Anotace

- Grafové algoritmy.

Grafově-optimalizační problémy

- Jak reprezentovat grafy – bylo v zimě, nyní už snad dokážete i implementovat.
- Jak grafy prohledávat – také bylo (do šířky a do hloubky), nyní umíte též implementovat.
- Hledání nejkratší cesty, minimální kostra,
- topologické uspořádání, faktorová množina (vyšetřování komponent).
- Modifikace problémů: Maximální kostra, nejdelší cesta – čím se liší?

Nejkratší cesta

- **Dijkstrův algoritmus:** Modifikované prohledávání do šířky (netvoříme frontu nalezených vrcholů, ale strukturu organizujeme podle doby, kdy se do dotyčného vrcholu dostaneme).
- Povšimněte si, že se vlastně jedná o diskrétní simulaci (rozlijeme vodu na graf a čekáme, kdy voda doteče do jednotlivých vrcholů).
- **Bellman-Fordův algoritmus:** $n - 1$-krát zopakujeme: Z každého vrcholu zkus "natáhnout" cestu po všech hranách z něj vycházejících (tedy zlepši případnou nalezenou cestu).
- Algoritmus funguje i při záporném ohodnocení hran, nesmí ale být přítomna záporná kružnice (kružnice záporné délky).

Floyd – Warshallův algoritmus

aneb all-pairs shortest paths

- Další příklad dynamického programování!
- Pro trojici (a, u, v), kde u, v jsou vrcholy a a přirozené číslo počítáme nejkratší cestu z u do v o nejvýše a hranách.
- Postupujeme pro rostoucí a (projdeme všechny možné dvojice) tak, že natahujeme cestu o jedna kratší o "poslední" hranu.
- Jako by vybízelo k rekurzi...
- ... jenže jako obvykle velmi neefektivní.
- Tedy ji vybavíme cachí v podobě třírozměrného pole...
- ... a zjistíme, že rekurzi vůbec nepotřebujeme, že postačí čtyři cykly v sobě...

Floyd – Warshallův algoritmus pseudokód

```
for a:=1 to n-1 do
   for u in vrcholy do
      for v in vrcholy do
         for w in sousedi(v) do
            cache[a,u,v]:=min(cache[a,u,v],cache[a-1,u,w]+
                 length(w,v));
```

Minimální kostra

- Vstup: Graf s ohodnocenými hranami
- Cíl: Kostra s minimální váhou.
- Algoritmy: Kruskal, Borůvka, Jarník, Prim.
- Kruskalův: Setřiď hrany podle váhy, postupně zkoušej přidávat a koukej, zda jsme vytvořili kružnici (pokud ano, hranu nepřidej, jinak přidej).
- Borůvkův: Spojování komponent souvislosti: Vyber hranu s nejmenší vahou, která vychází z dané komponenty a přidej.
- Jarníkův (Primův): Pěstování stromu: K dosud postavenému stromu přidej hranu z něj vycházející, která má nejnižší váhu.
- Všechny algoritmy počítají to samé. Důkazy korektnosti budou příště.

Modifikace

- Hledání nejtěžší kostry...
- ... změň na každé hraně váhu z v_i na $-v_i$.
- Hledání nejdelší cesty...
- ... těžké (NP-těžké).
- Proč? Protože víme, kolik hran má kostra, ale nevíme, kolik hran má cesta.
- Tudíž i nalezení nejkratší cesty v (potenciálně záporně) ohodnoceném grafu je těžké (NP-těžký problém):
- Sedí za ním schovaná Hamiltonskost, tedy hledání cesty délky $n - 1$:

Modifikace II

- Vytvoř úplný graf, za hranu v původním grafu přidej hranu s váhou -1,
  za nehranu přidej hranu s váhou 1.
- Zkus všechny dvojice: Má pro nějaké nejkratší desta váhu $-n+1$?
- Těžkost problému spočívá v tom, že s jeho pomocí jsme schopni vyřešit jiný problém (který máme za těžký).

Topologické uspořádání

- Máme zadán orientovaný graf a ptáme se, zda existuje takové (lineární) uspořádání vrcholů, které je konzistentní se všemi orientovanými hranami.
- Algoritmus je až nečekaně jednoduchý:
- Dokud neodebereme poslední vrchol, opakujeme:
- Najdi vrchol vstupního stupně 1 a přidej na konec dosud utvořeného lineárního uspořádání.
- Pokud takový vrchol neexistuje (a zbývá neprázdný graf), ohlas, že to nejde.

Analýza algoritmu

- Korektnost algoritmu: Podmínka, kterou algoritmus testuje, je očividně postačující. A je také nutná, protože do vrcholu, který zařadíme před všechny ostatní, nesmí vést žádná hrana (má-li být větší, než všechny zbývající).
- Konečnost a složitost algoritmu je též jasná: V každém kroku koukneme na každou hranu nejvýše dvakrát (má dva konce). Složitost je tedy $O(mn)$.

Faktorová množina

Alias večírek s roky narození

- Máme neorientovaný graf určující ekvivalenci, od které neznáme všechny prvky jsoucí v dotyčné relaci a chceme vyšetřit třídy ekvivalence.
- Algoritmus: Každý vrchol má číslo. My mu přiřadíme ještě atribut "číslo komponenty", které bude pro začátek totožné.
- Následně jdeme přes všechny hrany a pro každou vyšetříme, zda mají její koncové vrcholy stejné číslo komponenty. Pokud ne, změníme **všechny** výskyty komponenty s vyšším číslem hodnotou komponenty s nižším číslem.
- Nakonec si spočítáme třídy ekvivalence a postupně vypíšeme (znamenáme si u každého čísla, zda už bylo vypsáno a když najdeme prvek ještě nevypsaný, zaznamenáme si číslo komponenty a vypíšeme všechny prvky z této komponenty).