Arbori indexați binar

Capitolul 22

- **Preliminarii**
- **Cazul unidimensional**
- **Cazul bidimensional**
- **Cazul tridimensional**
- **Cazul *n*-dimensional**
- **Rezumat**
- **Implementări sugerate**
- **Probleme propuse**
- **Soluțiile problemelor**

Problema determinării sumei elementelor unei subsecvențe a unui şir ale cărui valori se modifică în timp real apare destul de des în diferite aplicații. Ea a apărut, sub diverse forme şi la anumite concursuri de programare. În cadrul acestui capitol vom prezenta o structură de date care poate fi folosită pentru rezolvarea eficientă a acestei probleme.

22.1. Preliminarii

În cadrul acestei secțiuni vom prezenta câteva noțiuni introductive care sunt necesare pentru a înțelege modul în care acționează arborii indexați binar şi pentru a enunța problema determinării sumei elementelor unei subsecvențe a unui şir ale cărui valori se modifică în timp real.

22.1.1. Subsecvențe

Prin subsecvență înțelegem un subşir ale cărui elemente se află pe poziții consecutive în şirul inițial. De exemplu, (2, 3, 4) este o subsecvență a şirului (1, 2, 3, 4, 5) formată din al doilea, al treilea şi al patrulea element al şirului, în timp ce (1, 2, 4) nu este o subsecvență deoarece nu este formată din elemente aflate pe poziții consecutive.

Vom identifica o subsecvență prin extremitățile sale (poziția cea mai din stânga şi poziția cea mai din dreapta din şir). O subsecvență care începe în poziția *a* şi se termină în poziția *b* va fi notată prin <*a*, *b*>. Pentru şirul (1, 2, 3, 4, 5) subsecvența (2, 3, 4) va fi notată prin <2, 4> dacă numerotarea elementelor începe cu 1 sau prin <1, 3> dacă numerotarea începe de la 0.

Deseori, avem nevoie de informaţii referitoare la subsecvenţele unui şir cum ar fi suma elementelor, produsul lor, valoarea minimă, valoarea maximă etc. La prima vedere, rezolvarea acestei probleme pare destul de simplă (de exemplu, pentru sumă se parcurg elementele subsecvenţei şi se adună). Totuşi, acest algoritm este ineficient datorită faptului că necesită parcurgerea întregii subsecvenţe. Vom arăta că există algoritmi pentru care o astfel de parcurgere nu este necesară.

Dacă am efectua o singură dată sau de un număr limitat de ori o astfel de parcurgere, algoritmul ar putea părea performant, viteza de execuţie fiind relativ mare. Problema se complică dacă elementele şirului se modifică în timp real.

22.1.2. Modificări în timp real

Să presupunem că există două tipuri de operaţii care pot fi efectuate asupra unui şir. Primul tip constă în modificarea valorii unui element, în timp ce al doilea reprezintă interogări (cereri de informaţii) referitoare la anumite subsecvenţe ale şirului. De obicei, cele două tipuri de operaţii sunt "amestecate" în sensul că nu vor fi doar modificări urmate din interogări, ci putem avea o modificare, urmată de două interogări, urmate de cinci modificări, urmate de alte două interogări etc.

Aşadar, putem spune că elementele şirului se modifică în timp real şi interogările se referă la starea curentă a şirului (cea din momentul cererii de informaţii).

22.1.3. Enunţul problemei

În cele ce urmează vom prezenta un enunţ al problemei, particularizat pentru cazul în care interogările se referă la suma elementelor subsecvenţelor.

Se consideră un şir de numere întregi care are toate elementele nule. O modificare a unui element constă în adunarea unei valori la valoarea curentă (pot fi realizate şi scăderi care au forma adunării unor valori negative). Pe parcursul modificărilor pot apărea interogări referitoare la suma elementelor unei subsecvenţe a şirului.

Problema poate fi enunţată în multe alte forme. Una dintre ele ar putea fi următoarea:

Un furnizor lucrează cu N distribuitori; în fiecare moment distribuitorii pot efectua plăţi către furnizor sau pot cumpăra produse de la acesta. De asemenea, în fiecare moment furnizorul poate cere informaţii referitoare la suma totală a datoriilor pe care le au magazinele cu numere de ordine cuprinse între două valori date.

22.1.4. Un exemplu

Vom exemplifica acum evoluţia în timp real a unui şir format din cinci elemente, prezentând valorile şirului după fiecare modificare şi rezultatul fiecărei interogări.

Operația `Inițializare` constă în setarea la 0 a valorilor tuturor elementelor. Operația `Adună(i,x)` constă în adunarea valorii *x* la valoarea curentă a celui de-al *i*-lea element. Operația `Sumă(a,b)` furnizează suma elementelor subsecvenței <*a*, *b*>.

Operație	**Șir/Rezultat**
Inițializare	0 0 0 0 0
Adună(1,3)	3 0 0 0 0
Adună(4,5)	3 0 0 5 0
Sumă(3,3)	0
Sumă(4,4)	5
Adună(4,2)	3 0 0 7 0
Adună(2,3)	3 3 0 7 0
Sumă(2,5)	10
Sumă(2,4)	10
Adună(5,2)	3 3 0 7 2
Sumă(2,4)	10
Sumă(2,5)	12
Adună(3,1)	3 3 1 7 2
Adună(4,-2)	3 3 1 5 2
Sumă(2,5)	11
Adună(1,-3)	0 3 1 5 2
Adună(5,5)	0 3 1 5 7
Sumă(1,2)	3
Sumă(3,4)	6
Adună(2,-1)	0 2 1 5 7
Adună(3,3)	0 2 4 5 7
Sumă(5,5)	7
Sumă(1,2)	2
Sumă(1,5)	18

22.2. Cazul unidimensional

Din modul în care a fost enunțată problema, rezultă că operațiile sunt efectuate asupra unui tablou unidimensional; aşadar, acesta este cazul unidimensional al problemei. Vom prezenta în continuare trei algoritmi care pot fi folosiți pentru rezolvarea problemei şi apoi le vom studia performanțele.

22.2.1. Algoritmul ineficient

Cea mai intuitivă metodă de rezolvare constă în păstrarea unui vector cu valorile şirului, modificarea lor atunci când este necesar şi calcularea sumelor în momentul în care apar interogări.

Pentru a prezenta algoritmul vom considera că o modificare este codificată prin valoarea 1, iar o interogare prin valoarea 2. Ar fi necesară o a treia operație (codificată

prin 3) care ar indica faptul că nu mai există modificări sau interogări, deci execuția poate fi încheiată.

Versiunea în pseudocod este prezentată în continuare:

```
Algoritm ModificareIneficientă:
                                                    { inițializări }
  scrie 'Introduceți numărul de elemente: '
  citește N                                         { dimensiunea şirului }
  pentru i ← 1, N execută:
    a_i ← 0                                         { valorile inițiale sunt nule }
  sfârşit pentru
  scrie Introduceți codul operației:
  citește cod
  cât timp cod ≠ 3 execută:
    dacă cod = 1                                    { modificare }
      atunci
        scrie 'Introduceți indicele elementului modificat: '
        citește ind
        scrie 'Introduceți valoarea care va fi adunată
               (valoare negativă pentru scăderi): '
        citește val
        a_ind ← a_ind + val
      altfel                                        { interogare }
        scrie 'Introduceți extremitățile subsecvenței: '
        citește st, dr
        suma ← 0
        pentru i ← st, dr execută:
          suma ← suma + a_i
        sfârşit pentru
        scrie 'Suma elementelor secvenței este ', suma
    sfârşit dacă
    scrie Introduceți codul operației:
    citește cod
  sfârşit cât timp
sfârşit algoritm
```

Se observă că modificările se efectuează în timp constant deoarece implică doar accesarea unui element al vectorului şi modificarea valorii sale.

