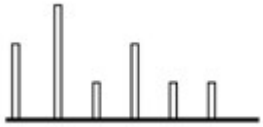
Zadaci sa rešenjima Republičko takmičenje 2006.

zadatak: Domine

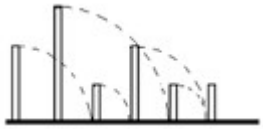
U našem malom gradu Abenišbeu ljudi imaju različite navike i običaje, od ostatka države. Između ostalog, kod njih se domine igraju sa velikim brojem domina različitih veličina. Na nekim dominama se nalazi i do 200 brojeva poređanih u red, a na nekim i samo jedan broj. Jovica i Perica su nabavili ovakve čudne domine, ali uputstvo nije bilo u kutiji. A pošto oni nisu iz Abenišbea, ne znaju kako se igra sa dominama različitih veličina. Zato su odličili da prave nizove domina koji se ruše kad se gurne samo jedna domina. Posle nekoliko dana slaganja, složili su dugačak niz domina, sa jednakim rastojanjem između domina, ali ne znaju koliko će domina da padne ako se gurne prva. A pošto su se dosta namučili da slože, ne žele da isprobaju i izbroje domine koje su pale. Zato vi treba da im pomognete i da izračunate broj domina koje će pasti.



Ulaz:
U prvom redu ulazne datoteke **domine.in** nalazi se prirodan broj n ($1 \leq n \leq 100000$) koji predstavlja broj domina na stolu. U sledećih n redova se nalazi visina svake domine ($1 \leq h \leq 200$). Rastojanje između susednih domina je 1. Da bi jedna domina oborila drugu, rastojanje između njih mora biti (strogo) manje od visine te domine.
Izlaz:
U izlaznu datoteku **domine.out** ispisati broj domina koje će pasti ako se gurne prva domina.
Primer:

domine.in	domine.out
6 2 3 1 2 1 1	5

Objašnjenje:



Prva domina ruši drugu, druga treću i četvrtu, četvrta petu, a šesta domina ostaje da stoji.

Rešenje

Domine obrađujemo redom, pri čemu stalno pratimo koliko domina ruše do sada obrađene domine. Domina na poziciji i visine h ruši sve domine do i+h-1 (uključujući i nju). Složenost algoritma je O(n).
Pseudokod

```
i = 1;
pada = 1;
while (pada < n) && (i ≤ pada)
         if ( h[i]+i-1 > pada )
            pada = h[i]+i-1;
         i = i + 1;
if (pada>n) pada = n;
write(pada);
```

fajl: domine.cpp

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

```
int main()
{
    freopen("domine.in","r",stdin);
    freopen("domine.out","w",stdout);
    int i,pada,n,h;
    scanf("%d",&n);
    i = 1;
    pada = 1;
    while ((pada<n) && (i<=pada))
    {
          scanf("%d",&h);
        if ( h+i-1 > pada )
           pada = h+i-1;
        i++;
    }
    if (pada>n) pada=n;
    printf("%d\n",pada);
  return 0;
}
```

zadatak: Mornar Đura

Mornar Đura je veliki avanturista i nakon završene karijere, odlučio je da krene na put oko sveta. Krenuo je iz svog rodnog mesta, obišao veliki broj gradova i na kraju se vratio kući. Pošto je Đura čovek u penziji, počeo je da zaboravlja. Želeo je da rekonstruiše redosled gradova koje je posećivao, međutim, sve što mu je ostalo su bile autobuske, avionske i karte za brodove kojima se prevozio. Pomozite Đuri da rekonstruiše svoje putešestvije koje će prepričavati unucima.
Na svakoj karti se nalazi ime kompanije koja prevozi, ime polazišta i odredišta. Poznato je da Đura, avanturista u duši, nikada ne posećuje isti grad dva puta (ako to nije njegov rodni grad, naravno).
Ulaz:
U prvom redu tekstualne datoteke **mornar.in** se nalazi broj karata koje je Đura prikupio posle putovanja ($N \leq 20000$), a nakon toga, posle razmaka, sledi ime Đurinog rodnog grada. U narednih *N* redova sledi opis svake od karata. Svaki red je zapisan na sledeći način:
Ime1:Ime2->Ime3
Ime1 je ime kompanije, *Ime2* je ime polaznog grada, dok je *Ime3* destinacija. Svako ime (uključujući i ime rodnog grada) se sastoji isključivo od slova engleskog alfabeta. Prvo slovo imena je veliko, dok su ostala mala. Redosled pojavljivanja karata u ulazu ne mora biti isti kao i redosled kojim je Đura putovao.
Izlaz:
U prvom i jedinom redu izlazne datoteke **mornar.out** ispisati N+1 ime, tako što će prvo i poslednje ime biti ime Đurinog rodnog grada. Između svaka dva imena obavezno staviti ,,->''. Ne treba dodavati razmake, niti druge znakove.
Primer:

