# 第三次上机

## 地址

第三次上机

## 题目列表

- 923 Bamboo的小吃街
- 924 Bamboo和巧克力工厂
- 905 AlvinZH的奇幻猜想——三次方
- 915 双十一的抉择
- 930 ModricWang's Polygons
- 936 ModricWang的导弹防御系统
- 904 Winter is coming

# 解题报告

# A Bamboo的小吃街

## 分析

经典的两条流水线问题，题目描述基本类似于课件中的流水线调度，符合动态规划最优子结构性质
关键的动态规划式子为：
dp[0][j] = min(dp[0][j - 1], dp[1][j - 1] + t[1][j - 1]) + p[0][j] //保存在左边第j个店铺时已经用的时间
dp[1][j] = min(dp[1][j - 1], dp[0][j - 1] + t[0][j - 1]) + p[1][j] //保存在右边第j个店铺时已经用的时间 即到达i边第j个店铺，可以从i边第j-1个店铺过来，也可以从另一边的j-1个店铺过来，后者需要加上过马路的时间；两者都要加上在第i边第j个店铺停留的时间
最后比较dp[0][n-1]和dp[1][n-1]，找到最小值即为所求。

循环店铺数，同时更新两边的时间

### 这是上机时给出的伪代码

```
dp[2][maxx]
p[2][maxx]
t[2][maxx]
//dp[0][j]  在左边第j家店时已经用的时间
//p t 如题所示
dp[0][0]=p[0][0]
dp[1][0]=p[1][0]
for j=1:n
    dp[0][j] = min(dp[0][j - 1], dp[1][j - 1] + t[1][j - 1]) + p[0][j]
        dp[1][j] = min(dp[1][j - 1], dp[0][j - 1] + t[0][j - 1]) + p[1][j]
end
取两者中较小者
```

### 代码

```
const int maxx = 510;
int dp[2][maxx];
int p[2][maxx];
int t[2][maxx];
int main()
{
        int n;
        while (~scanf("%d", &n))
        {
                memset(dp, 0, sizeof(dp));
                int i, j;
                for (i = 0; i<2; i++)
                        for (j = 0; j<n; j++)
                                scanf("%d", &p[i][j]);
                for (i = 0; i<2; i++)
                        for (j = 0; j<n - 1; j++)
                                scanf("%d", &t[i][j]);

                dp[0][0] = p[0][0];
                dp[1][0] = p[1][0];
                for (j = 1; j<n; j++)
                {
                        dp[0][j] = min(dp[0][j - 1], dp[1][j - 1] + t[1][j - 1]) + p[0][j
                        dp[1][j] = min(dp[1][j - 1], dp[0][j - 1] + t[0][j - 1]) + p[1][j
                }
                long long  ans = dp[0][n - 1]<dp[1][n - 1] ? dp[0][n - 1] : dp[1][n - 1];
                printf("%lld\n", ans);
        }
}
```

# B Bamboo和巧克力工厂

## 分析

三条流水线的问题，依然是动态规划，但是涉及的切换种类比较多。比较易于拓展到n条流水线的方式是三层循环，外层是第k个机器手，里面两层代表可切换的流水线
核心dp语句：cost[i][k] = min(cost[i][k], cost[j][k-1]+t[j][i]+p[i][k]) 也可以在A题的基础上详细的列出所有可能的路线切割情况然后找到最小值。

注意本题与A题中t的含义不同

### 上机时给出的伪代码

```
//数组从0开始
const int maxx= 510;
 p[3][maxx];
 t[3][3];
 cost[3][maxx];//dp主体
N=2;  M为输入值
void resolve(int N,int M)
{
    初始化cost为无穷大
    三条流水线的初始值为对应的P值：
    cost[i][0] = p[i][0];
    for k=1: M-1
        for i=0:2
            for j = 0:2
            cost[i][k]  =  min(cost[i][k], cost[j][k-1]+t[j][i]+p[i][k]);
            end
         end
     end

    找出三条线路中在最后一个机器手时总用时的最小值即为所求
}
```

