Abordări diverse folosind tehnica sumelor parţiale

Mirel Coşulschi
mirelc@central.ucv.ro

Mihai Gabroveanu
mihaiug@central.ucv.ro

April, 2023

1 Enunţ problema

Se consideră `A` un tablou bidimensional cu n linii, n coloane şi elemente numere naturale. O zonă triunghiulară a tabloului, reprezentată de tripletul `(lin, col, k)`, este o zonă de forma unui triunghi dreptunghic cu catetele de lungime egală cu `|k|`, definită astfel:

1. Pentru $k > 0$, zona este compusă din k linii:
   - pe prima linie a zonei se află elementele $A[lin][col]$, $A[lin][col+1]$, ..., $A[lin][col+k-1]$;
   - pe a doua linie a zonei se află elementele $A[lin+1][col]$, $A[lin+1][col+1]$, ..., $A[lin+1][col+k-2]$;
   - pe a treia linie a zonei se află elementele $A[lin+2][col]$, $A[lin+2][col+1]$, ..., $A[lin+2][col+k-3]$;
   - ...
   - pe ultima linie a zonei se află elementul $A[lin+k-1][col]$.
2. Pentru $k < 0$, zona este compusă din $|k| = -k$ linii:
   - pe prima linie a zonei se află elementul $A[lin-|k|+1][col]$;
   - pe a doua linie a zonei se află elementele $A[lin-|k|+2][col-1]$, $A[lin-|k|+2][col]$;
   - ...
   - pe ultima linie a zonei se află elementele $A[lin][col-|k|+1]$, $A[lin][col-|k|+2]$, ..., $A[lin][col]$.

Suma elementelor ce compun o zonă triunghiulară se numeşte suma zonei.

Să se realizeze un program care, cunoscând tabloul A şi Q zone triunghiulare, determină cea mai mare dintre sumele zonelor.

Date de intrare

Fişierul de intrare `triunghi.in` conţine pe prima linie numărul natural n, cu semnificaţia din enunţ. Pe următoarele n linii se găsesc câte n valori naturale, reprezentând elementele tabloului A.

Pe linia $n+2$ se află numărul natural Q, reprezentând numărul zonelor triunghiulare. Pe următoarele Q linii se găsesc tripletele de valori `lin col k`, care reprezintă cele Q zone, în forma descrisă în enunţ. Valorile aflate pe aceeaşi linie a fişierului sunt separate prin câte un spaţiu.

Date de ieşire

Fişierul de ieşire `triunghi.out` va conţine o singură linie pe care va fi scris un număr natural reprezentând suma maximă cerută.

Restricţii şi precizări

- $3 \leq n \leq 1000$; $1 \leq Q \leq 100000$; $2 \leq |k| \leq n$;
- Valorile din tablou sunt numere naturale din intervalul $[1, 100]$.
- Liniile şi coloanele tabloului A sunt numerotate de la 1 la n (liniile de sus în jos, iar coloanele de la stânga la dreapta).
- $|k|$ reprezintă modulul numărului k (k, pentru $k \geq 0$, respectiv $-k$, pentru $k < 0$).
- Se garantează că orice zonă triunghiulară dintre cele Q este complet inclusă în tabloul A.

(Triunghi, OJI 2020, `https://www.pbinfo.ro/probleme/3438/triunghi5`)

2 Modalităţi de rezolvare

2.1 Varianta I

Se calculează suma elementelor pentru fiecare linie: se notează cu $sl_{i,j}$ - suma elementelor de pe linia i, de la prima coloană până la coloana j.

Astfel vom avea:

$$sl_{i,j} = a_{i,1} + a_{i,2} + \ldots + a_{i,j} \tag{1}$$

Pentru o zonă triunghiulară `(lin, col, k)`, vom calcula suma elementelor aflate în interiorul acelei zone triunghiulare sub forma unei sume de linii:

- pentru $k > 0$ avem

```
for (i = 0; i < k; i++) {
  suma_zona_triunghiulara += sl[l + i][c + k - 1 - i] - sl[l + i][c - 1];
}
```

- pentru $k < 0$ avem

```
k = -k;
for (i = 0; i < k; i++) {
  suma_zona_triunghiulara += s[l - k + 1 + i][c]
                           - s[l - k + 1 + i][c - 1 - i];
}
```

Complexitatea algoritmului este $O(n^2 + Q \times n)$.

Codul sursă al programului este următorul:

Listing 1: `triunghi-v1.c`

```
#include <stdio.h>

#define NMAX 1000
```

```
int s[NMAX + 1][NMAX + 1];

int main() {
  FILE *fin, *fout;
  int n, i, j, v, q, l, c, k, total, smax;

  fin = fopen("triunghi.in", "r");
  fout = fopen("triunghi.out", "w");

  fscanf(fin, "%d", &n);
  for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
      fscanf(fin, "%d", &v);

      s[i][j] = s[i][j - 1] + v;
    }
  }

  smax = 0;
  fscanf(fin, "%d", &q);
  for (j = 0; j < q; j++) {
    fscanf(fin, "%d %d %d", &l, &c, &k);

    total = 0;
    if (k > 0) {
      for (i = 0; i < k; i++) {
        total += s[l + i][c + k - 1 - i] - s[l + i][c - 1];
      }
    } else {
        k = -k;
        for (i = 0; i < k; i++) {
          total += s[l - k + 1 + i][c] - s[l - k + 1 + i][c - 1 - i];
        }
    }

    if (total > smax) {
      smax = total;
    }
  }

  fprintf(fout, "%d\n", smax);

  fclose(fin);
  fclose(fout);

  return 0;
}
```

2.2 Varianta a II-a

Se calculează suma elementelor pentru fiecare zonă dreptunghiulară de coordonate $(1,1)$ - (l,c). Se notează cu $sd_{l,c}$ - suma elementelor aflate în interiorul zonei dreptunghiulare de coordonate $(1,1)$ - (l,c).

