Josef Doležal

Veta otázky BI-AG1
ZS 2016/17

Acyklický orientovaný graf

zdroj a stok

Acyklický orientovaný graf Uspořádaná dvojice (V, G), kde V je neprázdná množina vrcholů a E množina orientovaných hran taková, že neobsahuje cyklus.

Zdroj Vrchol, do kterého nevede žádná hrana.

Stok Vrchol, ze kterého nevede žádná hrana.

Věta o existenci zdroje v orientovaném grafu

Věta o existenci zdroje v orientovaném grafu

Každý orientovaný graf, který neobsahuje cyklus, má alespoň jeden zdroj.

Věta o existenci zdroje v orientovaném grafu

Věta o existenci zdroje v orientovaném grafu

Každý orientovaný graf, který neobsahuje cyklus, má alespoň jeden zdroj.

Algoritmus topologického uspořádání orientovaného grafu

Algoritmus topologického uspořádání orientovaného grafu

1. Zařaď do fronty všechny vrcholy se vstupním stupněm 0.
2. (dokud není fronta prázdná) Vyber vrchol z počátku fronty
    - - Vypiš vybraný vrchol
    - - Pro každého následníka zkontrontroluj, jestli po odstranění hrany má vstupní stupeň 0.
    - - Pokud má stupeň 0, přidej ho do fronty.

Topologické uspořádání orientovaného grafu

Topologické uspořádání orientovaného acyklického grafu $G = (V, E)$ je takové pořadí vrcholů $v_1, v_2, \ldots, v_n$ grafu G, že pro každou hranu $(v_i, v_j) \in E$ platí $i < j$.

Neorientovaný graf

Neorientovaný graf je uspořádaná dvojice (V, E), kde

1. V je množina vrcholů
2. E je množina hran

Hrana je dvouprvková podmnožina V.

Orientovaný graf

Orientovaný graf G je uspořádaná dvojice (V, E), kde

- V je neprázdná konečná množina vrcholů
- E je množina orientovaných hran

Orientovaná hrana $(u, v) \in E$ je uspořádaná dvojice různých vrcholů $u, v \in V$.
Říkáme, že u je předchůdce v a v je následník u.

Úplný graf K_n

Úplný graf K_n

Úplný graf na n ($n \geq 1$) vrcholech K_n je graf $\left(v, \binom{V}{2}\right)$, kde $|V| = n$.

Úplný bipartitní graf K_n

Úplný bipartitní graf K_n

Nechť $n \geq 1$ a $m \geq 1$. Úplný bipartitní graf $K_{n,m}$ s n vrcholy v jedné partitě a m vrcholy v druhé partitě je graf $(A \cup B, \{\{a, b\} | a \in A, b \in B\})$, kde $A \cap B = \emptyset$, $|A| = n$ a $|B| = m$.

Kružnice C_n

Kružnice C_n

Nechť $n \geq 1$. Kružnice délky n (s n vrcholy) je graf $(1, \ldots, n, i, i+1 | i \in 1, \ldots, n-1 \cup 1, n)$.

Cesta P_m

Cesta P_m

Nechť $m \geq 0$. Cesta délky m (s m hranami) je graf $(0, \ldots, m, i, i+1 | i \in 0, \ldots, m-1)$.

Doplněk grafu G

Doplněk grafu G

Doplněk $\overline{G}$ grafu $G = (V, E)$ je graf $\left(V, \binom{V}{E} \setminus E\right)$.

Izomorfismus grafů

Nechť G a H jsou dva grafy. Funkce $f : V(G) \to V(H)$, která:

- je bijekcí,
- pro každou dvojici $u, v \in V(G)$ platí: $(u, v, \in) E(G) \Leftrightarrow f(u), f(v) \in E(H)$.

Automorfismus

Automorfismus G je izomorfismus se sebou samý, tedy $f : V(G) \to V(G)$, která:

- je bijekcí,
- pro každou dvojici $u, v \in V(G)$ platí: $(u, v, \in) E(G) \Leftrightarrow f(u), f(v) \in E(G)$.

Stupeň vrcholu $deg_G(v)$

Stupeň vrcholu $deg_G(v)$

Počet hran grafu G obsahujících vrchol v.

