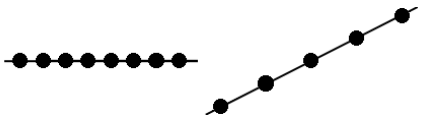
IOI 2002

Cel mai important concurs internațional de programare pentru liceeni s-a desfășurat în orașul sud-coreean Yong-In în cursul lunii august. Vă prezentăm acum enunțurile problemelor propuse spre rezolvare la a paisprezecea ediție a Olimpiadei Internaționale de Informatică.

P060213: Broscuțele buclucașe

În *Coreea*, *cheonggaeguri*-i, niște broscuțe mici, au devenit legendari pentru neastâmpărul lor. Această reputație este bine-meritată deoarece broscuțele sar prin lanurile de orez, culcând la pământ plantele. Dimineața, după ce sunt studiate plantele culcate la pământ, se dorește identificarea broscuței care a produs cele mai mari daune. O broscuță va sări întotdeauna în linie dreaptă, iar salturile au aceeași lungime.

Două broscuțe diferite pot efectua salturi de lungimi diferite sau în direcții diferite.



Lanul de orez are plantele aranjate într-un caroiaj asemănător cu cel din figura 1 și fiecare broscuță intră și iese din lan, după cum se poate vedea în figura 2.

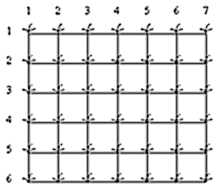


Fig. 1: Un lan de orez

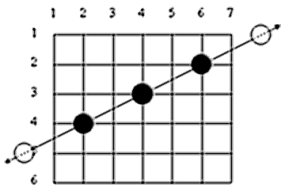


Fig. 2: Drumul unei broscuțe

Multe broscuțe pot trece prin lanul de orez, sărind de la o plantă la alta. Fiecare aterizare are loc pe o plantă care este culcată la pământ, după cum se poate vedea în figura 3. Se observă că o plantă poate fi folosită ca loc de aterizare de mai multe broscuțe. Bineînțeles, dimineața nu pot fi observate traseele broscuțelor și salturile în afara lanului, dar poate fi cercetată situația în care a ajuns lanul, așa cum se vede în figura 4.

Folosind figura 4, pot fi reconstituite toate traseele posibile pe care le-au urmat broscuțele prin lanul de orez. Ne interesează doar broscuțele care, pe parcursul traversării lanului, au aterizat pe cel puțin trei plante. Un astfel de

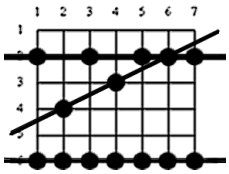


Fig. 3: Mai multe drumuri

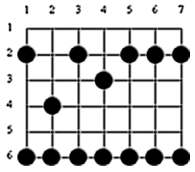


Fig. 4: Un lan dimineața

traseu este numit ***traseu al broscuței***. Așadar, cele trei trasee prezentate în figura 3 pot fi considerate a fi trasee ale broscuțelor.

Pot fi identificate și alte trasee posibile ale broscuțelor. S-ar putea ca traseul vertical care urmează prima coloană a caroiajului să fi fost urmat de o broscuță al cărei salt are lungimea 4. Totuși, broscuța respectivă a aterizat doar pe două plante, deci nu ne interesează un astfel de traseu.

Traseul pe diagonală care conține plantele distruse de pe pozițiile de coordonate (2, 3), (3, 4) și (6, 7) conține trei plante distruse, dar broscuța ar fi trebuit să efectueze salturi de lungimi diferite, deci nici un astfel de traseu nu ne interesează.

De asemenea, se observă că, pe un traseu parcurs de o broscuță pot exista plante distruse pe care broscuța nu a aterizat. Un exemplu în acest sens este planta din poziția (2, 6) care se află pe traseul orizontal de pe a doua linie (care este parcurs de o broscuță).

De fapt, distrugerea anumitor plante s-ar putea să nu poată fi explicată de nici unul dintre traseele broscuțelor.

