MARILENA OPREA RADU MARIN

Mult succes!!
M. Clulba.

TEHNICI DE OPTIMIZARE
CULEGERE DE PROBLEME

EDITURA INFODATA
CLUJ 2007

Referent: **Prof. Dr. Ferucio Laurenţiu ŢIPLEA**
Redactor: **drd. Tiberiu Socaciu**
Tehnoredactare computerizată: **Afrodita Timofti**

Descrierea CIP a Bibliotecii Naţionale a României

OPREA, MARILENA

Tehnici de optimizare : culegere de probleme / Marilena Oprea,Radu Marin. - Cluj-Napoca : Infodata, 2007

Bibliogr.

ISBN 978-973-88224-6-7

I. Marin, Radu

004

Editura InfoData
CP 522, OP 9
Cluj-Napoca
Email: office@editurainfodata.ro
Web: http://www.editurainfodata.ro

ISBN (10) 973-88224-6-7
ISBN (13) 978-973-88224-6-7

PREFAŢĂ

Programarea de performanţă este o provocare şi un domeniu extrem de dificil. Dar cu perseverenţă şi talent micii informaticieni îşi pot împlini visele: să obţină rezultate deosebite la concursurile judeţene, naţionale şi internaţionale, să devină programatori renumiţi. Pentru aceasta au nevoie, ca în toate domeniile de vârf, de o pregătire corespunzătoare şi nu în ultimul rând de modele.

Informatica de performanţă se face în proporţie de 60% în afara programei şcolare şi ca urmare oricât de mult ar dori un profesor să-şi îndrume elevii în cadrul orelor de curs pe calea performanţei la nivel naţional şi internaţional, elevii trebuie să fie conştienţi că obţinerea unei reuşite în acest domeniu va fi în cea mai mare parte rezultatul studiului lor individual. Dar au oare elevii la îndemână materialele necesare: manuale, culegeri cu modele de rezolvare, reviste şi articole cu ultimele noutăţi sau cerinţe în domeniu?

În acest context, o culegere de probleme ce explică şi prezintă soluţiile unor probleme propuse în cadrul concursurilor de informatică naţionale şi internaţionale nu poate fi decât binevenită. Uneori o implementare bună face cât o mie de explicaţii, în alte cazuri explicarea soluţiei face posibilă descifrarea unei implementări ermetice.

În această culegere am ales în primul rând probleme ale căror rezolvări necesită *algoritmi pe biţi*. De regulă, în manualele de informatică, operatorii pe biţi sunt menţionaţi fără a fi însoţiţi de exemple care să evidenţieze importanţa lor în economia algoritmilor pe biţi. Colecţia de probleme pe această temă pe care o oferim cititorilor este suficientă pentru a-i face pe aceştia să ajungă să înţeleagă şi să stăpânească această tehnică de lucru. Am dezvoltat tema *Heap-uri*, propusă ca lecţie opţională în anumite manuale alternative la clasa a XI-a, demonstrând prin exemplele alese uşurinţa cu care se poate lucra cu aceste structuri şi utilitatea lor practică. Ţinând cont că *Programarea dinamică* va fi introdusă în programa şcolară a clasei a XI-a, considerăm că prin capitolul al III-lea al acestei culegeri venim în sprijinul celor ce vor dori să aprofundeze noţiunile de la clasă.

Îi mulţumim domnului prof. dr. Ferucio Laurenţiu Ţiplea de la Facultatea de Informatică, Universitatea „Al.I.Cuza", Iaşi, pentru parcurgerea atentă a materialului şi pentru observaţiile făcute.

Le mulţumim familiilor noastre, tuturor celor care au crezut în noi şi ne-au încurajat să publicăm această culegere de probleme.

Autorii

Prof.Dr. Ferucio Laurenţiu Ţiplea
Facultatea de Informatică
Universitatea "Al.I.Cuza" Iaşi

REFERAT,

asupra lucrării „Tehnici de optimizare – Culegere de probleme de informatică" elaborată de Prof. Marilena Oprea şi de Radu Marin, student la Facultatea de Informatică Iaşi

Încurajarea tinerilor informaticieni români şi sprijinirea acestora pentru a ajunge pe culmi profesionale reprezintă nu numai o necesitate a învăţământului informatic românesc, dar şi un exemplu de dăruire profesională de care societatea noastră are mare nevoie. Am fost plăcut surprins să văd un astfel de exemplu prin intermediul lucrării asupra căreia îmi voi exprima părearea în cele ce urmează.

Lucrarea "Tehnici de optimizare – culegere de probleme" constituie o colecţie de 50 de probleme rezolvate şi 30 de probleme propuse, marea majoritate a acestora fiind propuse la olimpiade de informatică. Sunt prezentate clar trei clase majore de tehnici utilizate în rezolvarea problemelor: tehnici bazate pe reprezentarea datelor, tehnici bazate pe heap-uri şi tehnici de programare dinamică. Fiecare clasă de tehnici este precedată de considerente teoretice după care este ilustrată pe un număr de probleme.

Tehnicile bazate pe reprezentarea datelor fac apel la metode ingenioase de efectuare a unor operaţii cu date reprezentate binar. Se acordă atenţie deosebită operatorilor pe biţi, iar problema minimizării resurselor spaţiu şi timp constituie o constantă a lucrării. Heap-urile, un subiect delicat al algoritmicii, primesc o tratare naturală şi uşor de înţeles de către elevi, iar programarea dinamică este bogat exemplificată astfel încât să se vadă clar utilitatea acesteia.

Problemele rezolvate şi propuse sunt bine alese şi prezentate într-un mod foarte accesibil. Aceasta face ca, culegerea de probleme să nu constituie doar o simplă colecţie de probleme (rezolvate şi propuse) ci şi un material util pentru asimilarea şi înţelegerea respectivelor tehnici. Personal, găsesc cartea elaborată de prof. Marilena Oprea şi de Radu Marin, student la Facultatea de Informatică Iaşi, ca fiind un material extrem de util elevilor, informaticienilor, material ce poate completa prin excelenţă necesarul de informaţie al cititorului dornic de a pătrunde tainele algoritmicii şi programării de performanţă. Publicarea acesteia constituie un act de cultură informatică românească.

11 ianuarie 2006 Prof. Dr. F.L. Ţiplea

C U P R I N

78 CAPITOLUL 2

105 CAPITOLL

CAPITOLUL1

TEHNICI BAZATE PE REPREZENTAREA DATELOR

Operatori pe biți

Operatorii pe biți oferă acces direct la reprezentarea binară a datelor în memorie. Ei pot fi aplicați doar operanzilor de tip întreg cu sau fără semn.

În afară de operatorul ~ care este unar, toți operatorii C prezentați în tabelul de mai jos sunt binari.

DENUMIRE OPERATOR	OPERATORI C/C++	FORMULĂ	MOD DE FUNCȚIONARE
ŞI	&	a&b=1	dacă a=b=1, altfel 0
COMPLEMENTARE	~	~a=0	dacă a=1, altfel 0
SAU	|	a|b=0	dacă a=b=0, altfel 1
SAU EXCLUSIV	^	a^b=1	dacă a şi b au valori diferite, altfel 0
DEPLASARE LA STÂNGA	<<	c<<d	Operatorul << realizează deplasarea la stânga pentru operandul din stânga(c), cu numărul de poziții dat de operandul din dreapta(d).
DEPLASARE LA DREAPTA	>>	c>>d	Operatorul >> realizează deplasarea la dreapta pentru operandul din stânga(c), cu numărul de poziții dat de operandul din dreapta(d).

OBSERVAȚIE: a şi b sunt biți iar c şi d sunt numere întregi.

Exemple de folosire a operatorilor pe biți

Operatorii pe biți au aplicații în diferite domenii precum securitatea datelor şi viruşii informatici sau în animație. Astfel, putem ascunde conținutul unui fişier pentru cititorul neautorizat, sau putem *infecta* datele dintr-un calculator folosind o funcție, un şablon care să complementeze fiecare bit (folosind operatorul ~).

Devirusarea fişierului se obține aplicând încă o dată funcția de criptare pe fişierul codificat.

Acelaşi efect se poate obține folosind operatorul xor care aplicat de două ori pe acelaşi bit duce la obținerea inițială a valorii bitului. Operatorul binar xor(^) este asociativ, comutativ iar 0 este element neutru x^0=x, x^!x=1.

Operatorul binar xor este folosit de regulă în problemele în care diversele variabile pot avea doar două stări ce se pot modifica alternativ.

În cadrul orelor de programare, folosim de obicei în rezolvarea problemelor operatorii pe biți ca să testăm, setăm, resetăm sau ca să inversăm bitul k dintr-un număr dat.

Primele trei formule din următorul tabel au fost folosite în codificările ce apar în toate implementările problemelor din primul capitol al acestei culegeri, de aceea este necesar să încercați să le citiți de mai multe ori şi să le înțelegeți.

Formula C	Efect
1 << k	**Se obține numărul 2^k** în care bitul k este 1, iar restul biților sunt
n = n | (1 << k)	**Se setează bitul k din n la 1.**
n & (1 << k)	**Se verifică starea bitul k din n.**
n=n>>1	n=n/2
n=n<<1	n=2*n
n = n & ~(1 << k)	Se setează bitul k din n la 0.
n = n ^ (1 << k)	Bitul k din n se inversează, restul biților rămânând neschimbați.
n=n ^ 1	Schimbă ultimul bit din n.
n=n&(n-1)	Şterge bitul cel mai puțin semnificativ.
x^x=0	Folosind operatorul xor(^) putem anula dublurile unui număr natural dat.

EX.1: Fie a şi b două numere naturale, întregi fără semn:

- a^=a atribuie variabilei **a** valoarea 0
- a^=b^=a^=b interschimbă numerele naturale a şi b
- a|=~a atribuie variabilei **a** valoarea 2^{16}-1

EX.2: Se consideră un şir de numere în care, cu excepţia uneia, fiecare valoare întâlnită apare de un număr par de ori. Se cere determinarea singurului element care apare de un număr impar de ori.

Indicaţie: Vom folosi x^x=0 şi 0^x=x care fac ca dublurile din şir să se anuleze reciproc, în *produsul xor* regăsindu-se doar numerele cu număr impar de apariţii. Astfel, calculând produsul xor al elementelor din şir obţinem elementul căutat.

```
void afis(unsigned int n)
{
  unsigned int p=0,m;
       for(i=1;i<=n;i++)   {cin>>m; p^=m;}
  return p;
}
```

Optimizarea memoriei folosind vectori de biţi

Un vector de biţi este un vector în care fiecare element poate avea valoarea 0 sau 1. De exemplu, putem reprezenta o mulţime cu valori mici folosind un vector de biţi. Valoarea 1 va indica faptul că elementul respectiv aparţine acelei mulţimi şi 0 în caz contrar.

EX.1: Vector de biţi: **0 0 0 1 0 1 0 0**
Poziţie bit: **0 1 2 3 4 5 6 7**

Astfel elementul 0 va fi indicat de bitul 0, elementul 1 de bitul 1, şi aşa mai departe până la elementul 7 indicat de bitul cu numărul 7. În exemplul de mai sus numerele 3 şi 5 aparţin mulţimii. Reprezentarea în memorie a vectorului de biţi este o variabilă pe 8 biţi de tip **char** cu valoarea $2^3+2^5 = 40$.

În exemplul dat, deoarece avem maxim 8 elemente, acestea pot fi reprezentate printr-o singură variabilă. În practică, în majoritatea cazurilor vom avea nevoie de mai multe variabile de tip întreg şi folosim vectori de întregi.

Pentru a accesa bitul elementului corespunzător unui număr dat X, îl vom împărţi pe X la numărul de elemente ce încap într-o grupare (în exemplul nostru la 8) şi vom obţine indicele variabilei din vector sau numărul grupei în care este stocat bitul căutat.

Restul împărţirii lui X la 8 reprezintă poziţia bitului cu informaţia căutată în interiorul variabilei (grupei).

EX.2: Verificăm dacă elementul 11 face parte dintr-o mulţime dată. Mulţimea este reprezentată printr-un vector cu grupări de câte 8 biţi.

Valoare bit: 0 0 1 1 1 1 0 0 | 0 0 1 **0** 0 0 0 0| 0 . . .
Valori asociate: 0 1 2 3 4 5 6 7 | 8 9 10 **11** 12 13 14 15| 16 . . .
0 1 2 3 4 5 6 7 | 0 1 2 3 4 5 6 7 | 0 . . .
[primul char(8)] | [al doilea char] | [al treilea. .]. .

Elementul 11 se află în a doua grupă pe a treia poziţie (8x1+3=11). Deoarece bitul corespunzător poziţiei 11 este 0 rezultă că numărul 11 nu face parte din mulţimea dată.

Avantajele folosirii vectorilor de biţi sunt mari, chiar dacă în prezent memoria nu mai reprezintă o problemă aşa mare cum era în vremea calculatoarelor cu sisteme pe 16 biţi.

Operaţiile de împărţire respectiv modulo care sunt mari consumatoare de timp, în lucrul cu vectorii de biţi se pot înlocui. De exemplu putem folosi în locul câtului şi a restului împărţirii lui X la 8 deplasări pe biţi respectiv o mască de biţi compusă.

Măşti de biţi

Acum ştim cum să stabilim poziţia unui element într-un vector de biţi, dar trebuie să ştim şi cum să accesăm un bit dintr-un vector de biţi. Pentru acest lucru folosim măştile de biţi.

O mască este o variabilă care aplicată valorii ce conţine informaţia căutată (starea unui bit sau a mai multor biţi) returnează starea acesteia: 0 sau 1. Aplicarea se face folosind operatorul & pe biţi.

Ex.1: Masca necesară pentru a accesa bitul 2 al unei variabile pe 8 biţi:
masca: 0 0 1 0 0 0 0 0
& (operaţia aplicată)
variabila:1 0 0 1 1 1 0 1
rezultat: 0(starea bitului 2 este 0)

Dacă ar fi rezultat o valoare diferită de 0 atunci starea bitului 2 ar fi fost 1.

Ex.2: Măştile biţilor 0 până la 7:

indice bit: 0 1 2 3 4 5 6 7
val mască: 1 2 4 8 16 32 64 128

Ex.3: Pentru un şir de caractere dat, folosind o mască convenabilă şi operatorul **sau exclusiv** pe biţi putem face ca literele mari să devină litere mici şi invers, din litere mici să obţinem litere mari.

```
Citim sirul x
n=lungimea lui x
masca=32
Pentru i=0, n-1 executa
    y[i]=masca ^ x[i];
Sf. pt
Afisam sirul y
```

În tabelul din FIG.1 urmăriţi bitul al doilea numărat de la stânga la dreapta. El corespunde măştii=32, aleasă ca urmare a faptului că diferenţa dintre literele mari şi cele mici este 32. Literele mari au numerele asociate în tabelul Ascii cuprinse între 65 ('A') şi 90 ('Z'), reprezentările lor în baza 2 având toate valoarea 0 pe bitul al doilea numărat de la stânga spre dreapta, corespunzător lui 32.

Aplicând **sau exclusiv**, acest bit se setează pe 1 mărind astfel cu 32 ordinul asociat caracterului (65+32=97) transformând astfel litera mare în literă mică.

Analog, numerele asociate în tabela Ascii literelelor mici au în reprezentarea în baza doi, bitul al doilea numărat de la stânga spre dreapta setat pe 1 iar operatorul **sau exclusiv** îi schimbă valoarea în 0 micşorând numărul asociat caracterului în tabela Ascii cu 32 şi prin aceasta transformând literele mici în litere mari.

FIG. 1

X= 'ABab'	Numarul asociat în tabelul Ascii	X[i] în baza 2	y[i]= masca ^ x[i]	Numarul asociat în tabelul Ascii	Y= 'abAB'
X[1]='A'	Ord(X[1])=65	100000001	110000001	Ord(Y[1])=97	Y[1]='a'
X[2]='B'	Ord(X[2])=66	100000010	110000010	Ord(Y[2])=98	Y[2]='b'
X[3]='a'	Ord(X[3])=97	110000001	100000001	Ord(Y[3])=65	Y[3]='A'
X[4]='b'	Ord(X[4])=98	110000010	100000010	Ord(Y[4])=66	Y[4]='B'

EX. 4: Pentru extragerea unei porţiuni din reprezentarea unui număr folosim o mască (un tipar) în care biţii respectivi sunt pe 0 (sau 1):

- ~0 << k are ultimii k biţi pe 0, restul pe 1
- ~(~0 << k) are ultimii k biţi pe 1, restul pe 0
- ~(~0 << k) << p are k biţi pe 1, începând de la bitul p, şi restul 0

EX. 5: Să se afişeze reprezentarea în baza 2 a unui număr dat x, verificând starea fiecărui bit în parte.

```
void afis(unsigned int x)
{
  for(int k=0;k<=31;k++)
                if(x&(1<<k))  cout<<1;
                  else cout<<0;
}
```

Măşti de biţi compuse

Pentru a decupa anumiţi biţi din cadrul unei variabile vom folosi măştile de biţi compuse.

De exemplu masca A=0 1 0 1 0 0 1 1 aplicată unei valori obţine informaţia referitoare la biţii 1, 3, 6 şi 7.

Pentru a obţine o mască compusă (care indică starea mai multor biţi) se adună valorile măştilor acestor biţi (2+8+64+128=202).

În cazul exemplelor date până acum cu numere grupate câte 8, pentru a găsi bitul asociat unui număr dat X aveam nevoie de numărul grupei şi de poziţia bitului în cadrul grupei („Vectori de biţi", EX. 2).

Pentru a obţine numărul grupei înlocuim expresia X/8 cu formula mai rapidă **X>>3.** Pentru a obţine poziţia bitului în cadrul grupei (X%8) avem nevoie de cei mai nesemnificativi 3 biţi ai numărului X (0,1 şi 2), deci masca va fi: 1 1 1 0 0 0 0 0 = 7.

Aplicând această mască unui număr vom obţine restul împărţirii numărului la 8 (**X & 7 =X % 8**) dar cu un timp de execuţie mai rapid.

Optimizarea timpului de execuţie folosind vectori de biţi

Utilizarea vectorilor de biţi nu are nici un impact asupra timpului de execuţie, în schimb memoria folosită este de 8 ori mai mică.

În continuare vom vedea cazuri în care folosirea vectorilor de biţi este recomandată în special pentru optimizarea timpului de execuţie.

Să considerăm că avem un vector de elemente 0/1 (vector de biți). Definim o operație pe acest vector aplicarea unei scăderi, adunări, înmulțiri, împărțiri, deplasări, etc. asupra tuturor elementelor. Deoarece operația se face individual pe fiecare element, costul va fi dat de numărul de elemente al vectorului. Pentru un vector de N elemente aplicarea unui set de M operații are o complexitate N*M. Pentru N=1000 şi M=5000 un astfel de algoritm nu s-ar încadra să spunem în 0.1 secunde. Totuşi această complexitate poate fi micşorată printr-un factor constant (evident şi timpul de execuție este redus).

Revenind la vectorul nostru, să observăm pentru început că fiecărui element îi este aplicat acelaşi tip de operație. De ce să le efectuăm individual când le putem realiza pentru un grup (bucket) mai mare: de 8, 16 sau de 32 de elemente.

Să considerăm că operația este de adunare, iar vectorul nostru de biți este reprezentarea unui număr în baza 2. Nu este nevoie să simulăm în întregime o adunare pe biți deoarece asta ar însemna să reinventăm roata. Putem să îi grupăm în "bucket-uri" de câte 32 de biți (reprezentați prin tipul long în C++) şi să aplicăm operația o singură dată pe acest grup. Oricum calculatorul efectuează adunările mult mai repede decât dacă le-am simula noi pe biți. Astfel numărul total de operații este M*N/32(număr de grupe X o operație pe grupă) ceea ce pentru valorile N şi M anterioare este o complexitate mai mult decât mulțumitoare.

Toate problemele grupate în primul capitol al acestei culegeri încearcă să vă familiarizeze cu această tehnică de codificare a datelor folosind în rezolvarea lor atât codificări pe 8 dar şi pe 16 sau 32 biți.

Problemele enunțate şi rezolvate în această carte au fost realizate pentru compilatoare pe 32 de biți. Rezolvările pot fi implementate şi pentru compilatoare mai vechi precum Borland C 3.1 însă acest mediu este depăşit de cerințele concursurilor de informatică actuale, iar gestionarea dificilă a memoriei într-un astfel de mediu nu face decât să complice rezolvarea problemelor.

În prezent, în toate concursurile internaționale se folosesc doar compilatoare pe 32 biți. Astfel soluțiile pot fi compilate folosind compilatoare precum cel din mediul Dev-C++ (recomandat pentru utilizatorii de Windows) sau g++ pentru utilizatorii de Linux. Pentru programatorii ce folosesc mediul Visual C++, soluțiile pot fi adaptate pentru a rula corect astfel: deoarece în Visual C++ nu există tipul de date long long, se va folosi tipul _int64 echivalent cu acesta.

Deplasări pe biți

Deplasările pe biți (shiftări) calculează de fapt înmulțirile (<<) cu 2 şi împărțirile (>>) la 2 pentru un număr dat.

C/C++	Efect
A<<B	$A^* 2^B$
A>>B	$A^* 2^{-B}$

Operațiile de deplasare pe biți vor fi folosite în locul împărțirilor pentru a îmbunătăți timpul de acces. O deplasare la dreapta cu o poziție reprezintă împărțirea unui număr la 2. De exemplu 11>>1=5 este acelaşi lucru cu 11/2.

Deoarece biții sunt mutați către un capăt, la celălalt capăt se adaugă zerouri. O deplasare nu este o rotație, biții deplasați dincolo de capăt fiind pierduți.

O astfel de operație este cel puțin la fel de rapidă ca o adunare (dacă stăm şi ne gândim la modul de funcționare pe biți). Astfel costul obținerii "segmentului" (X>>3=X/8) în care se află valoarea lui X este echivalent cu o adunare. Mai rămâne de văzut obținerea eficientă a deplasamentului.

Într-o implementare, pentru a găsi bitul asociat unui număr dat X avem nevoie de numărul grupei şi de poziția bitului în cadrul grupei. În cazul în care implementăm hărțile de biți folosind vectori de tipul **unsigned int** (16 biți), numărul grupei este dat de formula X>>4 iar în cazul în care folosim tipul **unsigned long int** (32 biți), numărul grupei este dat de formula X>>5.

Ex1. Să se afişeze reprezentarea în baza 2 a unui număr dat **a.**

```
void numar(unsigned int a)
{
  unsigned int i;
 for(i=(1<<15);i;i=i>>1)
    if(a&i) cout<<1;
       else cout<<0;
}
```

OBS: 1<<15 setează bitul cel mai semnificativ şi calculează cea mai mare valoare posibilă pe tipul întreg(16 biți) şi anume 32 768.

OBS: i=i>>1 deplasează 1 pe şirul de 16 biţi către dreapta, împărţind succesiv la 2 valorile 32768, 16384, 8192, ... , 1 obţinând astfel măştile corespunzătoare fiecărui bit.

Spre deosebire de implementarea din **EX.**6, secţiunea „Măşti de biţi" în care secvenţa repetitivă este dată de poziţia bitului în şirul de biţi, în această implementare se parcurg măştile corespunzătoare biţilor.

Ex2. Afişaţi produsul a două numere x; y folosind algoritmul a la russe.

Rezolvare: Se iniţializează produsul cu 0. Se parcurge reprezentarea binară a lui x de la poziţia cea mai nesemnificativă la poziţia cea mai semnificativă şi dacă cifra binară curentă i este 1, atunci cel de-al doilea număr y se înmulţeşte cu 2^i şi se adună la valoarea produsului.

```
unsigned int produs(unsigned int x, unsigned int y)
{
  unsigned int P;
 for( P=0;x!=0;x>>=1,y<<=1)
    if(x&1)
        P+=y;
 return P;
}
```

Ex3. Fie n un număr natural scris în baza 10 şi b (2≤ b ≤ 256) o bază de numeraţie. Să se scrie un program care afişează în baza 10 toate numerele mai mici sau egale cu n, care scrise în baza b folosesc numai cifrele 0 şi 1. Numerele vor fi scrise în ordine strict crescătoare.

EX: n=11, b=3 **R:** 0, 1, 3, 4, 9, 10

Rezolvare: Se va opta pentru reprezentarea numerelor mai mici ca n în baza 2 urmată de calcularea numerelor în baza 10.

```
unsigned int numar(unsigned int n, unsigned int b)
{
 int i,nr,pb;
 for ( i=2; ; i++)
   {
    for (nr=0, x=i, pb=1; x; x>>=1, pb*=b)
            nr+=(x&1)*pb;
    if (nr<=n)  cout<<nr<<endl;
              else break;
  }
}
```


PROBLEME PROPUSE

PB1. Scrieţi o funcţie care calculează lungimea unui cuvânt de pe calculatorul gazdă, adică numărul de biţi dintr-un int. Funcţia să fie portabilă în sensul că acelaşi cod sursă să lucreze pe toate calculatoarele.
PB2. Scrieţi o funcţie care roteşte întregul n la dreapta cu b poziţii.
PB3. Scrieţi o funcţie care inversează din 1 în 0 şi viceversa cei n biţi ai lui x care încep de la poziţia p, lăsându-i pe ceilalţi neschimbaţi.
PB4. Scrieţi o funcţie care pentru n, k şi p cunoscute îl transformă pe n a.î. are pe ultimele poziţii cei k biţi ai lui n începând cu bitul p şi în rest 0.
PB5. Scrieţi o funcţie care n, k şi p cunoscute îl transformă pe n a.î. are cei k biţi începând cu bitul p la fel ca ai lui n şi restul biţilor pe 0.
PB6. Să se realizeze o funcţie care returnează (cadrat la dreapta) câmpul de lungime n biţi al lui x care începe la poziţia p. Presupunem că bitul 0 este cel mai din dreapta şi că n şi p sunt valori pozitive.
PB7. Folosind operatori la nivel de bit (>>, &) verificaţi câte numere mai mici decât un **n** dat au în reprezentarea în baza 2 un număr egal de 1 şi de 0.
PB8. Folosind măşti simple de biţi şi operatorul &, determinaţi câte cifre de 1 apar în scrierea binară a numărului n.
PB9. Fie secvenţa:

```
int x,a= -5;
x=a<<2;
cout<<x;
```

Ce se afişează? a)3 b) -20 c) -10
PB10. Se dau două numere mici (cuprinse între 0 şi 15). Se cere să se stocheze ambele numere, folosind un singur octet.
PB11. O proprietate interesantă a lui *sau exclusiv(^)* spune că aplicând unei variabile de două ori acest operator obţinem valoarea iniţială a variabilei. Folosiţi această proprietate ca să obţineţi criptarea unui fişier.
PB12. Se consideră un număr natural n şi n-1 numere naturale distincte aparţinând intervalului [1,n]. Să se determine singurul număr natural din intervalul [1,n] care nu apare în şirul celor n-1.
Indicaţie: Folosiţi operatorul xor după modelul ex.1 de la *Exemple de folosire a operatorilor pe biţi.*
PB13. Să se verifice egalitatea a^b=(!a&b)||(a&!b).
PB14. Calculaţi suma elementelor pare dintr-o secvenţă dată fără a folosi operatorul modulo.

SUBMULȚIMI PE BIȚI

Considerente teoretice

O submulțime a unei mulțimi cu n elemente se poate reprezenta printr-un șir de biți în care un element i aparține submulțimii dacă șirul de biți are setată poziția i cu 1.

EX: Șir de 8 biți: 0 0 **1** 0 0 0 0 0 0 0 **1** 0 0 0 0 0 0
Submulțimea: 0 1 **{2}** 3 4 5 6 7 8 9 **{10}** 11 12 13 14 15 16
este formată din elementele 2 și 10.

Operație	Expresie C/C++
Introducerea numărului k în mulțime (setarea bitului k cu 1 de la dreapta la stânga) **Ex: Pentru k=2 setam 000000100**	a[k/8] | (1<<k%8)
Verificarea apartenenței numărului la mulțime (dacă bitul k e setat pe 1 de dreapta la stânga)	a[k/8] & (1<<k%8)
Introducerea numărului k în mulțime (setarea bitului k cu 1 de la stânga la dreapta) **Ex: Pentru k=2 setam 001000000**	a[k/8] | (128>>k%8)
Verificarea apartenenței numărului la mulțime (dacă bitul k e setat pe 1 stânga la dreapta)	a[k/8] & (128>>k%8)

OBS. Reamintim formulele rapide k/8=k>>3 și k%8=k&7 discutate anterior. În funcție de implementare, se vor adapta formulele corespunzător tipurilor int(16) și long int(32).

Generarea submulțimilor

Fie n număr natural. Să se afișeze toate submulțimile formate cu numere de la 1 la n. Vom lua în calcul reprezentarea submulțimilor folosind vectorul caracteristic:

$$v[i]=\begin{cases}1 & \text{dacă i este element al submulțimii}\\ 0 & \text{altfel}\end{cases}$$

Submulțimile pot fi generate pornind de la zero prin adunarea repetată cu 1 în baza 2. Vom considera vectorul caracteristic ca fiind un șir de biți, iar ca submulțimi numerele de la 0 la 2^N.

Pentru N=3 avem urmatoarele 2^3 submulțimi generate, numerotate astfel:

Submulțime	Baza 2	Număr asociat mulțimii
Vidă	000	0
{1}	001	1
{2}	010	2
{1, 2}	011	3
{3}	100	4
{1,3}	101	5
{2,3 }	110	6
{1,2,3}	111	7

Submulțime

Pentru i=1, 2^N execută

Afișează poziția biților lui i care sunt setați pe 1;

sf pentru.

PROBLEME REZOLVATE

GRAFURI BIPARTITE

»»»ENUNȚ:

Fie G=(V,E) un graf. Prin graf bipartit înțelegem o partiție a lui V=AUB și A∩B= Φ și fiecare muchie are o extremitate în A și alta în B.