Okolí stupně $N_G(v)$

Uzavřené okolí

Množina všech sousedů vrcholu v v grafu G.
Množinu $N_G[v] = N_G(v) \cup \{v\}$ nazveme uzavřené okolí.

Regulární graf

Graf G je r-regulární, pokud stupeň každého vrcholu je r.
Graf je regulární, pokud je r-regulární pro nějaké r.

Princip sudosti a jeho důsledek

Pro každý graf $G = (V, E)$ platí

$$\sum_{v \in V} deg_G(v) = 2|E|$$

.

Z tohoto vztahu plyne, že počet vrcholů lichécho stupně je sudý.

Reprezentace grafu

Matice sousednosti

Čtvercová matice $A_G = (a_{ij})_{i,j}^n$ je definována předpisem:

$$\begin{cases} 1 & \{v_i, v_j\} \in E \\ 0 & \text{jinak} \end{cases}$$

Reprezentace grafu

Seznam sousedů

Pro každý vrchol v grafu G uchováváme seznam sousedů (např. spojový seznam). Paměťová složitost je $|V| + 2|E|$.

Indukovaný podgraf

Graf H je indukovaný podgraf grafu G, když $V(H) \subseteq V(G)$ a $E(H) = E(G) \cap \binom{V(H)}{2}$.
Podgraf se značí $H \leq G$.

Sled

Sled

Doplnit

Cesta v grafu

Podgraf izomorfní nějaké cestě P. Délka cesty je počet hran. V ohodnoceném grafu je pak délka součtem ohodnocení jednotlivých hran.

Souvislá komponenta

Indukovaný podgraf H grafu G, který:

- je souvislý $(\forall u, v \in V : \exists P\,(u, v))$,
- vyvrací existenci souvislého podgrafu F, $F \neq H$ takového že $H \subseteq F$.

V inkluzi se jedná o maximální souvislý podgraf.

DFS

Prohledávání do hloubky

Po výběru počátečního vrcholu p z V spuštěn rekurzivní algoritmus:

- Pokud je p otevřený vrať se ($return$).
- Označ p jako otevřený.
- Pro každého následníka spusť rekurzivně DFS algoritmus.
- Po projití všech následníků označ uzel jako uzavřený.

Orientovaná cesta

P_m

Orientovaná cesta P_m

Nechť $m \geq 0$. Orientovaná cesta s m hranami P_m je graf $(\{0, \ldots, m\}, \{(i, i+1) \,|\, i \in \{0, \ldots, m-1\}\})$.
(Oproti standardní cestě se jedná o množinu uspořádaných dvojic)

Orientovaná kružnice

C_n

Orientovaná kružnice C_n

Nechť $n \geq 2$. Orientovaná kružnice s n vrcholy je graf
$(\{1, \ldots, n\}, \{(i, i+1) \,|i \in \{1, \ldots, n-1\}\} \cup \{(n, 1)\})$

Vstupní stupeň

$$deg_G^+(v)$$

Vstupní stupeň $deg_G^+(v)$

Počet orientovaných hran hran orientovaného grafu G končících ve vrcholu v.

Výstupní stupeň

$$deg_G^+(v)$$

Výstupní stupeň $deg_G^+(v)$

Počet orientovaných hran hran orientovaného grafu G vycházejících z vrcholu v.

Symetrizace

orientovaného grafu

Neorientovaný graf $sym(G) = (V', G')$ kde $V' = V$, a $u, v \in E'$ právě když $(u, v) \in E$ nebo $(v, u) \in E$.

Slabá souvislost

Graf $G = (V, E)$, jehož symetrizace $sym(G)$ je souvislá.

Silná souvislost

Graf, kde pro každé vrcholy $u, v \in V$ existuje orientovaná cesta z u do v
a současně existuje orientovaná cesta z v do u (ne nutně ta samá).

Strom, les a list

Strom Graf G, který je souvislý a acyklický.

Les Graf G, který neobsahuje kružnice (nesouvislý, komponenty jsou stromy).

List Vrchol v jehož stupeň $deg_G(v) = 1$.

Tvrzení o existenci listů

Tvrzení o existenci listů

Každý strom T s alespoň 2 vrcholy obsahuje alespoň 2 listy.
Lze dokázat pomocí hledání nejdelší cesty.