Datorită faptului că pentru calcularea sumei elementelor unei subsecvențe este necesară parcurgerea tuturor elementelor subsecvenței, această operație are ordinul de complexitate O(l), unde l este lungimea subsecvenței.

22.2.2. Vector de sume

Prima încercare de optimizare constă în găsirea unui algoritm mai rapid pentru calcularea sumei elementelor unei subsecvențe. O posibilitate relativ simplă este păstrarea unui vector b a cărui elemente b_i reprezintă suma primelor elemente ale şirului a. Pentru a găsi suma elementelor unei subsecvențe <*st*, *dr*> vom efectua o simplă diferență: $b_{dr} - b_{st-1}$.

Pentru a calcula sumele pentru subsecvențe de forma <1, *dr*> vom avea nevoie de elementul b_0 care va avea întotdeauna valoarea 0. Astfel, pentru o subsecvență de această formă suma elementelor va fi $b_{dr} - b_0 = b_{dr}$.

Aşadar, folosind acest artificiu, ordinul de complexitate al unei interogări va fi $O(1)$ pentru că este necesară doar o simplă scădere pentru furnizarea rezultatului. Din nefericire, în momentul efectuării unei modificări pentru elementul i, toate elementele b_j pentru care $j \geq i$ trebuie să îşi modifice valoarea.

Ca urmare, operația de modificare a celui de-al i-lea element nu se mai realizează în timp constant, deoarece trebuie modificate $N - i + 1$ valori. Aşadar, ordinul de complexitate devine liniar.

Astfel, am reuşit să înlocuim algoritmul liniar de determinare a sumei elementelor unei subsecvențe cu un algoritm având ordinul de complexitate $O(1)$ cu prețul creşterii timpului de execuție a algoritmului de modificare de la unul constant la unul liniar.

Se observă că nu mai avem nevoie de şirul a folosit pentru algoritmul anterior, fiind suficientă păstrarea valorilor elementelor şirului b. Prezentăm versiunea în pseudocod a acestui algoritm, folosind aceleaşi coduri pentru operațiile efectuate:

```
Algoritm ModificareVectorDeSume:
  scrie 'Introduceți numărul de elemente: '
  citeşte N                                        { dimensiunea şirului }
  pentru i ← 0, N execută:
    bi ← 0                                         { valorile inițiale sunt nule }
  sfârşit pentru
  scrie 'Introduceți codul operației: '
  citeşte cod
  cât timp cod ≠ 3 execută:
    dacă cod = 1                                   { modificare }
      atunci
        scrie 'Introduceți indicele elementului modificat: '
        citeşte ind
        scrie 'Introduceți valoarea care va fi adunată
               (valoare negativă pentru scăderi): '
        citeşte val
        pentru i ← ind, N execută:
          bi ← bi + val
        sfârşit pentru
```

```
      altfel                                                { interogare }
        scrie 'Introduceţi extremităţile subsecvenţei: '
        citeşte st, dr
        scrie 'Suma elementelor secvenţei este ', b_dr - b_st-1
    sfârşit dacă
    scrie 'Introduceţi codul operaţiei: '
    citeşte cod
  sfârşit cât timp
sfârşit aalgoritm
```

22.2.3. Arbori indexaţi binar

Practic, pentru algoritmii anteriori una dintre cele două operaţii se efectuează în timp constant, în timp ce cealaltă se efectuează în timp liniar. Aşadar, dacă numărul de modificări este aproximativ egal cu cel al interogărilor, timpul de execuţie va fi aproximativ acelaşi.

Dacă numărul modificărilor este mult mai mare decât cel al interogărilor, atunci este preferabilă folosirea algoritmului naiv.

Dacă, dimpotrivă, numărul interogărilor este mult mai mare decât cel al modificărilor, atunci algoritmul bazat pe vectorul de sume este mai performant. Totuşi, în cazul mediu, ambii algoritmi sunt liniari.

În cele ce urmează vom prezenta o structură de date care permite efectuarea în timp logaritmic atât a modificărilor, cât şi a interogărilor. Structura de date pe care o propunem este numită *arbore indexat binar*.

Aşadar, vom avea o structură arborescentă care va permite efectuarea interogărilor în timp logaritmic în condiţiile în care şi modificările se efectuează în timp logaritmic. Pentru a folosi această structură de date, va trebui să considerăm că elementele şirului sunt numerotate începând cu 1.

Arborele va fi păstrat sub forma unui vector c în care fiecare element i va conţine suma elementelor subsecvenţei $<i - 2^k + 1, i>$, unde k este numărul zerourilor terminale din reprezentarea binară a lui i.

Aşadar, elementele de pe poziţiile impare ale arborelui vor păstra suma elementelor unor subsecvenţe formate dintr-un singur element (aflat pe o poziţie impară în şirul a). În elementele de pe poziţiile de forma $4 \cdot k + 2$ (un zero terminal în reprezentarea binară a poziţiei) vom păstra suma elementelor unor subsecvenţe formate din două elemente. În elementele de pe poziţiile de forma $8 \cdot k + 4$ (două zerouri terminale în reprezentarea binară a poziţiei) vom păstra suma elementelor unor subsecvenţe formate din patru elemente.

În general, dacă reprezentarea binară a poziţiilor au p zerouri terminale, atunci elementele din arbore vor păstra sume ale elementelor unor subsecvenţe cu 2^p elemente.

Vom considera că, la un moment dat, şirul (format din 15 elemente) este (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15).

Valorile vectorului *c* (cel care reprezintă arborele indexat binar) sunt:

Element	**Poziție**	**Semnificație**	**Valoare**
c_1	0001	Sumă(1, 1)	1
c_2	0010	Sumă(1, 2)	3
c_3	0011	Sumă(3, 3)	3
c_4	0100	Sumă(1, 4)	10
c_5	0101	Sumă(5, 5)	5
c_6	0110	Sumă(5, 6)	11
c_7	0111	Sumă(7, 7)	7
c_8	1000	Sumă(1, 8)	36
c_9	1001	Sumă(9, 9)	9
c_{10}	1010	Sumă(9, 10)	19
c_{11}	1011	Sumă(11, 11)	11
c_{12}	1100	Sumă(9, 12)	42
c_{13}	1101	Sumă(13, 13)	13
c_{14}	1110	Sumă(13, 14)	27
c_{15}	1111	Sumă(15, 15)	15

Vom prezenta în continuare modul în care se efectuează modificările, exemplificând pe arborele prezentat anterior. În acest scop, vom modifica valoarea celui de-al doilea element din 2 în 8 (se adună valoarea 6).

