Clasele XI-XII

asmin — Sursa : `asmin.c`, `asmin.cpp` sau `asmin.pas`

Se consideră un arbore (graf conex aciclic) cu **N** vârfuri, fără rădăcină fixată. Drept rădăcină, poate fi ales oricare dintre vârfuri. Să presupunem că a fost ales vârful cu numărul **T**. Între oricare vârf şi **T** există un drum unic care conţine fiecare vârf al arborelui cel mult o singură dată (un drum între vârfurile **i** şi **j** este o secvenţă de vârfuri, care începe cu **i**, se termină cu **j**, iar între oricare două vârfuri consecutive există o muchie în arbore). Fiecărui vârf **i** (inclusiv **T**) trebuie să i se asocieze o valoare V_i, mai mare sau egală cu **0**, astfel încât suma valorilor vârfurilor de pe drumul dintre **i** şi rădăcina **T**, împărţită la **K**, să dea restul R_i. Se defineşte costul arborelui cu rădăcina fixată în **T**, C_T, ca fiind suma valorilor asociate fiecărui nod. Dintre toate posibilităţile de alegere a valorilor V_i care respectă condiţia precizată anterior, se va alege aceea pentru care C_T este minim.

Se constată uşor că alegând alt vârf drept rădăcină, de exemplu, vârful **S** (diferit de **T**), C_S nu este neapărat egal cu C_T.

Cerinţă

Dându-se un arbore cu **N** vârfuri, un număr întreg **K** şi valorile R_i, **i=1,2,..,N**, corespunzătoare fiecărui vârf, determinaţi acele vârfuri **T** care pot fi alese drept rădăcină, pentru care costul C_T este minim (adică $C_T \le C_S$, oricare ar fi **S** diferit de **T**), precum şi costul respectiv.

Date de intrare

Pe prima linie a fişierului de intrare **asmin.in** se află **2** valori întregi: **N** şi **K**. Pe următoarele **N-1** linii se află câte două numere întregi **a b**, separate printr-un spaţiu, având semnificaţia că există muchie între vârfurile **a** şi **b**. Vârfurile sunt numerotate de la **1** la **N**. Pe următoarea linie se află **N** numere întregi, reprezentând valorile R_i, **i=1**,**2**,..,**N**.

Date de ieşire

Pe prima linie a fişierului de ieşire **asmin.out** se vor afişa două valori întregi: **C** şi **M**. **C** reprezintă costul minim posibil al arborelui. **M** reprezintă numărul de vârfuri care pot fi alese drept rădăcină şi pentru care se obţine costul **C**. Pe a doua linie se află **M** numere întregi separate prin câte un spaţiu, scrise în ordine crescătoare, reprezentând numerele vârfurilor ce pot fi alese ca rădăcină astfel încât să se obţină costul **C**.

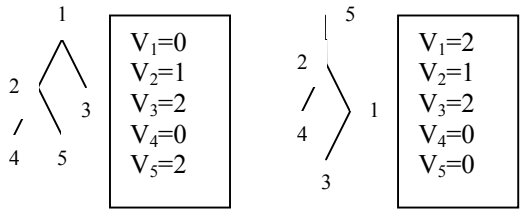
Restricţii şi precizări

- $2 \le N \le 16000$
- $2 \le K \le 1000$
- $0 \le R_i \le K-1$
- Cel puţin **40%** din testele folosite la evaluare vor avea $N \le 1000$

Exemplu

asmin.in	asmin.out
5 3 1 2 1 3 2 4 2 5 0 1 2 1 0	5 2 1 5

Cei doi arbori obţinuţi (împreună cu valorile asociate vârfurilor) sunt următorii:

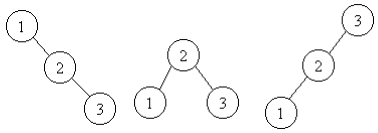


Timp maxim de execuţie: 0.2 secunde/test (atât sub Windows, cât şi sub Linux)

