Algoritmi in podatkovne strukture 1

Primer izpita

Ime in priimek: _____

Vpisna številka: _____

Pri reševanju si lahko pomagate z enim A4 listom zapiskov.

Svoje odgovore pišite jasno in jih utemeljite.

Ocenjuje se odgovore na izpitni poli. Dodatni listi so namenjeni pomoči pri reševanju in jih ne oddate.

Čas reševanja: 90 min.

Naloga:	1	2	3	4	Skupaj
Točke:	25	25	25	25	100
Ocena:					

1. Pri izračunu konveksne ovojnice množice n točk smo obravnavali postopek, ki je določil, katere daljice med pari točk so del konveksne ovojnice, vendar so bile orientirane in naštete v poljubnem vrstnem redu. Sedaj želimo seznam d daljic krožno urediti tako, da bo prva točka naslednje daljice enaka drugi točki prejšnje. Seveda obstaja več takih ureditev, zato bo sprejemljiva katerakoli od njih. Daljice bodo podane s pari točk, točke pa so oštevilčene od 1 do n.

[10] (a) Opišite čim bolj učinkovit algoritem, ki izračuna tako krožno ureditev daljic. Navedite in utemeljite tudi njegovo časovno in prostorsko zahtevnost.

[15] (b) V C++ implementirajte funkcijo `ovojnica`, ki bo sprejela število točk in seznam daljic ter vrnila urejen seznam daljic v skladu z zgornjimi zahtevami.
Implementacija lahko ustreza manj učinkovitemu postopku od prej opisanega, vendar bo zato prejela manj točk (v tem primeru jo tudi na kratko opišite). Pri implementaciji lahko uporabljate vso funkcionalnost standardne knjižnice C++.

```
1 vector<PII> daljice = {{2, 5}, {7, 10}, {3, 5}, {3, 10}, {7, 2}};
2 vector<PII> krog = ovojnica(12, daljice);
3 // {{3, 10}, {10, 7}, {7, 2}, {2, 5}, {5, 3}}
```

Rešitev:

- Gotovo bo obstajala rešitev, ki se začne s prvo daljico v trenutni orientaciji, recimo x, y. Sedaj moramo izvesti še $d - 1$ korakov dodajanja naslednjih daljic. Iščemo daljico, ki vsebuje vozlišče y. Ker bo vsaka točka nastopala samo v dveh daljicah, si lahko za vsako vnaprej pripravimo seznam sosedov. Tako moramo na vsakem koraku preveriti samo dve možni naslednji točki, ena bo enaka prejšnji točki x, druga pa iskani naslednji točki z. Prostorska in časovna zahtevnost sta $O(n + d)$ - zgradimo seznam sosedov za n točk in jih nato obravnavamo $O(d)$.

- Implementacija

```
1  vector<PII> ovojnica(int n, vector<PII> daljice) {
2          vector<VI> adj(n+1);
3          for (auto [a,b] : daljice) {
4                  adj[a].push_back(b);
5                  adj[b].push_back(a);
6          }
7          auto [x,y] = daljice[0];
8          vector<PII> krog = {{x,y}};
9          for (int it=1;it<daljice.size();it++) {
10                 int z1=adj[y][0], z2=adj[y][1], z;
11                 if (z1!=x) z=z1; else z=z2;
12                 krog.push_back({y,z});
13                 x=y; y=z;
14         }
15         return krog;
16 }
```

2. Kajakaški center je obiskala skupina n turistov, ki se želijo udeležiti "spusta" po Ljubljanici. V centru imajo na voljo kajake, ki sprejmejo največ dva turista. Poleg tega imajo vsi kajaki enako nosilnost k, ki je ne sme preseči skupna teža kajakašev v njem. Za vsakega turista poznamo njegovo težo t_i. Kakšno je najmanjše število kajakov, ki jih potrebujejo za sočasen spust po Ljubljanici? Predpostavite lahko, da bo rešitev obstajala - najtežji turist ne presega nosilnosti kajaka.

