Konstrukcija algoritama indukcijom

Princip indukcije koristi se da se ulaz za problem svede na jedan ili više manjih. Ako se svođenje može uvek izvesti, a bazni slučaj se može rešiti, onda je algoritam definisan indukcijom, odnosno dobijen je rekurzivni algoritam.
Osnovna ideja je da se pažnja skoncentriše na smanjivanje problema, a ne na njegovo direktno rešavanje.
Miodrag Živković *ALGORITMI*

1. Zadato je $n + 1$ prirodnih brojeva $a_1, a_2, \ldots, a_n$ i K. Naći algoritam za ispitivanje da li postoji podniz od p indeksa (1<=p<=n) $1 <= i_1 < i_2 < \ldots < i_p <= n$ takvih da je $a_{i1} + a_{i2} + \ldots + a_{ip} = K$. Vremenska, odnosno prostorna složenost algoritma treba da budu redom $O(nK)$, odnosno $O(K)$.

Ideja algoritma je da se redom za svaki podskup $\{a[1], a[2], \ldots, a[i]\}$ ($1 <= i <=\cdot n$) skupa $\{a[1], a[2], \ldots, a[n]\}$ odredi da li u njemu postoji podskup takav da je zbir vrednosti njegovih elemenata jednak k (za sve $k \in \{0, \ldots, K\}$). Te informacije čuvamo u vektoru P od $K + 1$ elemenata (P[0..K]).
Koristimo činjenicu da u skupu $\{a[1], a[2], \ldots, a[i]\}$ postoji podskup takav da je zbir vrednosti njegovih elemenata jednak k akko u skupu $\{a[1], a[2], \ldots, a[i-1]\}$ postoji podskup takav da je zbir vrednosti njegovih elemenata jednak k ili $k - a[i]$. Dakle, pojačana induktivna hipoteza je da umemo da rešimo problem ne samo za sumu elemenata koja je jednaka K, već i za elemente čija je suma manja od K.

Zato u i-tom prolazu kroz petlju čuvamo podatke o skupu $\{a[1], a[2], \ldots, a[i-1]\}$ u vektoru Prev od $K + 1$ elemenata (Prev[0..K]).

Algoritam daje odgovor ”da” (postavlja promenjivu Postoji na true) ako nakon izlaska iz petlje P[K] ima vrednost true, tj. u skupu $\{a[1], a[2], \ldots, a[n]\}$ postoji podskup takav da je zbir vrednosti njegovih elemenata K. Inace daje odgovor ”ne” (postavlja promenljivu Postoji na false).

```
Algoritam  Ranac;  //C-like pseudo kôd
Ulaz: {a[1],...,a[n]}, K
Izlaz: Postoji (bool)
main ()
{
Prev[0] = true;
for (k=1; k<= K; k++) Prev[k] = false;
for (i=1; i<=n; i++)
{
  for (k=0; k<=K; k++)
   {
     P[k] = false;
      if (Prev[k] == true) P[k] = true;
       else
          if (k>=a[i])
          if ((Prev[k-a[i]]==true) P[k] = true;
   }
for (j=1; j<=K; j++) Prev[j] = P[j];
}

if (P[K] == true) Postoji = true;
else Postoji = false;

}
```

Primer tabele formirane pri rešavanju problema ranca.
Ulaz su $n = 4$ predmeta veličina 8; 5; 4 i 3, a veličina ranca je $K = 11$.
Simbol "*I*" označava da postoji rešenje koje obuhvata predmet iz odgovarajuće vrste; simbol "*O*" označava da postoji rešenje, ali samo bez tog predmeta; simbol "-" označava da ne postoji rešenje.

	0	1	2	3	4	5	6	7	8	9	10	11
$k_1 = 8$	O	-	-	-	-	-	-	-	I	-	-	-
$k_2 = 5$	O	-	-	-	-	I	-	-	O	-	-	-
$k_3 = 4$	O	-	-	-	I	O	-	-	O	I	-	-
$k_4 = 3$	O	-	-	I	O	O	-	I	I	O	-	I

Prev[0]=true =>P[0]=true
i=1: a[1]=8 => Prev[0]=true => P[8]=true => Prev[8]=true;
i=2: a[2]=5 => Prev[0]=true => P[5]=true => Prev[5]=true;
i=3: a[3]=4 => Prev[0]=true => P[4]=true => Prev[4]=true; => Prev[9]=true => P[9]=true
i=4: a[4]=3 => Prev[0]=true => P[3]=true => Prev[3]=true; =>Prev[7]=true; => Prev[11]=true => P[11]=true

