Dve chuťovky

©kuko 25.9.2019

1 Teória – Union Find

- motivácia: Kruskalov algoritmus pre najlacnejšiu kostru, kde potrebujeme vedieť dynamicky pridávať hrany do grafu a zisťovať, či sú dané dva vrcholy v jednom komponente súvislosti, teda či existuje cesta, ktorá ich spája
- trochu abstraktnejšie: na začiatku máme n prvkov, každý v jednoprvkovej množine; chceme vedieť:
  - `find(x)` – zistí meno (jedinej) množiny obsahujúcej x.
  - `union(A,B)` – nahradí množiny A a B ich zjednotením $A \cup B$
- jednoduché riešenie: každú množinu reprezentujeme ako strom, koreň je reprezentant množiny
- `find(x)` nájde koreň; `union(x,y)` nájde korene stromov s x a y – ak sú v tom istom strome, tak nič, inak napojí jeden koreň pod druhý
- nevýhoda: `find` trvá čas úmerný dĺžke cesty a tá môže byť v najhoršom prípade až lineárna
- vylepšenie #1, union by rank: rank vrcholu je na začiatku nula, `union` napojí stroj s menším rankom pod väčší; ak boli ranky rovnaké, novému koreňu zvýšime rank o 1
- indukciou sa ľahko dokáže, že:
  - strom s rankom r má výšku najviac r
  - strom s rankom r má aspoň 2^r vrcholov
  - teda rank r môže mať najviac $n/2^r$ vrcholov
- z toho vyplýva, že max. rank je $\lg n$, čo je zároveň max. hĺbka vrcholov a teda časová zložitosť klesne na $O(\lg n)$
- rank vrcholu je na začiatku nula, potom sa zvyšuje, kým je koreň a pripájajú sa k nemu stromy s rovnakým rankom, až sa možno vrchol napojí pod iný koreň a odvtedy sa jeho rank už nemení
- vylepšenie #2, path compression: vo `find` keď nájdeme koreň, prejdeme celú cestu ešte raz a napojíme všetky vrcholy na ceste priamo pod koreň
- asymptotická zložitosť jedného `find`u sa nezmení (čas sa možno zdvojnásobí), zato cesty sa budú skracovať

```
int p[1000]; // ak p[x] > 0, p[x] je otec x; ak p[x] <= 0, rank(x) = -p[x]
int find(int x) {
  if (p[x] <= 0) return x; else return p[x] = find(p[x]);
}
int union(int x, int y) {
  int rx=find(x), ry=find(y); // korene
  if (rx == ry) return;
  if (p[rx] == p[ry]) { p[rx]=ry; p[ry]--; }
  else if (p[ry] < p[rx]) p[rx]=ry;
  else p[ry] = rx;
}
```

- zložitosť jednej operácie je stále $O(\log n)$ v najhoršom prípade, preto m operácií trvá $O(m \log n)$
- ukážeme si, že tento odhad je dosť pesimistický a každá postupnosť m operácií trvá najviac $O((m+n)\log^* n)$
- iterovaný logaritmus ($\log^* n$) je počet stlačení tlačítka lg na kalkulačke, kým nedostaneme číslo ≤ 1

n	1	2	4	16	2^{16}	2^{65536}	$2^{2^{65536}}$
$\log^* n$	0	1	2	3	4	5	6

- $\log^*$ je veľmi pomaly rastúca funkcia (hoci rastie do nekonečna)

- pre porovnanie počet atómov v pozorovateľnom vesmíre sa odhaduje medzi 10^{78} a 10^{82}
- Tarjan a Leeuwen dokázali, že skutočná zložitosť je $O(n+m\alpha(n+m,n))$, kde α je tzv. inverzná Ackermannova funkcia, ktorá rastie ešte oveľa pomalšie, ale pre veľké n rastie do nekonečna; ukázali tiež, že toto je aj dolný odhad a teda m operácií union findu dokázateľne netrvá lineárny čas, ale o chlp väčší

