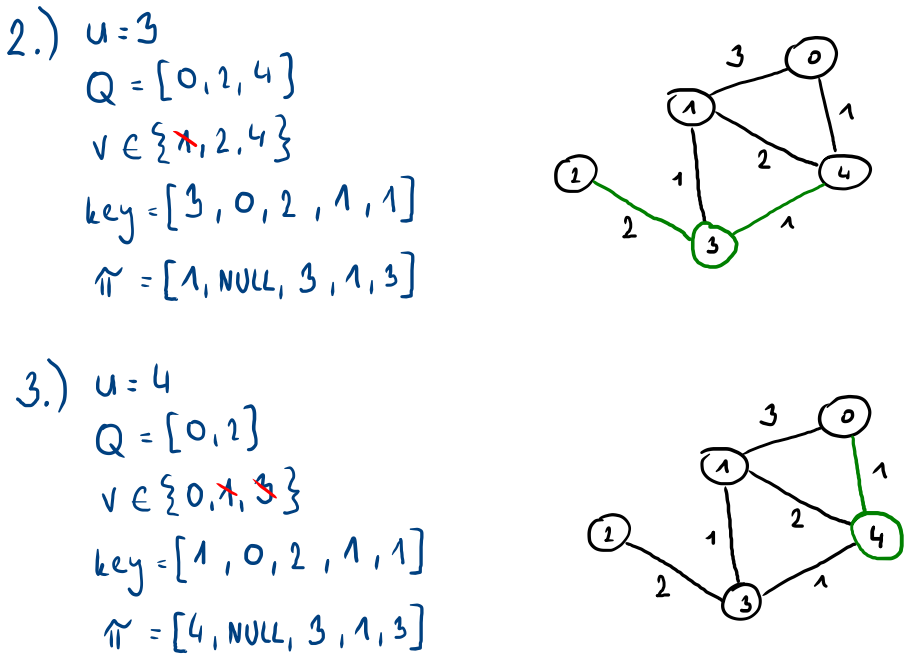
1.5.1 Složitost

- Řádky 4-12 – $O(n)$ za použití binární haldy (n je počet uzlů).
- Řádky 14-15 – While cyklus na řádku 14 se provede n-krát a protože *extract_min* stojí $O(\log(n))$, tak je celková složitost $O(n \cdot \log(n))$.
- Řádek 17 – For cyklus se provede $O(m)$ krát, protože dálka všech seznamů sousedů je dohromady $2m$ (m je počet hran).
- Řádek 19-21 – $O(1)$.
- Řádek 23 – stojí $O(\log(n))$, kvůli provedení operace *decrease_key* ve frontě Q.
- Jelikož $m > n$, tak celkem $O(n \cdot \log(n) + m \cdot \log(n)) = O(m \cdot \log(n))$.

1.5.2 Příklad

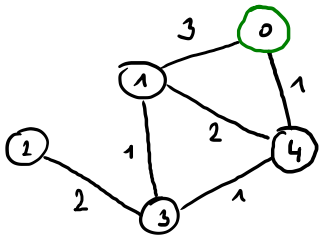


Obrázek 1.8: Příklad, část 1.

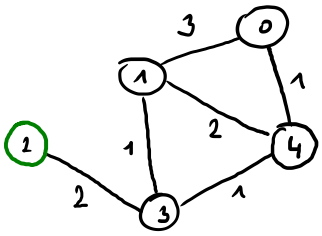


Obrázek 1.9: Příklad, část 2.

4.) u = 0
Q = [2]
v ∈ {~~1, 4~~}
key = [1, 0, 2, 1, 1]
π = [4, NULL, 3, 1, 3]

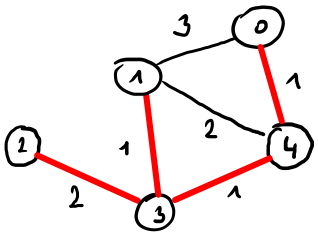


5.) u = 2
Q = [ ]
v ∈ {~~3~~}
key = [1, 0, 2, 1, 1]
π = [4, NULL, 3, 1, 3]



Obrázek 1.10: Příklad, část 3.

