Teorie

pondělí 19. února 2024 11:02

1. základní pojmy

graf - vrchol - nějaký objekt - propojený hranami - vztahy/relace
matematicky je hrana neuspořádaná dvojce vrcholů

obecný graf
multigraf - povolený vícenásobný hrany mezi dvěma vrcholama, bez smyček
obyčejný graf - bez smyček i bez vícenásobnými
    smyčka - hrana (1, 1) třeba

{1,2} - množina neorientovaná dvojce/hrana
(1,2) - posloupnost z 1 do 2

orientovaný graf - uspořádaná dvojce množin a hran kde hrana je **uspořádaná** dvojice vrcholů
    - přirozenější pro PC fun fact

cesta Pn (n délka), kružnice Cn (n počet vrcholů), úplný graf (všechny možné hrany)
bipartitní graf - dvě parity/skupiny, hrany jen mezi paritama Kn,m
rovinný graf - nakreslení aby se hrany neprotínali, strom - oboje později v přednášce

sled - vrchol hrana vrchol hrana,.. - můžou se opakovat
tah - neopakují se hrany (domeček jedním tahem)
cesta - sled kde se neopakují ani hrany ani vrcholy
kužnice - cesta (nebo tah) ve které v1=vn+1 (první a poslední vrcholy se shodujou)
    eulerovská kružnice je tah né cesta - eulerovský tah s totožným počátkem a koncem
eulerovský tah - obsahuje všechny hrany

podgraf podmnožina hran s podmnožinou hran
indukovaný podgraf - smažeme jen ty hran který potřebujeme smazat když smažeme daný vrchol, žádný jiný hrany nemažem

isomorfismus - mezi dvěma grafy, v podstatě jde o tom zjistit zda jsou grafy stejné jen s jinak pojmenovanýma vrcholama
asi těžký jelikož permutace přejmenování je faktoriál hrubá síla je big číslo, když je struktura jednoduchá tak pohoda chapeš
    Stejne ale jinak znazornene grafy

cesta a kružnice v grafu je když n je i počet vrcholů asi

nesouvislý - neexistuje cesta mezi vrcholy - graf je rozdělený
maximalní - nejde zvětšit
komponenta grafu je souvislý pod graf, který je maximální (nejde zvětšit) - čili kus nesouvislého grafu chapeš

vzdálenost je nejkratší cesta (metrika), může být jak přirozené tak reálné číslo (když mají hrany reálnou velikost)

excentricita - vzdálenost vrcholu k nejvzdálenějšímu vrcholu
- [ ] jsem línej přemýšlet - průměr grafu, poloměr, reiferní vrcholy, centrální vrcholy, matice vzdáleností

stupeň vrcholu - počet hran vycházejících z vrcholu
skóre - neklesající posloupnost stupňů v grafu

- [ ] relace - idk

2. Algoritmizace

fukce ze vstupu získat výstupy
porovnání: rychlost (časová složitost), paměťová náročnost (paměťová složitost), složitost implementace a robustnost
porovnání dvou funkcí (jedna je větší než druhá)
Asymptotická složitost nezávisí na výkonu pc, kvalitě překladače, zručnosti programátora atd.
    Nejhorší případ
    Pouze pro dost veliká data
Vs průměrná složitost, amortizovaná(pro velké množství opakovaných operací, prostě řeší případy jako insert do dynamického)
binární vs fibonacciho halda
    Figonacciho zrychluje třeba zmenšení klíče v dijik (O(1))

Složitost operací používaných v Dijkstrově algoritmu

	Binární halda	Fibonacci halda
Odeber Min.	$O(\lg n)$	$O(\lg n)$
Zmenšení Klíče	$O(\lg n)$	$O(1)$

invariance

pro dokázání že algoritmus funguje
invariant je vlastnost dat, platí od začátku do konce, její platnost na konci znamená splnění úkolu
    čili taková podmínka která musí furt platit po průběhu algoritmu

Kdybys chtěl se doučit, invarianty:

Důkaz správnosti algoritmu

Invariant: Vlastnost dat, která:
- platí na počátku (po inicializaci)
- zachovává se
- její platnost na konci implikuje splnění úkolu

