Vypracované otázky k MSZ pro rok 2022

Specializace NNET

19. 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).

Acyklický graf Acyklický graf je graf bez cyklů.

Strom Graf, který je souvislý a neobsahuje žádnou kružnici, 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.

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 $V - S$.

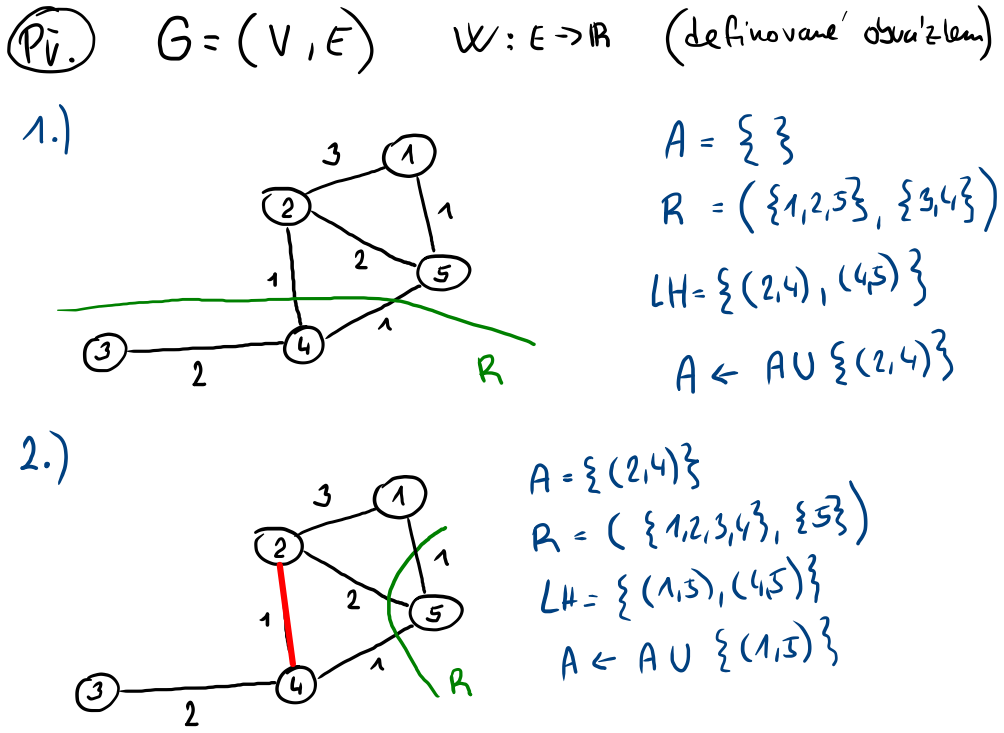
Respektování Nechť $A \subseteq E$ je množina hran. Řez $(S, V - S)$ respektuje množinu hran A, pokud žádná hrana v A nekříží řez $(S, V - S)$.

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.

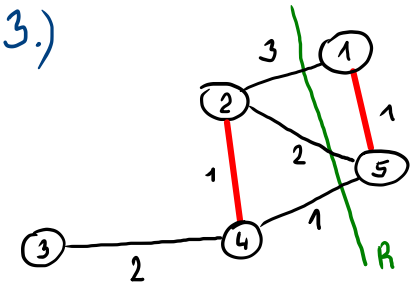
```
1  def generic_mst(G):
2      # A je mnozina hran rozpracovane minimalni kostry
3      A = {}
4      while netvori_kostru(A, G):
5          for edge in G.E:
6              if je_bezpecna(A, edge):
7                  A += {edge}
8      return A
```

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.



Obrázek 1.1: Příklad, část 1.

3.)



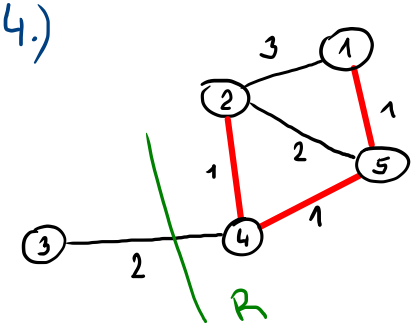
$A = \{(1,5),(2,4)\}$

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

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

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

4.)



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

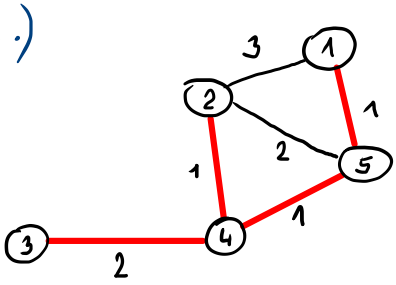
$R = (\{1,2,4,5\},\{3\})$

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

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

Obrázek 1.2: Příklad, část 2.

5.)