### 代码

```
const int maxx= 510;
int p[3][maxx];
int t[3][3];
int cost[3][maxx];
const int MAX = (1<<30);
void Resolve(int n,int m)
{
    int i,j;
```

```
    for(i= 0; i<n; i++)
        for(j =0; j<m; j++)
            cost[i][j]  = MAX ;
    for(int i= 0; i<n; i++)
        cost[i][0] = p[i][0];

    for(int k = 1; k<m; k++)
        for(int i = 0; i<n; i++)
            for(int j = 0; j<n; j++)
            {
                cost[i][k]  =  min(cost[i][k], cost[j][k-1]+t[j][i]+p[i][k]);
            }
    int ans = 500000000;
    for( i  =0; i<n; i++)
    {
        if(cost[i][m-1]<ans)
            ans = cost[i][m-1];
    }
    printf("%d\n",ans);
}
int main()
{
    int m;
    while(scanf("%d",&m)!=EOF)
    {
        int i,j;
        for(i = 0; i<3; i++)
            for( j = 0; j<m; j++)
                scanf("%d",&p[i][j]);
        for(i= 0; i<3; i++)
            for( j = 0; j<3; j++)
                scanf("%d",&t[i][j]);
        Resolve(3,m);
    }
}
```

//发现上机时有了伪代码还没过的同学主要是失误在输入上，两题的t含义并不同

# 905 AlvinZH的奇幻猜想——三次方

## 思路

中等题。题意简单，题目说得简单，把一个数分成多个立方数的和，问最小立方数个数。

脑子转得快的马上想到贪心，从最近的三次方数往下减，反正有1^3在最后撑着保证减完。不好意思这是错的，应为1,27,64,125...等立方数之间并不是倍数关系，不能构成贪心策略。举个反例：96=64+8+8+8+8=64+27+1+1+1+1+1，答案明显是5，二贪心会算到7。

既然不是贪心，那就是DP了，没毛病。先讲一下常规做法吧，是这样想的：相当于把一个数化成几份，求最小划分的份数，那么把所求数看成背包的总体积，每个立方数的值看作每个物品的体积，价值都看做1，问题就转化为完全背包（因为每个立方数可以取多次）恰好装满求最小的价值，仔细想想。

所以，套一下完全背包的板子吧，但是，这里的一个问题是**完全背包装满**，这怎么办呢（可能由于本题可以保证装满这个问题不怎么显眼）。举一反三，假设有可能装不满，怎么办？这个问题初始化dp数组时就可以解决。以下方法对于01背包同样适用：

- 普通01背包or完全背包：初始化为0；
- 01背包or完全背包装满求价值最小：初始化为一个大数值如 $INF$ (0x3f3f3f3f)；
- 01背包or完全背包装满求价值最大：初始化为一个小数值如 $-INF$ (-0x3f3f3f3f)；

为什么呢？对于本题，初始化为大数值，如果没装满，最后dp[n]会依然是INF，因为我们每次比较取的都是较小值。第二次练习赛的G题就是01背包装满求最大价值，初始化为最小值即可。（熬夜写了这么多，希望大家看得到QAQ）

这题还有另外的解法，那就是类似打表，先把所有数的最小立方个数通过迭代计算得到，然后 $O(1)$ 时间取得答案。具体可见队列迭代的参考代码二以及for循环迭代的参考代码三。其实这里面也是有DP的思想，因为每次迭代会用到之前的结果，对于多组数据来讲，同样可以节省时间。简单易懂，可以学习一下。

