数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 大二 | 姓名 | 丁彦君 | 学号 | 2329403126 |
| 实验布置日期 | | 2024.9.10 | | 提交  日期 | 9.16 | | 成绩 |  |

课程实践实验1-2（选做）

## 一、问题描述及要求

1.

问题：

s是序列 A0， A1，... An-1 （可能有负数），求 A0 ~ An-1 的一个子序列 Ai ~ Aj，使得 Ai 到 Aj 的和最大。

要求：

（1）编写程序实现算法是否正确

（2）用事前分析法分析这3个算法的时间复杂度

（3）为s.n设定一个适当的规模假设算法开始时间为t1，结束时间是t2，通过t2-t1可以衡量算法实际需要的运行时间。记录并对比这3个算法的实际运行时间的差异，看看它们与（2）中估量的时间数量级是否一致。

(4) 最后给出对这3个算法的分析结论。

2.

问题：

按顺序从小到大依次输入从1到n的所有数字。

要求：

（1）编写程序实现算法是否正确

（2）测试当n达到多少时，算法1会崩溃，并对比此时算法2是否能运行成功。

（3）递归程序的时间复杂度的确定有时比较难以分析，有一种方法是将递归转换成非递归后来分析其的时间复杂度，当然两者是不完全等价的，本题两个算法时间复杂度都为O（n），

当n取一个适当值时，请计算程序实际运行时间，看看这个结论是否正确。

## 二、概要设计

（1）**对实验内容的理解：**

第一个问题：找出连续的最大子序列和

第二个问题：顺序输出1-n并分析递归算法的时间复杂度。

（2）**程序结构设计：**

第一个问题：

算法1：暴力法（三重循环遍历整个序列）（贪心）。

算法2：对比算法1略有改进（二重循环遍历序列）。

算法3：卡丹算法的变种（类似于动态规划），利用局部最优解来逐步构建全局最优解。

第二个问题：

算法1：递归

算法2：一次for循环

## 三、详细设计

**\*第一个问题：**

算法1：设置双指针i和j，分别从数组的最左边和i的右边开始遍历，再引入一个k指针负责在[I,j]的区间遍历求出此区间的连续和。最后找出最大的那个输出。

k



算法2：还是双指针，但是省去了算法1的k指针，直接让j指针从i开始遍历到数组末尾。

j



算法3：类似于动态规划，利用局部最优解来构建全局最优解，从头到尾仅需要遍历一次。

**重点分析：**设定一个ans保存当前最大的值，一个res保存当前已遍历的连续序列的和，当下一步操作后此时序列和小于0时，更改指针i的起始位置为当前位置，并重置res为0，,接着继续向后遍历。



res

ans

**第二个问题：**

算法1：递归算法

算法2：顺序结构的基本实现（for循环直接输出）

## 四、实验结果

第一个问题：

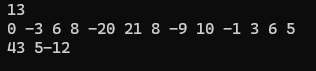
算法1：

测试输入：第一行：13 第二行：0 -3 6 8 -20 21 8 -9 10 -1 3 6 5

测试目的：验证该算法是否正确可行。

正确输出：43 5-12

实际输出：



错误原因：无

测试结论：通过。

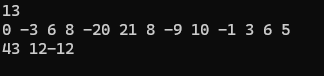
算法2：

测试输入：第一行：13 第二行：0 -3 6 8 -20 21 8 -9 10 -1 3 6 5

测试目的：验证该算法是否正确可行。

正确输出：43 5-12

实际输出：



错误原因：在循环内部多行的情况下没有用大括号，错误使用python的代码方式，导致最后记录指针位置的l和r值一直在改变直至循环结束。

测试结论：未通过。

改正方式：规范代码的编写，在循环外侧加上大括号。

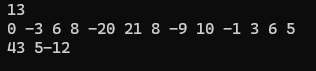
算法3：

测试输入：第一行：13 第二行：0 -3 6 8 -20 21 8 -9 10 -1 3 6 5

测试目的：验证该算法是否正确可行。

正确输出：43 5-12

实际输出：



错误原因：无

测试结论：通过。

第二个问题

算法1,2：（均通过）



## 五、实验分析与探讨

测试结果的分析：

1. 总结算法的时空复杂度：

第一个问题：

算法1：时间复杂度：O(n^3) 空间复杂度：O(n)

算法2：时间复杂度：O(n^2) 空间复杂度：O(n)

算法3：时间复杂度：O(n) 空间复杂度：O(n)

