

Dinamičko programiranje - zadaci

Dinamičko programiranje (DP) je nastalo kao način rešavanja jedne klase algoritamskih problema, u kojima se traži optimalno rešenje, odnosno rešenje koje maksimizira ili minimizira neku zadatu veličinu. Kasnije se naziv preneo na širu klasu problema koji se rešavaju u osnovi istom idejom, a to je rešavanje manjih problema istog tipa i njihovo kombinovanje radi dobijanja rešenja polaznog, većeg problema.

DP je tehnika oblikovanja rešenja koja rastavlja složen problem na manje složene potprobleme/korake.

Za DP važno je usvojiti specifičan način razmišljanja pomoću kog treba:

1. pronaći sva stanja

Pri rešavanju probleme upotrebom DP-a, potrebno je rastaviti problem na **stanja**. Stanje je rešenje nekog malog dela problema.

2. Na temelju prethodno izračunatih stanja, izračunati novo stanje (moramo znati relaciju)

3. Dinamike (algoritmi koji koriste DP) izračunavaju stanja takvim redom, tako da se niti jedno stanje ne izračunava dva puta.

4. Poznavati početna stanja

1.

Napisati rekurzivnu funkciju koja računa n-ti element u Fibonačijevom nizu. Popraviti funkciju tako da se problemi manje dimenzije rešavaju samo jedanput. (dinamičko programiranje).

Analiza rešenja

U slučaju da se rekurzijom problem svodi na više manjih podproblema koji se mogu preklapati, postoji opasnost da se pojedini podproblemi manjih dimenzija rešavaju veći broj puta. Na primer,

$\text{fibonacci}(20) = \text{fibonacci}(19) + \text{fibonacci}(18)$

$\text{fibonacci}(19) = \text{fibonacci}(18) + \text{fibonacci}(17)$

tj. problem fibonacci(18) se resava dva puta. Problemi manjih dimenzija će se rešavati još veći broj puta.

Rešenje za ovaj problem je kombinacija rekurzije sa tzv. "dinamičkim programiranjem" tj.

podproblemi se rešavaju samo jednom, a njihova rešenja se pamte u memoriji (obično u nizovima ili matricama), odakle se koriste ako tokom rešavanja ponovo budu potrebni.

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

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

```
#define MAX 50
```

```
/* Niz koji cuva resenja podproblema. */
```

```
int f[MAX];
```

```
/* Funkcija izracunava n-ti fibonacijev broj */
```

```
int fibonacci(int n)
```

```
{
```

```
/* Ako je podproblem vec resen, uzimamo gotovo resenje! */
```

```
if(f[n] != 0)
```

```
return f[n];
```

```
/* Izlaz iz rekurzije */
```

```
if(n < 2)
```

```
return f[n] = 1;
```

```
else
```

```
/* Rekurzivni pozivi */
```

```
return f[n] = fibonacci(n - 1) + fibonacci(n - 2);
```

```
}
```

```

/* Test program */
int main()
{
    int n, i;
    /*Inicijalizuje se niz*/
    for(i=0; i<MAX; i++)
        f[i] = 0;
    scanf("%d", &n);
    printf("%d\n", fibonacci(n));
    return 0; }

```

2. Programer Vasa radi za tajnu organizaciju i mora uočiti pravilnost između dva niza podataka. Cilj mu je da za dva niza brojeva pronađe dužinu najdužeg (ne obavezno povezanog) podniza koji se pojavljuju u oba niza. Ulazni podaci su n i m ($1 \leq n, m \leq 1000$), koji redom predstavljaju dužine oba niza brojeva. Nakon toga sledi n celih brojeva i zatim m celih brojeva. Ispisati dužinu najdužeg zajedničkog podniza.

ULAZ	IZLAZ
10 9	5
3 2 4 1 5 8 7 3 6 9	
2 7 5 1 7 3 4 2 6	

Napomena: Najduži zajednički podniz (LCS, longest common subsequence) je 2 1 7 3 6 ili 2 5 7 3 6.

Šta je stanje u ovom DP problemu? Nazovimo nizove redom A, B.

Stanje $DP[x][y]$ definišemo kao dužinu LCS u prvih $x+1$ elemenata prvog niza A i prvih $y+1$ elemenata niza B.

Šta je relacija u ovom DP problemu?

