Cuprins

Date de ieşire

Fişierul `desert.out` conţine o singură linie pe care este scris un număr real reprezentând distanţa maximă ce poate fi parcursă de Indiana Jones, exprimată în kilometri.

Restricţii

- `1≤N≤100`
- `2≤K≤50`
- `5≤P≤20`
- Distanţa maximă parcursă va fi afişată cu trei zecimale, cu rotunjire.

Exemple

desert.in	desert.out	desert.in	desert.out
2 3 10	60.000	4 3 10	76.000

Olimpiada Municipală de Informatică, Iaşi, 2003

Soluţie

Pentru a calcula distanţa maximă care poate fi parcursă având la dispoziţie `n` recipiente vom utiliza o funcţie recursivă `dist()`.

Dacă la oază există un recipient, acesta va fi turnat în rezervor, iar distanţa maximă ce poate fi parcursă cu el este `k/p`. Dacă la oază există două recipiente, unul va fi pus în rezervor, celălalt va fi pus în maşină şi astfel se parcurge distanţa `2*k/p`.

Dacă numărul de recipiente este mai mare de două, turnăm conţinutul unui recipient în rezervor, al doilea recipient îl punem în maşină şi parcurgem o distanţă `x`, lăsăm recipientul plin aici, după care ne întoarcem la oază. La oază punem în maşină alt recipient, parcurgem aceeaşi distanţă `x`, lăsăm recipientul plin şi revenim la oază. După `2n-3` astfel de drumuri, la distanţa `x` de oază se află `n-1` recipiente pline. Vom calcula distanţa `x` astfel încât după `2n-3` drumuri de lungime `x` rezervorul maşinii să fie gol (`x=k/((2*n-3)*p)`). Suntem într-o situaţie similară cu cea iniţială: rezervorul maşinii este gol, avem `n-1` recipiente cu benzină, dar am parcurs deja o distanţă `x`.

```
#include<stdio.h>
#define InFile "desert.in"
#define OutFile "desert.out"
int n, k;
long double p;

long double dist(int n)
{ if (n<=2)  return k*n/p;
  return k/((2*n-3)*p)+dist(n-1); }

int main()
{ FILE *fin, *fout;
  long double d;
```

```
fin=fopen(InFile,"rt");
fscanf(fin, "%d %d %Lf", &n, &k, &p);
fclose(fin);
p=p/100;
d=dist(n);
fout=fopen(OutFile, "wt");
fprintf(fout, "%.3Lf\n", d);
fclose(fout);
return 0; }
```

Windows

Vasile foloseşte un sistem de operare care deschide pe ecran numeroase ferestre. Ecranul este împărţit în pătrate elementare (care au aria `1*1`), formând un caroiaj în care liniile sunt numerotate de la `1` de sus în jos, iar coloanele sunt numerotate de la `1` de la stânga la dreapta. Astfel, fiecare pătrat elementar de pe ecran poate fi identificat specificând numărul liniei şi numărul coloanei pe care se află. Fiecare fereastră este un dreptunghi format din unul sau mai multe pătrate elementare. O fereastră nou deschisă poate să se suprapună (parţial sau total) peste alte ferestre, deschise în prealabil. Putem închide o fereastră dacă executăm un click în pătratul elementar ce constituie colţul din dreapta-sus al ferestrei (dacă acesta este vizibil).

Scrieţi un program care să determine numărul minim de click-uri necesare pentru a închide prima fereastra pe care am deschis-o.

Date de intrare

Fişierul de intrare `win.in` conţine pe prima linie un număr natural `N`, reprezentând numărul de ferestre deschise pe ecran.

Fiecare dintre următoarele `N` linii conţine câte patru numere naturale separate prin câte un spaţiu, `R1`, `S1`, `R2` şi `S2`, cu semnificaţia „am deschis o fereastră care are colţul din stânga-sus pe linia `R1` şi coloana `S1`, respectiv colţul din dreapta-jos pe linia `R2` şi coloana `S2`". Ferestrele se deschid în ordinea în care apar în fişierul de intrare.

Date de ieşire

Fişierul de ieşire `win.out` va conţine o singură linie pe care va fi scris numărul minim de click-uri necesare pentru a închide prima fereastră deschisă.

Restricţii

- $0 < N \leq 100$
- $1 \leq R1 \leq R2 \leq 10000$
- $1 \leq S1 \leq S2 \leq 10000$

Exemple

win.in	win.out	win.in	win.out	win.in	win.out
3 3 1 6 4 1 2 4 6 2 3 5 5	3	3 4 1 6 3 2 2 5 5 1 4 3 6	3	3 3 3 4 4 1 1 2 2 5 5 6 6	1

Olimpiada Municipală de Informatică, Iaşi, 2005

Soluţie

Vom reprezenta o fereastră ca o structură cu patru câmpuri: coordonatele colţului din stânga-sus şi coordonatele colţului din dreapta-jos al ferestrei.