▸▸▸SOLUȚIE:

Se observă că mulțimile A şi B sunt complementare (de ex: 1şi 6).

Submul țimea A	Număr asociat mulțimii	Baza 2	Numere Comple-mentare	Baza 2	Număr asociat mulțimii	Submul țimea B
{1}	1	001	⟷	110	6	{2,3 }
{2}	2	010		101	5	{1,3}
{1, 2}	3	011		100	4	{3}

Nu se vor luau în considerare mulțimile complementare 0 şi 2^n-1.

▸▸▸PROGRAM:

```
//pentru grafuri cu maxim 32 de noduri
#include <iostream.h>
int n,i,j;
int main()
{
cin>>n;
//se generează submultimile A, submultimile B fiind complementare cu
cate un A
for(i=1;i<(1<<n-1);i++)
  {
  cout<<"A=";
  for(j=0;j<n;j++)
   if (i&(1<<j)) cout<<j+1;
  cout<<endl;
   cout<<"B=";
    for(j=0;j<n;j++)
   if (!(i&(1<<j))) cout<<j+1;
   cout<<endl;
 }
return 0;
}
```


SUBMULȚIMI DE SUMĂ DATĂ

▸▸▸ENUNȚ:

Se cere să se determine dacă o valoare dată S se poate scrie ca sumă de numere dintr-un vector dat. Fiecare număr din vector se poate folosi o singură dată.

Date de intrare

Pe prima linie a fişierului de intrare **fis.in** se află numărul de numere n şi suma S.

Pe a doua linie se găsesc numerele separate prin spații.

Date de ieşire

Pe prima linie a fişierului de ieşire **fis.out** se va găsi cuvântul *Posibil* sau *Imposibil.*

Exemplu:

FIS.IN	FIS.OUT
3 15 5 3 7	Posibil

▸▸▸SOLUȚIE:

Se aplică programarea dinamică. Dacă valori[j] ($1 \le j \le n$) este o valoare dintre cele n citite din fişierul de intrare, atunci valorile v[j]+i ($0 \le i \le S-v[j]$) sunt sume de valori date dacă şi valorile **i** se pot scrie ca sumă de numere din vectorul dat. În implementarea de față se setează cu 1 într-un vector de biți poziția bitul **i** dacă **i** este sumă de valori date.

▸▸▸PROGRAM:

```
#include <fstream.h>
#define Max 100000
ifstream f("fis1.in ");
ofstream g("fis2.out");
unsigned long int valori[100],S,c[Max/32+1],n;
 void citire()
 {long int i;
 f>>n>>S;
 for(i=1;i<=n;i++)
 f>>valori[i];
```

```
        f. close();
        }
        int main()
        { long int j,i, octet,poz;
        citire();
        c[1]=1;
         for(j=1;j<=n;j++)
           for(i=S-valori[j];i>=0;i--)
            {
              octet=(i>>5)+1;
              poz=i%32;
              if (c[octet] & (1<<poz))
               {
                 octet=((i+valori[j])>>5)+1;
                 poz=(i+valori[j])%32;
                 c[octet]=c[octet]|(1<< poz);
         }
        }
        octet=(S>>5)+1;
        poz=S%32;
        if(c[octet] & (1<<poz))
          g<<"Posibil!";
        else
          g<<"Imposibil!";
       g. close();
       return 0;
}
```

OBSERVAȚII:

• C[1]=1 setează bitul corespunzător numărului 0 cu 1.

• Parcurgerea descrescătoare în intervalul [S-valori[j]. . 0] ne asigură că în suma S numerele vor intra o singură dată.

Pentru datele de intrare precizate avem următoarea evoluție a şirului de biți:

C[1]=1
Se setează bitul 0
0 1 2 3 4 5 6 7 9 10 11 12 13 14 15 16....
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0...

Pentru valoarea 5
Se setează bitul5 // 0+5
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16....
1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0...
Pentru valoarea 3
Se setează bitul 8 // 5+3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16....
1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0
Se setează bitul 3 // 0+3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16....
1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0....
Pentru valoarea 7
Se setează bitul 15 // 7+8
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16....
1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 **1** 0....
Se setează bitul 12 //7+5
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16....
1 0 0 1 0 1 0 0 1 0 0 0 1 0 0 **1** 0....
Se setează bitul 10 // 7+3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16....
1 0 0 1 0 1 0 0 1 0 1 0 1 0 0 **1** 0
Se setează bitul 7 //7+0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16....
1 0 0 1 0 1 0 1 1 0 1 0 1 0 0 **1** 0....

BITSORT

»»ENUNȚ:

Se dă un vector de N elemente numere naturale distincte din intervalul [1,1000000]. Să se sorteze acest vector în ordinea crescătoare a valorilor.

Date de intrare:

Pe prima linie a fişierului **bitsort.in** se află un număr N, reprezentând dimensiunea vectorului.

Pe următoarea linie se află N valori, reprezentând elementele din vector.

Date de ieşire:
Pe singura linie a fişierului de ieşire **bitsort.out** se vor afla cele N valori, sortate crescător.

EXEMPLU	BITSORT.IN	BITSORT.OUT
	5	1 2 3 7 9
	1 7 2 9 3	

Restricţii:
0 < N < 1000001

»»»SOLUŢIE:

Pentru sortarea elementelor se va folosi metoda count-sort. Vom utiliza un vector de biţi în care vom reţine starea fiecărei valori din intervalul [1,1000000] şi anume 1 dacă aceasta este în vector şi 0 în caz contrar. La început toţi biţii vor fi setaţi pe 0 şi pe parcursul citirii elementelor din fişier vom marca cu 1 starea acestora. La afişare, parcurgem vectorul de biţi şi tipărim elementele ale căror stare este 1.

»»»PROGRAM:

```
#include<stdio.h>
#define MAXN  1000001
int N; char sort[MAXN/8+1];
void citesteDate()
{int i,a;
 freopen("bitsort.in ","r",stdin);
 scanf("%d",&N);
 for(i=1;i<=N;i++)
   {scanf("%d",&a);
    sort[a>>3] |= (1<<(a&7));
   }
}
void afiseazaRez()
{int i;
 freopen("bitsort.out","w",stdout);
 for(i=0;i<MAXN;i++)
  if (sort[i>>3] & (1<<(i&7)))
     printf("%d ",i);
}
```



```
int main()
{
 citesteDate();
 afiseazaRez();
 return 0;
}
```

REPARAŢII

»»»ENUNŢ:

Primăria oraşului Focşani îşi propune să gestioneze mai bine banii publici economisind din banii destinaţi reparării străzilor. Se doreşte identificarea intersecţiilor programate în această lună să intre în reparaţii atât pentru introducerea gazelor cât şi pentru repararea conductelor de apă.

Cerinţă:
Pornind de la graficele de intrare în lucru a celor maxim 32 de străzi programate şi ştiind că la o intersecţie se întâlnesc doar două străzi să se afişeze intersecţiile cerute prin afişarea numerelor străzilor ce se intersectează.

Date de intrare:
În fişierul de intrare **focsani.in** se vor găsi pe prima linie trei numere naturale k,n,m cu următoarele semnificaţii: k este numărul de străzi ce vor intra în reparaţie, n este numărul străzilor programate pentru introducerea ţevilor de gaze, m este numărul străzilor programate pentru schimbarea ţevilor de apă.
Pe a doua linie se află o succesiune de numere reprezentând numerele străzilor pe care se introduc gazele.
Pe a treia linie se află o succesiune de numere reprezentând numerele străzilor pe care se schimbă ţevile de apă.

Date de ieşire:
În fişierul de ieşire **focsani.out** vor fi scrise atâtea linii câte intersecţii care îndeplinesc condiţiile din enunţ vor fi găsite. Pe fiecare linie vor fi scrise separate printr-un spaţiu străzile ce formează intersecţia.
Restricţii:

0< k,n,m <32

EXEMPLU	FOCSANI.IN	FOCSANI.OUT
	7 6 4 1 4 3 7 1 4 5 7 3	1 4 3 7

Timp de execuție pe test: 0.1 secunde

⋙SOLUȚIE:

Pentru fiecare dintre cele două şiruri de date de intrare se va construi câte o matrice de adiacență reprezentate prin doi vectori de întregi mari.

Al i-lea număr semnifică în reprezentarea binară modul în care se intersectează strada i cu toate celelalte străzi.

Se vor intersecta (aplicând operatorul &) cele două hărți de biți şi obținem intersecțiile pe care se lucrează simultan.

În memorie, aceste hărți de biți sunt realizate de fapt simetric (poziția 1 fiind poziția cea mai din dreapta).

Şir 1	A	Sir 2	B	C=A*B	Rezultat
1,4,3,7	0001000 0000000 0001001 1010000 0000000 0000000 0010000	1,4,5,7,3	0001000 0000000 0000001 1000100 0001001 0000000 0010100	0001000 0000000 0000001 1000000 0000000 0000000 0010000	1 4 3 7

⋙PROGRAM:

```
#include <fstream.h>
unsigned long int a[32],b[32],c[32],n,m,x,y,k;
ifstream f("focsani.in ");
ofstream g("focsani.out");
int main()
```



```
{//citirea traseelor
f>>k>>n>>m>>x;
for(int i=2;i<=n;i++)
 {f>>y;
  a[x]=a[x]|(1<<y);
  a[y]=a[y]|(1<<x);
 x=y;
 }
f>>x;
for( i=2;i<=m;i++)
{
 f>>y;
  b[x]=b[x]|(1<<y);
  b[y]=b[y]|(1<<x);
  x=y;
  }
//determinarea strazilor comune
for(i=1;i<=k;i++)  c[i]=a[i]&b[i];
//parcurgere deasupra diagonalei principale
// pentru a afişa o singură dată o pereche rezultat
for ( i=1;i<=k ;i++)
{
  for(int j=k;j>=i+1;j--)
    if (c[i]& (1<<j)) g<<i<<' '<<j<<endl;
}
g<<endl;
return 0;
}
```

A^B MOD C

⋙ENUNȚ:

Se dau trei numere naturale A,B,C. Să se calculeze numărul A^Bmod C.

Date de intrare:

În fişierul de intrare **abc.in** se vor găsi pe prima linie cele trei numere A,B,C.

Date de ieşire:

În fişierul de ieşire **abc.out** se va afişa numărul cerut în enunţ.

Restricţii:
0 < A,B,C < 2 000 000 000
Timp de execuţie pe test: 0.1 secunde

»»»SOLUŢIE:

Vom considera reprezentarea binară a numărului B. Folosind formula de recurenţă:

$$[A^B] = \begin{cases} [A^{B/2}]^2 & \text{dacă B este par} \\ [A^{B/2}]^2 * A & \text{dacă B este impar} \end{cases}$$

Vom parcurge numărul B pornind de la bitul cel mai semnificativ spre cel mai nesemnificativ, la fiecare pas ridicând la pătrat numărul şi înmulţindu-l cu A în cazul unui bit 1 întâlnit. Complexitatea algoritmului este logB, foarte mică în comparaţie cu timpul de execuţie acordat.

»»»PROGRAM:

```
// O(logB) sau O(1) în solutia aceasta;Memoria folosita: O(1)
#include<stdio.h>
FILE *in,*out;
unsigned long long ABC,A,B,C;
unsigned long i;
int main()
{
   in=fopen("abc.in","r");
   fscanf(in,"%lld %lld %lld",&A,&B,&C);
   fclose(in);
   for(ABC=1,i=1<<31;i;i>>=1)
    if (i&B) ABC=(((ABC*ABC)%C)*A)%C;
      else   ABC=(ABC*ABC)%C;
   out=fopen("abc.out","w");
   fprintf(out,"%u\n",(unsigned long)ABC);
   fclose(out);
   return 0;
}
```


OPERAŢII

»»»ENUNŢ:

Notăm cu c şi r câtul şi restul împărţirii unui număr nr la 2^k, unde k este un număr natural nenul. Asupra numărului putem efectua succesiv următoarele operaţii:

O1(nr,k) reprezintă transformarea numărului nr în numărul $2^k*(2c+1)+r$ pentru orice rest r
O2(nr,k) reprezintă transformarea numărului nr în numărul $2^{k-1}*c+r$ doar dacă $r<2^{(k-1)}$.

Cerinţă:
Se dau m şi n două numere naturale nenule. Efectuaţi asupra numerelor m şi n operaţii succesive O1 sau O2, pentru valori alese ale lui k, astfel încât după un număr finit de operaţii cele două numere să devină egale, iar valoarea astfel obţinută să fie minimă.

Date de intrare:
Fişierul de intrare **operatii. in** conţine pe o singură linie: m n - două numere naturale nenule, separate printr-un spaţiu, reprezentând cele două numere date.

Date de ieşire:
Fişierul de ieşire **operatii.out** conţine pe cele i+j+3 linii următoarele:
Pe prima linie, **nmin** - numărul natural nenul, reprezentând valoarea minimă obţinută din m şi n prin aplicarea unor succesiuni de operaţii;
Pe a doua linie, **i** - numărul operaţiilor efectuate asupra primului număr m;

Pe următoarele i linii:
op1 k1
...
opi ki - perechi de numere reprezentând operaţia(1 sau 2) şi respectiv valoarea lui k pentru operaţia respectivă, separate printr-un spaţiu;

Pe linia i+2:

j -numărul operațiilor efectuate asupra celui de al doilea număr n;

Pe următoarele j linii:
op1 k1
...
opj kj- perechi de numere reprezentând operația(1 sau 2) și respectiv valoarea lui k pentru operația respectivă, separate printr-un spațiu

Restricții:
1 < m, n ≤2 000 000 000

EXEMPLU	OPERATII.IN	OPERATII.OUT
	11 45	15 2 2 3 1 2 2 2 2 2 4

Timp maxim de executare/test: 1 secundă

⋙SOLUȚIE:

Reprezentând numerele în baza doi, se observă că prima operație inserează un 1 pe poziția k, iar cealaltă realizează ștergerea unui 0 de pe o poziție dată. Numărul minim se realizează ștergând din fiecare număr zerourile și adăugând câțiva de 1 la numărul mai mic pentru a-l egala cu celălalt.

⋙PROGRAM:

```
//O(logM+ logN)
//Memorie folosita: O(1)
#include <stdio.h>
FILE *in,*out;
unsigned long op1,op2,min,minN,minM,M,N,i,j;
int main()
{
    in=fopen("operatii.in ","r");
    out=fopen("operatii.out","w");
    fscanf(in,"%ld %ld",&M,&N);
```



```
    fclose(in);
    for(i=1,op1=0,minM=0;i<=M;i<<=1)
     if (i&M) minM++;
          else op1++;
    for(i=1,op2=0,minN=0;i<=N;i<<=1)
     if (i&N) minN++;
          else op2++;
    if (minM>minN) min=minM; else min=minN;
    fprintf(out,"%u\n",(1<<min)-1);
    op1+=min-minM;
    fprintf(out,"%d\n",op1);
    for(i=1<<31,j=1;i>1;j++,i>>=1)
     {
       if ((i&M)==0 && i<=M) fprintf(out,"2 %d\n",33-j);
     }
     if (min!=minM)
      for(i=1;i<=min-minM;i++)
       fprintf(out,"1 1\n");
     op2+=min-minN;
     fprintf(out,"%d\n",op2);
     for(i=1<<31,j=1;i>1;j++,i>>=1)
       if ((i&N)==0 && i<=N) fprintf(out,"2 %d\n",33-j);
     if (min!=minN)
      for(i=1;i<=min-minN;i++)
       fprintf(out,"1 1\n");
      fclose(out);
      return 0;
}
```

JOCUL NIM

⋙ENUNȚ:

Nim este denumirea dată unui joc de două persoane. Inițial există N grămezi cu Gi elemente fiecare (i=1,N). Jucătorii iau alternativ un anumit număr de obiecte dintr-o singură grămadă. Aceştia pot selecta orice grămadă şi pot lua un număr de obiecte cuprins între 1 şi numărul de obiecte din acea grămadă. Când nu mai rămâne nici un obiect în nici o grămadă, persoana care trebuie să mute în acel moment pierde.

Cerință:

Dându-se un număr N de grămezi, precum şi posibilitatea de a muta primul, precizați dacă aveți strategie de câştig sau nu şi dacă da, precizați prima mutare pe care o veți face pentru a câştiga.

Date de intrare:

În fişierul de intrare **nim.in** se află pe prima linie un număr natural N reprezentând numărul de grămezi.

Pe următoarea linie, separate printr-un spațiu, se află N numere reprezentând numărul de obiecte din fiecare grămadă.

Date de ieşire:

În fişierul de ieşire **nim.out** se află pe prima linie răspunsul DA/NU la prima cerință.

Dacă răspunsul este DA, atunci pe a doua linie a fişierului se vor afla două numere, separate printr-un spațiu, reprezentând indicele grămezii din care se vor lua obiecte precum şi numărul acestora.

Restricții:

0 < N < 50000
0 < Gi < 2 000 000 000
Timp de execuție pe test: 0.1 secunde

EXEMPLUL 1

NIM.IN	NIM.OUT
2	DA
2 4	2 2

EXEMPLUL 2

NIM.IN	NIM.OUT
3	NU
5 1 4	

*Se presupune că adversarul joacă optim.

⋙SOLUȚIE:

Pentru a rezolva această problemă trebuie să definim câteva noțiuni de bază. Să considerăm pentru început reprezentările binare ale numerelor de elemente din fiecare grămadă. Numim **SUMA NIM** a unui grup de numere ca fiind numărul rezultat din aplicarea operatorului binar **sau exclusiv(^)** acelor numerelor. De exemplu pentru un şir de 3 numere 38, 7, 2 suma nim va fi 38 ^7^2.

Modul de funcționare al operației ^:

N=3
38 = 0 1 1 0 0 1
7 = 1 1 1 0 0 0
2 = 0 1 0 0 0 0
SN = 1 1 0 0 0 1 (1,3,2,0,0,1)

Starea fiecărui bit din sumă este dată de paritatea sumei pe coloana din dreptul acestuia.

Să ne întoarcem la problemă. Se poate observa că *suma nim* a unor valori nule este 0. Astfel, în finalul jocului *suma nim* a grămezilor va fi 0.

Datorită principiului de funcționare al operației **sau exclusiv**, dacă la început *suma nim* a grămezilor este 0 şi mai sunt elemente de luat, în orice mod am alege obiecte dintr-o grămadă, *suma nim* rezultată va fi diferită de 0.

Invers dacă *suma nim* este diferită de 0, atunci întotdeauna există o mutare prin care se ajunge la o *sumă nim* egală cu 0. De aici tragem concluzia de bază în rezolvarea problemei. Dacă *suma nim* inițială este 0, atunci cel care începe jocul nu are strategie de câştig, altfel acesta poate câştiga.

Pentru a determina grămada şi numărul de obiecte ce trebuie luate se calculează *suma nim* inițială. Se alege o grămadă al cărei număr are primul bit 1 (cel mai semnificativ) pe aceeaşi poziție cu cel al *sumei nim*. Aceasta va fi grămada aleasă pentru mutare. Se recalculează *suma nim* excluzând această grămadă, iar numărul de obiecte ce trebuie luate este dat de diferența dintre numărul de elemente din grămada aleasă şi noua *sumă nim*.

Complexitatea algoritmului este O(N).

⋙PROGRAM:

```
//Memoria folosita: 200 KB (se poate O(1) dacă se citeste fişierul de intrare de 2 ori)
#include<stdio.h>
#define MAXN 50001
unsigned long g[MAXN],SN,N,i,bit;
FILE *in,*out;
int main()
{
```

```
in=fopen("nim.in","r");
out=fopen("nim.out","w");
fscanf(in,"%ld",&N);
for(i=1;i<=N;i++)
fscanf(in,"%ld",&g[i]);
fclose(in);
for(i=1,SN=0;i<=N;i++)
SN^=g[i];
for(bit=1<<31;bit;bit>>=1)
if (bit&SN) break;
if (bit)
  { fprintf(out,"DA\n");
    for(i=1;i<=N;i++)
    if (bit&g[i]) {fprintf(out,"%d %ld\n",i,g[i]-(SN^g[i]));break;}
  }
  else fprintf(out,"NU\n");
fclose(out);
return 0;
}
```

CIRCUITE

»»»ENUNȚ:

Pe o placă de circuite integrate sunt lipiți N pini și sunt legați între ei prin niște linii fine schițate pe placă. Acestea au fost concepute în așa fel încât să transmită informații doar într-un singur sens. Pentru a putea funcționa corect, orice semnal transmis dintr-un pin trebuie să ajungă la toate celelalte.

Cerință:

Fiind date N și configurația rețelei, determinați dacă placa este bună sau nu și afișați pentru fiecare pin starea legăturii cu ceilalți pini.

Date de intrare:

Pe prima linie a fișierului **board.in** se află N, numărul de pini.

Pe următoarele N linii se află câte N numere 0 sau 1.

Pe linia i+1, cel de-al j-ulea număr va fi 1 dacă există legătură de la pinul i către pinul j sau 0 în caz contrar.

Date de ieșire:

Pe prima linie a fișierului de ieșire **board.out** se află răspunsul *DA*, în cazul în care se pot transmite informații de la orice pin către oricare alt pin sau *NU* în caz contrar.

Pe următoarele N linii se pot afișa câte N valori 0 sau 1. Pe linia i+1, al j-ulea număr va fi 1 dacă din pinul i se pot transmite informații către pinul j; sau 0 dacă nu este posibil acest lucru.

Restricții:

$0 < N < 32$

»»»SOLUȚIE:

Se va aplica algoritmul Roy-Warshall pentru determinarea matricei lanțurilor într-un vector de numere. Simularea acestui algoritm pe biți va duce la economie de memorie dar și la un timp mai mic de execuție. Această implementare se poate extinde pentru un **n** cu valori destul de mari cu rezultate spectaculoase.

»»»PROGRAM:

```
#include <fstream.h>
#define MaxN 32
ifstream f("board.in");
ofstream g("board.out");
unsigned long int a[MaxN],c,n,m,i,j,k,el,depl,masca;
void Citire()
{
f>>n;
for(i=1;i<=n;i++)
 for(j=1;j<=n;j++)
    {f>>c;
  if(c)
   a[i]=a[i]|(1<<j%32);
     }
f.close();
}
void Warshall()
{
 for (int k=1;k<=n;k++)
   for (int i=1;i<=n;i++)
    if (a[i]&(1<<k%32))  a[i]=a[i]|a[k];
}
```

```
void Afisare()
{int DA=1;
for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
   if (! (a[i]&(1<<j%32)))
      DA=0;
   if (DA) g<<"DA"<<endl;
       else g<<"NU"<<endl;
for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
   if (a[i]&(1<<j%32))
      g<<1<<' ';
      else g<<0<<' ';
   g<<endl;
 }
g.close();
}
int main()
{Citire();
Warshall();
Afisare();
return 0;
}
```

VECINI

»»»ENUNȚ:

În Drumul Taberei există un bloc ciudat. În primul an când a fost construit (să presupunem anul 1) avea un singur etaj, după care în fiecare an un nou etaj, aşa încât în anul X va avea X etaje. Dar acesta nu e singurul lucru ciudat. Şi felul în care este ocupat fiecare etaj este foarte ciudat.

La etajul 1 (numerotarea se face de jos în sus) stă tot timpul administratorul, deci este ocupat. De asemenea ultimul etaj, fiind nou, este tot timpul ocupat. Restul etajelor însă sunt ocupate (sau libere) după următoarele reguli:

- dacă anul trecut etajul curent şi etajul de sub acesta au fost ocupate atunci etajul va fi liber anul acesta;
- dacă anul trecut etajul curent şi etajul de sub acesta au fost libere atunci etajul va fi liber şi anul acesta;
- dacă anul trecut etajul curent a fost ocupat, iar cel de sub acesta a fost liber, atunci etajul curent va fi ocupat;
- dacă anul trecut etajul curent a fost liber, iar cel de sub el a fost ocupat, atunci etajul curent va fi ocupat.

Cerință:

Scrieți un program care determină configurația etajelor după N ani, adică în anul N.

Date de intrare:

Fişierul de intrare **vecini.in** conține doar numărul N pe prima linie.

Date de ieşire:

În fişierul **vecini.out** se vor scrie N numere separate printr-un spațiu. Numerele reprezintă starea fiecărui etaj, începând cu etajul 1. Reprezentarea este 0 pentru liber şi 1 pentru ocupat.

Restricții:

1 < N <100 001

Timp maxim de execuție pe test: 0.1 secunde

EXEMPLU	VECINI.IN	VECINI.OUT
	5	10 0 0 1

»»»SOLUȚIE:

Considerăm blocul ca fiind un vector de biți. Ideea de bază de la care se porneşte este observarea schimbărilor atunci când sunt adăugate noi etaje.

Constatăm cu uşurință că adăugând 2^1 etaje la un bloc de 5 etaje (10001) noul bloc va avea în plus doi de unu (1010101).

Cele 2 etaje cu unu inițiale nu şi-au schimbat pozițiile, în schimb, la deplasamentul 2 față de pozițiile acestora a apărut câte un 1. Generalizând, pentru orice număr N de etaje (N=putere a lui 2) adăugate, va apărea pentru fiecare 1 din vector la deplasamentul N față

de poziția respectivă un nou 1 (noua valoare de pe acea poziție va fi "valoarea inițială" ^ 1).

O soluție simplă ar fi să descompunem pe N-1 (deoarece la starea 100 se vor adauga n-1 etaje) în puteri ale lui 2, pentru fiecare putere parcurgem vectorul şi la fiecare valoare 1 întâlnită pe poziția i aplicăm **sau exclusiv (^)** 1 pe poziția i+deplasament. Complexitatea este O(n*logn).

O complexitate liniară se poate obține astfel. Descompunem pe N-1 în puteri de 2. Folosim un vector în care stocăm pozițiile pe care vom avea în final valoarea 1. Inițial avem doar deplasamentul 1. Pentru fiecare putere adăugăm în vector noile deplasamente obținute din cele deja existente la care se adaugă puterea respectivă.

»»»PROGRAM:

```
//complexitate O(n)
#include <fstream.h>
#define MAXN 100002
int l,sol[MAXN],depl[MAXN],n,i,j;
ifstream in("vecini.in ");
ofstream out("vecini.out");
int main()
{
in>>n;
depl[l=1]=1;
for(i=1;i<=n;i<<1)
if (i&n)
  {
    for(j=1;j<=l;j++) depl[l+j]=depl[j]+i;
    l=l<<1;
}
for(i=1;i<=l;i++)
   sol[depl[i]]=1;
for(i=1;i<=n;i++)
   out<<sol[i]<<" ";
out. close();
return 0;
}
```


DOILAN

»»»ENUNȚ:

Se dă un număr natural N. Să se calculeze numărul 2^N.

Date de intrare:
Pe prima linie se va găsi: **N** – un număr natural nenul.

Date de ieşire:
În fişierul **doilan.out** se va afla numărul: $\mathbf{2^N}$.

Restricții:
0 < N < 100 001
Timp de execuție / test: 0.8 secunde pe un Pentium III la 1GHz.

EXEMPLU	DOILAN.IN	DOILAN.OUT
	7	128

Problemă propusă de Marin Radu

»»»SOLUȚIE:

Soluția acestei probleme constă în a grupa câte 9 cifrele numărului scris în baza 10. La fiecare înmulțire a numărului mare folosim 2^{29} în loc de 2 şi reducem de 29 ori numărul de înmulțiri.

Formula care dă complexitatea algoritmului este [3*(N*N/ 18)] /261. Pentru N=100 000, numărul de operații este 12 milioane, număr ce se încadrează în 0.6 secunde.

»»»PROGRAM:

```
// O(N*N/780) 0.6 secunde
// Memoria folosita: 123KB
#include <stdio.h>
#define cifre 9
#define MAXN 110000/cifre
FILE *in,*out;
int N,size;
```

```
unsigned long v[MAXN];
void calc2laN()
{int lim,rest,i,j;
unsigned long long r;
unsigned long unitate;
for(unitate=1,i=1;i<=cifre;i++,unitate*=10);
lim=N/29+1;
rest=N%29;
size=1;
v[size-1]=1;
for(i=1;i<lim;i++)
{
for(r=0,j=0;j<size;j++)
{unsigned long long P=r+((unsigned long long)(v[j])<<29);
 r=P/unitate;v[j]=P-r*unitate;
}
if (r) v[size++]=r;
}
for(r=0,j=0;j<size;j++)
{unsigned long long P=r+((unsigned long long)(v[j])<<rest);
 r=P/unitate; v[j]=P-r*unitate;
}
if (r) v[size++]=r;
size--;
}
void afiseaza()
{int i,j,c[20];
 out=fopen("doilan2f.out","w");
 for(j=1;v[size];j++,v[size]/=10)
    c[j]=v[size]%10;
 for(i=j-1;i>=1;i--)
 fprintf(out,"%d",c[i]);
 for(i=size-1;i>=0;i--)
 {for(j=1;j<=cifre;j++,v[i]/=10)
  c[j]=v[i]%10;
  for(j=cifre;j>=1;j--)
  fprintf(out,"%d",c[j]);
 }
  fclose(out);
}
void citeste()
```



```
{
   in=fopen("doilan.in ","r");
   fscanf(in,"%d",&N);
   fclose(in);
}
int main()
{
    citeste();
    calc2laN();
    afiseaza();
    return 0;
}
```

VITEZA

⋙ENUNȚ:

În proiectarea cu procesoare comunicarea este foarte importantă. Pentru ca două procesoare să comunice între ele este necesar ca acest lucru să se întâmple la aceeaşi viteză. Să presupunem că procesorul A comunică la viteza maximă de 12 (biți pe microsecundă). Acest lucru înseamnă că procesorul A poate comunica la viteza de 12, dar şi la orice divizor al acestuia (1,2,3,4,6).

