Opisi algoritama

Zadatke, test primjere i rješenja pripremili: Frane Kurtović, Alen Rakipović, Gustav Matula, Ivan Katanić i Ante Đerek. Primjeri implementiranih rješenja su dani u priloženim izvornim kodovima koji nužno ne odgovaraju u svim detaljima ovdje opisanim algoritmima.

Zadatak: Soundex

Predložio: Ante Đerek
Potrebno znanje: `for` petlja, stringovi

Zadatak rješavamo direktnom implementacijom koraka Soundex algoritma opisanih u tekstu zadataka. Pritom, možemo koristiti već postojeće metode za manipulaciju stringova ili ih možemo sami brzo i efikasno implementirati. Primjerice, za brisanje svih pojavljivanja određenog slova iz stringa možemo unutar `for` petlje u novi string kopirati sve znakove osim onih koji odgovaraju slovu koje brišemo. Jedan trik kojim se uvelike može pojednostaviti i ubrzati implementacija je da pohranimo tablicu zamjene slova znakovima u konstantu i tako izbjegnemo pisanje velike količine `if` uvjeta. Konstanta može biti primjerice string u kojemu dolaze svi suglasnici koje zamijenjujemo, a neposredno nakon svakog od njih znamenka kojom ga zamijenjujemo.

Zadatak: Ograda

Predložio: Ante Đerek
Potrebno znanje: geometrija, `while` petlja, `for` petlja

Stvorimo za početak 1000×1000 binarnu matricu koja opisuje pašnjak. Odgovarajuća ćelija matrice ima vrijednost 1 ako je to polje dio tora, a 0 u suprotnom. Za svaki brid matrice sada možemo odlučiti je li on dio ograde. Svaki brid nalazi se između 2 polja. Ako je jedno od tih polja dio tora, a drugo nije, onda je taj brid dio ograde. Brid ograde možemo promatrati kao vezu između njegovih rubnih točaka. Na taj način načinili smo graf kojemu su vrhovi točke koordinatnog sustava koje čine ogradu. Taj graf je ciklus. Sve što nam preostaje je riješiti se uzastopnih paralelnih bridova. Najjednostavniji način za to napraviti je pohraniti ciklus kao niz točaka te za svake tri uzastopne provjeriti nalaze li se na istom horizontalnom ili vertikalnom pravcu, ako se nalaze tada srednju od tri točke brišemo. Tako sažeti ciklus ispisujemo počevši od najlijevije najdonje točke. Postoje dva načina takvog ispisa ciklusa, a od njih odabiremo onaj kod kojeg će prvi brid biti horizontalan.

Zadatak: Tornjevi

Predložio: Gustav Matula
Potrebno znanje: algoritmi skeniranja linijom, brzo sortiranje

Za početak pokušajmo prebrojiti zaštićena polja u stupcu x. Uvedimo oznake $\mathrm{minL}(x)$, $minR(x)$, $\mathrm{maxL}(x)$ i $\mathrm{maxR}(x)$ redom za redak najnižeg tornja lijevo od x, najnižeg tornja desno od x, najvišeg tornja lijevo od x, i najvišeg tornja desno od x. Tada su zaštićena polja u stupcu x upravo ona s retkom u intervalu $[\max(\mathrm{minL}(x), minR(x)) + 1, \min(\mathrm{maxL}(x), \mathrm{maxR}(x)) - 1]$ (ukoliko interval nije dobro definiran, tj. lijeva granica je veća od desne, nema zaštićenih polja).

Uz ovu opservaciju zadatak možemo riješiti stupac po stupac, s lijeva nadesno, tako da održavamo retke ekstremnih tornjeva lijevo i desno od trenutnog stupca (alternativno možemo za sve prefikse i sufikse niza tornjeva izračunati minimalni i maksimalni redak). Nije potrebno promatrati sve stupce, već samo one koji sadrže tornjeve, te njima susjedne, jer u intervalu stupaca koji ne sadrže tornjeve svi stupci imaju isti broj zaštićenih polja. Složenost je $O(N \log N)$, jer je tornjeve prvo potrebno sortirati.