Věta o trhání listů

Věta o trhání listů

Je-li $G = (V, E)$ graf na alespoň 2 vrcholech a $v \in V(G)$ je list. Pak:

- G je strom.
- $G - v$ je strom.

Vlastnosti stromů

Vlastnosti stromů

- G je strom.
- Pro každé dva vrcholy $u, v \in V$ existuje právě jedna cesta z u do v.
- G je souvislý a vynecháním libovolné hrany vznikne nesouvislý graf.
- G je souvislý a platí $|V| = |E| + 1$.

Kostra grafu

Nechť G je souvislý.
Podgraf K grafu G nazveme kostrou G, pokud $V(K) = V(G)$ a K je strom.

Vzdálenost dvou vrcholů $d(u, v)$

Vzdálenost dvou vrcholů $d(u, v)$

Délka nejkratší cesty v G spojující u a v.
Pokud cesta neexistuje (jsou z jiných komponent),
pak $d(u, v) = \infty$.

Prohledávání do šířky

BFS

Prohledávání do šířky BFS

DFS začíná výběrem počátečního vrcholu s a dej mu hodnotu 0, následně:

- Označ všechny vrcholy jako nenalezené.
- Přidej s do fronty.
- Dokud není fronta prázdná
  - Odeber vrchol fronty v a pro každého jeho následníka w:
  - Je-li w nenalezený, označ ho jako nalezený a jeho hodnotu nastav na hodnotu $v + 1$, w přidej do fronty

Vlastnosti kostry BFS

Doplnit

Řadící algoritmy

BubbleSort

Funguje na principu probublávání velkých prvků. Algoritmus vezme dva prvky a pokud jsou ve špatném pořadí, prohodí je. Následně se posouvá o prvek dál. Ukončuje se ve chvíli, kdy v jednom běhu neproběhlo žádné prohození.
Složitost $O(n^2)$, stabilní, in-place, datově citlivý.

Řadící algoritmy

SelectSort

Funguje na principu vyhledávání nejnižšího prvku. Vstup se rozdělí na seřazenou a neseřazenou posloupnost. V každém kroku se vybere minimum z neseřazené a vloží se na konec seřazené. Volné místo se vyplní sešoupnutím prvků. Složitost $O(n^2)$, nestabilní, in-place a datově necitlivý.

Řadící algoritmy

InsertSort

Na principu řazení vkládáním. Vstup se rozdělí na seřazenou a neseřazenou posloupnost. V každém kroku se vezme první prvek neseřazené posloupnosti a vloží se na správné místo v seřazené.
Složitost $O(n^2)$ (v lepším případě $O(n)$), stabilní, in-place a datově citlivý.

Třídící algoritmus

TopSort

Třídící algoritmus TopSort

- Pro každou hranu (u, v), proveď $D(u)+ = 1$. Všechny vrcholy v, které mají $D(v) = 0$ přidej do fronty.
- Dokud není fronta prázdná vezmi a vypiš vrchol v z čela fronty a pro každou hranu směřující z něho do w proveď $D(w)- = 1$, pokud nyní $D(w) = 0$, zařaď ho do fronty.

Řadící algoritmy

Vlastnosti

Pamětová náročnost Rozlišují se In-place a Out-of-place algoritmy.

Stabilita Stabilní, pokud správně seřazené prvky ze vstupu mají stejné pořadí i na výstupu.

Citlivost Určuje, jestli se mění časová složitost na základě vstupu.

Zakořeněný strom

předek, potomek, otec a syn

Zakořeněný strom Uspořádaná dvojice (T, k), kde $k \in V(T)$ je jeden zvolený vrchol stromu T zvaný **kořen**.

Předek a potomek Leží-li u na cestě z v do kořene, pak je u **předek** a v **potomek**.

Otec a syn Pokud je navíc $\{u, v\} \in E(T)$ hrana, u je **otec** a **syn**.

Binární strom

Strom, který splňuje:

- je zakořeněný,
- každý vrchol má nejvýše dva syny,
- u synů rozlišujeme, který je pravý a který levý.

