**北京邮电大学软件学院**

**2016－2017学年第一学期实验报告**

**课程名称： 算法设计与分析**

**项目名称： 动态规划法**

**项目完成人：**

**姓名：\_\_\_肖逸敏\_\_\_\_\_学号：\_\_\_2014211990\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**指导教师：\_\_\_\_\_李朝晖\_\_\_\_\_\_\_\_\_\_\_\_\_**

**日 期： 2016 年 12 月15 日**

1. **实验目的**
   1. 深刻理解并掌握动态规划法的设计思想；
   2. 提高应用动态规划法设计算法的技能；
2. **实验内容**

**基本题1：0—1背包问题（动态规划法）**

给定n种物品和一背包。物品i的重量是wi，其价值为vi，背包的容量为C。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大?

**基本题2：广告牌选取问题（动态规划法）**

假设你正在管理一条公路的广告牌建设，这条路从西到东M英里。广告牌可能的地点假设为x1,x2,x3…xn，处于［0,M]中。若在xi放一块广告牌，可以得到ri>0的收益。

国家公路局规定，两块广告牌相对不能小于或等于5英里之内。

如何找一组地点使你的总收益达到最大？

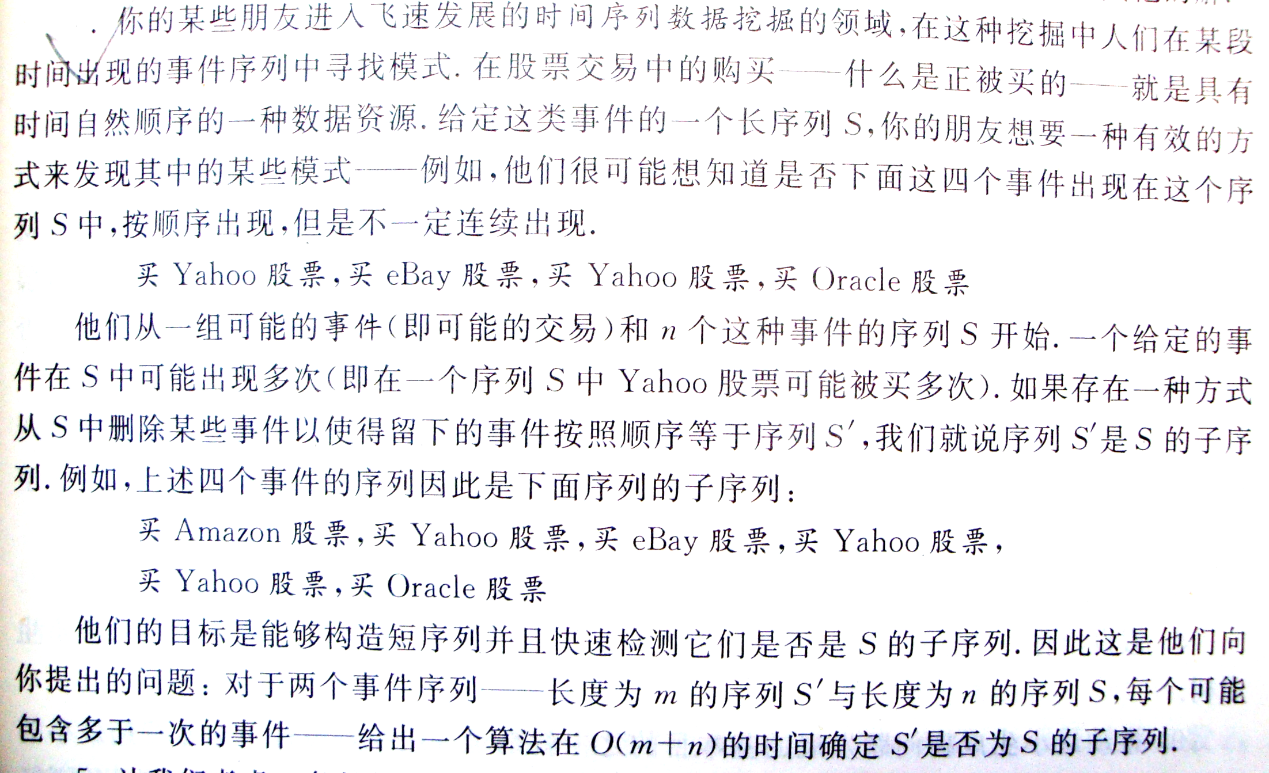
**提高题1：汽车加油行驶问题（动态规划法）**

给定一个N\*N的方形网格，设其左上角为起点，坐标(1，1)，X轴向右为正，Y 向下为正，每个方格边长为1。一辆汽车从起点出发驶向右下脚终点，其坐标为 (N,N)。在若干个网格交叉点处，设置了油库，可供汽车在行驶途中加油。汽车在行驶过程中应该遵守如下规则：  
 (1) 汽车只能沿网格边行驶，装满油后能行驶K条网格边。出发时汽车已装满油，起点与终点处不设置油库；  
 (2) 当汽车行驶经过一条网格边的时候，若其X坐标或者Y坐标减小，则应付费B ，否则免付费用；  
 (3) 汽车行驶过程中遇油库应该加油并付加油费用A；  
 (4) 在需要时可在网格点处增设油库，并付增设费用C(不含加油费用A)  
 (5) 上述(1)~(4)中的各数都是正整数