Se observă că trebuie modificate toate valorile vectorului *c* care reprezintă sume ale unei subsecvențe care conține al doilea element al şirului; acestea sunt: c_2, c_4 şi c_8. Reprezentările binare ale acestor trei poziții sunt `0010`, `0100` şi `1000`. Se observă în continuare că, fiecare astfel de reprezentare (evident, cu excepția primeia) poate fi obținută din cea anterioară prin adunarea valorii 2^r, unde r reprezintă numărul de zerouri terminale ale reprezentării binare a poziției anterioare.

Aşadar, ar trebui să efectuăm astfel de adunări până în momentul în care poziția obținută este mai mare decât dimensiunea şirului şi să creştem cu 6 valorile tuturor elementelor de pe pozițiile de la fiecare pas.

Vom exemplifica procedeul şi pentru al treilea element. Prima poziție este `0011`; numărul zerourilor terminale este 0, deci vom aduna valoarea $2^0 = 1$. Noua poziție va fi 3 + 1 = 4, deci am ajuns la poziția 4 a cărei reprezentare binară este `0100`. Numărul zerourilor terminale este 2, deci vom aduna valoarea $2^2 = 4$, poziția devenind 4 + 4 = 8. Reprezentarea binară este `1000`, deci avem trei zerouri terminale. Valoarea adunată va fi $2^3 = 8$, deci ajungem în poziția 8 + 8 = 16, aşadar am depăşit dimensiunea şirului.

Se observă că elementele c_3, c_4 şi c_8 sunt cele a căror valoare depinde de valoarea elementului de pe a treia poziție.

Algoritmul care realizează operația de modificare este următorul:

- Se identifică poziția elementului care trebuie modificat.

- Cât timp poziţia curentă este cel mult egală cu dimensiunea şirului:
  - Se modifică valoarea elementului de pe poziţia curentă.
  - Se determină numărul *k* al zerourilor terminale din reprezentarea binară a poziţiei curente.
  - Noua poziţie se determină adunând valoarea 2^k la poziţia curentă.

Se observă că numărul elementelor modificate este cel mult egal cu numărul cifrelor reprezentării binare a numărului de elemente din şir, deci operaţia are ordinul de complexitate $O(\log N)$.

Mai trebuie prezentat modul în care este efectuată operaţia de interogare. Vom încerca să aplicăm o metodă similară celei propuse în cazul algoritmului care foloseşte un vector de sume. Pentru aceasta, dacă dorim să determinăm suma elementelor subsecvenţei <*st*, *dr*>, vom determina sumele elementelor subsecvenţelor <1, *st* – 1> şi <1, *dr*>, efectuând apoi o simplă diferenţă.

Operaţiile efectuate sunt, oarecum, inversele celor de la operaţia de modificare. Vom porni din poziţia dată şi, la fiecare pas, vom determina următoarea poziţie scăzând valoarea 2^k unde *k* reprezintă numărul zerourilor terminale din reprezentarea binară a poziţiei curente. Ne vom opri în momentul în care poziţia curentă devine 0.

De exemplu, pentru a determina suma elementelor subsecvenţei <1, 11> vom porni din poziţia 11 a cărei reprezentare binară este `1011`. Numărul zerourilor terminale este 0, deci vom scădea valoarea $2^0 = 1$, ajungând în poziţia 11 – 1 = 10. Reprezentarea binară a acesteia este `1010`, deci numărul zerourilor terminale este 1. Valoarea scăzută este $2^1 = 2$, deci se ajunge în poziţia 10 – 2 = 8, a cărei reprezentare binară este 1000. De data aceasta avem trei zerouri terminale, deci vom scădea valoarea $2^3 = 8$ ajungând în poziţia 8 – 8 = 0.

Aşadar, am "trecut" prin poziţiile 11, 10 şi 8. Adunând valorile elementelor corespunzătoare (c_{11}, c_{10} şi c_8) obţinem 11 + 19 + 36 = 66, adică exact valoarea căutată.

Este uşor de observat că c_{11} reprezintă suma elementelor subsecvenţei <11, 11>, c_{10} reprezintă suma elementelor subsecvenţei <9, 10>, iar c_8 reprezintă suma elementelor subsecvenţei <1, 8>, aşadar, în momentul calculării sumei, fiecare element de pe o poziţie mai mică sau egală cu 11 este adunat o singură dată.

Pentru a determina suma elementelor unei subsecvenţe de forma <1, *dr*> vom folosi următorul algoritm:

- Se identifică elementul de pe poziţia *dr*.
- Cât timp poziţia curentă este diferită de 0:
  - Se adună la suma totală valoarea elementului de pe poziţia curentă.
  - Se determină numărul *k* al zerourilor terminale din reprezentarea binară a poziţiei curente.
  - Noua poziţie se determină scăzând valoarea 2^k din poziţia curentă.

Evident, pentru a determina suma elementelor unei subsecvențe de forma <*st*, *dr*> vom aplica de două ori algoritmul prezentat: o dată pentru subsecvența <1, *dr*> şi altă dată pentru subsecvența <1, *st* – 1>, efectuând la final diferența valorilor obținute.

De exemplu, pentru determinarea sumei elementelor subsecvenței <4, 13> vom calcula mai întâi sumele elementelor subsecvențelor <1, 13> şi <1, 3>.

Acestea sunt:

$$\text{Sumă}(1, 13) = c_{13} + c_{12} + c_8 = 13 + 42 + 36 = 91$$

şi

$$\text{Sumă}(1, 3) = c_3 + c_2 = 3 + 3 = 6,$$

diferența lor fiind 91 – 6 = 85 care este egală cu valoarea `Sumă(4, 13)`.

Se observă că numărul elementelor adunate este cel mult egal cu numărul cifrelor reprezentării binare a poziției elementului dat, deci operația are ordinul de complexitate $O(\log N)$.

Din nou, se observă că nu avem nevoie de păstrarea elementelor şirului *a*, fiind suficientă păstrarea valorilor vectorului *c* (cel care reprezintă arborele indexat binar).

Varianta în pseudocod a algoritmului de rezolvare a problemei folosind un arbore indexat binar este următoarea:

```
Algoritm ModificareEficientă:
                                                    { inițializări }
  scrie 'Introduceți numărul de elemente: '
  citeşte N                                    { dimensiunea şirului }
  pentru i ← 0, N execută:
    c_i ← 0                                  { valorile inițiale sunt nule }
  sfârşit pentru
  scrie 'Introduceți codul operației: '
  citeşte cod
  cât timp cod ≠ 3 execută:
    dacă cod = 1 atunci                                  { modificare }
      scrie 'Introduceți indicele elementului care modificat: '
      citeşte ind
      scrie 'Introduceți valoarea care va fi adunată
             (valoare negativă pentru scăderi):
      citeşte val
      poz ← 0            { poziția celui mai nesemnficativ bit cu valoarea 1 }
      cât timp ind ≤ N execută:
        c_ind ← c_ind + val
        cât timp ind & 2^poz = 0 execută:
                      { & reprezintă operația de conjuncție logică (ŞI-logic) }
          poz ← poz + 1
        sfârşit cât timp
        ind ← ind + 2^poz
```

```
        poz ← poz + 1                { valoarea poz este incrementată şi nu }
                  { reinițializată cu 0 deoarece ştim că acum avem cel puțin poz + 1 }
        { zerouri terminale, prin adunare adăugându-se cel puțin un zero terminal }
      sfârşit cât timp
    altfel                                                        { interogare }
      scrie 'Introduceți extremitățile subsecvenței: '
      citeşte st, dr
                                                  { calcularea primei sume }
      s1 ← 0                                                      { inițializare }
      poz ← 0              { poziția celui mai nesemnficativ bit cu valoarea 1 }
      cât timp dr > 0 execută:
        s1 ← s1 + cdr
        cât timp dr & 2poz = 0 execută:
          poz ← poz + 1
        sfârşit cât timp
        dr ← dr - 2poz
        poz ← poz + 1    { prin scădere se adaugă cel puțin un zero terminal }
      sfârşit cât timp
                                        { calcularea celei de-a doua sume }
      st ← st - 1             { trebuie calculată valoarea pentru st - 1 }
      s2 ← 0                                                  { inițializare }
      poz ← 0            { poziția celui mai nesemnficativ bit cu  valoarea 1 }
      cât timp st > 0 execută:
        s2 ← s2 + cst
        cât timp st & 2poz = 0 execută:
          poz ← poz + 1
        sfârşit cât timp
        st ← st - 2poz
        poz ← poz + 1
      sfârşit cât timp
      scrie 'Suma elementelor subsecvenței este: ', s1 - s2
    sfârşit dacă
    scrie 'Introduceți codul operației: '
    citeşte cod
  sfârşit cât timp
sfârşit algoritm
```

