***2022***



**算法设计与分析实践报告**



|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS2009 |
| 学 号： | U202015562 |
| 姓 名： | 徐雨梦 |
| 完成日期： | 2022.5.18 |

目 录

[1. 完成情况 1](#_Toc5980)

[2. POJ2506解题报告 1](#_Toc24002)

[2.1题目分析 1](#_Toc32495)

[2.2 算法设计 2](#_Toc21946)

[2.3性能分析 3](#_Toc7185)

[2.4运行测试 3](#_Toc15316)

[3.POJ2228解题报告 4](#_Toc19300)

[3.1题目分析 4](#_Toc15005)

[3.2 算法设计 4](#_Toc5095)

[3.3性能分析 7](#_Toc20756)

[3.4运行测试 7](#_Toc17563)

[4 POJ1700解题报告 8](#_Toc5714)

[4.1题目分析 8](#_Toc21502)

[4.2 算法设计 8](#_Toc7029)

[4.3性能分析 10](#_Toc14540)

[4.4运行测试 11](#_Toc26113)

[5 POJ3660解题报告 11](#_Toc28009)

[5.1题目分析 11](#_Toc3592)

[5.2 算法设计 11](#_Toc7127)

[5.3性能分析 13](#_Toc9947)

[5.4运行测试 13](#_Toc7722)

[6. 总结 14](#_Toc3612)

[6.1 实验总结，可以写几点做题中学到程序技术和技巧 14](#_Toc8152)

[6.2 心得体会和建议 15](#_Toc23643)

# **1. 完成情况**

（1）做的题目的情况

所做题目为实验指导书中提供的题目。

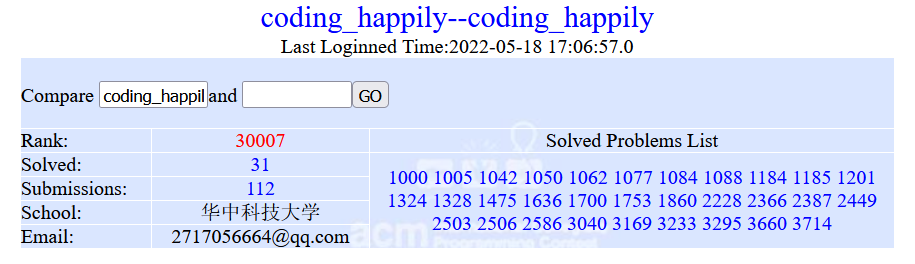


图1 做的题目

1. 最后通过的题目的情况

最终通过的题目是27道必做题+4道选做题，总共31道。

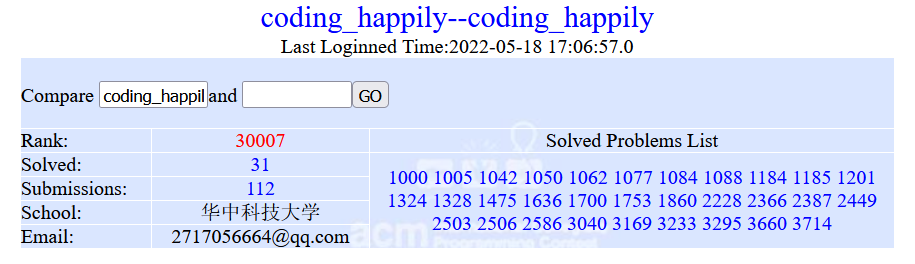


图2 完成的题目

# **2. POJ2506解题报告**

## **2.1题目分析**

POJ2506：Tiling

对于一个2 x n的纸带，可以对他分解成2x1和2x2的矩形，那么有多少种分解方案。

## **2.2 算法设计**

采用分治思想。一个n长度的纸带，可以分为两种情况，即最后一段为2x2的和最后一段为1x2的。而对于2x2的矩阵，可以为一个2x2的，也可以为两个1x2上下拼接所得的。故一个n长度的纸带的分解方案有n-1长度的纸带的分解方案加上n-2的分解方案的2倍所得。

而在本道题目中，该递归式所得到的An（n长度的纸带的分解方案数）的结果超出了计算机所能表示的一个数的范围，因此我们需要用一个变长的数组保存An的结果，按照运算规则得到结果。

