Matej Delinčák, ID: 102895

Zadanie 3 – Popolvár

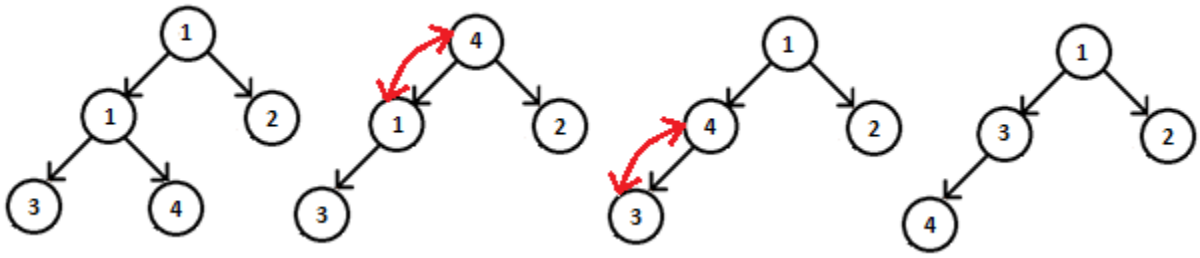
Problematika

V kráľovstve drak uniesol princezné a Popolvár má za úlohu zabiť draka a zachrániť princezné v čo najkratšom čase. Na tento problém som použil Dijkstrov algoritmus s binárnou haldou pre výber najmenej ohodnotených vrcholov. A pre výber najkratšej cesty medzi princeznami a drakom som použil permutácie všetkých ciest medzi týmito vrcholmi.

Binárna halda (Minimálna binárna halda)

Aby som zefektívnil výber najmenej ohodnoteného vrcholu pre Dijkstrov algoritmus, tak som namiesto obyčajného poľa použil binárnu haldu. Je to binárny strom, ktorý si vo vrchole udržiava najmenšie ohodnotený vrchol a jeho potomkovia sú buď väčší alebo rovný. Ja som tento strom implementoval vo forme poľa. Táto dátová štruktúra zabezpečuje operáciu insert/push v zložitosti O(1), resp. O(log n) v prípade, že vkladáme najmenší prvok. V najhoršom prípade, keď vložím najmenší prvok, musím cez haldu obnoviť. Prejdem celú vetvu až ku koreňu a porovnávam hodnoty, či sú väčšie alebo nie.

A operáciu delete/pop v zložitosti O(log n). Kedy vymažem prvý prvok (koreň) a za tento prvok nahradím posledný v poli. Následne musím zase obnoviť haldovú vlastnosť.



Operácia pop/delete najmenšieho prvku z haldy

Moja implementácia

Mám vytvorené dve štruktúry a to: *VERTEX* – vrchol v mape, ktorý má súradnice, ohodnotenie, index v halde a smerník na predošlý, aby som vedel vystavať cestu a *MIN_HEAP* – štruktúra haldy, ktorá si udržiava veľkosť

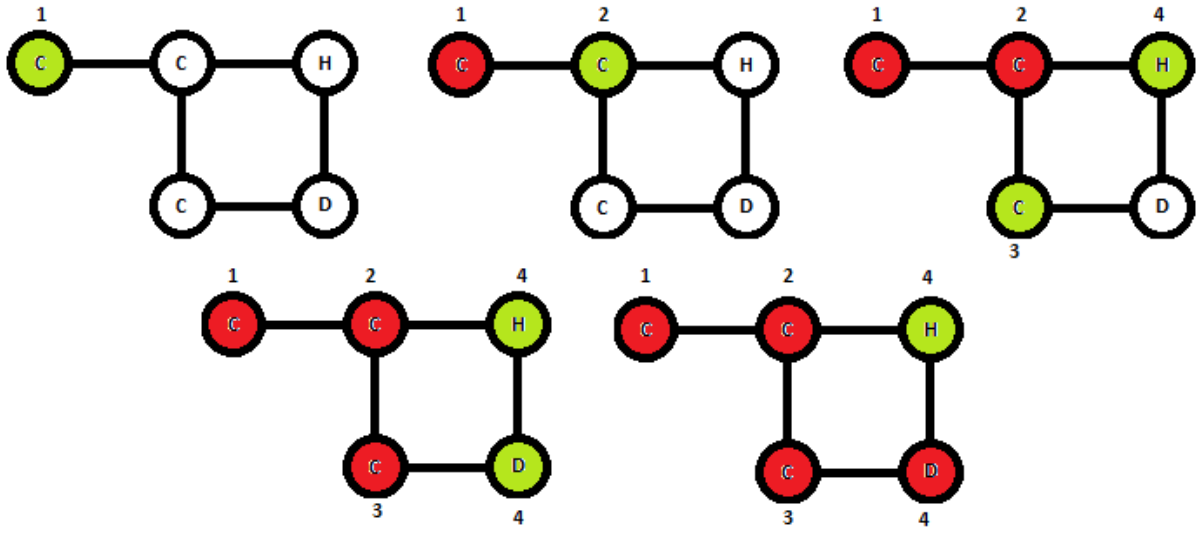
Funkcie, ktoré som použil:

