
算法分析与设计

实验 02 李飞飞 软工 2206 202105710309

实验目的

- 掌握分治和递归的概念。
- 掌握分治法的基本思想及算法复杂性分析。
- 掌握二分搜索技术和正整数划分法。
- 掌握大数乘法和 Strassen 矩阵乘法。
- 掌握合并排序，快速排序和线性时间选择的原理和程序执行过程。

实训内容

1、试论述合并排序和快速排序的各自优缺点，并举例说明。

归并排序和快速排序都是常用的排序算法，它们各有优缺点。下面我将对它们进行论述，并举例说明。

归并排序：

1. 优点：

- 稳定性：是稳定的排序算法,不改变相对位置
- 适用性广泛：归并排序适用于各种数据结构，包括数组、链表等，因为其操作依赖于元素之间的比较和合并，而不需要随机访问。
- 时间复杂度稳定：归并排序的时间复杂度为 $O(n \log n)$ ，无论是最好情况、最坏情况还是平均情况，都能保持这一复杂度。

2. 缺点：

- 空间复杂度高：需要临时数组，其空间复杂度为 $O(n)$ 。
- 不适合小规模数据：归并排序在实践中对于小规模数据的排序性能略逊于其他排序算法，因为其分治的递归过程对于小规模数据而言较为昂贵。

3. 例子：

- 考虑一个数组 [3, 1, 4, 1, 5, 9, 2, 6, 5]，通过归并排序对其进行排序：
- 将数组分成单个元素的子数组：[3], [1], [4], [1], [5], [9], [2], [6], [5]
- 逐步合并相邻的子数组并排序：

[1, 3], [1, 4], [1, 4, 5], [2, 9], [5, 6], [1, 3, 4, 5, 5, 6, 9]

快速排序:

1. 优点:

- 原地排序: 空间复杂度 $O(1)$
- 平均情况下高效: 在平均情况下, 快速排序的时间复杂度为 $O(n \log n)$, 并且常数因子较小, 因此它是一种高效的排序算法。
- 适用于大规模数据: 快速排序在处理大规模数据时具有良好的性能, 因为其分而治之的策略能够有效地利用现代计算机的缓存机制。

2. 缺点:

- 不稳定性: 一种不稳定的排序算法, 改变相对位置
- 最坏情况下效率低: 在最坏情况下, 即待排序数组已经有序或者逆序排列时, 快速排序的时间复杂度会退化到 $O(n^2)$, 因此其性能不稳定。

3. 例子:

- $[3, 1, 4, 1, 5, 9, 2, 6, 5]$
- 选择一个基准值 (例如选择第一个元素 3)。
- 将数组分成两部分, 小于基准放左, 大于放右边: $[1, 1, 2], [3], [4, 5, 9, 6, 5]$
- 递归地对左右两部分进行快速排序。
- 最终合并得到排序后的数组: $[1, 1, 2, 3, 4, 5, 5, 6, 9]$