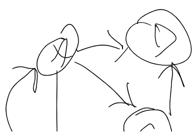
Správnost insert sortu

Invariant:
Na začátku for cyklu obsahuje $A[1 \dots i-1]$ prvních $i-1$ prvků původního pole, setříděných.
Inicialiazace:
První prvek původního pole tvoří setříděnou posloupnost délky 1.
Zachování:
Do cyklu vstupuje setříděná posloupnost prvků $A[1 \dots i-1]$. Z cyklu vystupuje setříděná posloupnost prvků $A[1 \dots i]$.
Ukončení:
For cyklus skončí pro $i = n+1$, pole $A[1 \dots n]$ obsahuje setříděnou posloupnost prvků celého původního pole.

3. Stromy

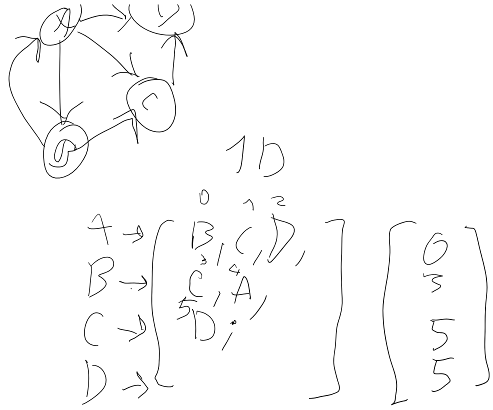
Souvislý graf bez kružnice
důkaz - lemma - sporem
písemka - definujte strom
věta o stromech/theorem:
    jednoznačnost cesty - nezi dvěma vrcholy existuje právě jedna cesta
    maximálnjí graf bez kružnice - každá přidaná hrana vytvoří kružnici
    minimální souvislý graf
    eulerův vzorec - souvislý a má o jeden víc vrcholů než hran

písemka - může tam být ten první důkaz ale to je max, zbylý no need
- [ ] Má tam ale pěkný důkazy na potrénování maybe



eulerův vzorec - souvislý a má o jeden víc vrcholů než hran

písemka - může tam být ten první důkaz ale to je max, zbylý no need
Má tam ale pěkný důkazy na potrénování maybe



kořenový strom
strom který má jeden vrchol označený jako kořen - orientujeme hrany od listů ke kořeni (weird)
každý vrchol kromě kořene má právě jednoho předka
předek a jeho potomek
Hladina, výška, šíčka, k-ární strom (binární)

datové struktury pro ukládání grafů
binární haldu jde na pohodu hodně způsoby, klidně i v 1D poli, ale normální grafy už tak nejdou

seznam hran (+ počet vrcholů) - graf G 5 vrcholů - (1,2),(3,1),(2,4),(4,3)

matice sousednosti

- [x] incidenční matice
    - řádky jsou vrcholy
    - sloupce jsou hrany - 1ky u odkud a kam, zbytek nuly

seznamy sousedů - pro a b c se dá do pole jejich sousedy po sobě a do druhého se dá kdy ti sousedi pro a b c jednotlivě začínají
- vlastně matice sousednosti ale na řádcích jsou jenom sousedi (bez null hodnot) + uloženo jen v dvou 1D polích pog
- nejlepší

Řídký graf je prej graf, který nemá moc hran oproti maxu možnému

4. procházení grafem
Depth First Search (DFS)
jdu do vrcholu, označím ho jako visitnutý, pro každoho souseda který ještě není visitnutý ho visitnu
vznikne les stromů
- tenhle strom může vypadat hodně odlišně záležíc na tom odkud začneme prohledávat

závorkovací věta nemůžem uzavřít závorku před tím než uzavřeme vnořenou jí
- d(u) - otevírání u (previsit), f(u) - zavírání u (postvisit)
- d(u) < d(v) - první otevřem u a pak v
    - Buď proto že jsou v je vnořený - v se zavře před tím než se zavře u
    - nebo že jsou oba disjunktní - u se zavře před otevřením