U algoritmu se izvršava:
jedna operacija provere uslova, dve operacije dodele,
jedna *K*-tostruka petlja sa jednom operacijom dodele,
jedna *nK*-tostruka slozena petlja (tj. dvostruka petlja po *n* i po *K*) sa najviše
tri operacije dodele i dve operacije provere uslova u svakom prolazu kroz unutrasnju petlju, i jednom *K* +1-strukom petljom sa jednom operacijom dodele pri svakom prolazu kroz spoljasnju petlju.
Dakle, vremenska slozenost algoritma je:
$T(n) \cdot K + c1 + n(K + c2(K + 1)) = O(nK)$

Prostorna složenost algoritma je $O(K)$, jer koristi dva pomoćna vektora od $K + 1$ elemenata (P i Prev) i lokalne promenljive *i* i *k*.

2. Dat je skup intervala na pravoj, predstavljenih koordinatama krajeva. Konstruisati algoritam složenosti $O(n \log n)$ za nalaženje svih intervala koji se sadrže u nekom drugom intervalu iz skupa.

Rešenje

Svaki od datih intervala ćemo posmatrati kao jednu duž čiji levi i desni kraj odgovara levom i desnom kraju intervala.
Algoritam IntervaliPresek (parovi koordinata levih i desnih krajeva duži)
Ulaz: (l1,r1), ..., (ln,rn) /*parovi koordinata levih i desnih krajeva duži*/
Izlaz: (lk,rk)
/*parovi koordinata levih i desnih krajeva duži koje se sadrže u drugoj duži */
Ideja: duž nije sadržana u drugoj duži ako je levi kraj duži manji od
levog kraja druge duži
ili je desni kraj duži veći od desnog kraja druge duži.
POJAČANA IH: U skupu sa manje od n duži umemo da odredimo i označimo duži koje su sadržane u nekoj drugoj duži istog skupa i umemo da odredimo do tada najveći desni kraj

Opis algoritma:

1. sortiranje duži u rastućem poretku po njihovim levim krajevima
(ako dve duži imaju isti levi kraj, dodatno se sortiraju po desnim krajevima u opadajućem poretku)

2. u prvom koraku ne biva markirana prva duž (jer nije sadržana ni u jednoj duži, jer desni krajevi duži su u opadajućem poretku ako im je jednak levi kraj), a do tada najveći desni kraj duži je vrednost desnog kraja prve duži

3. pretpostavimo da je problem rešen za prvih n-1 duži (tako što su markirane sve duži koje su sadržane u nekoj drugoj duži i da je poznat do tada najveći desni kraj za tih n-1 duži)

4. Razmotrimo n-tu duž:

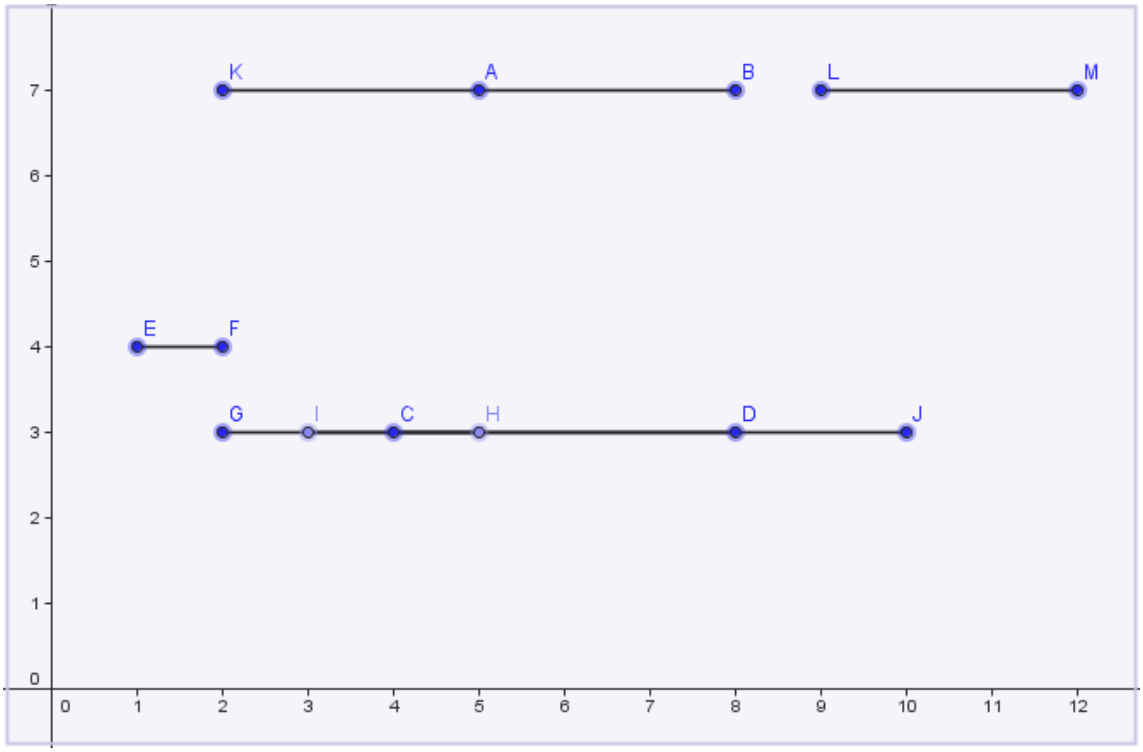
Ako je njen desni kraj veći od do tada najvećeg desnog kraja, onda ona nije sadržana ni u jednoj duži, te je ne markiramo i njen desni kraj stavimo za do tada najveći desni kraj.