Clasele XI-XII
căutare

Sursa: `cautare.c, cautare.cpp sau cautare.pas`

Ştim cu toţii ce este un arbore binar de căutare. Este acel arbore binar în care informaţia din orice nod este mai mare decât informaţiile nodurilor din subarborele stâng al nodului respectiv şi mai mică decât cele din subarborele drept.
Când se caută o informaţie într-un arbore binar de căutare, începem din rădăcina arborelui şi comparăm cu informaţia din rădăcină. Dacă informaţia căutată e mai mică decât informaţia din rădăcină, se continuă căutarea în subarborele stâng, iar dacă e mai mare în subarborele drept. Dacă cele două informaţii sunt egale căutarea se termină cu succes. Dacă în direcţia în care continuăm căutarea (stânga sau dreapta) subarborele nu mai are noduri înseamnă că informaţia căutată nu se găseşte în arbore. Evident, numărul de comparaţii efectuate la o căutare depinde de distanţa dintre rădăcină şi nodul în care se găseşte informaţia, respectiv cel la care putem decide că informaţia nu se află în arbore.
Ce s-ar întâmpla dacă înainte de a construi arborele binar de căutare am şti ce informaţii urmează să fie căutate în el? Nu cumva am putea construi arborele în aşa fel încât să minimizăm numărul de comparaţii efectuate?
De exemplu cu informaţiile 1, 2 şi 3 putem construi un arbore binar de căutare în următoarele 3 moduri (din totalul de 5 posibile):



Dacă vom căuta informaţiile (1, 1, 3, 1, 2), atunci timpul total de căutare este pentru arborele din stânga 1+1+3+1+2 = 8, pentru arborele din centru 2+2+2+2+1 = 9, iar pentru arborele din dreapta 3+3+1+3+2 = 12
Deci arborele din stânga este cel adecvat pentru căutările noastre, iar timpul total de căutare minim este 8.
De remarcat că dacă se caută o informaţie care nu există în arbore atunci numărul de comparaţii efectuate la căutare este egal cu nivelul ultimului nod interogat. De exemplu, dacă se caută informaţia 4 pe cei trei arbori atunci timpii vor fi: 3, 2, respectiv 1 (de la stânga la dreapta).

Cerinţă

Scrieţi un program care, pentru anumite informaţii căutate, determină timpul total de căutare minim.

Date de intrare

Din fişierul **cautare.in** se citeşte de pe prima linie numărul `T` de teste. În fişier urmează cele `T` teste. Pentru fiecare test în fişier sunt scrise următoarele linii:

- prima linie conţine `N`, numărul de noduri ale arborelui de căutare şi `M` numărul de interogări, separate prin spaţiu;
- pe următoarea linie urmează `N` numere întregi distincte, separate prin câte un spaţiu, reprezentând informaţiile din nodurile arborelui
- pe fiecare dintre următoarele `M` linii sunt câte 2 numere întregi separate printr-un spaţiu, reprezentând un număr căutat şi respectiv de câte ori a fost căutat (între 1 şi 100).

Date de ieşire

În fişierul **cautare.out** se va scrie, pentru fiecare test câte o linie care conţine timpul total minim de căutare pentru interogările din testul respectiv.

Restricţii

- `0<T, N<101; 0<M<1001`
- informaţiile nodurilor arborelui sunt numere întregi din intervalul `[-10000, 10000]`
- informaţiile căutate sunt numere întregi din intervalul `[-1000000, 1000000]` şi pot fi căutate de un număr de ori cuprins între `1` şi `100`.

Exemplu

cautare.in	**cautare.out**	**Explicaţii**
3 3 3 1 2 3 1 3 2 1 3 1 3 3 1 2 3 1 50 2 49 3 51 3 2 1 2 3 2 1 4 20	8 251 22	Primul test este cel discutat. În al doilea test se interoghează 1 de 50 de ori, 2 de 49 de ori si 3 de 51 de ori. Cel mai bun rezultat se obţine în cazul arborelui din centru, şi anume 251 (2*50+49+2*51). Al treilea test conţine o interogare a lui 4 (care nu se află în arbore) de multe ori (20). Evident că cel mai bun e arborele din dreapta, pe care ne dăm seama repede că 4 nu se află în arbore. Timpul total este (1*20+2).

Observaţii:

- Pentru un fişier se ia punctajul maxim dacă toate testele din fişier sunt corecte, altfel 0 puncte.
- Pentru 5 fişiere de test (din 10) `T<4`.

Timpul maxim de execuţie pentru un fişier de test: 1.5 secunde/Windows respectiv 0.3 secunde/Linux.