Când două procesoare care au viteza de comunicație diferită vor să comunice, în mod obişnuit, începe o negociere. E clar că amândouă pot să comunice la viteza 1 şi încep cu această viteză. Dacă pot comunica, incrementează viteza. Astfel se stabileşte viteza maximă la care pot comunica amândouă. Însă acest procedeu este foarte dezavantajos pentru că stabilirea vitezei durează foarte mult, în special în cazul vitezelor foarte mari. Aşa că inginerii au proiectat un al treilea procesor care primeşte vitezele celor procesoare şi le comunică înapoi viteza maximă la care pot comunica. Evident fiind vorba de procesoare, toate datele sunt în binar.

Cerință:

Scrieți un program care simulează activitatea celui de-al treilea procesor, adică având două numere în binar, reprezentând vitezele celor 2 procesoare, află viteza maximă la care pot comunica.

Date de intrare:

Pe prima linie a fişierului de intrare **viteza.in** este scris numărul N, reprezentând numărul de biţi a vitezelor.

Pe a doua linie sunt daţi biţii vitezei primului procesor, despărţiţi de un spaţiu. Biţii sunt daţi în ordine, începând de la cel mai puţin semnificativ.

Pe a treia linie sunt daţi biţii vitezei celui de-al doilea procesor, despărţiţi de un spaţu. Biţii sunt daţi în ordine, începând de la cel mai puţin semnificativ. Ambele viteze sunt nenule.

Date de ieşire:

Prima linie a fişierului **viteza.out** va conţine N biţi, reprezentând viteza maximă la care procesoarele pot comunica. Biţii trebuie de asemenea despărţiţi de câte un spaţiu şi daţi în ordine, începând cu cel mai puţin semnificativ.

Restricţii

$1 \le N \le 10000$

Timp maxim de execuţie/test: 0.1 secunde

EXEMPLU

VITEZA.IN	VITEZA.OUT	COMENTARII
5 0 0 1 1 0 0 0 0 1 0	0 0 1 0 0	Procesorul A comunică la 12, iar procesorul B la 8. Cea mai mare viteză la care pot comunica amândouă este 4.

Problemă propusă de Marius Andrei la concursul. Campion

SOLUŢIE:

Algoritmul ce trebuie aplicat este un CMMDC pentru 2 numere mari. Complexitatea este N*N, adică aproximativ 3 secunde. Pentru a obţine limita de 0.1 s, vom trece numerele din baza 2 în baza 2^{32} (baza 4 miliarde) grupând biţii în întregi pe 32 biţi. Astfel numărul de operaţii efectuate la o scădere va fi N/32 şi deci complexitatea va scădea prin acest factor constant. Soluţia propusă a luat punctaj maxim în cadrul concursului.

PROGRAM:

```
//O(N*N/BITI) 0.1 secunde
//Memoria folosita: 24KB
```



```
#include <fstream.h>
#define MAXN 10010
#define BITI 63 //poate lua orice valoare de la 1 la 63 (1 slow, 63 fastest)
ifstream in("viteza.in ");
ofstream out("viteza.out");
short int rez[MAXN];
long long powerLO,powerHI,bit[BITI+10];
long long NB1[MAXN/BITI+10],NB2[MAXN/BITI+10];
int sizeNB1,sizeNB2,N,L;
void citesteDate()
{int X,sw,zero0=0,zero1=0,id,i;
 in>>N;
 bit[0]=1;
 for(i=1;i<BITI;i++)
 bit[i]=bit[i-1]<<1;
 powerLO=bit[BITI-1];
 powerHI=powerLO<<1;
 NB1[0]=NB2[0]=1;
 for(i=1,id=-1,sw=0,sizeNB1=1;i<=N;i++)
 {
 in>>X;
 if (X && zero0==0) {zero0=i;sw=1;}
 if (sw){
        id++;
        if (id==BITI){sizeNB1++;id=0;}
        if (X) NB1[sizeNB1]+=bit[id];
       }
 }
 for(i=1,id=-1,sw=0,sizeNB2=1;i<=N;i++)
 {
 in>>X;
 if (X && zero1==0) {zero1=i;sw=1;}
  if (sw)
   {
   id++;
   if (id==BITI) {id=0;sizeNB2++;}
   if(X) NB2[sizeNB2]+=bit[id];
   }
 }
L=zero0;
```

```
if (L>zero1) L=zero1;
in. close();
for(;NB1[sizeNB1]==0;sizeNB1--);
for(;NB2[sizeNB2]==0;sizeNB2--);
}
//***********o. k.*************************************
void scadere12()
{ int carry,i;
for(i=1,carry=0;i<=sizeNB2 || carry;i++)
 {
  NB1[i]-=NB2[i]+carry;
  if (NB1[i]<0) {NB1[i]+=powerHI; carry=1;}
  else carry=0;
  }
for(;NB1[sizeNB1]==0;sizeNB1--);
}
//***********ok*************************************
void scadere21()
{int carry,i;

for(i=1,carry=0;i<=sizeNB1 || carry;i++)
 {
  NB2[i]-=NB1[i]+carry;
  if (NB2[i]<0) {NB2[i]+=powerHI; carry=1;}
  else carry=0;
  }
for(;NB2[sizeNB2]==0;sizeNB2--);
}
//***********ok*************************************
void shiftNB1()
{int t=0,i,odd;
for(i=sizeNB1;i>0;i--)
{odd=NB1[i]&1;
NB1[i]=(NB1[i]>>1);
if (t) NB1[i]+=powerLO;
t=odd;
}
for(;NB1[sizeNB1]==0;sizeNB1--);
}
//***********ok*************************************
void shiftNB2()
```



```
{int t=0,i,odd;

for(i=sizeNB2;i>0;i--)
{odd=NB2[i]&1;
NB2[i]=(NB2[i]>>1);
if (t) NB2[i]+=powerLO;
t=odd;
}
for(;NB2[sizeNB2]==0;sizeNB2--);
}
//***********ok*************************************
void scadere()
{int equal;
if (sizeNB1>sizeNB2) scadere12();
 else if (sizeNB1<sizeNB2) scadere21();
else
{
 for(equal=sizeNB1;equal>0 && NB1[equal]==NB2[equal];equal--
);
 if (equal){
        if (NB1[equal]>NB2[equal]) scadere12(); else scadere21();
       }
  else sizeNB1=0;
}
}
//***********ok*************************************
void update()
{int i,j;
if (sizeNB1==0)
for(i=1;i<=sizeNB2;i++)
 for(j=0;j<BITI;j++)
 {rez[L++]=NB2[i]&1;NB2[i]=NB2[i]>>1;}
else
for(i=1;i<=sizeNB1;i++)
 for(j=0;j<BITI;j++)
 {rez[L++]=NB1[i]&1;NB1[i]=NB1[i]>>1;}
}
//***********ok*************************************
void GCD_NONbinar()
{
for(;sizeNB1 && sizeNB2;)
```

```
    {
    if ((NB1[1]&1)!=(NB2[1]&1)) {if (NB1[1]&1) shiftNB2(); else
shiftNB1();}
        else
          scadere();
    }
    update();
    }

    void afiseazaRez()
    {int i;
    for(i=1;i<=N;i++)
    out<<rez[i]<<" ";
    out. close();
    }
    int main()
    {
    citesteDate();
    GCD_NONbinar();
    afiseazaRez();
    return 0;
    }
```

BILE

»»»ENUNȚ:

Avem N bile (numerotate de la 1 la N), oricare două bile având greutăți diferite. Pentru a descoperi bila cu greutatea mediană (deci cea de a (N+1)/2-a bilă, în ordinea greutăților) putem utiliza o balanță cu două platane.

Putem plasa câte o bilă pe fiecare platan şi astfel putem afla care dintre cele două bile este mai grea. Ca urmare a rezultatelor obținute în urma unui set de astfel de cântăriri, putem elimina unele dintre bile, despre care putem afirma cu siguranță că nu au greutatea mediană.

De exemplu să considerăm N=5 bile între care s-au efectuat M=4 cântăriri obținând următoarele rezultate:

Bila 2 este mai grea decât bila 1.
Bila 4 este mai grea decât bila 3.
Bila 5 este mai grea decât bila 1.
Bila 4 este mai grea decât bila 2.

Din rezultatele de mai sus, deşi nu putem determina exact bila cu greutatea mediană, putem afirma că bila 1 şi bila 4 nu pot avea greutatea mediană, deoarece bilele 2, 4, 5 sunt mai grele decât bila 1, iar bilele 1, 2, 3 sunt mai uşoare decât bila 4. Prin urmare putem elimina aceste două bile.

Cerință:

Scrieți un program care să determine pe baza unui set de cântăriri dat numărul de bile care nu pot avea greutatea mediană.

Date de intrare:

Fişierul de intrare **bile.in** conține pe prima linie numărul natural N, reprezentând numărul de bile şi numărul natural M, reprezentând numărul de cântăriri, separate printr-un spațiu.

Fiecare dintre următoarele M linii conține două numere naturale cuprinse între 1 şi N cu semnificația "bila corespunzătoare primului număr este mai grea decât bila corespunzătoare celui de al doilea număr". Este posibil ca în fişierul de intrare să existe mai multe cântăriri pentru aceeaşi pereche de bile. Evident la fiecare cântărire obțineți acelaşi rezultat.

Date de ieşire:

Fişierul de ieşire **bile.out** conține o singură linie pe care se află numărul de bile care nu pot avea greutatea mediană.

Restricții:

N număr natural impar, 0<N≤300, M≤5000
Timp maxim de executie/test: 0.1 secunde

EXEMPLU	BILE.IN	BILE.OUT
	5 4 2 1 4 3 5 1 4 2	2

Problemă propusă de prof. Emanuela Cerchez la concursul. Campion

»»»SOLUȚIE:

Problema constă în a determina folosind două parcurgeri în adâncime (sau lațime, este indiferentă alegerea) a grafului ce are drept

noduri cele n bile şi respectiv muchii orientate cele m cântăriri. La fiecare parcurgere, pentru fiecare nod se menţine o mulţime cu nodurile mai mici respectiv mai mari ca rang.

Deoarece mulţimea unui părinte depinde de submulţimile fiilor, pentru fiecare muchie se va realiza reuniunea mulţimii fiului(nodul spre care indică muchia) şi mulţimea părinte. Având M operaţii de reuniune pe o mulţime cu N elemente reprezentată printr-un vector complexitatea algoritmului ajunge la N*M. Pentru restricţiile impuse, această soluţie este mulţumitoare având în vedere că se poate obţine punctajul maxim fără prea mare efort.

Majoritatea concurenţilor s-ar opri aici (asta se şi recomandă într-un concurs în care timpul este foarte presant). În schimb, dacă aveţi destul timp la dispoziţie, puteţi face unele mici modificări ce v-ar putea clasa înaintea tuturor celor care au obţinut acelaşi punctaj dar folosind o soluţie mai puţin rapidă (dacă nu, oricum veţi fi apreciat pentru originalitatea soluţiei).

Revenind la problemă, ceea ce se poate face pentru a micşora numărul de procesări este să optimizăm operaţia de reuniune (undeva de vreo 30 de ori mai rapidă, dacă se poate). Se observă uşor că un element va fi în mulţimea finală dacă apare cel puţin în una din mulţimile ce se reunesc. Pe înţelesul calculatorului ar fi cam aşa: v[i]= v1[i] | v2[i]. Cum operaţia SAU(|) binară se realizează în acelaşi timp pe toţi biţii unui număr iar valoarea care ne interesează nu poate fi decât 0 sau 1 vom reprezenta mulţimile prin vectori binari. Evident vom alege tipul long pentru a efectua operaţia pe cât mai multe elemente simultan (adică 32).

În final complexitatea algorimului va fi N*M/32 (fără a mai lua în calcul avantajul economisirii memoriei) ceea ce înseamnă că timpul de execuţie va fi de 30 de ori mai mic (vă puteţi imagina ce diferenţă este faţă de rezolvarea obişnuită), lucru ce va bloca la 0 indicatorul timer-ului pentru toate testele.

⋙PROGRAM:

```
// O(N*M/SPEED_MULTIPLIER)
// Memorie folosita: 102Kb
#include <stdio.h>
#define SPEED_MULTIPLIER 32  // 1x - normal, 32x - foarte rapid
#define DIV 5
#define MOD 31
#define MAXN 302
```



```
FILE *in=fopen("bile.in ","r"),*out=fopen("bile.out","w");
int n,m,mark[MAXN],sel[MAXN],maimic[MAXN],maimare[MAXN];
char A[MAXN][MAXN];
unsigned long  M[MAXN][MAXN/SPEED_MULTIPLIER+1];
void citesteDate()
{   int i,a,b;
    fscanf(in,"%d %d",&n,&m);
    for(i=1;i<=m;i++)
    {fscanf(in,"%d %d",&a,&b);
     A[a][b]=1;
    }
    fclose(in);
}
void BFS(int nod)
{int i,j,lim;
  if (sel[nod]) return;
  sel[nod]=1;
  for(i=0,lim=n/SPEED_MULTIPLIER;i<=lim;i++) M[nod][i]=0;
  for(i=1;i<=n;i++)
   if (A[nod][i])
    {  BFS(i);
       for(j=0;j<=lim;j++) M[nod][j]|=M[i][j];
}
  M[nod][nod>>DIV]|=(unsigned long)(1)<<(nod&MOD);
}
void calcMaiMic()
{ int i,j;
  for(i=1;i<=n;i++) sel[i]=0;
  for(i=1;i<=n;i++)
  if (!sel[i])
        BFS(i);
  for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
   if ( (M[i][j>>DIV]&((unsigned long)(1)<<(j&MOD)))  && j!=i)
maimic[i]++;
}
void calcMaiMare()
{ int i,j;
  for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
  if (A[i][j]==1)
```

```
        {A[i][j]=0;A[j][i]=2;}
      for(i=1;i<=n;i++) sel[i]=0;
      for(i=1;i<=n;i++)
      if (!sel[i]) BFS(i);
      for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
       if ( (M[i][j>>DIV]&((unsigned long)(1)<<(j&MOD)))  && j!=i)
maimare[i]++;
    }
    void afiseaza()
    {int i,BAD=0;
      for(i=1;i<=n;i++)
      {
       if (maimic[i]>(n>>1)) mark[i]=1;
       if (maimare[i]>(n>>1))mark[i]=1;
      }
      for(i=1;i<=n;i++)
      if (mark[i]) BAD++;
      fprintf(out,"%d\n",BAD);
      fclose(out);
    }
    int main()
    {
        citesteDate();
        calcMaiMic();
        calcMaiMare();
        afiseaza();
        return 0;
    }
```

MARKET

»»»ENUNȚ:

Pescarul Clement a prins N peşti şi i-a dus la piața de peşte. Peştele **i** are masa **Ti**. Oamenii din piață împachetează peştele în pachete de greutăți diferite. Fiecare pachet poate conține unul sau mai mulți peşti.

Cerința:

Presupunând că eşti primul client, poți alege greutatea pachetului tău. Câte mase diferite pentru pachet se pot obține cu cei N peşti ?

Date de intrare:

Pe prima linie a fişierului de intrare **market.in** se găseşte numărul N.

Pe următoarele N linii se află câte un număr reprezentând masele fiecărui peşte.

Date de ieşire:

Fişierul de ieşire **market.out** va conține un număr întreg K reprezentând numărul posibilităților de a alege masa pachetului.

Restricții:

0 < N <501 reprezintă numărul de peşti

0 < Ti < 1001 reprezintă masa peştelui i

Timp de execuție pe test: 0.2 secunde

EXEMPLU:	**MARKET.IN**	**MARKET.OUT**
	5 27	800
		200
		354
		18
		182

Problemă probusă la BOI 2000

»»»SOLUȚIE:

Problema are o rezolvare foarte simplă, fiind cunoscută la primele lecții de PD: dacă există pesti[i] astfel încât MEM[i-pesti[i]]=1 marcăm cu MEM[i]=1.

Ceea ce pare a fi mai greu este însă reducerea complexității care în cazul defavorabil este **T**max (**N**max*Nmax)/2 estimată în numărul de operații aproximativ 125 milioane, adică 4 secunde.

Pentru a coborî sub limita de 0.2 secunde vom încerca să optimizăm algoritmul astfel:

Pentru reprezentarea vectorului caracteristic (de biți) vom folosi întregi pe 32 biți (nu 8 cum era obşnuit). La adăugarea unei noi mase nu vom mai lucra cu un singur bit ci cu o întreagă grupare realizând o

singură operație de shiftare. În final complexitatea se reduce prin factorul constant 32 obținându-se timpul de execuție cerut.

»»»PROGRAM:

```
// O(N*N*T/62) 4 Milioane de operatii  0.1 secunde
// Memoria folosită: 62. 5KB
#include <stdio.h>
#define MAXN 501
#define MAXT 1001
#define LIM MAXN*MAXT/32+1
#define rest 31
FILE *in,*out;
unsigned long MEM[LIM+1];
int pesti[MAXN];
int N,T;
unsigned long mask[32];
void citeste()
{int i;
 in=fopen("market.in ","r");
 fscanf(in,"%d",&N);
 for(i=1;i<=N;i++)
 fscanf(in,"%d",&pesti[i]);
 fclose(in);
 //preprocesare
 for(i=1;i<=31;i++)
 mask[i]=((1<<i)-1)<<(32-i);
}
void SHIFT(int s,int bz)
{unsigned long  r,d;
d=bz+(s>>5);  //noul deplasament
r=s & rest; //nr. de biti care trebuie mutati în urmatorul deplasement
if (r)
MEM[d+1] |= (MEM[bz] & mask[r])>>(32-r);
MEM[d] |= MEM[bz]<<r;
}
void calculeaza()
{int i,j,lim=0,sum=0;
MEM[0]=1;// suma 0 s-a obtinut
for(i=1;i<=N;i++)
{for(j=lim;j>=0;j--)
```



```
    if (MEM[j]) SHIFT(pesti[i],j);
sum+=pesti[i];
lim=(sum>>5)+1;
}
}
void afiseaza()
{int j,i,count=0,lim=LIM;
MEM[0]>>=1; //suma zero nu o consideram
for(i = 0;  i < lim; i++)
   for(; MEM[i] ; MEM[i] >>= 1) count  +=  MEM[i] & 1;
out=fopen("market.out","w");
fprintf(out,"%d\n",count);
fclose(out);
}
int main()
{
 citeste();
 calculeaza();
 afiseaza();
 return 0;
}
```

RADAR

»»»ENUNȚ:

În războiul oamenilor dus împotriva roboților, forțele aliate au un as în mânecă, un super-RADAR. Acesta are capacitatea de a detecta obstacolele şi bazele inamice.

De curând, un inginer priceput a construit un modul pe care l-a anexat radarului îmbunătățindu-i performanța. Astfel, radarul este acum capabil să estimeze pentru fiecare bază inamică distanța unui drum minim plecând din baza aliaților.

Considerând terenul de luptă de forma unui dreptunghi de dimensiuni N*M alcătuit din pătrate cu latura de o unitate, trupele aliaților se pot deplasa din poziția curentă în una din cele 4 direcții posibile E,V,N,S.

Aliații se decid să atace bazele inamice în funcție de distanța dintre acestea şi baza aliată. Astfel prima bază distrusă va fi cea mai apropiată de ei şi ultima va fi cea mai depărtată. Rolul vostru în toată povestea este acela de a scrie soft-ul modulului din radar.

Cerință:
Dându-se dimensiunile terenului, coordonatele obstacolelor, coordonatele bazei aliate precum şi cele ale bazelor inamice, determinați ordinea în care trebuie atacate bazele inamice precum şi distanța minimă până la acestea.

Date de intrare:
radar.in
N şi M - pe prima linie dimensiunile terenului
K - pe a doua linie numărul de obstacole
X1 şi **Y1** - pe a treia linie coordonatele primului obstacol
..... .
Xk şi **Y**k - pe linia K+2 coordonatele obstacolului K.
I - pe linia K+3 numărul de baze duşmane
X1 şi **Y1** - pe linie K+4 coordonatele primei baze duşmane(cu Id-ul 1)
..... .
Xi şi **Y**i - pe linia K+I+3 coordonatele ultimei baze duşmane(cu Id-ul I)
Xb şi **Y**b - pe ultima linie a fişierului de intrare sunt coordonatele bazei aliaților.

Date de ieşire:
radar.out
Dist1 **id**1 - pe prima linie distanța până la cea mai apropiată bază şi Id-ul acesteia.
.....
DistT **id**T - pe linia T(T≤I= distanța până la cea mai depărtată bază precum şi Id-ul ei.

Precizări:
· Trupele aliaților nu pot trece printr-un obstacol şi nu pot părăsi suprafața terenului.
· Numerotarea pătratelor se face pornind din colțul stânga-sus (0,0) şi terminând în colțul dreapta-jos (N,M).
· Fişierul de ieşire va conține cel mult I linii (I este numărul de baze inamice).
Dacă există baze inamice care nu pot fi atacate de aliați (nu există drum până la ele) atunci acestea nu se vor scrie, în schimb pe ultima linie a fişierului de ieşire se va scrie mesajul "Mission Incomplete! Targets Remaining: " urmat de numărul de baze inamice nedistruse.

Restricții:
0 < N,M,Xi,Yi < 1001
0 < K < 50001
0 < I < 4001

1. Deoarece modulul în care va fi instalat soft-ul are dimensiuni relativ mici, nu veți avea la dispoziție decât 0.4 MB de memorie RAM pentru date.
2. Viteza cu care va fi executat programul este comparabilă cu cea a unui Intel Pentium III la 1 GHz.
3. Pentru a fi eficient, programul trebuie să ruleze în mai puțin de 0.3 secunde, altfel aliații vor fi înfrânți.

Problemă propusă de Marin Radu

»»»SOLUȚIE:

Vom folosi 2 matrici de biți fiecare având dimensiunile M*N elemente. Pe prima matrice vom marca cu 0 pătratele libere iar cu 1 obstacolele şi pătratele atinse. Inițial vom sorta vectorul de baze dupa X şi Y folosind sortarea prin numărare.
Restul algoritmului este un binecunoscut Lee. Optimizat, algoritmul Lee poate rula în 0.1 secunde pentru o matrice N*M la care se adaugă 0.1 pentru citirea datelor (54000 de linii sunt destul de mult) şi se obține un timp bun de 0.2 secunde pentru cazul defavorabil.
Soluția prezentată în continuare rulează în 0.2 secunde şi foloseşte < 0.35 MB de RAM pentru cel mai defavorabil caz.

»»»PROGRAM:

```
// O(N*M)
// Memoria folosita: 349KB
#include <stdio.h>
#define biti 8
#define masca_rest 7
#define shift_bits 3
#define MAXN 1001
#define MAXbase 4001
#define stivaSize 8001
FILE *in,*out;
int T,Xb,Yb,N,M,K,I,count[MAXN];//+1KB
```

```
int mat[MAXN][MAXN/8+1], // +126KB
    Base[MAXN][MAXN/8+1];// +126KB
const int bit[8]={1,2,4,8,16,32,64,128}; //masca de biti
struct baza {short int id,x,y;}; //48 biti pentru reprezentarea unei
                                 //baze (Id şi coordonate)
baza stiva[stivaSize],baze[MAXbase],rsort[MAXbase];// +96KB
void next(int &x)
{
 x++;
 if (x==stivaSize) x=0;
}
void mark(int x,int y)
{
  mat[x][ y >> shift_bits]  |=  bit[y & masca_rest]; //very fast
}
void markbase(int x,int y)
{
   Base[x][ y >> shift_bits]  |=  bit[y & masca_rest]; //very fast
}
int stare(int x,int y)
{
  return mat[x][y >> shift_bits]  &  bit[y & masca_rest];
}
int estebaza(int x,int y)
{
  return Base[x][y >> shift_bits]  &  bit[y & masca_rest];
}
void insertbase(int x,int y,int id)
{
 markbase(x,y);
 baze[id]. id=id;
 baze[id]. x=x;
 baze[id]. y=y;
}
void afiseaza(int x,int y,int dist)
{int min,max,mid;
T--;
min=1;
max=l;
for(;min<=max;)
{mid=(min+max)>>1;
```



```
if (baze[mid]. x==x)
  { if (baze[mid]. y==y)
     {fprintf(out,"%d %d\n",dist,baze[mid]. id);return;}
   else
     if (baze[mid]. y>y)
        max=mid-1;
        else min=mid+1;
  }
  else if (baze[mid]. x>x)
        max=mid-1;
        else min=mid+1;
}
}
void radixsortX()
{int i;
 for(i=0;i<MAXN;i++) count[i]=0;
 for(i=1;i<=l;i++)
 count[baze[i]. x]++;
 for(i=1;i<MAXN;i++)
 count[i]+=count[i-1];
 for(i=l;i>0;i--)
 rsort[count[baze[i]. x]--]=baze[i];
 for(i=1;i<=l;i++)
 baze[i]=rsort[i];
}
void radixsortY()
{int i;
 for(i=0;i<MAXN;i++) count[i]=0;
 for(i=1;i<=l;i++)
 count[baze[i]. y]++;
 for(i=1;i<MAXN;i++)
 count[i]+=count[i-1];
 for(i=l;i>0;i--)
 rsort[count[baze[i]. y]--]=baze[i];
 for(i=1;i<=l;i++)
 baze[i]=rsort[i];
}
void fill()
{int X,Y,i,ps,pf;
//bordeaza matricea inainte de a aplica fill
for(i=0;i<=M+1;i++)
```

```
{mark(0,i);mark(N+1,i);}
for(i=0;i<=N+1;i++)
{mark(i,0);mark(i,M+1);}
ps=0; pf=1;
stiva[pf]. id=0;
stiva[pf]. x=Xb;
stiva[pf]. y=Yb;
mark(Xb,Yb);
for(;ps!=pf;)
{
next(ps);
X=stiva[ps]. x;
Y=stiva[ps]. y;
 if (stare(X-1,Y)==0)
 {next(pf);stiva[pf]. id=stiva[ps]. id+1;stiva[pf]. x=X-1;stiva[pf]. y=Y;
      if (estebaza(X-1,Y)) afiseaza(X-1,Y,stiva[pf]. id);
      mark(X-1,Y);
     }
 if (stare(X+1,Y)==0)
{next(pf);stiva[pf]. id=stiva[ps]. id+1;stiva[pf]. x=X+1;stiva[pf]. y=Y;
      if (estebaza(X+1,Y)) afiseaza(X+1,Y,stiva[pf]. id);
      mark(X+1,Y);
     }
 if (stare(X,Y-1)==0)
 {next(pf);stiva[pf]. id=stiva[ps]. id+1;stiva[pf]. x=X;stiva[pf]. y=Y-1;
      if (estebaza(X,Y-1)) afiseaza(X,Y-1,stiva[pf]. id);
      mark(X,Y-1);
     }
if (stare(X,Y+1)==0)
{next(pf);stiva[pf]. id=stiva[ps]. id+1;stiva[pf]. x=X;stiva[pf]. y=Y+1;
      if (estebaza(X,Y+1)) afiseaza(X,Y+1,stiva[pf]. id);
      mark(X,Y+1);
     }
}
}
void citeste()
{int i,x,y;
 in=fopen("radar.in ","r");
 out=fopen("radar.out","w");
 fscanf(in,"%d %d %d",&N,&M,&K);
 for(i=1;i<=K;i++)
```



```
 {fscanf(in,"%d %d",&x,&y); mark(x,y);}
 fscanf(in,"%d",&I);
 T=I;
 for(i=1;i<=I;i++)
 {fscanf(in,"%d %d",&x,&y); insertbase(x,y,i);}
 fscanf(in,"%d %d",&Xb,&Yb);
 fclose(in);
}
int main()
{int i,j;
 citeste();
 radixsortY();
 radixsortX();
 fill();
 if (T)
   fprintf(out,"Mission Incomplete! Targets Remaining: %d\n",T);
 fclose(out);
 return 0;
}
```

PERECHI

▸▸▸ENUNȚ:

Se consideră o pereche de numere naturale **a** şi **b**. Asupra perechii pot fi efectuate următoarele transformări:

- (**a**, **b**) devine (**a - b**, **b**);
- (**a**, **b**) devine (**a + b**, **b**);
- (**a**, **b**) devine (**b**, **a**).

Cerință:

Să se determine numărul minim de transformări succesive care duc la apariția perechii (**c**, **d**).

Date de intrare:

Fişierul de intrare **pairs.in** conține două linii. Pe prima linie se află numerele **a** şi **b** care formează perechea dată.

Pe a doua linie se află numerele **c** şi **d** care formează perechea care trebuie obținută. Numerele de pe o linie sunt separate prin spații.

Date de ieşire:

Fişierul de ieşire **pairs.out** trebuie să conţină o singură linie pe care se va afla numărul minim de transformări succesive necesare. În cazul în care perechea (**c**, **d**) nu poate fi obţinută, valoarea scrisă trebuie să fie **-1**.

Restricţii şi precizări:

$0 \le a, b, c, d \le 1000$

Timp de execuţie: 0.1 secunde/test.

Memoria disponibilă în timpul rulării: 0.5 MB de RAM.

EXEMPLU	**PAIRS.IN**	**PAIRS.OUT**
	2 2	2
	2 4	

Problemă propusă la concursul Bursele Agora 2003-2004

»»»SOLUŢIE:

Pentru început să considerăm matricea N*M ca fiind un graf neorientat. Fiecare nod are doi indici (x,y) deşi putem să-l referim şi cu o singură valoare (x*MAXy + y). Vom considera muchii cu cost de o unitate cele 3 operaţii de care dispunem. Astfel din nodul (x,y) vom avea muchii către nodurile (y,x), (x+y,y), (x-y,y). Modelând astfel problema, enunţul ne cere să determinăm un drum minim într-un graf neorientat având nodurile sursă şi destinaţie. Pentru determinarea lungimii minime vom folosi un algoritm de parcurgere în lăţime de complexitate N*M. Evident pentru a ne încadra în memoria disponibilă vom folosi o matrice de biţi.