第二个问题：

算法1：时间复杂度：O(n) 空间复杂度：O(n)

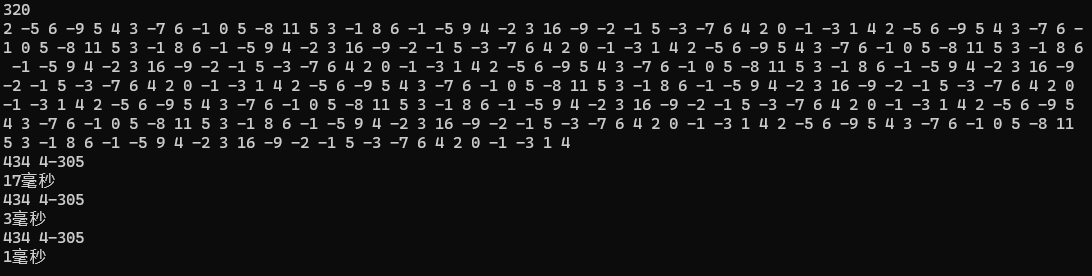
算法2：时间复杂度：O(n) 空间复杂度：O(1)

1. 探讨更多解决问题的途径：

第一个问题：除了使用双指针法暴力求解，还可以使用分治法进行求解，也可以运用动态规划进行求解（并可以进一步优化空间至0(1)），动态规划法和算法3类似，都是利用局部最优解来构建全局最优解。

1. 比较真实时间情况：

第一个问题：

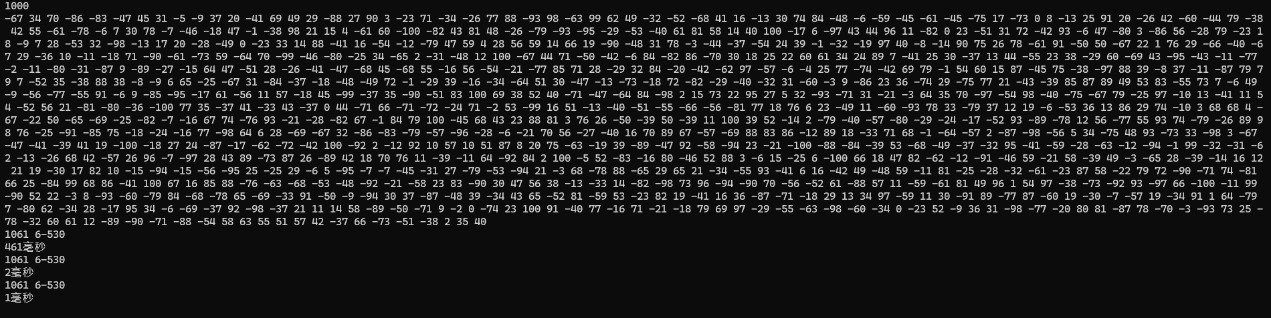


算法1:17ms

算法2:3ms

算法3:1ms

结果：符合



算法1:461ms

算法2:2ms

算法3:1ms

结果：稍微符合，不算完全符合

第二个问题：试过多次，用学校电脑时第一个递归算法不崩溃n最多到60569，60570开始崩溃，而第二个算法到100000都没崩溃。用自己电脑也是这样，第一个递归最多到25000左右，但是第二个能到50000以上。

当n = 10000时，算法1时间为6.26s，算法2时间为2.37s，两者差距很大。

说明在这里时间复杂度不准确，虽然都是O(n),但是具体运行时间差异明显。

问题探讨：

Q1：两个问题都很容易，主要在于代码的规范性不强，所以容易出错，比如在循环中没有使用{}，导致只循环后面第一行，进而导致结果出问题。

解决方法：规范代码格式，找出低级错误并进行改正。

## 六、小结

这两题的解决方法都比较基础，算法也很亲民。但在实际应用中，每种解决方法在处理大型数据集时，其耗时差异可能非常显著。这一现象深刻地揭示了一个重要的道理：在运用代码解决问题时，我们不仅需要关注如何有效地找到问题的解决方案，还必须重视解决方案的时间效率。在解决问题的过程中，我们应该努力优化代码和算法，以减少时间开支和资源消耗。算法的优化不仅可以显著提升程序的运行速度，还能有效降低计算资源的需求，从而在面对大规模数据时显得游刃有余。因此，关注并实施算法优化，是确保在处理实际问题时能够以最低的时间成本获得最佳解决方案的关键。这一过程不仅有助于提高系统的整体性能，还能提升处理大数据任务的能力，从而使得最终的解决方案更加高效和实用。在今后的学习过程中，我也应该注意到这一点，把注意力不仅要放在问题的解决上，还要放在其可行性和效率上从而实现算法效益最大化。

