题解一

链接：<http://acm.hust.edu.cn/vjudge/contest/view.action?cid=99901#problem/C>

题意：求有权无向图的最小环，环至少包括三个点。

思路：

设map[i,j]表示i到j的的距离。输入有重边，在处理输入的时候只保存最短边。

取环中一个点k，左右点是ij则map[i,k]和map[k,j]是固定的不能变，可改变的是没有加入k点的i,j之间的最短路,设为dist[i,j]。那么最短环的长度表示为dist[i,j]+map[i,k]+map[k,j]。

Floyd的最外层循环为k时，最短路还没有用k和更新ij之间的最短路，恰好符合要求。所以在还没有用k更新ij之间的最短路之前更新环，每次更新环时更新路径即可。

算法步骤：

输入输出Floyd算法的框架，三个循环，在更新路径前更新环的最短路径。题解一和其他的题解(都是DP)代码很短，步骤很清晰，所以算法步骤粗略写写。

算法复杂度：

输入输出两个循环不超过O(n2)，时间主要消耗在Floyd算法的框架里，三个n次循环，所以时间复杂度O(n3)，最多是二维数组，空间复杂度是O(n2)

代码：

const int maxn=110;

int dist[maxn][maxn], map[maxn][maxn]; //最短距离，原图

int pre[maxn][maxn]; // pre[i,j]记录最短路里，j前面一个点

int path[maxn]; // 答案路径

int n, m, num, minc; // num记录path里有多少个点，minc是最短环长度

int main()

{

int u, v, cost;

while(cin >> n && n){

if(n<0) break;

cin >> m;

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

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

dist[i][j]=map[i][j]=INF;

pre[i][j]=i;

}

}

for(int i=1; i<=m; i++){

scanf("%d %d %d",&u,&v,&cost);

if(dist[u][v]>cost) //重边

map[u][v]=map[v][u]=dist[u][v]=dist[v][u]=cost;

}

// floyd

minc=INF;

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

// k还没加入(i,j)最短路，更新最短环

for(int i=1; i<k; i++){

for(int j=i+1; j<k; j++){

int ans=dist[i][j]+map[i][k]+map[k][j];

if(ans<minc){ //找到最优解

minc=ans;

num=0;

int p=j;

while(p!=i){ //逆向寻找前驱遍历的路径并将其存储起来

path[num++]=p;

p=pre[i][p];

}

path[num++]=i;

path[num++]=k;

}

}

}

//用k更新i到j的最短路径

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

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

if(dist[i][j]>dist[i][k]+dist[k][j]){

dist[i][j]=dist[i][k]+dist[k][j];

pre[i][j]=pre[k][j];

}

}

}

}// end Floyd

if(minc==INF) puts("No solution.");

else{

printf("%d",path[0]);

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

printf(" %d",path[i]);

puts("");

}

}

return 0;

}

题解二

链接：<http://acm.hust.edu.cn/vjudge/contest/view.action?cid=99901#problem/B>

题意：长度小于100的字符串s只由四种字符"()[]"组成，求以该串为子串的最短的合法串。合法串递归定义为：

(1)空串合法

(2)如果S合法，则(S)、[S]合法

(3)如果A、B合法，则AB合法

思路：

设dp[i][j]为s(i,j)变为合法串后，合法串的长度或需要添加的字符的个数，状态转移：

(1)如果s[i]和s[j]匹配，dp[i,j]=dp[i+1,j-1]。

(2)如果不匹配，划分s(i,j)为s(i,k)和s(k+1,j)，划分后dp[i,j]=dp[i,k]+dp[k+1,j]，如果两个子串匹配的括号数最多，那么需要添加的字符比较最少，即dp[i,k+1]+dp[k+1,j]最小，k类似分段点，s(i,k)和s(k+1,j)可以分别输出，所以dp[i,j]=min(dp[i,k]+dp[k+1,j])

依赖关系：

求dp[i,j]要先求[i,k][k,j]，即大区间依赖于小区间，所以要从区间长度最小的开始遍历。要把小区间全部遍历完，就从0开始遍历起点。所以有

for len = 2 to n

for start = 0 to n-1

状态转移，遍历k

初始状态：

dp[i][i]一定需要匹配一个字符，所以dp[i,i]=1

输出

如果知道s中哪个是不能在s中匹配的，在该字符旁边加一个匹配的就可以了。s[i,j]需要dp[i,j]个字符匹配，分情况讨论情况：

如果i>j，非法区间

如果i==j，单个字符，直接输出对应的"()"或者"[]"

