



ZADATAK	ŠIFRAT	BAČVA	TVORNICA	INFRASTRUKTURA
ulazni podaci	standardni ulaz			
izlazni podaci	standardni izlaz			
vremensko ograničenje	1 sec	1 sec	1 sec	1 sec
memorijsko ograničenje	64 MB	64 MB	64 MB	64 MB
broj bodova	100	100	100	100
	400			



Kako je duljina ključa $K \leq 20$ možemo za svaki broj između 1 i 2^K provjeriti je li zapis tog broja u bazi dva mogući šifrat. Ako je, samo ispišemo njegov zapis u bazi dva.

```
int main (void){
    cin >> k;
    cin >> kljuc;
    cin >> x;
    cin >> z;
    for (int i=0;i<pow(2,k);i++){ //generira sva moguća rješenja
        if (provjeri(i)){ //provjerava da li je moguće rješenje pravo rješenje
            cout << baza2(i) << endl; //ispisuje pravo rješenje u odgovarajućem
obliku
        }
    }
    return 0;
}
```

Funkcija *provjeri* prima prirodan broj te uzastopnim dijeljenjem tog broja s dva, određuju znamenke u binarnom zapisu tog broja. Istovremeno prebraja koliko je bilo tih jedinica te koji broj se kreira na osnovu tih jedinica ako se gleda ključ.

```
bool provjeri(int a){
    int zbroj=0; //ukupan zbroj iskorištenih znamenki ključa
    int broj=0; //broj znamenki ključa koje smo uzeli
    for (int i=k-1;i>=0;i--){
        if (a%2==1){
            broj++;
            zbroj+=(kljuc[i]-'0');
        }
        a/=2;
    }
    return (broj==x&&zbroy==z);
}
```

Funkcija *baza2* za zadani prirodan broj vraća njegov zapis u bazi dva.

```
string baza2(int a){
    string s=""; //broj a pretvoren u binarni zapis
    for (int i=0;i<k;i++){
        s+=((a%2)+'0');
        a/=2;
    }
    reverse(s.begin(),s.end()); //potrebno je okrenuti zapis
    return s;
}
```



Na nekoj poziciji bačva se može nalaziti u tri položaja: na bazi, na plaštu tako da se može kotrljati gore-dolje ili na plaštu tako da se može kotrljati lijevo-desno. Pozicija na ploči (trenutni redak i stupac u kojem se bačva nalazi) te položaj bačve jedinstveno opisuje stanje.

Idući korak nam je odrediti cijene prelaska iz pojedinog stanja u ostala. Cijena je jednaka 1 ako bačvu moramo prevrnuti, a 0 ako ju samo kotrljamo. Ispisani su mogući prelasci:

(red, stupac, BAZA) -> (red+1, stupac, LIJEVO_DESNO), cijena = 1

(red, stupac, BAZA) -> (red-1, stupac, LIJEVO_DESNO), cijena = 1

(red, stupac, BAZA) -> (red, stupac+1, GORE_DOLJE), cijena = 1

(red, stupac, BAZA) -> (red, stupac-1, GORE_DOLJE), cijena = 1

(red, stupac, LIJEVO_DESNO) -> (red+1, stupac, BAZA), cijena = 1

(red, stupac, LIJEVO_DESNO) -> (red-1, stupac, BAZA), cijena = 1

(red, stupac, LIJEVO_DESNO) -> (red, stupac+1, LIJEVO_DESNO), cijena = 0

(red, stupac, LIJEVO_DESNO) -> (red, stupac-1, LIJEVO_DESNO), cijena = 0

(red, stupac, GORE_DOLJE) -> (red+1, stupac, GORE_DOLJE), cijena = 0

(red, stupac, GORE_DOLJE) -> (red-1, stupac, GORE_DOLJE), cijena = 0

(red, stupac, GORE_DOLJE) -> (red, stupac+1, BAZA), cijena = 1

(red, stupac, GORE_DOLJE) -> (red, stupac-1, BAZA), cijena = 1

Neki od navedenih prelazaka nisu mogući za retke i stupce na rubu ploče ili pokraj zapreka, no te slučaje lako provjerimo dodatnim ispitivanjem.