Binární minimová halda

Struktura tvaru binárního stromu, splňující:

- **Tvar haldy**: Strom má všechny hladiny kromě poslední plně obsazené. Poslední hladina je zaplně zleva doprava.
- **Haldové uspořádání**: Je-li v vrchol a s jeho syn, pak platí $k(v) < k(s)$.

Počet hladin

binární haldy

Počet hladin binární haldy

Binární halda s n prvky má $\lfloor \log n \rfloor + 1$ hladin.

Binární halda

vložení prvku

Binární halda dovoluje vložit prvek na pozici listu. Tímto ale mohlo být porušeno haldové pravidlo. Je tedy potřeba prvek *probublat* na správné místo. Probulání probíhá provnáním s hodnotou v rodiči (pokud je v rodiči větší, prohodí se).

Složitost je $O(\log n)$.

Binární halda

odstranění minima

Odstranit minimum není triviálně možné. Lze ho ale prohodit s nepravějším listem, následně odstranit a list probublat dolů na správné místo.
Složitost je $O(\log n)$.

Binární halda

reprezentace polem

Pro reprezentaci haldy lze snadno využít pole. Pokud uzly označíme čísly $1, \ldots, n$, pak pro vrchol v s indexem i platí:

- pravý syn má index $2i + 1$,
- levý syn má index $2i$,
- otec má index $\lfloor \frac{i}{2} \rfloor$,
- číslo $i \mod 2$ udává, zda-li v je pravý syn

Binární halda

algoritmus BuildHeap

Haldu lze složit v čase $O(n)$ zabubláním prvků, které nejsou listy.
Algoritmus vezme vrcholy $\lfloor \frac{n}{2} \rfloor, \ldots, 1$ a postupně na ně zavolá operaci BubbleDown.

Binární halda

řazení HeapSort

Prvky $x_1, \ldots, x_n$ vložíme do pole a zavoláme na něj *BuildHeap*.
Nyní opakovaně voláme *HeapExtractMin* a hodnoty ukládáme do výstupního pole.
Složitost řazení je $O(n \log n)$.

Amortizovaná analýza

nafukovacího pole

Uvažujme prázdné nafukovací pole. Potom celková časová složitost posloupnosti n operací **NPInsert** je $O(n)$, neboli amortizovaná složitost je $O(1)$.

Amortizovaná analýza

binární sčítačky

Uvažujme vynulovanou birnání sčítačku. Potom celková složitost posloupnosti n volání operace Inc měřená počtem bitových inverzí je $O(n)$, neboli amortizovaná složitost je $O(1)$.

Binomiální strom

B_k

Binomiální strom B_k

Uspořádaný (záleží na pořadí synů) zakořeněný strom, pro který platí:

1. B_0 je tvořen pouzen kořenem,
2. Pro $k \geq 1$ získáme B_k ze stromů $B_0, B_1, \ldots, B_{k-1}$ tak, že přidáme nový kořen a kořeny těchto stromů (v tomto pořadí) přidáme jako jeho syny.

Počet hladin, vrcholů a stupeň kořene

Binomiálního stromu B_k

Počet hladin, vrcholů a stupeň kořene Binomiálního stromu B_k

Počet hladin B_k je $k + 1$, počet vrcholů je 2^k a stupeň kořene je k.

Počet vrcholů v hloubce i

Binomiálního stromu B_k

Počet vrcholů v hloubce i Binomiálního stromu B_k

Počet vrcholů stromu B_k v hloubce i, $0 \leq i \leq k$, je $n_k(i) = \binom{k}{i}$.

Binomiální minimová halda

Binomiální minimová halda

Halda obsahující n prvků se skládá ze souboru binomiálních stromů $T = T_1, \ldots, T_l$, kde:

- Pro každý strom T_i platí haldové uspořádání.
- V souboru T se žádný řád binomiálního stromu nevyskytuje vícekrát.
- Soubor stromů je uspořádán vzestupně.

Výskyt stromu T

v Binomiální haldě

Výskyt stromu T v Binomiální haldě

Binomiální strom B_k se v souboru stromů n-prvkové binomiální haldy vyskytuje právě tehdy, když je ve dvojkovém zápisu čísla n k-tý nejnižší bit nastaven na 1.