Pentru o zonă triunghiulară `(l, c, k)`, se va calcula suma elementelor aflate în interiorul acelei zone triunghiulare astfel:

- pentru $k > 0$:

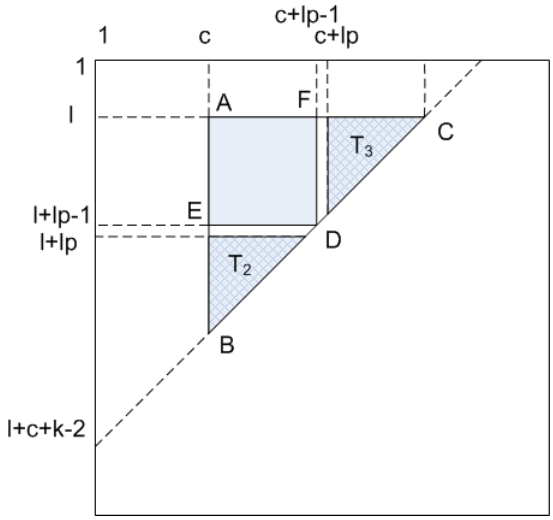

Fig. 1: Triunghiul se descompune într-un pătrat şi două triunghiuri dreptunghic-isoscele, mai mici.

Se va descompune triunghiul dreptunghic isoscel ($\triangle ABC$) într-un pătrat ($AFDE$) şi două triunghiuri dreptunghic-isoscele, mai mici (triunghiurile notate T_2 şi T_3) (a se vedea figura 1).

Suma elementelor aflate în interiorul zonei triunghiulare va fi:

```
suma_zona_triunghiulara = suma elementelor zonei patrate (l,c) - (l+lp-1, c+cp-1) # no
             + suma elementelor zonei triunghiulare (l + lp, c, k - lp)
             + suma elementelor zonei triunghiulare (l, c + lp, k - lp)
```

Se observă caracterul recursiv al acestei formule.

Se identifică pătratul de latură maximă (pătratul `AFDE`) înscris în zona triunghiulară `(l,c,k)` ($\triangle ABC$), având colţul din stânga-sus în punctul (l,c). Latura acestui pătrat va avea valoarea:

$$lp = k/2 + (k \mod 2) \tag{2}$$

Suma elementelor aflate în zona pătratului de coordonate (l,c) - $(l+lp-1, c+lp-1)$ (pătratul `AFDE` din figura 2) se calculează astfel:

$$sp = sd_{l+lp-1,c+lp-1} - sd_{l+lp-1,c-1} - sd_{l-1,c+lp-1} + sd_{l-1,c-1} \tag{3}$$

Suma elementelor pătratului de coordonate $A(l,c)$ - $D(l+lp-1, c+lp-1)$ se obţine din suma elementelor dreptunghiului de coordonate $M(1,1)$ - $D(l+lp-1, c+lp-1)$, din care se scade suma elementelor dreptunghiului de coordonate $M(1,1)$ - $E_1(l+lp-1, c-1)$ şi suma

elementelor dreptunghiului de coordonate $M(1,1)$ - $F_1(l-1, c+lp-1)$, la care se adaugă suma elementelor dreptunghiului de coordonate $M(1,1)$ - $A_1(l-1, c-1)$.

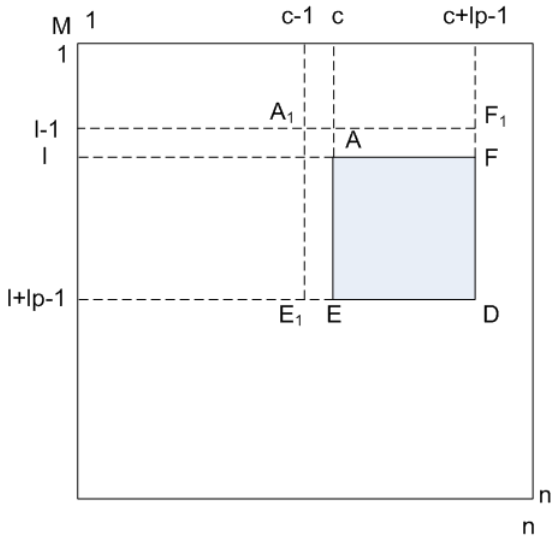

Fig. 2: Cum se poate calcula suma elementelor pătratului $AFDE$ cu ajutorul sumelor elementelor dreptunghiurilor de coordonate $M(1,1)$ - $D(l+lp-1, c+lp-1)$, $M(1,1)$ - $E_1(l+lp-1, c-1)$, $M(1,1)$ - $F_1(l-1, c+lp-1)$ şi $M(1,1)$ - $A_1(l-1, c-1)$

- pentru $k < 0$:

  Se identifică pătratul de latură maximă înscris în zona triunghiulară (l, c, k), având colţul din dreapta-jos în punctul (l, c). Latura acestui pătrat va avea valoarea:

  $$lp = |k|/2 + (|k| \mod 2) \tag{4}$$

  Suma elementelor aflate în zona pătratului de coordonate $(l - lp + 1, c - lp + 1)$ - (l, c) se determină astfel:

  $$sp = sd_{l,c} - sd_{l,c-lp} - sd_{l-lp,c} + sd_{l-lp,c-lp} \tag{5}$$

  Suma elementelor aflate în interiorul zonei triunghiulare va fi:

```
suma_zona_triunghiulara = sp
       + suma elementelor zonei triunghiulare (l, c - lp, lp - |k|)
       + suma elementelor zonei triunghiulare (l - lp, c, lp - |k|)
```

  Se observă caracterul recursiv al acestei formule.