Va trebui să scrieți un program care determină numărul maxim de aterizări necesare pentru parcurgerea unuia dintre traseele broscuțelor. Pentru figura 4, răspunsul este 7 și traseul corespunzător este cel care parcurge a șasea linie.

Date de intrare

Programul vostru va citi datele de intrare de la *intrarea standard*. Prima linie conține două numere întregi R și C, ($1 \le R, C \le 5000$), reprezentând numărul liniilor, respectiv al coloanelor, din lanul de orez. Cea de-a doua linie conține un singur număr întreg N ($3 \le N \le 5000$), care reprezintă numărul plantelor culcate la pământ.

Fiecare dintre următoarele N linii conține două numere întregi a ($1 \le a \le R$) și b ($3 \le b \le C$), separate printr-un singur spațiu, care reprezintă linia și coloana pe care se află una dintre plantele distruse. Fiecare plantă distrusă apare o singură dată.

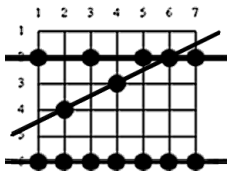
Date de ieșire

La *ieșirea standard* se va scrie o singură linie pe care se va afla un singur număr întreg care reprezintă numărul de plante culcate la pământ de broscuța care a produs cele mai mari daune. Dacă nu poate fi identificat nici un traseu al broscuței, se va scrie valoarea 0.

Exemple

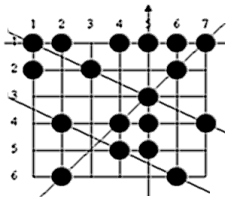
Intrare
```
6 7
14
2 1
6 6
4 2
2 5
2 6
2 7
3 4
6 1
6 2
2 3
6 3
6 4
6 5
6 7
```

Ieșire
```
7
```



Intrare
```
6 7
18
1 1
6 2
3 5
1 5
4 7
1 2
1 4
1 6
1 7
2 1
2 3
2 6
4 2
4 4
4 5
5 4
5 5
6 6
```

Ieșire
```
4
```



Timp maxim de execuție/test: 2 secunde
Memorie disponibilă: 64 MB

P060214: Utopia divizată

Frumosul ținut *Utopia* a fost afectat de război. După încheierea ostilităților, țara a fost împărțită în patru regiuni printr-o linie verticală (pe direcția nord-sud) și una orizontală (pe direcția est-vest). Intersecția acestor două linii a ajuns să fie cunoscută sub denumirea de ***punctul (0, 0)***. Toate cele patru regiuni formate au avut pretenția să folosească denumirea *Utopia* dar, pe măsura trecerii timpului, au ajuns să fie cunoscute drept *Utopia* 1 (regiunea din nord-est), 2 (regiunea din nord-vest), 3 (regiunea din sud-vest) și 4 (regiunea din sud-est). Orice punct din interiorul oricăreia dintre cele patru regiuni a ajuns să fie identificat prin distanța spre est și distanța spre nord față de punctul (0,0). Evident, aceste distanțe pot fi negative; ca urmare, un punct din *Utopia* 1 a ajuns să fie identificat printr-o pereche (*pozitiv*, *pozitiv*), unul din *Utopia* 2 printr-o pereche (*negativ*, *pozitiv*), unul din *Utopia* 3 printr-o pereche (*negativ*, *negativ*), iar unul din *Utopia* 4 printr-o pereche (*pozitiv*, *negativ*).

Utopia 2 (−,+)	*Utopia* 1 (+,+)
Utopia 3 (−,−)	*Utopia* 4 (+,−)

O problemă majoră a apărut datorită faptului că cetățenilor nu le era permis să traverseze frontierele. Din fericire, câțiva participanți la *IOI* veniți din *Utopia* au descoperit un mijloc simplu de teleportare. Dispozitivul necesită folosirea unor coduri (reprezentate de numere) și fiecare cod poate fi utilizat o singură dată.

