LuissMatics 2020
Testi e Soluzioni

LuissMatics

25–26 Aprile 2020

Problemi a cura di

Irene Finocchi, Giuseppe F. Italiano, Nicola Prezza, Fangqing Yuan

Testi dei problemi

Irene Finocchi, Giuseppe F. Italiano, Michele Lizzit, Nicola Prezza, Fangqing Yuan

Soluzioni dei problemi

Carlo Malagnino

Gestione della gara online

Michele Lizzit, Edoardo Morassutto, William Di Luigi, Fangqing Yuan, Francesco Redaelli

Sistema di gara

Terry: `https://github.com/algorithm-ninja/terry`.

Organizzatori della gara

Giuseppe F. Italiano, Luigi Laura

Testi e soluzioni dei problemi

Indice

Introduzione

LuissMatics 2020 si è svolta quest'anno il 24-25 aprile 2020. LuissMatics (Luiss Informatics, `https://www.luiss.it/admissions/programs-offered/management-and-computer-science/luissmatics`) è una gara on line di programmazione che continua la tradizione di Gator (`http://people.uniroma2.it/giuseppe.italiano/gator/`). In particolare, la prima GATOR si è svolta il 29-30 marzo 2014 e, dopo quattro edizioni, è diventata LuissMatics nel 2018. LuissMatics è aperta a tutti gli studenti delle scuole secondarie superiori e permette ai partecipanti di allenarsi in vista delle Olimpiadi Italiane di Informatica, che selezionano i migliori studenti tra tutte le scuole secondarie superiori italiane.

La gara è stata attiva per 48 ore sul Portale di Allenamento delle Olimpiadi Italiane di Informatica: dalle 00:01 di sabato 25 aprile alle 23:59 di domenica 26 aprile 2020. I partecipanti potevano accederci dal link `https://luissmatics.olinfo.it/`. Anche quest'anno Luiss assegnerà, in connessione con LuissMatics, una borsa di studio – esonero totale dal contributo unico – per il corso di Laurea Triennale in inglese in Management and Computer Science.

LuissMatics 2020 era composta di 4 problemi da risolvere in 5 ore, calcolate a partire da quando un partecipante si sarebbe collegato al server della gara. È stato quindi possibile scegliere le cinque ore più adatte alla gara in funzione dei propri impegni. Il livello di difficoltà dei problemi è stato in linea con i problemi della Selezione Territoriale delle Olimpiadi di Informatica. LuissMatics 2020 ha visto un totale di 587 partecipanti, di cui 39 (il 6.64%) a punteggio pieno.

Per qualsiasi informazione o domanda è possibile contattare gli organizzatori della gara: Prof. Giuseppe F. Italiano (`gitaliano@luiss.it`) e Prof. Luigi Laura (`llaura@luiss.it`).

Arrivederci a LuissMatics 2021!

Parigi (Parigi)

Difficoltà: *1*

Descrizione del problema

Michele è stato il primo vincitore della borsa di studio **LuissMatics** per frequentare il **Bachelor in Management and Computer Science** alla Luiss. Durante il suo terzo anno di studi, è stato ammesso allo scambio con l'Università di Paris Dauphine. Michele adora **Parigi**, sin da quando ha gareggiato per SWERC 2019! Ogni volta che si trova a Parigi, Michele cerca di visitare più monumenti possibili e ogni giorno annota nel suo diario quali monumenti ha visitato in quel giorno. Dopo vari viaggi e vari giorni passati a Parigi, Michele si chiede quanti monumenti diversi ha visitato in totale durante la sua permanenza a Parigi.

Aiuta Michele scrivendo un programma che, dato il suo diario, calcoli il **numero di monumenti diversi** che ha visitato durante tutti i suoi viaggi a Parigi.

Dati di input

La prima riga del file di input contiene un intero T, il numero di testcase.

Le successive righe mostrano i T testcase in ordine. Ogni testcase è composto da $N + 1$ righe:

- La prima riga contiene l'intero N, il numero di monumenti che Michele ha visitato durante tutti i suoi viaggi a Parigi.
- Le successive N righe contengono prima una data (nel formato YYYY-MM-DD), poi uno spazio, e infine il nome di un monumento. I nomi dei monumenti sono stringhe di caratteri di lunghezza arbitraria e possono contenere qualsiasi carattere stampabile, ad eccezione del carattere newline.

