6. Tahy / Kostry / Nejkratší cesty …

BI-EP2
Efektivní programování 2

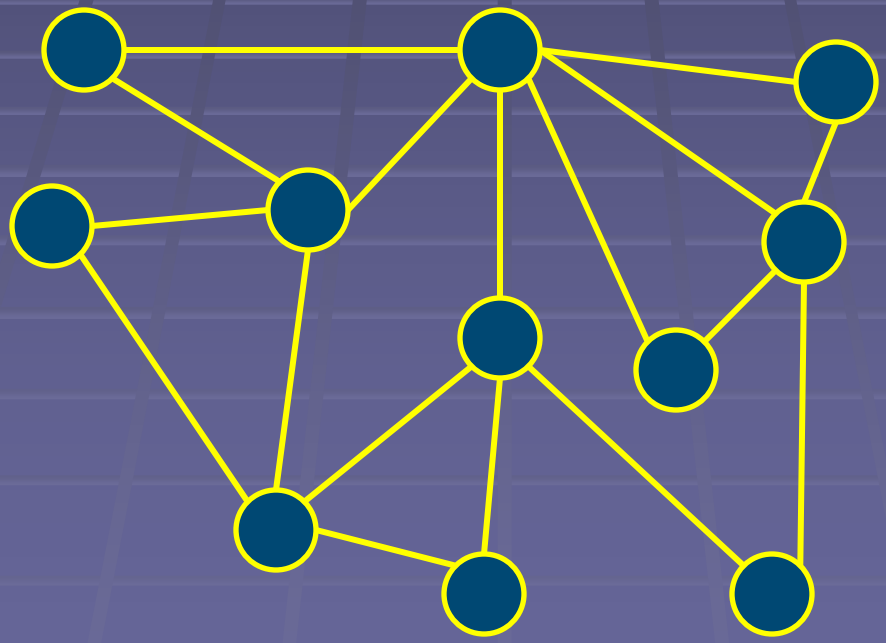
LS 2018/2019

Ing. Martin Kačer, Ph.D.

Katedra teoretické informatiky
Fakulta informačních technologií
České vysoké učení technické v Praze

Příklad: listonoš

- Listonoš chce rozvézt dopisy
- Jak každou ulicí projet právě jednou?



Pravidla pro pokrytí

- Kdy lze pokrýt graf jedním tahem?
  - musí být souvislý
  - musí být Eulerův nebo právě 2 liché stupně
- Kolik tahů obecně na souvislý graf?
  - Počet uzlů lichého stupně děleno dvěma
- A co když není souvislý?
  - Řešíme každou komponentu zvlášť

Jak to naprogramovat?

1. Rozdělit na komponenty souvislosti

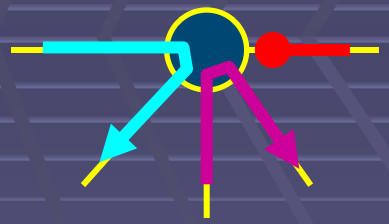
2. V každé spočítat uzly s lichým stupněm
   - 0 => jeden tah (a bude uzavřený)
   - N => N/2 tahů (otevřených)

- Jak tahy přesně stanovit?

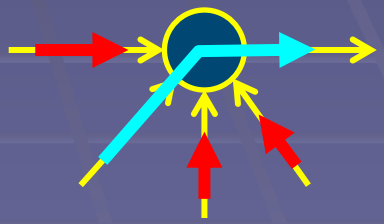
  (k diskuzi)

Orientované grafy

- Princip je stejný
- Místo „sudý stupeň“
  je „stejný vstupní a výstupní stupeň“



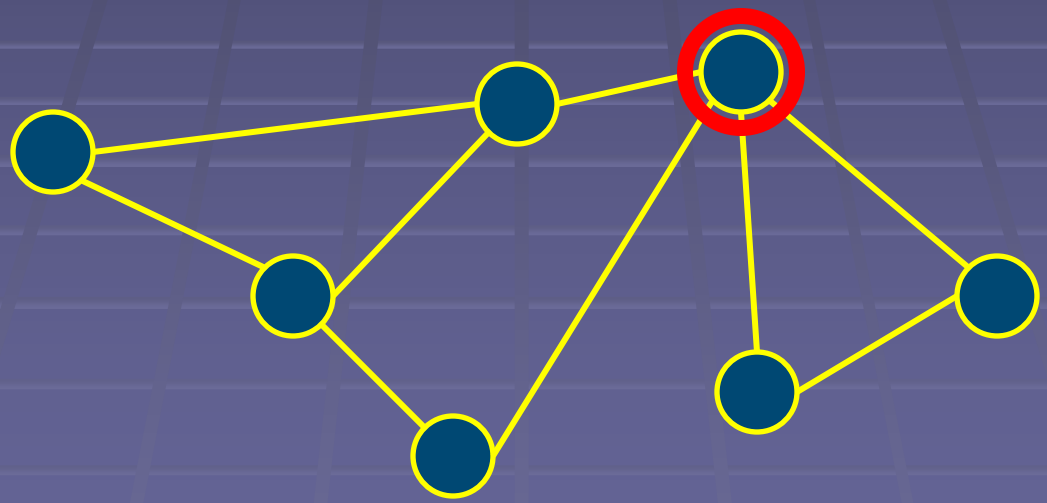
- Pozor, když se liší o víc než 1
  - Nutno započítat celý rozdíl