Clasele XI-XII

a007 Sursa: a007.c, a007.cpp sau a007.pas

Agentul **007** are de distrus o tabără de teroriști. Tabăra de teroriști este formată din mai multe obiective (depozite de muniție, pavilioane pentru teroriști, etc.), considerate punctiforme în plan. Agentul **007** primește de la serviciul de informații o hartă cu **n** obiective din tabăra teroriștilor, date prin coordonatele carteziene. Pe lângă hartă, agentul **007** mai primește și o armă specială (construită pentru această misiune). Arma primită are două țevi și permite tragerea simultană pe aceeași direcție (rectilinie), dar în sens invers a două rachete cu aceeași viteză. După ce se trage cu arma, odată cu atingerea unei ținte explodează și cealaltă rachetă (chiar dacă aceasta din urma nu și-a atins ținta).

Cerință
Agentul **007** vrea să distrugă tabăra cât mai repede și cu cât mai puține rachete, pentru acest lucru el studiază posibilitatea să se așeze **într-un punct** din tabără (diferit de obiective) care să permită trageri eficace, adică la fiecare tragere să distrugă câte două obiective simultan.
Determinați dacă este posibil să se găsească un astfel de punct.

Date de intrare
În fișierul **a007.in** pe prima linie se află numărul de teste **k**, după care urmează date pentru fiecare test. Pentru fiecare test pe o linie se află **n**, iar pe următoarele **n** linii sunt coordonatele obiectivelor din tabăra teroriștilor (separate printr-un spațiu în ordinea `abscisă ordonată`).

Date de ieșire
În fișierul **a007.out** se vor scrie **k** linii, pe fiecare linie se va scrie **1**, dacă există soluție și **0** dacă nu există soluție. În cazul în care există soluție se va scrie în continuare pe aceeași linie separate, printr-un spațiu coordonatele punctului cerut (numere reale trunchiate la 4 zecimale, în ordinea `abscisă ordonată`).

Restricții
0 ≤ n ≤ 10000
1 ≤ k ≤ 3
coordonatele punctelor sunt întregi din intervalul [-10000, 10000]

Observație
Un obiectiv este distrus dacă racheta explodează exact în punctul corespunzător lui.

Exemplu

a007.in	a007.out
2 4 10 0 10 10 0 10 0 0 6 0 0 10 0 2 10 12 0 5 0 7 0	1 5.0000 5.0000 0

Timp maxim de execuție/test 0.2 secunde (pentru Windows și Linux)

Clasele XI-XII

inter — Sursa : inter.c, inter.cpp sau inter.pas

În țara *Smar* sunt **N** autostrăzi, sub forma unor drepte în plan. Se ştie că la intersecții de drumuri (care includ şi autostrăzi) există un risc ridicat de accidente. De aceea polițiştii din această țară au hotărât stabilirea unei zone compacte care să includă toate intersecțiile şi în care să se supravegheze atent circulația. Din motive financiare zona trebuie să fie de perimetru minim.

Cerință
Scrieți un program care să determine aria zonei de supraveghere alese.

Date de intrare
Din fişierul **inter.in** se va citi de pe prima linie numărul de autostrăzi, iar de pe fiecare dintre următoarele **N** linii câte patru numere reale, separate prin câte un spațiu, reprezentând coordonatele a două puncte distincte ce determină câte o dreaptă. Ele sunt date în ordinea **X1 Y1 X2 Y2**, adică abscisa şi ordonata punctului 1, apoi abscisa şi ordonata punctului 2.

Date de ieşire
În fişierul **inter.out** se va scrie pe prima linie un singur număr real, cu două zecimale exacte (cu trunchiere), reprezentând aria zonei alese pentru supraveghere.

Restricții şi precizări
- Între oricare două autostrăzi există fix o intersecție.
- Aria suprafeței de supraveghere este strict pozitivă pentru datele de test.
- 5 teste din 10 vor avea **N < 501**.
- **2 < N < 5001**

Exemplu

inter.in	inter.out
4 0 0 1 0 0 0 0 2 0 2 1 0 -2 0 0 1	3.00

Timp maxim de execuție/test : 0.2 secunde pentru Linux, 1.5 secunde pentru Windows.

Clasele XI-XII

nr — Sursa : nr.c, nr.cpp sau nr.pas

Fie `x` un număr natural cu exact `n` cifre scris în baza `10`.

Cerință
Scrieți un program care să determine cel mai mic număr natural strict mai mare decât `x`, care are aceleaşi cifre ca şi numărul `x` şi care este palindrom.

Date de intrare
Fişierul de intrare `nr.in` conține două linii. Pe prima linie este scris `n`, numărul de cifre ale numărului `x`. Pe cea de a doua linie sunt scrise cele `n` cifre ale lui `x`.