Jednostavno dizajniramo DP algoritama koristeći induktivni pristup, tj. pretpostavimo da smo rešili potproblem za prefiks niza A dužine $x-1$ i prefiks niza B dužine $y-1$:

Ako $A[x]=B[y]$, onda: $DP[x][y]=\max\{DP[x][y-1], DP[x-1][y], DP[x-1][y-1]+1\}$

Inace: $DP[x][y]=\max\{DP[x][y-1], DP[x-1][y]\}$

Zaista novo stanje možemo izračunati proširujući pre izračunato stanje po x ili po y .

Ili ako su elementi $A[x]$, $B[y]$ jednaki, možemo proširiti stanje $DP[x-1][y-1]$ i dodati 1, jer LCS će biti produžen za još jedan član tj. $A[x]$ (ili $B[x]$).

Resenje

```

#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

```

```

int main()
{
    int n,m;
    cin>>n>>m;
    vector<int> A(n), B(m,0);
    vector<vector<int>> DP(n,B); //inicijalna stanja u resenju

```

```

    for(int i=0;i<n;i++) cin >>A[i];
    for(int i=0;i<m;i++) cin >>B[i];

```

```

    for(int i=0;i<n;i++)
        for(int j=0;j<m;j++) {
            if (i) DP[i][j]=max(DP[i][j], DP[i-1][j]);
            if (j) DP[i][j]=max(DP[i][j], DP[i][j-1]);
            if (A[i]==B[j])
                DP[i][j]=max(DP[i][j], ((i&&j)?DP[i-1][j-1]:0)+1);
        }

```

```

cout << DP[n-1][m-1] << endl;
return 0;
}

```

3. Šumadija, nekada prebogata šumama, danas se više time ne može ponositi. Zbog zaštite životne sredine pokušava se na svaki način sprečiti nesavesna seča drveća. Lokalni, novopeceni bogataš, kupio je poveliko kvadratno parče zemlje, na njemu želi da izgradi što je veće moguću kuću i to tako da i njena osnova bude kvadratnog oblika sa stranama paralelnim stranama placa. Mapa imanja je sastavljena iz kvadratića dimenzija 1x1. Jedno drvo zauzima tačno jedan takav kvadratić. Pošto je imanje veliko, vrlo je teško naći najveći kvadrat sa stranama paralelnim stranama placa na kome nema drveća. Vaš zadatak je da za zadatu mapu imanja na kojoj se nalaze pozicije drveća, nađete kvadrat čija je strana najduža na kome nema stabala. Dužina strane je broj kvadratica 1x1 duž njegove strane.

Sa standardnog ulaza se učitava sledeće: U prvom redu broj n ($n \leq 200$) koji je dužina strane imanja. Sledeći red sadrži broj p , broj stabala na imanju. Narednih p redova sadrže po dva broja koji su koordinate stabala. Na standardni izlaz treba ispisati samo jedan broj, dužina stranice najvećeg kvadrata koji ne sadrži ni jedno drvo.

Primer

Ulaz:

```

5
3
2 2
4 2
4 4

```

Izlaz:

```

3

```

Koordinate gornjeg levog temena traženog kvadrata su 1 3, a donjeg desnog 3 5.

	D			
	D		D	

Ideja

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	0	300	300	300	300	1	0	1	2	2	1	0	1	2	2	1	0	2	2	2
1	300	300	300	300	300	1	300	300	300	300	1	2	2	2	3	1	2	2	3	3
1	0	300	0	300	300	1	0	300	0	300	1	0	300	0	300	1	0	2	0	1
1	300	300	300	300	300	1	300	300	300	300	1	300	300	300	300	1	300	300	300	300

Rešenje :

```

#include <iostream>
#include <cstdio>
#include <cstring>

```

```

using namespace std;
#define MaxN 200
#define BLA 300
int a[MaxN+1][MaxN+1];

```

```

int min3(int x, int y, int z)
{
    int min=x;

```

```

    if (y<min) min=y;
    if (z<min) min=z;
    return min;
}

int main()
{ int i, j, p, n, X, Y, Max1=1;
  cin >> n >> p;

  for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
  {   if ((i==1) || (j==1)) a[i][j]=1;
      else
      a[i][j]=BLA;
  }

  for(i=1;i<=p;i++)
  {
    cin >>X>>Y;
    a[X][Y]=0;
  }

  for(i=2;i<=n;i++)
  for(j=2;j<=n;j++)
    if ( a[i][j]>0)
    {
      a[i][j]=1+min3(a[i-1][j-1],a[i-1][j],a[i][j-1]);
      if(a[i][j]>Max1) Max1=a[i][j];
    }
  cout<<Max1<<endl;

  return 0;
}