Konektivita grafů

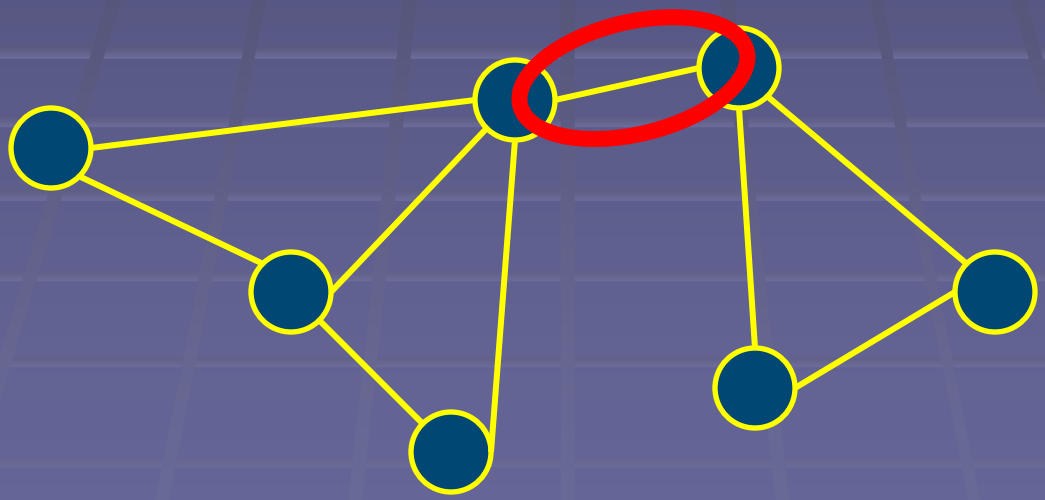
Artikulace

- Artikulace = uzel, jehož odebráním se zvýší počet souvislých komponent



Most

- Most = hrana, jejímž odebráním se zvýší počet souvislých komponent
  (= hrana, která není v žádném cyklu)



Další pojmy

- 2-souvislý (*biconnected*) graf
- 2-souvislá komponenta
  - bez artikulací

- Praktická stránka: vliv na spolehlivost
  - Výpadky uzlů
  - Výpadky spojů mezi uzly

Jak najít artikulace/mosty?

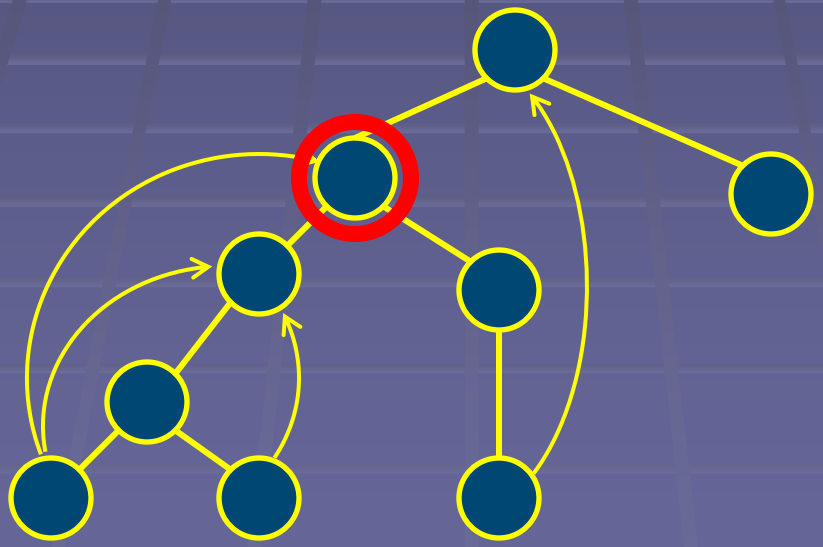
- Relativně jednoduchý algoritmus

- Zkusím odebírat uzly (příp. hrany)
- Testuji, zda graf je stále souvislý
  - (případně vzrostl počet komponent)

- Operační složitost?
  - **O(u.(u+h))** (artik.) **O(h.(u+h))** (mosty)

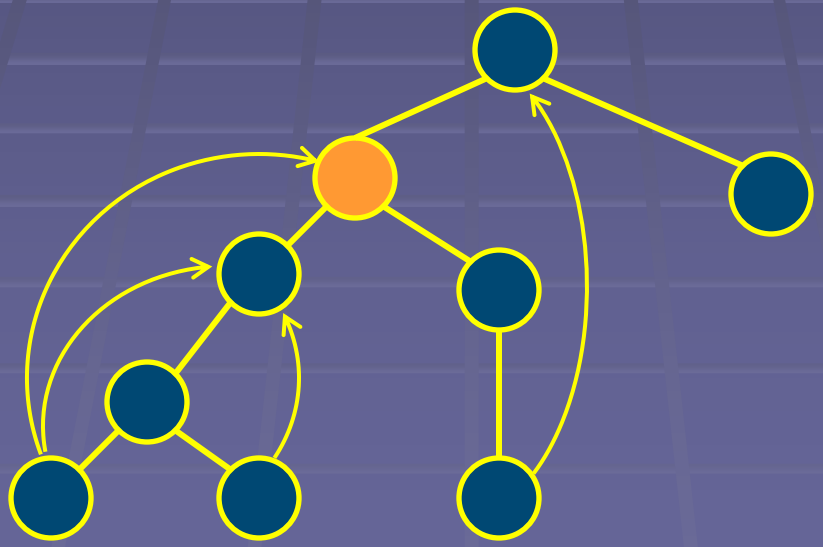
Hledání artikulací – efektivněji

- Jeden průchod hledání do hloubky
  - Vznikne strom + zpětné hrany
- Jak vypadá artikulace?



