**北京邮电大学软件学院**

**2022-2023学年第一学期实验报告**

**课程名称：** 算法分析与设计

**项目名称：** 实验项目五——贪心-动态规划法

**项目完成人：**

**姓名：** 王宇涵 **学号：** 2020211730

**指导教师：** 李朝晖

**日 期： 2023年 5 月 18 日**

1. **实验目的**

1、 深刻理解并掌握动态规划法的设计思想；

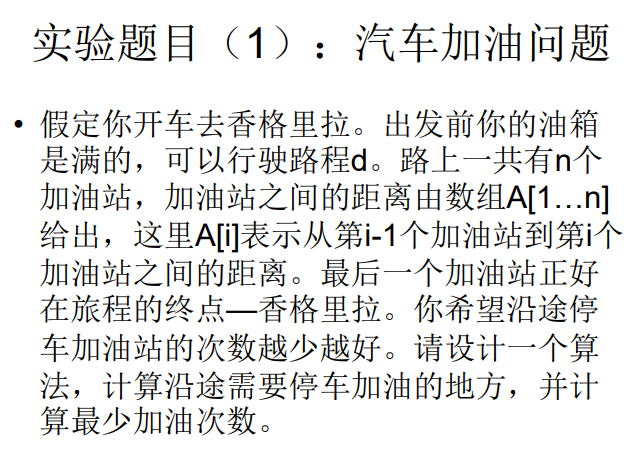
2、 提高应用动态规划法设计算法的技能；

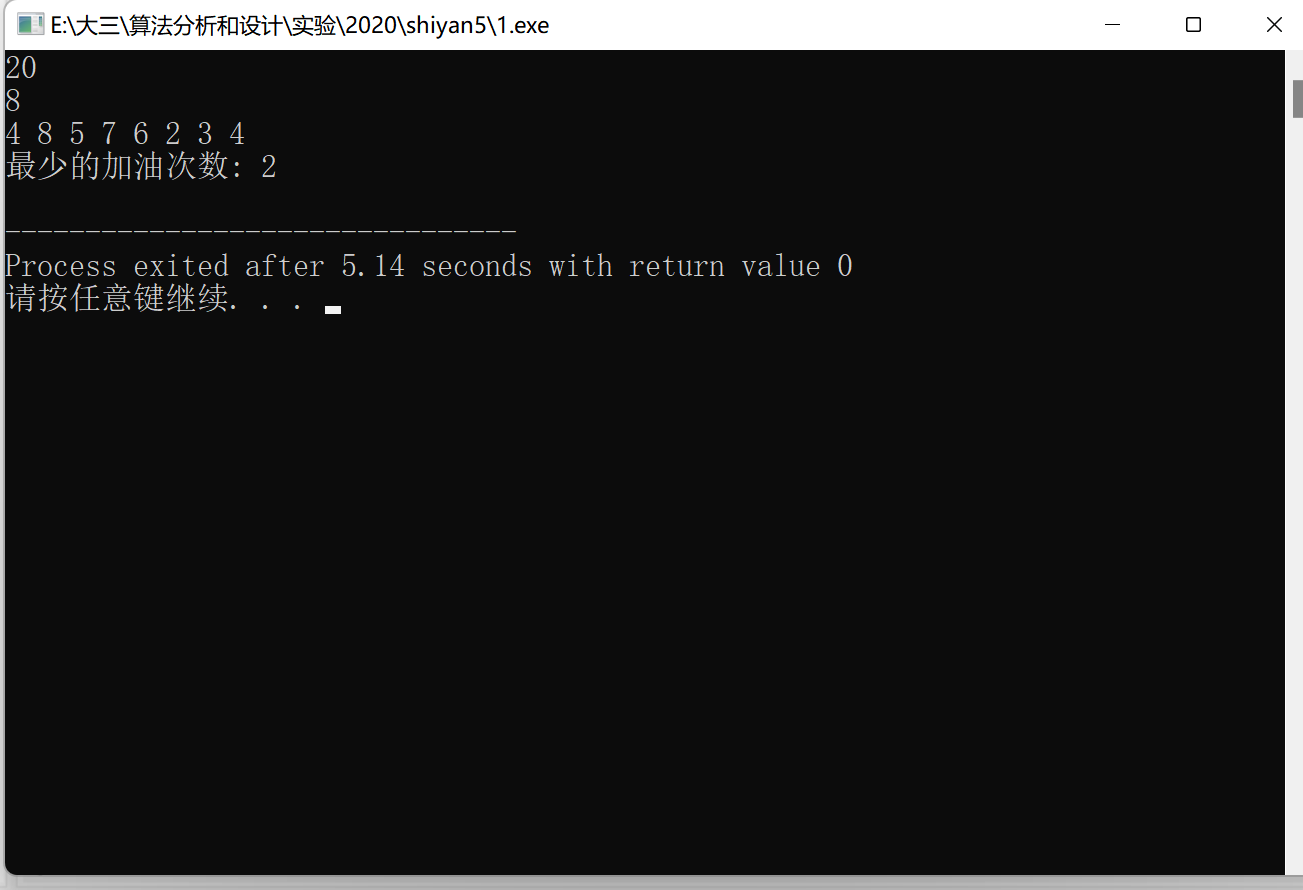
1. **实验内容**
2. (必做) 汽车加油问题 √
3. (选做) 汽车加油行驶问题 √
4. **实验环境**