22.3. Cazul bidimensional

Problema enunțată poate fi modificată astfel încât să lucrăm în spațiul bidimensional. Enunțul noii probleme ar putea fi următorul:

Se consideră o matrice de numere întregi care are toate elementele nule. O modificare a unui element al matricei constă în adunarea unei valori la valoarea curentă (pot fi realizate şi scăderi care au forma adunării unor valori negative). Pe parcursul modificărilor pot apărea interogări referitoare la suma elementelor unei submatrice.

Aşadar, în acest caz, interogările se referă la suma valorilor dintr-o "zonă dreptunghiulară" care face parte din matrice.

Evident, pentru reprezentarea datelor vom folosi tablouri bidimensionale în locul celor unidimensionale (matrice în locul vectorilor). Din nou, enunțul poate fi reformulat în diferite moduri. O variantă ar putea fi:

Se consideră un depozit de formă dreptunghiulară având lungimea M şi lățimea N. Depozitul este împărțit în regiuni de formă pătrată având latura egală cu unitatea. În fiecare moment, într-o anumită regiune pot fi depozitate sau luate lăzi cu mere. De asemenea, în fiecare moment proprietarul depozitului ar putea cere informații referitoare la numărul total al lăzilor de mere care se află într-o regiune dreptunghiulară a depozitului.

Vom adapta cei trei algoritmi descrişi pentru cazul unidimensional pentru a rezolva noua problemă şi vom studia apoi performanțele acestora.

De data aceasta, pentru operația de modificare vom avea nevoie de doi parametri care vor reprezenta linia şi coloana pe care se află elementul a cărui valoare va fi modificată.

Pentru interogări vom avea nevoie de patru parametri care vor reprezenta coordonatele colțurilor din stânga-sus şi dreapta-jos ale unui dreptunghi.

Vom exemplifica acum cazul bidimensional al problemei folosind o matrice cu trei linii şi trei coloane.

Operație	**Matrice/Rezultat**
Inițializare	0 0 0
	0 0 0
	0 0 0
Adună(2, 2, 3)	0 0 0
	0 3 0
	0 0 0
Adună(1, 3, 5)	0 0 5
	0 3 0
	0 0 0
Sumă(2, 1, 3, 3)	3
Adună(2, 1, 1)	0 0 5
	1 3 0
	0 0 0
Sumă(1, 2, 2, 3)	8
Adună(2, 2, -1)	0 0 5

```
                                1 2 0
                                0 0 0
Sumă(1, 2, 2, 3)                7
Adună(2, 2, 5)                  0 0 5
                                1 7 0
                                0 0 0
Adună(2, 2, -5)                 0 0 5
                                1 2 0
                                0 0 0
Adună(3, 2, 4)                  0 0 5
                                1 2 0
                                0 4 0
Sumă(1, 1, 3, 3)                12
```

22.3.1. Algoritmul ineficient

Primul algoritm descris pentru cazul unidimensional poate fi adaptat foarte uşor pentru a rezolva cazul bidimensional al problemei. Vom păstra valorile matricei, le vom modifica dacă este necesar şi vom calcula sumele cerute de interogări.

Versiunea în pseudocod este prezentată în continuare:

```
Algoritm Modificare2DIneficientă:
                                                    { inițializări }
  scrie 'Introduceți dimensiunile matricei: '
  citeşte M, N
  pentru i ← 1, M execută:
    pentru j ← 1, N execută:
      a_ij ← 0                                  { valorile inițiale sunt nule }
    sfârşit pentru
  sfârşit pentru
  scrie 'Introduceți codul operației: '
  citeşte cod
  cât timp cod ≠ 3 execută:
    dacă cod = 1                                            { modificare }
      atunci
        scrie 'Introduceți indicii elementului modificat: '
        citeşte indx, indy
        scrie 'Introduceți valoarea care va fi adunată
               (valoare negativă pentru scăderi): '
        citeşte val
        a_indx,indy ← a_indx,indy + val
      altfel                                            { interogare }
        scrie 'Introduceți coordonatele colțurilor: '
        citeşte st, sus, dr, jos
```

```
        suma ← 0
        pentru i ← sus, jos execută:
          pentru j ← st, dr execută:
            suma ← suma + aij
          sfârşit pentru
        sfârşit pentru
        scrie 'Suma elementelor dreptunghiului este: ', suma
    sfârşit dacă
    scrie 'Introduceți codul operației: '
    citeşte cod
  sfârşit cât timp
sfârşit algoritm
```

Modificările se efectuează tot în timp constant, deoarece implică doar accesarea unui element al matricei şi modificarea valorii sale.

Datorită faptului că pentru calcularea sumei elementelor dintr-un dreptunghi este necesară parcurgerea tuturor elementelor dreptunghiului, această operație are ordinul de complexitate $O(l \cdot h)$, unde l şi h sunt lungimile laturilor dreptunghiului.

22.3.2. Matrice de sume

Adaptarea algoritmului care foloseşte un vector de sume este destul de simplă. Vom păstra o matrice b ale cărei elemente b_{ij} vor conține sumele corespunzătoare dreptunghiurilor cu colțul din stânga-sus în poziția (1, 1) şi cel din dreapta-jos în poziția (i, j). Este uşor de observat că pentru a calcula valoarea `Sumă(st, sus, dr, jos)` poate fi folosită relația:

```
Sumă(st, sus, dr, jos) = Sumă(1, 1, dr, jos) -
   Sumă(1, 1, st - 1, jos) - Sumă(1, 1, dr, sus - 1) +
   Sumă(1, 1, st - 1, sus - 1).
```

Se observă că prin scăderea valorii `Sumă(1, 1, st-1, jos)` se scad valorile tuturor elementelor aflate la dreapta dreptunghiului considerat, iar prin scăderea valorii `Sumă(1, 1, dr, sus-1)` se scad valorile elementelor aflate deasupra dreptunghiului.

Aceste operații implică scăderea de două ori a tuturor elementelor aflate la stânga şi deasupra dreptunghiului considerat, motiv pentru care va trebui să adunăm valoarea `Sumă(1, 1, st-1, sus-1)`.

