dump-logo.jpg

Medijski pokrovitelji

Sponzori

Državno natjecanje / Osnovna škola (5. i 6. i 7. i 8. razred)   
Opis algoritama (Basic/Pascal/C/C++)

voditelj povjerenstva i izrade zadataka: Nikola Dmitrović, prof. mentor; ndmitrovic@infokup.hr



# Zadatak: Shuttle

U ovom zadatku treba za svaku znamenku analizirati koja lampa svjetli, a koja ne svijetli. Na osnovu analize tih slučajeva, naredbom odlučivanja se lako odredi o kojoj je znamenci riječ. Npr., naredbom odlučivanja provjeramo je li prikazana nula na sljedeći način:

## ako je ((prva=1) i (druga=1) i (treca=1) i (cetvrta=1) i (peta=1) i (sesta=1) i (sedma=0)) tada ispisi('0');

Slično radimo i za sve preostale znamenke.

## Potrebno znanje: naredba učitavanja, naredba odlučivanja

## Kategorija: ad hoc

# Zadatak: Sat

Sat se može riješiti na više načina. Zadani niz znamenki koji opisuje sate te zadani niz znamenki koji opisuje minute možemo učitati kao string te naredbom odlučivanja provjeriti sve kombinacije ovisno o tome svijetli li ili ne neka lampica. Drugi način je kreiranje cijelog broja koji odgovara broju sati (tako i za minute) provjeravajući koja lampica svijetli te neposredno zbrajajući odgovarajuće vrijednosti tih lampica. Zadatak možemo riješiti i uočavanjem da su vrijednosti pridružene pojedinoj lampici potencije broja dva. Opis ovog algoritma slijedi u nastavku:

učitaj(sat); učitaj (minuta);

koliko:=0; baza:=1;

dok je sat>0 radi

{

koliko:=koliko+baza\*(sat mod 10);

baza:=baza\*2;

sat:=sat div 10; // div je cjelobrojno djeljenje

};

ispiši(koliko);

koliko:=0; baza:=1;

dok je minuta>0 radi

{

koliko:=koliko+baza\*(minuta mod 10);

baza:=baza\*2;

minuta:=minuta div 10;

};

ispiši(koliko);

## Potrebno znanje: naredba učitavanja i ispisivanja, naredba ponavljanja

## Kategorija: ad hoc

# Zadatak: Poštar

Poštar se može podijeliti u dva dijela. Prvi dio zadatka je kada autubus doveze Jožu do kuće s kućnim brojem „1“. Tada se kvart kružnog oblika može promatrati kao ravna ulica u kojoj ima N kuća od kojim M moramo obići. Joža treba složiti obilazak zadanih kuća tako da broj paljenja motocikla bude minimalan. Ako razmislimo, zaključit ćemo da se optimalni obilazak postiže obilaženjem kuća u rastućem poretku kućnih brojeva. Da bi otkrili koliko je paljenja potrebno za obilazak zadanih kuća, trebamo provjeriti udaljenosti između susjeda te zabilježiti svaku udaljenost veću ili jednaku od tri!

učitaj(bus); učitaj (n); učitaj (m);

// *kvart* je struktura niz u koju učitavamo kućne brojeve koje želimo obići

za i:=1 do m radi učitaj (kvart[i]);

// algoritmom sortiranja po volji, sortiramo članove niza u rastućem poretku

sort(kvart);

min:=51; paljenje:=0;

ako je bus=1 tada // ako je autobus dovezao Jožu do kućnog broja „1“

{

// provjera udaljenosti kuće „1“ i kuće s najmanjim kućnim brojem

ako je kvart[1]-1>=3 tada inc(paljenje);

za j:=1 do m-1 radi // provjerimo udaljenost svih susjeda

ako je kvart[j+1]-kvart[j]>=3 tada

inc(paljenje);

ispiši('1 ',paljenje);

};

Drugi dio zadatka je kada autobus može Jožu dovesti ispred bilo koje kuće. Ideja slaganja optimalnog rasporeda je ista, kuće se obilaze u rastućem poretku kućnih brojeva. Da bi otkrili od koje kuće je najbolje krenuti u obilazak, potrebno je svaku kuću s popisa postaviti da je polazna i onda obići sve ostale kuće s popisa. Dodatna komplikacija je kada trebamo prelaziti s većih kućnih brojeva na manje. Ovo je moguće napraviti na više načina te ćemo samo opisati jedan školski primjer:

ako je bus=0 tada // ako je autobus dovezao Jožu do kućnog broja „0“

{

za i:=1 do m radi // svaku kuću s popisa postavimo za polaznu

{

paljenje:=0;//nužno je resetirati broj paljenja za svaki novi obilazak

// prvi dio obilaska, od kuće „kvart[i]“ do „kvart[m]“

za j:=i do m-1 radi

ako je kvart[j+1]-kvart[j]>=3 tada

inc(paljenje);