Nakon ove transformacije primjećujemo da nas zanima najkraći put u grafu u kojem je čvor predstavljen stanjem (red, stupac, položaj), a čvorovi su povezani bridovima prema gore navedenim prelascima između stanja. Najkraći put u tom grafu možemo pronaći pomoću dijkstrinog algoritma ili pomoću 0-1 bfs-a. 0-1 bfs je adaptacija pretraživanja u širinu u kojem imamo cijene bridova 0 i 1. Kada se širimo iz nekog čvora po bridu cijene 1 idući čvor stavljamo na kraj reda (queue-a), a kada se širimo po bridu cijene 0 idući čvor stavljamo na početak reda (queue-a). 0-1 bfs nudi najbrže rješenje za ovaj zadatak u složenosti $O(n * m)$. Detalje o tom algoritmu pogledajte u službenoj implementaciji rješenja ovog zadatka.

80% bodova se moglo osvojiti običnim bfs-om (pretraživanjem u širinu) tako da svaki put provjerimo jesmo li smanjili cijenu puta na čvor u koji se širimo, a 40% bodova se moglo osvojiti implementacijom bfs-a bez korištenja reda (queue-a).



Pronađimo stroj koji treba najviše poslova. Ako taj stroj treba obaviti k poslova onda nam treba barem k minuta da obavimo sve poslove.

Sljedećim sweep-line algoritmom ćemo rasporediti sve poslove. Definirajmo 2 vrste događaja:

- 1.) početak nekog posla
- 2.) kraj nekog posla

Događajima osim vrste definiramo indeks posla i i indeks stroja. Ako imamo posao i koji zahtjeva strojeve $[a, b]$ onda definiramo dva događaja:

Prvi ima vrstu 1, indeks posla i , indeks stroja a

Drugi ima vrstu 2, indeks posla i , indeks stroja $b + 1$

Održavat ćemo skup "zauzetih minuta" koji nazovemo S . Sada kad sweep-line dođe do događaja tipa 1, pronađe najmanju minutu od 1 do k koja nije u skupu S . Primjetite bitnu činjenicu: Kada bi sve minute od 1 do k bile u skupu S to bi značilo da postoji stroj koji odrađuje barem $k + 1$ poslova što je kontradikcija sa uvjetom na broj k . Dakle, uvijek ćemo moći dodijeliti slobodnu minutu od 1 do k . Tada tu minutu ubacimo u skup S .

Kada sweep-line dođe do događaja tipa 2, izbaci minutu koja pripada tom poslu iz skupa zauzetih minuta.

Kada bi naivno radili pretraživanje slobodnih minuta pri obradi događaja 1, onda bi složenost algoritma bila $O(N^2)$.

Kako bismo postigli složenost $O(N \log N)$ dovoljno je skup zauzetih minuta održavati u binarnom stablu. (struktura set u C++-u).

Za implementacijske detalje pogledajte službeno rješenje.



Promotrimo dva slučaja:

1. Ukoliko bilo koji grad zahtijeva 0 cesta tada zasigurno nije moguće zadovoljiti uvjete s obzirom da taj graf neće moći biti povezan sa ostatkom države.
2. Budući da svaka cesta počinje i završava u točno jednom gradu, ukoliko postoji traženi raspored cesta mora vrijediti $2n - 2 = D_1 + D_2 + \dots + D_n$.

Dakle ukoliko bilo koji od ova dva uvjeta nije zadovoljen treba ispisati NE. Tvrdimo da je u suprotnome odgovor uvijek DA.

Gornji uvjeti osiguravaju da uvijek postoji i takav da je $D_i = 1$. Povežemo li takav i s proizvoljnim j takvim da je $D_j \geq 2$ svodimo problem na analogan sa $n - 1$ gradova. (Uvjet $D_j \geq 2$ je potreban kako bi 1. gore i dalje vrijedio za grad j u novoj državi sa $n - 1$ gradova)

Ponavljajući taj korak dolazimo do slučaja sa $n = 2$ koji zadovoljava uvjete 1. i 2. za koji je jedina mogućnost 2 grada s jednom cestom koja ih spaja.

U implementaciji moramo pratiti sve gradove koji imaju $D_i = 1$ te sve gradove koji imaju $D_i > 1$. Koristeći strukturu queue moguće je implementirati gornji korak u $O(1)$. Prvi tip gradova dodajemo u Q1 a drugi u Q2, ali grad i dodajemo $D_i - 1$ puta te svaki puta kada izbacimo posljednje pojavljivanje grada i u Q2 dodajemo ga u Q1.

Takva implementacija ima složenost $O(n)$.