如果i<j，合法区间，分情况

如果s[i]和s[j]匹配，只需看s(i+1,j-1)看，递归输出 output(s[i]); output(i+1,j-1); output(j)；

如果s[i]和s[j]匹配不匹配，在dp[i,k]在状态转移时分段点k是唯一的，(i,k)和(k+1,j)是独立的，所以可以用一个pos[i,j]记录下k，然后分段输出，即output(i,k); output(k+1,j)

算法步骤：

步骤1：初始化dp[][]为无穷大，dp[i][i]=1，pos=-1。

步骤2：递推求dp[i][j]，然后根据pos递归输出新串。

代码：

int dp[110][110],pos[110][110];

void DFSprint(int l,int r) // 路径输出

{

if(l>r) return ;

if(l==r){

if(s[l]=='('||s[l]==')') cout << "()" ;

else cout << "[]" ;

}

else if(pos[l][r]==-1){

cout << s[l]; DFSprint(l+1,r-1); cout << s[r];

}

else{

DFSprint(l,pos[l][r]); DFSprint(pos[l][r]+1,r);

}

}

int main()

{

cin >> s ;

memset(dp,inf,sizeof(dp));

memset(pos,-1,sizeof(pos));

int len = s.size();

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

dp[i][i]=1,dp[i+1][i]=0;

for(int length=1; length<len; ++lenght){

for(int start=0; start+length<len; ++start){

dp[i][j]=len+1; // max

// s[i] s[j] 匹配

if( (s[i]=='('&&s[j]==')') || (s[i]=='['&&s[j]==']'))

dp[i][j]=min(dp[i][j], dp[i+1][j-1]);

// 状态转移

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

if(dp[i][j] > dp[i][k]+dp[k+1][j]){

dp[i][j] = dp[i][k]+dp[k+1][j];

pos[i][j] = k;

}

}

}

}

DFSprint(0,len-1);

cout << endl;

return 0;

}

题解三

链接：<http://acm.hust.edu.cn/vjudge/contest/view.action?cid=99901#problem/A>

题意：对整数序列(a0~an-1)求和，求和规则：不断取ai(0<i<n)，和加上ai-1\*ai\*ai+1，去掉ai，重复以上步骤直到只剩两个数。求最小和。

思路：

类似矩阵链相乘，设dp[i][j]为序列(i,j)之间最小的sum。对序列(i,j)，ai和aj最后会留下，所以最后一次求和一定是ai\*ak\*aj，(i<k<j)。以k为划分，则状态转移方程为dp[i,j] = min ( dp[i,j] , dp[i,k] + dp[k,j] + ai\*ak\*aj)

依赖关系分析:

求dp[i,j]要先求[i,k][k,j]，即大区间依赖于小区间，因此从区间长度最小的开始遍历。要遍历全部小区间，遍历起点。所以有

for len = min\_len to n

for start = 0 to n-1

/\*状态转移，遍历k \*/

初始状态:

dp设为无穷大，区间长度最小为3，可以先将dp[i,i+2]算好，递推时len从4开始枚举。这里dp[i,i+1]要设为0，做题时WA卡在这里，因为更新状态时k从i+1开始枚举，而j=i+len-1，则会出现dp[i,j] = dp[i,i+2]=dp[i,i+1] + dp[k,j] + ai\*ak\*aj，为了保证计算正确应该赋值为0。

算法步骤：

初始化，然后递推求dp[i][j][k]。

算法复杂度

时间复杂度O(n3),空间复杂度O(n3)

代码:

1、递推

#include <string>

#include <vector>

#include <cstdio>

#include <cstring>

using namespace std;;

const int inf = 0x3f;

int n,a[110];

int dp[110][110];

int main()

{

cin >> n;

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

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

// dp 初始化

memset(dp,inf,sizeof(dp));

for (int i=0;i<n-1;++i)

dp[i][i + 1] = 0;

for (int i=0;i<n-2;++i)

dp[i][i+2] = a[i]\*a[i+1]\*a[i+2];

for(int len=4;len<=n;++len){

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

int j=i+len-1;

if(j>n) break;