Complexitatea algoritmului recursiv pentru determinarea sumei elementelor aflate în interiorul unei zone triunghiulare este $O(n)$: prin urmare, complexitatea întregului algoritm este $O(n^2 + Q \times n)$.

Codul sursă al programului este următorul:

Listing 2: `triunghi-v2.c`

```
#include <stdio.h>

#define NMAX 1000
```

```
int s[NMAX + 1][NMAX + 1];

int getsumtriangle(int l, int c, int k) {
  int lp, sp;

  if ((k == 1) || (k == -1)) {
    return (s[l][c] - s[l][c - 1] - s[l - 1][c] + s[l - 1][c - 1]);
  }

  if (k > 0) {
    lp = k / 2 + k % 2;

    sp = s[l + lp - 1][c + lp - 1] - s[l + lp - 1][c - 1] - s[l - 1][c + lp - 1];
    sp = sp + s[l - 1][c - 1];

    return (sp + getsumtriangle(l + lp, c, k - lp)
               + getsumtriangle(l, c + lp, k - lp));
  } else {
      k = -k;
      lp = k / 2 + k % 2;

      sp = s[l][c] - s[l][c - lp] - s[l - lp][c];
      sp = sp + s[l - lp][c - lp];

      return (sp + getsumtriangle(l, c - lp, lp - k)
                 + getsumtriangle(l - lp, c, lp - k));
  }
}

int main() {
  FILE *fin, *fout;
  int n, i, j, v, q, l, c, k, total, smax;

  fin = fopen("triunghi.in", "r");
  fout = fopen("triunghi.out", "w");

  fscanf(fin, "%d", &n);
  for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
      fscanf(fin, "%d", &v);

      s[i][j] = s[i][j - 1] + s[i - 1][j] + v - s[i - 1][j - 1];
    }
  }

  smax = 0;
  fscanf(fin, "%d", &q);
  for (j = 0; j < q; j++) {
    fscanf(fin, "%d %d %d", &l, &c, &k);

    total = getsumtriangle(l, c, k);
```

```
    if (total > smax) {
      smax = total;
    }
  }

  fprintf(fout, "%d\n", smax);

  fclose(fin);
  fclose(fout);

  return 0;
}
```

2.3 Varianta a III-a

Definim următoarele structuri de date:

- $sd_{l,c}$ - suma elementelor aflate în interiorul zonei dreptunghiulare de coordonate $(1,1)$ - (l,c).
- $tl_{i,k}$ - suma elementelor zonei triunghiulare de coordonate $(i,1)$ - (i,k) - $(i+k-1,1)$.
- $tr_{i,k}$ - suma elementelor zonei triunghiulare de coordonate (i,n) - $(i-k+1,n)$ - $(i,n-k+1)$.
- $tu_{j,k}$ - suma elementelor zonei triunghiulare de coordonate $(1,j)$ - $(1,j+k-1)$ - (k,j).
- $td_{j,k}$ - suma elementelor zonei triunghiulare de coordonate (n,j) - $(n-k+1,j)$ - $(n,j-k+1)$.

Matricele tl, tr, tu şi td reprezintă, fiecare, câte o jumătate dintr-o matrice pătratică. Spaţiul de memorie necesar este $\sim 3*n*n$ elemente (un element reprezintă o variabilă întreagă).

Elementele matricei sd se determină după formula:

$$sd_{i,j} = sd_{i,j-1} + sd_{i-1,j} + a_{i,j} - sd_{i-1,j-1} \tag{6}$$

Elementele matricei tl se determină cu ajutorul relaţiei următoare:

$$tl_{i-k+1,k} = tl_{i-k+2,k-1} + (sd_{i-k+1,k} - sd_{i-k,k}) \tag{7}$$

Prezenăm în continuare un fragment de cod folosit pentru a iniţializa valorile tabloului `tl`:

```
for (i = n; i > 0; i--) {
  tl[i][1] = sd[i][1] - sd[i - 1][1];
  for (k = 2; k <= i; k++) {
    tl[i - k + 1][k] = tl[i - k + 2][k - 1] + (sd[i - k + 1][k] - sd[i - k][k]);
  }
}
```

Elementele matricei tu se determină cu ajutorul relaţiei următoare:

$$tu_{j-k+1,k} = tu_{j-k+2,k-1} + (sd_{k,j-k+1} - sd_{k,j-k}) \tag{8}$$

Iată un fragment de cod folosit pentru a iniţializa valorile tabloului `tu`:

```
for (j = n; j > 0; j--) {
  tu[j][1] = sd[1][j] - sd[1][j - 1];
  for (k = 2; k <= j; k++) {
    tu[j - k + 1][k] = tu[j - k + 2][k - 1] + (sd[k][j - k + 1] - sd[k][j - k]);
  }
}
```