   V kajakaškem centru so se odločili za sledeč algoritem. Turiste bodo postopoma dodeljevali v kajake in jih odstranjevali iz seznama nerazporejenih. Vsakič bodo najtežjemu nerazporejenemu turistu dodelili svoj kajak. Poleg tega mu bodo v kajak dodali še najlažjega nerazporejenega turista, če skupaj ne presegata kapacitete.

[10] (a) Kako bi čim bolj učinkovito implementirali opisani algoritem? Kakšni sta prostorska ter časovna zahtevnost in zakaj?

[15] (b) Dokažite, da opisani algoritem porabi minimalno število kajakov.

Rešitev:

- Turiste uredimo po teži. Obravnavali bomo vedno lažje turiste na eni strani in vedno težje turiste na drugi. Kandidate lahko torej vzdržujemo z dvema indeksoma, ki ju zmanjšujemo oz. povečujemo, dokler se ne srečata. Prostorska zahtevnost je linearna oz. konstantna, ker poleg vhodnih podatkov potrebujemo konstantno dodatno količino prostora, če uredimo seznam na mestu. Časovna zahtevnost je zaradi urejanja $O(n \log n)$, preostanek je linearen, ker bo skupaj $O(n)$ sprememb indeksov.
- Dokažemo s protislovjem in obravnavo različnih primerov. Ali je kdaj narobe, da najtežjega turista združimo z najlažjim, in bi drugačen par vodil do boljšega rezultata? Če najtežji turist ne more imeti poleg sebe niti najlažjega, mora biti sam. V nadaljevanju obravnavamo primere, ko je lahko v paru z najlažjim.
  - Recimo, da bi bilo bolje, če bi bil najtežji turist v kajaku sam. V tem primeru vemo, da lahko najlažjega prestavimo k njemu in tako ohranimo rešitev (če je imel najlažji svoj par) ali jo izboljšamo (če je bil najlažji v kajaku sam).
  - Recimo, da bi bilo bolje, če bi bil najtežji turist v paru z nekim drugim (ne najlažjim), recimo mu srednji. Tudi tu lahko zamenjamo najlažjega in srednjega. Oba sta namreč lahko v paru s katerimkoli drugim turistom, ker sta v predpostavljeni optimalni in v požrešni rešitvi v paru z najtežjim.

Vedno torej obstaja optimalna rešitev, ki se strinja s potezo opisanega algoritma.

3. Odgovorite na spodnja vprašanja o Dijkstrovem algoritmu.

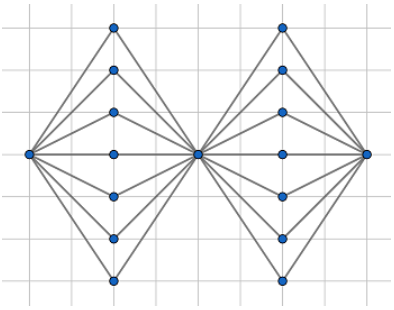
```
1  void Dijkstra_PQ(vector<VII> &adjw, int start,
2                   vector<int> &dist, vector<int> &prev) {
3      int n=adjw.size();
4      dist=vector<int>(n,-1); prev=vector<int>(n,-1);
5      priority_queue<PII, vector<PII>, greater<PII>> pq;
6      dist[start]=0; pq.push({0,start});
7      while (!pq.empty()) {
8          auto [d,x]=pq.top(); pq.pop();
9          if (dist[x]!=d) continue;  // ???
10         for (auto [y,w] : adjw[x]) {
11             int d=dist[x]+w;
12             if (dist[y]==-1 || d<dist[y]) {
13                 dist[y]=d; prev[y]=x; pq.push({d,y});
14             }
15         }
16     }
17 }
```

[10] (a) Na kratko opišite Dijkstrov algoritem - čemu je namenjen in kako deluje (opišite glavno idejo algoritma, ne podrobnosti implementacije).