mornar.in	**mornar.out**
5 Nis Jato:Beograd->Nis Soko:Cuprija->Jagodina Espreso:Nis->Aleksinac Simpleks:Jagodina->Beograd Raketla:Aleksinac->Cuprija	

Rešenje

Najpre ćemo sortirati karte po leksikografskom uređenju mesta polazišta. Posle toga, krećemo od Đurinog rodnog grada i ponavljamo postupak kojim tražimo sledeći grad, sve dok se ne vratimo u rodni grad. Da bismo našli koja je naredna destinacija, dovoljno je da u skupu karata pronađemo kartu u kojoj je tekući grad polazište. Pošto su nam karte sortirane prema mestu polaska, to ćemo najbrže pronaći binarnom pretragom.
Pseudokod

```
SORT-KARTE
current = NEXT(start)                                   //start je rodni grad
```

```
OUTPUT(start, "->")
WHILE start != current
         OUTPUT(current, "->")
         current = NEXT(current)
OUTPUT(start)
```

Složenost ovog algoritma je O(Nlog(N)). To je složenost sortiranja. Osim toga, program će proći kroz petlju N puta, dok u svakom prolasku kroz petlju izvrši jednu binarnu pretragu čija je složenost logaritamskog reda.

fajl: mornar.cpp

```
#include <stdlib.h>
#include <cstdlib>
#include <fstream>


using namespace std;

#define MAXN 20005
#define MAXL 20


typedef struct{
        char s[MAXL];
        char d[MAXL];
} karta;

int n;
karta t[MAXN];

// ucitavanje imena
// Najpre se propustaju svi karakteri dok se ne dodje do velikog slova, a potom se ucita
ime
void ucitaj(char* ss, ifstream &in){
      char c = 'a';
      int i = 0;
      while (c < 'A' || c > 'Z') in >> c;
      do {
            ss[i++] = c;
            in >> c;
            if (in.eof()) break;
      } while (c >= 'a' && c <= 'z');
      in.putback(c);
      ss[i] = 0;
}

// funkcija koju koristi quick sort za poredjenje
int uporedi_karte(const void* k1, const void* k2){
    karta* a = (karta*) k1;
    karta* b = (karta*) k2;
    return strcmp(a->s, b->s);
}

// pronalazi indeks karte sa datim polaznim mestom
// vrsi se binarna pretraga
int pronadji(char* s){
    int l = 0, d = n, m;
    while (1){
          if (l == d) return l;
          m = (l + d)/2;
          int x = strcmp(t[m].s,s);
          if (x > 0) d = m-1;
          else if (x < 0) l = m+1;
          else return m;
    }
```

```
}

int main(int argc, char *argv[])
{
    ifstream in("mornar.in");
    ofstream out("mornar.out");

    char start[MAXL], pom[MAXL];
    int i;
    in >> n;
    ucitaj(start, in);
    for (i = 0; i < n; i++){
        ucitaj(pom, in);
        ucitaj(t[i].s, in);
        ucitaj(t[i].d, in);
    }
    qsort(t, n, sizeof(karta), uporedi_karte);
    char *s;
    s = start;
    do{
       out << s << "->";
       int i = pronadji(s);
       s = t[i].d;
    } while  (strcmp((char*)start, s));
    out << start << endl;
    in.close();
    out.close();
    return EXIT_SUCCESS;
}
```