»»»PROGRAM:

```
//aloca 425Kb de memorie din Heap cei 75KB rămasi pot fi folositi la
suplimentarea stivei cu 25000 de elemente
#include <stdio.h>
#define MAXN 1001
#define stivaSize 100000   //+25000
const int bit[8]={1,2,4,8,16,32,64,128}; //masca de biti
long ps,pf,sl,sl2;
unsigned char  mat[MAXN][MAXN/8+1];
unsigned short int stivax[stivaSize];
unsigned char     stivay[stivaSize];
//sfarsit DATE
```



```
FILE *in,*out;
int SOL,PAS,a,b,c,d;
void initdata()
{
in=fopen("pairs.in ","r");
fscanf(in,"%d %d %d %d",&a,&b,&c,&d);
fclose(in);
}
void select(int x,int y)
{
 mat[x][y >> 3]  |=  bit[ y & 7];
}
int isSelect(int x,int y)
{
  return mat[x][ y >> 3]  &  bit[y & 7];
}
void ieşire()
{
out=fopen("pairs.out","w");
if (SOL)
fprintf(out,"%d\n",PAS);
else fprintf(out,"-1\n");
fclose(out);
}
int next(int x)
{
if (x+1==stivaSize) return 0;
else return x+1;
}
void insereaza(int x,int y)
{
pf=next(pf);
sl2++;
stivax[pf]=(unsigned int) (x<<2)+ (unsigned int) (y&3);
stivay[pf]= (unsigned char) (y>>2);
}
int main()
{int pas,go=1,A,B;
initdata();
pf=ps=SOL=0; sl=1;
select(c,d);
```

```
insereaza(c,d);
for(pas=1;sl && go;pas++,sl=sl2)
for(sl2=0;sl && go;sl--)
{
   ps=next(ps);      // 3B pentru un element
   A=stivax[ps]>>2;  //primii 14 biti sunt pt. coordonata X
   B=(int(stivay[ps])<<2)+(stivax[ps]&3);//ultimi 2 + 8 biti sunt pt. Y
   if (A+B<=1000)
    if (!isSelect(A+B,B))
   {select(A+B,B);
    if (A+B==a && B==b) {PAS=pas;go=0;SOL=1;}
    else insereaza(A+B,B);
    }
   if (A-B>0)
    if (!isSelect(A-B,B))
   {select(A-B,B);
    if (A-B==a && B==b) {PAS=pas;go=0;SOL=1;}
    else insereaza(A-B,B);
    }
    if (!isSelect(B,A))
   {select(B,A);
    if (B==a && A==b) {PAS=pas;go=0;SOL=1;}
    else insereaza(B,A);
    }
}
ieşire();
return 0;
}
```

CFR

»»»ENUNŢ:

Ca urmare a inundaţiilor din ultima perioadă, o serie de linii ale căilor ferate au fost distruse. Regia autonomă a căilor ferate doreşte să identifice zonele în cadrul cărora este posibilă circulaţia în ambele sensuri. Zonele trebuie să fie maximale ca număr de oraşe cu staţii CFR.

Date de intrare:

Fişierul de intrare **cfr.in** conţine pe prima linie numărul natural N, reprezentând numărul de oraşe şi numărul natural M, reprezentând numărul de legături. Pe următoarele m linii sunt trecute legăturile dintre oraşe respectând sensul de mers.

Date de ieşire:

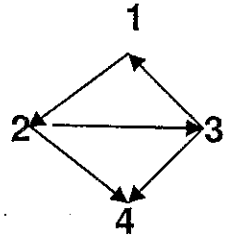
În fişierul de ieşire **cfr.out**, vor fi scrise pe câte o linie oraşele din zonele neafectate.

EXEMPLU	CFR.IN	CFR.OUT
	4 5	1 2 3
	1 2	4
	2 3	
	2 4	
	3 1	
	3 4	

»»»SOLUŢIE:

Pentru un nod oarecare se va face mulţimea predecesorilor şi mulţimea succesorilor în două şiruri de biţi de lungime n.

Pentru datele de intrare precizate ilustrăm aceste două şiruri de biţi în exemplul următor:



Pentru nodul 1 avem următorii vectori de biţi:

Succesori	1	1	1	1
Predecesori	1	1	1	0

Aceste două şiruri se intersectează(&) iar şirul rezultat ne va da prima componentă tare conexă: 1, 2, 3.

Se va căuta primul nod neselectat şi reluăm algoritmul.

Pentru nodul 4 avem următorii vectori de biţi:

Succesori	0	0	0	1
Predecesori	1	1	1	1

Aceste două şiruri se intersectează(&) iar şirul rezultat ne va da a doua componentă tare conexă: 4.

»»»PROGRAM:

```
//O(n^2)
#include <fstream.h>
#include <mem.h>
#define maxn 100
#define MAXN  maxn/32
unsigned long int a[maxn][maxn], viz[MAXN], pred[MAXN];
 unsigned long int succ[MAXN], comb[MAXN],n,m,x,y,k;
ifstream f("cfr.in");
ofstream g("cfr.out");
int are_predecesor(long int i)
{
  if ((pred[i>>5 +1])&(1<<i%32) )
   return 1;else return 0;
}
void setez_predecesor(long int i)
{
 pred[i>>5+1]=pred[i>>5+1]|(1<<i%32);
}
void setez_succesor(long int i)
{
 succ[i>>5+1]=succ[i>>5+1]|(1<<i%32);
}
int e_vizitat(long int i)
{
if ((viz[i>>5 +1])&(1<<i%32) )
  return 1;else return 0;
}
 int e_vizitat_comb(long int i)
{
if ((comb[i>>5 +1])&(1<<i%32) )
  return 1;else return 0;
}
```



```
void combinare()
{
 for (int j=1;j<=n;j++)
      comb[(j>>5+1)&(1<<j%32)]|=pred[(j>>5+1)&(1<<j%32)];
}
void setez_vizitat(long int i)
{
 viz[i>>5+1]=viz[i>>5+1]|(1<<i%32);
}
void citire()
{
  f>>n>>m;
 for(int i=1;i<=m;i++)
  {
      f>>x>>y;   a[x][y]=1;
   }
 f.close();
}
 void afisare(unsigned long int viz[MAXN])
 {
for(int i=1;i<=n;i++)
    if ((viz[i>>5 +1])&(1<<i%32) )
    g<<i<<' ';
    g<<endl;
 }
void predecesori(long int nod)
{
 setez_predecesor(nod);
 setez_vizitat(nod);
 for (int i=1;i<=n;i++)
   if (a[i][nod]&&!e_vizitat(i))
   predecesori(i);
}
void succesori(long int nod)
{
 setez_succesor(nod);
   setez_vizitat(nod);
   for (int i=1;i<=n;i++)
     if (a[nod][i]&&!e_vizitat(i))
    succesori(i);
 }
```

```
int main()
{
citire();
for (int i=1;i<=n;i++)
 if (!e_vizitat_comb(i))
    {
     memset(viz,0,sizeof(viz));
     memset(pred,0,sizeof(viz));
     predecesori(i);
     memset(succ,0,sizeof(viz));
     memset(viz,0,sizeof(viz));
     succesori(i);
     for (int j=1;j<=n;j++)
       pred[(j>>5+1)&(1<<j%32)]&=succ[(j>>5+1)&(1<<j%32)];
       afisare(pred);
       combinare();
     }
g.close();
return 0;
}
```

EXCURSIE

⋙ENUNȚ:

Elevii clasei a X-a B informatică vor să plece în excursie la Vatra Dornei. Organizatorul, cunoscând prietenii fiecărui elev, își notează în agendă un număr minim de telefoane ale unor elevi din grup, știind că anunțându-i pe aceștia tot grupul este anunțat. Se știe că dacă un elev x comunică imediat anunțul elevului z, nu este obligatoriu ca elevul z să-l anunțe pe elevul x.

Cerință:

Determinați dacă organizatorului îi este suficient notarea telefonului șefului clasei, iar dacă nu, determinați un algoritm eficient de identificare a grupurilor de elevi astfel încât dacă anunță pe oricare elev din grupul respectiv, toți elevii din grup să fie anunțați. Numărul grupurilor trebuie să fie minim.

Date de intrare:

Fișierul de intrare **excursie.in** conține pe prima linie numărul natural N reprezentând numărul de elevi și numărul natural M reprezentând numărul de telefoane ce vor fi date.

Pe următoarele m linii sunt trecute convorbirile de forma x y cu semnificația x îl telefonează pe y.

Date de ieșire:

În fișierul de ieșire **excursie.out** vor fi scrise pe câte o linie elevii dintr-un grup separați prin spațiu.

În cazul în care șeful clasei îi poate anunța pe toți elevii din clasă, se vor afișa toți elevii pe prima linie separați prin spațiu și pe a doua linie va fi scris mesajul "Posibil".

EXEMPLU	EXCURSIE.IN	EXCURSIE.OUT
	4 5 1 2 2 3 2 4 3 1 3 4	1 3 2 4

⋙SOLUȚIE:

Reprezentăm convorbirile ca arce și elevii ca noduri într-un graf orientat. Soluția aleasă determină optim componentele tare-conexe.

Mai întâi se determină în funcția **dft()** timpii finali (timpul l-a care un nod a fost prelucrat în timpul parcurgerii).

Pornind de la vârful care are ultimul timp final vom efectua o parcurgere în adâncime pe graful transpus prin funcția **dfe()**.

Vom relua algoritmul cu nodul cu cel mai mare timp final pentru care nu a fost determinată componenta tare conexă. Acest algoritm are complexitatea O(2*m). Pentru a reduce memoria folosită de la $O(N^2)$ la O(M) reținem graful nu ca pe o matrice de adiacență ci ca pe o matrice de pointeri în care pentru fiecare nod x dat reținem lista nodurilor adiacente din graful dat în a[x][0] iar lista nodurilor adiacente din graful transpus în a[x][1].

Vectorul **viz** în care sunt bifate nodurile parcurse este gestionat ca un vector de biți.

Obs. Spre deosebire de aloritmul prezentat la problema CFR, în afișarea soluțiilor orice nod va fi parcurs înaintea succesorilor săi.

»»»PROGRAM:

```
//O(2xm)
#include <fstream.h>
#include <mem.h>
#define maxn 1000
#define MAXN  maxn/32
struct elem{unsigned long int inf; elem* urm;};
unsigned long int viz[MAXN],n,m,x,y,k,c[maxn],nr;
elem *a[maxn][2],*p;
ifstream f("excursie.in");
ofstream g("excursie.out");
// i>>5 ==i/32,   i&31== i%32
//i>>5+1==a cata grupa de 32
//1<<i&31==in cadrul grupei al catelea bit
int e_vizitat(long int i)
{
 if ((viz[i>>5 +1])&(1<<i%32) )
   return 1;else return 0;
}
void setez_vizitat(long int i)
{
 viz[i>>5+1]=viz[i>>5+1]|(1<<i%32);
}
void citire()
{
 f>>n>>m;
 for(int i=1;i<=n;i++)a[i][0]=a[i][1]=NULL;
 for( i=1;i<=m;i++)
   {f>>x>>y;
    p=new elem;
    p->inf=y;
    p->urm=a[x][0];
    a[x][0]=p;
    p=new elem;
    p->inf=x;
    p->urm=a[y][1];
    a[y][1]=p;
     }
}
```



```
void dft(long int nod)
{
  setez_vizitat(nod);
 for (elem *p=a[nod][0];p;p=p->urm)
    if (!e_vizitat(p->inf))
    dft(p->inf);
  k++; c[k]=nod;
}
void dfe(long int nod)
{ g<<nod<<' ';
  setez_vizitat(nod);
  for (elem *p=a[nod][1];p;p=p->urm)
    if (!e_vizitat(p->inf))
     dfe(p->inf);
 }
int main()
{
citire();
memset(viz,0,sizeof(viz));
for (int i=1;i<=n;i++)
  if (!e_vizitat(i))
      dft(i);
   memset(viz,0,sizeof(viz));
for ( i=n;i>=1;i--)
  if (!e_vizitat(c[i]))
   {dfe(c[i]);nr++;
    g<<endl;}
if(nr==1) g<<"Posibil";
return 0;
}
```

XOR

»»»ENUNȚ:

Se consideră secvența $A_1, A_2, ..., A_N$ de întregi pozitivi.

Cerință:
Determinați un subșir $A_{i1}, A_{i2}, ..., A_{ik}$ ($1 \le i1 < i2 < ... < ik \le N$), astfel

încât A_{i1} XOR A_{i2} XOR ... XOR A_{ik} să aibă valoarea maximă posibilă.

Date de intrare:

Pe prima linie a fişierului **xor.in** se află N reprezentând numărul de elemente din secvenţă.

Pe a doua linie se află cele N valori A1, A2, ..., AN.

Date de ieşire:

Pe singura linie a fişierului de ieşire **xor.out** se află valoarea maximă posibilă ce se poate obţine aplicând XOR pe subşirul ales.

Exemplu:

xor.in	xor.out
3 11 9 5	14

Restricţii:

$0 < N \leq 10000$

$0 \leq Ai \leq 10^{18}$

»»»SOLUŢIE:

Problema nu este atât de grea precum pare. În cadrul soluţiei vom lucra cu valori de tip unsigned long long. Vom considera reprezentările binare ale celor N valori drept o matrice cu 64 de linii şi N coloane.

Definim şirul X1,X2,..XN cu valori 0 sau 1 având următoarea semnificaţie: Xi ia valoarea 1 dacă elementul i din vector face parte din subşirul de valoare maximă şi 0 în caz contrar. Notăm cu VMAX valoarea maximă ce se poate obţine şi presupunem că aceasta este egală 2^{64} -1.

În matricea construită mai adăugăm încă o coloană cu reprezentarea binară a lui VMAX. Se observă uşor că avem de rezolvat un sistem de 64 de ecuaţii liniare cu N necunoscute X1,X2...XN. Elementele de pe coloana N+1 reprezintă valorile ce urmează după semnul egal din fiecare ecuaţie. Astfel problema se reduce la a rezolva acest sistem de ecuaţii.

Pentru fiecare linie i pornind de la 63 (bitul cel mai semnificativ) vom alege primul termen nenul şi îl vom înlocui în celelate i ecuaţii. Deoarece pentru toate valorile ce intervin în calcule nu ne interesează decât paritatea acestora, vom lucra totul modulo 2.

De aici se observă uşor că înlocuirea unui termen Xk din ecuaţia i în ecuaţia j este echivalentă cu aplicarea operaţiei XOR pe cele 2 linii (linia j=linia j XOR linia i).

Un caz special ce trebuie tratat este cel în care întâlnim o ecuaţie j care are toţi termenii nuli cu excepţia valorii de după semnul egal. În acest caz, pentru a egala ecuaţia, vom schimba în 0 bitul j din reprezentarea binară a lui VMAX: Fiecare valoare din matrice ocupă un bit de memorie.

Pentru eficienţă o linie o vom reprezenta prin N+1/32 valori de tip unsigned int. În felul acesta, vom putea aplica operaţia XOR pe mai multe elemente în acelaşi timp şi vom reduce timpul de execuţie al programului.

Complexitatea finală este $O(64^2 * N/32 + 64 * N)$.

»»»PROGRAM:

```
//O (64^2 * N /32)
#include<stdio.h>
#define MAXN 5001
#define MAXBIT 64
#define MODULO 666777
int xs[MAXN],N,mark[MAXN];
long long VMAX,V[MAXN],bit[MAXBIT];
unsigned int REZ,NRSOL,ec[MAXBIT][MAXN/32+1];
void citesteDate()
{
  int i;
  freopen("xor.in","r",stdin);
  freopen("xor.out","w",stdout);
  scanf("%d",&N);
  for(i=1;i<=N;i++)
          scanf("%lld",&V[i]);
}
void proceseaza()
{
  int find,ok,t,i,j,k,c,prim,lim=(N+1)>>5;
  for(bit[0]=1,i=1;i<MAXBIT;i++) bit[i]=bit[i-1]<<1;
          for(i=0;i<MAXBIT;i++)
          for(j=1;j<=N;j++)
                  if (V[j]&bit[i])
                  ec[i][j>>5] |= bit[j&31];
```

```
    for(i=0;i<MAXBIT;i++)
            {ec[i][(N+1)>>5] |= bit[(N+1)&31];VMAX |= bit[i];}
    for(i=1;i<=N;i++) mark[i]=1;
    for(i=MAXBIT-1;i>=0;i--)
    {
        for(c=0,j=1;j<=N;j++)
          if (ec[i][j>>5]&bit[j&31]) {c++;xs[c]=j;}
          if (c==0) {if (ec[i][(N+1)>>5] & bit[(N+1)&31]) VMAX^=bit[i];}
              else
              {for(find=0,k=1;k<=c && !find;k++)
                {for(ok=0,j=i-1;j>=0 && !ok;j--)
                                if (ec[j][xs[k]>>5]&bit[xs[k]&31])
                                    ok=1;
                  if (ok) find=xs[k];
                }
               if (find)
               {
                mark[find]=0;
                for(j=i-1;j>=0;j--)
                        if (ec[j][find>>5]&bit[find&31])
                          for(k=0;k<=lim;k++) ec[j][k]^=ec[i][k];
               }
               else mark[xs[1]]=0;
              }
    }
    for(i=1;i<=N;i++)  if (mark[i]) NRSOL++;
    REZ=1;
    for(i=1;i<=NRSOL;i++)
    REZ=(REZ<<1)%MODULO;
}

void afiseazaRez()
{           printf("%lld\n%d\n",VMAX,REZ);   }

int main()
{
   citesteDate();
    proceseaza();
    afiseazaRez();
    return 0;
}
```


SPIONI

»»»ENUNȚ:

Mai multe agenții de spionaj, maxim m, își desfășoară activitatea în mai multe țări. În fiecare agenție sunt maxim n spioni. Anumite informații au caracter mai puțin cofidențial, în sensul că o știre de acest gen odată adusă în agenție toți agenții din acea agenție au acces la acestă informație. Presupunând că un spion dublu ar putea să transmită o știre unei alte agenții verificați dacă știrea adusă de agentul x va ajunge la agentul y din altă agenție.

Cerință:

Calculați numărul minim de agenții prin care trebuie să treacă știrea pentru ca, în final, aceasta să ajungă la agentul y.

Date de intrare:

Pe prima linie se află **n, m, pi, pf** ce reprezintă numărul de agenții, numărul de agenții, agentul de la care pleacă și agentul la care ajunge știrea.

Pe următoarele m linii sunt trecuți agenții celor m agenții, separați prin spațiu.

Date de ieșire:

Pe prima linie va fi numărul minim de agenții.

EXEMPLU:

tel.in	**tel.out**
11 4 1 10 1 3 6 5 3 6 7 11 7 9 9 10	3

Restricții:

1≤m≤n≤256

»»»SOLUȚIE:

Se va considera graful neorientat care are ca noduri agențiile. Între două agenții există muchie dacă au un agent sau mai mulți în

comun. Se va identifica agenția în care lucrează agentul **pi** şi printr-o parcurgere în lățime se va afla drumul minim până la agenția în care lucrează agentul pf.

Cum agentul **pi** poate fi agent dublu se va efectua pargurgerea în lățime din toate agențiile în care figurează agentul pi şi se va alege cel mai mic lanț dintre toate cele rezultate.

Pentru optimizare, matricea de adiacență va fi o matrice de biți. Astfel pentru a verifica dacă două agenții au agenți comuni revine la a verifica dacă întersecția lor (& pe biți) este nevidă. Complexitatea algoritmului este O(n*m).

⋙PROGRAM:

```
//x>>3 inlocuieste x/8- numarul grupei
//x&7 inlocuieste x%8- numarul pozitiei in grupa
#include <fstream.h>
#include <string.h>
#include <stdlib.h>
#define maxn 256
#define MAXN  (maxn/8)
int mat[maxn][MAXN];
int viz[MAXN],rez[MAXN],nr[MAXN],n,m,x,y,pi,pf; char b[100];
ifstream f("tel.in");
ofstream g("tel.out");
void citire()
{  char *p;
 f>>n>>m>>pi>>pf;
 f.get();
 for(int j=0;j<m;j++)
  {
   f.getline(b,100,'\n');
   p=strtok(b," ");
   while(p)
    { x=atoi(p);
      mat[j][(x>>3)]|=(1<<(x&7));
      //in grupul j setez in grupa de biti corespunzatoare lui x cu 1 locul lui x
     p=strtok(NULL," ");
    }
  }
 f.close();
}
```



```
int intersectie(int x, int y)
{
 for(int i=0;i<MAXN;i++)
  if(mat[x][i]&mat[y][i]) return 1;
   return 0;
}
int parcurgere(int grup, int pf)
{    int nr[MAXN]; int min=maxn;  int p,u;  p=u=1;
 memset(viz,0,sizeof(viz));  memset(nr,0,sizeof(viz));
  viz[grup]=1;
  rez[u]=grup;
  nr[grup]=0;
  while(p<=u)
  { grup=rez[p++];
    for(int i=0;i<m;i++)
      if ( intersectie(grup,i) && !viz[i])
       { u++;    viz[i]=1;     rez[u]=i;    nr[i]=nr[grup]+1;    }
   }
for(int i=1;i<=m;i++)
 if (mat[i][pf>>3] & 1<<(pf&7))
     if(min>nr[i]) min=nr[i];
return min;
}
int main()
 {
  int min,MIN=maxn;
  citire();
  for(int  j=0;j<m;j++)
   {
    if(mat[j][pi>>3]&(1<<(pi&7)))
      {
   min=parcurgere(j,pf);
   if (MIN>min) MIN=min;
      }
   }
  if (MIN==maxn)
     g<<"imposibil";
    else
     g<<min;
  g.close();
 return 0;}
```

PROBLEME PROPUSE

PB1. Se dau două şiruri cu n şi m elemente. Să se verifice dacă şirurile reprezintă mulţimi. Dacă nu sunt mulţimi, să se elimine elementele care se repetă şi apoi să se efectueze reuniunea, intersecţia şi diferenţa dintre cele două mulţimi. Definiţi submulţimile ca vectori de biţi.

EX: n=3, m=4 a=5,7,6 b=1,1,5,2 ; **R:** a={5,6,7}; b={1,2,5}; a&b={5}, a|b={1,2,5,6,7}; a-b={6,7} ; b-a={1,2}.

PB2. Se dau m submulţimi ale unei mulţimi cu k elemente. Să se numere şi să se afişeze elementele care se găsesc în exact n submulţimi din cele m date. Definiţi submulţimile ca vectori de biţi.

EX: m=3; n=2 ; k=5; m_1={1,2,5}; m_2={1,4,5};m_3={1,4};

R: nr=2 ;{4,5}

PB3. Fie s un număr dat. Să se sfişeze toate posibilităţile de a scrie acest număr ca sumă de k termeni distincţi (Vezi generarea submulţimilor).

EX: s=8; k=3

R: 1+2+5;1+3+4;

PB4. De Crăciun, mama doreşte să împartă cadourile celor doi fraţi astfel încât amândoi să fie mulţumiţi. Pe pachete sunt inscripţionate totalul preţurilor obiectelor conţinute: ciocolate= 40 lei, caiete= 50 lei, un trenuleţ= 100 lei, etc. Aflaţi dacă mama poate împărţi în mod egal ca valoare cadourile şi dacă nu, ajutaţi-o să obţină o diferenţă cât mai mică.

Indicaţie: Puteţi bifa într-un vector de biţi sumele numerelor obiectelor din pachete, asemănător soluţiei indicate la problema "Submulţimi de sumă dată". Se calculează suma totală a obiectelor şi se va calcula diferenţa dintre aceasta şi cea mai apropiată sumă de valoarea medie a sumei.

EX: n=3, 40,50,100

R: 90 şi 100

PB5. Fie n un număr natural scris în baza 10 şi b ($2 \le b \le 256$) o bază de numeraţie. Să se scrie un program care afişează în baza 10 toate numerele mai mici sau egale cu n, care scrise în baza b folosesc numai cifrele 0 şi 1. Numerele vor fi scrise în ordine strict crescătoare.

EX: n=11,b=3

R: 0,1,3,4,9,10

Indicaţie:Deşi această problemă a fost rezolvată în cadrul secţiunii „Deplasări pe biţi" reluaţi rezolvarea recurgând la o soluţie de

bifare într-un vector de biţi a puterilor bazei date, asemănător soluţiei indicate la problema "Submulţimi de sumă dată".

PB6. Fie a şi b două numere naturale. Să se efectueze a^b într-un număr cât mai mic de înmulţiri.(vezi problema A^Bmod C sau înmulţirea a la russe)

PB7. Într-un oraş există n pieţe, identificate prin numere.cuprinse între 1 şi n. Se ştie că din fiecare piaţă se ajunge în oricare piaţă folosind străzile existente în număr total m. Primăria a stabilit un sens unic de circulaţie. Stabiliţi dacă folosind sensul unic de crculaţie se poate ajunge din fiecare piaţă în oricare piaţă şi în caz contrar să se precizeze două pieţe x,y cu proprietatea că pornind din x nu se poate ajunge în y sau pornind din y nu se poate ajunge în x.

Indicaţie: Vezi problema EXCURSIE.

PB8. Să se reia problema CIRCUITE pentru N≤1000.

PB9. Să se determine cel mai mic număr prim natural care are exact K divizori.

Indicaţie: Cum numărul căutat este 2^{N+1} vezi problema DOILAN.

PB10. Să se scrie o funcţie care primeşte ca argumente două numere în formă binară (128 de biţi) şi returnează suma lor.

Observaţie: Se va lua în considerare dacă există sau nu depăşire.

PB11. Să se scrie o funcţie care primeşte ca argumente două numere în formă binară (128 de biţi) şi returnează diferenţa lor.

Indicaţie: Diferenţa este o sumă în care al doilea număr este transformat ~op2+1

PB12. Să se scrie o funcţie care primeşte ca argumente două numere în formă binară (128 de biţi) şi le compară returnând -1 dacă primul număr e mai mic decât al doilea, 0 dacă numerele sunt eagale şi 1 dacă primul număr este mai mare decât al doilea.

PB13. Să se scrie o funcţie care primeşte ca argumente două numere în formă binară (128 de biţi) şi returnează înmulţirea lor modulo p.

PB14. Să se scrie o funcţie care primeşte ca argumente două numere în formă binară (128 de biţi) şi returnează ridicarea la putere a lor modulo p.

PB15. Conform teoremei lui Euler-Fermat p este număr prim dacă pentru orice număr întreg n $\in$[1,p) este verificată formula $n^{p-1}\%p=1$. (vezi problema A^Bmod C)

Indicaţie: Acest algoritm este util pentru numere foarte mari (pe 128, 256, 512 biţi) caz în care se vor testa în jur de 100 de numere din intervalul [1,p) şi se vor folosi funcţiile realizate la problemele PB5-PB9.

PB16. Să se scrie o funcție care primește ca argumente două numere în formă binară (128 de biți) și returnează cel mai mare divizor comun al lor.

Indicație: Se va folosi rezultatul de la **PB8.**

PB17. Fie n produse de câte două numere prime. Să se afle numărul care apare în cele mai multe produse. Produsele sunt numere pe128 de biți.

Indicație: Se va folosi rezultatul de la **PB11.**

CAPITOLUL 2

HEAP-uri

Considerente teoretice

Un arbore binar se numește ***heap*** dacă are definită o relație de ordine pe informația conținută în noduri iar arborele este complet, excepție facând ultimul nivel al arborelui, care se.completează începând cu nodul cel mai din stânga. În funcție de modul în care informația este organizată în heap acesta poate fi:

- MaxHeap când pentru fiecare nod sunt îndeplinite simultan condițiile: valoare nod ≥ valoare nod subarbore stâng și valoare nod ≥ valoare nod subarbore drept;
- MinHeap când pentru fiecare nod sunt îndeplinite simultan condițiile: valoare nod ≤ valoare nod subarbore stâng și valoare nod ≤ valoare nod subarbore drept;
- Min-maxHeap când nivelurile sunt alternative, nivelurile cu număr par fiind niveluri minime, iar nivelurile cu număr impar fiind niveluri maxime. Rădăcina furnizează valoarea minimă iar fiul cu cea mai mare valoare furnizează valoarea maximă;
- Max-minHeap când nivelurile sunt alternative, nivelurile cu număr par fiind niveluri maxime, iar nivelurile cu număr impar fiind niveluri minime. Rădăcina furnizează valoarea maximă iar fiul cu cea mai mică valoare furnizează valoarea minimă;
- Deap când subarborele stâng este un minHeap și subarborele drept este un maxHeap. Rădăcina nu conține nici o informație;

- Heapuri de intervale când fiecare nod conține două valori x,y ce reprezintă un interval, x<y , iar intervalul fiecărui nod trebuie să includă intervalele corespunzătoare fiilor nodului respectiv;
- Heapuri binomiale B_n când există două heapuri B_{n-1} astfel încât rădăcina unuia este cel mai din stânga fiu al rădăcinii celuilalt, n>0. Pe fiecare nivel i sunt C^i_n noduri;
- Deque-uri este un caz particular de heap pe care îl aprofundăm în această culegere.

Operațiile care pot fi făcute în general pe heap-uri sunt:

- Inserarea unui element
- Extragerea unui element
- Crearea unui heap dintr-un vector oarecare
- Determinarea elementului maxim, minim sau a ambelor valori
- Sortarea elementelor
- Căutarea unei valori

Heap-urile au aplicații cu precădere în stabilirea priorităților. În această culegere ne vom ocupa de Maxheap-uri, Minheap-uri și de o structură particulară de heap numită Deque.