Sloučení

Binomiálních hald

Sloučení Binomiálních hald

Sloučení hald probíhá obdobně jako sčítání binárních čísel pod sebou. Postupně se slučují stromy stejného řádu způsobem:

- Pokud obě haldy obsahují stromy slučovaného řádu, vznikne strom B_{k+1} a bere se jako přenos.
- Pokud mám přenos a obě haldy mají stromy se stejným řádem, vypíše se přenos a sloučí se stromy B_{k+1} čímž vzniká nový přenos.
- Pokud mám přenos a jen jedna halda obsahuje strom B_{k+1}, sloučím ho s přenosem a vznikne nový přenos.
- Pokud mám přenos a ani jedna halda neobsahuje strom B_{k+1}, vypíšu ho.

Časová složitost je $O(\log n)$.

Vložení prvku

Binomiálních hald

Probíhá pomocí sloučení hald (z prvku se stane B_0).
Časová složitost $\Theta(1)$.

Odebrání minima

Binomiálních hald

Odebrání minima Binomiálních hald

- Najdeme v haldě strom T, jehož kořen je minimum. Odpojíme T z haldy.
- Odtrhneme všechny syny a vložíme je do nové haldy H'.
- Sloučíme BHMerge(H, H').

Časová složitost je $O(\log n)$.

Binární vyhledávací strom

BVS

Binární strom, v jehož každém vrcholu v je uložen unikátní klíč $k(v)$. Pro každý v pak musí platit:

- Pokud a je v levém podstromu, pak $k(a) < k(v)$.
- Pokud b je v levém podstromu, pak $k(b) > k(v)$.

Binární vyhledávací strom

Vložení prvku

Vkládám vrchol k do stromu T:

- Pokud T je prázdný, vytvoř vrchol z k a vrať ho.
- (nebo) Pokud k je menší než kořen, vlož k do $L(T)$.
- (nebo) Pokud k je větší než kořen, vlož k do $R(T)$.

Časová složitost je $O(n)$.

Binární vyhledávací strom

Odstranění prvku

- (nebo) Pokud je k je menší než kořen, mažeme z levého podstromu.
- (nebo) Pokud je k je větší než kořen, mažeme z pravého podstromu.
- (nebo) Pokud se rovná a má jednoho syna, nahradíme ho synem.
- (nebo) Pokud se se rovná a má oba syny, nahradíme ho následníkem.

Časová složitost je $O(n)$.

Binární vyhledávací strom

nalezení následníka

Binární vyhledávací strom nalezení následníka

Při hledání následníka vrcholu v může nastat:

- v má pravého syna: následník je minimum z pravého podstromu
- v je levý syn, vrať otce v
- (jinak) u je otec v, dokud u je pravým synem: $u \leftarrow \text{otec}(u)$, následně vrať minimum z otce u (může být *null*)

Binární vyhledávací strom

nalezení předchůdce

Binární vyhledávací strom nalezení předchůdce

Při hledání předchůdce vrcholu v může nastat:

- v má levého syna: následník je maximum z pravého podstromu
- v je pravý syn, vrať otce v
- (jinak) u je otec v, dokud u je levým synem: $u \leftarrow \text{otec}(u)$, následně vrať maximum z otce u (může být *null*)

AVL Strom

AVL Strom

Binární vyhledávací strom, kde pro každý vrchol v platí, že $|h(l(v)) - h(r(v)) \leq 1|$.
Tedy, že hloubky podstromů se mohou lišit maximálně o 1.

AVL Strom

Jednoduchá rotace

Pokud se ve vrcholu v liší hloubky pod stromů více neže o 1 a zároveň do v se nerovnost propagovala z levého (L), resp. pravého (R) syna a do toho z jeho levého (L), resp. pravého (R) syna pak provedeme LL, resp. RR rotaci.

AVL Strom

Dvojitá rotace

Pokud se ve vrcholu v liší hloubky pod stromů více neže o 1 a zároveň do v se nerovnost propagovala z levého (L), resp. pravého (R) syna a do toho z jeho levého (R), resp. pravého (L) syna pak provedeme LR, resp. RL rotaci. Jednoduchou rotací lze přejít do stavu LL nebo RR.