U suprotnom, ta duž je sadržana u drugoj duži, te je markiramo.

(BSO (bez smanjenja opštosti) pretpostavimo da ne postoje duži čiji su i levi i desni krajevi jednaki).
Nakon primene algoritma markirane su duži koje su sadržane u drugoj.

Vremenska složenost algoritma je vremenska složenost koraka 1,2,3,4 tj. $O(n \log n) + O(1) + (n-1)*O(1) + O(1) = O(n \log n)$

3. *Prebrojavanje nesadrzanih duzi (za razmišljanje)*
Mali I je na papiru nacrtao n horizontalnih duži. Zatim je taj papir dao svom drugu J i zadao mu zadatak da prebroji sve duži. Međutim, ako se neke duži preklapaju, J neće primetiti da su to različite duži, već će ih posmatrati kao jednu duž. Dve duži se preklapaju ako imaju bar jednu zajedničku tačku. Koliko duži će J da vidi i prebroji? Ulazni podaci se učitavaju sa standardnog ulaza: prirodan broj n, broj duži. Potom se u n redova učitavaju po tri cela broja y_i , x_{ai} , x_{bi} koji označavaju da krajevi i-te duži imaju koordinate (x_{ai}, y_i) i (x_{bi}, y_i). Granice duži na ulazu nisu sortirane (x_a ne mora biti manje od x_b).



Rešenje:
U daljem delu analize rešenja pretpostavljamo da važi poredak $x_a < x_b$, tj. da smo ove vrednosti zamenili na samom ulazu ukoliko poredak nije bio dobar.
Radi lakšeg opisa algoritma, skup duži koje J vidi kao jednu **nazivamo grupom**. Prvo što možemo primetiti, a što se dalo i naslutiti, jeste da **se duži sa različitom y koordinatom nikada ne mogu naći u istoj grupi** odnosno preklopiti u nekoj tački. Dakle, za svako y problem možemo posmatrati nezavisno. Ovo je važna činjenica jer smo na osnovu nje problem sveli na duži koje se nalaze na "istoj visini".
Kako naći duži koje su na istoj visini? Trivijalno rešenje, koje naravno nije dobro, je da za svaku vrednost y koordinate tražimo duži na njoj.
Međutim, bolji pristup je da se **duži sortiraju po y koordinatama**. Ova ideja se sama nameće, jer na taj način dobijamo da se sve duži na istom nivou (ista y koordinata) nalaze jedna uz drugu. Konkretno, skup duži sa istog nivoa predstavlja podniz sortiranog niza.

Pažljivije analizirajmo sliku koja je ilustracija sledećeg test primera ulaza
7
7 5 8
3 8 4
4 1 2
3 5 2
3 3 10
7 2 5
7 12 9

Ako bi sortirali ove duži, dobili bi od ulaznog primera niz:
(3;2;8); (3;3;10); (3;4;8); (4;1;2); (7;2;5); (7;5;8); (7;9;12)

Uočimo:
duži sa koordinatom y = 3 su prve tri duži sortiranog niza
četvrta duž je jedina sa y = 4 koordinatom
duži sa koordinatom y =7 su peta, šesta i sedma duž sortiranog niza

Svaki od ovih skupova duži na istom nivou posmatramo posebno. Ovim smo dobili novi podproblem što i jeste cilj rešavanja problema induktivnim pristupom: za date duži na x-osi (odnosno istom nivou), naći broj duži koje J vidi

Dakle, sada možemo posmatrati duži sa krajevima $(a_1; b_1);(a_2; b_2)...(a_m; b_m)$.
Posmatrajmo duži $(a_i; b_i)$ i $(a_j; b_j)$ i pretpostavimo da one pripadaju ISTOJ grupi duži.
Ako se seku => njihova unija je duž koju one grade, tj. $(\min\{a_i, a_j\};\max\{b_i, b_j\})$.