代码段1 POJ2506

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

using namespace std;

int a[300][10000];

int main()

{

//memset(a, 0, sizeof(a));

int i;

a[0][0] = 1;

a[1][0]= 1;

//每一个i长度的纸都可以从i-1长度和i-2长度得到

for (i = 2; i <= 250; i++)

{

int c=0;//保存进位

for (int j = 0; j < 10000; j++)//计算每一位的值

{

a[i][j] = a[i - 1][j] + 2 \* a[i - 2][j] + c;

c = a[i][j] / 10;//更新进位

a[i][j] %= 10;//重新得到该位置的值

}

}

int n,j;

while (scanf("%d", &n) != EOF)

{

for (i = 9999; a[n][i] == 0; i--);

//for (j = 9999; a[n][j] == 0; j--);

for (; i >= 0; i--) printf("%d", a[n][i]);

printf("\n");

}

return 0;

}

## 

## **2.3性能分析**

由于计算结果与最后的结果的大小有关，记最后的结果为f，输入的数字大小为n，输入的数字个数为k，则时间复杂度为O(kn\*lg(f))。

## **2.4运行测试**

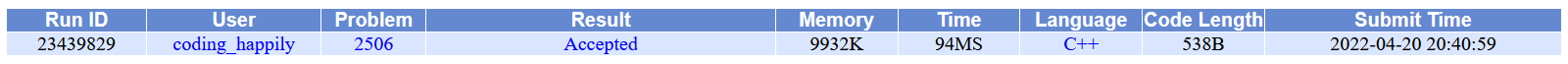


图3 POJ2506AC截图

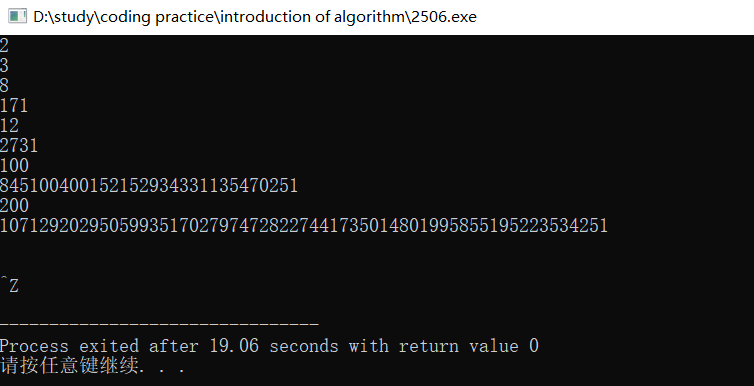


图4 POJ2506运行输出截图

# **3.POJ2228解题报告**

## **3.1题目分析**

POJ2228：Naptime

一天由 n 个小时构成，在第 i 个小时睡觉能够恢复 Ui 点体力。有一头牛要休息 b 个小时，可以不连续，但休息的第 1 个小时无法恢复体力。前一天的最后一个小时和第二天的第一个小时是连在一起的，求这头牛能恢复的体力最大值。

## 

## **3.2 算法设计**

采用动态规划的思想。母牛的休息时间可以由今天一直睡到明天，因此时间可以形成一个环，采用动态规划的方法进行求解该问题，以第一段时间为分隔，那么有两种情况，一种是休息时间全部在一天，另外一种情况是休息时间连续跨越了两天，也就是第一天的最后一段时间和第二天的第一段时间一定在休息。

我们用一个动态规划的数组保存最大休息值，dp[i][j][1]表示截止到第i个小时休息了j个小时并且第i个小时休息了的最大休息值，dp[i][j][0]表示截止到第i个小时休息了j个小时并且第i个小时没有休息的最大休息值。

那么每一次的dp值都由上一次dp的结果所求得。dp[i][j][0]由dp[i-1][j][0]和dp[i-1][j][1]之间的最大值所得，dp[i][j][1]由dp[i-1][j-1][1]+U[i]和dp[i-1][j-1][0]之间的最大值所得。

首先需要初始化第一个值，对于未跨越两天的休息时间来讲，第一个小时的休息时间的最大休息值一定为0；对于跨越了两天的休息时间来讲，第一个小时的休息时间的最大休息值为该时间对应的休息值。