请找一条总费用最少的最优行驶路线。

Input   
   输入的第一行是N，K，A，B，C的值，2 ≤ N ≤ 100， 2 ≤ K ≤ 10。第二行起是一个N\*N 的0-1方阵，每行N 个值，至N+1行结束。方阵的第i 行第j 列处的值为1 表示在网格交叉点（i，j）处设置了一个油库，为0 时表示未设油库。各行相邻的2 个数以空格分隔。   
Output   
   程序运行结束时，将找到的最优行驶路线所需的费用，即最小费用输出  
   
Sample Input   
9 3 2 3 6  
0 0 0 0 1 0 0 0 0  
0 0 0 1 0 1 1 0 0  
1 0 1 0 0 0 0 1 0  
0 0 0 0 0 1 0 0 1  
1 0 0 1 0 0 1 0 0  
0 1 0 0 0 0 0 1 0  
0 0 0 0 1 0 0 0 1  
1 0 0 1 0 0 0 1 0  
0 1 0 0 0 0 0 0 0  
   
Sample Output   
12

**提高题2：**

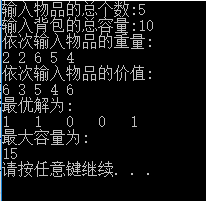


1. **实验环境**

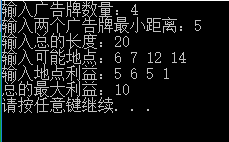
VS2013

1. **实验结果**

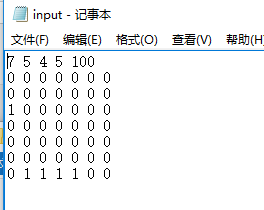
实验一：

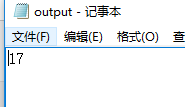


实验二：



实验三：





1. **附录**

实验一：

实验分析：

在0/1背包问题中，物体或者被装入背包，或者不被装入背包，只有两种选择。

循环变量i和j意义：前i个物品能够装入载重量为j的背包中

数组c意义：c[i][j]表示前i个物品能装入载重量为j的背包中物品的最大价值

若w[i]>j，第i个物品不装入背包，否则，若w[i]<=j且第i个物品装入背包后的价值>c[i-1][j]，则记录当前最大价值，替换为第i个物品装入背包后的价值。

实验代码：

#include<iostream>

using namespace std;

void KANPSACK\_DP(int c[50][50], int w[50], int v[50], int n, int C)

{

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

{

c[0][i] = 0;

}

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

{

c[i][0] = 0;

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

{

if (w[i] <= j)

{

if (v[i] + c[i - 1][j - w[i]] > c[i - 1][j])

c[i][j] = v[i] + c[i - 1][j - w[i]];

else c[i][j] = c[i - 1][j];

}

else c[i][j] = c[i - 1][j];

}

}

}

void OUTPUT\_SACK(int c[50][50], int x[50], int w[50], int n, int C)

{

for (int k = n; k >= 2; k--)

{

if (c[k][C] == c[k - 1][C])

x[k] = 0;

else {

x[k] = 1;

C = C - w[k];

}

}

x[1] = c[1][C] ? 1 : 0;

}

int main()

{

int c[50][50];

int w[50], v[50];

int x[50];

int C, n;

cout << "输入物品的总个数:";

cin >> n;

cout << "输入背包的总容量:";

cin >> C;

cout << "依次输入物品的重量:" << endl;

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

{

cin >> w[i];

}

cout << "依次输入物品的价值:" << endl;

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

{

cin >> v[i];

}

KANPSACK\_DP(c, w, v, n, C);

OUTPUT\_SACK(c, x, w, n, C);

cout << "最优解为:" << endl;

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

{

cout << x[i] << " ";

}

cout << endl << "最大容量为:" << endl;

cout << c[n][C] << endl;

system("pause");

return 0;

}

实验二：

实验分析：

考虑对于给定输入的一个最优解，在地点X(n)或者放广告牌，或者不放。

-如果不放，那么在地点X(1)，X(2)，……X(n)上的最优解实际上与放在地点X(1)，X(2)，…X(n-1)上的最优解是一样的。

-如果放，那么应该去掉Xn以及与距离它在y英里范围内的所有其他地点，并且找在剩下的点中的最优解。

当考察仅前j个地点X(1)，X(2)，…X(j)所定义的问题时，同样的推理也适用：在最优解中包含或不包含X(j)，具有同样的效果

我们令e[j]表示编号比j小，且距X(j)大于y英里的最东边的地点X(i)。等同于，找出编号比j小，且X(e[j])小于X(j)-y的第一个数。

实验代码：

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<cstdio>

#include<algorithm>

using namespace std;