pi() - předchozí vrchol

pokud $d[u] < d[v]$ *pak platí buď*

$$d[u] < d[v] < f[v] < f[u] \quad \textit{vnořené}$$

nebo

$$d[u] < f[u] < d[v] < f[v] \quad \textit{disjunktní.}$$

Nemůže tedy nastat situace

$$d[u] < d[v] < f[u] < f[v].$$

Procházení do hloubky - algoritmus

```
for u ∈ V do state[u] = None, π[u] = NULL
time = 1
for u ∈ V do
    if state[u] == None then DFSVisit (u)
procedure DFSVisit (u)
    state[u] = Open, d[u] = time++                // previsit
    for v ∈ Adj[u] do
        if state[v] == None then
            π[v] = u
            DFSVisit (v)
    state[u] = Closed, f[u] = time++              // postvisit
```

Pro některé aplikace algoritmu nejsou stavy podstatné. Navštívené vrcholy lze poznat z $\pi[v]$.

5. aplikace DFS a tak asi
TOPO seřazení
DAG - orientovaný acyklický graf (bez cyklů)

Na topologické seřazení grafu můžeme nahlížet jako na umístnění jeho vrcholů na horizontální přímce, přičemž všechny hrany jsou orientovány zleva doprava.
- výhodnější posupovat od zádu, jinak by jsme museli zepředu a pak stejně znova od zádu
- Neboli seřazení vrcholů tak, aby u kdyždé hrany (u,v) byl vrchol u před v

Přes DFS - při návštěvě open vrcholu -> není DAG
- Uzavřené se přidávají do seznamu
- Pseudo - to samý ale bez pi, f, d
    - A při possvizitě teda push do orderu

kahnův algoritmus
- projdeme hrany a dáme jim in degree - stupeň vrcholu a pak něco něco nevím
- dobrej pro dynamický grafy, jinak moc efektivní není

Topologické třídění - DFS zjednodušené

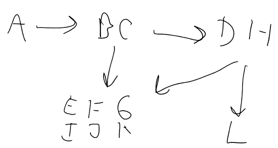
```
for u ∈ V do state[u] = None
for u ∈ V do Visit (u)
procedure Visit (u)
    if state[u] == Open then
        not a DAG
    if state[u] == Closed then
        return
    state[u] = Open
    for v ∈ Adj[u] do
        Visit (v)
    add u to the head of S
    state[u] = Closed
```

Kahnův algoritmus

```
L - množina vrcholů bez příchozí hrany
S - prázdná seznam vrcholů
while L is non-empty do
    odeber z L vrchol u
    přidej u do S
    for hrana e = (u, v) do
        odeber e z grafu
        if neexistuje příchozí hrana do v then
            přidej v do L
if G ještě obsahuje hrany then
    G není DAG
```

CPM - gantt chart - skipuju ale asi useful
- myslím že přes topologii hledáme rezervy v čase a podobně

Eulerův tah
uzvřenej (cyklus) vs neuzavřenej - stačí přidat jeden tah a je to převedení
- [x] Algorytmus - prchoází se bez vracení, rekurzivně (pomocí vracení) se pak doplňují ty vrcholy které jsme ještě neprošli - missnuli jsme je cestou



silně souvislá komponenta
Maximální množina vrcholů, kde pro každou uspořádanou dvojici (u,v) existuje cesta

Metagraf - výsledný orientovaný graf, vzniklý kontrakcí jeho SSK

Kosarajův algoritmus
- nemusíme časy postvisit, stačí topologický setřídění

Kosarajův algoritmus pro hledání SSK pomocí DFS:
1. Proveď DFS na G, urči časy postvisit $f[u]$.
2. Sestav transponovaný graf G^T (opačně orientovaný).
3. Proveď DFS na G^T. Hlavní cyklus sestupně podle hodnot $f[u]$.
4. Každý strom lesa průchodu odpovídá jedné komponentě.