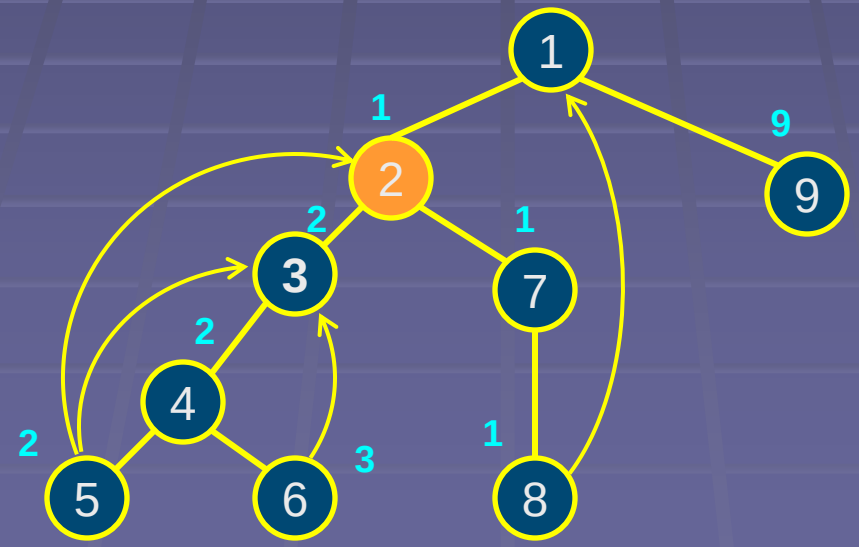
Hledání artikulací

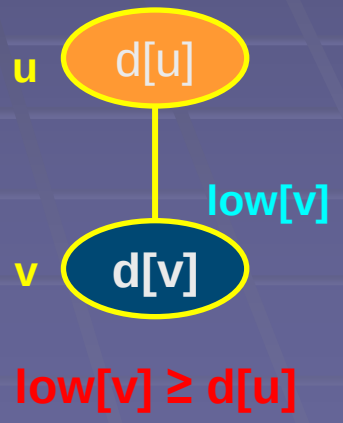
- Artikulace = vnitřní uzel (tj. ne list) stromu, z jehož potomka se nelze dostat do předků
  - Stačí jeden takový potomek
  - Může vést i do daného uzlu (artikulace)



Hledání artikulací – jak na to?

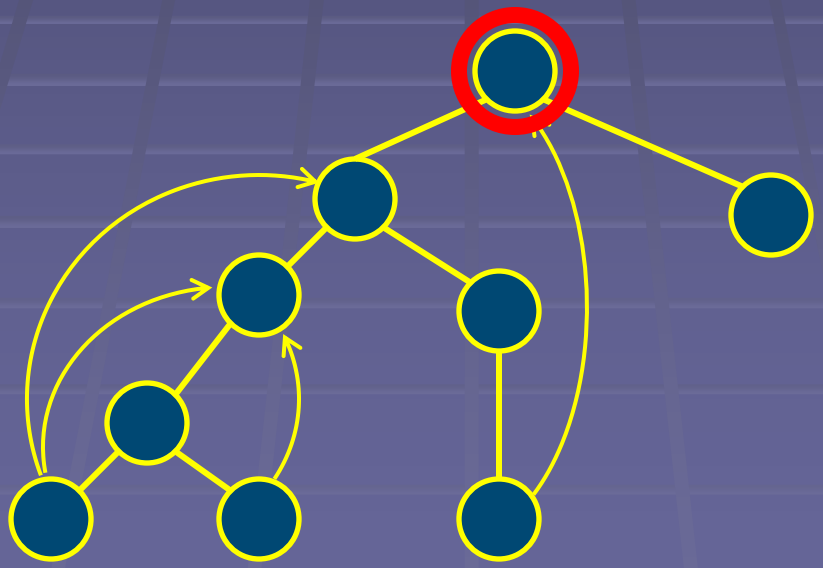
- Časové značky => pořadí uzlu – **d[u]**
- Rekurze vrací, do jakého nejnižšího uzlu se lze z kterého podstromu vrátit – **low[u]**





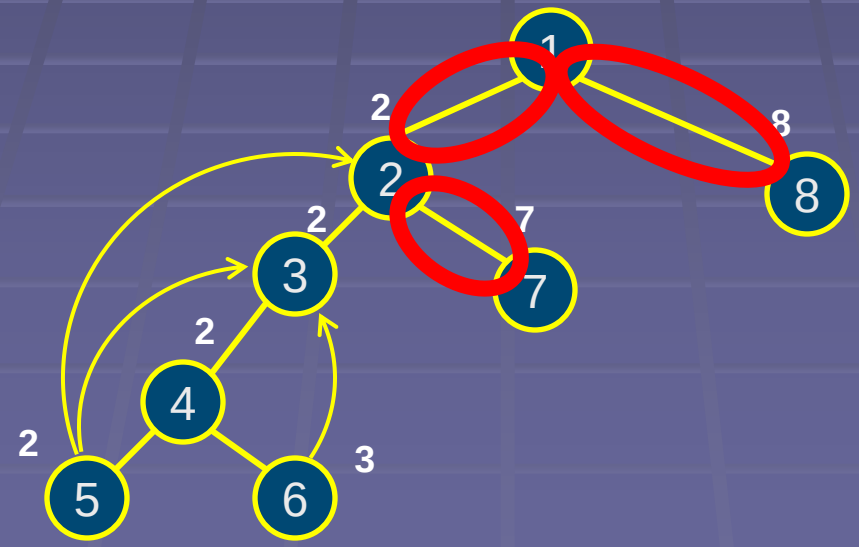
Artikulace – speciální případ

- Artikulace = kořen, pokud má víc dětí



Hledání mostů

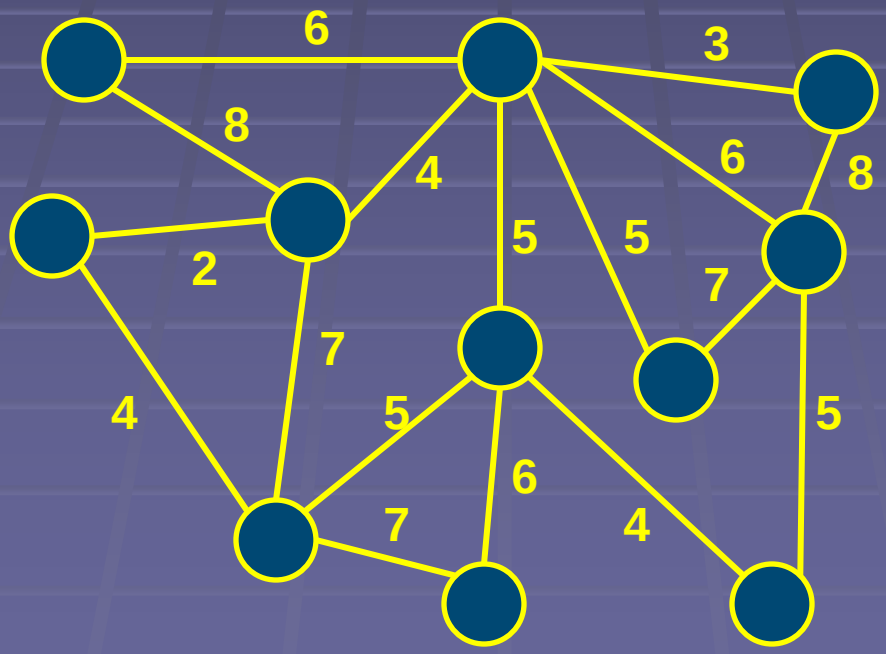
- Velmi podobně
- Most = stromová hrana, z jejíhož podstromu se nelze dostat k vyšším uzlům