Hashování

Pro univerzum klíčů U zvolíme konečnou množinu přihrádek $P = 0, \ldots, m$ (**hashovací tabulku**) a **hashovací funkci** $h : U \to P$, která každému klíči přiřadí jednu přihrádku. Množinu klíčů $K \subset U$ umisťujeme do přihrádek $\forall k \in K : h(k)$.

Hashování

s řetízky

Hashovací tabulka je pole m, přihrádek, které jsou buď prázdné nebo v nich začínají spojové seznamy uložených prvků.

Hashování

s otevřenou adresací

Každý klíč $k \in U$ má **vyhledávací posloupnost** $h(k, 0), \ldots, h(k, m-1)$, která určuje v jakém pořadí se budou přihrádky zkoušet. Pokud algoritmus narazí na zaplněnou přihrádku, zkusí další z posloupnosti.

Hashování

s lineárním přidáváním

Prohledávací posloupnost je dána funkcí

$$h(k, i) = (f(k) + i) \mod m$$

, kde $f(k)$ je hashovací funkce a i je počet neúspěšných pokusů. Funkce tedy zkouší přihrádky jdoucí po sobě.

Hashování

s dvojitým hashováním

Hashovací posloupnost je dána funkcí $h(k, i) = (f(k) + i \cdot g(k) \mod m)$ kde f a g jsou různé hashovací funkce a i je počet neúspěšných pokusů.

MergeSort

Vsupní posloupnost n prvků rozdělíme na $\lfloor \frac{n}{2} \rfloor$ a $\lceil \frac{n}{2} \rceil$ prvků a na obě poloviny zavoláme rekurzivně stejný algoritmus. Rekurze se zastavuje u jednoprvkové posloupnosti, která je seřazená, Při návratu z rekurze se seřazené posloupnosti slévají do jedné. Časová i paměťová složitost $\Theta(n)$.

Rychlé násobení čísel

x a y

Rychlé násobení čísel x a y

Spočívá v rozdělení čísel x a y na $x = a10^{\frac{n}{2}} + b$,
$x = c10^{\frac{n}{2}} + d$.

V prvním kroku lze vyjádřit:
$xy = ac10^n + (ad + bc)10^{\frac{n}{2}} + bd$ s $\Theta(n)$.

V druhém kroku lze rozepsat
$(ad + bc) = ((a + b)(c + d) - ac - bd)$ a dospět k
$xy = ac10^n + ((a + b)(c + d) - ac - bd)10^{\frac{n}{2}} + bd$
se složitostí $\Theta(n^{\log 3})$.

QuickSelect

k-tý nejmenší prvek

QuickSelect k-tý nejmenší prvek

Rekurzivní algoritmus, zvolím si pivota p a posloupnost rozdělím na L (prvky menší než p), P (prvky větší než p) a S (prvky rovné p).

- Pokud $k \leq |L|$ vrať $QuickSelect(L, k)$
- (nebo) je-li $k \leq |L| + |S|$ vrať p
- (nebo) vrať $QuickSelect(P, k - |L| - |S|)$

QuickSort

Obdobně jako *QuickSelect* rozděluje posloupnost na L, P a S podle pivota p. Rekurzivně seřadí L a P a části spojí za sebe a vrátí.

Dolní mez složitosti vyhledávání

v porovnávacím modelu

Každý deterministický algoritmus v porovnávacím modelu, který nalezne prvek v n-prvkové seřazené posloupnsti, pužije nejméně $O(\log n)$ porovnání.

Dolní mez složitosti řazení

v porovnávacím modelu

Každý deterministický algoritmus v porovnávacím modelu, který seřadí n-prvkovou posloupnost použije alespoň $O(n \log n)$ porovnání.

CountingSort

Řadí n prvků z množiny $\{0, \ldots, r\}$. Nejprve projde pole a spočítá, kolikrát tam který prvek je. Z tohoto počtu vypočte následně na jakou pozici prvek ve výstupním poli umístí (drží si tabulku kde si napočítá indexy). Když pak prochází vstupní pole, dívá se do tabulky indexů a index inkrementuje.