在第一次dp所有的值都被赋值完毕后，从dp[n][b][1]和dp[n][b][0]之间的最大值中得到未连续休息两天的最大休息值；在第二次dp所有的值都被赋值完毕以后，只能让dp[n][b][1]作为最大休息值，因为要求第一天的最后一段时间一定要休息。取二者的最大值作为整个过程的最大休息值。

而在本道题目中，n和b的值都比较的大，因此如果直接用n和b作为动态规划数组的规模的话会非常的大，而在这个题目中，每次的dp值只与上一次的dp值有关，因此我们可以只用让n规模缩为2，利用滚动循环来得到最后的dp结果。

代码段2 POJ2228

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <iostream>

using namespace std;

const int N = 4000, M = 4000;

int n, m, f[2][M][2], w[N], ans;

int main (){

scanf ("%d%d", &n, &m);

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

scanf ("%d", &w[i]);

if (m == 0)

{ printf ("0"); exit (0); }

memset (f, 0x80, sizeof f);

f[1 & 1][0][0] = 0;

f[1 & 1][1][1] = 0;//最后一个小时和前面一个小时没有连着，初始化为0

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

for (int j = 0; j <= m; j ++)

{

f[i & 1][j][0] = max (f[(i - 1) & 1][j][0], f[(i - 1) & 1][j][1]);

if (j - 1 >= 0) f[i & 1][j][1] = max (f[(i - 1) & 1][j - 1][0], f[(i - 1) & 1][j - 1][1] + w[i]);

}

ans = max (f[n & 1][m][1], f[n & 1][m][0]);

memset (f, 0x80, sizeof f);

f[1 & 1][1][1] = w[1];//最后一个小时和第一个小时连着休息了，初始化为恢复的体力

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

for (int j = 0; j <= m; j ++)

{

f[i & 1][j][0] = max (f[(i - 1) & 1][j][0], f[(i - 1) & 1][j][1]);

if (j - 1 >= 0) f[i & 1][j][1] = max (f[(i - 1) & 1][j - 1][0], f[(i - 1) & 1][j - 1][1] + w[i]);

}

ans = max (ans, f[n & 1][m][1]);

printf ("%d", ans);

return 0;

}

## **3.3性能分析**

由于需要对dp数组进行遍历确定每一个的值，而确定每一个值只需要常数时间，因为只需要比较上一次dp的两个位置的值即可，需要进行两次动态规划，因此如果有n段时间和b段必须休息的时间，那么时间复杂度为O(nb）。

## **3.4运行测试**

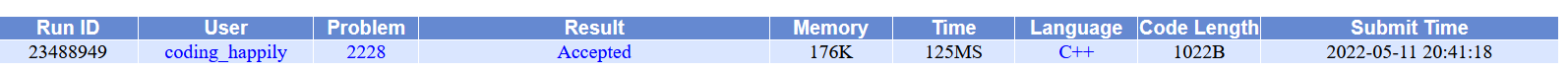


图3 POJ2228AC截图

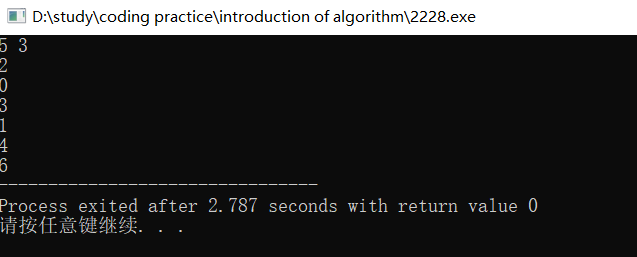


图4 POJ2228运行输出截图

# **4 POJ1700解题报告**

## **4.1题目分析**

POJ1700：Crossing River

每个人过河都有自己的过河时间。有 n 个人想过河，但只有一只小船，最多只能装 2 个人。每一次过河，过河时间为用时最多的那人过河时间，如果还有人没有过河，那么过去一个用时最少的送回船。问 n 人过河最少要多少时间。

## **4.2 算法设计**

采用贪心思想。由于每次过河起决定作用的都是用时更多的人，那么我们每次都考虑把过河需要时间最长的人渡过河去的方式。而每次两个人过河一定有一个人回来送船，因此送船的时间要尽可能的短，因此我们希望每次送船的都是所需要的时间最少的人。