Provocarea în fața căreia se află membrii echipei (și dumneavoastră) este de a ghida teleportorul din punctul (0, 0) care este poziția sa inițială, prin regiunile *Utopiei* într-o ordine specificată. Nu contează unde în interiorul unei regiuni ajunge teleportorul, dar veți avea o secvență de N numere care identifică regiuni în care teleportorul trebuie să ajungă.

S-ar putea să vi se ceară să ajungeți în aceeași regiune la două sau mai multe opriri consecutive. După părăsirea poziției inițiale, teleportorul nu are voie să ajungă pe o linie de graniță.

Veți primi o secvență de $2 \cdot N$ coduri pe care va trebui să le scrieți ca o secvență de N perechi de coduri, plasând semnul plus sau minius în fața fiecărui număr.

De exemplu, dacă teleportorul se află în poziția (x, y) și se folosește perechea de coduri ($+u$, $-v$), atunci teleportarea va avea loc din punctul (x, y) în punctul ($x + u$, $y - v$). Aveți la dispoziție cele $2 \cdot N$ numere și le puteți utiliza în ce ordine doriți, folosind în fața lor semnul pe care îl doriți.

Să presupunem că aveți la dispoziție codurile 7, 5, 6, 1, 3, 2, 4, 8 și trebuie să ghidați teleportorul prin regiuni în ordinea 4, 1, 2, 1. Folosirea secvenței de perechi de coduri (+7, -1), (-5, +2), (-4, +3), (+8, +6) duce la atingerea scopului, teleportorul deplasându-se, succesiv, în pozițiile (7, -1), (2, 1), (-2, 4) și (6, 10). Aceste puncte se află în *Utopia* 4, *Utopia* 1, *Utopia* 2, respectiv *Utopia* 1.

Se dau $2 \cdot N$ coduri distincte și o secvență de N numere care indică regiunile în care va trebui să ajungă teleportorul. Construiți o secvență de perechi de coduri, folosind codurile date, astfel încât teleportorul să treacă prin cele N regiuni, în ordinea dată.

Date de intrare

Datele de intrare vor fi citite de la *intrarea standard*. Prima linie va conține un număr întreg pozitiv N ($1 \le N \le 10000$). A doua linie va conține $2 \cdot N$ coduri, reprezentate prin numere întregi cuprinse între 1 și 100000. Ultima linie conține o secvență de N numere cuprinse între 1 și 4, care reprezintă regiunile în care trebuie să ajungă teleportorul. Numerele de pe aceeași linie sunt separate printr-un singur spațiu.

Date de ieșire
La ieșirea standard vor fi scrise *N* linii, fiecare conținând o pereche de coduri; fiecare cod va fi precedat de un semn (plus sau minus). Acestea sunt perechile de coduri care vor ghida teleportorul prin regiunile Utopiei, respectând ordinea dată. Cele două coduri de pe o linie vor fi separate printr-un singur spațiu și nu pot exista spații între semne și numere.

Dacă există mai multe soluții, va trebui generată una singură. Dacă nu există nici o soluție, la ieșirea standard se va scrie doar valoarea `0`.

Exemple

Intrare	*Ieșire*
`4`	`+7 -1`
`7 5 6 1 3 2 4 8`	`-5 +2`
`4 1 2 1`	`-4 +3`
	`+8 +6`

Intrare	*Ieșire*
`4`	`+3 -2`
`2 5 4 1 7 8 6 3`	`-4 +5`
`4 2 2 1`	`-6 +1`
	`+8 +7`

Timp maxim de execuție/test: 2 secunde
Memorie disponibilă: 32 MB

P060215: Compresie XOR

Trebuie să implementați o aplicație pentru un tip de telefon mobil care are un ecran alb-negru. Coordonatele orizontale (*x*) ale ecranului încep din partea stângă, iar coordonatele verticale (*y*) din partea de sus. Pentru aplicație aveți nevoie de diferite imagini care nu au aceleași dimensiuni. Pentru a nu fi nevoiți să stocați imaginile, vreți să le creați folosind biblioteca grafică a telefonului. Puteți presupune că înainte să începeți desenarea primei imagini, toți pixelii de pe ecran sunt albi. Singura operație grafică disponibilă este `XOR(L,R,T,B)`, al cărei efect este inversarea valorilor din dreptunghiul care are colțul din stânga-sus în poziția `(L,T)` și cel din dreapta-jos în poziția `(R,B)`.