// drugi dio obilaska, od kuće „kvart[m]“ do „kvart[1]“. Pazi, ako je riječ o obilasku s početkom u kućnom broju „kvart[1]“ tada se u obilasku ne smijemo vratiti skroz do te kuće (zato i<>1)

ako je (i<>1) i (N-kvart[m]+kvart[1]>=3) tada

inc(paljenje);

//treći dio obilaska, od kuće „kvart[1]“ do „kvart[i-1]“.

za j:=1 do i-2 radi

ako je kvart[j+1]-kvart[j]>=3 tada

inc(paljenje);

ako je paljenje<min tada

{

min:=paljenje;

gdje:=i;

};

};

ispiši(kvart[gdje],' ',min);

};

## Potrebno znanje: naredba ponavljanja, struktura niz, algoritam sortiranja

## Kategorija: ad hoc

# Zadatak: Othello

Stanje i promjene na igraćoj ploči zapisujemo i pratimo u dvodimenzionalnom polju znakova. Za svaku od n pozicija na koju igrač na potezu postavi pločicu, treba u osam mogućih smjerova od te pozicije provjeriti je li moguće u nizu zarobiti pločice s oznakama igrača koji nije na potezu. Za one smjerove u kojima je moguće zarobljavanje, treba prebrojiti koliko je takvih pločica u nizu, taj broj dodati na globalnu varijablu te svaku od njih „okrenuti“.

// ploca je dvodimenzionalno polje (8x8) znakova. Prvo polje je na poziciji (1,1).

// dx i dy su konstantni nizovi za koje vrijedi:

// dx[]=(0,1,1,1,0,-1,-1,-1); dy[]=(-1,-1,0,1,1,1,0,-1)

inicijaliziraj(ploca,'.');

ploca[4,4]:='X'; ploca[4,5]:='O';

ploca[5,4]:='O'; ploca[5,5]:='X';

učitaj(n); // učitaj broj odigranih poteza

flip:=0; potez:='X'; nije\_na\_potezu:='O'; // inicijalizacija

za i:=1 do n radi

{

učitaj(r,s);

ploca[r,s]:=potez;

za f:=1 do 8 radi

obradi(r,s,dx[f],dy[f]); // u svih 8 smjerova napravi obradu

t:=potez; potez:=nije\_na\_potezu; nije\_na\_potezu:=t;

};

ispiši(flip);

Pisanje potprograma „obradi“ ovisi o programskom jeziku te ćemo ovdje dati samo ideju tog potprograma.

u\_nizu:=0; rcopy:=r; scopy:=s; // u\_nizu pratimo broj zarobljenih pločica

// „provjera“ je potprogram koji provjerava jesmo li izašli s ploče

dok je provjera(r+pomakx,s+pomaky) i ploca[r+pomakx,s+pomaky]=nije\_na\_potezu radi

{

inc(u\_nizu); r:=r+pomakx; s:=s+pomaky;

};

// ako niz pločica završava onom koja je na potezu tada je pronađeno zarobljavanje

ako je ploca[r+pomakx,s+pomaky]=potez tada

{

flip:=flip+u\_nizu; r:=rcopy; s:=scopy;

za i:=1 do u\_nizu radi

{

ploca[r+pomakx,s+pomaky]:=potez;

r:=r+pomakx; s:=s+pomaky;

};

};

## Potrebno znanje: naredba ponavljanja, dvodimenzionalno polje znakova

## Kategorija: ad hoc, simulacija

# Zadatak: Mujo

Motivacijski zadatak koji se može riješiti s tri for petlje ili preko matematičke formule koju izvedemo iz formule za zbroj prvih n prirodnih brojeva.

učitaj(P);

učitaj(D);

učitaj(T);

ispiši((P+D+T)\*(P+D+T+1));

## Potrebno znanje: naredba učitavanja i ispisa, osnovne računske operacije

## Kategorija: ad hoc

# Zadatak: Warp

Rješenje ovog zadatka je relativno jednostavno ali koderski nešto zahtjevnije. Svakako treba imati dugačak (ansi-)string za završno rješenje, u koji se na početku kopira prva riječ (koja se više ne koristi).