Folosind acest artificiu avem nevoie de accesarea a patru elemente ale matricei b, deci operația se realizează în timp constant.

Din nou, în momentul modificării valorii unui element de coordonate (x, y), va trebui să modificăm valorile tuturor elementelor matricei b de coordonate (i, j) pentru care $i \geq x$ şi $j \geq y$. Aşadar, vom modifica $(M - x + 1) \cdot (N - y + 1)$ elemente, ordinul de complexitate devenind $O(M \cdot N)$.

Pentru a putea determina valori pentru dreptunghiuri în care colțul din stânga-sus are una dintre coordonate egală cu 1 va trebui să considerăm că valorile elementelor matricei sunt nule dacă cel puțin unul dintre cei doi indici este 0.

Prezentăm versiunea în pseudocod a algoritmului descris:

```
Algoritm ModificareMatriceSume:
                                                        { inițializări }
  scrie 'Introduceți dimensiunile matricei: '
  citeşte M, N                                    { dimensiunile matricei }
  pentru i ← 0, M execută:
    pentru i ← 0, N execută:
      b_ij ← 0                                  { valorile inițiale sunt nule }
    sfârşit pentru
  sfârşit pentru
  scrie 'Introduceți codul operației: '
  citeşte cod
  cât timp cod ≠ 3 execută:
    dacă cod = 1                                                { modificare }
      atunci
        scrie 'Introduceți indicii elementului modificat: '
        citeşte indx, indy
        scrie 'Introduceți valoarea care va fi adunată
               (valoare negativă pentru scăderi): '
        citeşte val
        pentru i ← indx, M execută:
          pentru i ← indy, N execută:
            b_ij ← b_ij + val
          sfârşit pentru
        sfârşit pentru
      altfel                                                { interogare }
        scrie 'Introduceți coordonatele colțurilor: '
        citeşte st, sus, dr, jos
        scrie 'Suma elementelor secvenței este: ',
               b_dr,jos - b_st-1,jos - b_dr,sus-1 + b_st-1,sus-1.
    sfârşit dacă
    scrie 'Introduceți codul operației: '
    citeşte cod
  sfârşit cât timp
sfârşit algoritm
```

22.3.3. Arbore de arbori indexați binar

Din nou, avem doi algoritmi în care una dintre operații este eficientă, în timp ce cealaltă necesită un timp de execuție mai mare. Folosind arborii indexați binar, vom putea

găsi algoritmi care realizează ambele operații într-un timp de ordinul $O(\log M \cdot \log N)$, unde M şi N sunt dimensiunile matricei.

Pentru a rezolva problema vom crea un arbore de arbori indexați binar. Aşadar, vom avea o matrice c ale cărei elemente c_{ij} vor reprezenta sumele elementelor din dreptunghiul care are colțul din stânga-sus în poziția $(i - 2^k + 1, j - 2^l + 1)$, iar cel din dreapta-jos în poziția (i, j). Valoarea k reprezintă numărul zerourilor terminale din reprezentarea binară a lui i, iar l reprezintă numărul zerourilor terminale din reprezentarea binară a lui j.

De exemplu, pentru matricea:

```
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16
```

elementele arborelui de arbori indexați binar vor avea următoarele semnificații:

```
Sumă(1,1,1,1) Sumă(1,1,1,2) Sumă(1,3,1,3) Sumă(1,1,1,4)
Sumă(1,1,2,1) Sumă(1,1,2,2) Sumă(1,3,2,3) Sumă(1,1,2,4)
Sumă(3,1,3,1) Sumă(3,1,3,2) Sumă(3,3,3,3) Sumă(3,3,3,4)
Sumă(1,1,4,1) Sumă(1,1,4,2) Sumă(1,3,4,3) Sumă(1,1,4,4).
```

Aşadar, valorile matricei care reprezintă arborele binar vor fi:

```
 1   3   3   6
 6  14  10  36
 9  19  11  23
28  60  36 136.
```

Să presupunem că trebuie să modificăm valoarea elementului de coordonate (2, 1) care face parte dintr-o matrice cu 8 linii şi 4 coloane. Vom aplica de două ori principiul descris pentru cazul unidimensional.

Vom identifica linia i pe care se află elementul şi vom efectua modificări în toate coloanele de pe acea linie în care acestea trebuie efectuate. Pentru aceasta vom identifica coloana j, vom determina numărul k al zerourilor terminale şi apoi vom aduna la j valoarea 2^k. Ajungem într-o nouă poziție şi continuăm procedeul până în momentul în care valoarea coloanei curente depăşeşte numărul coloanelor matricei.

În acest moment vom determina numărul zerourilor terminale l din reprezentarea binară a valorii i şi ne vom "muta" pe linia $i + 2^l$. Vom reveni la coloana j şi vom parcurge aceeaşi paşi ca şi la linia anterioară.

Vom continua până în momentul în care valoarea liniei curente va fi mai mare decât numărul de linii din matrice.

Pe fiecare linie vom modifica cel mult $\log_2 N$ elemente; în plus, vom modifica elemente de pe cel mult $\log_2 M$ linii, aşadar, ordinul de complexitate al operației de modificare a unui element este $O(\log M \cdot \log N)$.

Ca urmare, dacă se modifică elementul de coordonate (3, 1), atunci vor trebui modificate următoarele elemente ale matricei care reprezintă arborele de arbori indexați binar: c_{31}, c_{32}, c_{34}, c_{41}, c_{42}, c_{44}, c_{81}, c_{82} şi c_{84}.

În cazul în care dorim să efectuăm o interogare vom folosi aceeaşi metodă. Singura diferență este, din nou, faptul că valorile de forma 2^k sunt scăzute în loc să fie adunate.

Datorită faptului că, de data aceasta, avem nevoie de patru sume pentru a furniza rezultatul, algoritmul va fi aplicat de patru ori. Ordinul de complexitate al algoritmului va fi tot $O(\log M \cdot \log N)$.