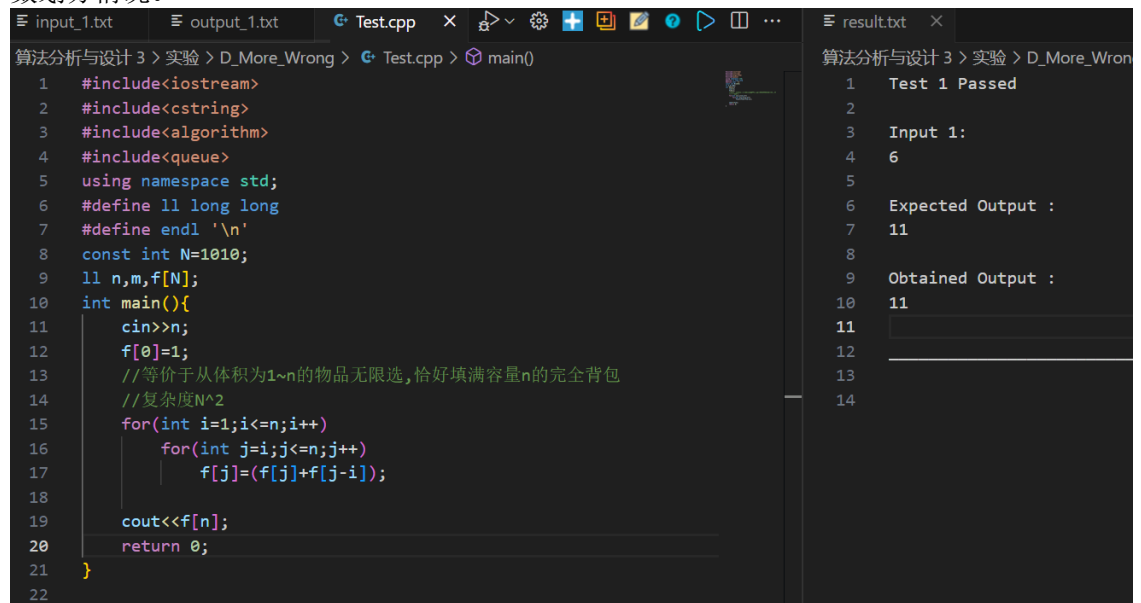
所有代码放在附件, 这里展示一个样例或对题目做回答

2、二分搜索技术的 7 种算法分析题: 2-2 (P36)。

都是 left 和 right 下标不对引起, 后面不单独说了

- 1) 错误, left 和 right 调整不对, $x=a[n-1]$ 死循环
- 2) 错误, $x=a[n-1]$ 返回错误
- 3) 错误, $x=a[n-1]$ 返回错误
- 4) 错误, 死循环
- 5) 正确, 返回满足的最右边的元素
- 6) 错误, $x=a[n-1]$ 死循环
- 7) 错误, $x=a[0]$ 死循环

3、算法设计题: 完善课本大整数划分程序 (例 2-5, P13), 输出一个正整数所有的整数划分情况。



```
1 #include<iostream>
2 #include<cstring>
3 #include<algorithm>
4 #include<queue>
5 using namespace std;
6 #define ll long long
7 #define endl '\n'
8 const int N=1010;
9 ll n,m,f[N];
10 int main(){
11     cin>>n;
12     f[0]=1;
13     //等价于从体积为1~n的物品无限选,恰好填满容量n的完全背包
14     //复杂度N^2
15     for(int i=1;i<=n;i++){
16         for(int j=i;j<=n;j++){
17             f[j]=(f[j]+f[j-i]);
18         }
19     }
20     cout<<f[n];
21     return 0;
22 }
```