## 分析

对于完全背包直接解法，时间复杂度为 $O(V*\sum(V/wi))$ 。

迭代的话复杂度差不了多少，由于多组数据的原因，迭代打表运行总时间显得更短些，不纠结这个。

扩展：完全背包装满问题：HDU 1114。

## 参考代码一：完全背包装满求最小价值

```
//
// Created by AlvinZH on 2017/10/24.
// Copyright (c) AlvinZH. All rights reserved.
//

#include <cstdio>
#include <cstring>
#define INF 0x3f3f3f3f
#define MaxSize 1000005

int weight[105];
int ans[MaxSize];

int main()
```

```
{
    for (int i = 1; i < 105; ++i)
        weight[i] = i * i * i;

    int n;
    while(~scanf("%d", &n))
    {
        for (int i = 0; i <= n; ++i)//初始化为最大值
            ans[i] = INF;

        ans[0] = 0;

        for (int i = 0; i < 105; ++i) {
            for (int j = weight[i]; j <= n; ++j) {
                if(ans[j] > ans[j-weight[i]] + 1)
                    ans[j] = ans[j-weight[i]] + 1;
            }
        }
        if(ans[n] == n) printf("Oh NO!\n");
        else printf("%d\n", ans[n]);
    }
}

/*
 * 完全背包恰好装满问题，求最小值。
 */
```

## 参考代码二：队列迭代

```
//
// Created by AlvinZH on 2017/10/24.
// Copyright (c) AlvinZH. All rights reserved.
//

#include <cstdio>
#include <cstring>
#include <queue>
#define INF 0x3f3f3f3f
#define MaxSize 1000005
using namespace std;

int weight[105];
int ans[MaxSize];
queue<int> Q;

void init()
{
    memset(ans, INF, sizeof(ans));
    for (int i = 1; i < 105; ++i)
```

```
        weight[i] = i * i * i;

    for (int i = 1; i <= 100; ++i) {
        ans[weight[i]] = 1;
        Q.push(weight[i]);
    }
    while(!Q.empty())
    {
        int w = Q.front();
        Q.pop();
        for (int i = 1; i <= 100; ++i) {
            int num = weight[i] + w;
            if(num > 1000000) break;
            if(ans[num] < INF) continue;
            ans[num] = ans[w] + 1;
            Q.push(num);
        }
    }
}

int main()
{
    init();
    int n;
    while(~scanf("%d", &n))
    {
        if(ans[n] == n) printf("Oh NO!\n");
        else printf("%d\n", ans[n]);
    }
}
/*
 * 思路：直接宽搜，把最开始的数扔进队列，反复用队列中的数去更新没更新的数就行了，注意数组别越界。
 */
```

## 参考代码三：for循环迭代

```
/*
 Author: 曾宥崴(13422)
 Result: AC      Submission_id: 391756
 Created at: Fri Nov 10 2017 18:26:40 GMT+0800 (CST)
 Problem: 905    Time: 353       Memory: 6612
*/

#include <cstdio>
#include <algorithm>
#include <iostream>
using namespace std;
```

```
int f[1000001],n;

int main()
{
    for (int i = 1; i <= 1000000; i++) f[i] = 1000000000;

    f[0] = 0;
    for (int i = 0; i <= 1000000; i++)
        for (int k = 1; i + k * k * k <= 1000000; k++)
            f[i+k*k*k] = min(f[i+k*k*k],f[i] + 1);

    while (scanf("%d",&n) != EOF)
        if (f[n] == n) printf("Oh NO!\n");
        else printf("%d\n",f[n]);

    return 0;
}
```

# 915 双十一的抉择

## 思路

中等题。简化题目：一共n个数，分成两组，使得两组的差最接近0，就是说要使两组数都尽可能的接近sum/2。

思路还是很混乱的，不知道如何下手，暴力也挺难的，还不能保证对。想一想，从一堆数中取出一些使得和尽可能接近sum/2，把sum/2当作背包总体积，每个数字当作每件物品的体积，价值都是为1，求的就是最大价值。完完全全的01背包问题，问题解决，具体可见参考代码。