Elementele matricei *tr* se determină după formula:

$$tr_{i+k-1,k} = tr_{i+k-2,k-1} + ((sd_{i+k-1,n} - sd_{i+k-2,n}) - (sd_{i+k-1,n-k} - sd_{i+k-2,n-k})) \qquad (9)$$

Valorile tabloului `tr` se pot iniţializa prin intermediul fragmentului de cod următor:

```
for (i = 1; i <= n; i++) {
  tr[i][1] = sd[i][n] - sd[i][n - 1] - sd[i - 1][n] + sd[i - 1][n - 1];
  for (k = 2; k <= n - i + 1; k++) {
    line1 = sd[i + k - 1][n] - sd[i + k - 2][n];
    line2 = sd[i + k - 1][n - k] - sd[i + k - 2][n - k];

    tr[i + k - 1][k] = tr[i + k - 2][k - 1] + (line1 - line2);
  }
}
```

Elementele matricei *td* se determină cu ajutorul relaţiei următoare:

$$td_{j+k-1,k} = td_{j+k-2,k-1} + ((sd_{n,j+k-1} - sd_{n,j+k-2}) - (sd_{n-k,j+k-1} - sd_{n-k,j+k-2})); \qquad (10)$$

Prezenăm în continuare un fragment de cod folosit pentru a iniţializa valorile tabloului `td`:

```
for (j = 1; j <= n; j++) {
  td[j][1] = sd[n][j] - sd[n - 1][j] - sd[n][j - 1] + sd[n - 1][j - 1];
  for (k = 2; k <= n - j + 1; k++) {
    column1 = sd[n][j + k - 1] - sd[n][j + k - 2];
    column2 = sd[n - k][j + k - 1] - sd[n - k][j + k - 2];

    td[j + k - 1][k] = td[j + k - 2][k - 1] + (column1 - column2);
  }
}
```

Pentru o zonă triunghiulară `(l, c, k)`, se va calcula suma elementelor aflate în interiorul acelei zone triunghiulare astfel:

- pentru $k > 0$:

  Cazul 1 corespunde situaţiei în care $l + c + k - 1 \leq n + 1$ adică toate vârfurile zonei triunghiulare `EFG` se află deasupra diagonalei secundare (a se vedea figura 3).

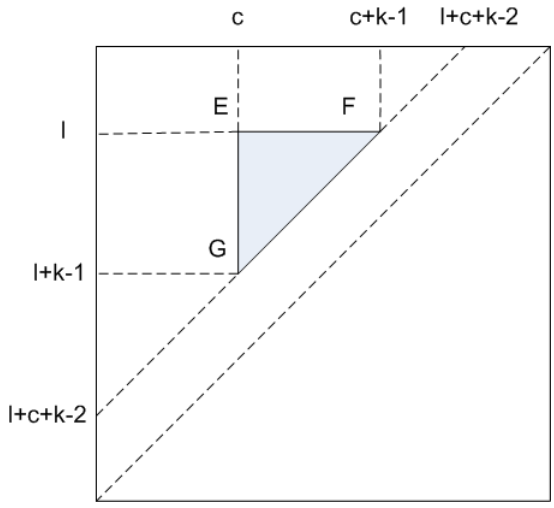

Fig. 3: Cazul 1: Toate vârfurile zonei triunghiulare se află deasupra diagonalei secundare.

În această situaţie vom proceda în felul următor: din suma elementelor zonei triunghiulare `ABC` (vârfurile au coordonatele $A(1,1)$, $B(l+c+k-2,1)$, $C(1,l+c+k-2)$) se va scădea suma elementelor zonelor dreptunghiulare D_1, D_2, D_3 şi suma elementelor zonelor triunghiulare T_2 şi T_3 (a se vedea figura 4).

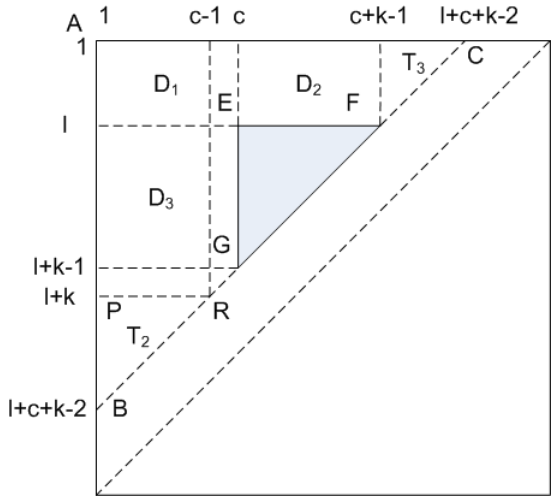

Fig. 4: Cazul 1: Toate vârfurile zonei triunghiulare se află deasupra diagonalei secundare - detaliat.

Suma elementelor zonelor dreptunghiulare D_1, D_2, D_3 se calculează astfel: suma elementelor dreptunghiului de coordonate $(1,1)$-$(l-1,c+k-1)$ plus suma elementelor dreptunghiului de coordonate $(1,1)$-$(l+k-1,c-1)$ din care se scade suma elementelor dreptunghiului de coordonate $(1,1)$-$(l-1,c-1)$:

$$S_{D_1} + S_{D_2} + S_{D_3} = sd_{l-1,c+k-1} + sd_{l+k-1,c-1} - sd_{l-1,c-1} \tag{11}$$

Suma elementelor zonei triunghiulare `ABC` corespunde valorii $tl_{1,l+c+k-2}$ (zona triunghiulară de coordonate $(1,1)$ - $(1,l+c+k-2)$ - $(l+c+k-2,1)$).