每次只能载两人过河，需要考虑将两个人送过去的最短时间，可以从所需时间最多的两个人开始考虑，考虑将两个人送过去然后船回来所需要的最短时间，那么可以分两种过河的方式。第一种是让时间最短的人送船回来，然后把其他人载过河，那么总时间就是载人需要的时间和送船需要的时间；第二种是，让需要时间最少的和次少的先过河，然后让所需时间最少的人回来送船，然后让所需时间最多的和所需时间次多的人过河，让留在那里的需要时间次少的人回来送船。

两种方式最后的结果都是将两个人送到对面并把船开回了出发岸。因此可以每送两人都比较一下所需要的时间，选取所需时间最少的那种方式。

首先对整个的过河时间进行排序，然后对过河时间从最高的人开始遍历，每次考虑将两个所需时间最多的两个人送过去的方式，选取所消耗的时间最短的方式，遍历完数组得到结果。

代码段3 POJ1700

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

int a[1009];

int main()

{

int t;

scanf("%d",&t);

while(t--)

{

int n,i,j,m,ans=0;

scanf("%d",&n);

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

{

scanf("%d",&a[i]);

}

sort(a+1,a+1+n);//排序，递增

m=n;//m保存当前需要过河的人的位置

while(m>0)

{

if(m>=4)//剩余人数

{

if(a[m]+a[m-1]+2\*a[1]>a[2]+a[1]+a[m]+a[2])//每次要考虑最慢的两个人怎么过河

{

ans+=a[2]+a[1]+a[m]+a[2];//第1个和第2给先过去，然后最快的回来，两个慢的过去，然后第二快的人再回去

}//结束后最慢的俩个人已经在河对岸了

else//第一个载最慢的过去，再回来，然后载第二慢的人过去，再回来

{

ans+=a[m]+a[m-1]+2\*a[1];

}//结束后最慢的两个人已经在河对岸

m-=2;

}

else if(m==3)

{

ans+=a[m]+a[1]+a[2];

m=0;

}

else if(m==2)

{

ans+=a[m];

m=0;

}

else if(m==1)

{

ans+=a[m];

m=0;

}

}

printf("%d\n",ans);

}

}

## 

## **4.3性能分析**

首先，假定问题的规模为n（过河的人数），排序需要（nlogn)的时间，循环需要进行n/2次，每一次的循环中，循环体的时间复杂度为O（1），故整个算法的时间复杂度为O（nlogn)。

## **4.4运行测试**

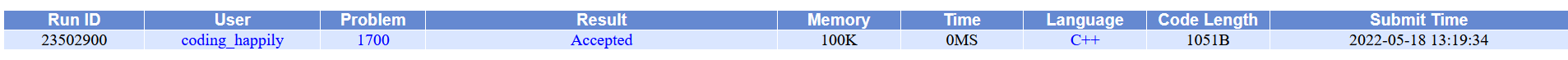


图5 POJ1700AC截图

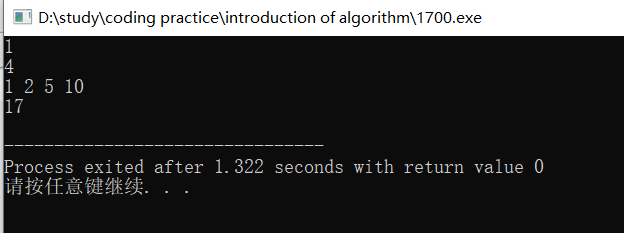


图6 POJ1700运行输出截图

# **5 POJ3660解题报告**

## **5.1题目分析**

POJ3660：Cow Contest

有 N 头奶牛在比赛，给出一系列条件，形如 A 战胜了 B。若 A 战胜 B，B 战胜 C，则我们认为 A 也能战胜 C。问，根据这一系列条件，有多少牛的排名是可以确定的？

## **5.2 算法设计**

采用图算法中的Floyd算法求图的传递闭包。

在这个题目中，如果A牛与其他的N-1头牛的胜败关系能够确定，那么A牛的排名就可以确定了，那么也就是在初始化关系矩阵之后采用floyd算法计算了图的传递闭包以后，我们可以查看该牛与其他牛是否已经能够确定关系，也就是传递闭包该点与其他点是否都存在路径，如果是，那么该牛可以确定自己的排名，否则就不可以。