// find a minimal state for dp[i][j]

for(int k=i+1;k<j;++k)

dp[i][j]=min(dp[i][j],dp[i][k]+dp[k][j]+a[i]\*a[k]\*a[j]);

}

}

cout << dp[0][n-1] << endl;

return 0;

}

2、记忆化搜索

主函数执行f(0,n-1)就是答案，记忆化搜索代码如下：

int f (int a, int b){

if(dp[a][b]!=-1) return dp[a][b]; // 已经计算过了

if(b-a==1) return dp[a][b] = 0; // dp[i,i+1]

int min = MAXN;

for(int i = a+1;i <= b-1;i++){

int l = f(a,i), r = f(i,b);

if (min > l + r + x[a]\*x[i]\*x[b])

min = l + r + x[a]\*x[i]\*x[b];

}

return dp[a][b] = min;

}

题解四

链接：[http://acm.hust.edu.cn/vjudge/contest/view.action?cid=99312#problem/B](http://acm.hust.edu.cn/vjudge/contest/view.action?cid=99312%23problem/B)

题意：给定储蓄罐空的和满的重量，有n种硬币，硬币有价值和重量，给出各种硬币的价值p[i]和对应的重量w[i]，求储蓄罐里面硬币的最小价值，如果没有符合要求的放硬币的方式，输出 “this is impossible”。

思路：

相当于完全背包求最小值，n中硬币对应n个物体，物体可以取无限次，存储罐里硬币重量(满罐减空罐)相当于背包的体积V。

法一：

直接扩展01背包的方程，用dp[i,v]表示取前i种硬币，存储罐重量最大为v时的最小价值。状态转移方程为：dp[i,v]=min(dp[i-1,v-k\*w[i]] + k\*p[i])，0<=k<= V/w[i]，取第i中硬币取k个。

递推要求N\*V个状态，求每一个状态需要时间为O（v/Weight[i]），总的时间复杂度为O(NV\*Σ(V/c[i]))，空间复杂度O(n2)

递推式：

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

for(int v=w; v<=V; ++v)

for(int k=0; k\*v<=V; ++k)

dp[i][v] = min(dp[i-1][v],dp[i-1][v-k\*w]+p\*k);

法二：

1、递推代码：(对经典代码的解释)

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

for(int j=w; j<=V; ++j)

dp[j] = min(dp[j],dp[j-w]+p);

2、空间优化：

只用dp[v]即可，状态转移为：dp[v]=min(dp[v-k\*w[i]] + k\*p[i])，第i遍循环时，dp[i][v]只依赖于dp[i-1][v]的状态，前面的dp[0…i-1,v]都没用了，最后输出u的也是在dp[][v]里找结果，所以不用保存这些状态。这样需要保存两个dp[0][v]、dp[1][v]替换这两个变量就可以。

进一步想，k=0时，相当于dp[i][v]=dp[i-1][v]，接着枚举k都是和dp[i][v]比较的，也就是说，第i-1层循环算完后的结果相当于第i层的第一个状态，所以只需维护一个数组dp[v]就够了。

3、时间优化：

用第i个物品，即第i层循环，用dp[v-w]来更新dp[v]时，dp[v-w]已经更新过了，因为循环按照v = w to V来更新的，如下：

dp[v] = min(dp[v]未更新, dp[v-w]已更新)+p) // 更新dp[v]

= min ( dp[v]未更新, min(dp[v-w]未更新, dp[v-2w] 已更新+p)+p) // 更新dp[v-w]

= min ( dp[v]未更新, dp[v-w]未更新+p, min(dp[v-2w] 未更新, dp[v-3w] 已更新+p)+2p) // 更新dp[v-2w]

= min ( dp[v]未更新, dp[v-w]未更新+p, dp[v-2w] 未更新+2p, … , dp[v-kw] 未更新+kp) // 更新dp[v-3w]

= min (dp[v-k\*w] 未更新+k\*p) (0<=k<=V/vi)

算法复杂度：

时间复杂度O(nV)，空间复杂度O(n)

算法步骤

步骤1：读入数据，初始化dp[i]为无穷大，dp[0]=0

步骤2：递推，递推公式如上，答案就是dp[V]