1.1 Analýza

- rozdeľme všetky ranky na „poschodia“: dve špeciálne spodné poschodia pre ranky 0 a 1 a ďalej ranky v rozsahu $(1,2]$, $(2,4]$, $(4,16]$, $(16,2^{16}]$, $(2^{16},2^{65536}]$...
- vo všeobecnosti (okrem spodných dvoch) budeme mať poschodia s rozsahom rankov $(k,2^k]$ ... a nasledovať bude poschodia $(2^k,2^{2^k}]$, $(2^{2^k},2^{2^{2^k}}]$,... teda rozsahy exponenciálne rastú
- z toho vyplýva, že poschodí je $\leq \log^* n + 2$

- hovorí sa, že „čas sú peniaze“; my budeme mať žetóny, ktorými budeme platiť za výpočty; 1 žetón $= O(1)$ času
- dokážeme, že pomocou $O((n+m)\log^* n)$ žetónov vieme zaplatiť všetky operácie
- žetóny si rozdelíme na dva druhy:
  - F-žetóny – na každý `find` si vyhradíme $\log^* n + 3$ žetóny (spolu $O(m \log^* n)$)
  - P-žetóny – každý vrchol, ktorý Postúpi na Poschodie $(k,2^k]$ dostane Príspevok 2^k žetónov; už sme si povedali, že rank k dosiahne len $n/2^k$ vrcholov a teda na jedno poschodie nám stačí $(n/2^k)\times 2^k = n$ žetónov; keďže poschodí je $O(\log^* n)$, dokopy potrebujeme $O(n\log^* n)$ žetónov
- stačí dokázať, že keď budeme so žetónmi rozumne hospodáriť, vystačia nám; hospodáriť budeme nasledovne:
  - čas, ktorý potrebuje funkcia `find` je úmerný dĺžke cesty ku koreňu; za každý prechod po hrane (pointeri) musíme zaplatiť
  - za prvé dva skoky a za posledný skok do koreňa a prácu v ňom zaplatíme troma F-žetónmi
  - za každý prechod od x ku $p[x]$, kde $p[x]$ je na vyššom poschodí ako x zaplatíme F-žetónmi (keďže poschodí je $\log^* n$, vystačia nám)
  - naopak, ak x aj $p[x]$ sú na rovnakom poschodí, zaplatí to x zo svojich P-žetónov (tých, ktoré dostal, keď postúpil na dané poschodie)
- ostáva zdôvodniť, že P-žetóny nám vystačia
- všimnime si, že ak vrchol x platí, $p[x]$ nie je koreň (lebo posledný skok platíme F-žetónom) a teda má aspoň starého otca $p[p[x]]$ a $rank(p[p[x]]) > rank(p[x])$ a rank koreňa je väčší ako rank otca x
- keď teda pri kompresii napojíme x priamo pod koreň, tak rank $p[x]$ bude aspoň o 1 väčší ako predtým
- ešte raz: vždy, keď x zaplatí svojim P-žetónom, rank $p[x]$ sa zvýši
- ale po $\leq 2^k$ krokoch sa už $p[x]$ dostane na vyššie poschodie ako x – a odvtedy už všetky prechody z x na $p[x]$ platíme F-žetónmi; takže P-žetóny nám vystačia

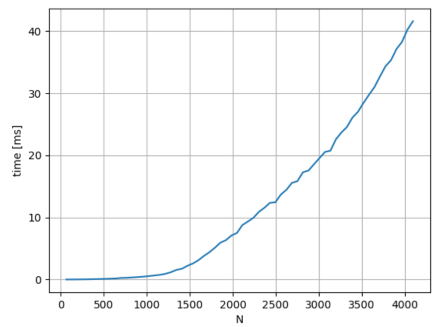
2 Prax – Transpozícia matice

- majme štvorcovú maticu $N \times N$, chceme ju transponovať (preklopiť podľa hlavnej diagonály)
- ako na to?
- nie je predsa nič jednoduchšie:

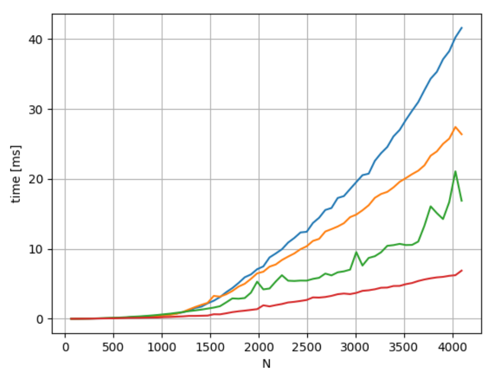
```
const int N = 1000;
int m[N][N];

void transpose() {
  for (int i=0; i<N; ++i)
```