Zadatak: Trigram

Predložio: Ante Đerek
Potrebno znanje: stringovi, for-petlja, skupovi

Za početak, moguće je relativno lagano napraviti rješenje koje isproba sve moguće podfraze zadanog teksta, za svaku od njih izračuna skup trigrama i njegovu sličnost sa skupom trigrama zadane fraze. Međutim, ovo rješenje je vremenske složenosti najmanje $O(n^3)$ te stoga nije dovoljno brzo da dobije sve bodove. Rješenje možemo ubrzati tako da u jednom prolazu izračunamo sličnost za sve moguće fraze koje počinju na određenoj startnoj poziciji počevši od kraćih prema dužima. Za traženu frazu najprije izračunamo skup A - skup svih trigrama koje sadrži. Skup trigrama možemo pamtiti kao trodimenzionalno polje veličine $27 \times 27 \times 27$ ili kao skup stringova (primjerice koristeći `set` u programskom jeziku C++). Za trenutnu podfrazu teksta ćemo održavati skup B - skup trigrama koji ona trenutno sadrži, kako dodajemo nove riječi na kraj fraze dobivamo nove trigrame koje dodajemo u skup B. Kako bi rješenje bilo dovoljno efikasno potrebno je održavati broj elemenata unije i broj elemenata presjeka skupova A i B. Pretpostavimo da dodajemo u B novi trigram t:

- Broj elemenata unije povećamo za 1 ako niti A niti B još ne sadrži t.
- Broj elemenata presjeka povećamo za 1 ako A već sadrži t, a B ga jošš ne sadrži.

Svaki puta kada dodamo novu riječ na kraj fraze sada jednostavno možemo izračunati sličnost sa traženom frazom i odabrati najsličniju frazu.

Zadatak: Ovce

Predložio: Ivan Katanić
Potrebno znanje: geometrija, algoritmi skeniranja linijom

Za rješavanje ovog zadatka potrebno je napraviti par opservacija. Prva od njih je da dva tora koja pokrivaju neka susjedna polja ili neka ista polja uvijek možemo spojiti u jedan tor koji će biti manjeg ili jednakog opsega nego početna dva zajedno. Druga je da će svaki tor biti pravokutnog oblika. Zašto? Uzmimo neki tor koji nije pravokutnog oblika i zamislimo najmanji pravokutnik koji ga obuhvaća. Taj pravokutnik će imati manji ili jednaki opseg nego početni tor, a obuhvaća sve što i on. Sada znamo da u optimalnom rješenju imamo jedan ili dva pravokutna tora koja se ne preklapaju. Prvi slučaj rješavamo tako da nađemo najmanji pravokutnik koji pokriva sve ovce (`https://en.wikipedia.org/wiki/Minimum_bounding_rectangle`). Drugi slučaj rješavamo uz opservaciju da je dva pravokutnika koja se ne preklapaju uvijek moguće odvojiti horizontalnom ili vertikalnom linijom. Sve parove pravokutnika koji su odvojivi vertikalnom linijom i obuhvaćaju sve ovce pronalazimo tako da pomičemo vertikalnu liniju s lijeva na desno i održavamo informacije o ovcama s lijeve odnosno desne strane linije. Informacije koje su nam potrebne su najdonja, najgornja, najlijevija te najdesnija ovca sa svake strane linije. One određuju najmanji pravokutnik koji obuhvaća ovce s jedne strane. Na sličan način rješavamo parove pravokutnika odvojene horizontalnom linijom.