De exemplu, să considerăm imaginea din figura 3; vom prezenta modul în care ar putea fi obținută aceasta. Prin aplicarea operației `XOR(2,4,2,6)` asupra imaginii inițiale, se obține imaginea din figura 1. Apoi, prin aplicarea operației `XOR(3,6,4,7)` asupra imaginii obținute, se va ajunge la cea din figura 2. Pentru a obține imaginea din figura 3 va trebui să aplicăm operația `XOR(1,3,3,5)` asupra imaginii din figura 2.

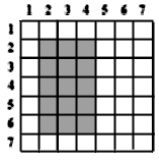


Fig. 1: XOR(2,4,2,6)

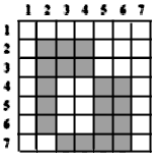


Fig. 2: XOR(3,6,4,7)

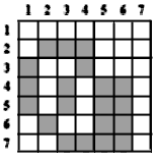


Fig. 3: XOR(1,3,3,5)

Dându-se mai multe imagini alb-negru, trebuie să generați fiecare poză pornind de la un ecran alb și folosind cât mai puține operații `XOR`. Vi se pun la dispoziție toate fișierele de intrare care descriu imaginile și trebuie să obțineți fișierele de ieșire corecte, eventual scriind un program.

Date de intrare
Există zece seturi de date, în fișierele **`XOR1.IN`**, **`XOR2.IN`**, ..., **`XOR10.IN`**. Prima linie a unui fișier de intrare conține un număr întreg *N* ($5 \le N \le 2000$) care indică faptul că imaginea are *N* linii și *N* coloane. Restul liniilor conțin informații referitoare la imaginea care trebuie obținută. Fiecare linie conține *N* numere întregi care reprezintă valorile pixelilor de pe linia repectivă. Numerele pot avea valoarea `0` (reprezintă un pixel alb) sau 1 (reprezintă un pixel negru).

Date de ieșire
Pentru fiecare fișier de intrare va trebui să creați un fișier de ieșire cu dimensiunea mai mică decât 1 MB. Prima linie a unui fișier de ieșire va fi `#FILE xor I`, unde `I` reprezintă numărul de ordine al fișierului de intrare corespunzător. Cea de-a doua linie va conține numărul *K* care reprezintă numărul de operații `XOR` descrise în fișier. Următoarele *K* linii vor reprezenta câte o operație `XOR`, în ordinea în care acestea sunt executate. Fiecare linie va conține patru numere întregi care reprezintă parametrii `L`, `R`, `T` și `B` ai operației.

Exemplu

`XOR0.IN`
```
7
0 0 0 0 0 0 0
0 1 1 1 0 0 0
1 0 0 1 0 0 0
1 0 1 0 1 1 0
1 0 1 0 1 1 0
0 1 0 0 1 1 0
0 0 1 1 1 1 0
```

`XOR0.OUT`
```
#FILE xor 0
3
2 4 2 6
3 6 4 7
1 3 3 5
```

P060216: Programarea pachetelor

Un calculator trebuie să proceseze o secvență de *N* operații. Operațiile sunt numerotate de la 1 la *N*, astfel încât secvența este 1, 2, ..., *N*. Secvența de operații trebuie partiționată în unul sau mai multe pachete, fiecare pachet fiind format din operații consecutive în secvență. Procesarea începe la momentul `0`.