Na primer, na nasoj slici su to duzi CD, GH.
Dakle, potrebno je naći ovaj podniz nadovezanih duži koje se preklapaju ili dodiruju.

Definiximo novi niz x i niz open na sledeći način:
$x[2k + 1] = a_k$, $open[2k + 1] = 1$
$x[2k] = b_k$, $open[2k] = -1$

Definišimo strukturu sa poljima (x,y,open).
Za svaku duž sa ulaza, u niz p ovih struktura ubacujemo $(xa_i, y_i, 1)$ i $(xb_i, y_{i,} 1)$. Zatim sortiramo ovaj niz najpre po y, a ako su jednake y koordinate, sortirajmo po x u neopadajući poredak, a za jednake i x i y po open (monotonost pri sortiranju po y nije bitna).
Nakon toga se krećemo po nizu i stalno inkrementiramo vrednost numOpen za vrednost open iz strukture i brojimo koliko puta uzima vrednost nula.

Glavni deo implementacije (bez učitavanja tj. inizijalizacije niza p) se može iskodirati:

```
int compare (const void * a, const void * b)
{
POINT A = *(POINT *)a;
POINT B = *(POINT *)b;
if ((A.y > B.y) || ((A.y == B.y) && (A.x > B.x)) || ((A.y == B.y) && (A.x == B.x) && (A.open < B.open)))
return 1;
return -1;
}
int solve()
{
int toReturn = 0;
qsort (p, 2 * n, sizeof(POINT), compare);
int numOpen = 0;
for (int i = 0; i < 2 * n; i++)
{
numOpen = numOpen + p [i].open;
if (numOpen == 0)
```

```
toReturn++;
}
return toReturn;
}
```

Vremenska složenost prikazanog algoritma je O(n log n), jer dominatan broj koraka potiče od sortiranja, dok nakon sortiranja potreban je samo jedan prolazak kroz niz. Memorijska složenost je linearna po n.

U odnosu na ovaj problem mogu se razmatrati i rešenja sledećih podproblema:
a) za date duži na Ox-osi izračunati ukupnu dužinu koje one prekrivaju
b) za date duži na Ox-osi naći tačku koja pripada najvećem broju duži
(rešenje je maksimalna vrednost koju uzima numOpen iz gore izloženog rešenja);

4. Neka je P konveksan poligon zadat nizom temena u cikličnom poretku. Opisati algoritam koji određuje da li je zadata tačka q u unutrašnjosti poligona P. Vreme izvršavanja u najgorem slučaju treba da bude $O(\log n)$.

Rešenje:
Proizvoljna dijagonala P_iP_j konveksnog poligona $P_1P_2 \ldots P_n$ deli njegovu unutrašnjost na tri dela.
1. Jedan deo je unutrašnjost konveksnog poligona, određenog temenima Pi, Pj i temenima polaznog poligona koja su sa jedne fiksirane strane prave $PiPj$ (poredak je isti kao u polaznom poligonu). Ovaj deo je ceo sa jedne strane prave $PiPj$.
2. Drugi deo se analogno odredjuje, i ceo je sa druge strane prave $PiPj$.
3. Treći deo je otvorena duž $PiPj$.

Primećujemo da je tačka u unutrašnjosti polaznog poligona *akko* je u jednom od ova tri disjunktna dela.

Neka je $T(n)$ vreme potrebno da se odredi da li tačka q pripada konveksnom poligonu P koji ima n temena.

Koristimo induktivni pristup, odnosno rešenje je rekurzivno.
Polazni slučaj je ako je $n = 3$ (uslov izlaska iz rekurzije): Označimo temena poligona (trougla) slovima A, B i C. Potrebno je ispitati da li se tačke A i q nalaze sa iste strane prave BC, zatim, da li se tačke B i q nalaze sa iste strane prave AC i ispitati da li se tačke C i q nalaze sa iste strane prave AB (što se može uraditi u konstantnom vremenu).

Induktivni korak:
Za $n > 3$, problem da li tačka q pripada poligonu $P_1P_2 \ldots P_n$, svešćemo na problem da li tačka q pripada poligonu $P_1P_2 \ldots P_k$ ili poligonu $P_kP_{k+1} \ldots P_nP_1$, gde je $k = [n/2] + 1$.
Odredimo k, zatim proverimo sa koje strane prave P_1Pk je tačka q. Za ovo nam je potrebno vreme c (konstantna veličina).