Zadatak: Zatvor

Predložio: Gustav Matula
Potrebno znanje: binarno pretraživanje, algoritmi skeniranja linijom, brzo sortiranje

Prvo bi bilo korisno proći rješenje zadatka Tornjevi. Prvo primijetimo da, kako god izgledao skup zaštićenih polja, ako u njemu postoji kvadrat veličine $a > 1$, tada postoji i kvadrat veličine $a - 1$, pa rješenje možemo tražiti binarnim pretraživanjem. Preostaje efikasno napraviti provjeru za neku veličinu a. Dakle tražimo interval od a stupaca koji imaju interval od a redaka, tako da su polja u presjeku ta dva skupa zaštićena. Za neki interval stupaca radimo istu stvar kao za jedan stupac u zadatku Tornjevi. Ako su $\text{minL}, \text{minR}, \text{maxL}, \text{maxR}$ redom minimalni redak nekog tornjeva lijevo od intervala, minimalni redak tornjeva desno od intervala, itd. tada je interval redaka $[\max(\text{minL}, \text{minR}) + 1, \min(\text{maxL}, \text{maxR}) - 1]$

zaštićen za sve stupce u intervalu stupaca. Te minimume i maksimume možemo nakon sortiranja po stupcima izračunati za sve prefikse i sufikse niza tornjeva. Dakle tražimo interval od a stupaca koji ima interval zaštićenih redaka veličine a ili više. Po svim bitnim intervalima stupaca možemo proći s dva indeksa, od kojih jedan prati desnu granicu intervala, a drugi lijevu (koja je točno $a-1$ stupaca lijevo od desne). Za desne granice potrebno je promatrati samo stupce susjedne onima koji sadrže tornjeve, pa je složenost provjere linearna. Ukupna složenost je $O(N \log N)$ zbog sortiranja i binarnog pretraživanja s linearnom provjerom.

Zadatak: Meso

Predložio: Alen Rakipović
Potrebno znanje: pohlepni algoritam, graf

Cilj nam je da pronađemo jelovnik koji se sastoji od što više blagih jela pa možemo najprije razmatrati slučaj kada pripremimo samo blaga jela. Ovaj je jelovnik dobar za sve osim za one goste koji na svojoj listi nemaju niti jedno blago jelo. Kako svaki gost mora imati barem jedno jelo na listi, te najviše jedno ljuto jelo na listi to znači da je jelovnik dobar osim točno za one goste čija se lista sastoji od točno jednog ljutog jela. Za takve goste i nemamo druge opcije osim da pripremimo ta ljuta jela koja su jedina na njihovoj listi. Ovim razmišljanjem dolazimo do sljedećeg iterativnog pohlepnog algoritma:

1. Odaberi gosta A koji ima samo jedno ljuto jelo k na svojoj listi.
2. Pripremi jelo k u ljutoj varijanti.
3. Izbriši gosta A te izbriši obje varijante jela k sa listi svih ostalih gostiju.

Kada se dogodi slučaj da više ne postoji gost A sa točno jednim ljutim jelom na svojoj listi onda smo gotovi, ostatak rješenja se sastoji samo od blagih varijanta jela.

Zadatak: Budžet

Predložio: Ante Đerek
Potrebno znanje: dinamičko programiranje, problem ruksaka

Zadatak je na prvi pogled sličan problemu ruksaka (potrebno je iz skupa brojeva odabrati neku grupu čija je suma jednaka zadanom broju) te se rješava sličnim metodama dinamičkog programiranja. Razmatrajmo neki niz uzastopnih brojeva u ulazu $a_1, a_2, \ldots, a_n$. Ako vrijedi $a_1 = a_2 + a_3 + \ldots + a_n$ onda stavke $a_2, \ldots, a_n$ mogu biti direktne podstavke od a_1 te a_{n+1} može biti sljedeća stavka na istom nivou nakon stavke a_1. Mogući su i drugi scenariji u kojima je a_{n+1} sljedeća stavka na istom nivou nakon a_1 – primjerice ako je $a_2 = a_3$ te $a_1 = a_2 + a_4 + \ldots + a_n$. Cilj nam je da dinamičkim programiranje odredimo sve takve scenarije. Definirajmo $N(k)$ kao skup svih pozicija l takvih da je moguće da a_l bude sljedeća stavka na istom nivou nakon stavke a_k odnosno da postoji dobar budžet oblika