6.) key = [1, 0, 2, 1, 1]
π = [4, NULL, 3, 1, 3]



Obrázek 1.11: Příklad, část 4.

Kapitola 2

Hledání nejkratších cest ze zdrojového uzlu do všech ostatních uzlů grafu (Bellman-Fordův algoritmus, Dijkstrův algoritmus).

2.1 Metadata

- Předmět: Grafové algoritmy (GAL)
- Přednáška:
  - 7) Nejkratší cesty z jednoho vrcholu, Bellman-Fordův algoritmus, nejkratší cesta z jednoho vrcholu v orientovaných acyklických grafech.
  - 8) Dijkstrův algoritmus. Nejkratší cesty ze všech vrcholů.
- Záznam:
  - 2020-11-05

2.2 Úvod a kontext

Viz. „Úvod a kontext“ v předchozí otázce.

Cena cesty Nechť $G = (V, E)$ je ohodnocený graf s váhovou funkcí $w : E \mapsto \mathbb{R}$. Cena cesty $p = \langle v_o, v_1, \ldots, v_k \rangle$ je suma

$$w(p) = \sum_{i=0}^{k} w(v_i, v_{i+1})$$

.

Cena nejkratší cesty Cena nejkratší cesty z u do v je

$$\delta(u, v) = \begin{cases} min(\{w(p) : u \overset{\text{p}}{\Rightarrow} v\}) \\ \infty \text{ pokud cesta neexistuje} \end{cases}$$

.

Nejkratší cesta Nejkratší cesta z u do v je pak libovolná cesta p taková, že $w(p) = \delta(u, v)$.

Cena cesty se záporným cyklem Pokud na cestě z u do v existuje záporný cyklus (cyklus jehož celková cena je záporná), pak $\delta(u, v) = -\infty$.

Záporné ohodnocení hran Pokud na cestě z u do v neexistuje záporný cyklus, tak algoritmy pracují dobře i se záporným ohodnocením hran.

Reprezentace cesty Cestu reprezentujeme pomocí pole předchůdců π.

Hledání nejkratších cest ze všech uzlů do jednoho Tento problém lze řešit stejnými algoritmy. Graf se transponuje (převrácení orientace hran), provede se algoritmus pro problém „hledání nejkratších cest ze jednoho uzlu do všech ostatních uzlů“ a poté se transponuje zpět.

Reprezentace nejkratší cesty Nejkratší cestu grafu $G = (V, E)$ reprezentujeme pomocí pole předchůdců π, kde $\pi[v]$ označuje předchůdce uzlu $v \in V$ na nejkratší cestě. Podgraf předchůdců pak je $G_\pi = (V_\pi, E_\pi)$, $V_\pi = \{v \in V : \pi[v] \neq \text{NULL}\} \cup \{s\}$, $E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}$. V okamžiku dokončení algoritmu výpočtu nejkratších cest je G_π strom nejkratších cest. Tj. kořenový strom obsahující nejkratší cesty ze zdroje s do všech ostatních uzlů.

2.3 Pomocné funkce

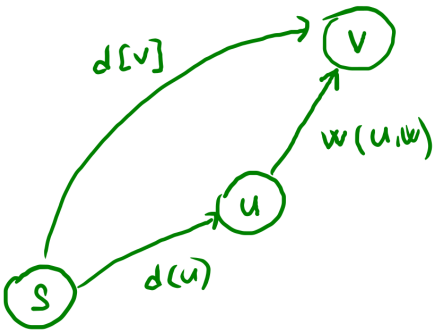
Představené algoritmy pracují z důvodu efektivity se sledy a nikoliv s cestami (bylo by nutné stále kontrolovat, zda nebyla porušena podmínka cesty), ačkoliv je problém nazývá hledání nejkratší cesty.

```
1 def initialize_single_source(G, s):
2     # G je graf
3     # s je vychozi uzel
4     for v in G.V:
5         d[v] = INF # d je pole vzdalenosti
6         pi[v] = NULL # pi je pole predchudcu
7     d[s] = 0
```