```

II način

```

#include <iostream>
#include <cstring>

using namespace std;

int main (){
  int n, p;
  cin >> n >> p;
  int pos[p][2];
  for (int i = 0; i < p; ++i){
    cin >> pos[i][0] >> pos[i][1];
  }
  int polje[n][n];
  memset (polje, 0, sizeof (polje));
  for (int i = 0; i < p; ++i){
    polje[pos[i][0] - 1][pos[i][1] - 1] = 1;
  }
  bool test = true;
  for (int i = n; i > 0; --i){

```

```

for (int x = 0; x < n - i + 1; ++x){
    for (int y = 0; y < n - i + 1; ++y){
        test = true;
        for (int x2 = 0; x2 < i; ++x2){
            for (int y2 = 0; y2 < i; ++y2){
                if (polje[x2 + x][y2 + y] == 1){test = false; break;}
            }
            if (!test){break;}
        }
        if (test){cout << i; return 0;}
    }
}
}
cout << 0;
}

```

5. U kvadratu dimenzija $N \times N$, gde je $N < 13$, polja su popunjena ciframa od 0 do 9. Ako su vrste i kolone kvadrata označene brojevima od 0 do $N-1$ napisati program kojim se ispisuje maršuta kojom se iz gornjeg levog polja (0,0) stiže do donjeg desnog polja ($N-1, N-1$) tako da su zadovoljeni sledeći uslovi:

- 1) sa svakog polja je dozvoljeno preći samo na polje ispod ili na polje desno od tog polja;
- 2) od svih maršuta, koje zadovoljavaju prethodni uslov ispisati onu čija je suma cifara, u poljima preko kojih se ide, maksimalna.

6.

U novoj eri elektronskog poslovanja, bankomati sve više postaju svakodnevna pojava. Banke žele da se klijenti što krace zadržavaju kod bankomata, kako bi opslužili što više ljudi, i zbog toga zahtevaju da se traženi iznos novca klijentima isplaćuje sa što manje novčanica. Međutim, bankomate su programirali lenji programeri koji su upotreбили najjednostavniji algoritam tako da se u svakom njegovom koraku bira najveća novčanica koja je manja ili jednaka od preostalog iznosa. Na primer, za sistem sa novčanicama od 1, 7 i 10 dinara, iznos od 14 dinara biće isplaćen pomoću jedne novčanice od 10 i 4 novčanice od 1 dinara, ukupno 5 novčanica. Ovo ne odgovara zahtevu banke, jer se 14 dinara može isplatiti pomoću samo dve novčanice od 7 dinara.

Vaš zadatak je da napišete program koji će za dati sistem novčanica, koji uvek sadrži i apoen od jednog dinara, naći najmanji iznos koji se može isplatiti u manje novčanica od broja novčanica koji bi bankomat iskoristio.

Ulaz: U prvom redu ulaza nalazi se prirodan broj n , broj novčanica u sistemu ($1 \leq n \leq 50$). U drugom redu ulaza nalaze se n različitih prirodnih brojeva razdvojenih razmakom, vrednosti novčanica u sistemu. Svaki od njih nije veći od 500000, a jedan od njih je 1.

Izlaz: U izlaz upisati jedan prirodan broj, najmanji iznos koji bankomati ne isplaćuju u najmanjem broju novčanica. Ako za dati sistem bankomati uvek isplaćuju iznos u najmanjem broju novčanica, napisiti 0.

Primer:

```

ulaz
3
1 7 10

```

```

izlaz
14

```

```

#include <iostream>
#include <algorithm>
#include <cmath>
#include <climits>

```

```

using namespace std;

#define MaxN 50
#define MaxMonete 500000

int DP[2*MaxMonete+1], greedy[2*MaxMonete+1];
int monete[MaxN];
int main()
{ int n;

  cin >> n;
  for (int i=0;i<n;i++) cin >> monete[i];

  sort(monete, monete+n);
//max velicina moneta je monete[n-1]
  for (int i=1; i<2*monete[n-1];i++)
  {
    DP[i]=INT_MAX; //tabela za dinamicko programiranje
    for (int j=0; j<n; j++)
      if (i>=monete[j] )
      { //paralelno popunjavamo tabelu pohlepnog
        //rasitnjavanja i rasiitnjavanja din.prog.
        //za sve i koji su manji od tekuće monete[j]
        DP[i]=min(DP[i],DP[i-monete[j]]+1);
        greedy[i]=greedy[i-monete[j]]+1;
      }
    //ako se rasiitnjavanje pohlepnoim metodom
    //i din.prog. razlikuju, stampamo najmanji takav iznos
    if (DP[i]!=greedy[i])
    { cout<<i<<endl;
      return 0;
    }
  }
  cout<<"0"<<endl;
  return 0;
}