U zavisnosti od prethodnog rezultata znamo da li ćemo u sledećem koraku proveravati da li tačka q pripada poligonu $P_1P_2 \ldots P_k$ ili poligonu $P_k P_{k+1} \ldots P_nP_1$.
Dakle, problem se za $n > 3$ svodi na ispitivanje da li tačka pripada poligonu kome je broj temena
$[n/2] + 1$ ili $[n/2] + 2$ (u zavisnosti od parnosti broja n i položaja tačke q).

Zato asimptotski važi rekurentna relacija
$T(n) = T(n/2) + c$.
Na osnovu master teoreme, pošto je $a = 1$, $b = 2$, $k = 0$ sledi da je $T(n) = O(\log n)$, odnosno, vrednost T(n) se asimptotski ponaša kao $O(\log n)$.

Pri svođenju problema može se desiti da se tačka nalazi baš na pravoj odredjenoj dijagonalom kojom delimo poligon na dva dela. U tom slučaju algoritam završava rad, i daje odgovor ”da” ako se tačka nalazi na otvorenoj duži (dijagonali), a odgovor ”ne” ako se nalazi na ostatku prave.
Napomena: U ovom zadatku moramo voditi računa o strukturi podataka sa kojom radimo, tj. o načinu predstavljanja poligona. Jer, ako bismo pri svakom svođenju pravili novi niz temena, bilo pomeranjem elemenata, bilo kopiranjem u novi niz, vreme potrebno za svođenje ne bi bilo ograničeno konstantom, već bi bilo $O(n)$, što bi imalo uticaja na

ukupnu složenost algoritma. Zato ne diramo polazni niz temena, a tekući poligon predstavljamo preko dva indeksa, *L* i *R* , tj. poligon je *P*[*L*]*P*[*L* + 1] . . . *P*[*R*]). Prilikom svođenja samo menjamo ove indekse.

5. Neka je *G=(V,E)* stablo sa *n* čvorova. Cilj je formirati simetričnu kvadratnu matricu A reda *n*, čiji elemenat *(i,j)* je jednak rastojanju između čvorova v_i i v_j. Konstruisati algoritam složenosti $O(n^2)$ koji rešava ovaj problem ako je stablo zadato listom povezanosti.

matrica rastojanja A	...	i	...	N
...				
j		$\mathbf{d_S}(v_i,v_j)$		
...				
n				

Koristimo induktivni pristup:

- rešenje je jednostavno za stablo sa jednim ili dva čvora
- neka je pretpostavka da se zna rešiti problem za stablo sa *n-1* čvorova

POLAZNE PRETPOSTAVKE I OZNAKE:

Neka je dato stablo ***S*** sa ***n*** čvorova i neka ***v*** je list tog stabla i
neka ***w*** je jedini čvor susedan sa ***v*** (po definiciji lista u binarnom stablu, jasna je njegova egzistencija i jedinstvenost).
POSTUPAK
Uklanjanjem lista *v* iz *S* dobija se stablo S' sa ***n-1*** čvorova, za koje se u $O(n^2)$ koraka može izračunati matrica A svih rastojanja (sledi iz induktivne hipoteze).

1. Neka su *u, t* proizvoljni čvorovi različiti od *v*

Tada vrednost rastojanja između ta dva cvora ostaje nepromenjena u matrici *A* (jer $d_S(u,t)=d_{S'}(u,t)$).

2. Neka je *u* čvor različit *v*. Rastojanje čvora *u* od čvora *v* se određuje pomoću rastojanja od čvora *w*, tj.

$d_S(u,v)= d_{S'}(u,w) +1$, tj.
A[u,v]=A[u,w]+1

3. A[i,i]=0, jer je rastojanje čvora od samog sebe nula

Algoritam bi izgledao ovako: koren se označava sa v_1, i upisuje se nula u *A*[1, 1]. Zatim se stablo obilazi, npr. po dubini. Pređeni čvorovi redom dobijaju oznake $v_2, v_3 \ldots v_n$, i za svaki čvor se vrši gore opisani postupak:
kod prolaska kroz čvor v_i čvor se uz poznati indeks *p* neposrednog pretka, u *A*[*j, i*] i *A*[*i, j*] upisuje vrednost *A*[*j, p*]+1 za sve *j* od 1 do *i*−1. U *A*[*i, i*] upisuje se nula. Znači, u *i*−*tom* koraku novih 2*i*−1 elemenata matrice dobija traˇzenu vrednost.