Vom prezenta în continuare versiunea în pseudocod a algoritmului de rezolvare a problemei în cazul bidimensional, folosind un arbore de arbori indexați binar.

```
Algoritm Modificare2Deficientă:
                                                    { inițializări }
  scrie 'Introduceți dimensiunile matricei: '
  citeşte M, N
  pentru i ← 0, M execută:
    pentru j ← 0, N execută:
      c[i,j] ← 0                                    { valorile inițiale sunt nule }
    sfârşit pentru
  sfârşit pentru
  scrie 'Introduceți codul operației: '
  citeşte cod
  cât timp cod ≠ 3 execută:
    dacă cod = 1                                    { modificare }
      atunci
        scrie 'Introduceți indicii elementului modificat: '
        citeşte indx, indy
        scrie 'Introduceți valoarea care va fi adunată
               (valoare negativă pentru scăderi):
        citeşte val
        pozi ← 0
        cât timp indx ≤ M execută:
          pozj ← 0
          j ← indy
          cât timp j ≤ N execută:
            c[indx,j] ← c[indx,j] + val
            cât timp indx & 2^pozj = 0 execută:
              pozj ← pozj + 1
            sfârşit cât timp
```

```
      j ← j + 2^pozj
      pozj ← pozj + 1
    sfârşit cât timp
    cât timp indx & 2^pozi = 0 execută:
      pozi ← pozi + 1
    sfârşit cât timp
    indx ← indx + 2^pozi
    pozi ← pozi + 1
  sfârşit cât timp
altfel                                          { interogare }
  scrie 'Introduceți coordonatele colțurilor: '
  citeşte st, sus, dr, jos
  { calcularea primei sume }
  s ← 0
  x ← jos
  y ← dr
  pozi ← 0
  cât timp x ≥ 0 execută:
    pozj ← 0
    j ← y
    cât timp y ≥ 0 execută:
      s1 ← s1 + c_x,j
      cât timp x & 2^pozj = 0 execută:
        pozj ← pozj + 1
      sfârşit cât timp
      j ← j - 2^pozj
      pozj ← pozj + 1
    sfârşit cât timp
    cât timp x & 2^pozi = 0 execută:
      pozi ← pozi + 1
    sfârşit cât timp
    x ← x - 2^pozi
    pozi ← pozi + 1
  sfârşit cât timp
  s1 ← s
  s ← 0
  { calcularea celei de-a doua sume }
  x ← jos
  y ← st - 1
  ...                          { se aplică exact aceeaşi secvență }
  s2 ← s
  s ← 0
```

```
        { calcularea celei de-a treia sume }
        x ← sus - 1
        y ← dr
        ...                          { se aplică din nou exact aceeaşi secvenţă }
        s3 ← s
        s ← 0
        { calcularea celei de-a patra sume }

        x ← sus - 1
        y ← dr - 1
        ...                          { se aplică din nou exact aceeaşi secvenţă }
        s4 ← s
        scrie 'Suma elementelor dreptunghiului este: ',
              s1 - s2 - s3 + s4
    sfârşit dacă
    scrie 'Introduceţi codul operaţiei: '
    citeşte cod
  sfârşit cât timp
sfârşit algoritm
```

22.4. Cazul tridimensional

Problema enunţată poate fi modificată astfel încât să lucrăm în spaţiul tridimensional. Enunţul ar putea fi următorul:

Se consideră un tablou tridimensional care conţine numere întregi şi toate elementele sale sunt nule. O modificare a unui element al tabloului constă în adunarea unei valori la valoarea curentă (pot fi realizate şi scăderi care au forma adunării unor valori negative). Pe parcursul modificărilor pot apărea interogări referitoare la suma elementelor unui subtablou "paralelipipedic" .

Aşadar, în acest caz, interogările se referă la suma valorilor dintr-o "zonă paralelipipedică" a tabloului. Evident, pentru reprezentarea datelor vom folosi tablouri tridimensionale. Şi acest enunţ poate fi reformulat în diferite moduri. O variantă ar putea fi:

O zonă a spaţiului este reprezentată de un paralelipiped format din cuburi cu latura egală cu unitatea. Dimensiunile paralelipipedului sunt M, N şi P. În fiecare sector (cub) pot sosi sau pleca nave spaţiale. De asemenea, în fiecare moment amiralul flotei poate cere informaţii referitoare la numărul total de nave spaţiale dintr-o regiune paralelipipedică din această zonă a spaţiului.

Nu vom mai prezenta pe larg cele trei tipuri de algoritmi; vom face doar câteva precizări care ne vor ajuta să generalizăm problema pentru spaţii *n*-dimensionale.

În primul rând, în fiecare poziție în care algoritmii pentru cazul bidimensional conțineau două cicluri imbricate, pentru algoritmii care rezolvă cazul tridimensional vom avea trei cicluri. Ca urmare, ordinul de complexitate al primilor doi algoritmi va deveni $O(M \cdot N \cdot P)$, ordinul de complexitate al celui de-al treilea algoritm va fi $O(\log M \cdot \log N \cdot \log P)$.

În al doilea rând, elementele tabloului tridimensional vor fi identificate prin trei indici. Aşadar, pentru o modificare vom avea nevoie de trei parametri "geometrici", iar pentru o interogare de şase astfel de parametri.

În al treilea rând, pentru ultimii doi algoritmi va trebui să calculăm opt sume în loc de patru. Formula de calcul va fi următoarea:

```
Sumă(x1, y1, z1, x2, y2, z2) =
      Sumă(1, 1, 1, x2, y2, z2) -
      Sumă(1, 1, 1, x1 - 1, y2, z2) -
      Sumă(1, 1, 1, x2, y1 - 1, z2) -
      Sumă(1, 1, 1, x2, y2, z1 - 1) +
      Sumă(1, 1, 1, x1 - 1, y1 - 1, z2) +
      Sumă(1, 1, 1, x1 - 1, y2, z1 - 1) +
      Sumă(1, 1, 1, x2, y1 - 1, z1 - 1) -
      Sumă(1, 1, 1, x1 - 1, y1 - 1, z1 - 1).
```

Mai precizăm faptul că pentru cel de-al treilea algoritm vom folosi un tablou tridimensional care va reprezenta un arbore de arbori indexaţi binar.

Toate celelalte aspecte ale celor trei algoritmi descrişi pot fi adaptate foarte simplu pentru a rezolva problema tridimensională.

21.5. Cazul *n*-dimensional

Vom generaliza acum problema pentru a putea fi enunțată într-un spațiu cu un număr oricât de mare ca dimensiuni. O variantă a enunțului este:

Se consideră un tablou n-dimensional care conține numere întregi şi toate elementele sale sunt nule. O modificare a unui element al tabloului constă în adunarea unei valori la valoarea curentă (pot fi realizate şi scăderi care au forma adunării unor valori negative). Pe parcursul modificărilor pot apărea interogări referitoare la suma elementelor unui subtablou.

Aşadar, în acest caz, interogările se referă la suma valorilor dintr-o "zonă *n*-dimensională" a tabloului. Evident, pentru reprezentarea datelor vom folosi tablouri *n*-dimensionale.

De data aceasta, cele două cicluri imbricate din algoritmii pentru cazul bidimensional (cele trei din algoritmii pentru cazul tridimensional) vor fi înlocuite de *n* cicluri imbricate.

Ca urmare, ordinul de complexitate al primilor doi algoritmi va depinde de produsul celor *n* dimensiuni, iar ordinul de complexitate al celui de-al treilea algoritm va depinde de produsul logaritmilor celor *n* dimensiuni.

Evident, vom avea acum *n* parametri "geometrici" pentru o modificare şi $2 \cdot n$ parametri de acest tip pentru o interogare. Numărul sumelor care trebuie calculate pentru o interogare (la al doilea şi al treilea algoritm) este, pentru cazul general, 2^n.

Ţinând cont de aceste observaţii, algoritmii prezentaţi pot fi folosiţi pentru rezolvarea unor probleme de acest tip în care numărul dimensiunilor este oricât de mare.

22.6. Rezumat

În cadrul acestui capitol am prezentat o structură de date care permite rezolvarea eficientă a unei categorii de probleme. Pentru a evidenţia performanţele algoritmilor care folosesc o astfel de structură de date, am prezentat şi alte două categorii de algoritmi, mult mai uşor de implementat, dar ineficienţi.

De asemenea, am prezentat modul în care pot fi generalizaţi algoritmii folosiţi pentru cazul cel mai simplu (cel unidimensional) pentru a rezolva problema pentru cazuri în care numărul dimensiunilor este oricât de mare.