Dati di output

Il file di output deve essere composto da T righe, ciascuna delle quali contenente la dicitura `Case #x: y` dove `x` è il numero del testcase (a partire da 1) e `y` è un intero positivo (o nullo): il numero di monumenti diversi visitati da Michele.

Assunzioni

- $1 \leq T \leq 10$.
- $1 \leq N \leq 100$.

Esempi di input/output

Input:

2

9

2019-12-30 Tour Eiffel

2019-12-31 Tour Saint-Jacques

2019-12-31 Centre Georges Pompidou

2020-01-23 Tour Eiffel

2020-01-23 Invalides

2020-01-23 Arc de Triomphe

2020-01-26 Tour Saint-Jacques

2020-01-26 Panthéon

2020-01-27 Sacré Cœur

5

2019-12-29 Tour Eiffel

2019-01-05 Tour Saint-Jacques

2020-01-05 Tour Eiffel

2020-01-05 Arc de Triomphe

2020-01-06 Tour Saint-Jacques

Output:

```
Case #1:  7
Case #2:  3
```

Soluzione

Per risolvere il problema bisogna individuare il numero di monumenti diversi visitati da Michele durante tutti i suoi viaggi a Parigi. Occorre perciò analizzare le visite singolarmente, aggiornando l'insieme dei monumenti diversi incontrati. Per ogni monumento incontrato (indipendentemente dalla data in cui esso sia stato visto) basta controllare se è già presente nell'insieme, nel caso in cui non ci fosse deve essere aggiunto. Una struttura dati `set` è sufficiente per i nostri scopi. Alla fine è sufficiente stampare il numero di monumenti diversi visitati.

```cpp
#include<bits/stdc++.h>

using namespace std;

int main(){

    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);

    int T, N;
    cin >> T;
    for(int t=1;t<=T;t++){
        int ris=0;
        cin >> N;
        getchar();
        set<string>l;
        for(int i=0;i<N;i++){
            string s;
            getline(cin, s);
            s=s.substr(11);
            if(l.count(s)==0){
                l.insert(s);
                ris++;
            }
        }
        l.clear();
        cout << "Case #" << t << ": " << ris << "\n";
    }
}
```

Covid (Covid)

Difficoltà: *2*

Descrizione del problema

Giuseppe e Irene sono molto preoccupati per la pandemia da COVID-19, e vogliono mettere a frutto le loro competenze accademiche di data scientist per costruire un'app che possa essere utile a diminuire il contagio. L'app che hanno in mente permette di monitorare il numero di casi positivi rilevati in vari luoghi fornendo una serie di informazioni utili, tra cui una stima della probabilità di contagio di una persona a seconda che viva più o meno vicino ad un alto numero di persone già infette.

Scaricando dati dalla rete, Giuseppe e Irene si procurano innanzitutto una mappa dei contagi, oltre ad altre informazioni sulla residenza delle persone da monitorare. Hanno quindi a disposizione i seguenti dati:

- N = numero di persone da monitorare
- N coppie (X_i, Y_i), con $1 \leq i \leq N$, che rappresentano il luogo in cui risiede ciascuna delle N persone
- Z = numero di punti geografici per cui si dispone di una stima del numero di casi positivi rilevati in quel luogo
- La mappa dei contagi, ovvero Z triple (X_j, Y_j, P_j), con $1 \leq j \leq Z$, che rappresentano una coordinata geografica (X_j, Y_j) e il numero P_j di casi positivi rilevati in quel luogo
- Una distanza $D > 0$

In modo molto semplificato, assumono che la probabilità di infettarsi di una persona che vive nel luogo con coordinate (X, Y) sia tanto maggiore quanto maggiore è il numero totale di casi positivi rilevati a una distanza $\leq D$ da (X, Y).

Ricorda che la distanza (Euclidea) tra due punti (X, Y) e (A, B) può essere stimata come $\sqrt{(X-A)^2 + (Y-B)^2}$.

Il tuo compito è di aiutare Giuseppe e Irene a sviluppare l'app, identificando, tra le N persone, ***chi ha la maggiore probabilità di sviluppare il contagio***.