Zona triunghiulară T_2 este determinată de triunghiul PRB (vârfurile au coordonatele $P(l+k,1)$, $R(l+k,c-1)$, $B(l+c+k-2,1)$). Suma elementelor acestei zone triunghiulare corespunde valorii $tl_{l+k,c-1}$.

În mod asemănător, suma elementelor zonei triunghiulare T_3 corespunde valorii $tu_{c+k,l-1}$.

$$S_{EFG} = tl_{1,l+c+k-2} - (sd_{l-1,c+k-1} + sd_{l+k-1,c-1} - sd_{l-1,c-1}) - tl_{l+k,c-1} - tu_{c+k,l-1} \quad (12)$$

```
suma_zona_triunghiulara = tl[1][l + c + k - 2]
                        - (sd[l - 1][c + k - 1] + sd[l + k - 1][c - 1]
                           - sd[l - 1][c - 1])
                        - tl[l + k][c - 1]
                        - tu[c + k][l - 1]
```

Cazul al 2*-lea* corespunde situaţiei în care $l+c+k-1 > n+1$ adică cele două puncte, F şi G, ale laturii paralele cu diagonala secundară se află poziţionate sub diagonala secundară (a se vedea figura 5).

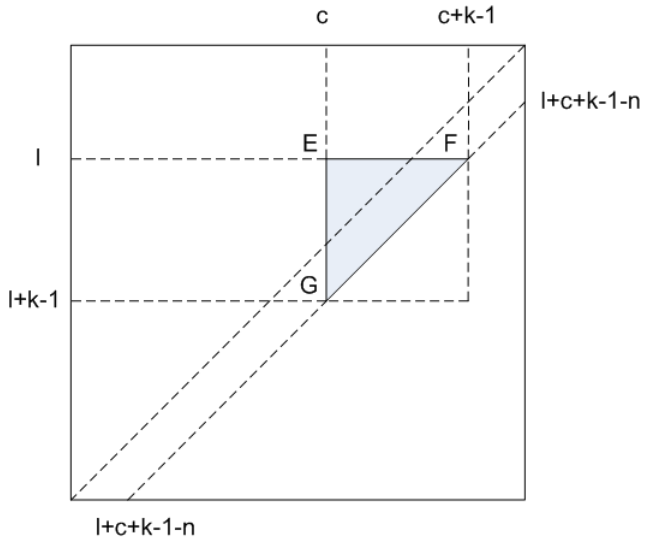

Fig. 5: Cazul 2: Cele două puncte, F şi G, ale laturii paralele cu diagonala secundară se află poziţionate sub diagonala secundară.

Pentru a calcula suma elementelor zonei triunghiulare EFG se va proceda astfel: din suma elementelor zonei dreptunghiulare (pătrată) EFHG de coordonate $E(l,c)$-$H(l+k-1,c+k-1)$, se va scădea suma elementelor zonei triunghiulare HVU (a se vedea figura 6).

$$S_{\triangle EFG} = S_{EFHG} - S_{\triangle HVU} \quad (13)$$

Suma elementelor zonei dreptunghiulare EFHG se determină astfel:

$$S_{EFHG} = sd_{l+k-1,c+k-1} - sd_{l-1,c+k-1} - sd_{l+k-1,c-1} + sd_{l-1,c-1} \quad (14)$$

Suma elementelor zonei triunghiulare HVU se va calcula folosind formulele prezentate pentru Cazul al 3-lea (vârfurile zonei triunghiulare HVU au coordonatele $H(l+k-1,c+k-1)$-$V(l+1,c+k-1)$-$U(l+k-1,c+1)$).

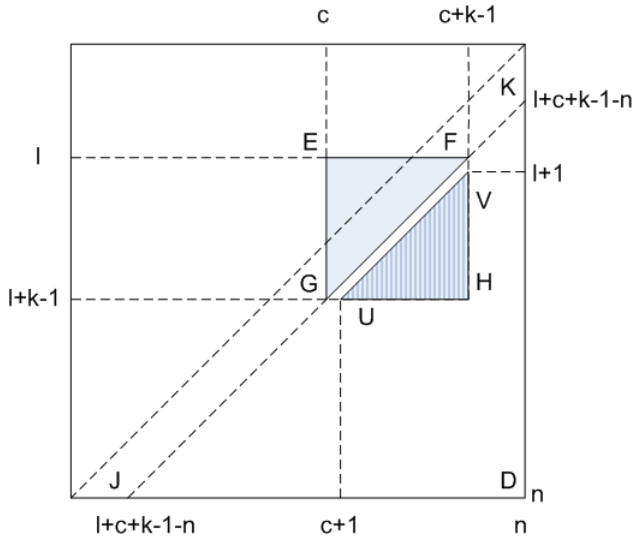

Fig. 6: Cazul 2: Cele două puncte, F şi G, ale laturii paralele cu diagonala secundară se află sub diagonala secundară - detaliu

```
suma_zona_triunghiulara = sd[l + k - 1][c + k - 1] - sd[l - 1][c + k - 1]
                          - sd[l + k - 1][c - 1] + sd[l - 1][c - 1]
                - suma_zona_triunghiulara (l + k - 1, c + k - 1, -(k - 1))
```