Pachetele sunt procesate unul câte unul, începând cu primul pachet și respectând regulile care vor fi prezentate în continuare. Dacă un pachet *b* conține operații care au numere de ordine mai mici decât operațiile din pachetul *c*, atunci pachetul *b* este procesat înaintea pachetului *c*. Operațiile dintr-un pachet sunt procesate succesiv. Imediat după procesarea tuturor operațiilor dintr-un pachet, calculatorul furnizează datele de ieșire corespunzătoare tuturor operațiilor din pachet. ***Timpul de ieșire*** al unei operații *j* este dat de momentul în care se termină procesarea pachetului care conține operația *j*.

Un *timp de setare* S este necesar pentru inițializarea calculatorului pentru fiecare pachet. Pentru fiecare operație i, se cunoaște *factorul de cost* F_i și *timpul de execuție* T_i necesare pentru procesarea sa. Dacă un pachet conține operațiile x, $x + 1$, … , $x + k$ și procesarea sa începe la momentul t, atunci timpul de ieșire ale tuturor operațiilor din pachet este $t + S + (T_x + T_{x+1} + \ldots + T_{x+k})$. Rețineți faptul că datele de ieșire corespunzătoare tuturor operațiilor dintr-un pachet sunt furnizate simultan. Dacă timpul de ieșire al unei operații i este O_i, *costul* său este $O_i \cdot F_i$.

De exemplu, să presupunem că avem 5 operații și timpul de setare este $S = 1$. Timpii de execuție ai operațiilor sunt $(T_1, T_2, T_3, T_4, T_5) = (1, 3, 4, 2, 1)$, iar factorii de cost sunt $(F_1, F_2, F_3, F_4, F_5) = (3, 2, 3, 3, 4)$. Dacă operațiile sunt partiționate în trei pachete ({1, 2}, {3} și {4, 5}), atunci timpii de ieșire ai operațiilor sunt $(O_1, O_2, O_3, O_4, O_5) = (5, 5, 10, 14, 14)$, iar costurile operațiilor sunt $(C_1, C_2, C_3, C_4, C_5) = (15, 10, 30, 42, 56)$.

Costul total corespunzător unei partiționări este dat de suma costurilor tuturor operațiilor. Pentru acest exemplu, costul total este 153.

Trebuie să scrieți un program care, dându-se timpul de setare, o secvență de operații, timpii și factorii lor de cost, calculează costul total minim.

Date de intrare

Programul vostru va citi datele de intrare de la *intrarea standard*. Prima linie conține numărul N al operațiilor ($1 \le N \le 10000$). Cea de-a doua linie va conține timpul de setare S, care este un număr întreg astfel încât $0 \le S \le 50$. Următoarele N linii conțin informațiile referitoare la operațiile 1, 2, …, N, în această ordine. Liniile conțin câte două numere, separate prin câte un spațiu. Primul dintre ele este timpul de execuție T_i al operației ($1 \le T_i \le 100$), iar al doilea este factorul de cost F_i al acesteia ($1 \le F_i \le 100$).

Date de ieșire

La *ieșirea standard* se va scrie o singură linie pe care se va afla un singur număr care reprezintă costul total minim care poate fi obținut.

Exemple

Intrare	*Ieșire*
2 50 100 100 100 100	45000

Intrare	*Ieșire*
5 1 1 3 3 2 4 3 2 3 1 4	153

Timp maxim de execuție/test: 0,1 secunde
Memorie disponibilă: 32 MB

P060217: Stații de autobuz

În orașul *Yong-In* se plănuiește construirea unei rețele de autobuze cu N stații. Fiecare stație este la un colț de stradă. *Yong-In* este un oraș modern, așadar harta sa este un caroiaj de pătrățele având aceleași dimensiuni.

Două dintre stații (H_1 și H_2) trebuie să fie selectate ca *stații centrale*. Aceste stații sunt legate între ele printr-o linie directă și fiecare dintre celelalte N - 2 stații vor fi conectate direct fie cu H_1, fie cu H_2, dar nu cu ambele și cu nici o altă stație.

Distanța dintre oricare două stații este dată de cea mai scurtă rută care folosește străzile. Cu alte cuvinte, distanța dintre o stație aflată în poziția (x_1, y_1) și una aflată în poziția (x_2, y_2) este dată de $|x_2 - x_1| + |y_2 - y_1|$.