Vreme izvršavanja opisanog algoritma za problem dimenzije n je T(n), tj.
$T(1) = c + c'$
T(n)=T(n-1) + (2n-1)*c1 , jer iz matrice stabla sa n-1 čvorom se za dodatnih (2n-1)*c1 koraka dolazi do matrice stabla sa n čvorova, gde vrednosti matrice za k-tu vrstu i k-tu kolonu se računaju za const*(2n-1) koraka (matrica A je dimenzije n*n)

Dakle, T(n)=T(n-1) + (2n-1)*c1 =T(n − 2) + (2n − 3)c1 + (2n − 1)c1=...= T(1) + c2*[2n+2(n-1)+ ... +1] - n*(c)=n(n+1)*c2 + n*c3=$O(n^2)$

6.

Svetsko prvenstvo vozača Formule 1 prošle godine imalo je dosta trka sa neizvesnim krajem i česte promene najboljeg vozača u ukupnom poretku. Zna se da je svetski prvak onaj vozač koji na kraju prvenstva (tj. nakon zadnje trke) ukupno ima **najviše bodova**. U Svetskom prvenstvu učestvuje **N** vozača. Na kraju svake trke (i na kraju poslednje), svi vozači dobijaju bodove. Pobednik trke dobija **N bodova**, drugoplasirani **N - 1 bod** i tako sve do poslednjeg u trci koji dobija **1 bod**. Dva vozača ne mogu zauzimati istu poziciju. Napišite program koji, na osnovu broja bodova za svakog vozača pre poslednje trke, ispisuje koliko vozača nakon poslednje trke može imati najveći

zbir bodova i tako postati novi svetskim prvak. Ako više vozača na kraju prvenstva ima isti najveći broj bodova, tada se svi oni proglašavaju svetskim prvakom.

ULAZNI PODACI

U prvom redu standardnog ulaza nalazi se prirodan broj **N** ($3 \leq \mathbf{N} \leq 300\,000$), broj vozača koji učestvuju u prvenstvu. U sledećih **N** redova nalazi se po jedan ceo broj **Bi** ($0 \leq \mathbf{Bi} \leq 2\,000\,000$, **i** = 1, ..., **N**), broj bodova i-tog vozača.

IZLAZNI PODACI

U jedini red standardnog izlaza ispišite traženi broj vozača.

Vremensko ogranicenje 1 s
Memorijsko ogranicenje 32MB

TEST PRIMERI

PRIMER 1	**PRIMER 2**
ulaz	**ulaz**
3	5
8	15
10	14
9	15
izlaz	12
3	14
	izlaz
	4

Rešenje

Sortirajmo vozače opadajuće po broju bodova. Uzmimo nekog **K**-tog vozača u tom sortiranom poretku i zapitajmo se, može li on biti svetski prvak?

Razmislimo o najpovoljnijoj mogućoj poslednjoj trci za K-tog vozača. On u toj trci osvaja **N** bodova, trenutni prvak osvaja samo 1 bod, trenutni viceprvak samo 2 boda i tako dalje.

Uopšte, vozač na poziciji **p** < **K** osvaja **p** bodova; vozači na pozicijama **p** > **K** nas i ne zanimaju jer su u ovom scenariju u ukupnom zbiru sigurno slabiji od **K**-tog.

Hoće li u takvoj situaciji **K** biti svetski prvak? Hoće ako ima veći zbir od svih ostalih, odnosno ako je
$\mathbf{a}[\mathbf{K}] + \mathbf{N} \geq \max\{\ \mathbf{a}[1] + 1, \mathbf{a}[2] + 2, \ldots, \mathbf{a}[\mathbf{K} - 1] + \mathbf{K} - 1\ \}$
gde je sa **a**[**p**] označen broj bodova **p**-tog vozača u opadajuće sortiranom nizu pre poslednje trke.

Rešenje koje ponovo računa gornji maksimum za svako **K** (nazovimo ga maks[**K**]) ima složenost O($\mathbf{N}^2$) i sporo je, s obzirom na vremensko ograničenje od 1s.