Maxheap-uri

Deoarece un heap este un arbore binar complet el se reprezintă eficient folosind un vector în care:

- fiecare element al vectorului este un nod al arborelui;
- orice nod a[i] are ca și fiu stâng nodul a[2*i] și ca fiu drept nodul a[2*i+1];
- orice nod a[i] are ca părinte nodul a[i/2], exceptând rădăcina care nu are tată;
- rădăcina heapului este elementul a[1].

Vom prezenta în continuare implementarea operațiilor specifice unui maxHeap, programele putând fi ușor transformate în vederea obținerii unui minHeap.

Scufundarea unui nod

Când un nod a[i] are o valoare mai mică decât fii săi trebuie să reconstruim heap-ul prin coborârea nodului până când acesta nu mai are fii, sau fii săi au valori mai mici decât valoarea lui.

Procedura *scufunda* are rolul de a reaşeza în heap valoarea a[i] astfel încât noul subarbore cu vârful în i să devină heap.

```
Void scufunda (int a[], int n, int i)
{  int left, right, max, temp;
  left = 2*i;
  right = 2*i + 1;
  if ((left<=n) && (a[left]>a[i]))  max = left;   else max = i;
   if ((right<=n) && (a[right]>a[max]))  max = right;
    if (max!=i)
        {
          temp = a[i];
          a[i] = a[max];
          a[max] = temp;
          scufunda (a, n, max);}
}
```

Dacă această implementare este mai uşor de citit vom da în continuare o implementare cu o viteză mai mare de execuţie. Complexitatea acestui program este O(logn).

```
void scufunda(int a[], int n, int i)
{
   int fiu,temp;
   while ((i<<1)<=n)
    {
      fiu=i<<1;
      if ( (i<<1) |1<=n && a[(i<<1)|1]>a[i<<1])  fiu|=1;
      if (a[fiu]>a[i])
       {
        temp = a[i];
        a[i] = a[fiu];
        a[fiu] = temp;
         i=fiu;
       }
      else break;
    }
}
```


OBSERVAŢII:

i<<1	=fiul stâng
(i<<1)|1	=fiul drept
(i<<1)<= n	=există fiul stâng?
((i<<1)|1<= n	= există fiul drept?
i>>1	=părintele lui i

Ridicarea unui nod

Se aplică atunci când un fiu are valoarea mai mare decât a tatălui său. Este operaţia inversă scufundării. Complexitatea acestui program este O(logn).

```
void ridica(int i)
{
        int val=a[i];
        while(i>1 &&  val>a[i>>1])
         {
        a[i]=a[i>>1]; i>>=1;
        }
        a[i]=val;
}
```

Inserarea unui nod

Un nou nod se inserează pe poziţia n+1 şi îl ridicăm până la poziţia curentă. Complexitatea acestui program este O(logn).

```
void insereaza(int &n,int val)
{
   a[++n]=val;
   ridica(n);
}
```

Crearea unui heap pornind de la un vector

Această operaţie o putem realiza în mai multe moduri. Astfel putem crea un maxHeap inserând pe rând cele n noduri cu o complexitatea de O(n* logn).

```
void creareHeap1(int a[], int n )
{
  int i,m=0;
  for(i=2; i<=n; i++)
          insereaza(m,i-1);
}
```

Putem realiza un maxHeap şi urcând fiecare nod în heap. Complexitatea acestui program este O(n* logn).

```
void creareHeap2(int a[], int n )
{
  int i;
  for(i=1; i<=n;i++)
          ridica(i);
}
```

Putem realiza un maxHeap pornind de la observaţia că un element al vectorului care nu are descendenţi este întotdeauna un heap. Pentru un vector de lungime n, putem construi un heap cu elementele respective, dacă apelăm procedura *scufunda* pentru fiecare nod imediat superior frunzelor, apoi pentru nodurile de pe nivelurile superioare pâna la rădăcină, adică pentru elementele de pe poziţiile n/2, n/2-1...1. Complexitatea acestui program este O(n). În implementările următoare vom folosi această variantă.

```
void creareHeap(int a[], int n )
{
  int i;
  for(i=( n >>1); i>0; i--)
          scufunda(a, n, i);
}
```

Extragerea unui nod

De regulă, ne interesează nodul rădăcină care are valoarea cea mai mare(sau cea mai mică), dar funcţia prezentată sub numele *extrage* funcţionează pentru orice nod din arbore.

După ce extragem nodul dorit, pentru a păstra structura de heap nealterată vom plasa ultimul element în nodul care se doreşte a fi extras(eliminat) apoi refacem heap-ul astfel:

- Dacă valoarea nodului este mai mică decât cea a unui fiu atunci îl scufundăm
- Dacă valoarea nodului este mai mare decât a tatălui atunci îl ridicăm

```
void extrage (int &n,int i)
{
a[i]=a[n--];
if(i>1 && a[i]>a[i>>1])   ridica(n,i);
  else scufunda(a,n,i);
}
```

Complexitatea acestui program este O(logn). Heap-urile sunt eficiente în furnizarea unor valori maxime şi minime dar sunt mai puţin eficiente în realizarea unor căutări.

Heapsort

Pentru un şir dat, prin apelarea funcţiei *creareHeap* putem construi un heap cu elementele şirului. Acest heap are în vârful său elementul cel mai mare.

Ca să obţinem cea mai eficientă procedură de sortare care se bazează pe comparaţii vom repeta procedura de interschimbare a elementului din vârf cu ultimul element al şirului, apoi apelăm *scufunda* pentru vârful şirului pentru a reordona heap-ul. La fiecare interschimbare micşorăm dimensiunea heapului cu 1.

Aceasta este sortarea heapsort, care are o complexitate O(n*logn) pentru cazul cel mai defavorabil.

```
void heapsort(int a[], int n)
{
        int i, temp;
        creareHeap(a, n);
        for (i=n; i>0; i--) {
                temp = a[1]; a[1] = a[i]; a[i] = temp;
                n--;
                scufunda(a, n, 1);
        }
}
```

PROBLEME REZOLVATE

COMENZI

›››ENUNȚ:

Într-un magazin sunt n raioane. Ştiind că magazinul se aprovizionează săptămânal, ajutați-l pe patron să premieze primele k raioane cu cel mai bine vândut produs. Fiecare raion intră în concurs cu câte un sortiment. Un produs „s-a vândut bine" dacă numărul de bucăți vândute raportat la stocul inițial este cât mai aproape de 100%.

Date de intrare:

Pe prima linie din fişierul **comenzi.in** se găsesc numerele N şi K.

Pe următoarele n linii se găsesc separate prin spațiu câte trei numere reprezentând numărul raionului, stocul inițial şi stocul final.

Date de ieşire:

Pe k linii din fişierul **comenzi.out** se vor afişa pe câte o linie un număr reprezentând numărul de raion şi un procent reprezentând cât la sută s-a vândut din cel mai cerut produs din raionul respectiv.

Exemplu:

comenzi.in	comenzi.out
7 5	5 100%
1 100 25	7 100%
2 20 5	6 62%
3 44 10	2 25%
4 25 0	1 25%
5 1 1	
6 8 5	
7 10 10	

Restricții:

1≤k≤n≤100

›››SOLUȚIE:

Se va realiza un maxHeap după procentul de vânzare O(n). Se vor extrage k valori maxime O(n). Se reface maxHeap-ul după fiecare extragere O(n*logn).

›››PROGRAM:

```
#include <iostream.h>
#include <fstream.h>
struct com{int nr,stoci, stocf,ef;}a[100];
int n,k;
ifstream.f("comenzi.in");
ofstream g("comenzi.out");

void scufunda(com a[], int n, int i)
{
   int fiu;
   com temp;
   while ((i<<1)<=n)
     {  fiu=i<<1;
  if((i<<1)|1<n &&a[(i<<1)|1].ef>a[i<<1].ef)   fiu|=1;
  if (a[fiu].ef>a[i].ef)
   {
     temp = a[i];
     a[i] = a[fiu];
     a[fiu] = temp;
     i=fiu;
    }
    else break;
     }
}
void creareHeap(com a[], int n )
 {
  for(int i=(n>>1); i>0; i--)
       scufunda(a, n, i);
 }
int main()
{
   f>>n>>k;
   for(int i=1;i<=n;i++)
     {
```

```
    f>>a[i].nr>>a[i].stoci>>a[i].stocf;
    a[i].ef=a[i].stocf*100/a[i].stoci;
   }
  f.close();

 creareHeap(a,n);
  for(i=1;i<=k;i++)
  {
   g<<a[1].nr<<' '<<a[1].ef<<'%'<<endl;
   a[1]=a[n];n--;
   scufunda(a,n,1);
  }
 g.close();
return 0;
}
```

TRASEE

»»»ENUNȚ:

Bogdan s-a mutat în Bucureşti. Ştie adresele prietenilor săi şi mai are la îndemână harta Bucureştiului pe care şi-a notat kilometrajul unor trasee fără semafoare, dar care încep şi se termină în stații binecunoscute.

Cerință:

Ajutați-l pe Bogdan să-şi traseze pe hartă drumurile către prietenii săi pe trasee cu kilometraj cunoscut, astfel încât să plătească cât mai puțin la taxi.

Date de intrare:

Pe prima linie din fişierul **dj.in** se găsesc numerele N, M şi X reprezentând numărul de stații, numărul de trasee cu kilometraj trecut pe hartă şi stația de plecare.

Pe următoarele n linii se găsesc separate prin spațiu câte trei numere reprezentând intersecția inițială, intersecția finală şi kilometrajul corespunzător străzii.

Date de ieşire:

Pe fiecare linie din fişierul **dj.out** se vor afişa separate prin spațiu nodurile drumului de cost minim (stațiile iar în finalul liniei, între paranteze rotunde, costul drumului(kilometrii).

Exemplu:

dj.in	**dj.out**
5 9 4	4 1 (1)
1 2 1	4 1 2 (2)
1 3 9	4 3 (2)
1 5 3	4 (0)
2 3 7	4 1 5 (4)
2 4 3	
4 1 1	
4 3 2	
5 2 4	
5 4 2	

»»»SOLUȚIE:

O problemă veşnic tânără este problema drumului minim. Algoritmul lui Dijkstra poate fi folosit cu o complexitate de $O(n^2)$ în cazul în care nu avem costuri negative şi dorim aflarea costului minim al drumului dintre două noduri date sau dintre un nod şi celelalte noduri.

Dacă în cazul unui graf dens algoritmul Dijkstra este mai eficient, în cazul în care graful este rar este de preferat algoritmul Dijkstra optimizat cu minHeap numit şi *Dijkstra -modificat*.

Optimizarea constă în extragerea pe rând dintr-un minHeap a nodului cu drumul de cost parțial cel mai mic în vederea actualizării cu această valoare a tuturor drumurilor care trec prin acel nod.

Vă prezentăm un program care foloseşte *Dijkstra-modificat* în vederea afişării a tuturor drumurilor de cost minim pornind dintr-un nod dat. Programul are o complexitate O(max(m,n)*logn), iar dacă graful este conex are complexitatea O(m*logn).

Pentru a reduce memoria folosită de la $O(N^2)$ la O(M) putem reține graful nu ca pe o matrice de adiacență ci ca pe un vector de liste de vecini.

»»PROGRAM:

```
// O(E*logV)
#include <fstream.h>
#include <mem.h>
# define inf 30000
ifstream f("dj.in");
ofstream g("dj.out");
unsigned int d[100];//costurile drumurilor minime pt. fiecare nod
unsigned int q[100];// minHeap
unsigned int t[100];// parintele fiecarui nod in vederea reconstituirii drumului
unsigned int,n,k,m,i,j,x; int a[100][100];//matricea de adiacenta
void scufunda(unsigned int d[], int n, int i)
{
  int fiu,temp;
  while ((i<<1)<=n)
      {  fiu=i<<1;
   if(fiu+1<=n && d[q[fiu+1]]<d[q[fiu]]) fiu++;
   if (d[q[fiu]]<d[q[i]])
     {
       temp = q[i];
       q[i] = q[fiu];
       q[fiu] = temp;
       i=fiu;
      }
      else break;
       }
}
void ridica(int i)
{
   int val=q[i];
   while(i>1 &&  d[val]<d[q[i/2]])
     { q[i]=q[i>>1];i=i/2;}
   q[i]=val;
}
void afis(int i)
{
   if (i>0)
       {
       afis(t[i]); g<<i<<' ';
       }
}
```



```
int main()
{
 f>>n>>m>>x;
 for (int i=1;i<=n;i++)
  for (int j=1;j<=n;j++)
    a[i][j]=inf;
 for (k=1;k<=m;k++)
  {
  f>>i>>j; f>>a[i][j];
  }
 f.close();
 for (i=1;i<=n;i++)
    {
       q[i]=i; t[i]=0;  d[i]=inf;
    }
     k=n;
     d[x]=0;
     for (i=k/2;i>0;i--)      scufunda(d,n,i);
    while (k>0 && d[q[1]]<inf)
    {
     j=q[1];  cout<<j<<endl;  q[1]=q[k];
     k--;
     scufunda(d,k,1);
     for (i=1;i<=k;i++)
      if (d[q[i]]> d[j]+a[j][q[i]])
      {
         d[q[i]]=d[j]+a[j][q[i]];
         t[q[i]]=j;
         ridica(i);
      }
   }
    for (i=1;i<=n;i++)
    {
     if (d[i]<inf) {    afis(i);  g<<'('<<d[i]<<')'<<endl; }
           else g<<"Nu exista"<<endl;
    }
    g.close();
    return 0;
}
```

DEQUE (DOUBLE QUEUE)

Considerente teoretice

În numeroase probleme de informatică apare deseori nevoia de a utiliza o structură de date asupra căreia să se efectueze inserări şi extrageri de date, dar mai ales să se răspundă la anumite întrebări într-un timp cât mai scurt.

Spre exemplu, pentru o mulţime în care se inserează şi se extrag elemente, se doreşte aflarea valorii maxime sau minime după fiecare astfel de operaţie. Structura potrivită în acest caz este un Heap, în care operaţiile de inserare şi extragere se implementează în O(logN) iar operaţia de query în O(1), unde N este dimensiunea Heap-ului.

În următoarele aplicaţii ne vom concentra asupra unui caz particular de Heap, în care ordinea de extragere a elementelor coincide cu cea de inserare.

Cea mai simplă metodă prin care poate fi vizualizat acest caz particular este un vector de elemente(date) pe care se deplasează de la stânga spre dreapta un interval de interes. Pentru acest interval, toate cele trei operaţii de inserare, ştergere şi query se pot implementa în timp O(1) folosind o structură simplă şi foarte eficientă pe care o vom prezenta în cele ce urmează.

Să considerăm mai întâi un maxHeap (un Heap cu funcţia query=valoarea maximă din structură) şi să observăm ce avantaje ne aduce cazul particular. Atunci când inserăm un element nou, ştim că acesta va sta în structură mai mult timp decât celelalte deja prezente acolo (datorită ordinii de extragere LILO) şi prin urmare toate elementele din Heap cu valoare mai mică decât acesta nu vor mai putea influenţa niciodată răspunsul la query, deci se pot elimina.

Să considerăm toate elementele din Heap ordonate descrescător într-o listă asupra căreia avem acces la ambele capete.

Operaţia de inserare o vom implementa în cadrul funcţiei **insert** în felul următor: iniţial, lista conţine 0 elemente, deci ordinea descrescătoare este îndeplinită. La fiecare inserare, prin capătul drept vom elimina toate elementele mai mici decât cel curent, iar la sfârşit vom adăuga noul element. În acest mod, lista se păstrează mereu sortată, mai mult, poziţiile elementelor din structură apar în vector în ordine crescătoare.

Operaţia de extragere va fi implementată în cadrul funcţiei **query** astfel: se alege elementul din capătul stâng (cel cu valoarea maximă) şi

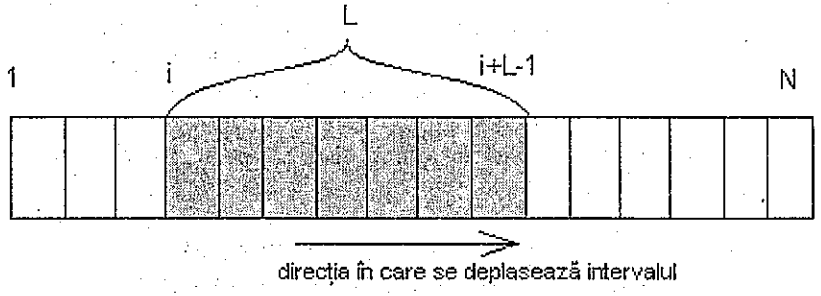
se verifică dacă poziţia acestuia în vector este în interiorul intervalului de interes, iar dacă nu, acesta este eliminat şi se repetă procedeul pentru următorul element până se ajunge la un element din intervalul de interes. Răspunsul la query este exact valoarea acestuia, deoarece este cea mai mare valoare din intervalul dorit.

Cum fiecare element este inserat o singură dată şi tot o singură dată este extras, numărul total de operaţii pentru cele N extrageri, inserări şi query este O(N) deci O(1) amortizat pentru fiecare operaţie în parte.

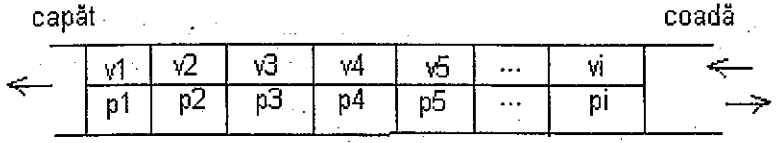
Structura astfel construită poartă numele de **deque** şi are următoarele caracteristici:

- Deque reprezintă o structură de date de tip listă (coadă) având două capete, unul pentru inserări iar celălalt pentru query.
- Structura poate înlocui cu succes un caz particular de Heap, atunci când elementele sunt extrase din structură în ordinea în care au fost inserate.

Un exemplu concret la care ne putem gândi este un vector de numere şi un interval de L elemente consecutive ale acestuia. Iniţial se aleg primele L numere din stânga, urmând ca apoi la fiecare pas acest interval să se deplaseze cu o poziţie spre dreapta. La fiecare pas se vor pune întrebări de genul «care este cea mai mică valoare din interval, sau care este cea mai mare valoare din interval», întrebări la care se poate da răspunsul în O(logL) operaţii implementând un Heap sau O(1) în cazul unui Deque.



Un deque arată în felul următor:



- Prin **capăt** elementele pot fi doar scoase, spre deosebire de **coadă** de unde pot intra şi ieşi elemente.
- După cum se poate observa fiecare element inserat în listă are două câmpuri Vi – valoarea (informaţia) şi Pi – poziţia pe care se află în vector.
- În funcţie de tipul de Heap pe care îl înlocuieşte Deque, elementele din listă sunt ordonate descrescător (maxHeap), respectiv crescător (minHeap) după V, proprietate ce trebuie păstrată după fiecare operaţie de inserare.
- De reţinut faptul că **p1<p2<...<pi** indiferent de tipul de Deque folosit.

Operaţia de inserare în deque constă în doi paşi:

1. cât timp mai sunt elemente în deque, dacă proprietatea de listă sortată se anulează cu inserarea noului element, este extras din coadă ultimul element.
2. noul element este inserat în coadă.

Funcţia query include şi operaţia de extragere:

1. cât timp mai sunt elemente în deque, dacă poziţia în vector a primul element din listă (p1) este mai mică decât poziţia de început a intervalul curent de lungime L(se află în afara intervalului, mai exact în dreapta lui, deci nu ne mai interesează), este extras elementul din capăt.
2. valoarea elementului din capăt este returnat drept răspuns la query.

După o analiză amortizată a numărului de operaţii se obţine O(1) pentru ambele operaţii insert şi query (fiecare element din vector este inserat şi extras o singură dată).

PROBLEME REZOLVATE

MAX-MIN

»»»ENUNŢ:

Se consideră un vector de N numere naturale. Să se găsească un interval de L numere consecutive astfel încât valoarea minimă din interval să fie cât mai mare.

Date de intrare:

Pe prima linie din fişierul **interval.in** se găsesc numerele N şi L.

Pe a doua linie se găsesc separate prin spaţiu cele n numere întregi.

Date de ieşire:

Pe prima linie din fişierul **interval.out** se va găsi mesajul: "Intervalul cu valoarea minima cât mai mare este:".

Pe a doua linie se vor scrie L numere separate prin spaţiu reprezentând intervalul cu cel mai mare minim. Complexitatea algoritmului este O(N).

Exemplu:

interval.in	interval.out
10 4 1 6 5 4 3 9 8 0 2 11	Intervalul cu valoarea minima cât mai mare este: 5 4 3 9

Restricţii:

1≤L≤N≤1000000

»»»SOLUŢIE:

```
#include<fstream.h>
#define MAXN 1000000
int MAX,S,N, L, P, capat, coada, A[MAXN], deque[MAXN][2];
ifstream f("interval.in");
ofsream g("interval.out");
void insert(int v,int p)
{
```

```
  while (capat<=coada && v<deque[coada][0]) coada--;
  coada++;deque[coada][0]=v;deque[coada][1]=p;
}
int query()
{
  while(capat<=coada && deque[capat][1]<P) capat++;
  return deque[capat][0];
}
int main()
{int i;
   f>>N>>L;
   for(i=1;i<=N;i++) f>>A[i];
   capat=1;   MAX=-1;
   for(i=1;i<L;i++)  insert(A[i],i);
   for(i=L,P=1;i<=N;i++,P++)
   {
     insert(A[i],i);
      if (query()>MAX) MAX=query(),S=i;
    }
   g<<"Intervalul cu valoarea minima cat mai mare este:\n";
   for(i=S-L+1;i<=S;i++)      g<<A[i]<<' ';
return 0;
}
```

MONEZI

⋙ENUNȚ:

Se consideră N tipuri de monezi de diferite valori V_i. Pentru fiecare tip i de monedă Costel are în buzunar C_i exemplare. El dorește să achite la bancă o rată în valoare de S lei folosind un număr minim de monezi.

Cerință:

Ajutați-l să determine o modalitate de a obține suma S folosind un număr minim de monezi din buzunar.

Date de intrare:

Pe prima linie a fișierului de intrare **monezi2.in** se găsește N, numărul de tipuri de monezi. Pe fiecare din următoarele N linii se află

câte 2 numere V_i și C_i reprezentând valoarea și numărul de exemplare de tipul i pe care le deține Costel. Pe ultima linie a fișierului se găsește S, suma ce trebuie obținută.

Date de ieșire:

Pe prima linie a fișierului de ieșire **monezi2.out** se află M, numărul minim de monezi necesare pentru a forma suma S.

Pe a doua linie se află M numere X_i, reprezentând cele M valori ale monezilor ce compun suma S. Ordinea în care sunt scrise acestea nu contează.

Exemplu:

monezi.in	monezi.out
5 1 5 2 3 3 2 4 9 5 2 7	2 3 4

Restricții:

$1 \le N, C_i \le 500$

$1 \le S, V_i \le 1000$

⋙SOLUȚIE:

Problema se rezolvă folosind programarea dinamică. Se consideră matricea D[i][j] = numărul minim de monezi necesare pentru a obține suma j folosind doar primele i tipuri de monezi.

Suma j se obține luând un număr x, $0 \le x \le C_i$, de monezi de tipul i, și completând suma rămasă j – x* V_i cu ajutorul primelor i-1 tipuri. Evident pot exista mai multe modalități de a obține suma j, din care o vom alege pe aceea cu număr minim de monezi. Relația de recurență pentru D este:

$$D[i][j] = \begin{cases} \min\{ x + D[i-1][j-x*V_i],\ 0 \le x \le C_i \} \\ \text{INF, dacă suma j nu se poate obține folosind primele i tipuri de monezi.} \end{cases}$$

Rezultatul este D[N][S]. O implementare banală a recurenței folosind trei for-uri(două pentru i şi j, şi al treilea pentru x) are complexitatea $O(N^2 *S)$.

Complexitatea poate fi redusă la O(N*S) optimizând soluția cu ajutorul unor **Deque**-uri. Pentru fiecare tip de monedă i, şirul de sume de la 0 la S se împarte în alte V_i şiruri numerotate de la 0 la V_i -1 de forma $k*V_i$, $k*V_i+1$, $k*V_i+2$, ..., $k*V_i+V_i-1$.

Suma j va face parte din şirul [j modulo V_i]. Vom aplica algoritmul de mai sus, dar puțin modificat: Nu vom mai folosi al treilea for pentru x, iar când ne deplasăm cu for-ul j, fiecare valoare D[i-1][j] o vom insera într-un deque aparținând şirului [j%V_i]. Fiecare din cele V_i deque-uri va reține ultimele C_i sume mai mici decât j din şirul respectiv.

La fiecare pas D[i][j] se va calcula pe baza valorii minime aflate în deque-ul şirului [j%V_i]. Pentru a putea fi comparate direct, valorile se introduc în deque modificate. Astfel pentru suma j se va introduce în deque-ul [j%V_i] valoarea D[i-1][j] + y, unde $\mathbf{j + y*V_i = MAXS*V_i + [j\%V_i]}$.

Y reprezintă numărul de monezi de valoare V_i ce mai trebuie adăugate la j pentru a obține suma MAXS*V_i + [j%V_i]. MAXS este o constantă definită la începutul programului. În acest mod, două valori D[i-1][j] şi D[i-1][j-l*V_i] pot fi comparate direct iar minimul dintre ele va fi util pentru a determina valorile D[i][j+m*V_i] următoare.

Înțelegerea şi implementarea acestei soluții necesită răbdare şi atenție.

»»»PROGRAM:

```
#include<stdio.h>
#define MAXN 501
#define MAXS 1001
#define INF 10000000
struct qeueNode {int x,v;} QS[MAXS][MAXS];
int S,N,V[MAXN],C[MAXN],D[MAXN][MAXS]ş
int U[MAXN][MAXS],PS[MAXS][2];
void citesteDate()
{int i;
 freopen("monezi.in","r",stdin);
 scanf("%d",&N);
 for(i=1;i<=N;i++)
   scanf("%d %d",&V[i],&C[i]);
 scanf("%d",&S);
}
```



```
void initDeques()
{int i;
 for(i=0;i<MAXS;i++) PS[i][0]=1,PS[i][1]=0;
}
void insert(int i,int sum)
{int p1,p2,q,y,nV;
 q=sum%V[i];
 p1=PS[q][0];
 p2=PS[q][1];
 y=(MAXS*V[i] + q - sum) / V[i];
 nV = D[i-1][sum] + y;
 while(p1<=p2 && QS[q][p2].v > nV) p2--;
 p2++;
 QS[q][p2].v= nV;
 QS[q][p2].x=sum;
 PS[q][1]=p2;
}
int query(int i,int sum)
{int p1,p2,q;
  q=sum%V[i];
  p1=PS[q][0];
  p2=PS[q][1];
  while (p1<=p2 && ((sum - QS[q][p1].x)/V[i])>C[i]) p1++;
  PS[q][0]=p1;
  return QS[q][p1].x;
}
void proceseaza()
{int i,j,x;
   for(i=1;i<=N;i++) D[i][0]=0;
   for(i=1;i<=S;i++) D[0][i]=INF;
   for(i=1;i<=N;i++)
     {
  initDeques();
  for(j=0;j<=S;j++)
    {insert(i,j);
     x=query(i,j);
     U[i][j]=(j-x)/V[i];
     D[i][j]=D[i-1][x] + U[i][j];
    }
     }
}
```

```
void afiseazaRez()
{int i;
 freopen("monezi.out","w",stdout);
 if (D[N][S]==INF)
   printf("IMPOSIBIL!\n");
 else
   {
    printf("%d\n",D[N][S]);
    for(;N;S-=U[N][S]*V[N],N--)
  for(i=1;i<=U[N][S];i++)
     printf("%d ",V[N]);
   }
}
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

SUMA MIN

»»»ENUNȚ:

Se consideră o secvență de **N** numere întregi. Să se determine un interval de lungime cuprinsă între L şi U cu suma elementelor minimă.

Date de intrare:
Pe prima linie a fişierului de intrare **secv.in** se dau N, L şi U.
Pe a doua linie se găsesc separate prin câte un spațiu numerele întregi.

Date de ieşire:
Pe prima linie a fişierului de ieşire **secv.out** se va trece suma minimă.
Pe a doua linie se vor trece două numere separate prin spațiu reprezentând capetele intervalului.

Exemplu

Secv.in	Secv.out
6 2 4	-1
4 6 8 -5 7 -3	4 6

Restricții:
1≤L≤U≤N≤1000000
Complexitatea algoritmului este O(N).

»»»SOLUȚIE:

Problema se rezolvă în felul următor: parcurgem vectorul de la stânga spre dreapta şi determinăm pentru fiecare poziție i subsecvența de sumă minimă care se termină pe această poziție. Vom exprima suma prin Xi – Xj, unde Xi şi Xj reprezintă suma elementelor până la i, respectiv j.

Pentru a minimiza suma, Xj trebuie să aibă o valoare cât mai mare. Valoarea lui j este variabilă şi trebuie să îndeplinească condițiile j + L ≤i şi i≤ j+U.

Pentru ai obține pe j cu Xj maxim în timp O(1) vom întreține un deque cu toate valorile Xj care îndeplinesc cele 2 condiții. La fiecare pas, odată cu avansarea lui i, se inserează în deque Xj, iar prin capătul celălalt sunt scoase toate valorile aflate la o distanță mai mare de U față de noua poziție. Complexitatea algoritmului este O(N).

»»»PROGRAM:

```
#include<stdio.h>
#define MAXN 1000001
#define INF 1000000000
int pA,pB,REZ,N,U,L,V[MAXN],deque[MAXN];
int poz[MAXN],X[MAXN],query[MAXN];
void citesteDate()
{int i;
 freopen("secv2.in","r",stdin);
 scanf("%d %d %d",&N,&L,&U);
 for(i=1;i<=N;i++)  scanf("%d ",&V[i]);
}
void proceseaza()
{int i,j,s1,s2;
  for(s1=1,s2=1,i=1;i<=N;i++)
    {
```

```
    X[i]=X[i-1]+V[i];
    while (s1<=s2 && deque[s2]<X[i]) s2--;
    s2++;
    deque[s2]=X[i];
    poz[s2]=i;
    while (s1<=s2 && poz[s1]+U-L<i) s1++;
    query[i]=poz[s1];
   }
 REZ=INF;
 for(i=L;i<=N;i++)
   {j=query[i-L];
    if (X[i]-X[j]<REZ)
      {REZ=X[i]-X[j];pA=j+1;pB=i;}
   }
}
void afiseazaRez()
{
 freopen("secv2.out","w",stdout);
 printf("%d\n%d %d\n",REZ,pA,pB);
}
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

MATRICE

»»»ENUNȚ:

Se consideră o matrice de dimensiune NXM care conține numere naturale. Să se determine un dreptunghi de arie maximă astfel încât diferența dintre valoarea maximă şi minimă din regiune să fie mai mică sau egală cu D.

