# 1 POJ2506解题报告

## 1.1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。

## 1.2 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;

}

## 1.3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))。

## 1.4运行测试



图1 POJ2506AC截图

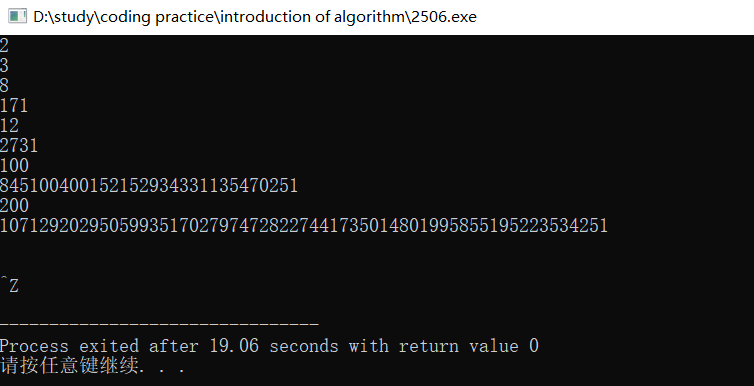


图2 POJ2506运行输出截图

# 2 POJ2228解题报告

## 2.1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。

## 2.2 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;

}

## 2.3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）。

## 2.4运行测试

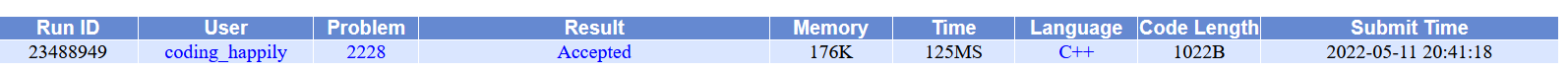


图3 POJ2228AC截图

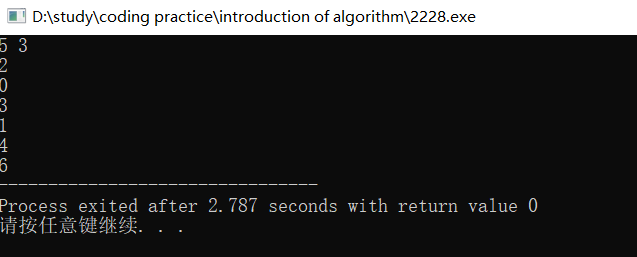


图4 POJ2228运行输出截图

# 3 POJ1700解题报告

## 3.1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。

## 3.2 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

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);

}

}

## 3.3性能分析

由于map中使用平衡二叉树进行数据的存储，因此每次查找所需要的时间为，则总的程序运行的时间复杂度为。

## 3.4运行测试

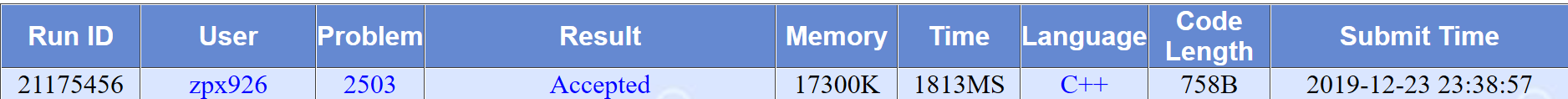


图5 POJ2042AC截图

图6 POJ1042运行输出截图

# 4 POJ2503解题报告

## 4.1题目分析

POJ2503：Babelfish

输入一个字典，字典格式为n行“英语 外语”的对照表。然后是若干个外语单词，让你翻译，输出它们对应的英语单词。如果字典中不存在这个单词，则输出”eh”。

## 4.2 算法设计

题目给出的输入格式是一对一的，一个英语对应一个外语单词，因此从设计思路上用map较为简单。首先根据读入的单词写入两个map中，一个储存英文单词和是否出现过，另一个储存英文单词和对应的外语单词。

在查找时，首先搜索是否存在的map，如果未搜索到，则直接输出”eh”；如果是否存在为真，则继续搜索第二个map，将对应的外语单词进行输出。

代码段1 POJ2503

#include<iostream>

#include<string>

#include<map>

using namespace std;

int main(void)

{

return 0;

}

## 4.3性能分析

由于map中使用平衡二叉树进行数据的存储，因此每次查找所需要的时间为，则总的程序运行的时间复杂度为。

## 4.4运行测试

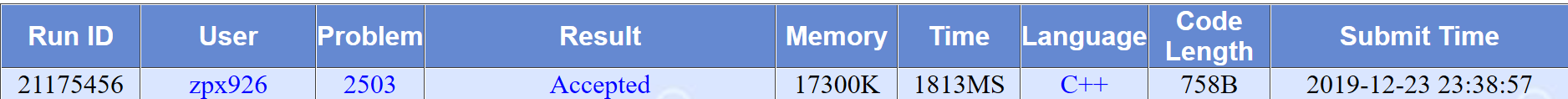


图1 POJ2503AC截图

xxx运行输出截图