LexCountingSort

Používá se k řazení n k-tic z množiny $\{1, \ldots, r\}^k$.
Algoritmus řadí pomocí algoritmu *CountingSort* podle i-té souřadnice zleva (vezme i-tou souřadnici všech k-tic a podle ní odzadu řadí).
Paměťová složitost $\Theta(k \cdot (n + r))$, časová složitost $\Theta(kn + r)$.

Nejdelší rostoucí podposloupnost

Pole se prochází odzadu. Pro každý prvek se udržuje délka nejdelší rostoucí posloupnosti. Pro každý prvek v se nejprve uloží hodnota 1 a následně se prochází všechny prvky w za ním. Pokud je prvek $h(v) < h(w) + 1$ pak $h(v) = h(w) + 1$. Na konci pro každé číslo máme nalezenu nejdelší rostoucí podposloupnost.

Triangulace kovexního mnohoúhelníku

Triangulace kovexního mnohoúhelníku

Rozdělení mnohoúhelníku na trojúhelníky pomocí diagonál (hrana $a_i a_j$ kde a_i a a_j) jsou nesousední vrcholy. Každá triangulace má $n-3$ diagonál. Minimální triangulace je nejmenší ze součtů délek diagonál. Pro každý vrchol $j = \{1, \ldots, n\}$ číslo $M[i,j]$ vyjadřuje MTM i vrcholů za sebou počínaje j. Definujeme $M[2,j] = 0$ a $M[3,j] = |a_j a_{j+2}|$

Následně pak platí:

$M[i,j] = min_{1 \leq k \leq i-2}(M[k+1,j] + M[i-k, j+k]) + ||a_j a_{j+i-1}||.$

MTM je $M[n,1] - ||a_1 a_n||$.

Editační vzdálenost řetězců

Editační operace na řetězci je *vložení*, *smazání* nebo *změna* jednoho znaku.

Editační vzdálenost řetězců $x = x_1, \ldots, x_n$ a řetězců $y = y_1, \ldots, y_m$ zn. $L(x, y)$ je nejmenší počet editačních operací potřebných k tomu, abychom z prvního řetězce vytvořili druhý.

Editační vzdálenost řetězců

algoritmus

Editační vzdálenost řetězců algoritmus

Rekurzivní algoritmus vzdálenosti suffixů (tedy měří se od konce). V každé iteraci následně provedu $EditRec(i, j)$:

- Pokud $i > n$ vracím $m - j + 1$, pokud $j > m$ vracím $n - i + 1$
- $\ell_z = EditRec(i + 1, j + 1)$
- Pokud $x_i \neq y_j$: $\ell_z = \ell_z + 1$.
- $\ell_s = EditRec(i + 1, j)$, $\ell_v = EditRec(i, j + 1)$.
- Vrať $min\{\ell_z, \ell_s, \ell_v\}$.

Minimální kostra

ohodnoceného grafu

Existuje $G = (V, E)$ souvislý neorientovaný graf a $w : E \to \mathbb{R}$ je **váhová funkce**, která přiřazuje hranám čísla - váhy.

Minimální kostra je taková, která má mezi všemi kostrami minimální váhu.

Jarníkův algoritmus

hledání minimální kostry

Jarníkův algoritmus hledání minimální kostry

- Začíná se se stromem, který jeden vrchol a žádnou hranu.
- V dalším kroku se vybere incidentní hrana s tímto vrcholem s nejmenší váhou.
- Opakujeme postup: vybereme nejlehčí z hran, která vede ze stromu do původního grafu. Tento postup se opakuje dokud v původním grafu jsou vrcholy.

Časová složitost $O(nm)$, paměťová $O(n+m)$, kde $n = |V|$ a $m = |E|$.

Elementární řez grafu

Nechť A je podmožina vrcholů grafu $G = (V, E)$ a B její doplněk ($B = V \backslash \mathrm{A}$). Množina hran $\{a_i, a_j\}$ kde $a_i \in A$ a $a_j \in B$ je **elementární řez** grafu G určený množinami A a B.

Lemma řezech grafu

Lemma řezech grafu

Nechť G je graf s unikátními vahami, R nějaký jeho elementární řez a e nejlehčí hrana řezu tohoto řezu. Pak e leží v každé minimální kostře grafu G.