#define N 100000

struct Po{

int pre;

int pos, val;

bool operator < (const Po &rhs)const{

return pos<rhs.pos;

}

}e[N];

int ans[N];

int find(Po \*array, int size, int key)

{

int first = 0, len = size - 1;

int half, mid;

while (len>1)

{

half = len / 2;

mid = first + half;

if (array[mid].pos >= key)

{

len = half;

}

else{

first = mid;

len = len - half;

}

}

return first;

}

int main()

{

int i, n;

int y,m;

printf("输入广告牌数量：");

scanf("%d", &n);

printf("输入两个广告牌最小距离：");

scanf("%d", &y);

printf("输入总的长度：");

scanf("%d", &m);

e[0].pos = 0;

printf("输入可能地点：");

for (i = 1; i <= n; i++)//第一个位置放一个空广告牌

scanf("%d", &e[i].pos);

printf("输入地点利益：");

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

scanf("%d", &e[i].val);

sort(e, e + n + 1);//对位置由小到大排序

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

e[i].pre = find(e, n+1, e[i].pos - y);

ans[1] = e[1].val;

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

ans[i] = max(e[i].val + ans[e[i].pre], ans[i - 1]);

printf("总的最大利益：%d\n", ans[n]);

system("pause");

return 0;

}

实验三：

实验分析：

从出发点开始，计算出某点到达其周边的点的最小花费，并记录在一个三维数组minCost[i][j][p]中，下标p表示这一点的油量，每一种油量的情况都有一种花费，这样，我们在计算下一点的同样的问题时，我们就可以利用之前已经计算的数据了，大大减小了重复计算。

根据分析，我们可以得到这样的结论，对于某点（i,j）,其最小花费为从 点（i-1,j），（i,j-1），（i,j+1）,(i+1,j)四点中出发的花费较小的那点。

实验代码：

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

void main()

{

int i, j, k, p, q, x, y;

int N, K, A, B, C;

int map[81][81];//记录各个点的信息

int cost[81][81][10];

int min;

int s[4][3];//4种行走方式

FILE \*fp;

//从输入文件中读数据

fp = fopen("input.txt", "rt");

fscanf(fp, "%d%d%d%d%d", &N, &K, &A, &B, &C);

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

for (j = 1; j <= N; j++)

fscanf(fp, "%d", &map[i][j]);

fclose(fp);

//初始化行走方式的数组

s[0][0] = -1;

s[0][1] = 0;

s[0][2] = 0;

s[1][0] = 0;

s[1][1] = -1;

s[1][2] = 0;

s[2][0] = 1;

s[2][1] = 0;

s[2][2] = B;

s[3][0] = 0;

s[3][1] = 1;

s[3][2] = B;

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

for (j = 1; j <= N; j++)

for (k = 0; k <= K + 1; k++) //注意这里！！！！

cost[i][j][k] = 10000;

//走到1,1点的费用为0

for (k = 0; k <= K; k++)

cost[1][1][k] = 0;

y = 1;

while (y != 0)//修改的行数，当y等于0时说明所有数据为最优数据，跳出循环

{

//printf("%d\n",y);

y = 0;

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

{

for (j = 1; j <= N; j++)

{

if (i != 1 || j != 1) //i,j同时为0时跳到下一次循环

{

for (p = 0; p <= K; p++)//剩余的油量

{

min = 10000;

for (q = 0; q<4; q++)//4种走法

{

if ((i == 1 && q == 0) || (j == 1 && q == 1) || (i == N&&q == 2) || (j == N&&q == 3))//边界情况

continue;

if (cost[i + s[q][0]][j + s[q][1]][p + 1] + s[q][2]<min)

min = cost[i + s[q][0]][j + s[q][1]][p + 1] + s[q][2];

}

if (min<10000)//说明有路可以走过去

{

if (cost[i][j][p]>min + A\*map[i][j]) //花费等于min+A\*map[i][j]是合理情况，不合理情况时y++ 用来做标记

y++;

cost[i][j][p] = min;

if (map[i][j] == 1)

{

cost[i][j][0] += A;

for (x = 1; x <= K; x++)

cost[i][j][x] = cost[i][j][0];

break;

}

}

else //没有路可以过去

{

cost[i][j][p] = cost[i][j][0] + C + A;

for (x = p + 1; x <= K; x++)

cost[i][j][x] = cost[i][j][p];

break;

}

}

}

}

}

}

fp = fopen("output.txt", "w");

fprintf(fp, "%d\n", cost[N][N][0]);

fclose(fp);

system("pause");

}