**ДИНАМИЧНО ОПТИМИРАНЕ**

**Зад. 1. Най-голяма сума в подредица**

Дадена е редица от числа. Да се намери най-голямата сума в подредица.

Тази задача може да се реши с наивен алгоритъм и с динамично оптимиране. Разликата в скоростта на изпълнение при голям брой входни данни е очевидна. При бавния алгоритъм със сложност О(n2) и при големи стойности на n, например n=500000, програмата работи около 20 мин. При същия брой елементи динамичния алгоритъмът се изпълнява за по-малко от секунда. Можем да сравним работата на алгоритмите с голям брой случайни числа. Изискват #include <cstdlib> и #include <ctime>. В main() добавяме srand(time (NULL)); и в цикъла a[i]=rand();

Наивното решение за всеки елемент образува всички суми TempSum от него до края и всеки път сравнява с максималната сума, която първоначално е равна на първия елемент на масива. Ако намери по-голяма текуща сума, я запомня в MaxSum.

#include <iostream>

using namespace std;

int main()

{

int a[500000];

int n, i; cin>>n;

for (i=0; i<n; i++)

cin >> a[i];

int MaxSum =a[0];

for (i=0; i<n; i++)

{

int TempSum=0;

for (int j = i; j<n;j++)

{

TempSum = TempSum+a[j];

if ( TempSum > MaxSum ) MaxSum = TempSum;

}

}

cout << MaxSum<<endl;

return 0;

}

С техниката динамичното оптимиране елементите се обхождат веднъж и текущата им сума се натрупва в TempSum. Добавяме следващ елемент към сумата, само ако това прави временната сума по-голяма от текущия елемент. Иначе от текущия елемент започва изчисляване на нова текуща сума /на нова подредица/. При всяка промяна на текущата сума тя се сравнява с максималната.

#include <iostream>

using namespace std;

int main()

{

int a[500000];

int n, i; cin>>n;

for (i=0; i<n; i++)

cin >> a[i];

int MaxSum=a[0];

int TempSum=a[0]; //или =0, и цикълът е от 0

for (i=1; i<n; i++)

{

TempSum=max(TempSum+a[i],a[i]);

MaxSum=max(MaxSum, TempSum);

}

cout << MaxSum<<endl;

return 0;

}

ІІ начин:

if (TempSum + a[i] >MaxSum)

TempSum = TempSum + a[i], //добавяме следващ елемент

else TempSum = a[i] // започваме ново натрупване на текуща сума

ІІІ начин:

TempSum = (TempSum + a[i] >MaxSum)? TempSum + a[i]:a[i];

**Зад. 2. Най-малка сума в подредица**

Дадена е редица от числа. Да се намери най-малката сума в подредица.

Упътване:

І начин:

TempSum = min(TempSum+a[i],a[i]);

MinSum = min(MinSum, TempSum);

ІІ начин: От задача 1, при въвеждане на елементите на масива ги умножаваме по -1 и след това извеждаме с знак минус получената максимална сума.

**Зад. 3. Сума на подредица, най-близка до 0**

Дадена е редица от числа. Да се намери най-близката до 0 сума на подредица.

Ако елементите са а1, а2, а3, .. аn, създаваме масив с, така че с0=0, сi = a1+a2 …+ai, т.е. ci = ci-1 + ai.

Тогава: cn – cm = a1 + a2 + … + an – (a1 + a2 +… + am) = am+1 + … + an.

Например: с6 – с3 = а1 + а2 + а3 + а4 + а5 + а6 – а1 – а2 – а3 = а4 + а5 + а6.

Следователно разликата на всеки два елемента на масива с ще дава сума на някои елементи на масива а.

Ако подредим елементите на масива с, разликите между два съседни елементи ще са най-малки, напр. между с1 и с2 разликата е най-малка сравнена с разликите на с1 и с3, с1 и с4, с1 и с5 и т.н. Аналогично разликата на с2 и с3 е най-малка, сравнена с разликата между елементите с2 и с4, с2 и с5 и т.н.

Следователно измежду тези разлики между съседни елементи търсим най-близкото до 0. В примера долу е 1.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ИНДЕКС | 0 | | 1 | | 2 | | 3 | | 4 | | 5 | |
| а |  | | 2 | | 1 | | -8 | | -7 | | 6 | |
| с | 0 | | 2 | | 3 | | -5 | | -12 | | -6 | |
| с - сортиран | -12 | | -6 | | -5 | | 0 | | 2 | | 3 | |
| ралики |  | 6 | | 1 | | 5 | | 2 | | 1 | |  |

#include <iostream>

#include <algorithm>

using namespace std;

int main()

{

int a[1000], c[1000];

int n, i; cin>>n;

c[0][0]=0

for (i=1; i<=n; i++)

{cin >> a[i];c[i]=c[i-1]+a[i];}

sort (c, c+n+1);

int Sum = c[1]-c[0];

for (int i = 2; i<= n; i++)

if (c[i]-c[i-1]<Sum ) Sum = c[i]-c[i-1];

cout << Sum << endl;

return 0;

}

При тест с 5 елемента: 2, 4, -8, -7, 6 би трябвало да изведе -1, но отново извежда 1. Проблемът е, че при сортирането на масива с винаги от по-голям елемент вадим по-малък и получаваме положително число.

**Зад. 4. Максимална сума в правоъгълна област**

Дадена е матрица NxM с елементи цели числа. Да се намери максималната сума в правоъгълна подобласт на матрицата.

#include <iostream>

#include <algorithm>

using namespace std;

int a[1000][1000], b[1000][1000];

int main()

{

int n, m, i, j, k, TempSum, MaxSum;

cin >> n >> m;

for (j = 1; j <= m; j++) b[0][j] = 0;

for (i = 1; i <= n; i++)

for (j = 1; j <= m; j++)

{

cin >> a[i][j];

b[i][j] = b[i-1][j] + a[i][j];

}

MaxSum = b[1][1];

for (i = 1; i <= n; i++)

for(k = 1; k <= i; k++)

{

TempSum = 0;

for( j = 1; j <= m; j++)

{

TempSum = max (TempSum + b[i][j] - b[i-k][j], b[i][j] - b[i-k][j]);

MaxSum = max(MaxSum, TempSum);

}

}

cout << MaxSum << endl;

return 0;

}