Seconda gara

Alien ABC

An alien race decided to create a new alphabet (alienabc) based on the English alphabet. For each letter in the English alphabet, they can:

- Exclude the letter entirely.
- Include the letter (e.g., a → a).
- Include the letter in a doubled form (e.g., a → aa) (aa is one letter in their new alphabet).
- Include both the single and double versions of the letter (e.g., a → a aa).

Fonte: https://training.olinfo.it/task/ois_alienabc

They wrote down the alienabc, each alien letter exactly once, in their own, alien alphabetic order. The order of letters might differ from the English alphabetic order, and a letter and its doubled version (e.g., a and aa) do not have to be adjacent in the alienabc. Unfortunately, they did this without any separator.

Can you reconstruct the alienabc? You only have to insert spaces between the alien 'letters'. For example, if they wrote **adccb**, then the alienabc **is clearly "a d cc b"**. The reconstruction might not be unique, for example aaa can be coming from a aa or aa a as well. In such case you can choose any of them. Finally, it can happen that they made some mistake and the given string does not correspond to a valid alienabc.

For example **aabaa could come form aa b aa**, but in an alienabc each ’letter’ must appear at most once. In such cases you should output -1.

Constraints

- $1 \leq T \leq 1000$.
- The length of S is between 1 and 100 (inclusive), and it contains only lowercase English letters.

Esempi

Input	Output
3 abcd eeezzoppoo aabacccbbbd	a b c d e ee zz o pp oo -1

Soluzione && Codice

```
int T;
cin >> T;
int alphabet[26]={0};
bool frequance_alphabet_single[26]={false};
bool frequance_alphabet_double[26]={false};

for (int ii = 0; ii < T; ii++){//O(T)
        string S;
        cin >> S;
        int S_len=S.size();
        bool lasciapassare=true;

        for (int i = 0; i < S_len; i++)//O(len_S)
            ++alphabet[S[i]-'a'];

        for(int i = 0; i < 26 && lasciapassare; i++){//O(1)
            if(alphabet[i]>3){
                cout<<"-1\n";
                lasciapassare=false;
            }
        }
```

```
    if(lasciapassare){//I still could have: xx ... x ... x, which is invalid so -1
        for (int i = 0; i < S_len && lasciapassare; i++){//O(len_S)
            if(i+1<S_len && S[i]==S[i+1]){//for not being out-of-bound
            //double char
                if(!frequance_alphabet_double[S[i]-'a']){
                    frequance_alphabet_double[S[i]-'a']=true;
                    ++i;//skip ahead because I've read also the second char
                } else{
                    cout<<"-1\n";
                    lasciapassare=false;
                }
            } else{//--> S[i] is single and different
                if(!frequance_alphabet_single[S[i]-'a'])//means that there wasnt so far
                    frequance_alphabet_single[S[i]-'a']=true;//means one found for the first time
                else{//means the char was already found, so duplicate, so print(-1)
                    cout<<"-1\n";
                    lasciapassare=false;
                }
            }
    }
```

```
        if(lasciapassare){//printing the valid sentence
            for (int i = 0; i < S_len; i++){//O(len_S)
                if(i<S_len && i+1<S_len && S[i]==S[i+1]){//for not being out-of-bound
                                                        //format: xx
                    cout<<S[i]<<S[i]<<" ";
                    i+=1;
                } else//format: x
                    cout<<S[i]<<" ";
            }
            cout<<'\n';
        }
        lasciapassare=true;
    }
        for (int i = 0; i < 26; i++){//O(1), re-init for next cycle
            alphabet[i]=0;
            frequance_alphabet_single[i]=false;
            frequance_alphabet_double[i]=false;
        }
        lasciapassare=true;
} return 0;
```

Complessità?

```
for (int ii = 0; ii < T; ii++) //--> O(T)
```

```
for (int i = 0; i < S_len; i++) //--> O(len_S)
```

```
for(int i = 0; i < 26 && lasciapassare; i++) //--> O(26), ovvero O(1) (why so?)
```

```
/*ogni istruzione if-else --> O(1)*/
```

costo finale : O(T) _ O(len_S)--> **O(T _ len_S)**

Codice commentato

```
int alphabet[26]={0};//lo uso per contare le frequenze del char
```

Lo uso come contatore così se viene che un char è maggiore di 3, allora sicuramente è non valido (vedremo dopo bene).

