Subsecvenţa de sumă maximă

Mirel Coşulschi

April, 2023

1 Enunţ problema

Se dă un vector de lungime N, $a_1, a_2, \ldots, a_n$ cu elemente numere întregi ($a_i \in \mathbb{Z}$). Să se determine o subsecvenţă a sa de suma maximă.

O subsecvenţă a unui vector reprezintă un subşir de elemente ale acestuia aflate pe poziţii consecutive (o subsecvenţă a şirului este de forma $< a_i, a_{i+1}, \ldots, a_j >$ cu $1 \leq i \leq j \leq n$, iar suma subsecvenţei este $a_i + a_{i+1} + \ldots + a_j$).

Se va citi de la tastatură, pe prima linie, un număr natural n, reprezentând lungimea şirului. Următoarea linie va conţine n numere întregi separate printr-un spaţiu, reprezentând, în ordine, elementele şirului.

$1 \leq n \leq 100.000$ iar elementele şirului vor avea cel mult 4 cifre. Dacă şirul conţine mai multe secvenţe de sumă maximă, se va determina cea cu indicele de început cel mai mic, iar în caz de egalitate, cea mai scurtă.

2 Modalităţi de rezolvare

2.1 Varianta I

Se vor genera toate subsecvenţele şirului de forma $< a_i, a_{i+1}, \ldots, a_j >$ cu $1 \leq i \leq j \leq n$, şi vom calcula suma fiecărei subsecvenţe; se va determina suma de valoare maximă, şi vom păstra valoarea maximă şi coordonatele primului şi ultimului element al subsecvenţei de sumă maximă.

```
#include <stdio.h>

#define NMAX 100001

short a[NMAX];   // vector ce contine elementele sirului

int main() {
  int n,         // numarul de elemente al sirului
      i, j, k,   // variabile folosite in instructiunile de ciclare
      total,     // suma elementelor unei secvente
      smax,      // suma de valoare maxima a unei secvente
      left,      // indicele primului element al secventei de suma maxima
      right;     // indicele ultimului element al secventei de suma maxima

  // se citeste dimensiunea sirului
  scanf("%d", &n);
```

```
  // se citesc elementele sirului
  for (i = 1; i <= n; i++) {
    scanf("%hd", &a[i]);
  }

  smax = a[1];
  left = right = 1;
  for (i = 1; i <= n; i++) {   // limita din stanga a secventei
    for (j = i; j <= n; j++) { // limita din dreapta a secventei
      // se calculeaza suma elementelor din secventa a[i]...a[j]
      total = 0;
      for (k = i; k <= j; k++) {
        total += a[k];
      }

      // se determina suma de valoare maxima a unei secvente
      if (total > smax) {
        smax = total;
        left = i;
        right = j;
      }
    }
  }

  printf("%d %d", left, right);

  return 0;
}
```

Algoritmul corespunzător primei variante are o complexitate-timp de $O(n^3)$.

2.2 Varianta a II-a

Pentru cea de-a doua variantă de rezolvare, se vor calcula următoarele sume parţiale a_1, a_1+a_2, $a_1+a_2+a_3$, ..., $a_1+a_2+\ldots+a_n$ corespunzătoare subsecvenţelor $< a_1 >$, $< a_1, a_2 >$, $< a_1, a_2, a_3 >$, ..., $< a_1, a_2, \ldots, a_n >$:

$$sum_j = \sum_{k=1}^{j} a_j, \forall j = \overline{1, n}. \tag{1}$$

Pe baza acestor valori, se poate determina suma valorilor elementelor oricărei subsecvenţe $< a_i, a_{i+1}, \ldots, a_j >$ astfel:

$$\sum_{k=i}^{j} a_k = sum_j - sum_{i-1},\ 1 \leq i \leq j \leq n. \tag{2}$$

```
#include <stdio.h>

#define NMAX 100001

short a[NMAX];
```

```
int sum[NMAX];    // vector ce contine valorile sumelor partiale

int main() {
  int n, i, j, total, smax, left, right;

  scanf("%d", &n);
  for (i = 1; i <= n; i++) {
    scanf("%hd", &a[i]);

    sum[i] = sum[i - 1] + a[i];
  }

  smax = a[1];
  left = right = 1;
  for (i = 1; i <= n; i++) {   // limita din stanga a secventei
    for (j = i; j <= n; j++) { // limita din dreapta a secventei
      // se calculeaza suma elementelor din secventa a[i]...a[j]
      total = sum[j] - sum[i - 1];

      // se determina suma de valoare maxima
      if (total > smax) {
        smax = total;
        left = i;
        right = j;
      }
    }
  }

  printf("%d %d", left, right);

  return 0;
}
```