Dacă stațiile A și B sunt legate de aceeași stație centrală H_1, atunci lungimea traseului de la stația A la stația B este dată de suma distanțelor dintre A și H_1 și dintre B și H_1. Dacă stațiile sunt legate de stații centrale diferite (A de stația centrală H_1 și B de stația centrală H_2), atunci lungimea traseului dintre ele este dată de suma distanțelor dintre A și H_1, H_1 și H_2 și dintre B și H_2.

Autoritățile din *Yong-In* ar dori să se asigure că fiecare locuitor va putea ajunge în orice punct al orașului cât mai repede posibil. Ca urmare, se dorește alegerea celor două stații centrale astfel încât distanța maximă dintre oricare două stații să fie cât mai mică posibil.

O alegere P a două stații centrale și stabilirea stațiilor care sunt legate de cele două stații centrale este mai bună decât o altă alegere Q dacă și numai dacă cel mai lung traseu corespunzător alegerii P este mai scurt decât cel mai lung traseu corespunzător alegerii Q. Va trebui să scrieți un program care va determina lungimea celui mai lung traseu corespunzător celei mai bune alegeri P.

Date de intrare

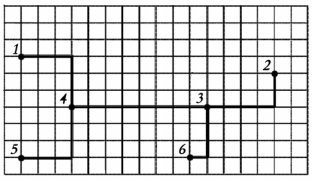
Programul vostru va citi datele de intrare de la *intrarea standard*. Prima linie va conține numărul N ($2 \le N \le 500$) al stațiilor. Fiecare dintre următoarele N linii conține coordonatele unei stații de autobuz. Aceste coordonate sunt numere întregi pozitive mai mici decât 5000. Nu pot exista două sau mai multe stații care se află în aceeași poziție.

Date de ieșire

La *ieșirea standard* se va scrie o singură linie pe care se va afla un singur număr care reprezintă cea mai mică lungime posibilă a celui mai lung traseu.

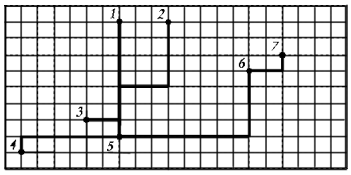
Exemple

Intrare	*Ieșire*
6 1 7 16 6 12 4 4 4 1 1 11 1	20



Intrare
```
7
7 9
10 9
5 3
1 1
7 2
15 6
17 7
```

Ieșire
```
25
```



Timp maxim de execuție/test: 4 secunde
Memorie disponibilă: 32 MB

P060218: Două bețe

Un ***băț*** este o secvență de cel puțin două celule consecutive ale unui caroiaj, așezate fie orizontal, fie vertical. Pe un caroiaj cu N linii și N coloane sunt așezate două bețe, unul vertical și unul orizontal. În figura 1, cele două bețe sunt reprezentate prin `X`-uri. Cele două bețe pot avea aceeași lungime sau, eventual, pot fi de lungimi diferite; mai mult, ele pot avea o celulă comună. Dacă, într-o diagramă cum este cea din figura 1 este posibil să interpretăm că o celulă face parte din unul din bețe sau din ambele, atunci vom considera că celula respectivă face parte din ambele bețe. De exemplu, nu se poate spune cu exactitate dacă celula (4, 4) face parte doar din bățul orizontal sau din ambele. Așadar, vom spune că cea mai de sus celulă a bățului vertical este (4, 4) și nu (5, 4).

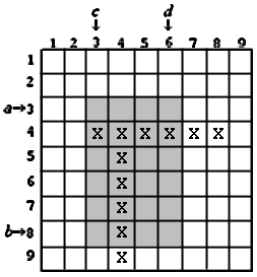


Fig. 1: Diagramă

Inițial, nu cunoaștem amplasamentul celor două bețe și sarcina dumneavoastră este de a scrie un program care determină pozițiile lor. Bățul orizontal va fi numit `BĂȚ1`, iar cel vertical `BĂȚ2`.