这里就不再详细讲解01背包了，请仔细研读《背包九讲》，务必学习到经典DP问题之背包问题的精髓。

## 分析

01背包的时间复杂度是 $O(V*N)$。

## 参考代码

```
//
// Created by AlvinZH on 2017/10/24.
// Copyright (c) AlvinZH. All rights reserved.
//

#include <cstdio>
```

```
#include <cstring>

int n, sum;
int V;//背包体积
int N[1005];//把数量同时看作物品的体积和价值
int dp[50005];

int main()
{
    while(~scanf("%d", &n))
    {
        sum = 0;
        memset(dp, 0, sizeof(dp));
        for (int i = 1; i <= n; ++i) {
            scanf("%d", &N[i]);
            sum += N[i];
        }

        V = sum / 2;//背包的体积
        //01背包
        for (int i = 1; i <= n; ++i) {
            for (int j = V; j >= N[i]; --j) {
                int temp = dp[j - N[i]] + N[i];
                if(dp[j] < temp) dp[j] = temp;
            }
        }

        if(sum - 2 * dp[V] == 0) printf("GF&SI\n");
        else printf("%d\n", sum - 2 * dp[V]);
    }
}

/* 分析：一共n个数，分两组，使得两组的差最接近0，就是说要使两组数都尽可能的接近sum/2。
 * 很自然的想到这是在类似于取东西，看怎么取得平均。
 * 所以只需要对一组进行求算使它最接近sum/2，那样的话另一组自然也是最接近的。
 * 典型的01背包问题，sum/2作为背包总体积，每袋糖数量作为价值和体积，求最多可以装多少。
 */
```

# 930 ModricWang's Polygons

## 思路

首先要想明白，哪些多边形可能是格点正多边形？

分情况考虑：

三角形不可能，因为边长为有理数的正三角形的面积为无理数，而格点三角形的面积为有理数，二者矛盾。

正四边形毫无疑问是可以的。

边数>4时，可以考虑无穷递降法：

以六边形为例，假如整点正六边形存在，一定有边长最小的一个，记作$A_1 A_2 A_3 A_4 A_5 A_6$。以$A_2$为中心，将$A_1$逆时针旋转90度，得到$B_1$。显然也是整点。类似定义$B_2 ... B_6$，它们也都是整点。【注：上面用到如下结论：一个整点绕另一个整点旋转90度，得到的点仍然是整点。证明很容易。】如你所见，$B_1 B_2 B_3 B_4 B_5 B_6$是一个更小的整点正六边形，矛盾。

对于边数>4的其他情况，也可以用类似方法证明。

参考自这里

于是该问题就变成了统计格点正四边形的数量。考虑枚举任意两个点形成的线段，那么由这一条线段可以形成的格点正四边形只有两个，并且另外两个点的坐标是可以直接算出来的。因此只要枚举任意两点的组合，验证以这两个点为相邻两点的格点正四边形是否存在即可。

时间复杂度$O(n^2)$，空间复杂度$O(n)$

## 代码

```
#include <bits/stdc++.h>
#define ll long long
#define mk make_pair
#define y1 yyy
using namespace std;

const int N = 1e3 + 5;

map<pair<int, int>, bool> M;
int x[N], y[N], n, ans;

int main() {
  int T;
  while (scanf("%d", &n) != EOF) {
    ans = 0;
    M.clear();
    for (int i = 1; i <= n; i++) {
      scanf("%d %d", x + i, y + i);
      M[mk(x[i], y[i])] = 1;
    }
    for (int i = 1; i <= n; i++) {
      for (int j = i + 1; j <= n; j++) {
        int dx = y[j] - y[i];
        int dy = x[i] - x[j];
        int ok = 0;
        if (M.count(mk(x[i] + dx, y[i] + dy)))
          ok++;
        if (M.count(mk(x[j] + dx, y[j] + dy)))
```

```
                ok++;
            if (ok == 2)
                ans++;
            ok = 0;
            if (M.count(mk(x[i] - dx, y[i] - dy)))
                ok++;
            if (M.count(mk(x[j] - dx, y[j] - dy)))
                ok++;
            if (ok == 2)
                ans++;
        }
    }
    printf("%d\n", ans / 4);
}
```