Dati di input

La prima riga del file di input contiene un intero T, il numero di testcase. Le successive righe mostrano i T testcase in ordine. Per ogni testcase:

- La prima riga contiene l'intero N
- Le successive N righe contengono le coppie X_i, Y_i, ciascuna su una riga diversa, rappresentate come due interi separati da uno spazio, senza parentesi
- La riga successiva contiene l'intero Z
- Le successive Z righe contengono le triple X_j, Y_j, P_j, ciascuna su una riga diversa, rappresentate come tre interi separati da uno spazio, senza parentesi

- La riga successiva contiene l'intero D

Dati di output

Il file di output deve essere composto da T righe, ciascuna delle quali contenente la dicitura `Case #x: y` dove `x` è il numero del testcase (a partire da 1) e `y` è un numero tra 1 ed N che rappresenta la persona con la massima probabilità di contagio (si può assumere che tale persona sia unica).

Assunzioni

- $1 \leq T \leq 10$
- $1 \leq N \leq 100$
- $1 \leq X \leq 100$, $1 \leq Y \leq 100$
- $1 \leq D \leq 10$
- $1 \leq P \leq 10$
- $1 \leq Z \leq 100$

Esempi di input/output

Input:

1

3

1 1

3 3

5 5

4

0 1 5

2 3 10

8 6 2

4 4 15

2

Output:

```
Case #1:  2
```

Spiegazione

- La persona 1, che vive in $(1, 1)$, si trova a distanza 1 da $(0, 1)$, a distanza $\sqrt{5} = 2.23$ da $(2, 3)$ e a distanza ancora maggiore dai punti $(4, 4)$ e $(8, 6)$. Quindi il numero di casi positivi a distanza al più 2 dalla persona 1 è pari a 5, ovvero il numero di casi positivi rilevati in $(0, 1)$.
- La persona 2, che vive in $(3, 3)$, si trova a distanza 1 da $(2, 3)$, a distanza $\sqrt{2} = 1.41$ da $(4, 4)$ e a distanza maggiore di 2 dai punti $(0, 1)$ e $(8, 6)$. Quindi il numero di casi positivi a distanza al più 2 dalla persona 2 è pari a $10 + 15 = 25$.
- La persona 3, che vive in $(5, 5)$, si trova a distanza $\sqrt{2} = 1.41$ da $(4, 4)$ e a distanza maggiore di 2 da tutti gli altri punti. Quindi il numero di casi positivi a distanza al più 2 dalla persona 3 è pari a 15.

La persona con il maggior numero di casi positivi nel suo vicinato è quindi la persona 2.

Soluzione

Per risolvere il problema bisogna individuare la persona che ha la maggiore possibilità di sviluppare il contagio, cioè quella che ha il numero massimo di casi positivi a una distanza minore di quella data (D). Per individuarla bisogna determinare per ogni persona il numero di casi positivi che rispettano tale condizione (calcolando la distanza con il teorema di Pitagora e confrontandola con D). Stampare quindi il numero corrispondente alla persona per cui il valore calcolato è massimo.

```cpp
#include<bits/stdc++.h>

#define NMAX 110

using namespace std;

int T,N,Z,D;
int X[NMAX],Y[NMAX],Xz[NMAX],Yz[NMAX],Cz[NMAX];

int main(){

    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);

    cin >> T;
    for(int t=1;t<=T;t++){
        cin >> N;
        for(int i=0;i<N;i++){
            cin >> X[i] >> Y[i];
        }

        cin >> Z;
        for(int i=0;i<Z;i++){
            cin >> Xz[i] >> Yz[i] >> Cz[i];
        }
        cin >> D;

        int massi=0, massip=0, cont;
        for(int i=0;i<N;i++){
            cont=0;
            for(int j=0;j<Z;j++){
                double dist=sqrt(pow(X[i]-Xz[j], 2) + pow(Y[i]-Yz[j], 2));
                if(dist<=D){
                    cont+=Cz[j];
                }
            }
            if(massi<cont){
                massi=cont;
                massip=i;
            }
        }
        cout << "Case #" << t << ": " << massip+1 << "\n";
    }
}
```

Fred (`Fred`)

Difficoltà: 3

Descrizione del problema

Data Science è alla base dei Social Media moderni. Non solo le piattaforme possono raccogliere ed analizzare i dati degli utenti, ma anche gli stessi utenti possono scoprire qualcosa di nuovo processando i dati pubblicamente accessibili.