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

A je minimální kostra

(minimálních už není možné udělat žádný, který by respektoval A)

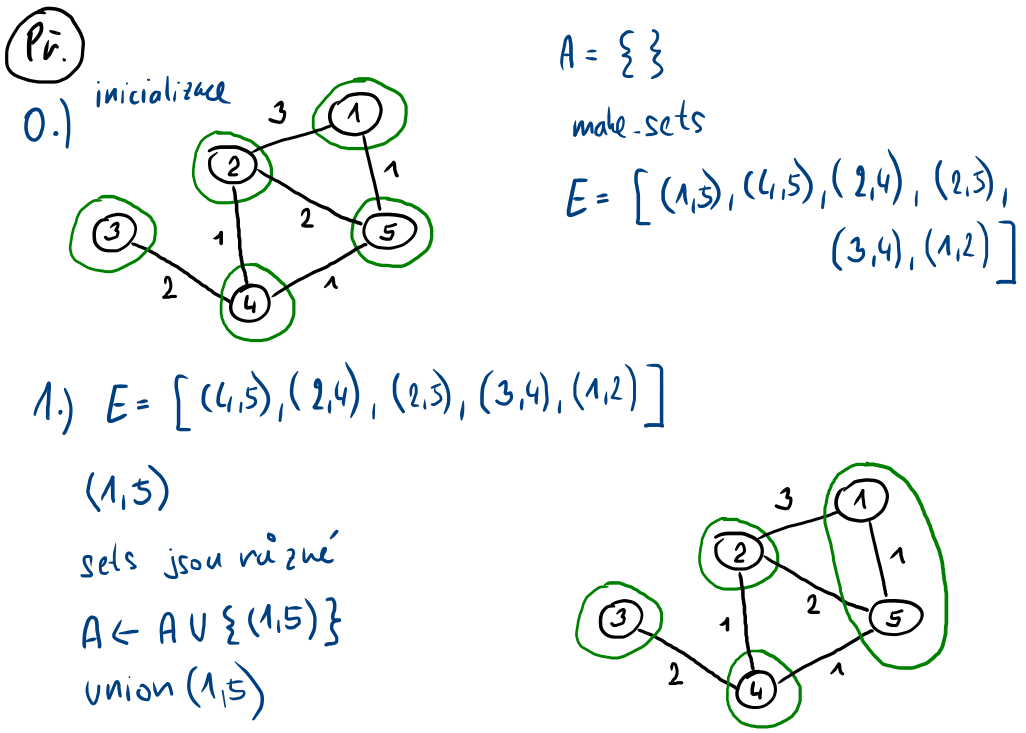
Obrázek 1.3: Příklad, část 3.

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



Obrázek 1.4: Příklad, část 1.

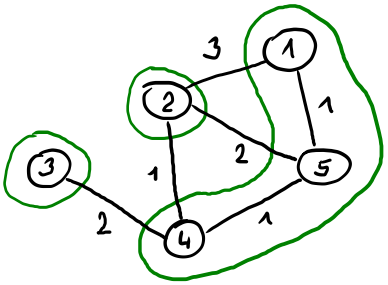
2.) $E = [(2,4), (2,5), (3,4), (1,2)]$

(4,5)

sets jsou různé

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

union (4,5)



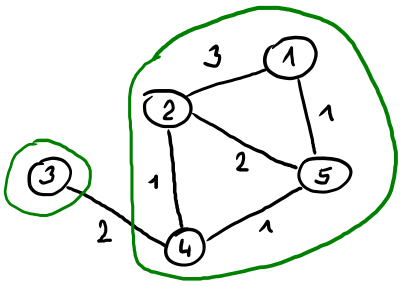
3.) $E = [(2,5), (3,4), (1,2)]$

(2,4)

sets jsou různé

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

union (2,4)

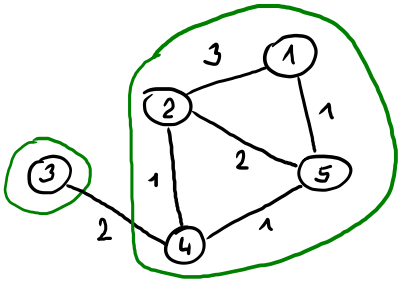


Obrázek 1.5: Příklad, část 2.

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

(2,5)

sets nejsou různé



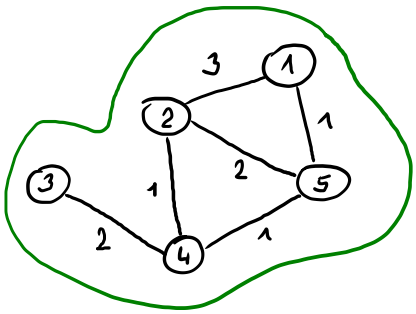
4.) $E = [(1,2)]$

(3,4)

sets jsou různé

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

union (3,4)



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

Obrázek 1.6: Příklad, část 3.