Výpis 2.1: Pomocná inicializační funkce. Složitost je $\Theta(n)$, kde n je počet uzlů.

```
1 def relax(u, v, w):
2     # u a v jsou uzly grafu
3     # w je vahova funkce
4     if d[v] > d[u] + w(u, v):
5         d[v] = d[u] + w(u, v)
6         pi[v] = u
```

Výpis 2.2: Pomocná funkce *relax*. Složitost je $O(1)$.



Obrázek 2.1: Ukázka činnosti funkce *relax*.

2.4 Bellman-Fordův algoritmus

Slouží pro řešení v obecných grafech, mohou obsahovat cykly a záporné hrany. Záporné cykly je však nutné detekovat a vrátit specifickou hodnotu. V podstatě se jedná o *brute force* algoritmus, provede se relaxace $n-1$-krát pro každou hranu.

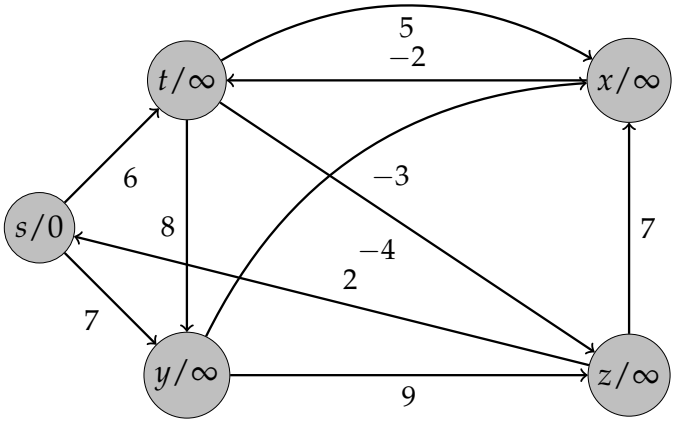
```
1  def bellman_ford(G, s, w):
2      # G je graf
3      # s je vychozi uzel
4      # w je vahova funkce
5
6      # faze inicializace
7      initialize_single_source(G, s)
8      n = len(G.V) # pocet uzlu
9
10     # faze relaxace: provedeni (n-1) * m relaxaci (m je pocet hran)
11     for _ in range(0, n-1):
12         for u, v in G.E:
13             relax(u, v, w)
14
15     # faze detekce zaporneho cyklu
16     for u, v in G.E:
17         if d[u] > d[v] + w(u, v):
18             return NULL
19
20     return pi
```

Výpis 2.3: Algoritmus Bellman-Ford. Proč $n-1$ iterací? Protože mezi libovolnými dvěma uzly v grafu, existuje cesta o maximálním počtu hran $n-1$.

2.4.1 Složitost

- Řádek 7, 8 – $\Theta(1)$.
- Řádky 11, 12, 13 – $(n-1) \cdot \Theta(m) = \Theta(n \cdot m)$, kde n je počet uzlů a m je počet hran grafu.
- Řádek 16, 17, 18 – $\Theta(m)$.
- Celkem $\Theta(n \cdot m)$.