Kostry grafu

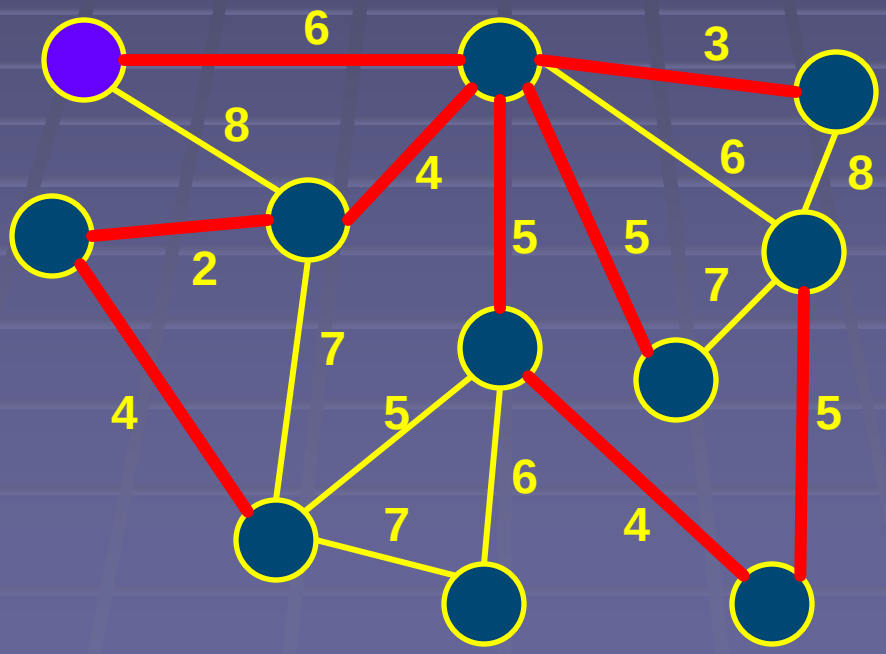
Jarník-Prim: princip algoritmu

- Postupně připojujeme uzel, který je nejblíže k již připojeným



Jarník-Prim: princip algoritmu

- Postupně připojujeme uzel, který je nejblíže k již připojeným



Jarník-Prim: základ kódu

```
for (Node n : allNodes()) {
  n.open = true;
  n.dist = +∞;
}
start.dist = 0;

while (anyOpenNodeLeft()) {
  Node n = findOpenWithMinDist();
  n.open = false;
  selectEdge(n.prev, n);
  for (Node x : neighbors(n)) {
     if (x.open && x.dist > edgeLen(n, x)) {
        x.dist = edgeLen(n, x);
        x.prev = n;
     }
} }
```

inicializace

připoj nejbližšího

uprav vzdálenosti jeho sousedů

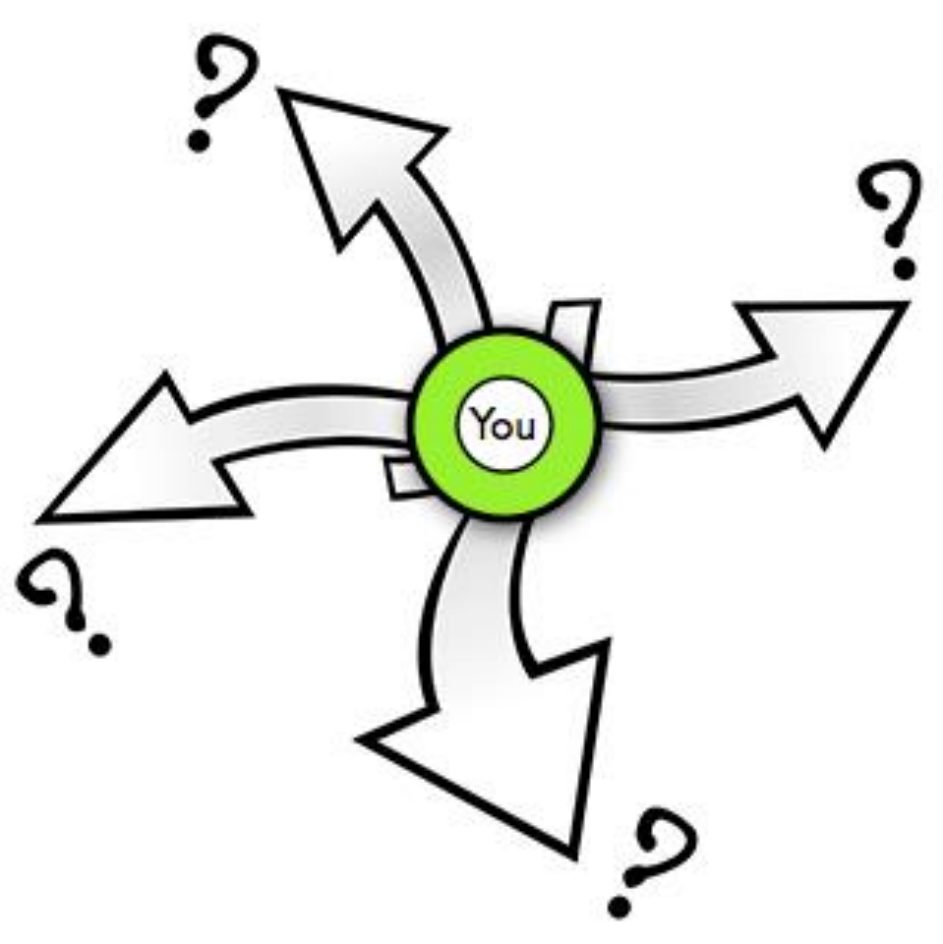
Jarník-Prim: složitost

- Jaká je složitost?
- Záleží na implementaci prioritní fronty!!
  - Obyčejné pole => **$O(u^2)$**
  - Binární halda => **$O(h \cdot \log u)$**
    - (což může být i víc než u pole!)
- A také na reprezentaci grafu (jako vždy)