1.5 Primův-Jarníkův algoritmus

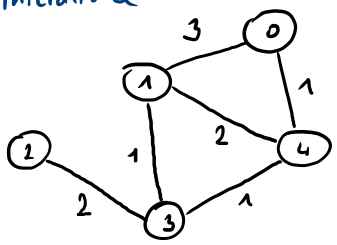
Primův algoritmus buduje tzv. *A* strom. Má zadaný určitý uzel, ze kterého hledá nejbližší další uzel, který by připojil. A pak další a další.

```
def prim_mst(G, r):
    # r je vychozi uzel

    for u in G.V:
        # v poli key je ulozeno, kolik stoji prechod do vrcholu na indexu
        key[u] = INF
        # pole predchudcu, reprezentace mnoziny A
        pi[u] = NULL

    key[r] = 0
    # Q je prioritni fronta, prvek s mensi hodnotou v poli key ma prednost
    Q = Queue(G.V)

    for u in Q.priority_deque():
        # pro vsechny jeho sousedy (Adj je seznam sousedu)
        for v in Adj[u]:
            # pokud je levnejsi cesta a jeste to neni prozkoumany uzel
            if v in Q and w(u, v) < key(v):
                pi[v] = u
                key[v] = w(u, v)

    return pi
```

Výpis 1.3: Primův algoritmus.

Př.

0.) inicializace



$r = 1$

$key = [\infty, 0, \infty, \infty, \infty]$

$\pi = [NULL, NULL, NULL, NULL, NULL]$

$Q = [0, 1, 2, 3, 4]$

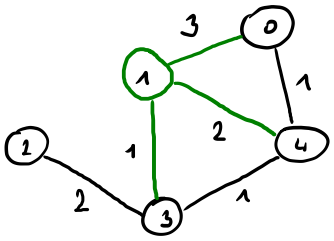
1.) $u = 1$

$Q = [0, 2, 3, 4]$

$v \in \{0, 3, 4\}$

$key = [3, 0, \infty, 1, 2]$

$\pi = [1, NULL, NULL, 1, 1]$



Obrázek 1.7: Příklad, část 1.

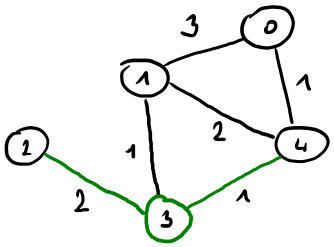
2.) $u = 3$

$Q = [0, 2, 4]$

$v \in \{\cancel{1}, 2, 4\}$

$key = [3, 0, 2, 1, 1]$

$\pi = [1, NULL, 3, 1, 3]$



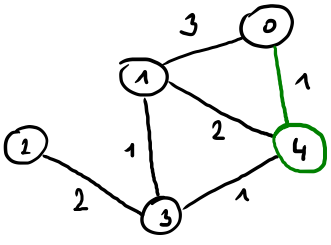
3.) $u = 4$

$Q = [0, 2]$

$v \in \{0, \cancel{1}, \cancel{3}\}$

$key = [1, 0, 2, 1, 1]$

$\pi = [4, NULL, 3, 1, 3]$



Obrázek 1.8: Příklad, část 2.

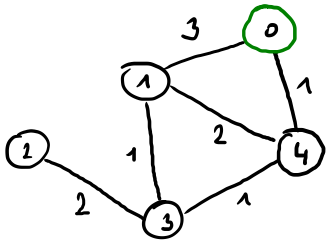
4.) $u = 0$

$Q = [2]$

$v \in \{1, 4\}$

$key = [1, 0, 2, 1, 1]$

$\pi = [4, NULL, 3, 1, 3]$



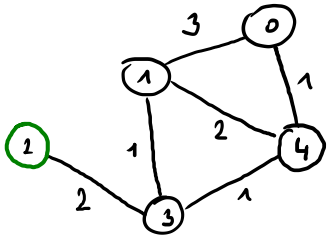
5.) $u = 2$

$Q = [\,]$

$v \in \{3\}$

$key = [1, 0, 2, 1, 1]$

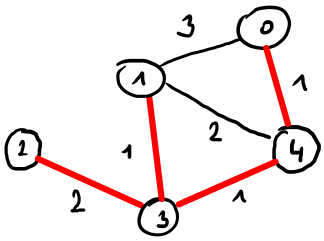
$\pi = [4, NULL, 3, 1, 3]$



Obrázek 1.9: Příklad, část 3.

6.) $key = [1, 0, 2, 1, 1]$

$\pi = [4, NULL, 3, 1, 3]$



Obrázek 1.10: Příklad, část 4.

Kapitola 2

Klasifikace algoritmů volby koordinátora, algoritmus Bully a jeho složitost.

2.0.1 Metadata

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

2.0.2 Úvod, kontext

todo

Kapitola 3

Podmínky konsistentního globálního stavu distribuovaného systému.

3.0.1 Metadata

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

3.0.2 Úvod, kontext

todo