# 936 ModricWang的导弹防御系统

## 思路

题意即为：给出一个长度为n的序列，求出其最长不降子序列。

考虑比较平凡的DP做法：

令$nums[i]$ 表示这个序列，$f[x]$ 表示以第$x$个数为结尾的最长的不降子序列的长度，状态转移方程为： $f[i] = (\max f[j] + 1) \qquad \text{when } nums[i] <= nums[j]$

f中的最大值即为答案。

时间复杂度$O(n^2)$，空间复杂度$O(n)$

当然也可以用时间$O(nlogn)$的方法做，不过数据这么小，用$O(n^2)$就可以了。

## 代码

```
#include <iostream>

using namespace std;

const int MaxN = 1023;

int nums[MaxN], f[MaxN];

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
```

```
    cout.tie(0);
    int n, ans;
    cin >> n;
    for (int i = 0; i < n; i++)
        cin >> nums[i];
    ans = f[0] = 1;
    for (int i = 1; i < n; i++) {
        f[i] = 0;
        for (int j = 0; j < i; j++)
            if (nums[j] >= nums[i] && f[j] >= f[i]) f[i] = f[j] + 1;
        if (f[i] > ans) ans = f[i];
    }
    cout << ans << "\n";
}
```

# 904 Winter is coming

## 思路

难题。首先简化问题， $n$ 个0与 $m$ 个1排成一列，连续的0不能超过x个，连续的1不能超过y个，求排列方法数。

显然会想到这是动态规划。最快想到的方法是 $dp[i][j][x][y]$ 表示已经有i个北境兵j个野人参与排列，且末尾有x个连续北境士兵或y个连续野人士兵的方案数。这方法显然是正确的，但是光是 $dp[200][200][10][10]$ 数组已经十分接近本题内存限制了，保证MLE。状态转移方法是大模拟，四层for循环，每次增加一个人放在最后，讨论各种情况。具体代码可见MLE参考代码，比较好理解。

不过这个方法已经和答案很接近了，只需要稍微优化一下。可以发现第三维和第四维很多空闲的空间被浪费了，我们没有必要用两个维度来分别记录有几个0或1，可以把第四维变成一个标志位，0代表北境军，1代表野人军，而第三维记录最后一段连续的人数，这样空间变成原来的1/6，算是简单的优化了一下，思想并没有变。具体可见优化代码，在此感谢孟尧提供。

本题还可以继续优化，换种方式，dp[i][j][k]：已经有i个北境兵j个野人参与排列，第三维k是标志位（0代表北境军，1代表野人军）的排列方法数。状态转移方程变为：： $dp[i][j][0]=\sum(dp[i-k][j][1])\%MOD$ ；其中 $k\in[1,min(i,x)]$ 。同理， $dp[i][j][1]=\sum(dp[i][j-k][0])\%MOD$ ；其中 $k\in[1,min(j,y)]$ 。相信你很快就能看懂，这里用 $dp[i-k][j][1]$ 来代表最后有k个0，相当于同时把三四维合并了，巧妙至极。具体可见最优参考代码。

## 分析

本题不卡时间，卡的是内存。目的是让大家在解决问题的时候有优化的思想（实际的目的是把它从中等题变成难题）。

DP只能意会，不可言传。大家在做DP题的时候一定要理清思路，一般是先不管空间，毕竟以空间换时间，大多数题都是先卡时间再卡空间的。

以本题为例粗略讲解一下DP，以后不会再讲。记住DP具备的两个要素：最优子结构和子问题重叠，见《算法导论》225页。本问题明显备最优子结构，最少的排列数是由多个较短一点的最少排列数组成。DP的多层循环也是有规律的，因为子问题的重叠，你得先把子问题算出来，才能计算更深层的。这里i和j从小到大地计算，保证所加上的都是已经计算过的，才不会出现问题，如果这题加一个dp[i+1][j][1]，那明显不对了，因为这个还没有计算过。状态转移方程有时候很微妙，需要一番数学推理。