"**LuissZone**" è una nuova piattaforma social. Questa piattaforma consente agli utenti di postare articoli su "LuissZone", e ogni utente, se vuole, può mettere like a ciascun post presente su "LuissZone". Tutti i post e tutti i 'like' sono pubblici e visibili a tutti gli utenti iscritti a "LuissZone".

Fred ama analizzare i dati provenienti da "LuissZone". Fred è particolarmente interessato nel determinare l'utente più popolare di una data piattaforma. Dopo anni di ricerca ha determinato 3 importanti parametri per decidere la popolarità di un utente:

- Il numero medio di parole nei post di un dato utente, denotato da $L = \frac{\sum L_i}{N}$, dove N è il numero di post che l'utente ha postato e L_i è il numero di parole di ciascun post.
- Il numero totale di 'like' che l'utente ha ricevuto da altri utenti (escludendo i like messi dall'utente ai suoi post), denotato da R.
- numero totale di like che l'utente ha aggiunto a post su "LuissZone" (includendo i like messi ai propri post), denotato da G.

Con questi tre parametri Fred ha inventato il '**Fred Coefficient**': $F = \frac{LR}{G+1}$

Il testo di ciascun post è composto da lettere e spazi. Una sequenza di lettere separata da spazi (o separata da uno spazio se a inizio/fine riga) è considerata una parola.

L'utente con il maggiore 'Fred Coefficient' è definito come l'utente più popolare. Nel caso di più utenti con lo stesso 'Fred Coefficient', l'utente più popolare è l'utente che viene prima in ordine alfabetico.

Il tuo compito ora è trovare **l'utente più popolare** in un dato dataset "LuissZone", determinato calcolando il 'Fred Coefficient'.

Dati di input

La prima riga del file di input contiene un intero T, il numero di testcase.

Le righe seguenti contengono i T testcase in ordine. Per ogni testcase:

- La prima riga è composta da: M, un intero, che rappresenta il numero di utenti in "LuissZone".
- La seconda riga è composta dai nomi degli utenti separati da spazi.
- La terza riga è composta da: N_1, N_2....N_M, interi separati da spazio, il numero di post postati da ogni utente, nello stesso ordine degli utenti indicati nella seconda riga.

- Le seguenti $2\sum N_i$ righe: il numero di post e 'like' di ciascun utente indicati in base all'ordine fornito nella seconda riga. Per ogni post, la prima riga è una stringa con il testo del post, la seconda riga contiene i vari username separati da spazio, che indicano gli utenti che hanno messo like al post. (è possibile che un post non abbia ricevuto 'like' e abbia quindi 0 'like')

Dati di output

Il file di output deve essere composto da T righe, ciascuna delle quali contenente la dicitura Case x: y dove x è il numero del testcase (a partire da 1) e y è una stringa che rappresenta l'utente più popolare del testcase.

Assunzioni

- $1 \leq T \leq 5$.
- $0 \leq M \leq 100$.
- $1 \leq N_i \leq 10$, per $1 \leq i \leq M$.
- Il numero di parole di ciascun post è al più 50. (Per ogni post, $0 \leq L \leq 100$).

Esempi di input/output

Input:

2

2

Fred Michele

1 1

Ciao

Michele

A presto

Fred

3

Fred Francesco Michele

2 1 1

Hello world

Michele

Luiss is great

Fred Michele Francesco

Go Taddy Bear!

Fred

Minecraft

Fred Francesco Michele

Output:

```
Case #1: Michele
Case #2: Fred
```

Spiegazione

`Case #1:`

Il 'Fred Coefficient' dei due utenti è:

- Fred: ha un post con numero medio di parole 1, ha ricevuto 1 like da utenti diversi da se, e ha messo 1 like in totale. $1 * 1/(1 + 1) = 0.5$
- Michele: ha un post con numero medio di parole 2, ha ricevuto 1 like da utenti diversi da se, e ha messo 1 like in totale. $1 * 1/(1 + 1) = 1$

`Case #2:`

Il 'Fred Coefficient' dei tre utenti è:

- Fred: ha due post con numero medio di parole 2.5, ha ricevuto 3 like da utenti diversi da se, e ha messo 3 like in totale. $2.5 * 3/(3 + 1) = 15/8$
- Francesco: ha un post con numero medio di parole 3, ha ricevuto 1 like da utenti diversi da se, e ha messo 2 like in totale. $3 * 1/(2 + 1) = 1$
- Michele: ha un post con numero medio di parole 1, ha ricevuto 2 like da utenti diversi da se, e ha messo 3 like in totale. $1 * 2/(3 + 1) = 0.5$