6. hledání nejkratší cesty
BFS breadth first search

Průchod do šířky, BFS

```
for u ∈ V do Color[u] = White
Color[s] = Gray
d[s] = 0
π[s] = NULL
ClearQueue
EnQueue(s)
while u = DeQueue do
    for v ∈ Adj[u] do
        if Color[v] == White then
            Color[v] = Gray
            d[v] = d[u] + 1
            π[v] = u
            EnQueue(v)
    Color[u] = Black
```

Každý vrchol vložíme a vyberem z fronty + projdeme každou hranu min. jednou - O(v+e)

Prioritní fronta má odeber min, zmenši klíč a naplň frontu(i operace přidání prvku)
- Halda
- O(log n)

Dijkstra

Dijkstra's algorithm

Vstup: vrcholy V a sousednosti $Adj[u] = \{e \in E | e(u, v)\}$ s ohodnocením $w(e)$, počáteční vrchol s
Výstup: vzdálenosti $d[i]$ od s, předci $\pi[i]$ stromu minimálních vzdáleností

```
for u ∈ V do d[u] = ∞
d[s] = 0; π[s] = NULL
Naplň prioritní frontu Q vrcholy V s prioritami d[·].
while u = OdeberMin(Q) do
    for e = (u, v) ∈ Adj[u] do
        alt = d[u] + w(e)
        if alt < d[v] then
            π[v] = u; d[v] = alt
            ZmenšiKlíč(Q,v)
```

Opět O(V*čas odeber min + E*čas zmenši klíč)

Pro řídké grafy:
PF = Binární halda: $O((|V| + |E|)\log|V|)$
PF = Fibonacciho halda: $O(|V|\log|V| + |E|)$

Heuristika je dobrá při hledání cesty na mapě třeba, mezi dvěma body
- Jen se přidá tohle:

  ZmenšiKlíč(Q,v, $d[v] + h[v]$)

Bellman-ford pro záporné váhy
- Jde po krocích a nachází délky cesty daného kroku

Floyd warshall řeší i dykly atd, prostě všechny nejkratší

Stromové dat. Struktury

Prioritní fronta

Odeber min, zmenši klíč, naplň frontu/přidej do fronty

	Seznam	Tříděné pole	Binární halda	Fibonačiho halda
Vložení	$O(1)$	$O(n)$	$O(\lg n)$	$O(1)$
Odeber Min.	$O(1)$	$O(1)$	$O(\lg n)$	$O(\lg n)*$
Najdi Min.	$O(n)$	$O(1)$	$O(1)$	$O(1)$
Odebrání	$O(1)$	$O(n)$	$O(\lg n)$	$O(\lg n)*$
Zmenšení Klíče	$O(1)$	$O(n)$	$O(\lg n)$	$O(1)*$
Spojení	$O(1)$	$O(n)$	$O(m\lg(n+m))$	$O(1)$

Fibo super u zmenšení klíče

Binární halda

Rodič vždy větší než potomci
Insert - na konec stromu se přidá a pak se prohrazuje s rodičem pokud je větší
zmenši klíč - zmenší prvek na indexu i a následně prvek probublá nahoru, u inesrtu prvek nechá stejný jen probublá nahoru
Delete (nebo odeber min??? Wut?) - poslední prvek se dá na kořen(místo min), odstraní se tíme min a pak se heapifuje - prvek na kořenu se probublá tam kam má
- Heapify - rodič se prohazuje se z nejmenším potomkem - probublává se dolů

Heap sort má worst case složitost O(n log n) což je pog, ale strašný na paměť

Disjunktní rozklad množiny

máme několik množin, třeba víc stromů
U stromu:
Find - pro zjištění v jakém stromu prvek je - jde od listu ke kořeni - O(výška):

```
function Find (x)
    if x.parent == NULL then
        return x
    else
```

assert

```
function Insert(a, A[1:n])
    přidej a na konec haldy (zvětš haldu)
    DecreaseKey(n + 1, a, A[1:n+1])

function DecreaseKey(i, key, A[1:n])
    assert key <= A[i]
    A[i] = key
    while i > 1 and A[Parent(i)] > A[i] do
        Swap(A[Parent(i)],A[i])
        i=Parent(i)
```