zadatak: Autoput

Vlada zemlje Bajtovije je odlučila da poboljša infrastrukturu puteva izgradnjom novog autoputa. Autoput ima oblik prave linije, koja je paralelna sa *Y* osom. Dužina puta može biti proizvoljno velika. Cilj izgradnje novog puta je da on bude dostupan što većem broju građana te zemlje. Poznato je da će žitelji određenog grada koristiti autoput ako najkraće rastojanje od njihovog grada do tog puta nije veće od graničnog rastojanja *D*.

Ulaz:

U prvom redu datoteke **autoput.in** se nalazi broj gradova u Bajtoviji, $N \leq 50000$ i broj *D*, koji su razdvojeni razmakom. U narednih *N* redova sledi opis svakog grada koji čine tri broja razdvojena razmakom. Najpre su upisane koordinate *X* i *Y*, a potom sledi broj žitelja tog grada. Svi brojevi u datoteci su prirodni i manji od 100000.

Izlaz:

U prvom i jedinom redu izlazne datoteke **autoput.out** ispisati maksimalan broj stanovnika Bajtovije koji će koristiti autoput. Garantuje se da taj broj neće preći dve milijarde.

Primer:

autoput.in	**autoput.out**
5 2 11 9 1000 2 5 2000 7 5 10000 5 8 1000 9 2 5000	16000

Objašnjenje:

Autoput će biti prava *X*=7, tako da će biti dostupan trećem, četvrtom i petom gradu iz ulaza.

Rešenje

Najpre ćemo sortirati gradove prema X koordinatama. Zatim, krenuvši sa leve na desnu stranu, vodimo računa o intervalu koji je manji ili jednak 2D. Kako novi grad ,,uđe'' u interval, tako trenutnoj sumi dodajemo broj stanovnika grada, a kada grad više nije u intervalu, onda smanjimo sumu za njegov broj stanovnika. U svakom trenutku proveravamo da li trenutna suma prevazilazi maksimalnu sumu.

Pseudokod

```
SORT-GRADOVI
l = 0
max = 0
FOR d = 0 to n-1
         WHILE x[d] - x[l] > 2D
                  l++
                  sum = sum - s[l]
         sum += s[d]
         IF sum > max THEN
                  max = sum
RETURN max
```

Složenost ovog algoritma je O(Nlog(N)). To je složenost sortiranja. Složenost petlje je linearna zbog toga što ukupan broj iteracija kroz WHILE petlju ne može da pređe N.

fajl: autoput.cpp

```
#include <cstdlib>
#include <fstream>
#include <stdlib.h>

using namespace std;

#define MAX 50005

// x i y su koordinate, a z je broj stanovnika
typedef struct{
        int x, y, z;
} city;


// koristi se za uporedjivanje gradova po x koordinati
// zbog quick sorta
int up(const void *px, const void* py){
    city* a = (city*) px;
    city* b = (city*) py;
    if (a->x < b->x) return -1;
    if (a->x > b->x) return 1;
    return 0;
}

int main(int argc, char *argv[])
{

    ifstream in("autoput.in");
    ofstream out("autoput.out");
    int x, y, z, n, d, i;
    int max = 0, sum = 0, l, r;
    city gradovi[MAX];

    in >> n >> d;
    for (i = 0; i < n; i++){
        city c;
        in >> c.x >> c.y >> c.z;
        gradovi[i] = c;
    }
    qsort(gradovi, n, sizeof(city), up);

    l = 0; max = gradovi[0].z; sum = gradovi[0].z;
    for (r = 1; r < n; r++){
        while (gradovi[r].x-gradovi[l].x>2*d)
              sum -= gradovi[l++].z;
        sum += gradovi[r].z;
        max >?= sum;
    }
```

```
    out << max << endl;
    in.close();
    out.close();
    return EXIT_SUCCESS;
}
```