Soluzione

Per risolvere il problema bisogna individuare l'utente del dataset il cui 'Fred Coefficient' è massimo. Occorre perciò calcolare per ogni utente il numero di parole scritte nei propri post, il numero di like assegnati e il numero di like ricevuti (prestando attenzione a non contare i like assegnati dagli utenti ai loro stessi post). Bisogna individuare il valore di 'Fred Coefficient' massimo tra gli utenti (utilizzando la formula $F = \frac{LR}{G+1}$). Stampare quindi il nome dell'utente il cui nome viene prima in ordine alfabetico tra quelli che possiedono tale coefficiente massimo.

```cpp
#include<bits/stdc++.h>

using namespace std;

int main(){

    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);

    int T, M;

    cin >> T;

    for(int t=1;t<=T;t++){

        cin >> M;

        vector<string>U(M);
        vector<int>nP(M);
        vector<int>par(M, 0);
        vector<int>G(M, 0);
        vector<int>R(M, 0);

        for(int i=0;i<M;i++){
            cin >> U[i];
        }
        for(int i=0;i<M;i++){
            cin >> nP[i];
        }
        getchar();
        for(int i=0;i<M;i++){
            for(int j=0;j<nP[i];j++){
                string post;
                int parole=0;
                getline(cin, post);
                for(int c=0;c<post.size();c++){
                    if(post[c]==' '){
                        parole++;
                    }
                }
                if(post.size()>0){
                    parole++;
                }
                par[i]+=parole;

                string like;
                getline(cin, like);
                int last=0;
                vector<string>utL;
```

```
50                  for(int c=0;c<like.size();c++){
51                      if(like[c]==' '){
52                          utL.push_back(like.substr(last,c-last));
53                          last=c+1;
54                      }
55                  }
56                  if(like.size()>0){
57                      utL.push_back(like.substr(last));
58                  }
59                  for(int l=0;l<utL.size();l++){
60                      for(int u=0;u<M;u++){
61                          if(U[u]==utL[l]){
62                              G[u]++;
63                              if(U[u]!= U[i]) R[i]++;
64                          }
65                      }
66                  }
67              }
68          }
69          double masF=-1, fred;
70          string masU;
71          for(int i=0;i<M;i++){
72              fred=double(double(par[i])/double(nP[i])*double(R[i])/double(G[i]+1));
73              if(fred>masF || (fred==masF && masU>U[i])){
74                  masF=fred;
75                  masU=U[i];
76              }
77          }
78          cout << "Case #" << t << ": " << masU << "\n";
79      }
80  }
```

Asseblaggio (`Assemblaggio`)

Difficoltà: 4

Descrizione del problema

Negli ultimi due decenni, la biologia è diventata una nuova fonte di **Big Data** a causa delle nuove tecnologie di sequenziamento di DNA. Nel 2003, il completamento del progetto genoma umano ha permesso agli scienziati di ricostruire approssimativamente il DNA della specie umana: una lunga sequenza (stringa) composta da lettere A, C, G, T (Adenina, Citosina, Guanina, Timina). Il passo successivo a questo importante traguardo è quello di ricostruire il DNA di singole persone. Luigi e Nicola sono i due nuovi data scientist assunti dalla Watson SpA. Il loro compito è di ricostruire la sequenza di DNA di Luigi usando gli strumenti a disposizione in laboratorio. Per prima cosa, i due scienziati estratto del DNA dal bulbo di un capello di Luigi; dopodichè, lo hanno dato in pasto ai sequenziatori in laboratorio: macchine in grado di "leggere" il DNA. I rudimentali sequenziatori del laboratorio, però, riescono a "leggere" sequenze lunghe al massimo 20 lettere. Gli scienziati hanno quindi dovuto organizzare una sorta di "puzzle":

- Hanno creato diverse copie del DNA di Luigi,
- Hanno spezzettato queste copie in pezzettini di lunghezza al più 20,
- Hanno letto i pezzettini con un sequenziatore.