Date de intrarea:

Pe prima linie din fişierul **matrice.in** se găsesc numerele naturale N, M şi D.

Pe următoarele N numere se găsesc câte M numere naturale separate prin câte un spațiu.

Date de ieşire:

Pe prima linie din fişierul **matrice.out** se va trece un număr ce va reprezenta aria dreptunghiului găsit.

Pe a doua linie se găsesc patru numere ce reprezintă coordonatele colțurilor dreptunghiului din stânga sus şi dreapta jos.

Exemplu:

matrice.in	**matrice.out**
4 5 2	6
1 2 3 4 5	3 1 4 3
10 9 8 7 6	
5 5 5 5 5	
6 5 4 3 2	

Restricții:

1≤N,M ≤200

»»»SOLUȚIE:

Problema are complexitate $O(N^3)$ folosind deque. Se fixează prin i şi j liniile de început respectiv de sfârşit ale dreptunghiului de interes şi se "comprimă" toate elementele de pe liniile cuprinse între acestea în doi vectori. Primul va reține pe poziția k elementul maxim aflat pe coloana k între liniile i şi j inclusiv, iar celălalt va reține elementul de valoare minimă.

Astfel problema se rezumă la a determina o subsecvență de lungime maximă pe un vector cu 2 valori în fiecare element, astfel încât diferența dintre minim şi maxim să nu depăşească valoarea D.

Pentru a realiza acest lucru vom construi două deque-uri, unul pentru maxim, iar celălalt pentru minim în care vom insera pe parcurs elementele din vector. Parcurgem vectorul pornind cu k=1 spre N şi reținem într-o variabilă pmin care inițial are valoarea 1, cea mai mică poziție care îndeplineşte condiția: diferența dintre maximul şi minimul elementelor din intervalul [pmin,k] este mai mică sau egală cu D.

Toate elementele din cele două deque-uri vor avea poziții cuprinse între pmin şi k. La fiecare pas, după ce efectuăm inserarea minimului şi maximului de pe poziția k, avansăm cu pmin şi reactualizăm deque-urile astfel încât condiția să fie din nou îndeplinită.

Lungimea maximă L pe care o vom determina pentru acest

vector este înmulțită cu j-i+1 şi aria rezultată o comparăm cu aria maximă determinată până în acel moment.

⋙PROGRAM:

```
#include<stdio.h>
#define MAXN 200
int ARIE,aX,aY,bX,bY,N,M,D,A[MAXN][MAXN],VMAX[MAXN];
int VMIN[MAXN],dequeMin[MAXN],pozMin[MAXN];
int dequeMax[MAXN],pozMax[MAXN];
void citesteDate()
{int i,j;
 freopen("matrice.in","r",stdin);
 scanf("%d %d %d ",&N,&M,&D);
 for(i=1;i<=N;i++)
   for(j=1;j<=M;j++)
     scanf("%d",&A[i][j]);
}
void subsecv(int h1,int h2)
{int h=h2-h1+1, pA,pB,L=0,MIN,MAX,k,pmin;
int  s1min,s1max,s2min,s2max;
 for(k=1,pmin=1,s1min=s1max=1,s2min=s2max=0;k<=M;k++)
   {
    while(s1min<=s2min && dequeMin[s2min]>VMIN[k])s2min--;
    s2min++;
    dequeMin[s2min]=VMIN[k];
    pozMin[s2min]=k;
    while(s1max<=s2max && dequeMax[s2max]<VMAX[k])
s2max--;
    s2max++;
    dequeMax[s2max]=VMAX[k];
    pozMax[s2max]=k;
    MIN=dequeMin[s1min];
    MAX=dequeMax[s1max];
    for(;MAX-MIN>D && pmin<=k;)
      {
       pmin++;
       while(s1min<=s2min && pozMin[s1min]<pmin) s1min++;
       while(s1max<=s2max && pozMax[s1max]<pmin) s1max++;
       MIN=dequeMin[s1min];
       MAX=dequeMax[s1max];
```



```
      }
    if (k-pmin+1>L)
      L=k-pmin+1,pA=pmin,pB=k;
   }
  if (L*h>ARIE)
    {ARIE=L*h;
     aX=h1;aY=pA;
     bX=h2;bY=pB;
    }
}
void proceseaza()
{int i,j,k;
  for(i=1;i<=N;i++)
    {for(k=1;k<=M;k++)
      VMIN[k]=VMAX[k]=A[i][k];
     for(j=i+1;j<=N;j++)
       {
        for(k=1;k<=M;k++)
          {if (A[j][k]>VMAX[k]) VMAX[k]=A[j][k];
           if (A[j][k]<VMIN[k]) VMIN[k]=A[j][k];
          }
        subsecv(i,j);
       }
    }
}

void afiseazaRez()
{
 freopen("matrice.out","w",stdout);
 printf("%d\n",ARIE);
 printf("%d %d %d %d\n",aX,aY,bX,bY);
}
int main()
{
citesteDate();
proceseaza();
afiseazaRez();
return 0;
}
```

PROBLEME PROPUSE

PB1. Se consideră o matrice de dimensiuni NxM cu valori din intervalul [0,100000]. Să se determine un dreptunghi de lungime minim L şi înălţime minim H astfel încât media aritmetică a elementelor conţinute în acesta să fie maxim. Valoarea maximă găsită se va afişa cu o precizie de 2 zecimale.

$0 < L <= N <=100$
$0 < H <= M <=100$

PB2. Se consideră o secvenţă de N numere întregi. Să se determine un interval de lungime cuprinsă între L şi U cu suma elementelor maximă.

PB3. Transportul de călători între n localităţi ale unui judeţ este asigurat cu ajutorul unor autobuze. Între oricare două localităţi I şi J există cel mult un autobuz direct, care merge o dată pe zi, având un orar cunoscut (ora de plecare din I şi ora de sosire în J, numere întregi). Să se găsească timpul total în care un călător poate ajunge dintr-o localitate oarecare X în altă localitate Y. Se consideră că persoana se afla în staţia din localitatea X la ora 0 şi trebuie să ajungă în Y pâna la ora 24 a aceleiaşi zile.

PB4. Fie o regiunea dreptunghiulară **M** x **N** km împărţită în zone de 1 x 1 km. Fiecare zona are o altitudine cunoscută, specificată pe hartă în metri faţă de nivelul mării. Astfel, toate altitudinile sunt numere întregi nenegative.Trebuie găsit un dreptunghi care să aibă laturile paralele cu ale regiunii. In plus, terenul nu trebuie ales la întâmplare. Se defineşte *diferenţa de altitudine* a unui teren ca fiind diferenţa dintre altitudinea maximă şi altitudinea minimă ale unor zone de 1 km^2 cuprinse în teren. D*iferenţa de altitudine* a terenului ales trebuie sa fie cât mai mică.

Dintre mai multe oferte de forma (**DX DY**), semnificând faptul că se poate alege oriunde în regiune un teren cu laturile de dimensiuni **DX** şi **DY** trebuie aflată cea care are *diferenţa de altitudine* minimă.

CAPITOLUL 3

PROGRAMARE DINAMICĂ

Considerente teoretice

Presupunem că avem o mulţime A şi dorim să realizăm o mulţime B. La un pas oarecare putem alege între mai multe elemente ale lui A. Putem lua mai multe decizii $d_1,d_2,...d_n$ şi fiecare dintre acestea poate transforma problema dintr-o stare s_{i-1} într-o stare s_i. Important este să luăm decizia optimă care conduce la o decizie optimă de asamblu. Programarea dinamică este o tehnică de programare care este utilizată pentru a optimiza o problema ce satisface *principiul optimalităţii*: într-o secvenţă optimă de decizii, fiecare subsecvenţă trebuie să fie de asemenea optimă. Acest principiu nu este întotdeauna valabil şi aceasta se întâmplă atunci când subsecvenţele nu sunt independente, adică atunci când optimizarea unei secvenţe intră în conflict cu optimizarea celorlalte subsecvenţe.

În lucrările de referinţă sunt prezentate trei strategii de rezolvare a problemelor folosind această metodă: înainte (când starea s_i depinde de stările s_{i-1}, s_{i-2}...), înapoi (când starea s_i depinde de stările s_{i+1}, s_{i+2}...) şi mixtă (când starea s_i depinde atât de stările s_{i-1}, s_{i-2} cât şi de stările s_{i+1}, s_{i+2}...).

Pentru a fi eficientă, metoda programării dinamice trebuie să rezolve o subproblemă o singură dată şi să memoreze această soluţie pentru a o folosi în cadrul unei alte subprobleme. Spunem că programarea dinamică evită rezolvarea de mai multe ori a aceleiaşi subprobleme prin memorarea rezultatelor parţiale. Memorând deciziile la fiecare etapă, putem folosi atât strategia înainte cât şi strategia înapoi şi putem de asemenea furniza rapid o soluţiei care conduce sistemul din starea iniţială în cea finală în condiţii de optim.

Soluţia se determină prin rezolvarea şi combinarea subproblemelor în ordinea crescătoare a dimensiunii lor, de jos în sus (*bottom-up),* ceea ce presupune cunoaşterea de la început a subproblemelor ce apar în descompunerea unei probleme.

Când rezolvăm o problemă folosind programarea dinamică vom parcurge următorii paşi:

- Stabilm structura soluției (vector sau matrice) astfel încât fiecare componentă să conțină o soluție optimă a unei subprobleme.
- Definim recursiv valoarea unei soluții optime: definim valorilor celor mai mici subpropbleme şi formula recursivă a termenului general.
- Calculăm de jos în sus o soluție optimă.
- Opțional, afişăm şirul de subprobleme care a dus la soluția optimă. Se pleacă de la soluția optimă şi la fiecare pas se alege problema care a dus la soluția optimă.

Prin folosirea acestei metode se obțin soluții fără a lua în considerare toate cazurile posibile, ceea ce în anumite situații îi conferă acestei tehnici un avantaj net în fața metodei Backtracking.

PROBLEME REZOLVATE

BINAR

›››ENUNȚ:

Se consideră toate numerele de N cifre scrise în baza 2. Dintre acestea, unele au proprietatea că pentru oricare poziție i, numărul de cifre 1 din stânga este mai mare sau egal cu numărul de cifre 0 din stânga, respectiv numărul de cifre 1 din dreapta este mai mare sau egal cu numărul de cifre 0 din partea dreaptă. Pentru un număr N dat, să se determine câte astfel de numere au proprietatea de mai sus.

Date de intrare:

Pe prima linie a fişierului de intrare **binar.in** se găseşte N, numărul de cifre.

Date de ieşire:

Pe singura linie a fişierului de ieşire **binar.out** se va afla numărul cerut.

Exemplul 1:

binar.in
6
binar.out
9

Exemplul 2:

binar.in
19
binar.out
25842

Observații:

Cele 9 numere din primul exemplu sunt: 101011, 101101, 101111, 110011, 110101, 110111, 111011, 111101, 111111.

Restricții:

$0 < N < 63$

›››SOLUȚIE:

Problema face parte din categoria problemelor de numărare şi se rezolvă prin programare dinamică. Se defineşte matricea S[i][j][k] ce reprezintă câte numere de i cifre cu diferența dintre numărul de cifre 1 şi numărul de cifre 0 egală cu j au pentru fiecare poziție diferența dintre numărul de cifre 1 şi 0 din dreapta mai mare sau egală cu k. Fixând ultima cifră cu cele două valori posibile se obține recurența:

S[i][j][k] = S[i-1][j-1][k-1] + S[i-1][j+1][k+1].

Rezultatul va fi dat de suma S[N][0][0] + S[N][1][0] + S[N][2][0] +...+ S[N][N][0].

Complexitatea algoritmului este $O(N^3)$.

›››PROGRAM:

```
#include<stdio.h>
#define MAXN 63
int N;
long long S[MAXN][MAXN][2*MAXN];
void citesteDate()
{
 freopen("binar.in","r",stdin);
 scanf("%d",&N);
}
void proceseaza()
{
int i,j,k;
 for(i=0;i>-MAXN;i--)
```

```
    S[0][0][i+MAXN]=1;
  for(i=1;i<=N;i++)
    for(j=0;j<=N;j++)
      for(k=-N;k<=N;k++)
    {
     if (j>0) S[i][j][k+MAXN] += S[i-1][j-1][k-1+MAXN];
     if (k<0) S[i][j][k+MAXN] += S[i-1][j+1][k+1+MAXN];
    }
}
void afiseazaRez()
{
int i;long long R;
 freopen("binar.out","w",stdout);
 for(i=R=0;i<=N;i++)
     R+=S[N][i][0+MAXN];
 printf("%lld\n",R);
}
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

GAME

⋙ENUNȚ(enunț modificat):

Se dau doi vectori A şi B de numere naturale cu N, respectiv M elemente pe care se defineşte un joc cu următoarele reguli:

Jocul constă într-o succesiune de mutări. La o mutare se aleg ultimele x elemente din primul vector care însumate dau s1 şi ultimele y din cel de-al doilea care însumate dau s2 (1≤x≤numere de elemente rămase în primul vector, 1≤y≤numere de elemente rămase în cel de-al doilea vector). Produsul s1*s2 este adăugat la o sumă S care la începutul jocului ia valoarea 0. După această operație, elementele alese sunt eliminate din vectori şi se trece la următoare mutare, până când cei doi vectori rămân fară elemente. Să se determine valoarea minimă posibilă ce o poate avea S la finalul jocului.

Date de intrare:

Pe prima linie a fişierului de intrare **game.in** se află două numere N şi M.

Pe a doua linie se află N numere, reprezentând elementele din vectorul A.

Pe cea de-a treia linie se află M numere reprezentând vectorul B.

Date de ieşire:

Pe singura linie a fişierului de ieşire **game.out** se va afla valoarea minimă a lui S.

Restricții:

1 < N,M < 2004

valorile din cei doi vectori sunt numere naturale < 100.

Exemplu:

game.in

4 5

1 2 3 4

5 4 3 2 1

game.out

23

Problemă propusă la BOI, 2004

Explicație: Se efectuează două mutări: prima în care se aleg 3 elemente din primul vector şi un element din al doilea vector, iar a doua mutare alege un element din primul şi 4 elemente din al doilea (2+3+4)*1 + 1*(5+4+3+2) = 9 + 14 = 23.

⋙SOLUȚIE:

Problema se rezolvă folosind programarea dinamică. Se ia o matrice în care S[i][j] este suma minimă ce se poate obține la finalul jocului, folosind doar primele i elemente din vectorul A şi primele j din B.

Rezultatul final va fi S[N][M]. Vom porni de la cea mai simplă formulă de recurență, pe care treptat o vom rafina folosindu-ne de unele observații utile.

S[i][j] = min{ S[k][l] + suma1(k+1..i) * suma2(l+1..j) |cu k≤i şi l≤j}

Fixăm cu două for-uri i şi j, apoi cu alte două for-uri k şi l găsim minimul pentru S[i][j].

Complexitatea în acest stadiu este $O(N^2*M^2)$, destul de mare pentru un programator serios.

O primă modalitate de reducere a complexității este aceea de a observa că fie k=i-1 fie l=j-1 în soluția optimă (adică fie x=1, fie y=1, sau ambele). Se poate demonstra foarte uşor acest lucru: presupunând prin reducere la absurd că la o mutare x>1 şi y>1, suma adăugată este s1*s2 care mai poate fi scrisă şi astfel (s1' + s1") * (s2' + s2") cu s1'+s1" = s1 şi s2'+s2" = s2 deoarece s1 şi s2 sunt formate din cel puțin două elemente fiecare.

Scrisă în forma aceasta, mutarea poate fi spartă în alte două mutări de cost total mai mic:

s1' * s2' + s1" * s2" < (s1' + s1") * (s2' + s2")

S[i][j]=min{ min{S[i-1][k] + A[i]*suma2(k+1..j)}, min{S[k][j-1]+suma1(k+1..i)*B[j]} }

Acum complexitatea soluției se reduce la O(N*M*(N+M)) adică aprox N^3, ceva mai bine decât prima soluție dar tot nu îndeajuns de bună.

Următoarea formă a soluției necesită puțin fler şi exercițiu în astfel de probleme de programare dinamică.

Vom considera trei matrici S1,S2,S3 fiecare având următoarele caracteristici:

S1[i][j] – suma minimă ce se poate obține la final luând x=1 şi y>1 la mutarea curentă.

S2[i][j] – suma minimă ce se poate obține la final luând x>1 şi y=1 la mutarea curentă.

S3[i][j] – suma minimă ce se poate obține la final luând x=1 şi y=1 la mutarea curentă.

Relațiile de recurență dintre cele trei matrici sunt:

S1[i][j] = A[i]*B[j] + min{S1[i][j-1], S3[i][j-1]}
S2[i][j] = A[i]*B[j] + min{S2[i-1][j], S3[i-1][j]}
S3[i][j] = A[i]*B[j] + min{S1[i-1][j-1], S2[i-1][j-1], S3[i-1][j-1]}

Rezultatul final va fi MIN{S1[N][M], S2[N][M], S3[N][M]}

Complexitatea algoritmului în forma finală este O(N*M).

»»»PROGRAM:

```
#include<stdio.h>
#define MAXN 2001
#define INF 1000000000
int N,M,A[MAXN],B[MAXN];
```



```
int S1[MAXN][MAXN],S2[MAXN][MAXN],S3[MAXN][MAXN];
void citesteDate()
{
int i;
 freopen("game.in","r",stdin);
 scanf("%d %d",&N,&M);
 for(i=1;i<=N;i++) scanf("%d",&A[i]);
 for(i=1;i<=M;i++) scanf("%d",&B[i]);
}
int min(int a,int b) {if (a<b) return a;return b;}
int min(int a,int b,int c) { return min(min(a,b),c);
}
void proceseaza()
{
int i,j;
 for(i=1;i<=N;i++) S1[i][0]=S2[i][0]=S3[i][0]=INF;
 for(i=1;i<=M;i++) S1[0][i]=S2[0][i]=S3[0][i]=INF;
 for(i=1;i<=N;i++)
   for(j=1;j<=M;j++)
     {
  S1[i][j] = A[i]*B[j] + min(S1[i][j-1],S3[i][j-1]);
  S2[i][j] = A[i]*B[j] + min(S2[i-1][j],S3[i-1][j]);
  S3[i][j] = A[i]*B[j] + min(S1[i-1][j-1],S2[i-1][j-1],S3[i-1][j-1]);
     }
}
void afiseazaRez()
{
 freopen("game.out","w",stdout);
 printf("%d\n",min(S1[N][M],S2[N][M],S3[N][M]));
}

int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

MUZICA

»»»ENUNȚ:

Americanul tocmai a încheiat o afacere de succes şi acum doreşte să dea o petrecere pentru a sărbători. El vrea ca petrecerea să țină cel puțin A ore dar nu mai mult de B ore.

În acest fel, invitații săi se vor simți bine şi nu vor uita să mai treacă şi pe acasă. Cum duce lipsă de muzică (ciudat nu? tocmai el), americanul îi roagă pe amicii săi să meargă să-i cumpere câteva CD-uri. Aceştia acceptă cu mare plăcere, mai ales că au de gând să-şi scoată ceva profit din treaba asta.

Astfel ei vor să-i prezinte americanului o factură cât mai mare, dar în realitate să cumpere cât mai ieftin. Acesta este dispus să plătească, atât timp cât durata totală a CD-urilor pe care le primeşte este egală cu cea de pe factură. Pe factură nu sunt scrise decât durata totală a CD-urilor precum şi costul total ce trebuie achitat.

Cerință:

Americanul vrea să ştie care este suma maximă şi minimă ce trebuie plătită pentru a cumpăra CD-uri cu lungimea totală egală cu cea de pe factură. Ajutați-l să afle astfel cât le iese "amicilor" pentru efortul depus.

Date de intrare:

Pe prima linie a fişierului de intrare **musik.in** se află două numere A şi B reprezentând intervalul de timp exprimat în ore în care să se încadreze petrecerea.

Pe a doua linie a fişierului se află N, numărul de CD-uri disponibile în magazin.

Pe următoarele N linii se găsesc câte două numere C_i şi T_i, reprezentând costul şi durata (în minute) a CD-ului i.

Date de ieşire:

Pe prima linie a fişierului de ieşiere **musik.out** se află durata petrecerii americanului exprimată în formatul XhY' (X ore şi Y minute).

Pe cea de-a doua linie se găseşte un singur număr D, profitul maxim ce se poate obține.

Exemplu:

musik.in
4 5
10
100 80
821 74
122 50
123 60
130 78
321 65
22 58
13 61
922 53
423 69

musik.out
4h6'
1891

Explicație:

Costul maxim:

Selectând CD-urile 2,3,9,şi 10 se obține:

Costul 821 + 122 + 922 + 423 = 2288, iar durata 74 + 50 + 53 + 69 = 246 minute.

Costul minim:

Selectând CD-urile 3,4,5, şi 7 se obține:

Costul 122 + 123 + 130 + 22 = 397, iar durata 50 + 60 + 78 + 58 = 246 minute.Profitul care le iese este 2288-397 = 1891.

Restricții:

$0 < N < 5001$
$0 < A < B < 24$
$0 < C_i < 1000$
$0 < T_i < 100$

»»»SOLUȚIE:

Problema poate fi împărțită în două subprobleme astfel pentru o valoare T să se determine:

1. suma maximă ce poate fi plătită pentru a cumpăra CD-uri cu o durată totală T;

2. suma minimă ce poate fi plătită pentru a cumpăra CD-uri de durată totală T.

Soluția problemei constă în a detemina un T în intervalul A-B astfel încât diferența dintre suma maximă şi minimă să fie cât mai mare. Cele două subprobleme se rozolvă cu ajutorul programării dinamice astfel:

Se consideră matricile MAX[i][j] = suma maximă ce se poate obține selectând din primele i CD-uri, câteva cu durata totală j. Analog MIN[i][j] = suma minimă.

Relația de recurență este:

MAX[0][j] = - INF

MIN[0][j]= INF

MAX[i][j] = max{MAX[i-1][j], MAX[i-1][j-T[i]] + C[i]}

MIN[i][j] = min{MIN[i-1][j], MIN[i-1][j-T[i]] + C[i]}

Drept soluție se va alege un T pentru care MAX[N][T]-MIN[N][T] să fie cât mai mare. Evident 60*A≤ T ≤ 60*B. Cantitatea de memorie folosită în program poate fi redusă semnificativ, observând că pentru a calcula MAX[i][j] şi MIN[i][j] nu avem nevoie decât de linia precedentă.Complexitate algoritmului este $O(N^2)$.

›››PROGRAM:

```
#include<stdio.h>
#define MAXT 1500
#define MAXN 5001
#define INF 1000000000
int TREZ,D=-INF,N,A,B;
int C[MAXN],T[MAXN],MAX[MAXN][MAXT],MIN[MAXN][MAXT];
void citesteDate()
{int i;
 freopen("musik.in","r",stdin);
 scanf("%d %d %d",&A,&B,&N);
 for(i=1;i<=N;i++)
   scanf("%d %d",&C[i],&T[i]);
}
void proceseaza()
{int i,j,L;
 L=B*60;
 for(i=1;i<=L;i++)
   MAX[0][i]=-INF,MIN[0][i]=INF;
 for(i=1;i<=N;i++)
   for(j=1;j<=L;j++)
```



```
     {
    MAX[i][j]=MAX[i-1][j];
    if (T[i]<=j)
      if (MAX[i-1][j-T[i]]+C[i]>MAX[i][j])
        MAX[i][j]=MAX[i-1][j-T[i]]+C[i];
    MIN[i][j] = MIN[i-1][j];
    if (T[i]<=j)
      if (MIN[i-1][j-T[i]]+C[i]<MIN[i][j])
        MIN[i][j]=MIN[i-1][j-T[i]]+C[i];
     }
for(i=A*60;i<=L;i++)
  if (MAX[N][i]-MIN[N][i]>D)
     D=MAX[N][i]-MIN[N][i],TREZ=i;
}
void afiseazaRez()
{int i,j;
 freopen("musik.out","w",stdout);
 printf("%dh%d'\n%d\n",TREZ/60,TREZ%60,D);
}
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

NUMERE

›››ENUNȚ:

Se consideră toate numerele de N cifre scrise în baza 10, inclusiv cele care au la început 0-uri. Pentru un N dat, în total sunt 10^N astfel de numere. Să se determine câte dintre ele au următoarea caracteristică: numărul de apariții a fiecărei cifre i este cuprins în intervalul închis [Ai,Bi].

Date de intrare:

Pe prima linie a fişierului de intrare **numere.in** se găseşte un singur număr N, numărul de cifre.

Pe următoarele 10 linii se găsesc câte 2 numere Ai şi Bi,

intervalul în care trebuie să fie numărul de apariții al cifrei i.

Date de ieşire:

Pe singura linie a fişierului de ieşire **numere.out** se va găsi răspunsul cerut.

Exemplu:

numere.in

```
4
0 1
2 3
0 6
1 8
0 10
0 9
1 1
0 11
0 13
0 17
```

numere.out

```
12
```

Explicații:

Cele 12 de numere sunt: 1136, 1163, 1316, 1361, 1613, 1631, 3116, 3161, 3611, 6113, 6131, 6311.

Restricții:

$0 < N < 30$

$0 \le Ai \le Bi < 30$

Numărul total de numere care îndeplinesc condițiile nu va depăşi valoarea 2^{63}.

»»»SOLUȚIE:

Problema se rezolvă folosind programarea dinamică. Vom considera o matrice S[i][j] - câte numere de i cifre există care conțin în scrierea lor doar cifrele 0..j şi care îndeplinesc restricțiile pentru cifrele 0..j.

Relația de recurență între S-uri o găsim foarte uşor. Să luăm de exemplu un număr din mulțimea S[i][j]. Numărul de cifre j care apar în scrierea acestuia poate fi Aj, Aj+1,..Bj. Considerăm că numărul de

apariții este k(Aj≤ k ≤Bj). Cele k cifre pot fi plasate în număr în Comb(i,k) moduri, iar dacă le eliminăm obținem un număr din S[i-k][j-1]. De aici relația de recurență este:

S[i][j] = Sumă cu k de la Aj la Bj de Comb(i,k) *S[i-k][j-1].

Rezultatul va fi dat de S[N][9]. Complexitatea este $O(10 * N^2)$ dacă preprocesăm într-un vector C[i][j] = Comb(i,j).

»»»PROGRAM:

```
#include<stdio.h>
#define MAXN 31
int N,A[10],B[10];
long long S[MAXN][10],C[MAXN][MAXN];
void citesteDate()
{int i;
 freopen("numere.in","r",stdin);
 scanf("%d",&N);
 for(i=0;i<10;i++)
   scanf("%d %d",&A[i],&B[i]);
}
void proceseaza()
{int i,j,k;
 for(i=0;i<=N;i++) C[i][0]=1;
 for(i=1;i<=N;i++)
   for(j=1;j<=i;j++)
  C[i][j]=C[i-1][j]+C[i-1][j-1];
 for(i=0;i<10 && A[i]==0;i++)    S[0][i]=1;
 for(i=A[0];i<=B[0];i++) S[i][0]=1;
 for(i=1;i<=N;i++)
   for(j=1;j<10;j++)
  for(k=A[j];k<=B[j] && k<=i;k++)
    S[i][j]+=C[i][k]*S[i-k][j-1];
}
void afiseazaRez()
{int i,j;
 freopen("numere.out","w",stdout);
 printf("%lld\n",S[N][9]);
}
int main()
{
 citesteDate();
 proceseaza();
```

```
afiseazaRez();
return 0;}
```

PACHETE

▸▸▸ENUNȚ:

Anul acesta Moș Crăciun vine cu trenulețul. De pe acum face pregătirile și are o mică dilemă. Moș Crăciun are N pachete de diferite culori în care sunt cadouri pentru cei mici iar trenulețul lui are K vagoane. El dorește să așeze cele N pachete în cele K vagoane într-un mod cât mai estetic și dorește să afle câte moduri distincte de așezare există. Ordinea vagoanelor și a pachetelor într-un vagon sunt irelevante.

Date de intrare:
Pe prima linie a fișierului de intrare mc.in se găsesc două numere naturale N și K.

Date de ieșire:
Pe singura linie a fișierului de ieșire mc.out se găsește numărul de modalități distincte.

Exemplu:

mc.in	mc.out
4 3	6

Restricții:
$0 < N \leq 100$
$0 < K \leq N$

▸▸▸SOLUȚIE:
Problema se rezolvă folosind programarea dinamică. Să considerăm o amplasarea a primelor i pachete în j vagoane. Vom încerca acum să plasăm și pachetul cu indicele i+1. Fie îl punem într-un vagon gol și se obține astfel o amplasarea a primelor i+1 pachete în j+1 vagoane sau îl punem în unul din cele j vagoane deja folosite și se obțin j moduri de amplasare a primelor i+1 pachete în j vagoane. Relația de recurență este:
P[i][j] = P[i-1][j-1] + P[i-1][j]*j
Complexitatea algoritmului este O(N*K)

▸▸▸PROGRAM:

```
#include<stdio.h>
#define MAXN 101
int N,K,P[MAXN][MAXN];
void citesteDate()
{
 freopen("mc.in","r",stdin);
 scanf("%d %d",&N,&K);
}
void proceseaza()
{
int i,j;
   P[0][0]=1;
  for(i=1;i<=N;i++)
     for(j=1;j<=i;j++)
    P[i][j]=P[i-1][j-1] + P[i-1][j]*j;
}
void afiseazaRez()
{
 freopen("mc.out","w",stdout);
 printf("%d\n",P[N][K]);
}
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

PARANTEZĂRI

▸▸▸ENUNȚ:

Fiind dat un număr natural N, să se determine numărul total de secvențe de N perechi de paranteze rotunde închise corect. De exmplu pentru N=2, secvențele corecte sunt (()) și ()(). Deci rezultatul în acest caz este 2.