zadatak: Dalekovodi

U Bajtoviji Đurica je odlučio da uvede struju. Spojio je n kuća pomoću m dalekovoda različitog kvaliteta. Za svaki dalekovod se zna posle koliko sati neodržavanja zarđa i prestaje da provodi struju. Ispostavilo se da bi, kad bi pustio mrežu u rad, došlo do kratkog spoja i velike katastrofe. Zato je Đurica uposlio Dragančeta da ispita mrežu. Draganče je ispostavio Đurici izveštaj, u kome se nalazi spisak k parova kuća koje ne smeju međusobno da budu spojene (ni direktno dalekovodom, ni preko drugih kuća) da bi mreža smela da se pusti u rad. Đurica može da bira koje dalekovode će da nastavi da održava, a koje ne. Đurica vas je uposlio da odredite koliko najmanje sati mora da prođe da bi mogao da pusti mrežu u rad, a Dragančeta za teži deo posla, da ispita koje dalekovode da održava, a koje ne, na osnovu vašeg izveštaja.

Ulaz:

U prvom redu ulazne datoteke **struja.in** nalazi se prirodni brojevi n, m i k ($1 \leq n$, $k \leq 10000$, $1 \leq m \leq 100\,000$). U sledećih m redova nalaze se po tri broja i, j, kv koja označavaju da su kuće i i j spojene dalekovodom kvaliteta kv ($1 \leq kv \leq 1\,000\,000\,000$, posle kv sati prestaje da propušta struju ako se ne održava). Zatim se u preostalih k redova nalazi po dva broja i i j, koja označavaju da kuće i i j ne smeju biti spojene, da bi mreža smela da se pusti u rad. Nijedan par gradova nije spojen direktno više od jednim dalekovodom, ni 2 puta naveden na Dragančetovom spisku.

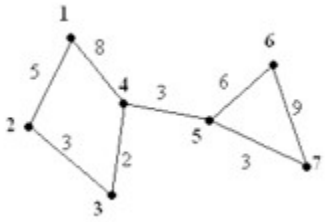
Izlaz:

U izlaznu datoteku **struja.out** upisati minimalan broj sati s koji mora da prođe da bi mreža smela da se pusti u rad.

Primer:

struja.in	**struja.out**
7 8 3 1 2 5 1 4 8 3 2 3 4 3 2 4 5 3 5 6 6 5 7 3 6 7 9 2 5 1 3 5 7	6

Objašnjenje:



Opis dalekovoda iz ulaza odgovara dalekovodu prikazanom na slici iznad. Moguće je osposobiti mrežu za 6 sati, ako prestanu da se održavaju dalekovodi 1-2, 3-4, 5-6, 5-7

Rešenje

Prvo treba primetiti da je zadatak moguće rešiti tako što se ne održava nijedan dalekovod, i da je potrebno izračunati posle koliko sati dati parovi kuća neće biti spojeni. Takođe treba primetiti da je potrebno posmatrati samo one sate u kojima bar jedan od dalekovoda prestaje da propušta struju. Proveru da li je posle nekog konkretnog sata moguće pustiti mrežu u rad, je moguće uraditi pomoću nalaženja komponetni

povezanosti (pomoću DFS-a ili BFS-a), i provere da li neki par kuća, koje ne smeju biti povezane, pripada istoj komponenti povezanosti. A pronalaženje odgovarajućeg broja sati je moguće binarnom pretragom, po svim mogućim satima.
Složenost binarne pretrage je O(log m), složenost nalaženja komponenti povezanosti O(m + n) (ako se graf čuva preko lista povezanosti), i provera parova O(k), pa je složenost algoritma O ((m + n + k) * log m).