Kruskalův algoritmus

hledání minimální kostry

Nechť G je graf s unikátními vahami, R nějaký jeho elementární řez a e nejlehčí hrana řezu tohoto řezu. Pak e leží v každé minimální kostře grafu G.

Kruskalův algoritmus

hledání minimální kostry

Kruskalův algoritmus hledání minimální kostry

Na principu vyhledávání nejlevnějších hran (vynechávají se ty, které tvoří cyklus).

- Seřaď hrany podle váhy a vytvoř strom $T = (V, \emptyset)$.
- Pro $i = 1, \ldots, m$ opakuj: pokud krajní body hrany e_i leží v různých komponentách, přidej je do stromu (v opačném případě tvoří cyklus)

Složitost je $O(m \log n + n^2)$.

Struktura Union-Find

pomocí keříků

Struktura Union-Find pomocí keříků

Struktura je reprezentovaná polem, kde si každý syn drží index rodiče. Find (udávající zda jsou dva vrcholy ve stejné komponentě) traverzuje přes rodiče až do kořene keříku a porovnává jestli mají stejný kořen. Každý kořen si drží svou hloubku. Při slučování dvou keřů se mělčí dát pod hlubší, tím se nezmění hlouba. Pokud mají stejnou hloubku, jeden se zapojí pod druhý a zvýší se hloubka nového kořene.
Kruskalův algoritmus má následně složitost $O(m \log n)$.

Dijkstrův algoritmus

nejkratší cesta v ohodnoceném grafu

Dijkstrův algoritmus nejkratší cesta v ohodnoceném grafu

- Všechny označím jako nenalezené, $h(v) = \infty$, $h(v_0) = 0$, v_0 otevřený
- Dokud existuje otevřený vrchol v, vyber takový, který má nejmenší $h(v)$. Pro následníky w vrcholu v:
  - Pokud $h(w) > h(v) + l(v, w)$: $h(w) = h(v) + l(v, w)$, stav(w) otevřený a $P(w) = v$.
  - Po průchodu všemi sousedy zavři v.

S použitím haldy je složitost $O((n + m) \cdot \log n)$.

Funguje i na grafech se záporným ohodnocením hran, ale bez záporných cyklů.

Relaxace

Relaxace je obecný případ Dijkstrova algoritmu, kde při vybírání otevřeného vrcholu se bere libovolný (nikoliv nejmenší). Na rozdíl od Dijkstry může jeden vrchol otevřít i uzavřít vícekrát.

Bellman-Fordův algoritmus

nejkratší cesta v ohodnoceném grafu

Bellman-Fordův algoritmus nejkratší cesta v ohodnoceném grafu

Funguje na podobném principu jako Dijkstrův algoritmus. Místo vybírání prvku s nejnižším bere vrchol z čela fronty pro jeho následníky provádí:

- Pokud $h(w) > h(v) + l(v, w)$:
  - $h(w) = h(v) + l(v, w)$
  - Pokud stav$(w) \neq$ otevřený, přidej w do fronty
  - stav(w) nastav na otevřený
  - uzavři v

Oproti Dijkstrovi také znovu otevírá již uzavřené uzly.

Bellman-Fordův algoritmus

nejkratší cesta v ohodnoceném grafu

Bellman-Fordův algoritmus nejkratší cesta v ohodnoceném grafu

Funguje na podobném principu jako Dijkstrův algoritmus. Místo vybírání prvku s nejnižším bere vrchol z čela fronty pro jeho následníky provádí:

- Pokud $h(w) > h(v) + l(v, w)$:
  - $h(w) = h(v) + l(v, w)$
  - Pokud stav$(w) \neq$ otevřený, přidej w do fronty
  - stav(w) nastav na otevřený
  - uzavři v

Oproti Dijkstrovi také znovu otevírá již uzavřené uzly. Složitost $O(nm)$

Zjednodušený Bellman-Fordův algoritmus

nejkratší cesta v ohodnoceném grafu

V cyklu pro $i = 1, \ldots, n$ vezmeme každou hranu a podíváme, jestli její cílový vrchol nemá větší hodnotu než počáteční zvývšený o váhu cesty, pokud ano vložíme novou hodnotu do v a nastavíme mu jako předka vrchol u.