```

7. Za cele brojeve zadate trougaono kao na slici odrediti put koji obezbeđuje najveću sumu, a ide od vrha i završava se na osnovi trougla. Dozvoljeni pravac kretanja je nadole: dijagonalno levo i dijagonalno desno.

```

              7
            3   8
          8   1   0
        2   7   4   4
      4   5   2   6   5

```

Ideja:

Trougaonu tabelu D ćemo spakovati u matricu sledećeg oblika:

```

7 0 0 0 0
3 8 0 0 0
8 1 0 0 0
2 7 4 4 0
4 5 2 6 5

```

i pravilo kretanja prevesti na: nadole pravo i nadole dijagonalno desno.

8. (POJ zadatak, 2009 ACM North Western European Regional Contest, Divisible Subsequences, Time Limit: 1000MS Memory Limit: 65536K)

Dat je niz prirodnih brojeva. Potrebno je prebrojati sve uzastopne podnizove (koji se nekad zovu i podniska, jer sadrže strogo uzastopne elemente niza) čija suma je deljiva zadatim brojem. Ovi podnizovi se mogu preklapati.

Na primerr, niz

2, 1, 2, 1, 1, 2, 1, 2

sadrži šest uzastopnih podnizova koji se deljivi sa 4: podniz prvih 8 brojeva, podniz {1,2,1} od 2. do 4. člana, podniz {1,2,1,1,2,1} od 2. do 7. člana, podniz {2,1,1} od 3. člana do 5. člana, podniz {1,1,2} od 4. do 6. člana, podniz {2,1,2} od 5. do 7. člana.

ULAZ:

Prva linija standardnog ulaza sadrži ceo broj c ($1 \leq c \leq 200$), broj test primera. Potom slede dve linije za svaki test primer.

Svaki test primer počinje redom u kom su navedena dva cela broja d ($1 \leq d \leq 1\,000\,000$) i n ($1 \leq n \leq 50\,000$), gde d je delilac sume podniza, n je dužina niza. U drugoj liniji test primera nalaze se članovi niza. Članovi niza su celi brojevi iz segmenta $[1, 1\,000\,000\,000]$.

IZLAZ:

Za svaki test primer, odšampajte u zasebnoj liniji jedan ceo broj, koji predstavlja broj uzastopnih podnizova deljivih sa zadatim brojem d .

Ulaz:

2
7 3
1 2 3
4 8
2 1 2 1 1 2 1 2

Izlaz:

0
6

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cstring>
```

```
using namespace std;
```

```
#define MAX_NUMBER 1000000
```

```
int Prebroj[MAX_NUMBER];
```

```
int main ()
```

```
{
    int c;
    scanf("%d",&c);
```

```
    while (c)
    {
        memset(Prebroj,0,sizeof(Prebroj));
        int d,n,Sum=0,rez=0;
```

```
        scanf("%d %d",&d,&n);
```

```
        Prebroj[0]=1;
```

```

for (int i=0;i<n;i++)
{
    int Number;
    scanf("%d",&Number);
    Sum=(Sum+(Number%d))%d;
    rez += Prebroj[Sum]++;
}