Pseudokod

```
// sat - sortiran niz svih satova u kojima se kvari neki dalekovod
// sat[l] - najmanji, sat[d] - najveci
while (d-l>1)
{
    s=(l+d)/2;

    if (moze(sat[s])
       d=s;
    else
       l=s;
}
if (moze(sat[l]))
         d=l;

write(sat[d])


dfs(int i,int komp,int kv)
{
    ozn[i]=komp;
    int j;
    (za sve cvorove j povezane sa cvorom i granom veceg kvaliteta od kv)
       if (ozn[j]==0)
          dfs(j,komp,kv);
}
int moze(int s)
{
    (postavi sve ozn na 0)
    int i,komp=1;
    (za sve cvorove i)
       if (ozn[i]==0)
          dfs(i,komp,s);
    komp++;

    int ind=1;
    (za sve parove kuca kuca1 i kuca2, dok je ind==1)
         ind=(ozn[kuca1]!=ozn[kuca2]);

    return ind;
}
```

fajl: struja.cpp

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

const int maxn=41000;
const int maxm=110000;

typedef struct{
  int p,q,kv;
} grana;

int kuca1[maxm],kuca2[maxm],sat[maxm],poc[maxn],ozn[maxn];
grana pov[2*maxm];
int n,m,k;

int upG(const void *px,const void *py)
```

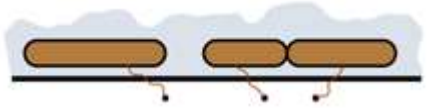
```
{
    grana *x=(grana *)px;
    grana *y=(grana *)py;
    if ((x->p)>(y->p)) return 1;
    if ((x->p)<(y->p)) return -1;
    if ((x->kv)<(y->kv)) return 1;
    if ((x->kv)>(y->kv)) return -1;
    if ((x->q)>(y->q)) return 1;
    if ((x->q)<(y->q)) return -1;
    return 0;
}
int upI(const void *px,const void *py)
{
    int *x=(int *)px;
    int *y=(int *)py;
    if (*x>*y) return 1;
    if (*x<*y) return -1;
    return 0;
}

void dfs(int i,int b,int s)
{
    ozn[i]=b;
    int j;
    for(j=poc[i];(j<poc[i+1]) && (pov[j].kv>s);j++)
       if (!ozn[pov[j].q])
          dfs(pov[j].q,b,s);
    return;
}



int moze(int s)
{
    memset(ozn,0,sizeof(ozn));
    int i,j=1;
    for(i=0;i<n;i++)
       if (!ozn[i])
          {dfs(i,j,s);j++;}

    int ind=1;
    for(i=0;(i<k)&&ind;i++)
       ind=(ozn[kuca1[i]]!=ozn[kuca2[i]]);
    return ind;
}

int main()
{
    freopen("struja.in","r",stdin);
    freopen("struja.out","w",stdout);

    int i,j,l,d,s;

    scanf("%d%d%d",&n,&m,&k);
    for(i=0;i<m;i++)
    {
       scanf("%d%d%d",&(pov[2*i].p),&(pov[2*i].q),&(pov[2*i].kv));
       pov[2*i].p--;pov[2*i].q--;
       pov[2*i+1].p=pov[2*i].q;
       pov[2*i+1].q=pov[2*i].p;
       pov[2*i+1].kv=pov[2*i].kv;
       sat[i+1]=pov[2*i].kv;
    }
    for(i=0;i<k;i++)
    {
       scanf("%d%d",&(kuca1[i]),&(kuca2[i]));
       kuca1[i]--;kuca2[i]--;
```

```
    }

    qsort(pov,2*m,sizeof(grana),upG);
    for(i=j=0;i<=n;i++)
    {
        while ((pov[j].p<i) && (j<2*m)) j++;
        poc[i]=j;
    }

    sat[0]=0;
    qsort(sat,m+1,sizeof(int),upI);
    for(i=1,j=0;i<=m;i++)
    {
       if (sat[j]!=sat[i])
       {
          j++;
          sat[j]=sat[i];
       }
    }

    l=0;d=j;
    while (d-l>1)
    {
          s=(l+d)/2;
          if (moze(sat[s]))
             d=s;
          else
             l=s;
    }
    if (moze(sat[l]))
       d=l;

    printf("%d\n",sat[d]);