## 附录：源代码

1、注意说明实验环境：Dev.cpp

2、

（1）1-1.cpp

代码

#include <iostream>

using namespace std;

void func1(int n,int\* arr)

{

int sum = 0;

int l = 0,r = 0;

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

{

for(int j = n - 1;j >= i;j--)

{

int res = 0;

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

res += arr[k];

if(res > sum)

{

sum = res;

l = i;

r = j;

}

}

}

cout << sum << " " << l << "-" << r << endl;

}

int main(){

int n;

cin >> n;

int\* arr = new int[n];

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

cin >> arr[i];

func1(n,arr);

delete[] arr;

return 0;

}

（2）1-2.cpp

代码

#include <iostream>

using namespace std;

void func2(int n,int\* arr)

{

int sum = 0;

int l = 0,r = 0;

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

{

int res = 0;

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

{

res += arr[j];

if(res > sum)

{

sum = res;

l = i;

r = j;

}

}

}

cout << sum << " " << l << "-" << r << endl;

}

int main(){

int n;

cin >> n;

int\* arr = new int[n];

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

cin >> arr[i];

func2(n,arr);

delete[] arr;

return 0;

}

（3）1-3.cpp

代码

#include <iostream>

using namespace std;

void func3(int n,int \*arr)

{

int sum = 0;

int res = 0;

int l = 0,r = 0;

int t = 0;

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

{

res += arr[i];

if(res < 0)

{

res = 0;

t = i + 1;

}

if(res >= sum)

{

sum = res;

l = t;

r = i;

}

}

cout << sum << " " << l << "-" << r << endl;

}

int main(){

int n;

cin >> n;

int\* arr = new int[n];

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

cin >> arr[i];

func3(n,arr);

delete[] arr;

return 0;

}

（4）2-1.cpp

代码

#include <iostream>

using namespace std;

void RecursivePrint(int n){

if(n > 0)

{

RecursivePrint(n - 1);

cout << n << " ";

}

}

int main()

{

int n;

cin >> n;

RecursivePrint(n);

}

（5）2-2.cpp

代码

#include <iostream>

using namespace std;

int main()

{

int n;

cin >> n;

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

cout << i << " ";

}

（6）test1.cpp

代码

#include <iostream>

#include <time.h>

using namespace std;

void func1(int n,int\* arr)

{

int sum = 0;

int l = 0,r = 0;

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

{

for(int j = n - 1;j >= i;j--)

{

int res = 0;

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

res += arr[k];

if(res > sum)

{

sum = res;

l = i;

r = j;

}

}

}

cout << sum << " " << l << "-" << r << endl;

}

void func2(int n,int\* arr)

{

int sum = 0;

int l = 0,r = 0;

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

{

int res = 0;

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

{

res += arr[j];

if(res > sum)

{

sum = res;

l = i;

r = j;

}

}

}

cout << sum << " " << l << "-" << r << endl;

}

void func3(int n,int \*arr)

{

int sum = 0;

int res = 0;

int l = 0,r = 0;

int t = 0;

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

{

res += arr[i];

if(res < 0)

{

res = 0;

t = i + 1;

}

if(res >= sum)

{

sum = res;

l = t;

r = i;

}

}

cout << sum << " " << l << "-" << r << endl;

}

int main() {

int n;

cin >> n;

int\* arr = new int[n];

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

cin >> arr[i];

clock\_t start\_time,end\_time;

start\_time = clock();

func1(n, arr);

end\_time = clock();

double duration1 = static\_cast<double>(end\_time - start\_time) \* 1000.0 / CLOCKS\_PER\_SEC;

cout << duration1 << "毫秒" << endl;

start\_time = clock();

func2(n, arr);

end\_time = clock();

double duration2 = static\_cast<double>(end\_time - start\_time) \* 1000.0 / CLOCKS\_PER\_SEC;

cout << duration2 << "毫秒" << endl;

start\_time = clock();

func3(n, arr);

end\_time = clock();

double duration3 = static\_cast<double>(end\_time - start\_time) \* 1000.0 / CLOCKS\_PER\_SEC;

cout << duration3 << "毫秒" << endl;

delete[] arr;

return 0;

}