Za 50% bodova (varijantu s NE), najlakše je imati niz riječi - stringova, te jednako dugi niz oznaka iskorištenosti. U svakom koraku (dok ne potrošimo riječi), jednostavno tražimo najbolju (min/max) riječ po tri kriterija i pamtimo njen indeks: prvo po duljini preklapanja s trenutnim rješenjem, zatim po duljini (dulja), i na kraju leksikografski (manja). Može se prvo uzeti prva neiskorištena riječ kao trenutno najbolje rješenje, pa ostale uspoređivati s njom (standardno traženje minimuma/maksimuma). Ako su dvije riječi identične, uzmemo bilo koju jer ćemo ionako u sljedećem koraku uzeti onu drugu (to piše i u tekstu). Tako najbolju riječ preklopimo na kraj niza i označimo da je iskorištena.

Pri traženju preklapanja je bitno da probamo što više preklopiti (tako da se riječ na početku preklapa preko cijelog sufiksa, te je znak po znak mičemo udesno dok se ne preklopi). Tu su dobre funkcije poput str(n)cmp i str(n)cpy (c), odnosno copy (pascal). Važno je paziti da ne preklopimo riječ dulju od trenutnog rješenja izvan granica niza, kao i da probamo rubne slučajeve naglašene u tekstu (potpuno preklapanje i preklapanje duljine nula).

Za slučaj s reversanim riječima, najlakše je poduplati niz riječi tako da je prvih n normalno, a drugih n reversano. Zatim se isti postupak radi sa svih 2n riječi (prvu treba označiti kao iskorištenu, ili ako se izbaci, onda 2n-2), uz napomenu da riječi na poziciji x i n+x (x od 0 do n-1) ispituju istu zastavicu iskorištenosti (jer su zapravo ista riječ, samo reversana).

U prilogu možete pronaći kod službenog rješenja i jedno alternativno rješenje ako samostalno ne pronađete svoje točno rješenje.

## Potrebno znanje: string, algoritmi traženja minimuma/maksimuma, algoritam kreiranja okrenute riječi;

## Kategorija: ad hoc simulacija

# Zadatak: Hitori

Ovaj zadatak se rješava pomoću rekurzije. Treba isprobati sve dopuštene kombinacije brisanja znamenki, te pogledati koja kombinacija ima najmanji broj brisanja i istovremeno u svakom retku i stupcu nema „duplih“ znamenki.

Postoje razni načini za implementaciju ove rekurzije. Na primjer, možemo svakom polju tablice pridružiti jedan broj od 0 do N\*N-1: polja u prvom stupcu redom imaju brojeve 0, 1, ..., N-1, u drugom stupcu N, N+1, ..., 2\*N-1, itd. Ako je X neki broj, onda on odgovara polju u retku X mod N, te stupcu X div N. Nadalje, pretpostavimo da imamo NxN polje obrisani takvo da je obrisani[r, s] jednako 0 ako polje (r,s) nismo obrisali, a jednako 1 ako jesmo. Na početku postavimo sve elemente ovog polja na 0. Također, u varijabli najmanjeObrisanih pamtimo najmanji broj polja koje treba obrisati tako da vrijede uvjeti zadatka; na početku ju postavimo na N\*N. Slijedi pseudokod za rekurziju.

rek (zadnjeObrisanoPolje, brojObrisanih)

* Ako vrijedi:
  + dosad obrisana polja rezultiraju time da u recima i stupcima nema duplikata
  + brojObrisanih < najmanjeObrisanih

onda smo pronašli novo najbolje rješenje, pa postavi najmanjeObrisanih=brojObrisanih, te polje obrisani prekopiraj u konačno rješenje.

* for polje = zadnjeObrisanoPolje+1, ..., N\*N-1
  + r = polje mod N; s = polje div N;
  + ako obrisani[r-1, s] = 0 i obrisani[r, s-1] = 0, onda
    - označi polje (r, s) kao obrisano: postavi obrisani[r, s] = 1;
    - pozovi rekurziju: rek (polje, brojObrisanih+1);
    - odznači polje (r, s) kao obrisano: postavi obrisani[r, s] = 0;

Uočite da smo u gornjem pseudokodu prije označavanja nekog polja kao obrisanog provjerili da njemu susjedna polja nisu već obrisana. Tu je provjeru potrebno napraviti upravo tada, a ne tek na kraju (kada se odlučimo za sva obrisana polja), zbog toga što bi u protivnom program isprobavao previše kombinacija brisanja i bio bi prespor za slučaj N=6.

U prilogu možete pronaći kod službenog rješenja i jedno alternativno rješenje ako samostalno ne pronađete svoje točno rješenje.

## Potrebno znanje: dvodimenzionalno polje, poznavanje ideje rekurzivnog načina rješavanja problema.

## Kategorija: rekurzija