Idk co je ten assert a key ale asi to je jedno

```
function AccessMin(A[1:n])
    return A[1]

function DeleteMin(A[1:n])
    Swap(A[n],A[1]);
    zmenši haldu na n − 1;
    Heapify(1, A[1:n-1])
```

Heapify - honestly idk ale postup je vlevo

Heapify

Vytvoření haldy s kořenem i pokud podstromy už jsou haldami:
$\texttt{Left}(i) = 2i$, $\texttt{Right}(i) = 2i + 1$

```
function Heapify (i,A[1 : n])
    // najdi minimum z i,l,r
    min = i; l = Left(i); r = Right(i)
    if l < n && A[l] < A[min] then min = l
    if r < n && A[r] < A[min] then min = r
    if min ≠ i then
        Swap (A[i],A[min])
        Heapify (min,A[1:n])
```

```
function Find (x)
  if x.parent == NULL then
    return x
  else
    x.parent = Find (x.parent)
    return x.parent
```

Disjunktní rozklad pomocí stromů - Union

```
function Union (xRoot, yRoot)
  if xRoot.rank > yRoot.rank then
    yRoot.parent = xRoot
  else if xRoot.rank < yRoot.rank then
    xRoot.parent = yRoot
  else if xRoot ≠ yRoot then
    yRoot.parent = xRoot
    xRoot.rank = xRoot.rank + 1
```

- UNION - kvuli vyvážení připojíme menší strom pod větší, výška vzroste o 1 jen při shodě výšek
- To samo zajistí FIND $O(\log n)$.

Dohromady: FIND s amortizovanou složitostí $O(\alpha(n))$

Kostra

Podgraf který je stromem a obsahuje všechny vrcholy

Prim-jarník:

Vstup: graf $G(V, E)$, ohodnocení hran
Výstup: minimální kostra S.

Inicializuj strom kostry S jedním vrcholem.
while *S neobsahuje V* **do**
  Najdi minimální hranu e z S do $V \setminus S$.;
  Přidej e ke stromu S.

O((E+V) log V)

BFS

Primův-Jarníkův algoritmus detailně

for $u \in V$ **do** $d[u] = \infty$
$d[s] = 0$; $\pi[s] = s$
Naplň prioritní frontu Q vrcholy V s prioritami $d[\cdot]$.
while $u =$ *OdeberMin*(Q) **do**
  $d[u] = -\infty$
  for $e = (u, v) \in Adj[u]$ **do**
    if $w(e) < d[v]$ **then**
      $\pi[v] = u$; $d[v] = w(e)$
      ZmenšiKlíč(Q,v)

Kruskal

Kruskalův algoritmus

Vstup: graf $G(V, E)$. Výstup: kostra L.

vytvoř les L z vrcholů V;
naplň S hranami E;
while $S \neq \emptyset$ *a L není strom* **do**
  odeber z S hranu e s nejmenší váhou;
  if *e spojuje vrcholy různých stromů v lese L* **then**
    spoj stromy;

Vstup: graf jako pole hran E. Výstup: seznam L hran kostry.

$n =$ počet vrcholů
setřiď pole hran E
while *size*$(L) < n - 1$ **do**
  odeber z E hranu e s nejmenší váhou
  (u,v) = e
  if *Find(u)* $\neq$ *Find(v)* **then**
    Union(Find(u), Find(v))
    přidej e do L

Borůvkův:

Borůvkův algoritmus - obecně

vytvoř les L z vrcholů V
while *L není strom* **do**
  for *komponenty L_i lesu L* **do**
    Najdi minimální hranu e_i z L_i ven.
  Přidej hrany $\{e_i\}$ do L.