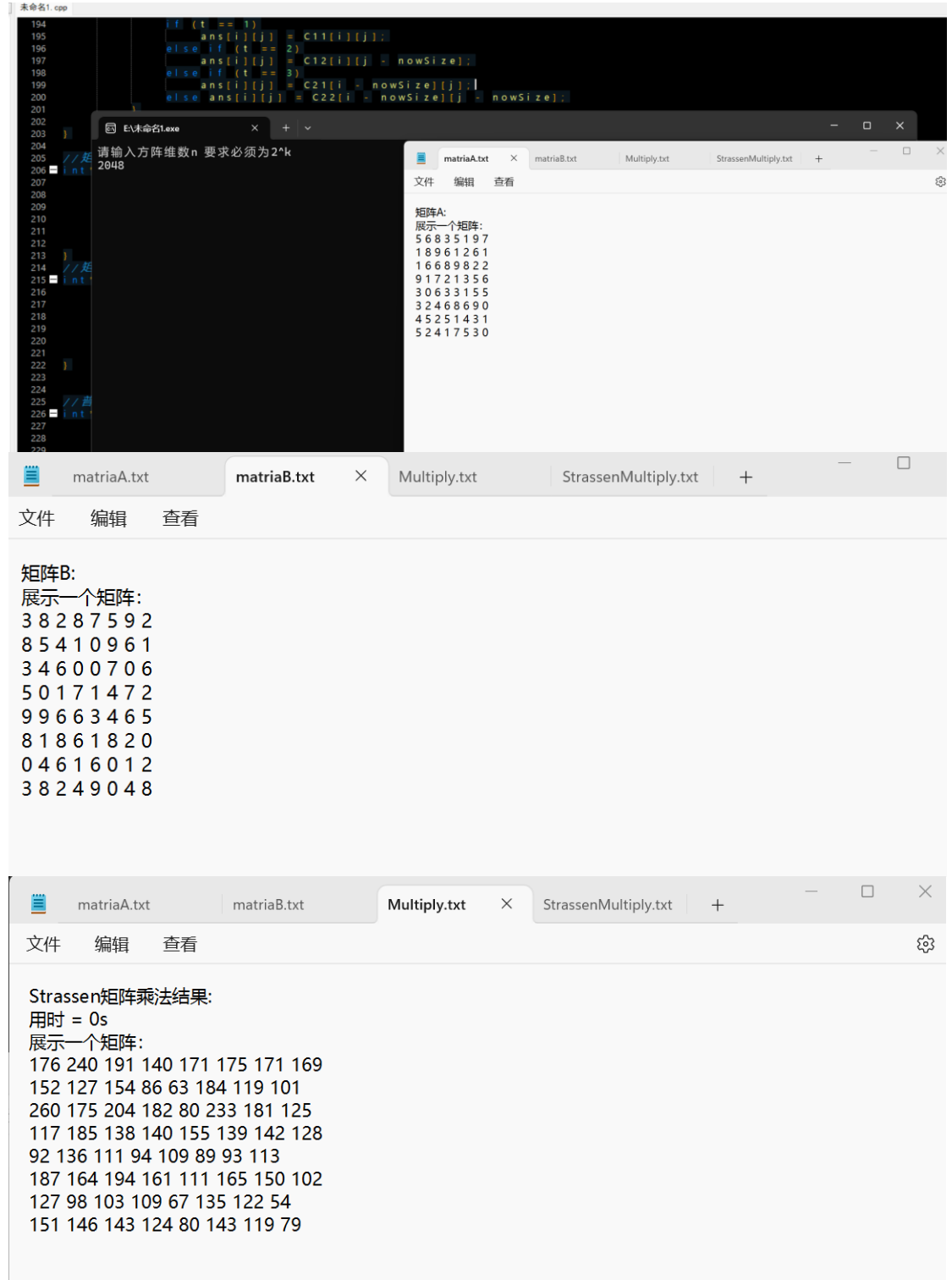
result.txt

```
1 Test 1 Passed
2
3 Input 1:
4 6
5
6 Expected Output :
7 11
8
9 Obtained Output :
10 11
11
12
13
14
```

4、算法设计题：根据课本第 2.5 节 Strassen 矩阵乘法的分治思想(P19)，编程实现两个矩阵的乘法，要求如下：

- 1) 输入矩阵维数 m 和 n ；
- 2) 自动生成两个矩阵的元素（为正整数），输出保存到文件 `matriaA.txt` 和 `matriaB.txt`。
- 3) 修改 m 和 n ，对比原始乘法和 Strassen 乘法的运算时间。

小规模正确性检验：



```
194 if (t == 1)
195     ans[i][j] = C11[i][j];
196 else if (t == 2)
197     ans[i][j] = C12[i][j] + nowSize;
198 else if (t == 3)
199     ans[i][j] = C21[i][j] + nowSize;
200 else ans[i][j] = C22[i][j] + nowSize;
201
202
203
204
205 // 输入方阵维数n 要求必须为2^k
206 int n;
207
208
209
210
211
212
213 // 生成矩阵A
214 int A[n][n];
215 int B[n][n];
216
217
218
219
220
221
222
223
224 // 生成矩阵B
225 int B[n][n];
226
227
228
229
```