Jarník-Prim: jak naprogramovat

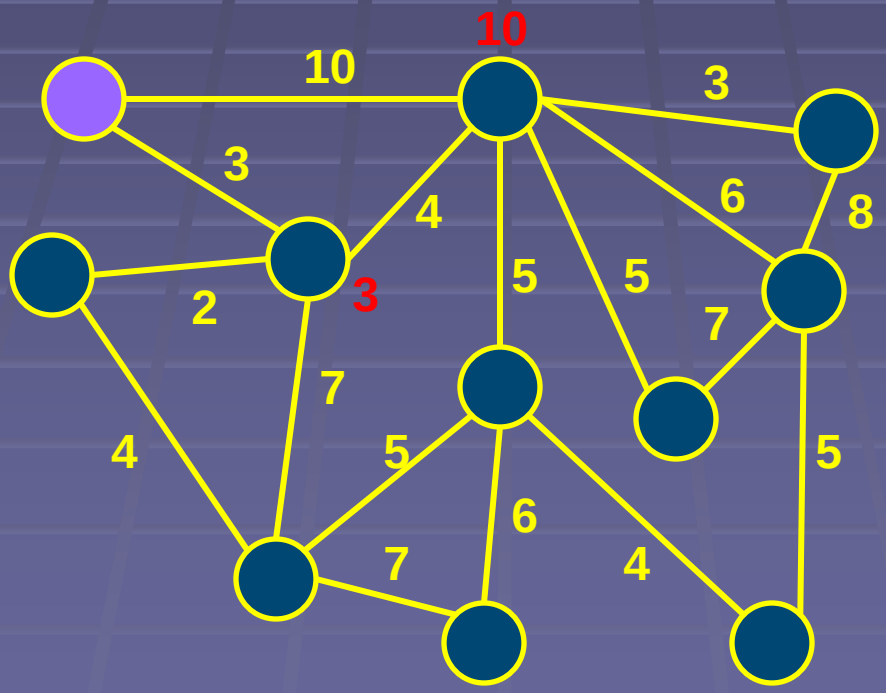
```
for (Node n : allNodes()) {
  n.open = true;
  n.dist = +∞;
}
start.dist = 0;

while (anyOpenNodeLeft()) {
  Node n = findOpenWithMinDist();
  n.open = false;
  selectEdge(n.prev, n);
  for (Node x : neighbors(n)) {
      if (x.open && x.dist > edgeLen(n, x)) {
          x.dist = edgeLen(n, x);
          x.prev = n;
      }
} }
```