Dopo un po' di lavoro, gli scienziati sono finalmente riusciti a leggere una sequenza di N pezzettini di DNA $P_1, P_2, ..., P_N$ con queste proprietà:

- Per ogni $1 \leq i < j \leq N$, il pezzettino P_i è stato letto dal DNA di Luigi a partire da una posizione p_i strettamente più a sinistra della posizione p_j dalla quale è stato letto il pezzettino P_j (ossia $p_i < p_j$).
- Ogni parte del DNA di Luigi è "coperta" da almeno un pezzettino.
- Pezzettini consecutivi possono accavallarsi in ogni modo, o possono non accavallarsi affatto.

Aiuta i due scienziati a ricostruire "la sequenza di DNA più corta" dalla quale i pezzettini possono essere stati estratti.

Dati di Input

La prima riga del file di input contiene un intero T, il numero di testcase. Le successive righe mostrano i T testcase in ordine. Ogni testcase è composto da $N + 1$ righe:

- La prima riga contiene l'intero N, il numero di pezzettini di DNA letti dal sequenziatore.
- Le successive N righe contengono gli N pezzettini $P_1, ..., P_N$ (ossia, stringhe di lunghezza al più 20 sull'alfabeto A, C, G, T).

Dati di Output

Il file di output deve essere composto da T righe, ciascuna delle quali contenente la dicitura Case x: y dove x è il numero del testcase (a partire da 1) e y è la lunghezza della sequenza di DNA più corta compatibile con gli N pezzettini in input.

Assunzioni

- $1 \leq T \leq 20$
- $1 \leq N \leq 10000$
- $|Pi| \leq 20$ ($|Pi|$ è la lunghezza dell'i-esimo pezzettino).

Esempi di input/output

Input:

2

6

AAACTACGACT

ACTAC

ACGACTAC

ACTACGG

CGGA

TTG

4

GATGAT

GATGTT

ATGT

GTTT

Output:

```
Case #1:  19
Case #2:  10
```

Spiegazione

`Case #1:`

Nel primo esempio mostrato, l'allineamento ottimale dei pezzettini è:

```
AAACTACGACT
  ACTAC
     ACGACTAC
           ACTACGG
               CGGA
                   TTG
```

e la sequenza risultante è quindi AAACTACGACTACGGATTG, di lunghezza 19. Notare che esistono pezzettini completamente "inglobati" da altri pezzettini e che due pezzettini consecutivi non devono necessariamente accavallarsi.

`Case #2:`

Nel secondo esempio, l'allineamento ottimale dei pezzettini è:

```
GATGAT
   GATGTT
    ATGT
      GTTT
```

e la sequenza risultante è quindi GATGATGTTT, di lunghezza 10.

Soluzione

Per risolvere il problema bisogna individuare la lunghezza della sequenza di DNA più corta compatibile. I vari frammenti ci vengono forniti in modo ordinato, secondo la posizione dalla quale vengono letti dal DNA di Luigi. Tale caratteristica ci permette di applicare un approccio greedy al problema. Per ognuno bisogna, infatti, individuare la posizione ottimale in cui inserire il frammento (il più a sinistra possibile) che sia però maggiore della precedente (per il primo frammento la posizione ottimale è 0), aggiungendo alla sequenza iniziale la parte del frammento che non si sovrappone, se presente. Così facendo ad ogni iterazione si individua la sequenza ottima (la più corta compatibile), fino a quel punto. Stampare quindi la soluzione della sequenza risultante.

```cpp
#include<bits/stdc++.h>

using namespace std;

int main(){

    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);

    int T, N;
    string ris;

    cin >> T;
    for(int t=1;t<=T;t++){
        cin >> N;
        cin >> ris;

        int pos=0, last=0, j;

        for(int i=1;i<N;i++){
            string s;
            cin >> s;

            bool st=false;
            for(j=last+1;j<ris.size() && !st;j++){
                st=true;
                for(pos=0;pos<s.size() && j+pos<ris.size() && st;pos++){
                    if(s[pos]!=ris[j+pos])
                        st=false;
                }
            }
            if(st && pos==s.size()){
                    last=j-1;
            }
            if(st && pos<s.size() && j-1+pos==ris.size()){
                ris=ris+s.substr(pos);
                last=j-1;
            }
            if(!st){
                last=ris.size();
                ris=ris+s;
            }
        }
        cout << "Case #" << t << ": " << ris.size() << "\n";
    }
}
```