请输入方阵维数n 要求必须为2^k
2048

矩阵A:
展示一个矩阵:
5 6 8 3 5 1 9 7
1 8 9 6 1 2 6 1
1 6 6 8 9 8 2 2
9 1 7 2 1 3 5 6
3 0 6 3 3 1 5 5
3 2 4 6 8 6 9 0
4 5 2 5 1 4 3 1
5 2 4 1 7 5 3 0

矩阵B:
展示一个矩阵:
3 8 2 8 7 5 9 2
8 5 4 1 0 9 6 1
3 4 6 0 0 7 0 6
5 0 1 7 1 4 7 2
9 9 6 6 3 4 6 5
8 1 8 6 1 8 2 0
0 4 6 1 6 0 1 2
3 8 2 4 9 0 4 8

Strassen矩阵乘法结果:
用时 = 0s
展示一个矩阵:
176 240 191 140 171 175 171 169
152 127 154 86 63 184 119 101
260 175 204 182 80 233 181 125
117 185 138 140 155 139 142 128
92 136 111 94 109 89 93 113
187 164 194 161 111 165 150 102
127 98 103 109 67 135 122 54
151 146 143 124 80 143 119 79

```
matriaA.txt  matriaB.txt  Multiply.txt  StrassenMultij x  +  -  □  ×

文件  编辑  查看

Strassen矩阵乘法结果:
用时 = 0s
展示一个矩阵:
176 240 191 140 171 175 171 169
152 127 154 86 63 184 119 101
260 175 204 182 80 233 181 125
117 185 138 140 155 139 142 128
92 136 111 94 109 89 93 113
187 164 194 161 111 165 150 102
127 98 103 109 67 135 122 54
151 146 143 124 80 143 119 79
```

大数据(256 维):

本来写的 2048 的结果空间占太多直接把我电脑卡崩了

但是数据不够大, 算法常数太大直接跑不过暴力了

```
"w", stdout); // 打开写入

n);

去结果:\n";
ble(end - st

y.txt", "w",

y(A, B, n);

去结果:\n";
ble(end - st

矩阵乘法结果:
用时 = 0.044s
展示一个矩阵:
648167 631145 680600 629586 645095 621495 582838 629143 617329 593801 603271 583946 598015
610684 643400 605878 617268 657954 625953 597210 639277 660844 613504 640592 601090 630672
656703 607187 626224 625519 655652 615648 637756 599269 628650 653639 638216 644302 611294
593123 578228 592393 591249 611257 600452 659092 626323 645949 636142 643319 603158 591381
528955 662347 594028 648902 632631 617481 608540 642177 640655 623776 654833 676625 661265
682581 585117 584219 692939 627635 623046 589405 690418 650085 673708 662889 628642 629521
```

```
rt, end;
矩阵乘法
multiply.txt", "w", stdout); // 打开写入
ock();
Multiply(A, B, n);
:());
rassen矩阵乘法结果:\n";
时 = " << double(end - st
n);
out);

sen
rassenMultiply.txt", "w",
ock();
strassenMultiply(A, B, n);
:());
rassen矩阵乘法结果:\n";
时 = " << double(end - st
n);
out);

因为我太懒, 所以放一起回
: st)
Array(t.first, t.second);
```

5、算法设计题 2-5(P39): 根据课本第 2.4 节大数乘的分治思想(P19), 按要求编程实现两个位大整数的乘法。

这个代码一直写不出来我就光证了一下, 没写代码, 写了矩阵乘法

地计算如下:

会遇到。它定义为: $a(n) = \min\{k | a(k) \geq n\}$, 即 $a(n)$ 是使 $n \leq a(k)$ 成立的最小的 k 值。
如: $a(1)=1, a(2)=1, a(3)=1, a(4)=2, a(5)=2, \dots$

设两个 n 进制大整数分别为 x, y ; $t = 10^{\frac{n}{3}}$

$x = \begin{matrix} A & B & C \\ t & t & t \end{matrix}$ 以下表示时 t 为将乘法以
等值字典序子形式表示
如: $BA = AB$, 记为 AB