Ali, maksimum je lako računati "u hodu": svaki put kad povećamo posmatrano **K**, uporedimo dosadašnji maksimum s novim elementom i po potrebi ažuriramo maksimum, jer važi formula
maks[**K** + 1] = max{ maks[**K**], **a**[**K**] + **K** }.
Ovako se složenost računanja rešenja svodi na O(**N**). Ukupna je složenost ipak O(**N** log **N**), zbog sortiranja.

```
#include <cstdio>
#include <algorithm>
#define Nmax 300000
using namespace std;

int a[Nmax];

bool poredi(int x, int y) { return x > y; }

int main () {
        int n;
        scanf("%d", &n);
        for (int i = 0; i < n; ++i) scanf("%d", a+i);
        sort(a, a+n, poredi);
        int s = 0;
        int maks = 0;
```

```
        for (int i = 0; i < n; ++i) {
                s += (a[i] + n >= maks);
                maks = max(maks, a[i] + i + 1);
        }
        printf("%d\n", s);
}
```

7. Sa ulaza se zadaje niz prirodnih brojeva čiji se kraj označava nulom. Konstruisati algoritam koji brojeve datog niza smešta u binarno stablo, tako da je ono PB stablo u svakom trenutku primene algoritma. (Za stablo kažemo da je PB stablo ako je za svaki njegov čvor razlika broja čvorova levog i desnog podstabla najviše 1.)

Rešenje:
Da bi stablo bilo PB, moramo ga popunjavati nivo po nivo.
Prvi element ulaznog niza ide u koren, sledeća dva u prvi nivo, naredna četiri u drugi itd.
Svaki nivo popunjava se izvesnim redom (popunjavanje nivo po nivo nije dovoljan uslov za PB stablo).
Na *n*-tom nivou moguće je upisati 2^n elemenata.
Prvi element koji se upisuje na n-ti nivo oznacimo sa a^n_0, sledeći sa a^n_1 itd.
Neka je *b* binarna reprezentacija od *n* cifara broja *k*.
Pozicija elementa a^n_k u stablu određuje se na sledeći način: pošavši od korena stabla, na *i*-tom nivou skreće se levo ako je *i*-ta cifra najmanje težine broja *b* nula, a desno inače.

Odgovarajući algoritam (C-like pseudo kod):

```
main ()
{
   int broj, nivo=0, popunjeno=0;
   read (broj);
   while (broj!=0)
   {
       if (popunjeno==2^nivo)
       {
          nivo = nivo + 1;
         popunjeno = 0;
       }
      dodaj_u_drvo (broj, nivo, popunjeno);
      popunjeno = popunjeno + 1;
     read (broj);
     }
}

void dodaj_u_drvo (int broj, int nivo, int popunjeno)
{
    int brojac, pom=popunjeno;
     for(brojac=0; brojac<nivo; brojac++)
     {
        if (pom & 1) // Da li je bit najmanje tezine 1?
            skreni_desno; // Ako jeste, idemo desno...
        else
           skreni_levo; // Inace levo.
         pom = pom/2; // Delimo sa 2 sto odgovara shift-ovanju za jedno mesto u desno.
      }
      ubaci(broj,tekuce_mesto); // Pronasli smo mesto u drvetu i  tu upisujemo novi broj.
}
```

Napomena: U programu su korištene pseudo naredbe *skreni desno* i *skreni levo* koje, naravno, treba zameniti konkretnim blokom naredbi koje zavise od načina implementacije binarnog stabla.

Da li program radi korektno? Da.

Obrazloženje: Recimo da na $n - ti$ nivo želimo da ubacimo $k - ti$ element. Gledamo n cifara binarne reprezentacije broja k i od korena stabla se spuštamo na već opisan način. Kako cifara ima n spustićemo se na $n - ti$ nivo, tj. na nivo koji trenutno popunjavamo. Takodje i svi brojevi $0, \ldots, 2^n - 1$ imaju različitu binarnu reprezentaciju pa ćemo tako brojeve u stablo smeštati na različita mesta (u okviru jednog nivoa).
Zašto je stablo PB u svakom trenutku primene algoritma?
Čvor $a^n{}_k$ možemo poistovetiti sa b, binarnom reprezentacijom od n cifara broja k.
Posmatrajmo proizvoljan čvor b i njegova podstabla. Elementi njegovog levog podstabla su oblika $x0b$, gde je x neka niska binarnih cifara. Elementi desnog podstabla imaju oblik $y1b$.
Prvi element koji se upisuje ispod čvora b je $0b$, i to u levo podstablo. Zatim ide $1b$, u desno podstablo. Naredni elementi koji se upisuju ispod b su, redom, $00b$, $01b$, $10b$, $11b$, $000b$, $001b$ itd. Vidimo da se naizmenično upisuju u levo i desno podstablo. Uopšte, ako je $x0b$ poslednji upisani element ispod b, sledeći element koji se upisuje ispod b će biti $x1b$, jer predstavlja prvi broj veći od $x0b$, sa sufiksom b.
Slično, posle $x1b$ ispod b će se upisati $y0b$, gde su x i y niske jednake dužine, a y predstavlja broj za 1 ve´ci od broja predstavljenog sa x. Ovo poslednje ne važi jedino u slučaju upisivanja broja $11 \ldots 1b$. Tada će sledeći element biti $00 \ldots 0b$, imaće cifru više i ići na sledeći nivo.
Primetimo da se u svakom slučaju elementi koji ”prolaze” kroz čvor ubacuju naizmenično u levo i desno podstablo, te uslov PB-stabla nije narušen u tom čvoru ni u jednom koraku algoritma. Kako ovo važi za svaki čvor, to je stablo PB-stablo u svakom trenutku primene algoritma.