Am exemplificat algoritmii pentru cazul în care interogările se referă la suma anumitor elemente. Cu toate acestea este evident faptul că, în cazul tuturor acestor algoritmi, această operaţie poate fi înlocuită cu altele (produsul elementelor, minimul sau maximul lor etc.).

22.7. Implementări sugerate

Pentru a vă însuşi noţiunile prezentate în cadrul acestui capitol vă sugerăm să realizaţi implementări pentru:

1. problema determinării sumei elementelor unei subsecvenţe a unui şir, folosind algoritmul ineficient;
2. problema determinării sumei elementelor unei subsecvenţe a unui şir, folosind un vector de sume;
3. problema determinării sumei elementelor unei subsecvenţe a unui şir, folosind un arbore indexat binar;
4. problema determinării sumei elementelor unei secţiuni a unei matrice, folosind algoritmul ineficient;
5. problema determinării sumei elementelor unei secţiuni a unei matrice, folosind o matrice de sume;
6. problema determinării sumei elementelor unei secţiuni a unei matrice, folosind un arbore de arbori indexaţi binar.

22.8. Probleme propuse

În continuare vom prezenta enunțurile câtorva probleme pe care vi le propunem spre rezolvare. Toate aceste probleme pot fi rezolvate folosind structura de date prezentată în cadrul acestui capitol. Cunoştinţele suplimentare necesare sunt minime.

22.8.1. Jocuri

Descrierea problemei

Un distribuitor de jocuri furnizează produse pentru *N* clienți; inițial nici un client nu a cumpărat nici un joc. Cei *N* clienți sunt identificați folosind numere cuprinse între 1 şi *N*.

O operațiune de vânzare are loc atunci când unul dintre cei *N* clienți cumpără jocuri în valoare de *x* €.

Pe măsură ce operațiunile de vânzare se desfăşoară, directorul companiei distribuitoare de jocuri doreşte să primească informații referitoare la vânzări. Astfel, el poate cere comunicarea sumei totale care ar trebui încasată de la clienții cu numere de ordine cuprinse între *i* şi *j*.

O operațiune de vânzare este descrisă prin trei numere naturale. Primul dintre ele este întotdeauna 1 şi identifică tipul operațiunii. Al doilea este un număr cuprins între 1 şi *N* şi identifică clientul căruia i se vând jocuri. Al treilea număr este *x* şi indică valoarea vânzării (exprimată în €).

O cerere de informații este descrisă tot prin trei numere naturale. Primul dintre ele este întotdeauna 2 şi identifică tipul operațiunii. Următoarele două sunt cuprinse între 1 şi *N* şi identifică clienții *i* şi *j*. În urma acestei cereri de informații, trebuie determinată suma vânzărilor către clienții *i*, *i* + 1, ..., *j* - 1, *j*.

Date de intrare

Prima linie a fişierului de intrare `JOCURI.IN` conține numerele *N* şi *K*, separate printr-un singur spațiu. Primul indică numărul clienților, iar al doilea numărul total de operațiuni de vânzare şi cereri de infomații.

Următoarele *K* linii conțin câte trei numere naturale, separate prin spații. Primul dintre aceste numere este *t*, şi poate avea valoarea 1 sau valoarea 2.

Dacă valoarea numărului *t* este 1, atunci următoarele două numere *i* şi *x* vor indica faptul că are loc o vânzare către clientul *i*, în valoare de *x* €.

Dacă valoarea numărului *t* este 2, atunci următoarele două numere *i* şi *j* vor indica faptul că se cere o statistică a vânzărilor către clienții cu numere de ordine cuprinse între *i* şi *j*. Vor fi luate în considerare doar vânzările descrise pe liniile anterioare liniei curente şi nu şi cele care vor fi descrise pe liniile următoare.

Date de ieşire
Fişierul de ieşire **JOCURI.OUT** va conţine câte o linie pentru fiecare dintre liniile din fişierul de intrare pentru care valoarea *t* este 2. Pe fiecare dintre aceste linii se va afla câte un număr care va indica rezultatul statisticii cerute de director.

Restricţii şi precizări
- $1 \le N \le 70000$;
- $1 \le K \le 200000$;
- suma totală a vânzărilor nu poate depăşi 2000000000;
- pot fi efectuate mai multe vânzări către acelaşi client.

Exemple

JOCURI.IN
```
7 9
1 3 1000
1 5 2000
2 4 6
1 4 3000
2 4 6
2 1 4
2 1 2
1 1 4000
2 1 7
```

JOCURI.OUT
```
2000
5000
4000
0
10000
```

Timp de execuţie: 1 secundă/test

22.8.2. Grădiniţă

Descrierea problemei
Copiii de la o grădiniţă au ieşit la joacă. Pe terenul de joacă se află o porţiune întinsă de nisip care este împărţită în parcele de formă pătrată a căror latură este egală cu unitatea. Aşadar, porţiunea de nisip poate fi considerată a fi un caroiaj cu *M* linii şi *N* coloane.

Iniţial, nu se află nici un copil în regiunea cu nisip. În fiecare moment pe o parcelă pot intra mai mulţi copii; de asemenea, în fiecare moment, de pe parcelă pot pleca mai mulţi copii.

Pe măsură ce copiii intră pe parcele sau ies de pe parcele, directoarea grădiniţei le cere educatoarelor să îi spună câţi copii se află într-o zonă dreptunghiulară a terenului. Zona dreptunghiulară este identificată prin colţul din stânga-sus şi colţul din dreapta-jos.

Intrarea sau ieşirea copiilor dintr-o parcelă este descrisă prin patru numere naturale. Primul dintre ele este întotdeauna 1 şi identifică tipul evenimentului. Următoarele două reprezintă coordonatele parcelei în care intră sau din care ies copii, iar ultimul indică numărul copiilor care intră sau ies (o valoare pozitivă indică faptul că intră copii pe parcelă, iar o valoare negativă indică faptul că ies copii de pe parcelă).

O cerere de informații este descrisă prin cinci numere naturale. Primul dintre ele este întotdeauna 2 şi identifică tipul evenimentului. Următoarele două reprezintă coordonatele parcelei din colțul stânga-sus al regiunii, iar ultimele două reprezintă coordonatele parcelei din colțul dreapta-jos. În urma unei astfel de cereri, trebuie determinat numărul total al copiilor care se află, în acel moment, în regiunea respectivă.

Date de intrare

Prima linie a fişierului de intrare `COPII.IN` conține numerele M, N şi K, separate printr-un singur spațiu. Primele două indică dimensiunile terenului, iar al treilea reprezintă numărul total al evenimentelor.

Următoarele K linii conțin câte patru sau cinci numere naturale, separate prin spații. Primul dintre aceste numere este t, şi poate avea valoarea 1 sau valoarea 2.

Dacă valoarea numărului t este 1, atunci linia va mai conține trei numere. Următoarele două vor identifica parcele pe care intră copii sau de pe care pleacă, iar ultimul va identifica numărul copiilor care intră (dacă numărul este pozitiv) sau pleacă (dacă numărul este negativ).