Nejkratší cesty



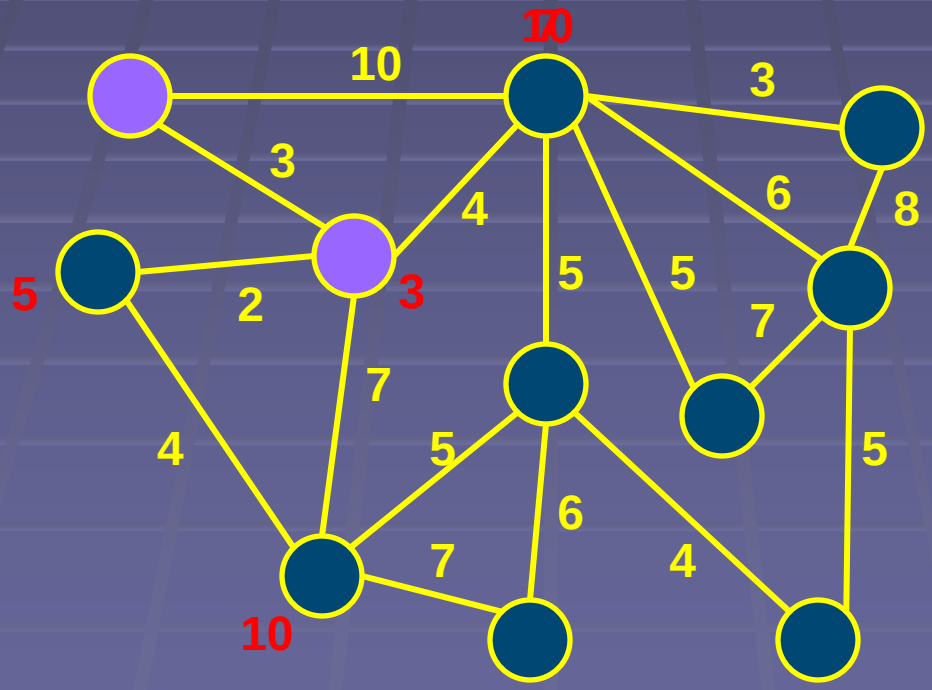
Dijkstra: princip algoritmu

- Postupně zavíráme uzly, které mají nejmenší vzdálenost



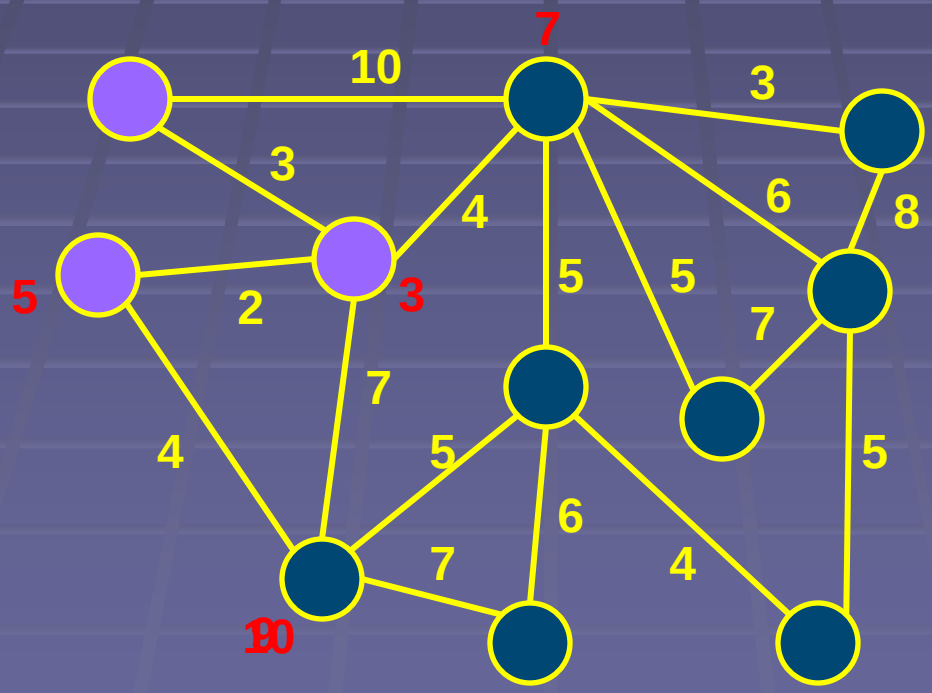
Dijkstra: princip algoritmu

- Postupně zavíráme uzly, které mají nejmenší vzdálenost



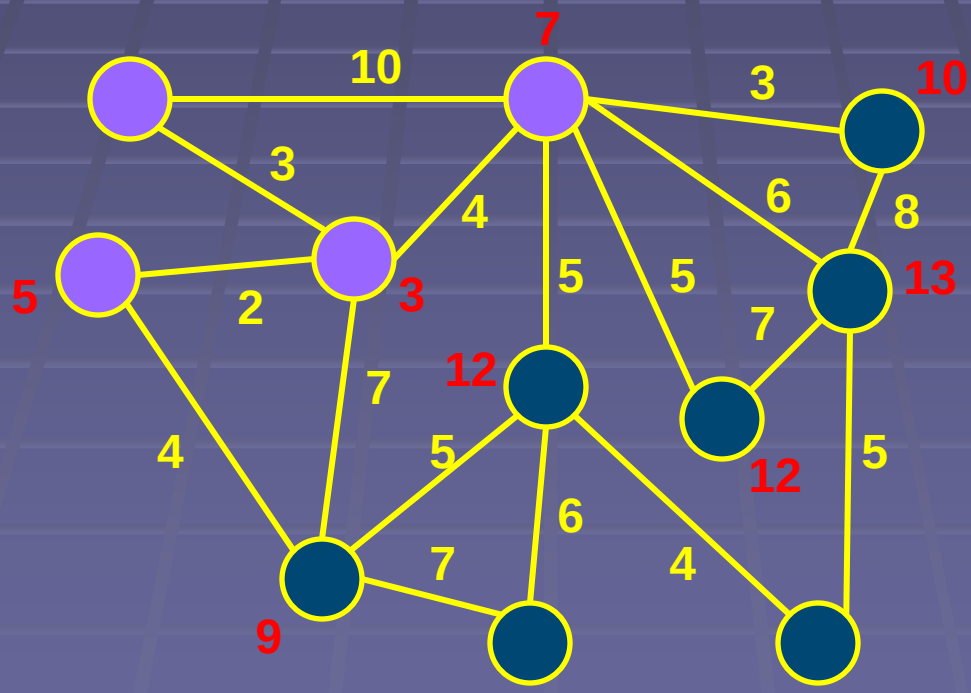
Dijkstra: princip algoritmu

- Postupně zavíráme uzly, které mají nejmenší vzdálenost



Dijkstra: princip algoritmu

- Postupně zavíráme uzly, které mají nejmenší vzdálenost



Dijkstra: základ kódu

```
for (Node n : allNodes()) {
  n.open = true;
  n.dist = +∞;
}
start.dist = 0;

while (anyOpenNodeLeft()) {
  Node n = findOpenWithMinDist();
  n.open = false;
  for (Node x : neighbors(n)) {
     if (x.dist > n.dist + len(n, x)) {
        x.dist = n.dist + len(n, x);
        x.prev = n;
     }
} }
```

inicializace

připoj nejlepšího

uprav vzdálenosti jeho sousedů

Dijkstra x Jarník-Prim

- V čem je rozdíl oproti hledání kostry?
  - Přestože řeší jinou úlohu, jsou si algoritmy strukturou podobné

```
while (anyOpenNodeLeft()) {
  Node n = findOpenWithMinDist();
  n.open = false;
  for (Node x : neighbors(n)) {
     int d = n.dist + len(n, x);
     if (x.dist > d) {
          x.dist = d; x.prev = n;
     }
} }
```

Dijkstra: složitost

- Jaká je složitost?
- Záleží na implementaci prioritní fronty!!
  - Obyčejné pole => **$O(u^2)$**
  - Halda => **$O(h \cdot \log u)$**
    - (což může být i víc než u pole!)
- A také na reprezentaci grafu (jako vždy)

Příklad: Velikost potrubí

- Graf modelující potrubí
- Ohodnocení hran = průměr trubky

- Jaký největší předmět lze poslat?