代码段4 POJ3660

#include <iostream>

#include <cstdio>

#include <cstring>

using namespace std;

int n,m;

int d[105][105];

void floyd()//floyd算法确定闭包矩阵

{

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

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

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

{

if(d[i][j]||(d[i][k]&&d[k][j]))

d[i][j]=1;

}

return;

}

int main()

{

int x,y;

while(scanf("%d%d",&n,&m)!=EOF)

{

memset(d,0,sizeof(d));

while(m--)

{

scanf("%d%d",&x,&y);

d[x][y]=1;

}

floyd();//对所有可以确定胜败关系的牛标记

int ans=0,num=0;

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

{

num=0;//初始化和该牛确定了胜败关系的数量

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

{

if(d[i][j]==1||d[j][i]==1)

num++;

}

if(num==n-1)//说明和其他的牛都存在胜败关系，那么该牛在排名中的位置可以确定

ans++;//记录可以确定排名的牛

}

printf("%d\n",ans);

}

return 0;

}

## **5.3性能分析**

假设问题的规模为n（为牛的数量），Floyd算法确定闭包矩阵的时间复杂度为O（n³），遍历矩阵确定有排名的牛的数量需要O（n²），故算法的时间复杂度为O（n³）。

## **5.4运行测试**



图7 POJ3660AC截图

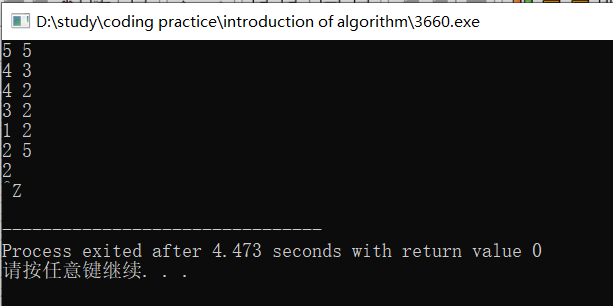


图8 POJ3660运行输出截图

# **6. 总结**

## **6.1 实验总结**

在这次的实验中，涉及到的题目有分治、动态规划、贪心算法和图的搜索与最短路径算法。

这些题目让我更加深刻的体会到了在课上学习到的理论知识。在分治、动态规划、贪心算法这三个反面，体会到了更多的算法设计思想；在图算法中，更加清楚了几个经典的图算法和它们的具体代码的实现。

在分治思想中，学习理解到了怎样分解问题的思想，并且学习了递归的妙用和递归化为递推的方法，将问题更加快速和高效的求解出来。

在动态规划中，学习理解了怎么分解子问题的思想，与分治不同的是，动态规划需要找问题的最优解，那么需要找到问题的最优子结构，采用数组保存子问题的最优解以及它们的状态。在这些题目中，我体会到了如何巧妙的设置数组来保存当前最优解和查找当前最优解与子问题最优解之间的关系。

在贪心算法中，学习理解了怎么找当前最贪心的一步，怎么确定这是当前的局部最优解，并将其转换为了一个更小的什么问题，在这些题目中，对这些问题的求解有了更深的体会。

在图算法中，将课上所学到的经典图算法的伪代码具体实现了出来，更加深刻的体会了这些算法的思路。

## **6.2 心得体会和建议**

这门实验课是和理论课结合很密切的实践课，将课上所讲的算法设计思想和经典图算法都包含了进去，让我们对理论课所学的东西有了更深刻的理解，同时还锻炼了我们的动手能力。

本次的实验都是在POJ上面完成的，在这个平台上，只有一个样例提供并且没有错误信息，因此这就需要我们自己在审题和写代码的过程中想到所有的边界情况和特殊的输入，以此来确定自己的代码是否有足够的健壮性。这对我来讲也是一个很大的挑战，因为之前的代码练习中大多都有错误样例提供给我们，因此我们可以很快的debug找出自己的逻辑错误，而现在需要我们脱离之前所依赖的样例，确实是一个不小的挑战。

当然，在这门课中也是收获颇多，因为代码实现本来就是需要学习很多的过程了，而测试+debug更是让我学习了更多的东西，也有了更“痛苦”的经历。