În concluzie, codul ce tratează cele două situaţii menţionate (Cazul 1 şi Cazul 2) este următorul:

```
if (l + c + k - 1 <= n + 1) { // deasupra diagonalei secundare
  total = sd[l - 1][c + k - 1] + sd[l + k - 1][c - 1] - sd[l - 1][c - 1];
  total = tl[1][l + c + k - 2] - total
          - ((l + k <= n) ? tl[l + k][c - 1] : 0)
          - ((c + k <= n) ? tu[c + k][l - 1] : 0);
} else { // sub diagonala secundara
    total = getsumtriangle(l + k - 1, c + k - 1, -(k - 1), n);

    total = sd[l + k - 1][c + k - 1] - sd[l - 1][c + k - 1]
            - sd[l + k - 1][c - 1] + sd[l - 1][c - 1] - total;
}
```

- pentru $k < 0$:

  Cazul al 3*-lea* corespunde situaţiei în care $l + c - k + 1 > n + 1$ adică toate vârfurile zonei triunghiulare `EFG` se află sub diagonala secundară (a se vedea figura 7).

  Din suma elementelor zonei triunghiulare `XYZ` se va scade suma elementelor zonelor dreptunghiulare D_4, D_5 şi D_6 şi suma elementelor zonelor triunghiulare T_5 şi T_6 (a se vedea figura 8).

$$S_{\triangle EFG} = S_{\triangle XYZ} - S_{D_4} - S_{D_5} - S_{D_6} - S_{T_5} - S_{T_6} \tag{15}$$

  Coordonatele zonei triunghiulare `XYZ` sunt $X(n, n)$-$Y(n, l + c - k + 1 - n)$-$Z(l + c - k + 1 - n, n)$). Suma elementelor acestei zone triunghiulare `XYZ` corespunde valorii $tr_{n,2*n-l-c+k}$. Reamintim că $tr_{i,k}$ se defineşte ca suma elementelor zonei triunghiulare

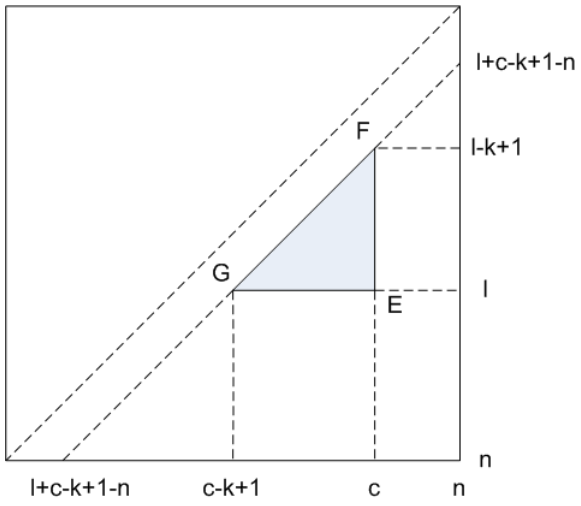

Fig. 7: Cazul 3: toate vârfurile zonei triunghiulare `EFG` se află sub diagonala secundară.

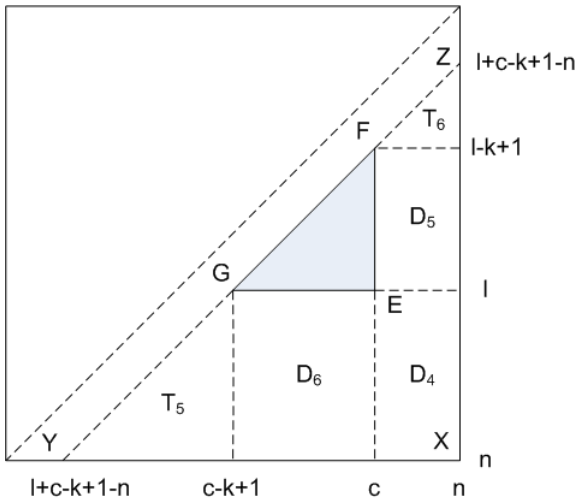

Fig. 8: Cazul 3: toate vârfurile zonei triunghiulare `EFG` se află sub diagonala secundară - detaliat.

de coordonate (i, n) - $(i-k+1, n)$ - $(i, n-k+1)$). Atunci pentru notaţiile $i = n$ şi $k = 2*n-l-c+k$ obţinem coordonatele zonei triunghiulare: $X(n, n)$ - $Y(n, l+c-k+1-n)$ - $Z(l+c-k+1-n, n)$.

$$S_{\triangle XYZ} = tr_{n,2*n-l-c+k} \tag{16}$$

Suma elementelor zonelor dreptunghiulare D_4, D_5, D_6 se calculează astfel: suma elementelor dreptunghiului de coordonate $(l-k+1, c+1)$-(n, n) plus suma elementelor dreptunghiului de coordonate $(l+1, c-k+1)$-(n, n) din care se scade suma elementelor dreptunghiului de coordonate $(l+1, c+1)$-(n, n):

$$S_{D_4} + S_{D_5} = sd_{n,n} - sd_{l-k,n} - sd_{n,c} + sd_{l-k,c} \tag{17}$$

$$S_{D_4} + S_{D_6} = sd_{n,n} - sd_{l,n} - sd_{n,c-k} + sd_{l,c-k} \tag{18}$$

$$S_{D_4} = sd_{n,n} - sd_{l,n} - sd_{n,c} + sd_{l,c}. \tag{19}$$

Zona triunghiulară T_5 este determinată de triunghiul `WQY` (vârfurile au coordonatele