Codice commentato

```
bool frequance_alphabet_single[26]={false};
```

Lo uso per contare le frequenze delle lettere aliene "singole"

```
bool frequance_alphabet_double[26]={false};
```

Invece, questo per le lettere aliene formate da doppioni del nostro alfabeto, tipo "xx"

```
for (int i = 0; i < S_len; i++)//O(len_S)
    ++alphabet[S[i]-'a'];
```

Usando S[i] come se fosse un indice, posso memorizzare la frequanza di ogni char **senza dover fare** "per ogni lettera cerca tutte le altre lettere" (ovvero O(n)*O(n-1) --> O(n^2) !!!)

Faccio S[i]-'a' perché se volessi memorizzare la frequanza di 'a' allora ad intuito so che dovrebbe essere nella prima posizione, quindi se ho S[i]=='a' allora faccio S[i](=='a')-'a', che verrà tradotto con 97-97-->0 (per convenzione ASCII, perché ogni char viene visto dal PC come un numero convenzionato).

```
if(alphabet[i]>3){
    cout<<"-1\n";
    lasciapassare=false;
}
```

Li unici formati ammessi sono:

- x - xx - xx x - x xx

Si vede che posso usare una lettera massimo tre volte, ergo se troviamo (in qualsiasi ordine) xxxx..., allora **sicuramente** NON è valido.
Quindi "lasciapassare" diventa false e non posso andare avanti per tale parola e stamparla, quindi ignoro tutto sotto e ricomincio il ciclo.

```
if(lasciapassare)//I still could have: x ... x, which is invalid so -1
```

Se siamo arrivatu a questo punto, significa che **100%** abbiamo al massimo 3 ripetizione di un char nella nostra lingua. MA il codice sopra **ripete 'x' due volte, il che per definizione NON è valido**. Quindi devo controllare con un booleano se per ogni lettera aliena (singole e doppione) esiste un doppione.

```
if(i+1<S_len && S[i]==S[i+1])//for not being out-of-bound
```

Prima verifico che "i+1" sia accessibile nel vettore.
Se il if è vero, significa che il char corrente e quello successivo sono uguali, ovvero ho trovato un doppione (double char).

```
//double char
if(!frequance_alphabet_double[S[i]-'a']){
    frequance_alphabet_double[S[i]-'a']=true;
    ++i;//skip ahead because I've read also the second char
} else{
    cout<<"-1\n";
    lasciapassare=false;
}
```

...

...

```
if(!frequance_alphabet_double[S[i]-'a'])
```

Significa che se la sua frequanza/contatore era zero/false, allora dichiaro che l'ho visto ora e lo metto a true.

```
//Poi faccio:
++i;
```

per muovermi al char successivo cosicché nel successivo for vado al char di DUE posizioni avanti, perché virtualmente ho già considerato sia quello attuale (i) ma anche il successivo nel if (i+1).

```
else{
    cout<<"-1\n";
    lasciapassare=false;
}
```

Significa che, se siamo in questo else, tale lettera aliena era già stata vista, **ovvero questo attuale è una ripetizione (non sono ammesse per definizione del problema).**

Quindi print(-1) e settiamo lasciapassare=false.

NB: [facciamo lo stesso per "frequance_alphabet_single"]

```
if(lasciapassare) //printing the valid sentence
```

Se siamo arrivatu fino a qui, vuol dire che lasciapassare==true, ergo possiamo continuare (ovvero "è valida" la parola/frase aliena).
Allora procedo a stamparla.

...

...

```
if(i<S_len && i+1<S_len && S[i]==S[i+1]){
    //format: xx
    cout<<S[i]<<S[i]<<" ";
    i+=1;
}
```

Se S[i]==S[i+1] (ovvero una doppia), allora li stampo senza spazio e poi lo spazio.

```
else//format: x
    cout<<S[i]<<" ";
```

Se siamo qui, allora è una singola e la stampo.

```
//re-init for next cycle
for (int i = 0; i < 26; i++){//O(1)
    alphabet[i]=0;
    frequance_alphabet_single[i]=false;
    frequance_alphabet_double[i]=false;
}
lasciapassare=true;
```

Se sono arrivatu qui, allora faccio un ciclo per resettare tutto per il prossimo ciclo/valutazione della parola successiva.