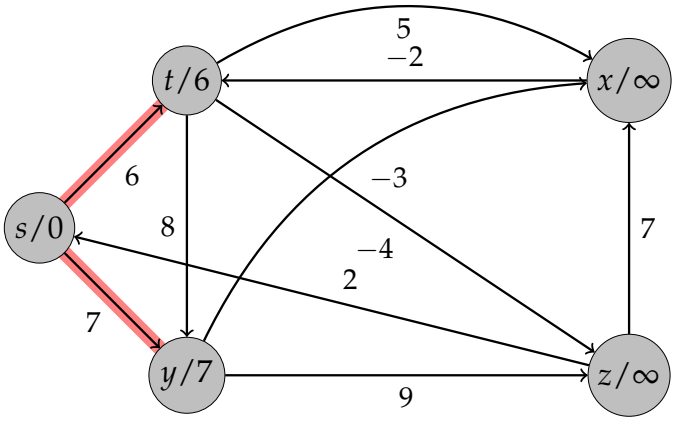
2.4.2 Příklad



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

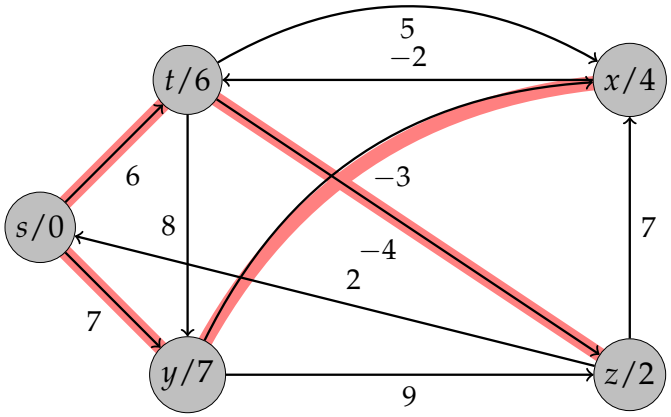
Obrázek 2.2: Příklad, část 1.



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

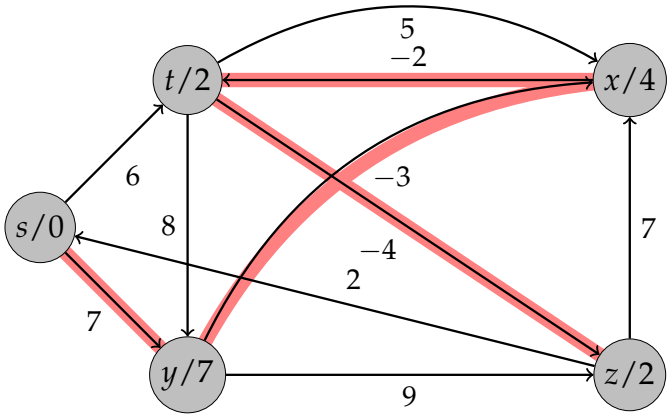
Obrázek 2.3: Příklad, část 2.



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

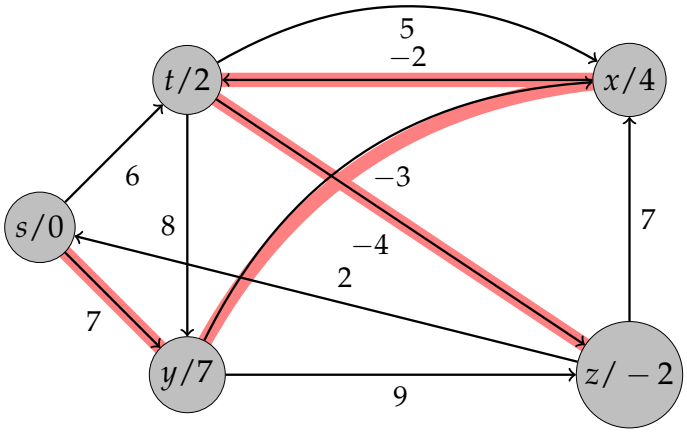
Obrázek 2.4: Příklad, část 3.



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

Obrázek 2.5: Příklad, část 4.



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

Obrázek 2.6: Příklad, část 5.