$$a_k, *a_{k+1}, * \ldots * a_{k+2}, \ldots, * \ldots * a_{l-1}.$$

Očigledno, uvijek je $k+1 \in N(k)$, a sve skupove $N(k)$ možemo izračunati dinamičkim programiranjem ako stavke procesuiramo od zadnje prema prvoj. Naime vrijedi da je $l \in N(K)$ ako je moguće naći niz $k_0, k_1, \ldots, k_m, k_{m+1}$ gdje je $k_0 = k+1$, $k_{m+1} = l$ i vrijedi $k_{i+1} \in N(k_i)$ te je suma $a_{k_0} + a_{k_1} + \ldots + a_{k_m}$ upravo jednaka a_k. Rješavanje ovog problema je ekvivalentno rješavanju problema ruksaka sa ograničenjem da nakon što odaberemo neki element a_t onda sljedeći odbrani element mora nužno biti iz skupa $N(t)$.

Zadatak: Stupci

Predložio: Gustav Matula
Potrebno znanje: dinamičko programiranje, segment tree/Fenwick tree

Pretpostavimo radi jednostavnosti da su sve visine različite. Za svaki stupac i izračunat ćemo dvije vrijednosti: L_i i R_i, maksimalni broj skokova ako od i-tog stupca krenemo lijevo, odnosno desno. Konačno rješenje za i-ti stupac je $\max(L_i, R_i)$. U početku sve vrijednosti L i R postavimo na nulu. Stupce obrađujemo od najvišeg prema najnižem. Za L_i nas zanima maksimalni broj skokova koje možemo napraviti ako krenemo desno od nekog stupca $j < i$, koji je viši od stupca i. Kako smo već obradili sve stupce više od i, a niz R je inicijaliziran s nulama, vidimo da $L_i = 1 + \max_{j<i} R_j$. Analogno $R_i = 1 + max_{j>i} L_j$. Traženje maksimuma možemo ubrzati tako da nizove L i R implementiramo nekom strukturom podataka kao npr. segment tree ili Fenwick tree.

Zadatak: Smjer

Predložio: Frane Kurtović
Potrebno znanje: najkraći put u grafu

Svaki par pozicija računamo zasebno. Jedan pristup je ispitati sve moguće slučajeve, tj. sve moguće međusobne odnose početne i završne točke, no dosta je teško biti siguran da su svi slučajevi pokriveni i lako se pogriješi u svakom pojedinom slučaju. Vrlo brzo postane jasno da uvijek postoji najkraći put koji se sastoji od svega nekoliko promjena smjera. Još jedna opservacija je da su bitne točke uvijek one kojima jedna koordinata odstupa najviše ± 1 od odgovarajuće početne ili završne koordinate. Intuitivno to ima smisla jer promjena koordinate za jedan mijenja smjer ceste, te je s time osigurano isprobavanje oba dva smjera, a plus i minus jedan osigurava da se isproba cesta prije i poslije završne točke. Ovo navodi na rješenje u kojem gradimo usmjereni graf u kojem su vrhovi točke u mreži, te tražimo najkraći put od čvora koji predstavlja početnu točku do završne točke. Težine bridova odgovaraju udaljenosti tih točaka u mreži. Za čvorove u grafu uzimamo svaki par koordinata iz skupova

$$\{x_1 - 1, x_1, x_1 + 1, x_2 - 1, x_2, x_2 + 1\} \text{ i } \{y_1 - 1, y_1, y_1 + 1, y_2 - 1, y_2, y_2 + 1\}.$$

Brid iz čvora A do čvora B postoji ako postoji cesta koja ima smjera od čvora A do čvora B. Najkraći put je najjednostavnije naći Floyd-Warshallovim algoritmom jer je broj čvorova u grafu jako mali.