- => Jiné operace při kombinování cest

Příklad: Velikost potrubí

- Jaký největší předmět lze poslat?
  - + → min min → max

```
while (anyOpenNodeLeft()) {
  Node n = findOpenWithMinDist();
  n.open = false;
  for (Node x : neighbors(n)) {
     int d = min(n.dist + len(n, x));
     if (x.dist < d) {
          x.dist = d; x.prev = n;
} } }
```

Další operace pro cesty

- Nejspolehlivější komunikační linka
  - + → × min → max

```
while (anyOpenNodeLeft()) {
  Node n = findOpenWithMinDist();
  n.open = false;
  for (Node x : neighbors(n)) {
     int d =     n.dist + len(n, x);
     if (x.dist < d) {
          x.dist = d; x.prev = n;
} } }
```

Další operace pro cesty

- Nejvýhodnější směnný kurz
  - + → × min → min

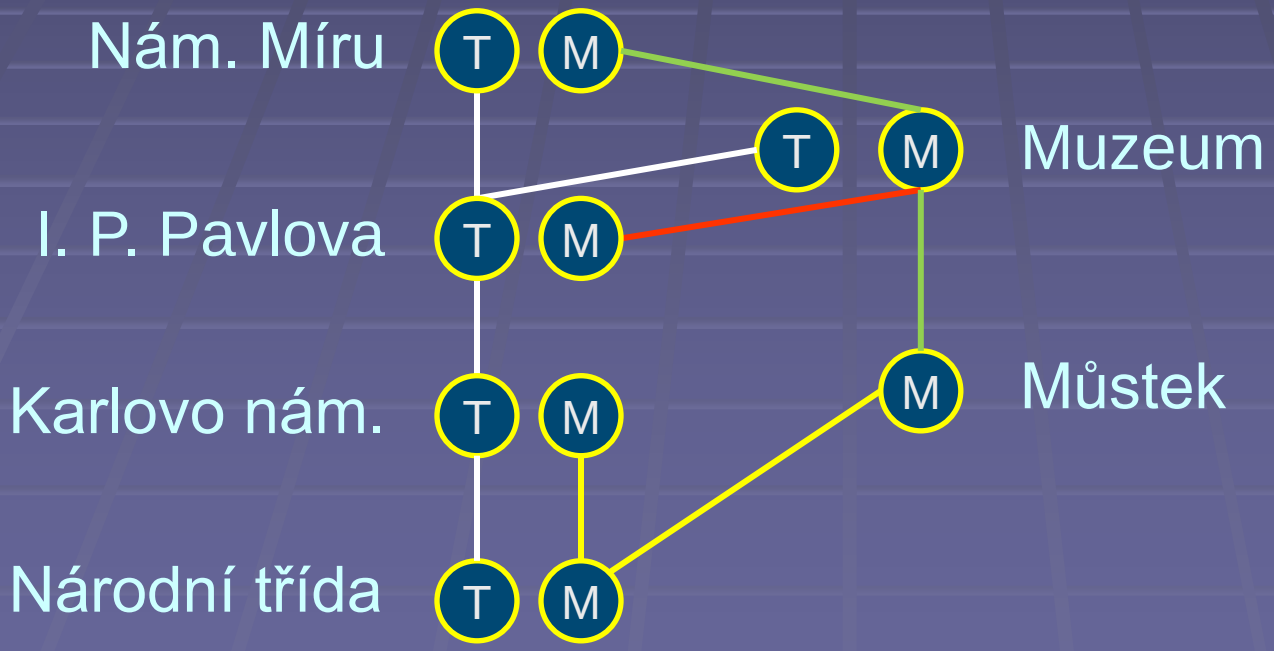
```
while (anyOpenNodeLeft()) {
  Node n = findOpenWithMinDist();
  n.open = false;
  for (Node x : neighbors(n)) {
     int d =     n.dist + len(n, x);
     if (x.dist > d) {
          x.dist = d; x.prev = n;
} } }
```

Příklad: Cestování MHD

- Doba jízdy mezi zastávkami
  - Pro jednoduchost to není jízdní řád
- Přestup:
  - Tramvaj-tramvaj: 1 min
  - Tramvaj-metro: 5 min
  - Metro-metro: 3 min

