2. domácí úkol - cukrovka a lasery

Vilém Zouhar

1

1.1 Popis

Hledáme vlastně cestu délky nejvýše n v grafu, který je tvořen pouze střeženými vrcholy a hrana mezi nimi existuje pokud v původním grafu existuje nestřežená cesta mezi nimi. Udělat ale takové scvrknutí efektivně je náročné, zejména pak pamatování si cesty. Můžeme to vyřešit za použití BFS a prioritní fronty, kde **vyšší prioritu mají nestřežené vrcholy**. Index vlny pak zvýšíme pouze tehdy, když přicházíme do nového střeženého vrcholu.

1.2 Korektnost

Invariant budiž, že v každém okamžiku se v proměnné každého vrcholu v **v.vlna** vyskytuje délka nejkratší cesty z počátku do tohoto vrcholu. *Nejkratší* zde znamená, že používá nejméně střežených vrcholů, nikoliv že používá nejméně vrcholů celkem. Když se chceme pohnout dál, tak můžou nastat 4 možnosti:

- Nacházíme se v nestřeženém vrcholu
  - Narazíme na nestřežený vrchol, který pokud je navštíven, tak má stejný index vlny, neboť ho navštívil někdo, kdo se vyskytuje ve stejné vlně, nebo dokonce ve stejné nestřežené komponentně, nebo není navštíven a my mu jen předáme naše číslo vlny a vložíme do fronty. Invariant nezkazíme.
  - Narazíme na střežený vrchol, který pokud byl navštíven, tak to byl opět někdo s nejhůře naším indexem vlny. Pokud navštíven není, tak mu dáme o jedna větší index a vložíme do fronty. Invariant stále platí.
- Nacházíme se ve střeženém vrcholu
  - Narazíme na nestřežený vrchol, který pokud byl navštíven a má nižší číslo vlny než my, tak byl navštíven lepší cestou kolem nás a tedy nemá smysl jej přidávat do fronty. Vyšší číslo mít nemůže, neboť prohledáváme po vlnách. Pokud nebyl navštíven, tak mu dáme naši vlnu a vložíme do fronty.
  - Narazíme na střežený vrchol, který pokud byl navštíven, tak má nejhůř stejnou vlnu jako my. Pokud nebyl, tak mu dáme o jedna větší vlnu a vložíme do fronty. Invariant je stále funkční.

1.3 Pseudokód

Prioritní fronta má v našem případě pouze dvě priority, tedy implementačně by se dala zařídit dvěmi samostatnými frontami a operace by stále trvali jen konstantu.

```
prioritni_fronta f
for each vrchol v:
 v.vlna = -1
f.push(zacatek)
while not f.empty():
 p = f.pop()
    for each vrchol s in p.sousedi:
     if s.vlna == -1:
      s.vlna = s.strezeny?p.vlna+1:p.vlna
            s.predchudce = p
            f.push(s)
vystup(spojovy seznam v cil)
```

1.4 Složitost

Děláme pouhopouhé BFS, ale jen si trochu pospřehazujeme frontu. Slovy Hérakleitosa: "Nevstoupíš dvakrát do téhož vrcholu." Proto je paměťová složitost *O(#vrcholů + #hran)*;

2a

2a.1 Popis

Vyzkoušíme všechny možné kombinace dvou nejefektivnějších metod ničení. Abychom to udělali efektivně, tak si nejprve budovy musíme seřadit a až pak je projít. Pokud na indexu i zjistíme, že $i + p[i] < n$, pak jsme našli vítěznou kombinaci, neboť zničíme i budov a $p[i]$-krát použijeme laser na první patro.

2a.2 Korektnost

To, že použít laser na první patro je vždy silnější nebo stejné než na jakékoliv jiné, je zřejmé. Pak nám tedy zbývá n možností rozdělení typů ničění (neboť nezáleží na pořadí). Pokud se zastavíme na nějakém i, pro které $i + p[i] < n$, pak stačí použít zničení celé budovy na i nejvyšších a zničit první patro u zbylých. To můžeme, neboť ti mají výšku nejvýše $p[i]$, protože se pohybujeme v seřazeném poli.

2a.3 Pseudokód

```
p = sort(budovy)
utility = budovy.length
for i in range(0, budovy.length): // [0, N)
    m = min(m, i+p[i])
vystup(m)
```

2a.4 Složitost

Třídit umíme v rychlosti $O(N \cdot log(N))$ (ať už úpravou insertion sort, nebo quicksortem). Pak už procházíme jen lineárně.

2b

2b.1 Popis, 2b.2 Korektnost

Jeden dlouhý pán mi při temné cestě na kolej prozradil, že to lze řešit i lineárně. Uvědomíme si, že když dostaneme budovu větší než N, pak se rozhodně vyplatí ji zničit celou. Jedním průchodem

tedy odstraníme všechny tyto budovy. Zbydou nám tedy budovy v rozsahu $[0, N]$, na které můžeme použít kýblsort a pokračovat stejně jak v předchozím řešení.

Pro jednoduchost tedy předbouráme tyto budovy (nastavíme výšku na 0), pustíme předchozí algoritmus a výsledky sečteme.

2b.3 Pseudokód

```
z = 0
for i in range(0, budovy.length):
    if budovy[i] > N:
        budovy[i] = 0
        z += 1

p = sort(budovy) // bucket sort

m = budovy.length
for i in range(0, budovy.length): // [0, N)
    m = min(m, i+p[i])
vystup(m+z)
```

2b.4 Složitost

Veškeré operace trvají lineárně, tedy i takto vylepšený algoritmus poběží v $O(N)$.