- V každém kroku se počet komponent zmenší alespoň na polovinu.
- složitost $O(E \log V)$
- vhodný pro paralelizaci

Pro každý strom najdeme nejmenší hranu, následně ty nejmenší vyberem a opakujem s novými stromy

Řez - množina **hran** mezi A a V \ A

Barvení

k-obarvení vrcholů - zobrazení vrcholů do barev, aby vrcholy spojené hranou měli různou barvu
Chromatické číslo grafu - minimální počet barev nutný k obarvění
Chromatický polynom - počet různých obarvení max. k barvami

Přesný algoritmy - exponencionální složitost (backtracking)
Používáme aproximace

Hladové barvení

for $v \in V$ **do**
  přiřaď v minimální barvu $b(v)$ nevyskytující se u jeho sousedů

Hladové barvení

```
for v ∈ V do
    přiřaď v minimální barvu b(v) nevyskytující se u jeho sousedů
```

náhodně/stupeň vrcholu/stupeň nasycení

kontrakce

```
while G není úplný do
    najdi nespojenou dvojici (x, y);
    proveď kontrakci x a y;
obarvi výsledný úplný graf;
zpětně rozpojuj kontrahované vrcholy a dávej jim stejné barvy;
```

dvojce může hledat s maximálním počtem sousedních vrcholů
nebo heuristika založená na max nezávislých množin

přes nezávislé množiny

Algoritmus se strategií hledat nezávislé množiny postupně.

```
for k=1,... do
    M = ∅;
    Y =všechny neobarvené vrcholy;
    while Y ≠ ∅ do
        zvol v vrchol z Y;
        přidej v do M;
        odstraň z Y sousedy v;
    obarvi vrcholy v M barvou k;
```

postupně přidává nezávislé množiny do M
buď náhodně nebo podle stupně

- [ ] duální graf
- [ ] eulerův vzorec

pro rovinný graf vždy budou stačit 5 barev

max. toky

Síť je uspořádaná čtveřice $G = ((V, E), c, s, t)$, kde:

1. (V, E) je orientovaný graf
2. $c : V \times V \to R^+$ jsou kapacity hran, $c(u, v) = 0$ pokud (u, v) neexistuje,
3. $s \in V$ je vstupní vrchol
4. $t \in V$ je výstupní vrchol.

tok sítí definuje tok pro každou dvojici vrcholů - obraze
    zobrazení f : V × V → R
    omezený - menší= než kapacita - f <= c
    -f = f
    tok 0 pro neexistující vrcholy
velikost toku - celkový výtok ze vstupního

reziduální graf - kapacita snížená o tok - pokud kládná tak in

ford-fulkerson

Schéma algoritmu:

inicializuj $f(u, v) = 0$;
while *dokud existuje zlepšující cesta* p **do**
    zvyš tok f podél cesty p

Zlepšující cesta je cesta z s do t v residuálním grafu.

edmons-karp přidává BFS na reziduální graf
    O(V*E^2)

kdyby náhodou

Edmons-Karp verze Ford-Fulkersona

1. nastav tok f na nulu (na všech hranách)
2. pro f vytvoř reziduální graf G_f
3. proveď BFS na G_f ze zdroje s
4. dopředné hrany tvoří čistou síť G_f^*
5. pokud BFS nenalezne cestu do t algoritmus končí
6. jako vylepšující cestu p vezmeme nejkratší cestu z s do t na G_f (nalezenou pomocí BFS)
7. vypočteme kapacitu zlepšující cesty $c_f(p) = \min\{c_f(e) | e \in p\}$
8. tok všemi hranami v cestě p zvětšíme o kapacitu cesty:
   for $(u, v) \in p$ **do**
       $f(u, v) = f(u, v) + c_f(p)$;
       $f(v, u) = -f(u, v)$;
9. pokračujeme bodem 2)

Párovaní grafu či co

Bipartitní graf - je obarvitelný dvěma barvami.
Tj. V lze rozělit na disjunktní množiny U a W, tak že pro každou hranu (u, w) je $u \in U$ a $w \in W$.

algoritmus - DFS nebo BFS, do U liché vrstvy do W sudé, pokud je už navštívený vrchol ve stejné paritě tak G není bipartitní -> objeví se cyklus liché délky

bipartitní má cylky pouze o sudé délky

párování

množina nesousedících hran

sousední hrany mají společný vrchol

max < největší < prefektní

hladový algoritmus se převede na maximální tok

edmonův algoritmus - páruje pomocí zlepšovacích cest + detekuje květy - kontraktuje je do jednoho