[15] (b) Zgoraj je podan primer implementacije Dijkstrovega algoritma. Čemu je namenjena označena 9. vrstica? Kaj bi se zgodilo s pravilnostjo in časovno zahtevnostjo algoritma, če bi označeno vrstico izbrisali? Utemeljite, zakaj ni spremembe, ali podajte primer, kjer do nje pride in kakšna je.

Rešitev:

- Dijkstrov algoritem izračuna najkrajše poti iz izbranega vozlišča do vseh ostalih vozlišč v uteženem grafu brez negativnih uteži. Gradi drevo najkrajših poti z dodajanjem vozlišč od bližnjih proti bolj oddaljenim - vsakič doda najbližje vozlišče iz okolice že dodanih.
- Označena vrstica ignorira stare vrednosti, ki jih ne brišemo iz prioritetne vrste. Izbris ne vpliva na pravilnost, močno pa lahko vpliva na učinkovitost. Če ima neko vozlišče veliko vhodno in izhodno stopnjo (recimo v in i), je lahko v-krat dodano v prioritetno vrsto in zato vsakič obravnavamo i njegovih sosedov. V spodnjem primeru je časovna zahtevnost kar $O(n^2 \log n)$.



4. Izračunati želimo produkt matrik $A_1A_2 \dots A_n$. Pri tem je i-ta matrika velikosti $h_i \times w_i$. Da jih lahko množimo med seboj, morajo imeti kompatibilne velikosti, torej velja $w_i = h_{i+1}$. Produkt matrik je asociativna operacija. Vrstni red množenja matrik je nepomemben s stališča rezultata $(AB)C = A(BC)$. Pomemben pa je s stališča števila osnovnih operacij (množenj števil), ki so potrebne za izračun rezultata. Za izračun produkta matrik A in B, ki sta dimenzij $h_a \times w_a$ in $h_b \times w_b$ ($w_a = h_b = x$), potrebujemo $h_a x w_b$ množenj števil. Iščemo najbolj učinkovit način izračuna produkta matrik $A_1A_2 \dots A_n$, ki bo zahteval najmanj množenj števil.

[7] (a) Predstavite rekurzivno formulo za najmanjše število osnovnih operacij, ki so potrebne za izračun produkta matrik $A_1A_2 \dots A_n$.
Namig: razmislite o množenju matrik, ki bo izvedeno nazadnje.

[6] (b) Kako bi problem rešili z uporabo memoizacije?

[6] (c) Kakšni sta časovna ter prostorska zahtevnost takega postopka in zakaj?

[6] (d) Kako pa bi rešili problem s pristopom dinamičnega programiranja od spodaj navzgor?

Rešitev:

- Naj bo $f(i,j)$ iskano število osnovnih operacij za izračun $A_i \dots A_j$.
  $f(i,j) = min_{k:i \leq k < j} f(i,k) + f(k+1,j) + h_i w_i w_j$ in $f(i,i) = 0$
- Za vsak par i, j, ki definira podproblem $f(i,j)$, shranimo rezultat in ga kasneje uporabimo.
- Imamo n^2 podproblemov in za rešitev vsakega obravnavamo n možnosti. Prostorska zahtevnost je $O(n^2)$, časovna pa $O(n^3)$
- Od spodaj navzgor bi računali število operacij za produkte vedno daljših podseznamov. Za izračun produkta $A_i \dots A_j$ potrebujemo rešitve krajših produktov $A_i \dots A_k$ in $A_{k+1} \dots A_j$ ($i \leq k < j$).

```
for i=1..n
  for j=i+1..n
    f[i][j] = ...
```

Zgornji vrstni red ni ok, ker za $f(i,j)$ potrebujemo npr. $f(i+1,j)$, ki ga še nimamo. Pravilen bi bil spodnji vrstni red.

```
for len=2..n
  for i=1..n-len+1
    j = i+len-1
    f[i][j] = ...
```