2.5 Dijkstrův algoritmus

Slouží pro řešení v acyklických grafech bez záporných hran. Pro takto omezený problém existují rychlejší algoritmy než pro problém v obecných grafech.

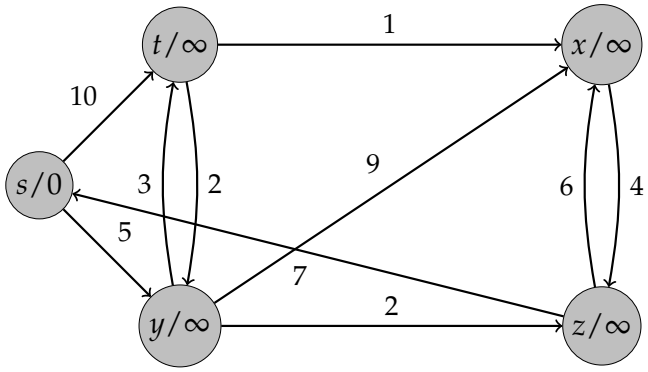
```
1  def dijkstra(G, s, w):
2      # G je graf
3      # s je vychozi uzel
4      # w je vahova funkce
5
6      # faze inicializace
7      initialize_single_source(G, s)
8      Q = Queue(G.V) # prioritni fronta uzlu
9      S = {}# mnozina uzlu, ktera uz byla prozkoumana
10
11     # faze relaxace
12     while not Q.empty():
13         u = Q.extract_min(d) # vrati prvek z Q s nejmensi hodnotou v d
14         S += {u}
15         # pro vsechny sousedy uzlu u (Adj je seznam sousedu)
16         for v in Adj[u]:
17             relax(u, v, w)
18
19         Q.decrease_key(d) # aktualizace prioritni fronty
20
21     return d, pi
```

Výpis 2.4: Algoritmus Dijkstra.

2.5.1 Složitost

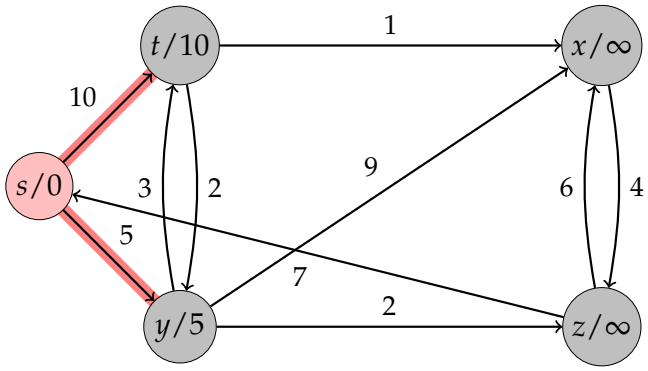
- Předpokládejme implementaci prioritní fronty pomocí pole.
- Řádek 8, 18 – $O(1)$.
- Řádek 11 – While cyklus se provede n-krát, kde n je počet uzlů.
- Řádek 12 – $O(n)$, najítí minima v poli uzlů. Celkově (s cyklem) $O(n^2)$.
- Řádek 16 – $O(m)$, pro všechny hrany. Celkově (s cyklem) $O(m \cdot n)$.
- Celkem $O(n^2 + m) = O(n^2)$.
- Pro řídké grafy lze využít implementaci fronty pomocí binární haldy a získat tak $O(m \cdot \log(n))$.
- Při implementaci fronty pomocí Fibonacciho haldy dostaneme časovou složitost $O(n \cdot \log(n) + m)$.

