**实验内容：**

比较求解最大子段和的三个算法。方法分别为遍历所有起止位置及长度、遍历所有起止位置、贪心单次遍历，时间复杂度分别为O(n3)、O(n2)、O(n)。

**测试结果：**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 算法 |  |  |  |  |  |  |
| 1 | N | 10 | 100 | 500 | 1000 | 3000 |
| Time(ms) | ≈0 | 1 | 46 | 362 | 9785 |
| 2 | N | 10 | 100 | 500 | 1000 | 3000 |
| Time(ms) | ≈0 | ≈0 | ≈0 | 1 | 10 |
| N | 5000 | 10000 | 30000 | 50000 |  |
| Time(ms) | 28 | 104 | 937 | 2588 |  |
| 4 | N | 10 | 100 | 500 | 1000 | 3000 |
| Time(ms) | ≈0 | ≈0 | ≈0 | ≈0 | ≈0 |
| N | 5000 | 10000 | 30000 | 50000 | 100000 |
| Time(ms) | ≈0 | ≈0 | ≈0 | ≈0 | 1 |
| N | 200000 | 300000 | 500000 | 800000 | 1000000 |
| Time(ms) | 1 | 1 | 2 | 2 | 3 |
| N | 3000000 | 5000000 | 8000000 | 10000000 |  |
| Time(ms) | 9 | 18 | 26 | 35 |  |

**源代码：**

#include <iostream>

#include <ctime>

#include <cstdlib>

using namespace std;

int algo1(const int A[], int N)

{

int ThisSum, MaxSum, i, j, k;

/\* 1\*/ MaxSum = 0; /\* initialize the maximum sum \*/

/\* 2\*/ for (i = 0; i < N; i++) /\* start from A[ i ] \*/

/\* 3\*/ for (j = i; j < N; j++) { /\* end at A[ j ] \*/

/\* 4\*/ ThisSum = 0;

/\* 5\*/ for (k = i; k <= j; k++)

/\* 6\*/ ThisSum += A[k]; /\* sum from A[ i ] to A[ j ] \*/

/\* 7\*/ if (ThisSum > MaxSum)

/\* 8\*/ MaxSum = ThisSum; /\* update max sum \*/

} /\* end for-j and for-i \*/

/\* 9\*/ return MaxSum;

}

int algo2(const int A[], int N)

{

int ThisSum, MaxSum, i, j;

/\* 1\*/ MaxSum = 0; /\* initialize the maximum sum \*/

/\* 2\*/ for (i = 0; i < N; i++) { /\* start from A[ i ] \*/

/\* 3\*/ ThisSum = 0;

/\* 4\*/ for (j = i; j < N; j++) { /\* end at A[ j ] \*/

/\* 5\*/ ThisSum += A[j]; /\* sum from A[ i ] to A[ j ] \*/

/\* 6\*/ if (ThisSum > MaxSum)

/\* 7\*/ MaxSum = ThisSum; /\* update max sum \*/

} /\* end for-j \*/

} /\* end for-i \*/

/\* 8\*/ return MaxSum;

}

int algo4(const int A[], int N)

{

int ThisSum, MaxSum, j;

/\* 1\*/ ThisSum = MaxSum = 0;

/\* 2\*/ for (j = 0; j < N; j++) {

/\* 3\*/ ThisSum += A[j];

/\* 4\*/ if (ThisSum > MaxSum)

/\* 5\*/ MaxSum = ThisSum;

/\* 6\*/ else if (ThisSum < 0)

/\* 7\*/ ThisSum = 0;

} /\* end for-j \*/

/\* 8\*/ return MaxSum;

}

void genRandom(int\* num, int n)

{

srand(time(nullptr));

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

{

num[i] = rand() % 10;

}

}

void test(int\* num, int n)

{

clock\_t pt, nt;

cout << "n = " << n << ":" << endl;

if (n <= 3000)

{

pt = clock();

algo1(num, n);

nt = clock();

cout << "ALGORITHM 1: " << nt - pt << " CLOCKS(" << (nt - pt) / CLOCKS\_PER\_SEC << " sec)" << endl;

}

if (n <= 50000)

{

pt = clock();

algo2(num, n);

nt = clock();;

cout << "ALGORITHM 2: " << nt - pt << " CLOCKS(" << (nt - pt) / CLOCKS\_PER\_SEC << " sec)" << endl;

}

pt = clock();

algo4(num, n);

nt = clock();

cout << "ALGORITHM 4: " << nt - pt << " CLOCKS(" << (nt - pt) / CLOCKS\_PER\_SEC << " sec)" << endl;

cout << endl;

}

int main()

{

ios::sync\_with\_stdio(false);

int\* num = new int[10000000];

genRandom(num, 10000000);

test(num, 10);

test(num, 100);

test(num, 500);

test(num, 1000);

test(num, 3000);

test(num, 5000);

test(num, 10000);

test(num, 30000);

test(num, 50000);

test(num, 100000);

test(num, 200000);

test(num, 300000);

test(num, 500000);

test(num, 800000);

test(num, 1000000);

test(num, 3000000);

test(num, 5000000);

test(num, 8000000);

test(num, 10000000);

delete[] num;

}

**结果分析：**

结果符合预期。不同时间复杂度的算法耗时在规模逐步增大时差距明显。

这让我深刻体会到面对同一问题时，使用一个好的算法会对解决问题产生的巨大的时效优势。