$y = \begin{matrix} D & E & F \\ t & t & t \end{matrix}$

$xy = (At^2 + Bt + C) \times (Dt^2 + Et + F)$

展开: $= ADt^4 + AEt^3 + Aft^2 + BDt^3 + BEt^2 + Bft + CDt^2 + CEt + CF$

整理: $= A(Dt^4 + (AE + BD)t^3 + (AF + BE + CD)t^2 + (CE + BF)t + CF)$

可得: $\begin{cases} O(1) & n=1,2 \\ 4T(\frac{n}{3}) + O(n) & n \geq 3 \end{cases}$

根据 Master 定理 $T(n) = O(n^{\log_3 4}) = O(n^2)$

考虑简化:

$AE + BD = (A-B)(E-D) + AD + BE$

$CE + BF = (C-B)(E-F) + CF + BE$

经过计算发现 t^2 系数无法简化

\therefore 简化后需要 5 次乘法, 若干次加法 (写坏层数)

设位置 3,

$T(n) = \begin{cases} O(1) & n=1,2 \\ 5T(\frac{n}{3}) + O(n) & n \geq 3 \end{cases} \Rightarrow T(n) = O(n^{\log_3 5}) = O(n^{1.46})$

后经查阅资料发现可以扩展到一般式, 不过这里就不写了

证：前面证明有误

$$x = u_0 + u_1 t + u_2 t^2$$

$$y = v_0 + v_1 t + v_2 t^2$$

$$w = x \cdot y = w_0 + w_1 t + w_2 t^2 + w_3 t^3 + w_4 t^4$$

取 $t_1 = 0 \quad t_2 = -2 \dots -2 \quad -1 \quad 1$ 得五个方程组：

可证
至多
5次
求法

$$\begin{cases} a = w(t_1) = v_0 v_0 \\ b = w(t_2) = (u_0 + 2v_1 + 4u_2)(v_0 + 2v_1 + 4v_2) \\ c = w(t_3) = (u_0 + 2u_1 + 4u_2)(v_0 + 2v_1 + 4v_2) \\ d = w(t_4) = (v_0 + u_1 + u_2)(v_0 + v_1 + v_2) \\ e = w(t_5) = (u_0 + u_1 + u_2)(v_0 + v_1 + v_2) \end{cases}$$