Vom aborda problema recursiv: pentru a închide fereastra cu numărul `nr` trebuie să închidem, în prealabil, toate ferestrele deschise ulterior, care acoperă colţul din dreapta-sus al ferestrei `nr`. Acest lucru este realizat de funcţia recursivă `inchide()`, ce are ca parametru numărul ferestrei care trebuie să fie închisă. Pentru a nu închide de mai multe ori aceeaşi fereastră, vom reţine mulţimea ferestrelor închise în vectorul caracteristic `inchisa`.

```
#include <stdio.h>
#define INPUT_FILE  "win.in"
#define OUTPUT_FILE "win.out"
#define NMAX 100

int N;                  //N reprezinta numarul de ferestre
struct { int ls, ld, cs, cd;} f[NMAX];
//f - vector cu N componente in care retinem ferestrele
int inchisa[NMAX];
/* inchisa[i]=0 daca fereastra i este deschisa,
    respectiv 1 daca fereastra i este inchisa              */
int rez;
/* rez - variabila globala in care calculam numarul total
   de click-uri necesare                                   */

void citire(void)
{ int i;
  FILE * fin = fopen(INPUT_FILE, "r");
  fscanf(fin, "%d", &N);
  for (i=0; i<N; ++i)
      fscanf(fin, "%d %d %d %d", &f[i].ls, &f[i].cs,
                  &f[i].ld, &f[i].cd);
  fclose(file);
}
```

```
int peste(int a, int b)
/* functia returneaza valoarea 1 daca fereastra a este peste
coltul din dreapta-sus al ferestrei b si 0 in caz contrar  */
{ if (f[a].ls <= f[b].ls && f[a].ld >= f[b].ls &&
      f[a].cs <= f[b].cd && f[a].cd >= f[b].cd)
      return 1;
  return 0;
}

void inchide(int nr)
{/* functia inchide fereastra cu numarul nr */
  for (int i=N-1; i>=nr+1; i--)
    if (!inchisa[i] && peste(i, nr))
        inchide(i);
  inchisa[nr]=1;
  ++rez;
}

void rezolvare(void)
{ for (int i=0; i<N; i++) inchisa[i] = 0;
  rez = 0;
  inchide(0);
}
void afisare(void)
{ FILE * fout = fopen(OUTPUT_FILE,"w");
  fprintf(fout,"%d\n", rez);
  fclose(fout);
}

int main(void)
{ citire();
  rezolvare();
  afisare();  return 0;
}
```

Restaurarea unui apel de funcţie

Se citeşte din fişierul `f.in` un şir de maximum `100` de caractere care reprezintă o succesiune de nume de funcţii şi variabile, formate dintr-un singur caracter. Pentru fiecare funcţie ce intervine în şir se citesc numele şi aritatea (numărul de argumente) de pe o linie din fişier. Restauraţi şirul de caractere, punând la locul lor parantezele şi virgulele, astfel încât să reprezinte un apel de funcţie corect.

De exemplu, pentru fişierul de intrare:

```
fxgxyzhx
f 3
g 3
h 1
```

funcţia restaurată va fi:

```
f(x,g(x,y,z),h(x))
```

Soluţie

Vom memora datele de intrare în doi vectori: `NF`, în care reţinem numele funcţiilor, în ordinea din fişierul de intrare, şi `AF`, în care reţinem aritatea fiecărei funcţii (numărul de argumente). Pentru a restaura apelul de funcţie din şirul de caractere, citit de pe prima linie a fişierului de intrare, vom utiliza o funcţie recursivă denumită `Restaurare()`, care are un singur parametru (`i`), ce indică poziţia la care am ajuns cu restaurarea în şirul de intrare.

În funcţia `Restaurare()` verificăm dacă pe poziţia `i` în şirul de intrare este un nume de funcţie. Dacă da, se scrie numele funcţiei, urmat de „(", după care se restaurează (cu ajutorul funcţiei recursive) fiecare argument al funcţiei, separând argumentele prin virgulă. Numărul de argumente se obţine din vectorul `AF`. După ultimul argument restaurat se închide paranteza (se scrie pe ecran „)").

Dacă pe poziţia `i` în şirul de intrare nu este o funcţie, se scrie pe ecran caracterul respectiv (este o variabilă).

Observaţie

Soluţia se bazează în mod esenţial pe faptul că datele de intrare sunt corecte (deci apelul de funcţie poate fi restaurat).

```
#include <fstream.h>
#define NrMaxF 100
char NF[NrMaxF];  //NF contine numele fiecarei functii
char s[100];      //sirul care trebuie restaurat
int  AF[NrMaxF]; //AF contine aritatea functiilor
int  NrF;         //NrF - numarul de functii
void Citire();
int  AR(char);
void Restaurare(int &);

int main()
{ int i=0;
  Citire();
  Restaurare(i);
  return 0;
}
```

```
void Citire()
{ifstream f("f.in");
 char c;
 int a;
 f.getline(s, 100);
 while (!f.eof())
       { f>>c>>a;
         NF[NrF]=c; AF[NrF++]=a;          }
 f.close();}

int  AR(char c)
{//functia intoarce 0 daca c nu este un nume de functie,
 //respectiv aritatea functiei, in caz contrar
 for (int j=0; j<NrF; j++)
     if (NF[j] == c) return AF[j];
 return 0;}

void Restaurare(int & i)
{int k, j;
 cout<<s[i];
 k=AR(s[i++]);
 if (k)                //s[i] reprezinta un nume de functie
     {cout<<'(';
      for (j=1; j<k; j++)
           {Restaurare(i); //restauram argumentele functiei
            cout<<',';  }  //separandu-le prin virgule
       Restaurare(i);
       cout<<')';}
}
```

Exerciţiu

Modificaţi programul astfel încât să testeze dacă restaurarea este posibilă.

Imagine

Să considerăm o imagine alb-negru de dimensiune `L`×`L` pixeli. Un pixel poate fi alb (codificat cu valoarea `0`) sau negru (codificat cu valoarea `1`). Imaginile pot fi compresate în diverse moduri. Una dintre cele mai cunoscute scheme de compresie este următoarea:

1. Dacă imaginea este formată atât din pixeli `1`, cât şi din pixeli `0`, se reţine valoarea `1`, care indică faptul că imaginea va fi partiţionată în alte patru subimagini, aşa cum este descris la pasul 2. Altfel codificăm întreaga imagine ca `00` sau `01`, semnificând faptul că întreaga imagine este formată numai din pixeli `0`, respectiv numai din pixeli `1`.