ZADATAK	Joško	KHL	Okružen	Superjunaci
ulazni podaci	standardni ulaz			
izlazni podaci	standardni izlaz			
vremensko ograničenje	1 sec	1 sec	1 sec	2 sec
memorijsko ograničenje	256 MB	256 MB	256 MB	256 MB
broj bodova	**100**	**100**	**100**	**100**
	400			

Postojalo je nekoliko sekcija bodovanja, ovdje nećemo ulaziti u detalje kako riješit svaku pojedinačno već ćemo ispričati rješenje koje donosi maksimalan broj bodova.

Prvo što možemo primijetiti u zadatku je da će se Joško spustit na drugu razinu isključivo na pozicijama gdje dobiva prvu ili zadnju priliku kako bi se prebacio na drugu razinu i drugu dužinu. Sve te prilike događaju se na cjelobrojnim koordinatama, i to na koordinatama koje podudaraju počecima i krajevima dužina. Ukupno će biti takvih 2**N** koordinata na x-osi.

S obzirom da smo sada ograničili broj koordinata koje moramo promatrati na svakoj razini možemo zadatak riješit dinamičkim programiranjem. Poredajmo tih 2**N** koordinata s lijeva na desno i zapišimo ih u polje **p**.

Neka je dp(**n**, $\mathbf{p}_i$) = najkraće vrijeme do **n**-te razine i pozicije $\mathbf{p}_i$ na njoj od početne lokacije našeg junaka Joška. Kako bismo zadatak riješili dinamičkim programiranje potrebno je odrediti funkciju prijelaza, tj. opisati jedno stanje dinamike nekim drugim stanjima dinamike, koja su prethodno već izračunata. Na dva smo načina mogli doći do trenutnog stanja: ili smo se pomaknuli u desno s iste razine i pozicije $\mathbf{p}_{i-1}$ ili smo se spustili s razine iznad, neke razine **m, m < n**. Potrebno je za oba slučaja provjerit jesmo li uistinu mogli doći od tamo, ali uz pažljivo postavljanje početnih vrijednosti i obilaženja pozicija finalnu relaciju dinamičkog programiranje možemo zapisati kao dp(**n**, $\mathbf{p}_i$) = min (dp(**m**, $\mathbf{p}_i$), dp(**n**, $\mathbf{p}_{i-1}$) + $\mathbf{t}_n$ * ($\mathbf{p}_i$ - $\mathbf{p}_{i-1}$)), gdje je **m** neka razina više od razine **n** (**m < n**), a $\mathbf{t}_n$ vrijeme potrebno za preći jediničnu dužinu na trenutnoj razini. Za detalje pogledajte priloženi kod.

Ukupna vremenska složenost je O($\mathbf{n}^3$). Kao vježbu za čitatelja zgodno je spomenuti i postojanje rješenja složenosti O($\mathbf{n}^2$), pokušajte ga otkriti!

Potrebno znanje: polja, sortiranje

Kategorija: dinamičko programiranje

Programski kod

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

using namespace std;

const int MAXN = 105;
const int inf = 1 << 30;

int dp[MAXN][MAXN*2];
int n, m;
int a[MAXN], b[MAXN], t[MAXN];

int main (void){
  scanf("%d%d", &n, &m);
  for (int i = 0; i < MAXN; ++i)
    for (int j = 0; j < 2*MAXN; ++j)
      dp[i][j] = inf;

  vector <int> V;
  for (int i = 0; i < n; ++i) {
    scanf("%d%d%d", &a[i], &b[i], &t[i]);
    V.push_back(a[i]);
    V.push_back(b[i]);
```

```
  }

  sort(V.begin(), V.end());
  V.resize(unique(V.begin(), V.end()) - V.begin());
  dp[0][lower_bound(V.begin(), V.end(), a[0]) - V.begin()] = 0;

  for (int i = 0; i < n; ++i)
    for (int j = 0; j < 2*n; ++j)
      if (V[j] >= a[i] && V[j] <= b[i]) {
      if (j > 0) dp[i][j] = min(dp[i][j], dp[i][j-1] + (V[j] - V[j-1]) * t[i]);
      for (int k = 0; k < i; ++k) dp[i][j] = min(dp[i][j], dp[k][j]);
      }

  printf("%d\n", dp[n-1][lower_bound(V.begin(), V.end(), b[n-1]) - V.begin()]);
  return 0;
}
```

Rješenje zadatka pokažimo analizom koda pisanog u Pythonu.

```
N = int(input())
bodovi = 0
for kolo in range(N):
    L = input().split("/")
    L1 = []
    for i in L:
        L1 = L1 + i.split(":")
    domacin = gost = 0
    for i in range(0, len(L1), 2):
        domacin += int(L1[i])
        gost += int(L1[i + 1])
    opcija = len(L1)
    if kolo % 2 == 0: # domaćin je
        if opcija == 6: # gotovo u tri trećine
            if domacin > gost:
                bodovi += 3
        if opcija == 8: # produžetci
            if domacin > gost:
                bodovi += 2
            else:
                bodovi += 1
        if opcija == 10: # penali
            if domacin > gost:
                bodovi += 1
    else:
        if opcija == 6:
            if domacin < gost:
                bodovi += 3
        if opcija == 8:
            if domacin < gost:
                bodovi += 2
            else:
                bodovi += 1
        if opcija == 10:
            if domacin < gost:
                bodovi += 1