Algoritmul pentru cea de-a doua variantă are o complexitate-timp de $O(n^2)$. Se observă că, în contextul cerinţelor actuale ale problemei, nu este necesar să memorăm elementele şirului în vectorul a.

2.3 Varianta a III-a

Vom prezenta un alt algoritm de rezolvare pentru aceeaşi problemă, cunoscut sub numele de *algoritmul lui Kadane*[1], algoritm ce are o complexitate-timp liniară $O(n)$.

Fie următoarele notaţii:

- $smax_i$ = suma maximă pentru o subsecvenţă a şirului $a_1, a_2, \ldots, a_i$,
- $slast_i$ = suma maximă dintre toate subsecvenţele şirului $a_1, a_2, \ldots, a_i$, subsecvenţe care au limita din dreapta fixată pe poziţia i ($< a_i >$, $< a_{i-1}, a_i >$, $< a_{i-2}, a_{i-1}, a_i >$, ..., $< a_1, a_2, \ldots, a_i >$).

Să considerăm acum un şir $a_1, a_2, \ldots, a_n$, şi să începem evaluarea elementelor $smax_i$, $slast_i$, $i = \overline{1, n}$, conform definiţiei anterioare:

[1] https://en.wikipedia.org/wiki/Maximum_subarray_problem

- pentru $i = 1$, avem secvenţa $< a_1 >$: $slast_1 = a_1$ şi $smax_1 = a_1$.
- pentru $i = 2$, avem secvenţa $< a_1, a_2 >$: $slast_2 = \max\{a_1 + a_2, a_2\}$, $smax_2 = \max\{a_1, a_1 + a_2, a_2\}$.
  Pe baza valorilor calculate la pasul anterior ($i = 1$) avem: $slast_2 = \max\{slast_1 + a_2, a_2\}$ şi $smax_2 = \max\{smax_1, slast_2\}$.
- pentru $i = 3$, avem secvenţa $< a_1, a_2, a_3 >$: $slast_3 = \max\{a_1 + a2 + a_3, a_2 + a_3, a_3\}$, $smax_3 = \max\{a_1, a_1 + a_2, a_1 + a_2 + a_3, a_2, a_2 + a_3, a_3\}$.
  Pe baza valorilor calculate la pasul anterior avem:
  $slast_3 = \max\{\max\{a_1 + a_2 + a_3, a_2 + a_3\}, a_3\} = \max\{\max\{a_1 + a_2, a_2\} + a_3, a_3\}$ adică
  $slast_3 = \max\{slast_2 + a_3, a_3\}$.
  Analog $smax_3 = \max\{\max\{a_1, a_1+a_2, a_2\}, \max\{a_1+a_2+a_3, a_2+a_3, a_3\}\} = \max\{smax_2, slast_3\}$.
- etc.

Se determină următoarele relaţii de recurenţă:

$$\begin{aligned} slast_i &= \max\{slast_{i-1} + a_i, a_i\}, \\ smax_i &= \max\{smax_{i-1}, slast_i\}. \end{aligned}$$

Se observă că nu este nevoie să păstrăm toate valorile şirurilor $slast_i$ şi $smax_i$, deoarece elementul curent se calculează doar pe baza valorilor anterioare, iar pentru rezultatul problemei avem nevoie de ultima valoare a lui $smax$.