Dacă valoarea numărului t este 2, atunci linia va mai conține patru numere. Următoarele două numere reprezintă coodonatele coluțului stânga-sus al zonei pentru care se cer informații, iar ultimele două reprezintă coodonatele coluțului dreapta-jos al zonei respective. Vor fi luate în considerare doar intrările şi ieşirile descrise pe liniile anterioare liniei curente şi nu şi cele care vor fi descrise pe liniile următoare.

Date de ieşire

Fişierul de ieşire `COPII.OUT` va conține câte o linie pentru fiecare dintre liniile din fişierul de intrare pentru care valoarea t este 2. Pe fiecare dintre aceste linii se va afla câte un număr care va indica rezultatul statisticii cerute de directoare.

Restricții şi precizări

- $1 \le M, N \le 200$;
- $1 \le K \le 200000$;
- numărul total al copiilor aflați pe teren nu poate depăşi 2000000000;
- nu poate apărea situația în care numărul copiilor de pe o parcelă să fie negativ;
- pentru o cerere de informații referitoare la regiunea care are coordonatele (x_1, y_1) pentru colțul stânga-sus şi coordonatele (x_2, y_2) pentru colțul dreapta-jos, vom avea întotdeauna $x_1 \le x_2$ şi $y_1 \le y_2$;

- liniile caroiajului sunt identificate prin numere cuprinse între 1 şi *M*, iar coloanele sunt identificate prin numere cuprinse între 1 şi *N*.

Exemple

COPII.IN
```
2 2 10
1 1 1 2
1 2 2 3
2 1 1 2 2
1 1 1 -1
2 1 1 1 2
1 2 1 4
2 2 1 2 2
1 2 2 -2
2 2 1 2 2
2 1 1 2 2
```

COPII.OUT
```
5
1
7
5
6
```

Timp de execuţie: 1 secundă/test

22.8.3. Cpt. Jean-Luc Picard

Descrierea problemei

Din nefericire, nava stelară *U.S.S. Enterprise* a fost nevoită să intre în *Zona Neutră Romulană* pentru a duce la îndeplinire o misiune de salvare. Pentru a reuşi să treacă neobservaţi, *Cpt. Jean-Luc Picard* a început să studieze modul în care se deplasează păsările de pradă romulane în acea regiune a spaţiului.

Zona neutră poate fi privită ca fiind o regiune paralelipipedică formată din sectoare de formă cubică a căror latură este egală cu unitatea.

Iniţial, nu poate fi observată nici o navă romulană, deoarece toate folosesc dispozitivele de camuflare. În fiecare moment, într-un anumit sector pot apărea sau dispărea mai multe nave romulane.

Pe măsură ce navele apar şi dispar *Cpt. Jean-Luc Picard* cere rapoarte referitoare la numărul păsărilor de pradă vizibile în anumite zone de formă paralelipipedică. Zonele sunt identificate prin coordonatele a două colţuri opuse ale parelelipipedului.

Apariţia sau dispariţia unor nave romulane este descrisă prin cinci numere naturale. Primul dintre ele este întotdeauna 1 şi identifică tipul evenimentului. Următoarele trei reprezintă coordonatele sectorului în care apar sau din care dispar nave, iar ultimul indică numărul navelor care apar sau dispar (o valoare pozitivă indică faptul că navele devin vizibile, iar o valoare negativă indică faptul că navele şi-au activat dispozitivele de camuflare, devenind invizibile).

O cerere de informaţii este descrisă prin şapte numere naturale. Primul dintre ele este întotdeauna 2 şi identifică tipul evenimentului. Următoarele trei reprezintă coor-

donatele unui colț al paralelipipedului care descrie zona, iar ultimele trei reprezintă coordonatele colțului opus. În urma unei astfel de cereri, trebuie determinat numărul total al navelor vizibile care se află, în acel moment, în zona respectivă.

Date de intrare
Prima linie a fişierului de intrare **PICARD.IN** conține numerele M, N, P şi K, separate printr-un singur spațiu. Primele trei indică dimensiunile *Zonei Neutre*, iar al patrulea reprezintă numărul total al evenimentelor.

Următoarele K linii conțin câte cinci sau şapte numere naturale, separate prin spații. Primul dintre aceste numere este t, şi poate avea valoarea 1 sau valoarea 2.

Dacă valoarea numărului t este 1, atunci linia va mai conține patru numere. Următoarele trei vor identifica sectorul în care apar sau dispar nave, iar ultimul va identifica numărul navelor care apar (dacă numărul este pozitiv) sau dispar (dacă numărul este negativ)

Dacă valoarea numărului t este 2, atunci linia va mai conține şase numere. Următoarele trei numere reprezintă coodonatele unui colț al zonei pentru care se cer informații, iar ultimele trei reprezintă coodonatele colțului opus al zonei respective. Vor fi luate în considerare doar aparițiile şi disparițiile descrise pe liniile anterioare liniei curente şi nu şi cele care vor fi descrise pe liniile următoare.

Date de ieşire
Fişierul de ieşire **PICARD.OUT** va conține câte o linie pentru fiecare dintre liniile din fişierul de intrare pentru care valoarea t este 2. Pe fiecare dintre aceste linii se va afla câte un număr care va indica rezultatul statisticii cerute de *Cpt. Jean-Luc Picard*.

Restricții şi precizări
- $1 \le M, N, P \le 50$;
- $1 \le K \le 200000$;
- numărul total al păsărilor de pradă romulane nu poate depăşi 2000000000;
- nu poate apărea situația în care numărul navelor romulane dintr-un sector să fie negativ;
- pentru o cerere de informații referitoare la zona care are coordonatele (x_1, y_1, z_1) pentru un colț şi coordonatele (x_2, y_2, z_2) pentru colțul opus, vom avea întotdeauna $x_1 \le x_2$, $y_1 \le y_2$ şi $z_1 \le z_2$;
- prima dintre cele trei coordonate poate varia între 1 şi M, a doua poate varia între 1 şi N, iar a treia între 1 şi P.

Exemple

PICARD.IN	**PICARD.OUT**
1 2 2 10 1 1 1 1 2	5 1

```
1 1 2 2 3                    7
2 1 1 1 1 2 2                5
1 1 1 1 -1                   6
2 1 1 1 1 1 2
1 1 2 1 4
2 1 2 1 1 2 2
1 1 2 2 -2
2 1 2 1 1 2 2
2 1 1 1 1 2 2
```

Timp de execuție: 1 secundă/test

22.9. Soluțiile problemelor

Toate aceste probleme reprezintă simple aplicații directe ale utilizării arborilor indexați binar.

Problema ***Jocuri*** reprezintă cazul unidimensional, deci vom utiliza un simplu arbore indexat binar. Datorită faptului că efectuăm K operații de căutare sau actualizare, ordinul de complexitate va fi $O(K \cdot \log N)$.

Problema ***Grădiniță*** reprezintă cazul bidimensional; aşadar, pentru a o rezolva, vom utiliza un arbore de arbori indexați binar. Şi în această situație avem K operații, ordinul de complexitate al algoritmului de rezolvare a problemei fiind $O(K \cdot \log M \cdot \log N)$.

În sfârşit, problema ***Cpt. Jean-Luc Picard*** reprezintă cazul tridimensional, deci vom utiliza un arbore de arbori de arbori indexați binar. Datorită faptului că efectuăm tot K operații, ordinul de complexitate al algoritmului de rezolvare va fi $O(K \cdot \log M \cdot \log N \cdot \log P)$.