  return 0;
}
```

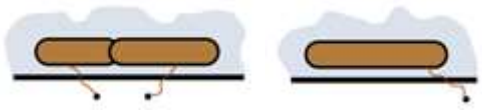
zadatak: Čamci

Stanovnici jednog malog sela bave se ribarenjem i programiranjem. Nakon završenog ribolova, ribari svoje čamce ostavljaju duž jedne obale uskog kanala. Svaki ribar ima jedan čamac i sopstveni stubić negde duž kanala koji mu služi za privezivanje čamca, pri čemu sme da priveže svoj čamac isključivo za svoj stubić. Čamci moraju biti ostavljeni tako da svaki čamac bude tačno uz svoj stubić, što znači da stubić mora biti negde neposredno pored čamca, tj. između dva kraja čamca (dozvoljeno je i da stubić bude neposredno pored nekog kraja čamca). Pošto je kanal uzan, najviše jedan čamac se može nalaziti na svakom poprečnom preseku kanala, odnosno svi čamci moraju biti privezani neposredno do obale. Čamci se mogu dodirivati svojim krajevima.
Zbog ovakvog vezivanja čamaca nije obavezno moguće da čamci svih ribara budu vezani istovremeno. Zato su ribari zamolili svoje prijatelje programere da im izračunaju koliko najviše čamaca može istovremeno biti vezano. Međutim, programerima je ovo bio pretežak zadatak, pa su i oni počeli da se bave ribarenjem, a zadatak su prepustili vama.

dozvoljena vezivanja



nedozvoljena vezivanja



Ulaz:
Ulazni podaci učitavaju se iz tekstualnog fajla **camci.in**. U prvom redu zapisan je samo broj ribara n ($1 \leq n \leq 10000$). U svakom od narednih n redova zapisani su prirodni brojevi l_i i p_i ($1 \leq l_i, p_i \leq 100000$) razdvojeni jednim razmakom, koji redom označavaju dužinu čamca i položaj (koordinatu) stubića i-tog ribara. Nijedna dva stubića nemaju istu koordinatu.
Izlaz:

U izlazni tekstualni fajl **camci.out** treba upisati samo jedan broj - koliko najviše čamaca može biti privezano istovremeno.

Primer:

camci.in	**camci.out**
7 5 9 2 17 6 10 3 11 2 16 4 13 5 6	5

Objašnjenje:

Najviše je moguće da 5 čamaca bude istovremeno vezano, na primer čamci 1, 2, 4, 5 i 7.

Rešenje

Zadatak je moguće rešiti na više načina od kojih je najefikasniji sledeći gramzivi algoritam:
Tokom algoritma održavamo skup privezanih čamaca S. Jedan po jedan čamac pokušavamo da ubacimo u taj skup idući sa leva na desno po koordinatama stubića. Na početku skup S sadrži samo najlevlji čamac privezan maksimalno u levo. Neka su l i r koordinate levog i desnog kraja poslednjeg ubačenog čamca, i neka trenutno posmatramo čamac opisan parom (l_i, p_i). Ako je $p_i \geq r$, tada ubacujemo i-ti čamac u skup S i privezujemo ga tako da ga postavimo što je moguće više levo. Ako je $p_i < r$, tada proveravamo da li važi $l_i < r - l$ i ako jeste izbacujemo poslednji čamac iz S, i umesto njega u S ubacujemo i-ti čamac, takođe što je moguće više levo, a ako ne važi, i-ti čamac preskačemo.
Indukcijom se može pokazati da za svako i ($1 \leq i \leq n$) važi sledeće tvrđenje: Nakon i koraka ovog algoritma u skupu S nalazi se najveći podskup skupa koji čine prvih i čamaca, koji se mogu istovremeno privezati, i to onaj podskup za koji važi da je desni kraj najdesnijeg čamca iz tog podskupa maksimalno levo.
Iz ovog tvrđenja je jasno da se nakon n koraka ovog algoritma dobija traženo rešenje.

Pseudokod