代码：

#include <cstdio>

#define min(a,b) (a<b)?a:b

const int inf = 1e9,maxn = 8000;

int dp[maxn];

int main()

{

int kase,V1,V ,n;

int p, w; // value,weight

scanf("%d",&kase);

while(kase--){

scanf("%d %d %d",&V1,&V,&n);

V -= V1;

// init

for(int i=1;i<=V; ++i) dp[i]=inf;

dp[0]=0;

// DP

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

scanf("%d %d",&p,&w);

for(int j=w; j<=V; ++j){

dp[j] = min(dp[j],dp[j-w]+p);

}

}

// output

if(dp[V]==inf) printf("This is impossible.\n");

else printf("The minimum amount of money in the piggy-bank is %d.\n",dp[V]);

}

return 0;

}

题解五

链接：<http://acm.hust.edu.cn/vjudge/contest/view.action?cid=98136#problem/C>

题意：窗口可放n面红蓝白三种旗，规定同色不相邻，蓝在红白之间。共有多少种放置方法。

思路：

设dp[i][j]表示有i面旗，第i面旗填颜色j(j=01表示红白)时的总数，第i面填j色时，i-1可以填1-j(红白相间)或者蓝色，两种填法的计算：

(1)填1-j时有dp[i-1][1-j]种

(2)填蓝色时i-2和i要填红白色才能将i-1的蓝色包围，即i-2要填1-j，共dp[i-2][1-j]种

所以，状态转移方程为dp[i][j]=dp[i-1][1-j]+dp[i-2][1-j]。答案是dp[n][1]+dp[n][0]。

空间优化：

每一轮循环都要计算

dp[i][0] = dp[i-1][1]+dp[i-2][1]

dp[i][1] = dp[i-1][0]+dp[i-2][0]

两式相加得：(dp[i][0] + dp[i][1]) = (dp[i-1][0]+ dp[i-1][1]) + (dp[i-2][0]+dp[i-2][1])

两两结构相同，去掉第二维，dp[i] 表示前i面旗的放置方法总数，第i面旗可以填红白色。

由上面的方程得：dp[i] = dp[i-1] + dp[i-2]

直接给dp[i]分类很快，就是凑题解字数时，发现这种降维的方法，和背包的降维不一样。

状态转移分析：

第i面红色，i-1面白色：dp[i-1]是i-1填红白色的总数，红白各一半，白色有dp[i-1]/2种

第i面红色，i-1面蓝色：i-2必须是白色，则有dp[i-2]/2种

第i面白色，i-1面红色：同第一种，有dp[i-1]/2种

第i面白色，i-1面蓝色：同第二种，有dp[i-2]/2种

四个加起来得到dp[i]。

初始化：

dp[i]依赖于前两个，所以至少要提供两个连续的dp[i]，可以是dp[0] dp[1]或dp[1] dp[2]

算法步骤：

步骤1：初始化为dp[i]=0,dp[1]=2

步骤2：递推求dp[i]，答案是dp[n]

算法复杂度：

初始化和递推都是一个循环，时间复杂度O(n)，空间复杂度O(n)

代码：

#include <cstdio>

const int MAX\_N = 46;

long long int cnt[MAX\_N];

int main()

{

int n;

scanf("%d", &n);

cnt[1] = cnt[2] = 2;

for (int i = 3; i <= n; ++i)

cnt[i] = cnt[i - 1] + cnt[i - 2];

printf("%lld\n", cnt[n]);

return 0;

}

突然想到前面那个完全背包的输入，第i个物品用完就不用了，不用用数组保存。

这道题只要维护三个值dp[i] dp[i-1] dp[i-2]就可以了，也就是

c = a + b;

b = a;

a = c; -> a+b未更新

就是

a = a + b;

b = a – b; // (a旧值+b)-b

维护两个变量就行了OoO ，循环用while连for里面的局部变量也省了，代码好短

算法步骤见代码，main函数四行，一行输入，一行计算，一行输出，一行return 0

代码：

#include <cstdio>

int n;

long long a=2,b=0; // a=[1] b=a[0]

int main()

{

scanf("%d", &n);

while(--n) a = a + b , b = a - b;

printf("%lld\n", a);

return 0;

}