- *MIN_HEAP* createHeap();* - vytvorenie haldy, ktorá nemusí mať veľkosť n*m, ale stačí omnoho menej (všimol som si, že dijkstra sa šíri ako vlna vody-stačilo by kolo 600 prvkov), ale pre istotu mám veľkosť 1000 prvkov (pre najväčšiu možnú mapu)
- *void swapInHeap(MIN_HEAP** root, int indexA, int indexB);* - vymení dve hodnoty v halde
- *void heapify(MIN_HEAP** root, int index);* - obnoví vlastnosť haldy
- *void insertHeap(MIN_HEAP** root, VERTEX* paNew);* - operácia push/vloženia do haldy
- *VERTEX* popFromHeap(MIN_HEAP** root);* - operácia vybratia minimálneho prvku z haldy

Dijkstrov algoritmus

Tento algoritmus ohodnocuje vrcholy v mape a hľadá najkratšiu cestu ku všetkým vrcholom k mape z daného počiatočného bodu. Na začiatku máme v halde začiatočný bod a všetky vrcholy okrem tohto majú ohodnotenie nekonečno. Do haldy pridáme jeho susedov s novým ohodnotením, ak je toto ohodnotenie menšie ako už sú ohodnotené. Takto pokračujeme cez celú mapu. Ak vchádzam do húštiny, tak k ohodnoteniu tohto vrcholu pridám hodnotu predchádzajúceho plus dva a zároveň si pamätám odkiaľ som k nemu prišiel. Ak vchádzam do cesty prirátavam jedna. Ale ak sa nachádza na tomto mieste skala, tak tento vrchol vôbec neberiem do úvahy.

Nižšie ukážem jednoduchú mapu a použitie Dijkstru. Červená farba značí už navštívený vrchol a zelená farba je vrchol, ktorý sa nachádza v halde. Na obrázku tri si vyberám cestu dole, lebo v halde je tento vrchol menší. Navštívenie si pamätám vo forme indexu v halde a tú nastavím na -1, vtedy viem, že vrchol už bol navštívený (namiesto ďalšieho atribútu *seen*), lebo na začiatku majú všetky vrcholy index 0 a v behu programu väčší ako 0.



Postup dijkstru v grafe – šíri sa ako vlna vody

Moja implementácia

V tejto časti mám definovanú štruktúru *EDGE*, ktora predstavuje cestu a obsahuje dĺžku cesty, čas cesty a samotnú cestu vo forme poľa súradníc.

Funkcie, ktoré som použil:

- *void relax(char **mapa, MIN_HEAP** heap, VERTEX** paNew, VERTEX** paTemp);* - táto funkcia mi zrelaxuje vrchol a vloží mi ho do haldy, prípadne ho updatne v halde
- *EDGE* createRoute(char **mapa, VERTEX* paVertex);* - funkcia vráti cestu, ktorú postaví z vrcholu *paVertex* na základe smerníkov na predošlé vrcholy. Tieto vrcholy sú ohodnotené vďaka dijkstrovému algoritmu.
- *void setMap(VERTEX*** mapOfV, char **mapa, int n, int m, int paStaY, int paStaX);* - dijkstrov algoritmus, ktorý vloží do haldy počiatočný vrchol a funguje dovtedy, pokým nie je prázdna halda – čiže ohodnotí celú mapu

Hľadanie najkratšej cesty

Ako posledná časť mi ostalo spojiť všetky tieto informácie a vystavať celkovú cestu. V poli si držím draka a princezné a každý tento vrchol má v poli najkratšie cesty ku ostatným, ktoré našiel dijsktra. Následne volám funkciu, ktorá mi generuje permutácie. Tieto permutácie znamenajú indexy v poli. Ak vygenerujem permutáciu čísel, tak volám funkciu *isPathBetter* a tá mi spočíta cestu na základe údajov, ktoré majú dané vrcholy k dispozícií. Ak je cesta kratšia pamätám si túto permutáciu čísel. Najlepšie to ukážem na príklade.

Majme 2 princezné a draka, ktorý je vždy na konci poľa dôležitých vrcholov a permutáciu čísel 213. A idem odzadu. Trojka predstavuje draka. Následne si z poľa vyberiem vrchol na pozícii jedna. Spojím cestu k drakovi a cestu od draka k číslu 1. A potom túto cestu spojím s cestou od vrcholu na indexe 1 do vrcholu na indexe 2. Keď už som takto pospájal cestu, vrátim ju a program je ukončený.