printf("%d\n",rez); c--;
}

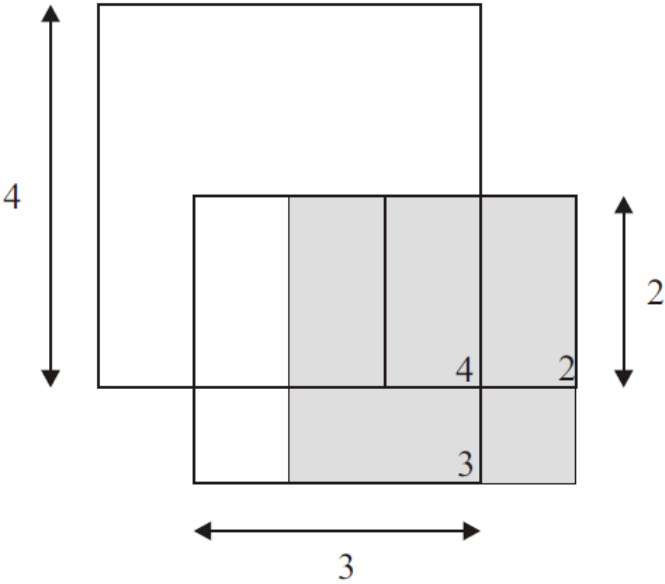
return 0;
}

```

9. U matrici Anxm, popunjenoj sa 0 i 1, odrediti kvadratni blok najveće dimenzije koji se sastoji samo od nula.

0	0	0	0	0	1
1	0	0	1	0	0
0	0	0	0	0	1
1	0	0	0	1	0
1	0	0	0	1	1

1	1	1	1	1	0
0	1	2	0	1	1
1	1	2	1	1	0
0	1	2	2	0	1
0	1	2	3	0	0



Slika 1.

10. Najduži zajednički podniz (eng. longest common subsequence): Data su dva niza **X** i **Y**. Naći niz najveće moguće dužine koji je podniz i za **X** i za **Y**.
Niz $x=(x_1, x_2, \dots, x_k)$ je podniz niza $y=(y_1, y_2, \dots, y_m)$, ako i samo ako postoji strogo rastući niz (i_1, i_2, \dots, i_k) indeksa, takav da za svako $j= 1, 2, \dots k$ važi $y_{i_j}=x_j$.
Na primer, niz $z=(A, N, A)$ je podniz niza $y=(G, L, A, D, N, A)$ gde je odgovarajući niz indeksa $(3, 5, 6)$. Ili $(2,4,5)$ ako je indeks prvog elementa niza nula, a ne jedan.
Ako su data dva niza **X** i **Y**, kažemo da je niz **Z** zajednički podniz od **X** i **Y**, ako je **Z** podniz i od **X** i od **Y**.
Niz **Z**=(A, N, A) je zajednički podniz nizova **X**=(B, A, N, A, N, A) i **Y**=(B, R, A, N, A) i njegova dužina je 3, ali **Z** nije najveći zajednički podniz od **X** i **Y**, već je to niz (B, A, N, A), čija je dužina 4.

ULAZ:
Prva linija sadrži string **X**. Druga linija sadrži string **Y**. Oba stringa sadrže najmanje jedan, a najviše 1000 karaktera.

IZLAZ:
Ispisati na standardni izlaz dužinu podniza najveće moguće dužine koji je podniz i za **X** i za **Y**.

Ulaz
aleks
abcdef

Izlaz:

2

Ulaz:

abc

def

Izlaz:

0

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cstdlib>
#define MaxN 1000
//    Common Sequence

#define max(a,b) ((a)>(b)?(a):(b))

int Dp[MaxN+1][MaxN+1];
char X[MaxN+1],Y[MaxN+1];
int main()
{ int i, j, lenX, lenY; char pom;
  //scanf("%s", X); //scanf("%c", &pom);
  //scanf("%s", Y);
  fgets(X, MaxN+1,stdin);
  fgets(Y, MaxN+1,stdin);
  lenX=strlen(X);
  lenY=strlen(Y);
  for (i=1;i<=lenX;i++)
    for(j=1;j<=lenY;j++)
      if (X[i-1]==Y[j-1]) Dp[i][j]=Dp[i-1][j-1]+1;
      else Dp[i][j]=max(Dp[i-1][j], Dp[i][j-1]);
  printf("%d", Dp[lenX-1][lenY-1]);

  return 0;
}
```

11.

Dato je **N** celih brojevi iz intervala [1, 100 000 000] koji predstavljaju listu **A**. Odrediti najduži niz koji se sastoji od datih brojeva tako da za svaka dva susedna a_i i a_j , tako da $i = j - 1$ važi a_i deli a_j , i ispisati nađeni niz. U slučaju da postoji više rešenja sa najdužim mogućim nizom, ispisati najmanje leksikografsko.

Za dva niza $X = (x_1, x_2, \dots, x_n)$ i $Y = (y_1, y_2, \dots, y_n)$ kažemo da je X leksikografski manje od Y ako postoji **i**, tako da za sve **j** < **i** važi $x_j = y_j$ i $x_i < y_i$.

Ulaz:

Prvi red standardnog ulaza sadrži ceo broj **N** ($2 \leq N \leq 1\,000$). Potom se u narednih **N** redova učitava lista brojeva $A = \{a_1, a_2, \dots, a_N\}$, u **i** + 1-om redu ceo broj $a_{\text{_i}}$ ($1 \leq a_{\text{_i}} \leq 100\,000\,000$).