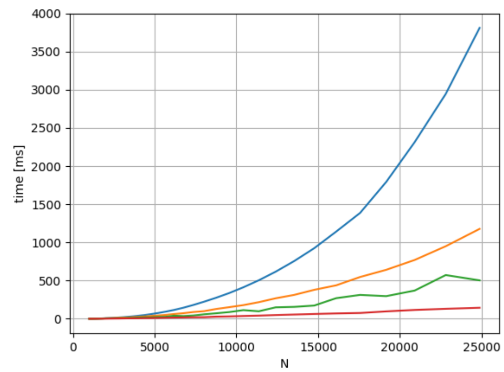
```
    for (int j=i+1; j<N; ++j)
      swap(m[i][j], m[j][i]);
}
```

- zložitosť: zjavne $\Theta(N^2)$, takže graf bude vyzerať nejak takto:

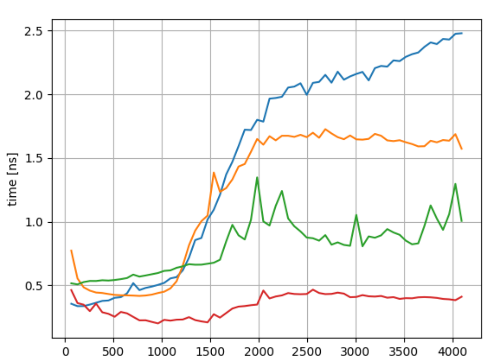


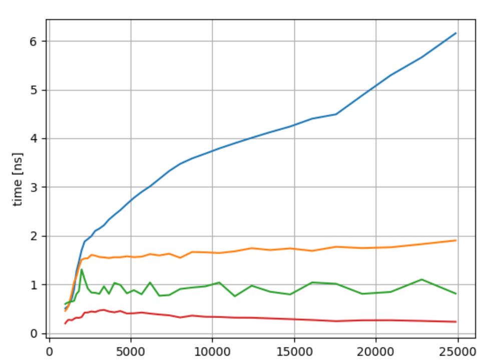
- ŠOK #1: existujú lepšie algoritmy
- ŠOK #2: náš kód je modrá čiara; vľavo pre $N \leq 4\,000$, vpravo do $25\,000$; čas v ms





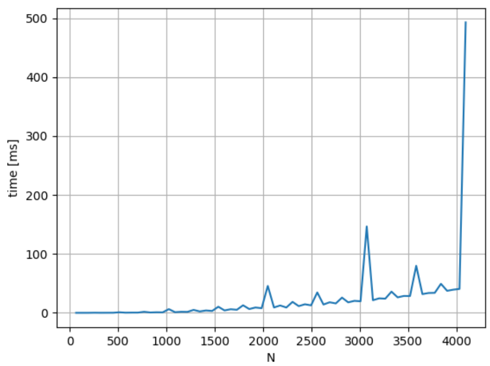
- to isté, ale čas deleno N^2 – to by mala byť konštanta (čas v ns):

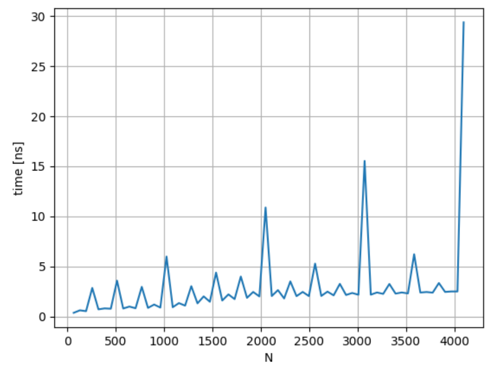




- prečo to stúpa? prečo to nie je konštanta?

- ŠOK #3: aj hentie grafy sú po odstránení drobnej/pomerne rafinovanej chyby; v skutočnosti graf pre náš kód vyzerá asi takto (vľavo v ms celkový čas; vpravo v ns delené N^2):





- čo majú znamenať tie špicaté výkyvy???