Funkcie, ktoré som použil:

- *EDGE* mergePaths(EDGE* paFirst, EDGE* paSecond);* - spojí dve hrany (cesty) do jednej
- *void isPathBetter(NODE** paPoints, EDGE* paBest, int* paArr, int* paN);* - zisťuje, či neexistuje lepšia cesta ako už mám
- *void generatePermutation(NODE** paPoints, EDGE* paBest, int* paArr, int* paN, int* nam);*
  - funkcia, ktorá generuje permutácie číselných indexov do poľa

Odhad zložitosti

Pamäťová zložitosť

V halde si vytváram veľkosť pre 1000 prvkov. Nezisťujem dynamicky veľkosť, lebo som neprišiel na vzorec ako. Mohol som použiť n*m veľkosť, ale je to podľa mňa zbytočné mrhanie. Tento prístup vytvára zbytočne veľkú haldu pre malé mapky, ale pre veľké mapy je to v poriadku. Čiže v prepočte ide o 28 kilobajtov.

Následne vytváram ešte maticu vrcholov pre dijkstru. Ak je ale vrchol skala, tak vrchol nevytváram – tým ušetrím miesto v pamäti. Z toho mi vychádza, že najhorší prípad je, keď budem mať samé cesty napríklad, tak veľkosť bude 280 kilobajtov.

Časová zložitosť

Najprv sa pozriem na zložitosť dijkstru. Každý vrchol musím vybrať z haldy a zároveň ohodnotiť jeho susedov, ktorí budú traja. Nech je počet vrcholov V. To znamená, že zložitosť bude ZD = V*logV + 3*logV. Z toho vyplýva najhoršia O(n log n).

Ďalej nech P je počet princezien. Pre každú princeznú a draka spustím jedného dijkstru. Z toho vychádza zložitosť P*ZD. Z toho vyplýva najhoršia O(n log n). Avšak problém nastáva v poslednej časti, kde permutujem a hľadám cesty. Zložitosť bude (P+1)!, čo je vlastne O(n!).

Celková najhoršia zložitosť bude teda O(n!).

Testovanie

Na testovanie som použil 10 testovacích vstupov, ktoré sú aj priložené. Okrem toho som použil aj veľa náhodne generovaných máp, ktoré som ale nepriložil, lebo je to zbytočné. Priložil som aj generátor máp (dá sa k nemu dostať cez voľbu 3). Zároveň som overil svoju implementáciu vo forme gracikého zobrazenia v simulátore: https://popolvar.surge.sh. V testovaní som sa zameral na tieto scenáre:

- **Minimálny vstup** – *"vstup1Minimal.txt"* – kde je hneď vedľa drak a hneď vedľa princezná. Cesta dĺžky bude tri. Na tomto vstupe som testoval najmenší možný vstup.
  Dĺžka cesty: 3
- **Jednočiarový vstup** – *"vstup2JednaCiara.txt"* – na vstupe máme 1D kráľovstvo, kde všetky princezné preskočí, zabije draka a potom ich postupne zachráni. Testujem, či nepoužívam greedy metódu na zachraňovaní princezien.
  Dĺžka cesty: 44
- **Nadstavba jednočiarového** – "*vstup3DveCiary.txt*" – podobný vstup, ale plus pod tým, mám jeden celý voľný riadok. Po tomto riadku by mal drak chodiť, aby obišiel húštiny.
- Dĺžka cesty: 42
- **Tri veľké náhodné mapy** – "*vstup4Velky.txt"* - Dĺžka cesty: 542
  "*vstup5Velky.txt"* - Dĺžka cesty: 478
  "*vstup6Velky.txt"* - Dĺžka cesty: 492
  -tieto vstupy majú prechod ku všetkým princeznám aj drakovi, žiadna komplikácia, len sú to veľké vstupy
- **Takmer prázdna mapa** – *"vstup7Prazdny.txt*" – mapa je prázdna s 5 princeznami a drakom, kde len jedna princezná je v záhrade. Na tomto vstupe testujem, či sa dokážem pohybovať po takmer prázdnej mape.
  Dĺžka cesty: 534
- **Len jedna princezná** – *"vstup8JednaP.txt"* – mapa je obyčajná, len sa na nej nachádza jeden drak a jedna princezná. Testujem, či mi funguje program aj na iný počet princezien ako 5.
  Dĺžka cesty: 286
- **Vstup, kde nie je prechod ku drakovi, ten je medzi skalami** – "*vstup9ZiadnaCestaDrak.txt".* Testujem, či mám ošetrené, ak neexistuje cesta k drakovi.
  Dĺžka cesty: 0
- **Vstup, kde nie je prechod ku jednej princeznej, tá je medzi skalami** – "*vstup10ZiadnaCestaPrincezna.txt".* Testujem, či mám ošetrené, ak neexistuje cesta k princeznej.
  Dĺžka cesty: 0
- **Vstup, kde je čas príliš krátky**– "*vstup11Nestihne.txt".* Testujem, či mám ošetrené, ak je príliš málo času na záchranu
  Dĺžka cesty: 0

Pri použití vášho testovača som naďabil na zle načítanie mapy. Preto som dorobil vo funkcii *zachran_princezne* otestovanie, či sa v mape nachádzajú validné znaky.