Date de ieşire
Fişierul de ieşire `nr.out` conține o singură linie pe care se află cel mai mic număr natural strict mai mare decât `x`, care are aceleaşi cifre ca şi numărul `x` şi care este palindrom. Dacă nu există soluție pe prima linie a fişierului de ieşire va fi scrisă valoarea `0`.

Restricții
- `2≤n≤1000`
- Numim palindrom un număr care citit de la stânga la dreapta, cât şi de la dreapta la stânga este acelaşi (de exemplu `1331`, `12321`, etc).
- Prima cifră a unui număr trebuie să fie nenulă.
- Prin aceleaşi cifre se înțelege că fiecare cifră de la 0 la 9 apare în rezultat de acelaşi număr de ori ca şi în numărul `x`.

Exemple

nr.in	**nr.out**
5 12022	0

nr.in	**nr.out**
5 12200	20102

Timp maxim de execuție: 0.1 secunde/test (atât sub Windows cât şi sub Linux).

Clasele XI-XII

proc — Sursa : proc.c, proc.cpp sau proc.pas

O aplicație ce trebuie executată pe un calculator multi-procesor constă din **N** fragmente de cod independente, ce pot fi rulate în paralel. Fiecare fragment trebuie executat în totalitate pe un singur procesor. Din dorința de a paraleliza cât mai mult aplicația, fragmentele de cod au dimensiuni mici şi, în consecință, timpi de execuție mici. Mai precis, execuția fiecărui fragment durează **UNUL** sau **DOUĂ** cicluri de ceas pe un procesor de tipul **Pentium IV**.
Sistemul pe care urmează să fie executată aplicația constă din **P** procesoare. Spre deosebire de majoritatea sistemelor de acest fel, însă, cele **P** procesoare au viteze de execuție diferite. Primul procesor este un **Pentium IV** şi este cel mai rapid. Al doilea procesor este de două ori mai lent decât primul, al treilea de trei ori mai lent … al **i**-lea procesor este de **i** ori mai încet decât primul. În aceste condiții, timpul de execuție al fiecărui fragment de cod diferă, în funcție de procesorul pe care va fi executat. Să prespunem că un segment de cod are timpul de execuție **T** (unde **T** este **1** sau **2**) pe primul procesor. Atunci pe un procesor **i**, timpul său de execuție va fi **i*T**.

Cerință
Ştiind că fragmentele de cod pot fi executate în orice ordine şi pe orice procesor şi că orice procesor poate executa, la un moment dat, un singur fragment de cod, determinați timpul minim după care se va termina execuția aplicației (adică a tuturor fragmentelor de cod). Timpul după care se termină aplicația este egal cu maximul dintre timpii după care fiecare procesor redevine disponibil. Timpul după care un procesor redevine disponibil este egal cu suma timpilor de execuție a fragmentelor de cod rulate pe procesorul respectiv.

Date de intrare
Prima (şi singura) linie a fişierului de intrare **proc.in** conține trei numere întregi, separate prin spații: **N** – numărul de fragmente de cod, **K** – numărul de fragmente de cod care au timpul de execuție pe un Pentium IV egal cu **1** (implicit, **N-K** au timpul de execuție egal cu **2**) şi **P** – numărul de procesoare ale sistemului.

Date de ieşire
Fişierul **proc.out** va conține o singură linie pe care se află timpul minim după care se termină de executat aplicația.

Restricții şi precizări
- **0 ≤ K ≤ N ≤ 1 000 000 000**
- **1 ≤ P ≤ 65 535**
- **Cel puțin 40% din teste vor avea N ≤ 2 000 şi P ≤ 2 000.**
- **Cel puțin 70% din teste vor avea N ≤ 65 535 şi P ≤ 16 383.**

Exemplu

proc.in	proc.out
4 3 2	4

Explicație
Pe primul procesor se execută un fragment de cod cu timpul de execuție (calculat pe un Pentium IV) egal cu **1** şi un fragment de cod cu timpul de execuție egal cu **2 =>** timpul după care acest procesor devine disponibil este **1*1 + 1*2 = 3**. Pe al doilea procesor se execută două fragmente de cod cu timpul de execuție (calculat pe un Pentium IV) egal cu **1 =>** timpul după care acest procesor devine disponibil este **2** [numărul de fragmente] *** (2*1)** [timpul de execuție al fiecărui fragment pe procesorul **2**] **= 4**.

Timp maxim de execuție: 0.2 secunde/test pentru Linux şi 1.5 secunde/test pentru Windows.