8. (glava 4.8) Nalaženje maksimalnog uzastopnog podniza
podsećanje: http://poincare.matf.bg.ac.rs/~jelenagr/pr/v2.htm

9. Neka je $x_1, x_2..., x_n$ niz realnih (ne obavezno pozitivnih) brojeva. Konstruisati algoritam složenosti $O(n)$ za određivanje podniza $x_i, x_{i+1},..., x_j$ sa najvećim mogućim proizvodom, $1 <= i <= j <= n$. Proizvod praznog podniza je po definiciji jednak 1.
ULAZ (niz 8 realnih brojeva)
2
3
-7
0.2
6
-3
2
99

IZLAZ
29937,6

Rešenje:

Rešenje je slično rešenju problema nalaženja podniza sa maksimalnim zbirom, pri čemu ovde treba dalje **pojačati induktivnu hipotezu** zbog promene znaka proizvoda elemenata posle množenja novododatim negativnim brojem.
Induktivna hipoteza: Pretpostavljamo da za niz $x_1, x_2, \ldots, x_m$, $m <= n$, znamo
podniz $x_i, x_{i+1},..., x_j$ sa maksimalnim proizvodom p_1
sufiks $x_r, x_{r+1},..., x_m$ sa maksimalnim proizvodom $p2$
sufiks $x_s, x_{s+1,...,} x_m$ sa minimalnim negativnim proizvodom $p3$.

Posle dodavanja broja x_{m+1}, nove vrednosti $p1$, $p2$, $p3$ su

$$(p_2, p_3) := \begin{cases} (x_{m+1}p_2, x_{m+1}p_3), & x_{m+1} > 0 \\ (1, -1), & x_{m+1} = 0 \\ (x_{m+1}p_3, x_{m+1}p_2), & x_{m+1} < 0 \end{cases}$$

$$p_1 := \max\{p_1, p_2\}.$$

Bez smanjenja opštosti može se pretpostaviti da su brojevi x_i različiti od nule: u protivnom nule razdvajaju niz na podnizove, za koje se problem rešava nezavisno, jer su svi proizvodi koji prelaze granicu nekog od ovih podnizova, jednaki nuli.

```
#include<stdio.h>
#include<stdlib.h>

int main()
{
float g_max, g_min, s_max, s_min, x[100], old_s_max;
int i,n;

g_max=1;    //globalni max proizvod podniza
s_max=1;     //max proizvoda sufiksa
s_min=1;      //min proizvod sufiksa
g_min=1;      //globalni min proizvod podniza

scanf("%d", &n);
for(i=1;i<=n;i++)scanf("%f",&x[i]);

printf("\n*****************************************************************");
printf("\nx[i]      |   s_max        g_max        s_min        g_min");

for (i=1;i<=n;i++)
 {
 if (x[i]==0)
  { s_max=1; s_min=1; }

 else
 if (x[i]>0)

  {
   if (x[i]*s_max<1)
     s_max=1;
    else
     {
      if (x[i]*s_max>g_max) g_max=x[i]*s_max;
      s_max=x[i]*s_max;
     }
   if (s_min<0)
    {
      if (x[i]*s_min<g_min) g_min=x[i]*s_min;
      s_min=x[i]*s_min;
    }
  }

 else
 if (x[i]<0)

   {
    old_s_max=s_max;

    if (x[i]*s_min<1) s_max=1;
     else
     {
      if (x[i]*s_min>g_max) g_max=x[i]*s_min;
      s_max=x[i]*s_min;
```

```
    }

   s_min=x[i]*old_s_max;
   if (g_min>0)
    g_min=old_s_max*x[i];
    else
     if (x[i]*old_s_max<g_min)
       g_min=x[i]*old_s_max;

  }

printf("\n%i. %6.2lf  |  %6.2lf       %6.2lf       %6.2lf       %6.2lf",i,x[i],s_max,g_max,s_min,g_min);
}
printf("\n*****************************************************************\n");
return 0;
}
```