```
Sortirati niz parova (l[i], p[i]) tako da bude rastući po p[i]

m = 1
r = p1
for i = 2 to n
        if p[i] ≥ r
                l = r
                r = max(p[i], r + l[i])
                inc(m)
        else
                r = min(r, max(p[i], l + l[i])

m je traženi maksimalan broj čamaca
```

Vremenska složenost ovog algoritma je $O(n \lg n)$ u čemu dominira vreme potrebno za sortiranje.
Zadatak je takođe moguće rešiti dinamičkim programiranjem na nekoliko načina.
Kao i u prethodnom rešenju, najpre sortirajmo parove (l_i, p_i) da budu rastući po p_i. Za $k \leq m$ posmatrajmo sve skupove koji sadrže tačno k ispravno privezanih čamaca izabranih od prvih m čamaca. Uočimo sve koordinate desnih krajeva najdesnijih čamaca iz tih skupova. Sa $S(m, k)$ označimo najlevlju od svih tih koordinata. Ako ne postoji ni jedan takav skup, onda neka je $S(m, k) = \infty$.
Da bismo izračunali $S(m, k)$ moramo da razmotrimo nekoliko mogućnosti. Ako isključimo mogućnost da m-ti čamac bude privezan, tada je $S(m, k) = S(m-1, k)$. Da bi m-ti čamac uopšte bio privezan potrebno je da se njegov stubić ne nalazi pored nekog drugog čamca, tj. mora da važi $S(m-1, k-1) \leq p_m$. Ako to važi, m-ti čamac privezujemo što je moguće više levo. Ako je $S(m-1, k-1) < p_m - l_m$ tada on neće dodirivati prethodni čamac i važiće $S(m, k) = p_m$. Suprotno, ako je $S(m-1, k-1) \geq p_m - l_m$ tada će on dodirivati prethodni čamac i važiće $S(m, k) = S(m-1, k-1) + l_m$. Konačno možemo da zapišemo formulu:

$$S(m, k) = \min(S(m-1, k), R)$$

gde je $R =$

∞	ako je	p_m	$<$	$S(m-1, k-1)$		
$S(m-1, k-1) + l_m$	ako je	$p_m - l_m$	$\leq$	$S(m-1, k-1)$	$\leq$	p_m
p_m	ako je			$S(m-1, k-1)$	$<$	$p_m - l_m$

Početne vrednosti su $S(m, 0) = -\infty$, jer ne postoji najlevlja tačka ako nema ni jednog čamca.

Željeno rešenje (najveći broj čamaca koji se mogu istovremeno privezati) je najveće k za koje $S(n, k)$ nije ∞ (jer $S(n, k) = \infty$ znači da istovremeno vezivanje k čamaca nije moguće).
Sada je lako napisati program koji izračunava $S(m, k)$ za sve m i k ($k \le m \le n$). Iako je S dvodimenzionalno, možemo koristiti samo jednodimenzionalan niz za S jer je za izračunavanje $S(m, k)$ dovoljno znati samo vrednosti iz m-1-og reda (jer da bismo dobili $S(m, k)$ koristimo samo $S(m-1, k-1)$ i $S(m-1, k)$).

Pseudokod

```
Sortirati niz parova (l[i], p[i]) tako da bude rastući po p[i]

S[0] = -inf
S[1] = inf
m = 0

for i = 1 to n
        for j = m+1 downto 1
                if S[j-1] ≤ p[i]
                        r = max(p[i], S[j-1] + l[i])
                else
                        r = inf
                S[j] = min(S[j], r)
        if S[m+1] < inf
                inc(m)
        S[m+1] = inf

m je traženi maksimalan broj čamaca
```

Vremenska složenost ovog algoritma je $O(n^2)$.
Ostala rešenja dinamičkim programiranjem se dobijaju ako se koristi činjenica da su koordinate stubića i dužine čamaca celi brojevi unutar relativno malog intervala.

fajl: camci.pas