2.5.2 Příklad



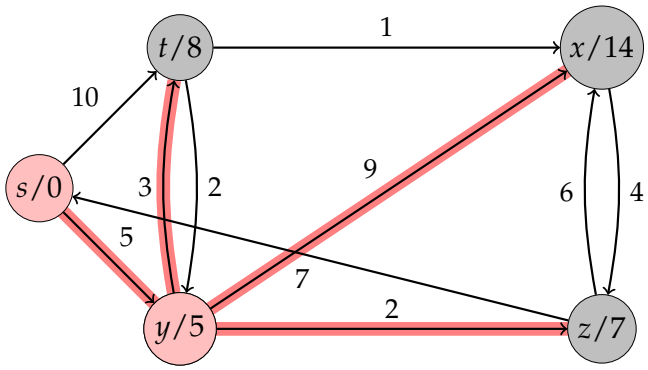
Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Obrázek 2.7: Příklad, část 1.



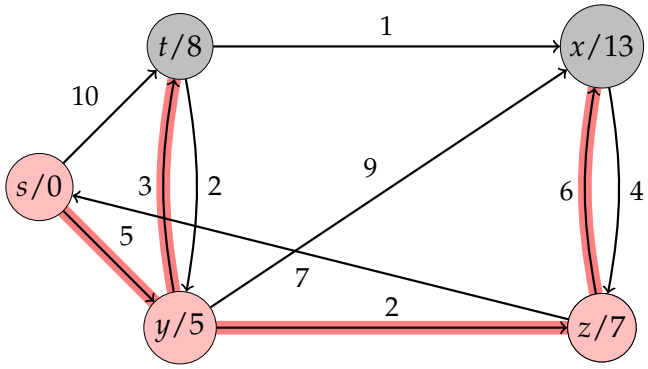
Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Obrázek 2.8: Příklad, část 2.



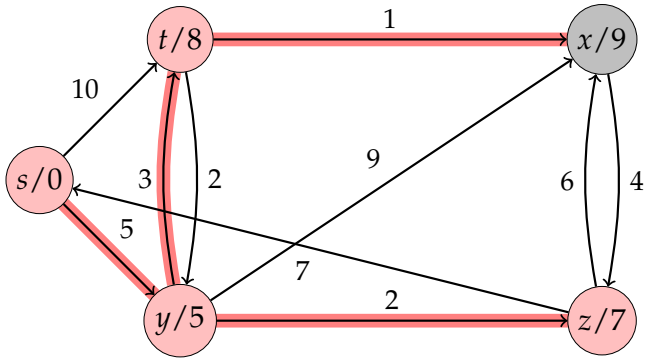
Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Obrázek 2.9: Příklad, část 3.



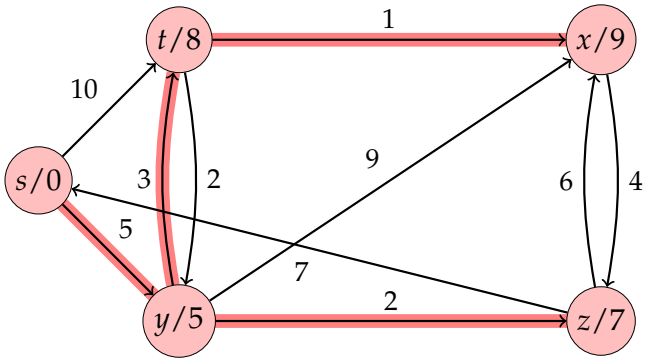
Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Obrázek 2.10: Příklad, část 4.



Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Obrázek 2.11: Příklad, část 5.



Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Obrázek 2.12: Příklad, část 6.

Kapitola 3

Klasifikace algoritmů volby koordinátora, algoritmus Bully a jeho složitost.

3.0.1 Metadata

- Předmět: Prostředí distribuovaných aplikací (PDI)
- Přednáška: 7 – Synchronizace
- Záznam: 2020-11-02

3.0.2 Úvod, kontext

todo

Kapitola 4

Podmínky konsistentního globálního stavu distribuovaného systému.

4.0.1 Metadata

- Předmět: Prostředí distribuovaných aplikací (PDI)
- Přednáška: 4 – Globální stav a snapshots
- Záznam: 2020-10-12

4.0.2 Úvod, kontext

todo