print(bodovi)
```

Zadatak rješavamo tako da utvrdimo koji je najmanji broj poteza potrebnih da Mirko uspije pobijediti Slavka. Mirku se sigurno isplati pobijediti u prvom potezu u kojem može jer će broj koraka koje mora napraviti sigurno biti najmanji. Zato ćemo zadatak rješavati tako da ispitamo može li Mirko pobijediti u prvom potezu, ako ne može onda može li u drugom, pa u trećem...

Možemo primijetiti da je Mirku najbolje izgraditi zid pravokutnog oblika. Svaki drugi oblik zida možemo ograditi sa zidom pravokutnog oblika koji će imati manje ili jednako polja koje pripadaju zidu. Također ne postoji oblik zida za koji bi vrijedilo da je broj koraka koje Mirko mora napraviti kako bi pobijedio manji nego za pravokutni zid. Pravokutnik koji ograđujemo možemo odrediti tako da saznamo koje je najgornje, najlijevije, najdonje i najdesnije polje na koje može doći Slavkova figura u potezu kojeg razmatramo. S obzirom da ne postoje nikakve prepreke koje bi ometale Slavkovu figuru, najgornju poziciju na koju može doći Slavko određujemo tako da nađemo najgornju moguću Slavkovu početnu poziciju i zamislimo da se Slavko u svakom svom potezu pomicao prema gore. Analogno nalazimo i najlijeviju, najdonju i najdesniju poziciju.

Nakon što smo za neki potez odredili pravokutnik koji moramo ograditi izračunamo broj koraka koji nam treba da dođemo do gornjeg lijevog vrha pravokutnika i broj koraka da ogradimo pravokutnik te ispitamo možemo li do tog poteza napraviti toliko koraka. Potrebno je još u svakom potezu provjeriti može li Slavko doći do ruba ploče, a to je lako pomoću već određenih pozicija.

Potrebno znanje: polja, Manhattan udaljenost

Kategorija: ad hoc

Programski kod

```
#include <cstdio>
#include <iostream>
#include <cstring>
using namespace std;
const int MAXN = 210;
const int INF = 1e9;

int n, m, sol;
char a[MAXN][MAXN];

int main(void) {
  scanf("%d %d", &n, &m);
  for (int i = 0; i < n; ++i) scanf("%s", a[i]);
  sol = INF;
  for (int k = 0; k <= min(n, m); ++k) {
    int r1 = INF, r2 = -INF, c1 = INF, c2 = -INF;

    for (int r = 0; r < n; ++r)
      for (int c = 0; c < m; ++c)
        if (a[r][c] == 'x') {
          r1 = min(r1, r - k - 1);
          r2 = max(r2, r + k + 1);
          c1 = min(c1, c - k - 1);
          c2 = max(c2, c + k + 1);
        }
    if (r1 < 0 || c1 < 0 || r2 > n - 1 || c2 > m - 1) {
      break;
    }
```

```
    int s = r1 + c1 + 2 * (r2 - r1) + 2 * (c2 - c1);
    if (s <= 10 * (k + 1)) sol = min(sol, 2 * (r2 - r1) + 2 * (c2 - c1));
  }

  if (sol == INF) sol = -1;
  printf("%d\n", sol);

  return 0;
}
```

Pronađimo najprije sve nedostupne nebodere. Za svakog superjunaka mogli bismo proći lijevo i desno da bismo utvrdili koje sve nebodere on može dosegnuti i označili te nebodere kao dostupne. Tada će nedostupni neboderi biti oni koji na kraju ostanu neoznačeni. Ovaj pristup je, nažalost, prespor jer u najgorem slučaju putujemo po N nebodera za svakog od N superjunaka (N * N).

Možemo li ovaj postupak ubrzati tako da nijedan neboder ne označimo više puta? Što ako superjunak dođe do već označenog nebodera u nekom smjeru – trebamo li nastaviti označavati njegove nebodere u tom smjeru dalje? Primjerice, uzmimo da je superjunak s početnog nebodera visine 10 došao do nebodera koji je već označio neki prethodni superjunak. Ako je taj prethodni superjunak potekao s nebodera visine 7, naš superjunak mora nastaviti jer će se možda pojaviti neboder visine 8 ili 9 koji još nije označen. No ako je prethodni superjunak potekao s nebodera visine 15, tada je on sigurno već dosegnuo sve što naš superjunak može dosegnuti pa se možemo zaustaviti.

Zaključujemo da smijemo prekinuti putovanje superjunaka kada dođemo do već označenog nebodera ako je on označen od nekog višeg superjunaka. Obrađujemo li superjunake od najvišeg do najnižeg, taj će uvjet biti zadovoljen i algoritam će biti dovoljno efikasan jer nijedan neboder nećemo više puta označiti. Što se tiče drugog pitanja iz zadatka, superjunaka možemo ukloniti tako da broj nedostupnih nebodera ostane isti ako taj superjunak u ovom postupku ne označava nijedan novi neboder, tj. ako je neboder na kojemu se on nalazi već označen.

```
n = int(input())
visine = list(map(int, input().split()))
superjunaci = input().split()

suvisni_superjunaci = 0
oznacen = [False for i in range(n)]

# Sortiranjem visina ne smijemo izgubiti originalne pozicije,
# zato sortiramo parove (visina, indeks):
visine_silazno = sorted(
    [(visine[i], i) for i in range(n)], key=lambda x: -x[0])

for v, i in visine_silazno:
    if superjunaci[i] == '1':
        if oznacen[i]:
            suvisni_superjunaci += 1
            continue

        # Putuj lijevo
        j = i
        while j >= 0 and visine[j] <= v and not oznacen[j]:
            oznacen[j] = True
            j -= 1

        # Putuj desno
        j = i + 1
```

```
        while j < n and visine[j] <= v and not oznacen[j]:
            oznacen[j] = True
            j += 1

print(n - sum(oznacen))
print(suvisni_superjunaci)
```