Prin urmare relaţiile de recurenţă anterioare devin:

$$\begin{aligned} slast &= \max\{slast + a_i, a_i\}, \\ smax &= \max\{smax, slast\}. \end{aligned}$$

```
#include <stdio.h>

#define INF 10000

int main() {
  int n,          // numarul de elemente al sirului
      i,          // variabila folosita in instructiunea de ciclare
      v,          // valoarea unui element al sirului
      smax,       // totalul elementelor unei secvente de suma maxima
      slast,      // suma elementelor subsecventei de suma maxima ce se termina
                  // pe pozitia curenta
      start,      // indicele primului element al secventei de suma maxima
                  // ce se termina pe pozitia curenta
      left,       // indicele primului element al secventei de suma maxima
      right;      // indicele ultimului element al secventei de suma maxima

  scanf("%d", &n);

  smax = slast = -INF;
  for (i = 1; i <= n; i++) {
    scanf("%d", &v);

    if (slast + v >= v) { // daca elementul curent v poate fi adaugat la secventa
```

```
                        // a[k]...a[i-1] a carei suma este pastrata in slast
      slast += v;
    } else {            // altfel pe pozitia curenta incepe o noua subsecventa
        slast = v;      // suma elementelor din subsecventa de suma maxima curenta,
                        // formata dintr-un singur element
        start = i;      // subsecventa de suma maxima incepe cu pozitia curenta
    }

    if (smax < slast) {// daca subsecventa care se termina pe pozitia curenta este
                        // mai mare decat valoarea maxima anterioara
      smax = slast;
      left = start;
      right = i;
    }
  }

  printf("%d %d", left, right);

  return 0;
}
```

3 Probleme

- `https://www.infoarena.ro/problema/ssm`
- `https://www.pbinfo.ro/probleme/297/secvsummax`
- `https://www.nerdarena.ro/problema/ssm`
- `https://www.nerdarena.ro/problema/colier`

  Aplicăm algoritmul lui Kadane de două ori: pentru a determina subsecvenţa de sumă maximă şi subsecvenţa de sumă minimă.

  Subsecvenţa circulară de sumă maximă poate fi o subsecvenţă de sumă maximă care nu este circulară sau o subsecvenţă circulară ce include elementele de la capete (de pe pozitiile n sau 1).

  Pentru a determina subsecvenţa de sumă maximă ce include elementele de pe poziţiile n sau 1, vom determina secvenţa de sumă minimă şi o vom scade din suma tuturor elementelor şirului.

  Pentru a determina subsecvenţa de sumă minimă, aplicăm algoritmul lui Kadane pentru şirul formată din aceleaşi elemente, dar cu semn schimbat.

  Vom folosi următoarele notaţii:

  – `maxsumlast` - subsecvenţa de sumă maximă ce include elementul curent;
  – `maxsumt` - subsecvenţa de sumăa maximă identificată până la momentul curent;
  – `s` - suma tuturor elementelor din secvenţă;
  – `hasZero` - indică existenţa unui element de valoare zero în secvenţă;
  – `minsumlast` - subsecvenţa de sumă minimă ce include elementul curent;

– `minsumt` - subsecvenţa de sumă minimă identificată până la momentul curent;

```
#include <stdio.h>

int main() {
  FILE *fin, *fout;
  int n, v, i, maxsumlast, minsumlast, maxsumt, minsumt, s, hasZero;

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

  fscanf(fin, "%d", &n);

  fscanf(fin, "%d", &v);

  maxsumlast = v;
  maxsumt = maxsumlast;

  minsumlast = -v;
  minsumt = minsumlast;

  s = v;
  hasZero = (v == 0);
  for (i = 1; i < n; i++) {
    fscanf(fin, "%d", &v);

    if (maxsumlast + v < v) {
      maxsumlast = v;
    } else {
        maxsumlast += v;
    }

    if (maxsumlast > maxsumt) {
      maxsumt = maxsumlast;
    }

    if (minsumlast - v < -v) {
      minsumlast = -v;
    } else {
        minsumlast -= v;
    }

    if (minsumlast > minsumt) {
      minsumt = minsumlast;
    }

    s += v;
    if (!v) {
      hasZero = 1;
    }
  }
```

```
  if ((!hasZero && (s + minsumt == 0)) || (maxsumt > s + minsumt)) {
    fprintf(fout, "%d\n", maxsumt);
  } else {
      fprintf(fout, "%d\n", s + minsumt);
  }

  fclose(fin);
  fclose(fout);

  return 0;
}
```

- `https://www.nerdarena.ro/problema/colier2`
- `https://www.pbinfo.ro/probleme/3844/ksum`

  Vom folosi următoarele notaţii:

  – `ssmk` - subsecvenţa de sumă maximă de lungime cel puţin k din intervalul curent;

  – `ssmkfi` - subsecvenţa de sumă maximă de lungime cel puţin k ce are drept ultim element pe $a[i]$;

  Se determină următoarele relaţii de recurentă:

  – $ssmkfi = max$(subsecvenţa de sumă maximă de lungime cel puţin k ce are drept ultim element pe $a[i-1] + a[i]$, suma elementelor din subsecvenţa de lungime k ce are drept ultim element pe $a[i]$);

  – $ssmk = max$(subsecvenţa de sumă maximă de lungime cel puţin k din intervalul anterior, subsecvenţa de sumă maximă de lungime cel puţin k ce are drept ultim element pe $a[i]$);

```
#include <stdio.h>

#define NMAX 100000

int a[NMAX + 1];
int s[NMAX + 1]; // s[i] - suma elementelor secventei a[1]...a[i]

int max(int u, int v) {
  return (u > v) ? u : v;
}

int main() {
  FILE *fin, *fout;
  int n, k, i, ssmk, ssmkfi;

  fin = fopen("ksum.in", "rt");
  fout = fopen("ksum.out", "wt");

  fscanf(fin, "%d %d", &n, &k);
  for (i = 1; i <= n; i++) {
    fscanf(fin, "%d", &a[i]);
```

```
  }

  for (i = 1; i <= n; i++) {
    s[i] = s[i - 1] + a[i];
  }

  ssmk = ssmkfi = s[k];
  for (i = k + 1; i <= n; i++) {
    ssmkfi = max(ssmkfi + a[i], s[i] - s[i - k]);

    ssmk = max(ssmk, ssmkfi);
  }

  fprintf(fout, "%d", ssmk);

  fclose(fin);
  fclose(fout);

  return 0;
}
```

- `https://www.pbinfo.ro/probleme/3410/submatrixsummax`

  Vom folosi următoarele notaţii:

  – `a[i][j]` - valoarea elementului de pe linia i şi coloana j;
  – `t[i][j]` - suma elementelor de pe coloana j, de la linia 1 la linia j: $a[1][j] + ... + a[i][j]$;
  – `b[c]` - suma elementelor de pe coloana c situate între linia i şi linia k.

  i şi k sunt două linii ale matricei între care se determină submatricea cu suma elementelor maximă.

  Se aplică algoritmul lui Kadane pentru determinare subsecvenţei de sumă maximă a unui şir de numere. Şirul de numere este şirul b.

```
#include <stdio.h>

#define NMAX 300

char a[NMAX + 1][NMAX + 1];
int t[NMAX + 1][NMAX + 1];
int b[NMAX + 1];

int max(int u, int v) {
  return (u < v) ? v : u;
}

int main() {
  int n, i, j, k, v, smax, lmax;

  scanf("%d", &n);
  for (i = 1; i <= n; i++) {
```

```
    for (j = 1; j <= n; j++) {
      scanf("%d", &v);

      a[i][j] = v;
    }
  }

  for (j = 1; j <= n; j++) {
    t[1][j] = a[1][j];
  }

  for (j = 1; j <= n; j++) {
    for (i = 2; i <= n; i++) {
      t[i][j] = a[i][j] + t[i - 1][j];
    }
  }

  smax = a[1][1];

  for (i = 1; i <= n; i++) {
    for (k = i; k <= n; k++) {
      // b[j] - suma elementelor de pe coloana j situate intre liniile i si k
      for (j = 1; j <= n; j++) {
        b[j] = t[k][j] - t[i - 1][j];
      }

      // aplicam algoritmul lui Kadane pentru a gasi subsecventa de suma
      // maxima din sirul b
      // lmax - suma maxima a unei subsecvente ce se termina cu b[j]
      // smax - suma maxima a unei subsecvente a sirului b[1]....b[j]
      smax = max(smax, b[1]);
      lmax = b[1];
      for (j = 2; j <= n; j++) {
        lmax = max(lmax + b[j], b[j]);

        smax = max(smax, lmax);
      }
    }
  }

  printf("%d", smax);

  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.

[3] M. Coşulschi, *Algoritmi fundamentali. Proiectare şi implementare*, Editura Universitaria, Craiova, 2015.