Fiecare celulă este reprezentată printr-o pereche de forma linie/coloană (r, c), colțul din stânga-sus al caroiajului fiind considerat (1, 1).

Fiecare băț este identificat prin cele două celule care reprezintă extremitățile sale: $\langle(r_1, c_1), (r_2, c_2)\rangle$. În figura 1 avem `ROD1 = <(4,3),(4,8)>` și `ROD2 = <(4,4),(9,4)>`.

Pentru a determina soluția va trebui să folosiți biblioteca pusă la dispoziție. Lungimea unei laturi a caroiajului este returnată de funcția `gridsize`, pe care o veți apela la început. Pentru a localiza bețele puteți folosi doar funcția `rect(a,b,c,d)`, care verifică regiunea dreptunghiulară $[a, b] \times [c, d]$ (de exemplu regiunea hașurată din figura 1), unde $a \le b$ și $c \le d$ (trebuie să fiți atenți la ordinea parametrilor). Dacă în regiunea respectivă există cel puțin o celulă care face parte dintr-un băț, atunci funcția va returna valoarea `1`; în caz contrar valoarea returnată va fi `0`. Așadar, pentru exemplul din figura 1, în urma apelului `rect(3,8,3,6)` este returnată valoarea `1`. Va trebui să scrieți un program care determină pozițiile bețelor folosind un număr limitat de apeluri ale funcției `rect`.

Rezultatul va fi furnizat folosind funcția `report(r1,c1,r2,c2,p1,q1,p2,q2)`, pusă la dispoziție de bibliotecă. Interpretarea apelului este `ROD1 = <(r1,c1),(r2,c2)>` și `ROD2 = <(p1,q1),(p2,q2)>`. Apelul acestei funcții va duce la întreruperea execuției programului.

Biblioteca

Vom prezenta în continuare modalitatea de utilizare a bibliotecii în versiunile pentru *FreePascal* și *GNU C/C++*.

Varianta FreePascal

Subprogramele care vor fi apelate sunt declarate astfel:
```
function gridsize:LongInt;
function rect(a,b,c,d:LongInt):Longint;
procedure report(r1,c1,r2,c2,p1,q1,p2,q2:
                                    Longint);
```
Codul sursă va trebui să conțină o linie de forma **uses** `prectlib;`.

Varianta GNU C/C++

Funcțiile care vor fi apelate sunt declarate astfel:
```
int gridsize();
int rect(int a,int b,int c,int d);
void report(int r1,int c1,int r2,int c2,
                int p1,int q1,int p2,int q2);
```
Codul sursă va trebui să conțină o linie de forma `#include "crectlib.h"`.

Restricții

- Puteți accesa datele de intrare doar prin intermediul funcțiilor `gridsize` și `rect`.
- Numărul liniilor și coloanelor din caroiaj este cuprins între 5 și 10000.
- Funcția `rect` poate fi apelată de cel mult 400 de ori. Dacă această funcție este apelată de peste 400 de ori, execuția programului va fi încheiată.
- Programul dumneavoastră va trebui să apeleze funcția `rect` de mai multe ori, în timp ce funcția `report` trebuie apelată o singură dată.
- Dacă un apel al funcției `rect` nu este valid, execuția programului va fi încheiată.
- Nu este permis ca programul dumneavoastră să citească date din fișiere, să scrie date în fișiere sau să folosească intrarea sau ieșirea standard.

Punctare

Dacă programul dumneavoastră furnizează rezultatul corect, atunci punctajul obținut depinde de numărul de apeluri ale funcției `rect`. Pentru fiecare test, veți primi întregul punctaj dacă sunt folosite cel mult 100 de apeluri. În cazul în care sunt folosite între 101 și 200 de apeluri, atunci veți primi 60% din punctaj. Pentru un număr de apeluri cuprins între 201 și 400, se va acorda 20% din punctaj.

Timp maxim de execuție/test: 1 secundă
Memorie disponibilă: 32 MB