$W(n, c-k)$, $Q(l+1, c-k)$, $Y(n, l+c-k+1-n)$). Suma elementelor acestei zone triunghiulare corespunde valorii $td_{c-k,n-1}$.

$$S_{T_5} = S_{\triangle WQY} = td_{c-k,n-1} \tag{20}$$

În mod asemănător, suma elementelor zonei triunghiulare T_6 corespunde valorii $tr_{l-k,n-c}$.

$$S_{T_6} = tr_{l-k,n-c} \tag{21}$$

Prin urmare, ecuaţia 15 devine:

$$S_{EFG} = tr_{n,2*n-l-c+k} - (S_{D_4} + S_{D_5} + S_{D_6}) - tr_{l-k,n-c} - td_{c-k,n-l} \tag{22}$$

```
suma_zona_triunghiulara = tr[n][2 * n - l - c + k]
                        - (getsumrectangle(l - k + 1, c + 1)
                           + getsumrectangle(l + 1, c - k + 1)
                           - getsumrectangle(l + 1, c + 1, n))
                        - tr[l - k][n - c]
                        - td[c - k][n - l]
```

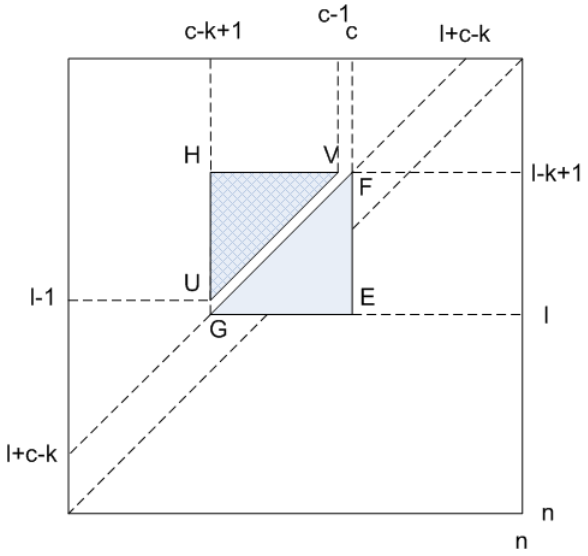

Fig. 9: Cazul 4: Cele două puncte, F şi G, ale laturii paralele cu diagonala secundară se află poziţionate deasupra diagonalei secundare.

Cazul al 4*-lea* corespunde situaţiei în care $l+c-k+1 \leq n+1$ adică cele două puncte, F şi G, ale laturii paralele cu diagonala secundară se află poziţionate deasupra diagonalei secundare (a se vedea figura 9).

Pentru a calcula suma elementelor zonei triunghiulare `EFG` se va proceda astfel: din suma elementelor zonei dreptunghiulare (pătrată) `EFHG` de coordonate $H(l-k+1, c-k+1)$-$E(l, c)$, se va scădea suma elementelor zonei triunghiulare `HVU`.

$$S_{\triangle EFG} = S_{EFHG} - S_{\triangle HVU} \tag{23}$$

Suma elementelor zonei dreptunghiulare `EFHG` se determină astfel:

$$S_{EFHG} = sd_{l,c} - sd_{l-k,c} - sd_{l,c-k} + sd_{l-k,c-k} \tag{24}$$

Suma elementelor zonei triunghiulare `HVU` se va calcula folosind formulele prezentate pentru *Cazul 1* (vârfurile zonei triunghiulare `HVU` au coordonatele $H(l+k-1, c+k-1)$ - $V(l-k+1, c-1)$ - $U(l-1, c-k+1)$).

```
suma_zona_triunghiulara = sd[l][c] - sd[l - k][c]
                                   - sd[l][c - k] + sd[l - k][c - k]
                        - suma_zona_triunghiulara (l - k + 1, c - k + 1, k - 1)
```

În concluzie, codul ce tratează cele două situaţii menţionate mai sus, *Cazul 3* şi *Cazul 4*, este următorul:

```
k = -k;
if (l + c - k + 1 <= n + 1) { // deasupra diagonalei secundare
  total = getsumtriangle(l - k + 1, c - k + 1, k - 1, n);
  total = sd[l][c] - sd[l - k][c] - sd[l][c - k] + sd[l - k][c - k] - total
     ;
} else { // sub diagonala secundara
    total = getsumrectangle(l - k + 1, c + 1, n)
          + getsumrectangle(l + 1, c - k + 1, n)
          - getsumrectangle(l + 1, c + 1, n);
    total = tr[n][2 * n - l - c + k] - total
          - ((l - k > 0) ? tr[l - k][n - c] : 0)
          - ((c - k > 0) ? td[c - k][n - l] : 0);
}
```

Complexitatea algoritmului este $O(n^2 + Q)$.
Codul sursă al programului este următorul:

Listing 3: `triunghi-v3.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NMAX 1000

int sd[NMAX + 1][NMAX + 1];
int** tl;
int** tr;
int** tu;
int** td;

int** alloc_upper_triangle(int n) {
  int i;
  int **a;

  a = (int**)malloc(sizeof(int*) * (n + 1));
  for (i = 0; i <= n; i++) {
```

```
    a[i] = (int*)malloc(sizeof(int) * (n - i + 1));
    memset(a[i], 0, sizeof(int) * (n - i + 1));
  }

  return a;
}

int** alloc_lower_triangle(int n) {
  int i;
  int **a;

  a = (int**)malloc(sizeof(int*) * (n + 1));
  for (i = n; i >= 0; i--) {
    a[i] = (int*)malloc(sizeof(int) * (i + 1));
    memset(a[i], 0, sizeof(int) * (i + 1));
  }

  return a;
}

int getsumrectangle(int l, int c, int n) {
  if ((l < 1) || (l > n) || (c < 1) || (c > n)) {
    return 0;
  } else {
      return (sd[n][n] - sd[l - 1][n] - sd[n][c - 1] + sd[l - 1][c - 1]);
  }
}

int getsumtriangle(int l, int c, int k, int n) {
  int total;

  if (k > 0) {
    if (l + c + k - 1 <= n + 1) { // deasupra diagonalei secundare
      total = sd[l - 1][c + k - 1] + sd[l + k - 1][c - 1] - sd[l - 1][c - 1];
      total = tl[1][l + c + k - 2] - total
              - ((l + k <= n) ? tl[l + k][c - 1] : 0)
              - ((c + k <= n) ? tu[c + k][l - 1] : 0);
    } else { // sub diagonala secundara
        total = getsumtriangle(l + k - 1, c + k - 1, -(k - 1), n);

        total = sd[l + k - 1][c + k - 1] - sd[l - 1][c + k - 1]
                - sd[l + k - 1][c - 1] + sd[l - 1][c - 1] - total;
    }
  } else {
      k = -k;
      if (l + c - k + 1 <= n + 1) { // deasupra diagonalei secundare
        total = getsumtriangle(l - k + 1, c - k + 1, k - 1, n);
        total = sd[l][c] - sd[l - k][c] - sd[l][c - k] + sd[l - k][c - k] - total;
      } else { // sub diagonala secundara
          total = getsumrectangle(l - k + 1, c + 1, n)
                  + getsumrectangle(l + 1, c - k + 1, n)
                  - getsumrectangle(l + 1, c + 1, n);
```

```
          total = tr[n][2 * n - l - c + k] - total
                  - ((l - k > 0) ? tr[l - k][n - c] : 0)
                  - ((c - k > 0) ? td[c - k][n - l] : 0);
      }
  }

  return total;
}

int main() {
  FILE *fin, *fout;
  int n, i, j, v, k, q, l, c, total, smax, line1, line2, column1, column2;

  fin = fopen("triunghi.in", "r");
  fout = fopen("triunghi.out", "w");

  fscanf(fin, "%d", &n);
  for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
      fscanf(fin, "%d", &v);

      sd[i][j] = sd[i][j - 1] + sd[i - 1][j] + v - sd[i - 1][j - 1];
    }
  }

  tl = alloc_upper_triangle(n);
  tu = alloc_upper_triangle(n);

  for (i = n; i > 0; i--) {
    tl[i][1] = sd[i][1] - sd[i - 1][1];
    for (k = 2; k <= i; k++) {
      tl[i - k + 1][k] = tl[i - k + 2][k - 1] + (sd[i - k + 1][k] - sd[i - k][k]);
    }
  }

  for (j = n; j > 0; j--) {
    tu[j][1] = sd[1][j] - sd[1][j - 1];
    for (k = 2; k <= j; k++) {
      tu[j - k + 1][k] = tu[j - k + 2][k - 1] + (sd[k][j - k + 1] - sd[k][j - k]);
    }
  }

  tr = alloc_lower_triangle(n);
  td = alloc_lower_triangle(n);

  for (i = 1; i <= n; i++) {
    tr[i][1] = sd[i][n] - sd[i][n - 1] - sd[i - 1][n] + sd[i - 1][n - 1];
    for (k = 2; k <= n - i + 1; k++) {
      line1 = sd[i + k - 1][n] - sd[i + k - 2][n];
      line2 = sd[i + k - 1][n - k] - sd[i + k - 2][n - k];

      tr[i + k - 1][k] = tr[i + k - 2][k - 1] + (line1 - line2);
```

```
    }
  }

  for (j = 1; j <= n; j++) {
    td[j][1] = sd[n][j] - sd[n - 1][j] - sd[n][j - 1] + sd[n - 1][j - 1];
    for (k = 2; k <= n - j + 1; k++) {
      column1 = sd[n][j + k - 1] - sd[n][j + k - 2];
      column2 = sd[n - k][j + k - 1] - sd[n - k][j + k - 2];

      td[j + k - 1][k] = td[j + k - 2][k - 1] + (column1 - column2);
    }
  }

  smax = 0;
  fscanf(fin, "%d", &q);
  for (j = 0; j < q; j++) {
    fscanf(fin, "%d %d %d", &l, &c, &k);

    total = getsumtriangle(l, c, k, n);

    printf("%d\n", total);

    if (total > smax) {
      smax = total;
    }
  }

  fprintf(fout, "%d\n", smax);

  fclose(fin);
  fclose(fout);

  return 0;
}
```

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introducere în Algoritmi*, Computer Libris Agora, Cluj-Napoca, 1999.

[2] M. Coşulschi, M. Gabroveanu, *Practica programării în C*, Editura Universitaria, Craiova, 2014.