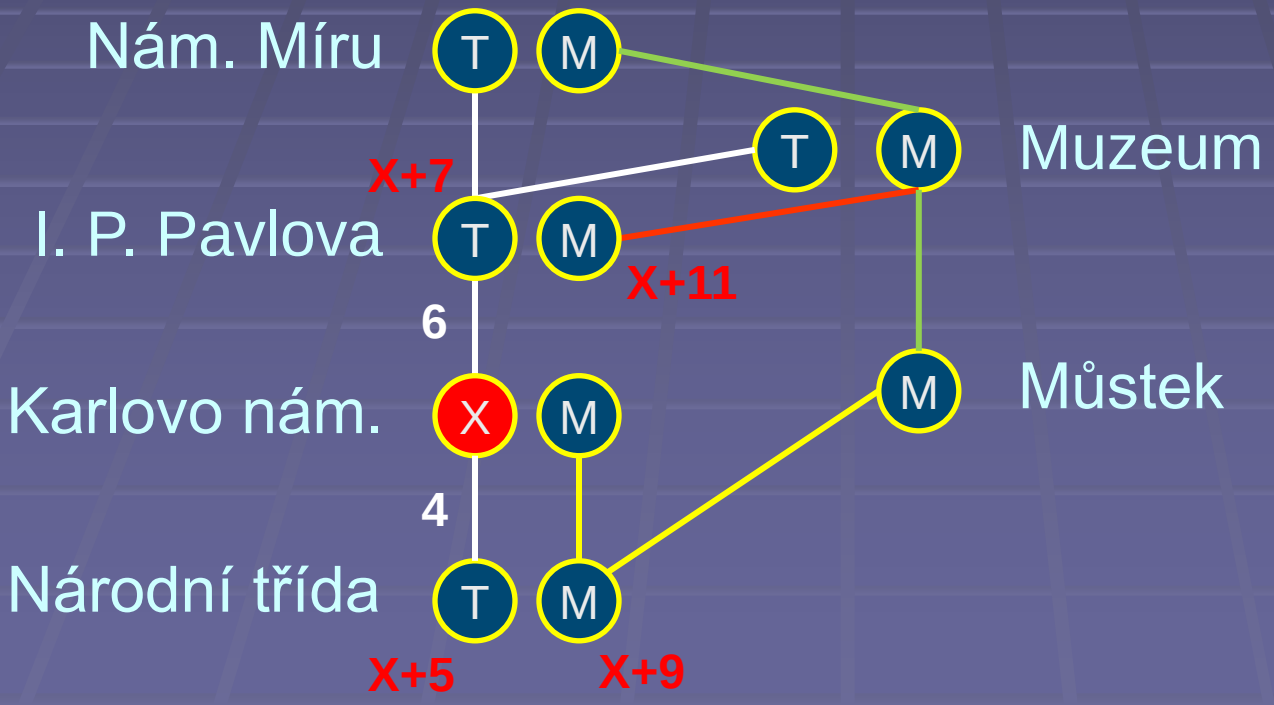
Příklad: Cestování MHD

- Nutné „rozdvojit“ uzly
  - Metro x Tramvaj



Příklad: Cestování MHD

- Pozor na zahrnování přestupů!
  - Nutno vzít v úvahu při „příjezdu“



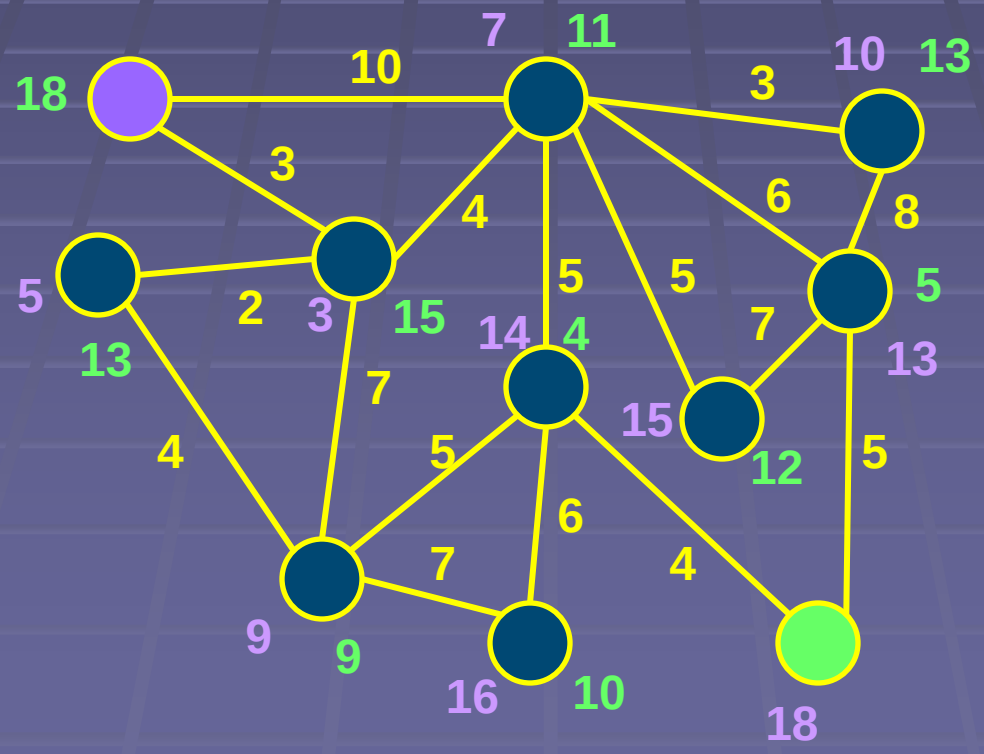
Příklad: Rande

- Dvě osoby skončí v nějakou dobu v práci
  - Pracovní doba nemusí být pro oba stejná
- Jak se mohou co nejrychleji potkat?
- Nezáleží na místě

- (úloha Catch the Bus, CTU Open 2007)

Příklad: Rande

- Pustíme Dijkstrův algoritmus pro oba
- Hledáme nejmenší maximum z obou časů



Tvorba úloh

Vaše „autorská úloha“

- Podmínka získání zápočtu !!!

1. Zadání
2. Vzorové řešení
3. Testovací data
4. Časový limit

Autorská úloha – zadání

- Čeština / angličtina (příp. slovenština)
  - Přesné
  - Jednoznačné
  - Stručné
  - Formát vstupu a výstupu
  - Jednoduché příklady

- Kdy je špatné?
  - Někdo nepochopí (a není to jeho chyba)

Autorská úloha – řešení

- C / C++ / Java
  - Správné
  - Přehledné (demonstruje postup)
  - S vysvětlením algoritmu (česky/anglicky)
    - Na začátku zdrojového kódu
    - Ve zvláštním souboru
- Kdy je špatné?
  - Existuje platné zadání, které nevyřeší

Autorská úloha – testovací data

- Odpovídající zadání (formát i data)
- Postihující maximum případů
  - Nejmenší a největší případy
  - Speciální případy
- Mohou být generována programem
  - Několik ručních „ukázkových“ vstupů

- Kdy jsou špatná?
  - Dovolí přijmout nesprávné řešení

Autorská úloha – časový limit

- Stačí krátké zdůvodnění
  - Jak moc je časový limit důležitý
  - Násobek času vzorového programu
  - Na čem selže neefektivní řešení (je-li takové)
- Ne (!) konkrétní časy

- Kdy je špatně?
  - Když dle toho nejde nastavit konkrétní limit

Autorská úloha – odevzdání

- Termín **1. května 2019** (21:00)
  - Dva a půl týdne před zápočtem
- Hodnocení kolegů od 4. května
  - Bez uvedení autora
  - Snažte se opravdu hodnotit úlohu
  - Objevování chyb (viz dříve)

Autorské úlohy – připomenutí

- Zadání
- Vzorové řešení
- Testovací data
- Stručný rozbor časového limitu

- Termín do **1. května**!

Úlohy

- Grafy
  - Toll (přemýšlejte, co je v této verzi neobvyklé)
  - Play on Words
  - Freckles
  - Tourist Guide