Killer Cages

Valerio is a big fan of different sudokus, despite being bad at solving them. He is particularly interested in those with killer cages. A killer cage is a region where all the numbers must be distinct and have a certain sum.
Valerio wants to get better at sudoku, so he is asking for your advice to improve. He will make you T questions of the following form:
Given N and K, is there a unique way to write N as sum of K distinct positive integers?

Fonte: https://training.olinfo.it/task/ois_killer

Constraints

- $1 \leq T \leq 10\,000$.
- $1 \leq N \leq 1\,000\,000\,000$.
- $1 \leq K \leq 1\,000\,000\,000$.

Esempi

Input	Output
6	
9 1	YES
5 2	NO
4 2	YES
11 4	YES
8 3	NO
12 6	NO

Soluzione && Codice

```
int T;
cin >> T;
for (int i = 0; i < T; ++i) {//O(T)
    unsigned long long N, K;
    cin >> N >> K;

    if(K==1) cout<<"YES\n";//why?
    else{
        unsigned long long sum_up_to_k=0;
        /*for (int j = 1; j <= K; j++)//O(K)
            sum_up_to_k+=j;
        */
        //instead use Gauss method:
        ++K;
        sum_up_to_k=(K*(K-1))/2;//O(1) and not O(K)

        if(sum_up_to_k==N || sum_up_to_k==N-1) cout<<"YES\n";
        else cout<<"NO\n";
    }
}
```

Complessità

```
for (int i = 0; i < T; ++i) //O(T)
```

Tutto qui, il resto è O(1).

Costo finale : O(T) * O(1)--> **O(T)**

NB: tale costo mooolto economico è possibile SOLAMENTE grazie al metodo del mitico Gauss per sommare i numeri da 1 a K (vedremo dopo).

Se non avessi saputo fare tale roba sarebbe costato O(T*K) !!!

```
unsigned long long N, K;
```

I ULL sefrvono perché alla peggio farei K*(K-1), ovvero solo 1'000'000'000*1'000'000'000 = 10^18

```
if(K==1) cout<<"YES\n";//why?
```

Prendiamo il caso d'esempio: 9 1

Si deduce che SEMPRE posso raggiungere il numero 9 con un solo numero...ovvero il numero in sé !!!

E questo vale per tutti i numeri se K==1.

```
for (int j = 1; j <= K; j++)//O(K)
    sum_up_to_k+=j;
```

Potrei sommare da 0 --> K con un for normale ... ma vi sfido (letteralmente scrivete il codice) a farlo.

...

(... nel mio calcolatore con il seguente codice)

```
unsigned long long res=0*1ULL;
for (unsigned long long i = 0; i < 10000000000*1ULL; i++){
    res+=i;
}
cout<<"Res: "<<res<<endl;
/*
> time ./a.out
Res: 49999999500000000

real    0m3.237s
user    0m3.234s
sys     0m0.001s

--> 3 SECONDIIIIIIIII, e ne avete 1 a disposizione.
ergo ecco a cosa serve la matematica ;)
*/
```

Ma perché devo sommare da 1 a K?

Esempio: 92 13
sum_up_to_13 --> 91, ergo basta che prenda la sommatoria appena fatta: 1+2+3+4+5+6+7+8+9+10+11+12+13 e cambiare un numero qualsiasi incrementandolo.

Esempio: 5 2, sum_up_to_2 --> 1+2=3, putroppo qui per arrivare a 5 da 3 devo aggiungere 2. MA a chi? Infatti potrei aggiungerlo sia al 1 che al 2, **quindi NON avrei una scelta UNICA**.

Da qui **deduciamo che se sum_up_to_k è minore di al massimo un numero da N, allora basta cambiarne esattamente uno, quindi una combinazione unica**.

```
++K;
sum_up_to_k=(K*(K-1))/2; //O(1) and not O(K)
```

Formula (una delle tante) di Gauss:

“ S = Sommatoria da k=1 ad n --> n*(n+1)/2

”

P.S.: se ci tenete **facciamo la dimostrazione in classe**

Fonte: [dimostrazione del metodo della sommatoria di Gauss]

```
if(sum_up_to_k==N || sum_up_to_k==N-1) cout<<"YES\n";
else cout<<"NO\n";
```

Se la sum_up_to_k vale già N, allora non dobbiamo fare nulla, ovvero che i K numeri bastano per scrivere la cifra N.

Se sum_up_to_k+1 vale N, allora significa che è bastato aggiungere un 1, ovvero basta prendere un dei K numeri e cambiarlo aggiungendoci 1.