Date de intrare:
În fişierul de intrare **paran.in** se află un singur număr natural N.

Date de ieşire:
În fişierul de ieşire **paran.out** se află un singur număr, rezultatul cerut.

Exemplu:
paran.in
3

paran.out
5

OBS: Cele cinci secvenţe parantezate corect sunt: ()()(), ()(()), (())(), ((())) şi (()()).

Restricţii:
2< N < 101

⋙SOLUŢIE:

O parantezare de N se exprimă sub forma E = (A)B, unde A şi B sunt alte două parantezări cu suma lungimilor egală cu N-1. De aici soluţia problemei este dată de o binecunoscută relaţie de recurenţă:
$P_n = P_0 * P_{n-1} + P_1 * P_{n-2} + ... + P_{n-1} * P_0$.

⋙PROGRAM:

```
#include<stdio.h>
#define MAXN 101
int N,P[MAXN];
void citesteDate()
{
 freopen("paran.in","r",stdin);
 scanf("%d",&N);
}
void proceseaza()
{
int i,j;
 P[0]=P[1]=1;
 for(i=2;i<=N;i++)
```



```
    for(j=0;j<i;j++)
      P[i]+=P[j]*P[i-j-1];
}
void afiseazaRez()
{
 freopen("paran.out","w",stdout);
 printf("%d\n",P[N]);
}
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

PARANTEZĂRI 2

⋙ENUNŢ:

Fiind date două numere naturale N şi M:
1. Să se determine numărul total de secvenţe de N perechi de paranteze rotunde şi pătrate închise corect.
2. Să se determine şi câte astfel de secvenţe există în cazul în care se poate folosi şi al treilea tip de paranteze, acoladele.
3. Câte secvenţe există folosind M tipuri de paranteze.

De exemplu pentru N=2 şi M=4,
1. secvenţele corecte sunt (()), ()(), [()], []{}, ({}), {}[], [[]], []{}.
   Deci rezultatul în acest caz este 8.
2. Numărul total de secvenţe este 18.
3. Pentru M=4 există 32 de secvenţe corect parantezate.

Date de intrare:
În fişierul de intrare **paran2.in** se află două numere naturale N şi M.

Date de ieşire:
În fişierul de ieşire **paran2.out** se află pe trei linii câte un singur număr, rezultatul cerut la subpunctul respectiv.

Exemplu:

paran2.in	paran2.out
3 4	40
	135
	320

Restricții:

2< N, M < 101

»»»SOLUȚIE:

Soluția este o generalizare a primei probleme pentru cazul în care sunt sunt folosite M tipuri de paranteze. Mai întâi se determină numărul de secvențe pentru cazul M=1 prin relația de recurență:

$P_n = P_0 * P_{n-1} + P_1 * P_{n-2} + \ldots + P_{n-1} * P_0$.

Acum pentru fiecare astfel de secvență putem folosi M tipuri de paranteze pentru fiecare pereche deci se obțin astfel M^N secvențe.

Rezultatul în cazul general este $P_n * M^N$.

»»»PROGRAM:

```
#include<stdio.h>
#define MAXN 101
int M,N,P[MAXN];
void citesteDate()
{
 freopen("paran.in","r",stdin);
 scanf("%d %d",&N,&M);
}
void proceseaza()
{
int i,j;
 P[0]=P[1]=1;
 for(i=2;i<=N;i++)
   for(j=0;j<i;j++)
     P[i]+=P[j]*P[i-j-1];
}
void afiseazaRez()
{int i,R;
 freopen("paran.out","w",stdout);
 R=P[N];
```



```
 for(i=1;i<=N;i++) R=R*2;
 printf("%d\n",R);
 R=P[N];
 for(i=1;i<=N;i++) R=R*3;
 printf("%d\n",R);
 R=P[N];
 for(i=1;i<=N;i++) R=R*M;
 printf("%d\n",R);
}
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

PARANTEZĂRI 3

»»»ENUNȚ:

Se consideră toate secvențele formate din N perechi de paranteze de două tipuri (rotunde şi pătrate) închise corect. Pentru o astfel de secvență E definim următoarele două proprietăți:

a) DR - adâncimea "rotundă" a secvenței, adică nivelul maxim de imbricare al parantezelor rotunde din secvență. Valoarea sa se defineşte astfel:

DR(E) = 1 + DR(E1), dacă E = (E1)
DR(E) = DR(E1), dacă E = [E1]
DR(E) = max{DR(E1),DR(E2)}, dacă E = E1E2

b) DP – adâncimea "pătrată" a secvenței, adică nivelul maxim de imbricare al parantezelor pătrate din secvență. DP se defineşte după cum urmează:

DP(E) = 1 + DP(E1), dacă E = [E1]
DP(E) = DP(E1), dacă E = (E1)
DP(E) = max{DP(E1),DP(E2)}, dacă E = E1E2

Cerință:

Fiind date numerele naturale N, A şi B să se determine numărul total de secvenţe de N perechi de paranteze rotunde şi pătrate care au DR = A şi DP =B.

Date de intrare:
În fişierul de intrare **paran3.in** se află pe o singuă linie trei numere naturale, N, A şi B.

Date de ieşire:
În fişierul de ieşire **paran3.out** se află un singur număr, rezultatul cerut.

Exemplu:

paran3.in	paran3.out
3 1 2	7

OBS:
Cele 7 secvenţe sunt ([[]]), ()[[]], [([])], [()[]], [[()]], [[]()], [[]]().

Restricţii:
1< N ≤ 30
1≤A+B≤N

⋙SOLUŢIE:

Se ia matricea P[i][j][k] – numărul de secvenţe de i perechi de paranteze care au DR≤j şi DP≤k. Forma unei secvenţe este:
E = (E1)E2 sau E = [E1]E2, de aici resultă relaţia de recurenţă:

```
P[i][j][k]= P[0][j-1][k] * P[i-1][j][k] +
            P[1][j-1][k] * P[i-2][j][k] +
            ..............................+
            P[i-1][j-1][k] * P[0][j][k] +

            P[0][j][k-1] * P[i-1][j][k] +
            P[1][j][k-1] * P[i-2][j][k] +
            ..............................+
            P[i-1][j][k-1]*P[0][j][k].
```

Rezultatul va fi P[N][A][B] – (P[N][A-1][B] + P[N][A][B-1] – P[N][A-1][B-1]). Complexitatea algoritmului este $O(N^4)$, la care se mai adaugă şi complexitatea operaţiilor pe numere mari.

⋙PROGRAM:

```
#include<stdio.h>
#define MAXN 31
int REZ,A,B,N,P[MAXN][MAXN][MAXN];
void citesteDate()
{
 freopen("paran3.in","r",stdin);
 scanf("%d %d %d",&N,&A,&B);
}
void proceseaza()
{
int i,j,k,l;
 for(i=0;i<=A;i++)
   for(j=0;j<=B;j++) P[0][i][j]=1;
 for(i=1;i<=N;i++)
   for(j=0;j<=A;j++)
     for(k=0;k<=B;k++)
      for(l=0;l<i;l++)
            {
             if (j) P[i][j][k] += P[l][j-1][k]*P[i-l-1][j][k];
             if (k) P[i][j][k] += P[l][j][k-1]*P[i-l-1][j][k];
            }
REZ = P[N][A][B] - P[N][A-1][B] - P[N][A][B-1] + P[N][A-1][B-1];
}
void afiseazaRez()
{
 freopen("paran3.out","w",stdout);
 printf("%d\n",REZ);
}

int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

PARTIȚII

»»»ENUNȚ:

Se consideră un număr natural N. Să se determine în câte moduri poate fi scris ca sumă de P numere naturale nenule distincte.

Date de intrare:
Pe prima linie a fișierului de intrare part.in se află două numere N și P.

Date de ieșire:
Pe singura linie a fișierului de ieșire part.out se află un număr X, numărul de moduri distincte.

Exemplu:

part.in	part.out
10 3	4

Explicație:
cele patru moduri sunt:
10 = 1 + 2 + 7
10 = 1 + 3 + 6
10 = 1 + 4 + 5
10 = 2 + 3 + 5

Restricții:
$0 < P \leq N \leq 300$

»»»SOLUȚIE:

În rezolvarea problemei se folosește programarea dinamică. Luăm toate șirurile de P numere oarecare ce dau suma N – P(P-1)/2: $x_1 \leq x_2 \leq x_2 \leq \ldots x_{p-1} \leq x_p$. Le considerăm în ordine crescătoarea deoarece ordinea lor nu este relevantă la numărare. Determinăm matricea S[i][j] – în câte moduri se poate scrie i ca sumă de j numere oarecare. Cele j numere trebuie să fie strict pozitive deci se poate da deoparte j din i și mai rămâne să se determine în câte moduri se poate scrie i-j ca sumă de cel mult j numere.

Relația de recurență este:
S[i][j] = S[i-j][1] + S[i-j][2] + ...+S[i-j][j]
Pentru a calcula S[i][j] în O(1) se folosește o matrice A[i][j] = sumă de S[i][k] cu k≤j.

S[i][j] = A[i-j][j]
A[i][j] = A[i][j-1] + S[i][j]

Acum să considerăm șirul 0,1,2,...P-1. Dacă luăm fiecare șir de P numere care dau suma N- P(P-1)/2 și le adunăm membru cu membru obținem:

$0 + X_1,\ 1 + X_2,\ 2 + X_3, \ldots P-1 + X_{p-1},\ P + X_p$ un șir de P numere distincte care însumate dau N.

Rezultatul este dat de S[N-P(P-1)/2][P].
Complexitatea algoritmului este $O(N^2)$.

»»»PROGRAM:

```
#include<stdio.h>
#define MAXN 301
int N,P,S[MAXN][MAXN],A[MAXN][MAXN];
void citesteDate()
{
 freopen("part.in","r",stdin);
 scanf("%d %d",&N,&P);
}
void proceseaza()
{int i,j;
   S[0][0]=1;
   for(i=0;i<=P;i++) A[0][i]=1;
   for(i=1;i<=N;i++)
     for(j=1;j<=P;j++)
    {if (i>=j) S[i][j] = A[i-j][j];
     A[i][j] = A[i][j-1] + S[i][j];
    }
}
void afiseazaRez()
{ freopen("part.out","w",stdout);
 if (2*N>=P*(P-1)) printf("%d\n",S[N - P*(P-1)/2][P]);
 else printf("0");
}
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

PĂTRATE BICOLORE

»»»ENUNȚ:

Plictisiți de munca lor zilnică, Nelu şi Costel se decid să-şi ia liber şi să se relaxeze puțin jucând un joc. Ei iau o cărticică de-a lui Ionuț cu "jocuri logice" şi aleg să joace „Pătrate Bicolore".

Acesta este un joc de două persoane cu două grămezi fiecare având N respectiv M jetoane. Jucătorii mută alternativ, iar o mutare constă în selectarea unei grămezi din care se elimină un număr X de jetoane cu condiția ca X să fie pătrat bicolor. Un număr se numeşte pătrat bicolor dacă este pătrat perfect iar numărul de cifre distincte din componența sa este egal cu 2.

De exmplu „144" este pătrat bicolor. Jocul se încheie atunci când nu se mai pot face mutări iar cel care trebuie să mute pierde.

Cum celor doi le place să concureze, vor să parcurgă R (număr impar) runde de joc. Va fi declarat învingător cel ce va câştiga cel puțin R/2+1 runde. Cel care începe întotdeauna jocul este Nelu deoarece este mai în vârstă.

Cerință:

Sarcina voastră este să determinați două seturi de câte R perechi de grămezi, unul pe care să-l câştige Nelu iar celălalt Costel ştiind că cei doi joacă optim.

Date de intrare:

În fişierul de intrare **nandc.in** se va afla numărul R.

Date de ieşire:

În fişierul de ieşire **nandc.out** se vor afla pe 2*R linii câte două numere Ni Mi, cele două seturi de câte R jocuri, primul fiind cel pe care îl câştigă Nelu.

Exemplu:

NANDC.IN	NANDC.OUT
3	3 2
	5 16
	20 9
	42 39
	15 16
	13 3

Explicație: Nelu câştigă jocurile 2,3,şi 5, iar Costel 1,4,6. Primul set este câştigat de Nelu cu scorul de 2:1 iar celălalt de Costel cu acelaşi scor.

Precizări:

$R \leq 2\,000$

Toate cele 2*R runde trebuie să fie distincte între ele.

»»»SOLUȚIE:

Această problemă se rezolvă foarte uşor folosind programarea dinamică.Inițial se alege valoarea maximă posibilă a unei grămezi, N aproximativ radical din R. Separat se precalculează într-un vector toate pătratele bicolore mai mici decât N.

Se contruieşte o matrice binară de stări S[i][j], unde i şi j sunt dimensiunile celor două grămezi. S[i][j]=1 dacă jucătorul care trebuie să mute are strategie de câştig şi 0 în caz contrar.

Pentru i şi j fixate cu două for-uri, se atribuie inițial S[i][j]=0, apoi se verifică toate mutările disponibile şi dacă se poate atinge o stare necâştigătoare atunci S[i][j] devine 1.

Primul set din soluție va fi format din oricare R perechi i,j, distincte pentru care S[i][j]=1 iar celălalt cu S[i][j]=0.

»»»PROGRAM:

```
#include<stdio.h>
#define MAXN 500
int R,b,S[MAXN][MAXN],sel[10],bic[MAXN];
void citesteDate()
{
 freopen("nandc.in","r",stdin);
 scanf("%d",&R);
}

void afiseazaRez()
{
int i,j,c;
 freopen("nandc.out","w",stdout);
 for(i=0,c=0;i<MAXN && c<R;i++)
 for(j=0;j<=i && c<R;j++)
   if (S[i][j]) printf("%d %d\n",i,j);c++;
```

```
 for(i=0,c=0;i<MAXN && c<R;i++)
 for(j=0;j<=i && c<R;j++)
   if (!S[i][j]) printf("%d %d\n",i,j),c++;
}

void proceseaza()
{
int x,c,i,j,k;
 for(i=1;i*i<MAXN;i++)
    {
     x=i*i;
     c=0;
     for(j=0;j<10;j++) sel[j]=0;
     while(x) {sel[x%10]=1;x/=10;}
     for(j=0;j<10;j++) c+=sel[j];
     if (c==2) bic[++b]=i*i;
    }
 for(i=0;i<MAXN;i++)
    for(j=0;j<=i;j++)
 {
  for(k=1;k<=b && bic[k]<=i;k++)
     if (!S[i-bic[k]][j])
        S[i][j]=1;
  for(k=1;k<=b && bic[k]<=j;k++)
     if (!S[i][j-bic[k]])
        S[i][j]=1;
   S[j][i]=S[i][j];
  }
}

int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```


PUZZLE

»»»ENUNȚ:

Se consideră un dreptunghi de dimensiuni NxM alcătuit din pătrățele de latură 1. Să se determine în câte moduri poate fi acoperit acesta folosind pattern-ul din figură.

Piesele nu se suprapun şi pot fi rotite în orice fel.

Date de intrare:
Pe prima linie a fişierului de intrare **puzzle.in** se află N şi M.

Date de ieşire:
Pe singura linie a fişierului de ieşire **puzzle.out** se află numărul cerut.

Exemplu:

puzzle.in	puzzle.out
2 3	2

Restricții:
$0 < N < 11$
$0 < M < 101$
Rezultatul nu va depăşi valoarea 2^{63}.

»»»SOLUȚIE:

Problema se rezolvă cu ajutorul programării dinamice. Fiecare dreptunghi de dimensiuni Nxl va fi caracterizat prin starea pătrățelelor de pe prima coloană şi numărul de coloane l. Deoarece fiecare pătrățel poate fi acoperit sau poate fi liber vom reprezenta starea acestora printr-un bit.

Definim matricea D[S< 2^N][L] = în câte moduri poate fi acoperit un dreptunghi NxL astfel încât prima coloană să aibă starea S iar restul pătrățelelor de pe celelalte L-1 coloane să fie acoperite.

Vom genera prin backtracking toate posibilitățile de a pune piese pe un dreptunghi Nx2 astfel încât fiecare pătrat să fie acoperit de cel mult o piesă. Vom nota cele două coloane C1 şi C2. şi stările acestora S1 şi S2. Definim S2' (S2 prim) inversul stării S2(pătratele acoperite

devin goale iar cele goale devin acoperite).

Relația de recurentă care se obține este:

D[S1][L] += D[S2'][L-1]

Pentru fiecare pereche de stări S1 şi S2 obținute prin backtracking adaugă D[S2'][L-1] la D[S1][L]. Complexitatea alg este aproximativ $O(N*M*2^N)$.

⋙PROGRAM:

```
#include<stdio.h>
#define MAXS 2000
#define MAXN 11
#define MAXM 100
int N,M,L,bk[MAXN],s1[MAXN],s2[MAXN];
long long D[MAXS][MAXM];
void citesteDate().
{
 freopen("puzzle.in","r",stdin);
 scanf("%d %d",&N,&M);
}
void back(int poz)
{
if (poz<N)
   {
    bk[poz]=0;
    back(poz+1);
    bk[poz]=1;
    if (poz+1<N)
      {bk[poz+1]=3;
       bk[poz+2]=0;
       back(poz+3);
      }
   bk[poz+1]=0;
   back(poz+2);
   bk[poz]=2;
   if (poz+1<N)
     {
      bk[poz+1]=4;
      bk[poz+2]=0;
      back(poz+3);
     }
   bk[poz+1]=0;
```



```
   back(poz+2);
   bk[poz]=3;
   bk[poz+1]=0;
   back(poz+2);
   bk[poz]=4;
   bk[poz+1]=0;
   back(poz+2);
  }
else
{
int i,S1,S2;
 for(i=1;i<=N;i++) s1[i]=0,s2[i]=1;
 for(i=1;i<=N;i++)
   if (bk[i]==1)
      s1[i]=1,s2[i]=s2[i+1]=0;
   else if (bk[i]==2)
      s1[i]=s1[i+1]=1,s2[i]=0;
   else if (bk[i]==3)
      s1[i]=s1[i+1]=1,s2[i+1]=0;
   else if (bk[i]==4)
      s1[i+1]=1,s2[i]=s2[i+1]=0;
 for(S1=S2=0,i=1;i<=N;i++)
    {S1=(S1<<1)+s1[i];
     S2=(S2<<1)+s2[i];
    }
 D[S1][L]+=D[S2][L-1];
 }
}
void proceseaza()
{
 D[0][1]=1;
 for(L=2;L<=M;L++)
   back(1);
}

void afiseazaRez()
{
freopen("puzzle.out","w",stdout);
printf("%lld\n",D[(1<<N)-1][M]);
}
```

```
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

ROATA NOROCULUI

▸▸▸ENUNȚ:

Doi buni amici, Nelu şi Costel, participă la un concurs televizat numit "Roata norocului".

Regulile acestui joc sunt următoarele: Pe un panou este fixată o roată împărțită în 12 sectoare egale de cerc, numerotate de la 1 la 12 în sensul acelor de ceasornic. Fiecare sector are în dreptul său un plic cu o întrebare pentru cei doi concurenți. Deasupra roții se află un ac fix care indică spre sectorul care se află sub el. Cei doi trebuie să răspundă alternativ la întrebări puse de către organizatori. La fiecare rundă, roata se învârte aleator, astfel încât probabilitatea ca acul să indice un anumit sector este 1/12. Dacă sectorul indicat de ac nu conține nici un plic, roata este învârtită în sensul acelor de ceasornic până când este întâlnit un sector cu plic.

Odată fixat sectorul, celor doi li se citeşte întrebarea, după care plicul care o conținea este dat deoparte. Dacă cel care trebuie să răspundă greşeşte, echipa este penalizată cu un punct, iar dacă răspunsul dat este corect, aceştia primesc un punct. Jocul se încheie atunci când fie echipa acumulează 6 puncte, fie când penalizarea totală ajunge la 6. Pentru fiecare întrebare se cunoaşte probabilitatea ca Nelu să dea răspunsul corect, precum şi ceea în care Costel răspunde corect.

Considerând că la prima întrebare pusă va răspunde Costel, să se determine probabilitatea ca echipa N&C să câştige cu o penalizare totală 0,1,2,..5 precum şi probabilitate de a pierde cu scorul 0,1,2,..5. În caz că vor câştiga, cei doi au de gând să doneze suma de bani în scopuri umanitare, ei mulțumindu-se doar cu distracția şi cu câteva halbe de bere.

Determinați şi probabilitatea ca gestul filantropic să fie posibil.

Date de intrare:

Pe fiecare din cele 12 linii ale fişierului **roatan.in** se află câte 2 numere reale, reprezentând probabilitățile ca Nelu, respectiv Costel să dea răspunsul corect la întrebarea i.

Date de ieşire:

Pe primele 6 linii ale fişierului **roatan.out** se află probabilitățile de câştig:

6:0, 6:1, 6:2, 6:3, 6:4, 6:5

Pe următoarele 6 linii se află probabilitățile de a pierde:

0:6, 1:6, 2:6, 3:6, 4:6, 5:6.

Pe ultima linie se află probabilitatea de a se înfăptui actul filantropic.

EXEMPLUL:

roatan.in	roata.out
0.2 0.3	0.00801
0.12 1.0	0.03322
0.99 0.569	0.05775
0.211 0.321	0.10476
0.111 0.23	0.11330
1 1	0.14264
0.12 0.9	0.01274
0.88 0.32	0.04241
0.13 0.64	0.08582
0.664 0.777	0.11577
0.987 0.123	0.14800
0.01 0.0033	0.13557
	0.45969

Precizări: Toate numerele din fişierul de ieşire vor fi tipărite cu o precizie de 5 zecimale.

▸▸▸SOLUȚIE:

Problema se rezolvă cu ajutorul programării dinamice. Fiecare stare posibilă ce poate apărea în decursul jocului este caracterizată de 3 elemente: scorul, penalizarea şi configurația panoului (plicurile rămase).

O configurație a roții o memorăm printr-un vector de 12 biți. Bitul i va fi 1 dacă plicul din dreptul sectorului respectiv a fost folosit, şi 0 în caz contrar. Un astfel de vector îl vom reprezenta printr-o valoare pe 12 biți.

Vom defini matricea:

P[concurent][configRoata][scor][penalizare] = probabilitatea de a se ajunge în această stare, dacă ultima întrebare este pusă concurentului cu numărul **concurent.**

Probabilitatea fiecărei stări din matrice este determinată simulând ultima rundă din cadrul acesteia.

Relația de recurență este:

P[COSTEL][confR][scor][pen] = suma de P[NELU][confR'][scor-1][pen] * C[i] *PR + P[NELU][confR'][scor][pen-1] * (1-C[i])*PR, unde confR' este o posibilă configurație a roții cu o rundă înainte.

Analog în cazul lui Nelu.

»»»PROGRAM:

```
#include<stdio.h>
#define doiLA12 4096
#define COSTEL 0
#define NELU 1
double N[13],C[13],P[2][doiLA12][7][7];
int st[13];
int next(int x){x++;if (x>12) return 1;return x;}
void citesteDate()
{
int i;
 freopen("roatan.in","r",stdin);
 for(i=1;i<=12;i++)
   scanf("%lf %lf",&N[i],&C[i]);
}
void proceseaza()
{
int i,s,p,j,k,l; double PR;
 P[NELU][0][0][0]=1;
 for(i=1;i<doiLA12;i++)
   for(s=0;s<=6;s++)
     for(p=0;p<=6;p++)
     {
      for(j=0;j<12;j++)
```



```
        if ((1<<j)&i) st[j+1]=1;
        else st[j+1]=0;
      for(j=1;j<=12;j++)
       if (st[j])
       {     for(k=1,l=next(j);st[l] && j!=l;l=next(l),k++);
                  PR=double(k)/12;
                   if (p<6 && s)
             {
 P[COSTEL][i][s][p]+=P[NELU][i-(1<<j-1)][s-1][p] * PR * C[j];
 P[NELU][i][s][p]+=P[COSTEL][i-(1<<j-1)][s-1][p] * PR * N[j];
             }
                   if (s<6 && p)
             {
  P[COSTEL][i][s][p]+=P[NELU][i-(1<<j-1)][s][p-1] * PR * (1-C[j]);
  P[NELU][i][s][p]+=P[COSTEL][i-(1<<j-1)][s][p-1] * PR * (1-N[j]);
             }
       }
     }
}
void afiseazaRez()
{
int i,j;double R,F;
 freopen("roatan.out","w",stdout);
 for(i=0,F=0;i<6;i++)
  {R=0;
   for(j=0;j<doiLA12;j++)
     R+=P[NELU][j][6][i] + P[COSTEL][j][6][i];
   printf("%.05lf\n",R);
   F+=R;
  }
 for(i=0;i<doiLA12;i++)
   if (P[COSTEL][i][6][0]>0.0)
     printf("%d %.05lf\n",i,P[NELU][i][6][0]);
 for(i=0;i<6;i++)
  {R=0;
   for(j=0;j<doiLA12;j++)
     R+=P[NELU][j][i][6] + P[COSTEL][j][i][6];
   printf("%.05lf\n",R);
  }
 printf("%.05lf\n",F);
}
```

```
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

ROBOT

▸▸▸ENUNȚ:

Se consideră o tablă de dimensiuni NxM formată din pătrate de latură 1. Unele dintre aceste pătrate pot fi libere, altele pot fi ocupate de obstacole.

Pe tablă se află un robot ghidat prin telecomandă, plasat în pătratul de coordonate X şi Y. În fiecare secundă acesta primeşte câte o comandă care îl face să se deplaseze în unul din cele patru pătrate vecine E,V,N,S, sau să rămână pe loc.

Cerință:

Să se determine câte secvențe distincte de L comenzi deplasează robotul în pătratul de coordonate Xf,Yf, astfel încât pe parcurs acesta să nu se lovească de marginele tablei sau de vreun obstacol.

Date de intrare:

Pe prima linie a fişierului de intrare robot.in se află dimensiunile tablei N şi M.

Pe următoarele N linii se află câte M numere, reprezentând starea pătratelor: 0 – liber, 1-ocupat.

Pe următoarea linie se află cinci numere X, Y, Xf, Yf, L, reprezentând coordonatele inițiale ale robotului, coordonatele finale şi lungimea secvenței de comenzi.

Date de ieşire:

Pe singura linie a fişierului de ieşire robot.out se află numărul de secvențe determinat.

Exemplu:
robot.in

```
10 10
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0 0 0
1 1 4 4 7
```

robot.out

```
70
```

Restricții:

$0 < N,M,L \leq 100$

▸▸▸SOLUȚIE:

Problema se rezolvă folosind programarea dinamică. Fie S[i][j][k] – câte secvențe de lungime k duc robotul în coordonatele i,j. În poziția asta se poate ajunde fie stând pe loc în secunda k, fie venind din una din cele patru direcții. Relația de recurență este:

S[i][j][k]= S[i][j][k-1] + S[i-1][j][k-1] + S[i+1][j][k-1] + S[i][j-1][k-1] + S[i][j+1][k-1].

Evident trebuie să ne asigurăm că robotul nu trece printr-un pătrat cu obstacol, deci S[i][j][k] = 0 pentru pătratele i,j ocupate cu obstacole. Rezultatul este S[Xf][Yf][L]. Complexitatea algoritmului este O(N*M*L*operații pe numere mari).

▸▸▸PROGRAM:

```
#include<stdio.h>
#define MAXN 20
int N,M,L,X,Y,Xf,Yf;
int D[MAXN][MAXN][MAXN],S[MAXN][MAXN];
```

```
void citesteDate()
{int i,j;
 freopen("robot.in","r",stdin);
 scanf("%d %d",&N,&M);
 for(i=1;i<=N;i++)
   for(j=1;j<=M;j++)
     scanf("%d",&S[i][j]);
 scanf("%d %d %d %d %d",&X,&Y,&Xf,&Yf,&L);
 for(i=0;i<=N+1;i++)
   S[i][0]=S[i][M+1]=1;
 for(i=0;i<=M+1;i++)
   S[0][i]=S[N+1][i]=1;
}

void proceseaza()
{int i,j,k;
  D[X][Y][0]=1;
  for(k=1;k<=L;k++)
    for(i=1;i<=N;i++)
   for(j=1;j<=M;j++)
     if (!S[i][j])
       D[i][j][k] = D[i][j][k-1] + D[i-1][j][k-1] + D[i+1][j][k-1] +
       D[i][j-1][k-1] + D[i][j+1][k-1];
}

void afiseazaRez()
{
 freopen("robot.out","w",stdout);
 printf("%d\n",D[Xf][Yf][L]);
}

int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```


SCÂNDURA

»»»ENUNȚ:

Niște hoți au furat dintr-un depozit de cherestea toate scândurile depozitate într-o încăpere nesupravegheată. Pe fiecare scândură hoții trasează niște linii care împart lungimea acesteia în părți mai mici care nu depășesc o anumită valoare N, ce reprezintă lungimea dubiței în care le vor încărca după ce acestea vor fi tăiate. Lungimile pe care le vor avea scândurile rezultate sunt exprimate în metri prin numere naturale. Poliția i-a prins în scurt timp pe hoți dar nu a reușit să recupereze marfa furată. Ei știu de la infractori doar că toate scândurile au fost marcate în mod diferit.

