Stack i queue zadaci

Problem 1. Dat je niz otvorenih i zatvorenih zagrada. Potrebno je utvrditi da li su zagrade dobro raspoređene i ispisati za svaku otvorenu zagradu gde se nalazi njena zatvorena zagrada. Primer dobro raspoređenih zagrada je (())(), dok u primeru ())(zagrade nisu dobro raspoređene.

Resenje. Primetimo da se prvo uparuju otvorene i zatvorene zagrade koje se nalaze na susednim mestima u početnom nizu. Zatim možemo izbrisati uparene zagrade i ponoviti postupak. Međutim ukoliko bi stalno prepravljali početni niz, vremenska složenost algoritma bi bila $O(n^2)$.

Pokušaćemo upotrebom steka da rešimo ovaj problem. Potrebno je primetiti da se zatvorene zagrada ispred koje ne postoji ni jedna zatvorena zagrada uparuje sa prvom zagradom ispred nje, te se te 2 zagrade mogu ukloniti iz početnog niza.

Kretaćemo se kroz početni niz od prvom karatkera ka poslednjem. Ukoliko je trenutno posmatrani karakter otvorena zagrada, na stek ćemo ubaciti poziciju zagrade. U slučaju da je trenutno posmatrani karakter zatvorena zagrada i ukoliko na steku postoji neki element, možemo upariti otvorenu zagradu koja se nalazi na vrhu steka i trenutno posmatranu zatvorenu zagradu. Ukoliko se desi da posmatrani karatker predstavlja zatvorenu zagradu i da je stek prazan, možemo zaključiti da zagrade nisu dobro raspoređene.

Problem 2. Dat vam je niz brojeva a dužine n. Vaš zadatak je da za svaki broj u nizu a nađete prvi element „levo“ od njega koji je veći od njega. Formalnije potrebno je da za svaki elemet niza a_i pronađete najveći broj $j < i$ tako da važi $a_j > a_i$, ukoliko takav broj ne postoji ispisati -1.

Rešenje. Ovaj problem se često pojavljuje kao potproblem nekih zadataka. Najjednostavije rešenje bi bilo da za svaki broj a_i prođemo kroz sve brojeve na pozicijama manjim od i i pronađemo traženi broj. Opisano rešenje ima algoritamsku složenost $O(n^2)$. Sada ćemo opisati algoritam koji koristi stek i ima algoritamsku složenost $O(n)$.

Posmatrajmo sledeću situaciju. Neka se prvi „levo“ veći broj od broja a_j nalazi na pozicji p, tj. neka važi $a_j \geq a_k$ za sve $p < k < j$. U posmatranoj situaciji, kada tražimo prvi veći broj od broja $a_i, i > j$ mi nemamo potrebe da proveravamo brojeve na pozicijama $\{p+1, p+2, \dots, j-1\}$ zbog činjenice da ukoliko broj na poziciji j nije veći od a_i onda sigurno nije ni jedan od brojeva na pozicijama $\{p+1, p+2, \dots, j-1\}$ veći od broja a_i zbog uslova da su svi oni manji od broja a_j. U slučaju kada je $a_j > a_i$ ponovo ne moramo da proveravamo brojeve na pomenutim pozicijama jer smo već našli broj veći od a_i.

Zbog ovog razmatranja nama je dovoljno da posmatramo brojeve pre pozicije i u opadajućem redosledu, tj. možemo da zanemarimo sve brojeve a_j za koje postoji broj $a_k \geq a_j, j < k < i$ („desno” od broja a_j postoji veći broj od a_j). Ovaj opadajući niz možemo da pamtimo uz pomoć steka. Pri pronalaženju rešenja za broj a_i sa steka ćemo izbacivati brojeve koji su manji od a_i, te ukoliko ostene

neki broj na steku, broj na vrhu steka je rešenje za broj a_i. Nakon izbacivanja iz steka svih manjih brojeva od a_i, ubacimo na stek broj a_i. Na ovaj način ćemo održavati stek u opadajućem redosledu, što nam je i potrebno.

Svaki element je jednom ubačen i jednom izbačen iz steka, tako da je vremenska složenost $O(n)$.

U zadatku se traži da se ispise pozicija prvog "levo" manjeg elementa, tako da ćemo na steku uz vrednost elementa čuvati i poziciju elementa u nizu.

Glavni deo programa bi mogao da izgleda ovako:

```
=================================================================
01     stack, lastElement
02     Init( stack, lastElement )
03     for i = 1 to n do
04            while not Empty( stack, lastElement ) and Seek( stack, lastElement
).value <= a[i] do
05                   Pop( stack, lastElement )
06            end while
07            if not Empty( stack, lastElement ) then
08                   print Seek( stack, lastElement ).position
09            else
10                   print "-1"
11            end if
12            Push( stack, lastElement, (a[i],i) )
13     end for
=================================================================
```

Problem 3. U prodavnicu ulazi n kupaca i za svakog kupca se zna koliko je minimalno vreme koje će se zadržati u prodavnici. Ova prodavnica je specifična po tome što kupac koji je prvi ušao u prodavnicu, mora prvi i da izađe, iako to znači da će ga drugi kupac morati čekati da završi svoju kupovinu, drugi mora da izađe pre trećeg, treći pre četvrtog, itd. Ukoliko znate za svaku osobu vreme ulaska u prodavnicu, kao i minimalno zadržavanje u prodavnici, potrebno je napraviti listu dešavanja u rastućem poretku po vremenu kada su se desila. Pod dešavanjima se smatra ulazak osobe u prodavnicu i izlazak osobe iz prodavnice. Jedan primer liste dešavanja:

Ulazak, osoba 1, vreme 5

Ulazak, osoba 2, vreme 10

Izlazak, osoba 1, vreme 11

Izlazak, osoba 2, vreme 2

Rešenje. Vidimo da se u skup ubacuju/izbacuju elementi kao kod queue. Da bismo koristili queue potrebno je prvo da sortiramo kupce prema vremenu ulaska u prodavnicu. Pri ubacivanju i-tog elementa u sortiranom nizu u queue, pogledaćemo sa početka queue koliko elemenata bi trebalo da izbacimo, tj. pre nego što kupac i uđe u prodavnicu, pogledaćemo koliko je njih izašlo u vremenu između ulaska osobe $i-1$ i osobe i. Kupci koji su napustili prodavnicu u tom intervalu se nalaze na početku queue. Posmatrajmo kupca j koji je na redu da izađe iz prodavnice. Znamo da je ušao u

prodavnicu u trenutku u_j, minimalno se zadržao t_j vremena i neka znamo da je kupac pre njega izašao u trenutku e_{j-1}. U slučaju da je $u_j + t_j < e_{j-1}$, možemo zaključiti da je kupac j morao da sačeka da kupac pre njega završi kupovinu, te je i on izašao isto u trenutku e_{j-1}. U suprotnom će važiti $e_j = u_j + t_j$. Pre nego što ubacimo osobu i u queue, potrebno je izbacivati osobe iz queue sve dokle važi $e_{prviNaQueue} < u_i$. Kada su sve osobe ušle u prodavnicu, nema više ubacivanja u queue, potrebno je isprazniti queue i paziti na vremena izlaska u odnosu na osobu koja je prethodno izašla, kao što je opisano.