联解得： $w_0 = a$

$$w_1 = \frac{b - c - 8(a - c)}{12}$$

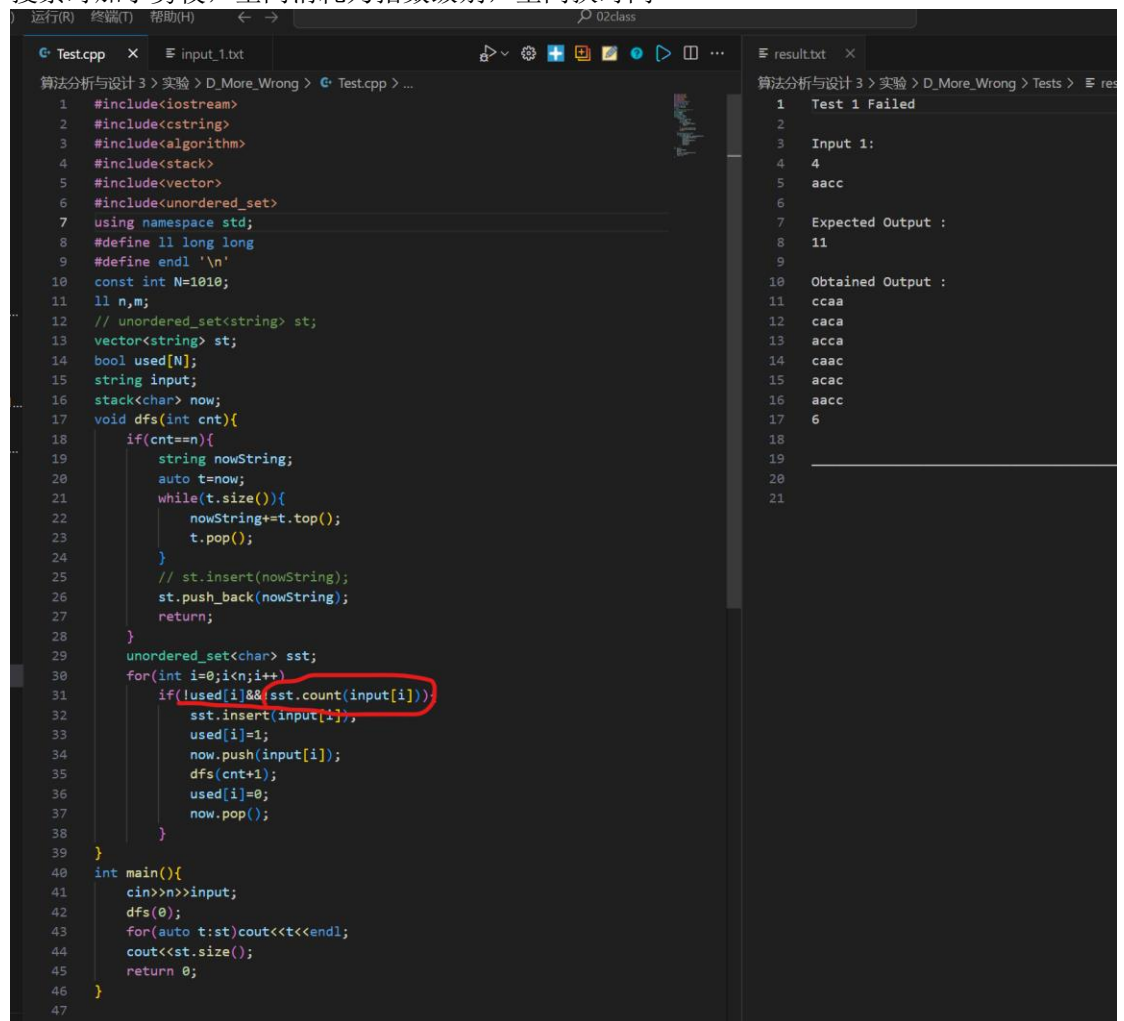
$$w_2 = \frac{-b - c + 16(a + c) - 30a}{24}$$

$$w_3 = \frac{-b + c + 2(a - c)}{12}$$

$$w_4 = \frac{b + c - 4(a + c) + 6a}{24}$$

6、算法设计题 2-5(P42): 编程实现有重复元素的排列问题。

搜索时加了剪枝, 空间消耗为指数级别, 空间换时间



```
1 #include<iostream>
2 #include<cstring>
3 #include<algorithm>
4 #include<stack>
5 #include<vector>
6 #include<unordered_set>
7 using namespace std;
8 #define ll long long
9 #define endl '\n'
10 const int N=1010;
11 ll n,m;
12 // unordered_set<string> st;
13 vector<string> st;
14 bool used[N];
15 string input;
16 stack<char> now;
17 void dfs(int cnt){
18     if(cnt==n){
19         string nowString;
20         auto t=now;
21         while(t.size()){
22             nowString+=t.top();
23             t.pop();
24         }
25         // st.insert(nowString);
26         st.push_back(nowString);
27         return;
28     }
29     unordered_set<char> sst;
30     for(int i=0;i<n;i++){
31         if(!used[i]&& sst.count(input[i])){
32             sst.insert(input[i]);
33             used[i]=1;
34             now.push(input[i]);
35             dfs(cnt+1);
36             used[i]=0;
37             now.pop();
38         }
39     }
40 }
41 int main(){
42     cin>>n>>input;
43     dfs(0);
44     for(auto t:st)cout<<t<<endl;
45     cout<<st.size();
46     return 0;
47 }
```

```
1 Test 1 Failed
2
3 Input 1:
4 4
5 aacc
6
7 Expected Output :
8 11
9
10 Obtained Output :
11 ccaa
12 caca
13 acca
14 caac
15 acac
16 aacc
17 6
18
19
20
21
```