Cerința:

Cunoscând lungimea inițială L pe care o aveau scândurile precum și lungimea N a dubei, determinați numărul maxim de scânduri furate de hoți.

Date de intrare:

În fișierul de intrare **scandura.in** se vor afla pe o singură linie cele două numere L și N cu semnificația din enunț.

Date de ieșire:

În fișierul de ieșire **scandura.out** se va afla un singur număr, rezultatul cerut.

Exemplu:

scandura.in	**scandura.out**
10 4	401

Restricții:

1≤ N≤L≤100000

»»»SOLUȚIE:

Problema se rezolvă prin programare dinamică. Se ia un vector S[i] – în câte moduri se poate marca o scândură de i metri astfel încât

să se respecte restricția de lungime din enunț (să încapă în dubă). Relația de recurență este:

S[i] = S[i-1] + S[i-2] + ...S[i-N]

Pentru a calcula S[i] în timp constant vom utiliza un vector A de sume parțiale ale vectorului S. Astfel:

S[i] = A[i-1] – A[i-N-1]

A[i] = A[i-1] + S[i]

Complexitate algoritmului este O(L) dar vor trebui implementate operațiile de adunare şi scădere pe numere mari. Programul nu implementează şi operațiile cu numere mari, acest lucru rămânând ca execiţiu de antrenament pentru cititori.

›››PROGRAM:

```
#include<stdio.h>
#define MAXL 100001
int N,L,S[MAXL],A[MAXL];
void citesteDate()
{ freopen("scandura.in","r",stdin);
 scanf("%d %d",&L,&N);
}
void proceseaza()
{int i;
 A[0]=S[0]=1;
 for(i=1;i<=L;i++)
   {
    if (i>N) S[i] = A[i-1] - A[i-N-1];
    else S[i]= A[i-1];
    A[i] = A[i-1] + S[i];
   }
}
void afiseazaRez()
{
freopen("scandura.out","w",stdout);
 printf("%d\n",S[L]);
}
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;}
```


SCÂNDURA II

›››ENUNȚ:

Proaspăt ieşiți de la închisoare, hoții de scânduri se reapucă de vechile obiceiuri. Acum şi-au schimbat puțin metoda de marcare a scândurilor furate care împart lungimea acestora în părți mai mici ce nu depăşesc o anumită valoare N, valoare ce reprezintă lungimea dubiței în care le vor încărca după ce acestea vor fi tăiate.

Liniile trasate pe o scândură împart lungimea acesteia în lungimile $x_1,x_2,x_3...$, astfel încât $x_1 \le x_2 \le x_3 \le ...$. Ei nu ştiu că în acest fel vor avea mai puține moduri distincte de marcare, deci mai puține scânduri furate şi mai puțin profit.

Cerința:

Cunoscând lungimea inițială L pe care o aveau scândurile precum şi lungimea N a dubei, determinați numărul maxim de scânduri furate de hoți.

Date de intrare:

În fişierul de intrare **scand2.in** se vor afla pe o singură linie cele două numere L şi N cu semnificația din enunț.

Date de ieşire:

În fişierul de ieşire **scand2.out** se va afla un singur număr, rezultatul cerut.

Exemplu:

scand2.in	scand2.out
10 4	23

Restricții:

$1 \le N \le L \le 1000$

›››SOLUȚIE:

Soluția acestei probleme este asemănătoare cu cea din prima variantă. Se ia o matrice S[i][j] – în câte moduri se poate marca o scândură de i metri astfel încât ultima bucată să aibă lungimea j.

Relația de recurență este următoarea:

S[i][j] = S[i-j][j] + S[i-j][j-1] + ...S[i-j][0]

Pentru a calcula S[i][j] în timp O(1) ne vom folosi de o matrice A[i][j] = numărul de moduri distincte de tăiere a unei scânduri de lungime i astfel încât ultima bucată să aibă lungimea mai mică sau egală cu j. Astfel:

S[i][j] = A[i-j][j]

A[i][j] = A[i][j-1] + S[i][j]

Rezultatul va fi A[L][N].Complexitate algoritmului este O(L*M) + operația de adunare pe numere mari.

»»»PROGRAM:

```
#include<stdio.h>
#define MAXL 1001
int N,L,S[MAXL][MAXL],A[MAXL][MAXL];
void citesteDate()
{ freopen("scand2.in","r",stdin);
 scanf("%d %d",&L,&N);
}
void proceseaza()
{int i,j;
 S[0][0]=1;
 for(i=1;i<=N;i++) A[0][i]=1;
 for(i=1;i<=L;i++)
   for(j=1;j<=N;j++)
     {
      if (i>=j) S[i][j] = A[i-j][j];
      A[i][j]=A[i][j-1] + S[i][j];
     }
}
void afiseazaRez()
{
 freopen("scand2.out","w",stdout);
 printf("%d\n",A[L][N]);
}
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```


SUBŞIR

»»»ENUNȚ:

Mihăiță a învățat la grădiniță un joc nou. El primește de la doamna educatoare două cuvinte (unele foarte lungi şi fără logică) şi trebuie să găsească un alt cuvânt de lungime maximă care să fie subşir comun al celor 2 cuvinte.

Mihăiță e băiat isteț şi găseşte repede un astfel de cuvânt. Nu se poate spune însă acelaşi lucru şi despre tatăl său, căruia fiul său îi dă mari bătăi de cap atunci când îl roagă să găsească şi el un astfel de cuvânt.

Cerință

Acum ca să aveți şi voi ceva bătăi de cap trebuie să determinați probabilitatea ca cei doi să se gândească la acelaşi cuvânt.

Date de intrare

Pe prima linie a fişierului de intrare **subsir.in** se găseşte primul cuvânt, iar pe a doua linie cel de-al doilea cuvânt.

Date de ieşire

Pe prima linie a fişierului de ieşire **subsir.out** se va afişa probabilitatea scrisă sub forma « 1 : X «

Restricții

Lungimea fiecărui şir este mai mică sau egală cu **500.**

Şirurile conțin doar litere mici ale alfabetului englez.

$X < 2^{63}$

Exemplu:

subsir.in	subsir.out
mihai	1:2
maia	

Observație: sunt două cuvinte de lungime maximă **mia** şi **mai,** patru variante posibile de alegere din care 2 sunt cu acelaşi cuvânt, deci probabilitatea este 2:4.

»»»SOLUȚIE:

Cerința problemei are la bază o problemă tipică de programare dinamică şi anume aflarea celui mai lung subşir comun al două şiruri. X-ul căutat este chiar numărul de subşiruri comune de lungime maximă.

În rezolvare se ia o matrice A[i][j] – cel mai lung subşir comun al prefixelor de lungime i, respectiv j ale celor două şiruri.

Relația de recurență este:

$$A[i][j] = \begin{cases} A[i-1][j-1] + 1 \text{ dacă } S1[i]==S2[j] \\ \text{sau} \\ \max\{A[i-1][j], A[i][j-1]\} \text{ dacă } S1[i]!=S2[j] \end{cases}$$

Pentru a număra câte astfel de subşiruri maximale există vom folosi o matrice C[i][j] – câte subşiruri comune maximale există pentru prefixele de lungime i şi j ale şirurilor, astfel încât acestea să se încheie exact pe poziţiile i, respectiv j.

Dacă S1[i]!=S2[j] atunci C[i][j] va avea valoarea 0. Dacă S1[i]==S2[j] atunci vom lua în considerare alte două cazuri:

1. A[i][j]==1 şi atunci C[i][j] va lua valoarea 1;
2. A[i][j]>1.

Dând deoparte ultimul caracter din subşirurile soluție, adică S1[i], vom încerca să căutăm valorile posibile pentru penultimul caracter.

Vom lua toate caracterele de la 'a' la 'z, iar pentru fiecare dintre acestea vom determina ultimele apariții x şi y mai mici decât i respectiv j din cele două şiruri, iar dacă A[x][y]+1==A[i][j] atunci valoarea C[x][y] va fi adăugată la C[i][j].

Pentru a găsi în timp constant ultima apariție a unui caracter într-un prefix de lungime i al unuia dintre cele două şiruri vom precalcula două matrici poz1[c][i], poz2[c][i]. La sfârşitul fiecărui şir vom adăuga acelaşi caracter 'a', iar rezultatul va fi C[N+1][M+1]. Complexitatea algoritmului este O(28*N*M).

»»»PROGRAM:

```
#include<stdio.h>
#include<string.h>
#define MAXN 501
char A[MAXN],B[MAXN];
int N,M,C[MAXN][MAXN],D[MAXN][MAXN];
int poz1[30][MAXN],poz2[30][MAXN];
```



```
void citesteDate()
{
 freopen("subsir.in","r",stdin);
 scanf("%s\n%s\n",&A[1],&B[1]);
 N=strlen(&A[1])+1;M=strlen(&B[1])+1;
 A[N]=B[M]='a';
}
int max(int a,int b)
{
  if (a>b) return a;return b;
}
void proceseaza()
{int i,j,k,x,y;
 for(i=1;i<=N;i++)
   for(j=1;j<=M;j++)
 if (A[i]==B[j])
    D[i][j] = 1+D[i-1][j-1];
 else D[i][j] = max(D[i-1][j],D[i][j-1]);
 for(i='a';i<='z';i++)
   poz1[i-'a'][0]=poz2[i-'a'][0]=-1;
 for(i=1;i<=N;i++)
   {
    for(j='a';j<='z';j++)
  poz1[j-'a'][i]=poz1[j-'a'][i-1];
   poz1[A[i]-'a'][i]=i;
   }
 for(i=1;i<=M;i++)
   {
    for(j='a';j<='z';j++)
  poz2[j-'a'][i]=poz2[j-'a'][i-1];
   poz2[B[i]-'a'][i]=i;
   }
C[0][0]=1;
for(i=1;i<=N;i++)
  for(j=1;j<=M;j++)
if (A[i]==B[j])
 {
  for(k='a';k<='z';k++)
    {x=poz1[k-'a'][i-1];
     y=poz2[k-'a'][j-1];
     if (x!=-1 && y!=-1)
```

```
            if (D[x][y]+1==D[i][j])
              C[i][j]+=C[x][y];
        }
      if (D[i][j]==1) C[i][j]=1;
    }
}
void afiseazaRez()
{
 freopen("subsir.out","w",stdout);
 printf("1 : %d\n",C[N][M]);
}
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

SUBŞIR 2

»»»ENUNŢ:

Se consideră un şir de caractere format din litere mici ale alfabetului englez având lungimea N. Să se determine un alt şir cu lungimea cuprinsă între L şi U care nu este subşir al şirului considerat. Dacă există mai multe variante se va alege prima în ordine lexicografică. În cazul în care nu există soluţie se va afişa "imposibil".

Date de intrare:

Pe prima linie a fişierului de intrare **subsir2.in** se vor afla trei numere N, L şi U, lungimea şirului dat, limita inferioară şi limita superioară pentru X.

Pe cea de-a doua linie se va găsi un şir format din N caractere, litere mici din alfabetul englez.

Date de ieşire:

Pe prima linie a fişierului de ieşire **subsir2.out** se va afla X, lungimea şirului găsit.

A doua linie va conţine şirul de lungime X cerut în enunţ.

Exemplu:

subsir2.in	**subsir2.out**
23 1 6	4
acestaestedoarunexemplu	aaaa

Restricţii şi precizări:

$0 < L \le X \le U \le N < 1001$

»»»SOLUŢIE:

Problema se rezolvă folosind programarea dinamică. Pentru fiecare j($1 \le j \le N$) vom număra cîte subşiruri distincte de lungime j există în şirul dat. De fapt, în soluţie nu ne interesează exact numărul lor, ci doar dacă se pot forma sau nu toate cele 26^j şiruri. La pasul următor vom căuta toate lungimile x cuprinse între L şi U pentru care nu se pot forma toate subşirurile şi vom determina minimul lexicografic pentru fiecare. Ca soluţie vom alege primul în ordine lexicografică dintre aceste şiruri.

S[i][j] = 1 dacă toate cele 26^j subşiruri distincte de lungime j se pot obţine cu primele i caractere din şirul dat, sau 0 în caz contrar.

Relaţia de recurenţă este:

$$S[i][j] = \begin{cases} 1 \quad \text{dacă} \quad S[poz['a'][i]-1][j-1] == 1 \\ \qquad \text{şi } S[poz['b'][i-1][j-1] == 1 \text{ şi } ... \\ \qquad \text{şi } S[poz['z'][i]-1][j-1] == 1, \\ \qquad \text{sau} \\ 0 \quad \text{dacă cel puţin una din cele 26 de condiţii nu este indeplinită.} \end{cases}$$

poz[c][i] = poziţia ultimei apariţii a caracterului c în prefixul de lungime i al şirului dat.

Matricea S se completează în complexitate $O(26 * N^2)$.

Acum, pentru o lungime x, vom determina primul subşir lexicografic care nu apare în şir astfel:

Pornim cu şirul vid din S[N][x] care trebuie să fie 0, apoi încercăm să fixăm următorul caracter c astfel încât S[poz[c][N]-1][x-1] să

fie tot 0, după care N devine poz[c][N] -1 şi îl decrementăm pe x.

Repetăm procesul până ajungem cu x egal cu 0. Se observă că în acest fel determinăm primul subşir în ordine lexicografică citit de la dreapta către stânga. Pentru a obţine un subşir corect, la începutul programului vom inversa şirul dat.

»»»PROGRAM:

```
#include<stdio.h>
#include<string.h>
#define MAXN 1001
int X,N,L,U,poz[30][MAXN],S[MAXN][MAXN];
char sir[MAXN],rez[MAXN],r[MAXN];
void citesteDate()
{int i,j;char a;
 freopen("subsir2.in","r",stdin);
 scanf("%d %d %d\n",&N,&L,&U);
 scanf("%s",sir);
 for(i=0,j=strlen(sir)-1;i<j;i++,j--)
    a=sir[i],sir[i]=sir[j],sir[j]=a;
}
void subsir(int l,char m[])
{int i,j,k,c,g;
  for(i=N,j=l,k=0;j>0;k++,j--)
    {
     for(c='a',g=1;c<='z' && g;c++)
   if (poz[c-'a'][i]<=0)
      m[k]=c,g=0;
  else
    if (!S[poz[c-'a'][i]-1][j-1])
      {g=0;m[k]=c;i=poz[c-'a'][i]-1;}
    }
}
void proceseaza()
{int i,j,k;
  for(i='a';i<='z';i++)
    poz[i-'a'][0]=-MAXN;
  for(i=1;i<=N;i++)
    {
     for(j='a';j<='z';j++)
   poz[j-'a'][i]=poz[j-'a'][i-1];
     poz[sir[i-1]-'a'][i]=i;
    }
  for(i=0;i<=N;i++) S[i][0]=1;
  for(i=1;i<=N;i++)
    for(j=1;j<=i;j++)
  {
   S[i][j]=1;
   for(k='a';k<='z' && S[i][j];k++)
     if (poz[k-'a'][i]>0)
       {if (S[poz[k-'a'][i]-1][j-1]==0)
            S[i][j]=0;
       }
     else S[i][j]=0;
  }
 for(i=L;i<=U;i++)
  if (!S[N][i])
    if (X==0)
  {   X=i;
   subsir(i,rez);
   printf("%d = %s\n",i,rez);
  }
    else
  {subsir(i,r);
   printf("%d = %s\n",i,r);
   if (strncmp(r,rez,X)<0)
     {strcpy(rez,r);
      X=i;
     }
  }
}
void afiseazaRez()
{ freopen("subsir2.out","w",stdout);
 if (X)
  printf("%d\n%s\n",X,rez);
 else printf("imposibil");
}
int main()
{ citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

SUBŞIR 3

»»»ENUNŢ:

Se consideră un şir de lungime N, format doar din litere mici ale alfabetului englez. Să se determine cel mai scurt şir de caractere care nu este subşir al şirului iniţial. Dacă există mai multe soluţii, se va alege primul în ordine lexicografică.

Date de intrare:
Pe prima linie a fişierului de intrare subsir3.in se găseşte un singur număr N, lungimea şirului. Pe cea de-a doua linie se va afla şirul.

Date de ieşire:
Pe prima linie a fişierului de ieşire subsir3.out se va afla X, lungimea minimă. A doua linie va conţine şirul cerut.

Exemplu:

subsir3.in	subsir3.out
20	1
oproblemainteresanta	c

Restricţii:
0 < N < 1000001

»»»SOLUŢIE:

Problema nu este deloc dificilă chiar dacă valorile destul de mari ale lui N creează impresia asta.

Soluţia foloseşte programarea dinamică. Se defineşte vectorul S[i] - cel mai scurt şir care nu este subşir pentru prefixul de lungime i.

Relaţia de recurenţă este:
poz[c][i] = poziţia ultimei apariţii a caracterului c în prefixul de lungime i.
poz[c][i] = -1 dacă c nu apare în prefixul de lungime i.
poz[c][i] = max { poz[c][i-1], i - dacă V[i]==c}
S[i] = 1 dacă există un caracter 'a' ≤ c ≤ 'z' astfel încât poz[c][i]=-1.
Sau S[i] = min {S[poz[x][i]-1] +1 }, 'a' ≤ x ≤ 'z'.
Vectorul S se determină în O(N).

»»»PROGRAM:

```
#include<stdio.h>
#include<string.h>
#define MAXN 1000001
#define INF 1000000000
int S[MAXN],N,poz[26][MAXN]; char sir[MAXN],rez[MAXN];
void citesteDate()
{
int i,j;char a;
 freopen("subsir3.in","r",stdin);
 scanf("%d\n",&N);
 scanf("%s",sir);
 for(i=0,j=strlen(sir)-1;i<j;i++,j--)
   a=sir[i],sir[i]=sir[j],sir[j]=a;
}
void proceseaza()
{
  int i,j,k;
  for(i='a';i<='z';i++) poz[i-'a'][0]=-INF;
  for(i=1;i<=N;i++)
    {for(j='a';j<='z';j++) poz[j-'a'][i]=poz[j-'a'][i-1];
     poz[sir[i-1]-'a'][i]=i;
    }
   S[0]=1;
   for(i=1;i<=N;i++)
     {
      S[i]=INF;
      for(j='a';j<='z';j++)
    if (poz[j-'a'][i]<0)
       S[i]=1;
    else if (S[poz[j-'a'][i]-1]+1<S[i])
              S[i] = S[poz[j-'a'][i]-1]+1;
     }
 for(i=N,j=0;j<S[N];j++)
   for(k='a';k<='z';k++)
     if (poz[k-'a'][i]<0)
    rez[j]=k,k='z';
      else if (S[poz[k-'a'][i]-1]+1==S[i])
        rez[j]=k,i=poz[k-'a'][i]-1,k='z';
}
```

```
void afiseazaRez()
{
 freopen("subsir3.out","w",stdout);
 printf("%d\n%s\n",S[N],rez);
}
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

SUBŞIR 4

»»»ENUNŢ:

Se dă un şir format din N litere mici aparţinând alfabetului englez. Se consideră toate subşirurile distincte ale acestuia, aranjate în ordine lexicografică. Să se determine cel de-al K-ulea subşir în această ordine.

Date de intrare:

Pe prima linie a fişierului de intrare **subsir4.in** se află N, lungimea şirului. Pe cea de-a doua linie se află şirul. Pe ultima linie se află număr K.

Date de ieşire:

Pe singura linie a fişierului de ieşire **subsir4.out** se află al K-ulea subşir în ordine lexicografică.

Exemplu:

subsir4.in	subsir4.out
5	po
prost	
6	

Explicaţie: cele 31 de subşiruri distincte în ordine lexicografică sunt: o, os, ost, ot, p, po, pos, post, pot, pr, pro, pros, prost, prot, prs, prst, prt, ps, pst, pt, r, ro, ros, rost, rot, rs, rst, rt, s, st, t.

Restricţii:

$0 < N < 1000001$

$0 < K < 8000000000000000001$

Numărul total de subşiruri distincte nu va depăşi 2^{63}.

»»»SOLUŢIE:

Pentru a rezolva problema folosim programarea dinamică. Vom defini un vector S[i] - câte subşiruri distincte, inclusiv şirul vid, se pot forma din primele i caractere din şirul iniţial. Vom mai utiliza o matrice poz[c][i] - poziţia ultimei apariţii a caracterului c în prefixul de lungime i. Acest vector îl vom procesa pe parcus, în acelaşi timp cu S. Relaţia de recurenţă pentru S este:

S[i] = 1 + S[poz['a'][i]-1] + S[poz['b'][i]-1] + S[poz['c'][i]-1] +... + S[poz['z'][i]-1]

În construcţia celui de-al K-ulea subşir, plecăm de la şirul vid şi repetăm următorii paşi:

Parcurgem caracterele 'a'..'z' cât timp K>S[poz[c][N]-1] şi scădem S[poz[c][N]-1] din K. Caracterul la care ne oprim va fi cel pe care îl vom adăuga la şirul soluţie. Odată fixat acest caracter, decrementăm K iar N devine poz[x][N]-1. Aplicăm acelaşi algoritm cât timp K>0. La final, şirul obţinut este cel de-al K-ulea subşir.

Deoarece algoritmul descris găseşte cel de-al K-ulea subşir lexicografic citit de la dreapta spre stânga, vom inversa şirul iniţial la începutul programului. Complexitatea algoritmului este O(26*N).

»»»PROGRAM:

```
#include<stdio.h>
#define MAXN 1000001
#define INF 1000000000
int N,poz[26][MAXN];
long long K,S[MAXN];
char sir[MAXN],rez[MAXN];
void citesteDate()
{int i,j;char a;
 freopen("subsir4.in","r",stdin);
 scanf("%d\n%s\n%lld",&N,sir,&K);
 for(i=0,j=N-1;i<j;i++,j--)     a=sir[i],sir[i]=sir[j],sir[j]=a;
}
void proceseaza()
{int i,j;
```

```
 for(i='a';i<='z';i++) poz[i-'a'][0]=-INF;
 S[0]=1;
 for(i=1;i<=N;i++)
  {
   for(j='a';j<='z';j++) poz[j-'a'][i]=poz[j-'a'][i-1];
   poz[sir[i-1]-'a'][i]=i;
   S[i]=1;
   for(j='a';j<='z';j++)
         if (poz[j-'a'][i]>0)  S[i]+=S[poz[j-'a'][i]-1];
  }
 for(i=0;K;i++)
   for(j='a';j<='z';j++)
  if (poz[j-'a'][N]>0)
   if (S[poz[j-'a'][N]-1]<K)   K-=S[poz[j-'a'][N]-1];
   else {rez[i]=j;K--;N=poz[j-'a'][N]-1;j='z';}
}
void afiseazaRez()
{
 freopen("subsir4.out","w",stdout);
 printf("%s\n",rez);
}
int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```

VOPSIREA BILELOR

⋙ENUNȚ:

Nicoleta aşează N bile albe în linie şi doreşte să vopsească în negru câteva dintre ele, astfel încât printre oricare M bile consecutive să se găsească cel puțin două bile negre. Nicoleta ştie de la tatăl ei că pentru a vopsi bila i îi sunt necesari Ci mililitri de vopsea.

Cerință:

Sarcina voastră este să o ajutați pe Nicoleta să calculeze cantitatea minimă de vopsea necesară vopsirii bilelor.

Date de intrare:

Pe prima linie se găsesc două numere întregi N şi M (2≤N≤10000, 2≤M≤100, M≤N). A doua linie conține N numere întregi C1, C2, ..., CN (1≤Ci≤10000).

Date de ieşire:

În fişierul de ieşire se va găsi un singur număr reprezentând cantitatea minimă de vopsea necesară.

Exemplu:

bile.in	bile.out
6 3	9
1 5 6 2 1 3	

Explicație:

În exemplu trebuie vopsite bilele cu numerele: 1, 2, 4, 5.

⋙SOLUȚIE:

În primul rând se constată faptul că problema poate fi rezolvată cu ajutorul programării dinamice, deoarece subproblemele depind unele de celelalte (nu sunt independente).

Condiția din enunț poate fi exprimată şi în alt mod mai convenabil: dinstanța dintre oricare două bile negre între care se mai află doar o singură bilă neagră nu trebuie să depăşească M.

În caz contrar între ele ar exista un interval de lungime M care să conțină doar o bilă neagră şi astfel nu s-ar respecta cerința problemei.

Se defineşte matricea A[i][j] – cantitatea minimă de vopsea necesară primelor i bile astfel încât ultimele 2 bile vopsite în negru să fie i şi i-j.

Vom folosi încă o matrice B[i][j] în care vom reține minimul dintre {A[i][k],k≤j}.

Relația de recurență dintre subprobleme este:

A[i][j] = C[i] + B[i-j,M-j] 1≤i≤N, 1≤j<M, 1≤j≤i
B[i][j] = min{ B[i][j-1], A[i][j] } 1≤i≤N, 1≤j<M,
Rezultatul problemei este B[N+1][M-1], luând C[i+1]=0.

▸▸▸PROGRAM:

```
#include<stdio.h>
#define MAXN 10000
#define MAXM 100
#define INF 1000000000
int N,M,C[MAXN],A[MAXN][MAXM],B[MAXN][MAXM];
void citesteDate()
{int i;
 freopen("bile.in","r",stdin);
 scanf("%d %d",&N,&M);
 for(i=1;i<=N;i++) scanf("%d",&C[i]);
}
void proceseaza()
{int i,j;
  for(i=1;i<=N+1;i++)
     for(j=0;j<M;j++)
   B[i][j]=A[i][j]=INF;
 for(i=1;i<=N+1;i++)
    {
     for(j=1;j<=i && j<M;j++)
      A[i][j]=C[i]+B[i-j][M-j];
     for(j=1;j<M;j++)
      if (A[i][j]<B[i][j-1]) B[i][j]=A[i][j];
      else B[i][j]=B[i][j-1];
    }
}
void afiseazaRez()
{
 freopen("bile.out","w",stdout);
 printf("%d\n",B[N+1][M-1]);
}

int main()
{
 citesteDate();
 proceseaza();
 afiseazaRez();
 return 0;
}
```


PROBLEME PROPUSE

PB1. Un număr de hoți ajung la un tezaur cu m seifuri, în fiecare seif existând un anumit număr de bani. Hoții vor lua banii dintr-un număr de seifuri consecutive ce respectă proprietatea: suma banilor din aceste seifuri să se împartă exact la cei n hoți și să fie cea mai mare posibilă de câștigat. Spuneți pe care dintre seifuri le vor deschide hoții. Dacă problema nu are soluție, se va afișa mesajul „Imposibil".

PB2. Se consideră o mulțime ordonată de șiruri de `N` biți. Mulțimea conține toate șirurile posibile de `N` biți care au cel mult `L` biți de `1`. Se cere să se tipărească al `C`-ulea șir de biți din această mulțime, în ordinea dată de numerele în baza `10` ce corespund șirurilor de biți.

PB3. Se consideră mulțimea formată din numerele naturale mai mici sau egale decât `N` (`N≤ 39`). O astfel de mulțime se poate partiționa în două submulțimi care au aceeași sumă. De exemplu, dacă `N=3`, mulțimea `{1,2,3}` se poate împărți în `{1,2}` și `{3}`. Se cere să se calculeze numărul de astfel de partiționări știind că nu are importanță ordinea mulțimilor dintr-o soluție (`{1,2}` și `{3}` reprezintă aceeași soluție ca `{3}` și `{1,2}`).

PB4. Fie o secvență de 0 și 1 de lungime n și un set de m secvențe de 0 și 1. Să se găsească o descompunere a șirului dat într-un număr minim de secvențe din cele m.

PB5. Fie un șir de n numere naturale. Să se afișeze un subșir de lungime maximă în care fiecare număr să aibă în reprezentarea în baza doi un număr de cifre mare decât numărul de cifre 1.

PB6. Fie un vector de numere de lungime n de numere naturale și două numere naturale a și b. Să se determine dacă se poate trece din a în b folosind următoarele operații permise:

- Se adună cu a oricâte numere din v.
- Se scad din a oricâte numere din v.

Fiecare număr poate fi adunat sau scăzut de mai multe ori.

PB7. Fie un șir de n stive de monede. Fiecare stivă i are inițial a(i) monede. Singura operație permisă este mutarea monedei de pe o stivă pe alta. Printr-un număr minim de mutări să se treacă într-o configurație în care diferența dintre numărul de monede dintre două stive consecutive să fie 1 sau -1.

PB8. Se dă o frază din cuvintele căreia s-au pierdut spațiile. Să se reconstituie fraza pe baza unui dicționar de cuvinte. În cazul în care există mai multe soluții să se afișeze cea cu număr minim de cuvinte.

BIBLIOGRAFIE

INFORMATICĂ PENTRU GRUPELE DE PERFORMANȚĂ –Lector Univ. Dr. Clara Ionescu, Prof. Adina Balan

MANUAL C++ PENTRU LICEE-Dorian Stoilescu

INIȚIERE ÎN LIMBAJUL C-Damian Costea

PROBLEME DE INFORMATICĂ-Radu Berinde, Dan Ghinea, Horia Andrei Ciochină,Cornel Margine, Prof. Dr. Adrian Atanasiu

CONCURSUL .CAMPION

C++ MANUAL COMPLET-Herbert Schildt

GINFO

AGORITMI FUNDAMENTALI O PERSPECTIVĂ C++- Răzvan Andone, Ilie Gârbacea