Garantuje se da će ulaz sadržati bar dva broja $a_{\text{_i}}$ i $a_{\text{_j}}$ tako da $a_{\text{_i}}$ deli $a_{\text{_j}}$.

OUTPUT:

Prvi red standardnog izlaza treba da sadrži prirodan broj **M** koji predstavlja dužinu nađenog niza koji zadovoljava

gore opisane uslove, a sastoji se od elemenata liste **A**. Svaki elemenat liste **A** može biti upotrebljen najviše jednom. U drugom redu treba ispisati **M** brojeva odvojenih tačno jednim razmakom koji predstavljaju niz **A**. Pre prvog i nakon poslednjeg elementa ne treba da se nalazi razmak, odnosno drugi red treba da sadrži tačno **M** - 1 razmak.

Ulaz:

20
8
2
5
3
18
19
4
11
10
9
2
17
15
14
20
10
5
13
12
14

Izlaz:

5
2 2 10 10 20

Ulaz:

6
1
3
5
7
11
13

Izlaz:

2
1 3

Objašnjenje:

Postoji 5 nizova sa dužinom 2:

(1, 3), (1, 5), (1, 7), (1, 11), (1, 13).

Od njih leksikografski najmanji je (1, 3).

12.

Mali Z je rešio da se bavi menadžerskim poslom. Nakon nekoliko dana traganja za poslom, konačno se zaposlio u poznatoj kompaniji zoogle. Već prvog dana dobio je ozbiljan zadatak:

Glavni direktor firme mnogo voli da gleda filmove, međutim, za to ima mnogo malo vremena. Mali Z je zato dobio zadatak da direktoru snimi filmove. Direktor je malom Z-u dao spisak filmova, i za svaki od filmova: vreme kada počinje, vreme kada se završava, i koliko će Z dobiti para ako snimi taj film.

Mali Z ima na raspolaganju samo jedan video rekorder, i treba da snimi tako da zaradi što je više moguće para.

Ulaz:

Sa standardnog ulaza se u prvom redu učitava prirodni broj N ($1 \leq N \leq 100000$) - broj filmova na spisku. Zatim se u svakom od narednih N redova nalaze tri prirodna broja. P , K i O . Gde je $1 \leq P < K \leq 2000000000$, i $1 \leq O \leq 1000$. P predstavlja vreme početka filma, K predstavlja vreme kraja filma, a O predstavlja Z-ovu zaradu sa snimljeni film.

Napomena: Ukoliko se jedan film završava u istom trenutku kada neki drugi film počinje, mali Z NE MOŽE da snimi oba filma. Drugim recima, ako je $K[a] = P[b]$, za neke filmove a i b , tada Z ne može snimiti i film a i film b .

Izlaz:

Na standardni izlaz ispisati samo jedan broj koji predstavlja maksimalnu Z-ovu zaradu.

Primer:

Ulaz:

```
3
1 5 2
2 6 3
6 8 2
```

Izlaz:

```
4
```

Dodatni primer:

Ulaz:

```
4
1 3 2
4 6 2
2 5 8
4 5 3
```

Izlaz:

```
8
```

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <vector>
```

```
using namespace std;
#define MaxN 100000
```

```
struct film
{
    int P, K, O;
    film ( int P1, int K1, int O1)
    {
        P = P1; K = K1; O = O1;
    }
}
```

```
inline friend bool operator < (const film & X, const film &Y )
```

```

    {
        return X.K < Y.K;
    }
};

```

```

long long int DP[MaxN];
vector <film> FilmNiz;

```

```

int main()
{
    int n,i; long long int tek = 0;
    scanf ("%d",&n);
    for (i=0;i<n;i++)
    {
        int P1,K1,O1;
        scanf("%d %d %d", &P1, &K1, &O1 );
        FilmNiz.push_back(film(P1,K1,O1));
    }
    //uredjen spisak prema zavrsetku filma
    sort(FilmNiz.begin(),FilmNiz.end());

    for (i=0;i<n;i++)
    {
        tek = max(tek,0LL + FilmNiz[i].O);
        film F = FilmNiz[i];
        F.K = F.P;
        int index = lower_bound (FilmNiz.begin(), FilmNiz.begin()+i,F)-FilmNiz.begin ();
        if (index==0) DP[i]=tek;
        else
        {
            index--;

            if (FilmNiz[index].K<FilmNiz[i].P)
                tek = max(tek,FilmNiz[i].O+DP[index]);

            DP[i] = tek;
        }
    }

    printf ("%lld\n", DP[n-1]);

    return 0;
}

```