Zadatak: Laseri

Predložio: Frane Kurtović
Potrebno znanje:

Neka niz R označava retke lijevih lasera, a niz S stupce gornjih lasera. Kažemo da je lijevi laser A *pobijedio* gornji laser B ako A prvi otopi polje $(R(A), S(B))$, tj. polje koje se nalazi na presjeku njihovih laserskih zraka. Prvi takav događaj je na polju $(R(0), S(0))$, te pobjeđuje lijevi laser ako je $R(0) > S(0)$. Pretpostavimo da je lijevi laser A pobijedio lasera B, što znači da će A otopiti to polje prije lasera B, te to polje neće predstavljati prepreku gornjem laseru B. Ključna opservacija je da su i svi laseri s većim retcima od lasera A u tom trenutku otopili polje u stupcu lasera B što znači da je za laser B moguće odmah izračunati vrijeme potrebno da otopi svoj stupac, i ono iznosi $N - len(R)$. Nakon toga uspoređujemo laser A s prvim laserom nakon B. Konačni algoritam izgleda tako da održavamo dva indeksa, A i B, koji označavaju koja dva lasera uspoređujemo trenutno. Za ta dva lasera uvijek vrijedi da su oni morali otopiti svako polje prije nego li su došli do polja $(R(A), S(B))$. Za onaj laser koji ne pobijedi znamo odmah izračunati rješenje jer on sigurno od tog polja na dalje neće otopiti niti jedno polje kroz kojeg prolazi laser, te povećavamo odgovarajući indeks za jedan. Situaciju kada oba dva lasera tope polje u isto vrijeme možemo promatrati kao da su oba dva lasera izgubila, osim što je potrebno dodati jednu sekundu u rješenje za oba dva lasera jer oba dva moraju otopiti zajedničko polje. Složenost ovog rješenja je $O(len(R) + len(S))$.

Zadatak: BST

Predložio: Ivan Katanić
Potrebno znanje: stabla, dinamičko programiranje na stablima

Zadatak od nas zapravo traži da ispremještamo brojeve u čvorovima tako da $in-order$ (`https://en.wikipedia.org/wiki/Tree_traversal#In-order`) obilazak stabla daje brojeve u sortiranom redosljedu. Iz toga znamo koji broj treba biti u kojem čvoru stabla na kraju premještanja. Promatrajmo jedan brid stabla, on dijeli stablo na dva dijela. Svaki broj koji će prijeći iz jednog dijela u drugi mora proći po tom bridu. Promotrimo neku fiksiranu vrijednost broja, x. Neka je $p(x)$ broj pojava broja x, prije svih premještanja, u jednom dijelu stabla, a $k(x)$ broj pojava broja x, nakon svih premještanja, u istom dijelu stabla. Broj x će preko našeg brida proći točno $|p(x) - k(x)|$ puta jer je to upravo razlika početne i završne frekvencije broja x u odabranom dijelu stabla. Primjetimo i da nije bitno koji od dva dijela stabla smo odabrali, razlika će biti ista, samo suprotnog predznaka. Dakle, ukupno rješenje zadatka je za svaki brid b i svaku vrijednost x brojeva iz stabla zbrojiti razliku početne i završne frekvencije broja x u jednom od dijelova na koje brid b dijeli stablo. Budući da je stablo ukorjenjeno prigodno je za svaki brid gledati donji dio, tj. podstablo ispod njega i tu računati razlike frekvencija. To je ekvivalentno gledanju podstabala svih čvorova stabla, osim korijena. Radi lakše implementacije možemo gledati i podstablo korijena (tj. cijelo stablo), zbroj razlika frekvencija tamo je ionako jednak nuli i ne utječe na ukupan rezultat. Razlike frekvencija računamo rekurzivno, za svaku vrijednost x, održavat ćemo $f(x) = p(x) - k(x)$, te sumu $|f(x)|$. Promatrajmo čvor i, s dva djeteta. Vrijednosti $f(x)$ za podstablo čvora i dobijemo tako da zbrojimo vrijednosti $f(x)$ njegove djece i k tome uzmemo u obzir sam čvor i, tj. početne i završne vrijednosti zapisane u njemu. Kako bi složenost ovog rješenja bila $O(NlogN)$ potrebno je u čvoru i "preuzeti" vrijednosti $f(x)$ od djeteta koje ima veće podstablo, a sve vrijednosti $f(x)$ od drugog djeteta koje su različite od nule procesirati jednu po jednu i na odgovarajući način uračunati u rezultat. Za detalje pogledajte priloženi kod.