```
const
  fin = 'camci.in';
  fout = 'camci.out';
  maxN = 10000;

type
  TBoat = record
            p, l : Longint;
          end;

var
  n, m : Integer;
  c : array[1..maxN] of TBoat;



procedure ReadInput;
var
  f : Text;
  i : Integer;
begin
  Assign(f, fin);
  Reset(f);
  Readln(f, n);
  for i := 1 to n do
    Readln(f, c[i].l, c[i].p);
  Close(f);
end;


procedure Sort;
var
  tmp : TBoat;
```

```
  procedure QuickSort(l, r : Integer);
  var
    i, j : Integer;
    p : Longint;
  begin
    i := l;
    j := r;
    p := c[(l + r) div 2].p;
    repeat
      while c[i].p < p do inc(i);
      while p < c[j].p do dec(j);
      if i <= j then
      begin
        tmp := c[i];
        c[i] := c[j];
        c[j] := tmp;
        inc(i);
        dec(j);
      end;
    until i > j;
    if l < j then QuickSort(l, j);
    if i < r then QuickSort(i, r);
  end;

begin
  QuickSort(1, n);
end;


function Max(a, b : Longint) : Longint;
begin
  if a > b then
    Max := a
  else
    Max := b
end;


procedure Solve;
var
  i : Integer;
  l, r, t : Longint;
begin
  Sort;

  m := 1;
  r := c[1].p;

  for i := 2 to n do
    if c[i].p >= r then
    begin
      l := r;
      r := Max(c[i].p, r + c[i].l);
      inc(m);
    end else
    begin
      t := Max(c[i].p, l + c[i].l);
      if t < r then
        r := t;
    end;
end;


procedure WriteOutput;
var
  f : Text;
begin
```

```
  Assign(f, fout);
  Rewrite(f);
  Writeln(f, m);
  Close(f);
end;



begin
  ReadInput;
  Solve;
  WriteOutput;
end.
```

fajl: camci2.pas

```
const
  fin = 'camci.in';
  fout = 'camci.out';
  maxN = 10000;
  inf = 999999999;

type
  TBoat = record
            p, l : Longint;
          end;

var
  n, m : Integer;
  c : array[1..maxN] of TBoat;



procedure ReadInput;
var
  f : Text;
  i : Integer;
begin
  Assign(f, fin);
  Reset(f);
  Readln(f, n);
  for i := 1 to n do
    Readln(f, c[i].l, c[i].p);
  Close(f);
end;


procedure Sort;
var
  tmp : TBoat;

  procedure QuickSort(l, r : Integer);
  var
    i, j : Integer;
    p : Longint;
  begin
    i := l;
    j := r;
    p := c[(l + r) div 2].p;
    repeat
      while c[i].p < p do inc(i);
      while p < c[j].p do dec(j);
      if i <= j then
      begin
```

```
        tmp := c[i];
        c[i] := c[j];
        c[j] := tmp;
        inc(i);
        dec(j);
      end;
    until i > j;
    if l < j then QuickSort(l, j);
    if i < r then QuickSort(i, r);
  end;

begin
  QuickSort(1, n);
end;


function Min(a, b : Longint) : Longint;
begin
  if a < b then
    Min := a
  else
    Min := b
end;


function Max(a, b : Longint) : Longint;
begin
  if a > b then
    Max := a
  else
    Max := b
end;


procedure Solve;
var
  i, j, k : Integer;
  s : array[0..MaxN+1] of Longint;
  r : Longint;
begin
  Sort;

  s[0] := -inf;
  s[1] := inf;
  m := 0;

  for i := 1 to n do
  begin
    for j := m+1 downto 1 do
    begin
      if s[j-1] <= c[i].p then
        r := Max(c[i].p, s[j-1] + c[i].l)
      else
        r := inf;

      s[j] := Min(s[j], r);
    end;

    if s[m+1] < inf then
    begin
      inc(m);
      s[m+1] := inf;
    end;
  end;
end;
```

```
procedure WriteOutput;
var
  f : Text;
begin
  Assign(f, fout);
  Rewrite(f);
  Writeln(f, m);
  Close(f);
end;



begin
  ReadInput;
  Solve;
  WriteOutput;
end.
```