## 最优参考代码

```
//
// Created by AlvinZH on 2017/10/24.
// Copyright (c) AlvinZH. All rights reserved.
//

#include <cstdio>
#include <cstring>
#include <iostream>
#define MOD 1000007
using namespace std;

int n, m, x, y;
int dp[205][205][2];

int main()
{
    while(~scanf("%d %d %d %d", &n, &m, &x, &y))
    {
        memset(dp, 0, sizeof(dp));
        for (int i = 0; i <= x; ++i)
            dp[i][0][0] = 1;
        for (int i = 0; i <= y; ++i)
            dp[0][i][1] = 1;

        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= m; ++j) {
                for (int k = 1; k <= min(i,x); ++k)
                    dp[i][j][0] = (dp[i][j][0] + dp[i-k][j][1]) % MOD;

                for (int k = 1; k <= min(j,y); ++k)
                    dp[i][j][1] = (dp[i][j][1] + dp[i][j-k][0]) % MOD;
            }
        }

        printf("%d\n", (dp[n][m][0] + dp[n][m][1]) % MOD);
    }
}
```

```
/* 把第三四维合并，因为我们只要在状态转移的时候保证最后连续一段不超过x或y就好了，第三维用来记录最
 * dp[i][j][k]：已经有i个北境兵j个野人参与排列,k为标志位（0代表北境军，1代表野人军）的排列方法数
 */
```

## 优化参考代码

```
/*
 Author: 孟爻(12593)
 Result: AC	Submission_id: 403884
 Created at: Sun Nov 12 2017 23:05:10 GMT+0800 (CST)
 Problem: 904	Time: 73	Memory: 11232
*/

#include <cstdio>
#include <cstring>

long f[205][205][15][2];
long M = 1000007;
long n,m,x,y;

int main() {
    while(~scanf("%ld%ld%ld%ld",&n,&m,&x,&y))
    {
        memset(f,0,sizeof(f));
        f[0][0][0][0]=1;
        for(long i=0; i<=n; i++) {
            for(long j=0; j<=m; j++) {
                if(i) {
                    for(long k=1; k<=x; k++)
                        f[i][j][k][0]=(f[i][j][k][0]+f[i-1][j][k-1][0])%M;
                    for(long k=0; k<=y; k++)
                        f[i][j][1][0]=(f[i][j][1][0]+f[i-1][j][k][1])%M;
                }
                if(j) {
                    for(long k=1; k<=y; k++)
                        f[i][j][k][1]=(f[i][j][k][1]+f[i][j-1][k-1][1])%M;
                    for(long k=0; k<=x; k++)
                        f[i][j][1][1]=(f[i][j][1][1]+f[i][j-1][k][0])%M;
                }
            }
        }
        long ans=0;
        for(long k=1; k<=x; k++)
            ans=(ans+f[n][m][k][0])%M;
        for(long k=1; k<=y; k++)
            ans=(ans+f[n][m][k][1])%M;

        printf("%ld\n",ans);
```

```
    }
}
```

## MLE代码

```
//
// Created by AlvinZH on 2017/10/24.
// Copyright (c) AlvinZH. All rights reserved.
//

#include <cstdio>
#include <cstring>
#define MOD 1000007

int n, m, x, y;
int dp[205][205][12][12];

int main()
{
    while(~scanf("%d %d %d %d", &n, &m, &x, &y))
    {
        memset(dp, 0, sizeof(dp));
        dp[0][0][0][0] = 1;
        for (int i = 0; i <= n; ++i) {
            for (int j = 0; j <= m; ++j) {
                for (int k = 0; k <= x; ++k) {
                    for (int l = 0; l <= y; ++l) {
                        if(dp[i][j][k][l] == 0) continue;
                        if(i != n && k != x)//末尾是北境兵
                        {
                            dp[i+1][j][k+1][0] += dp[i][j][k][l];
                            dp[i+1][j][k+1][0] %= MOD;
                        }
                        if(j != m && l != y)//末尾是野人兵
                        {
                            dp[i][j+1][0][l+1] += dp[i][j][k][l];
                            dp[i][j+1][0][l+1] %= MOD;
                        }
                    }
                }
            }
        }
        int ans = 0;
        for (int i = 1; i <= x; ++i)
            ans = (ans + dp[n][m][i][0]) % MOD;
        for (int i = 1; i <= y; ++i)
            ans = (ans + dp[n][m][0][i]) % MOD;

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

```
    }
}

/*
 * dp[i][j][x][y]表示已经有i个北境兵j个野人参与排列，且末尾有x个连续北境士兵或y个连续野人士兵的
 */
```

---

**BUAA-Soft-Algo-2016 is maintained by modricwang, Pacsiy and JoJoJun.**
This page was generated by GitHub Pages.