Windows 10 Dev-C++

1. **实验结果**

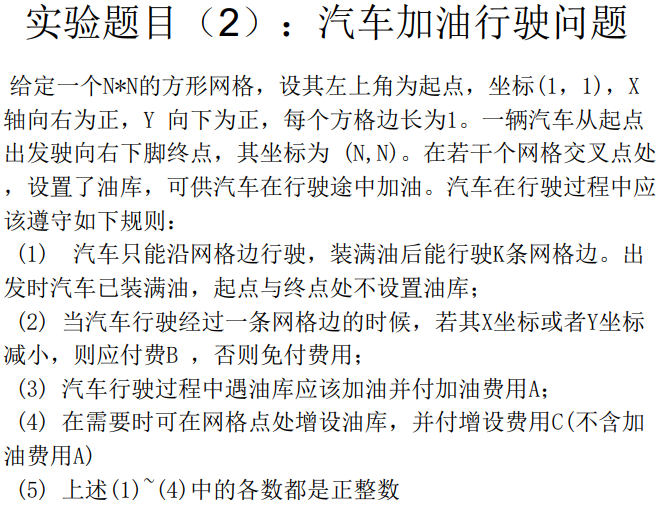
第一题 汽车加油问题

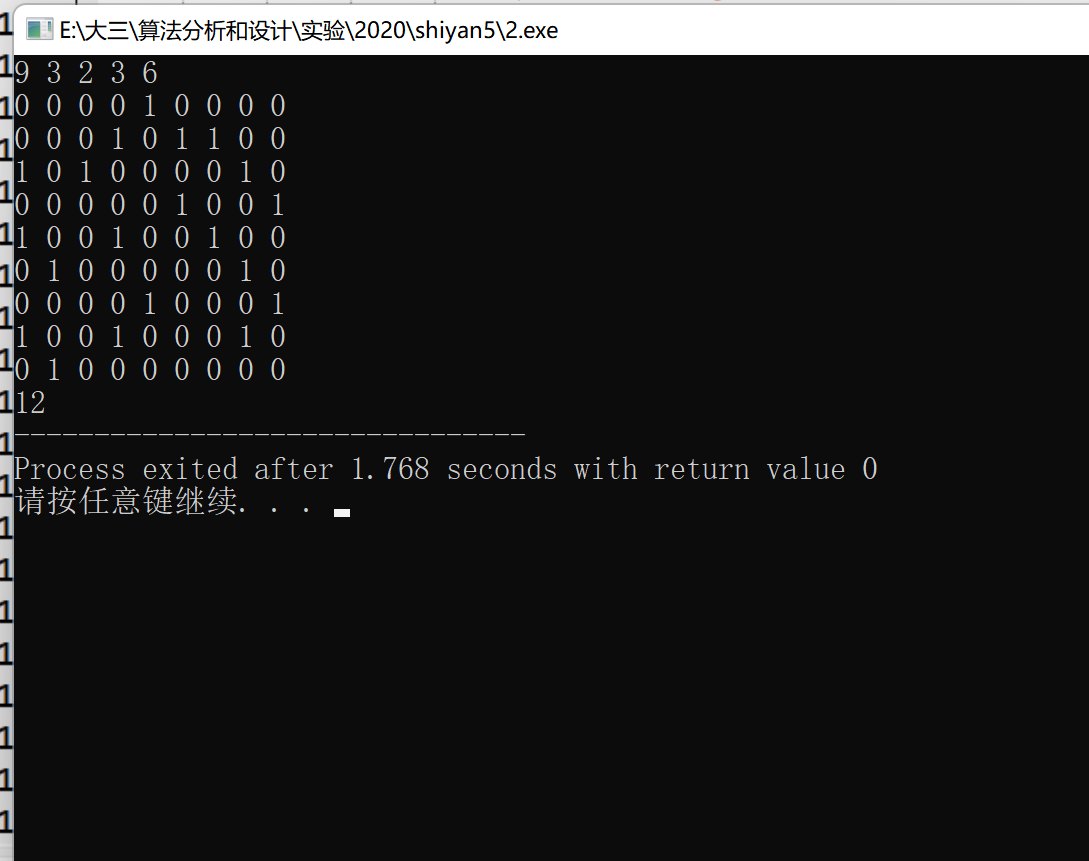






第二题 汽车加油行驶问题





1. **附录**

**项目1：汽车加油问题**

【整体思路】

* 假定你开车去香格里拉。出发前你的油箱是满的，可以行驶路程d。路上一共有n个加油站，加油站之间的距离由数组A[1…n]给出，这里A[i]表示从第i-1个加油站到第i个加油站之间的距离。最后一个加油站正好在旅程的终点—香格里拉。你希望沿途停车加油站的次数越少越好。请设计一个算法，计算沿途需要停车加油的地方，并计算最少加油次数。
* 本实验中采用贪心法解决该问题。

【问题分析】

* 贪心法策略：对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，算法得到的是在某种意义上的局部最优解。
* 找到汽车满油量时可以行驶的最大路程范围内的最后一个加油站。每到一个加油站就检查当前油量是否能够到达下一加油站：若能则继续行驶，若不能则加满油后上路。

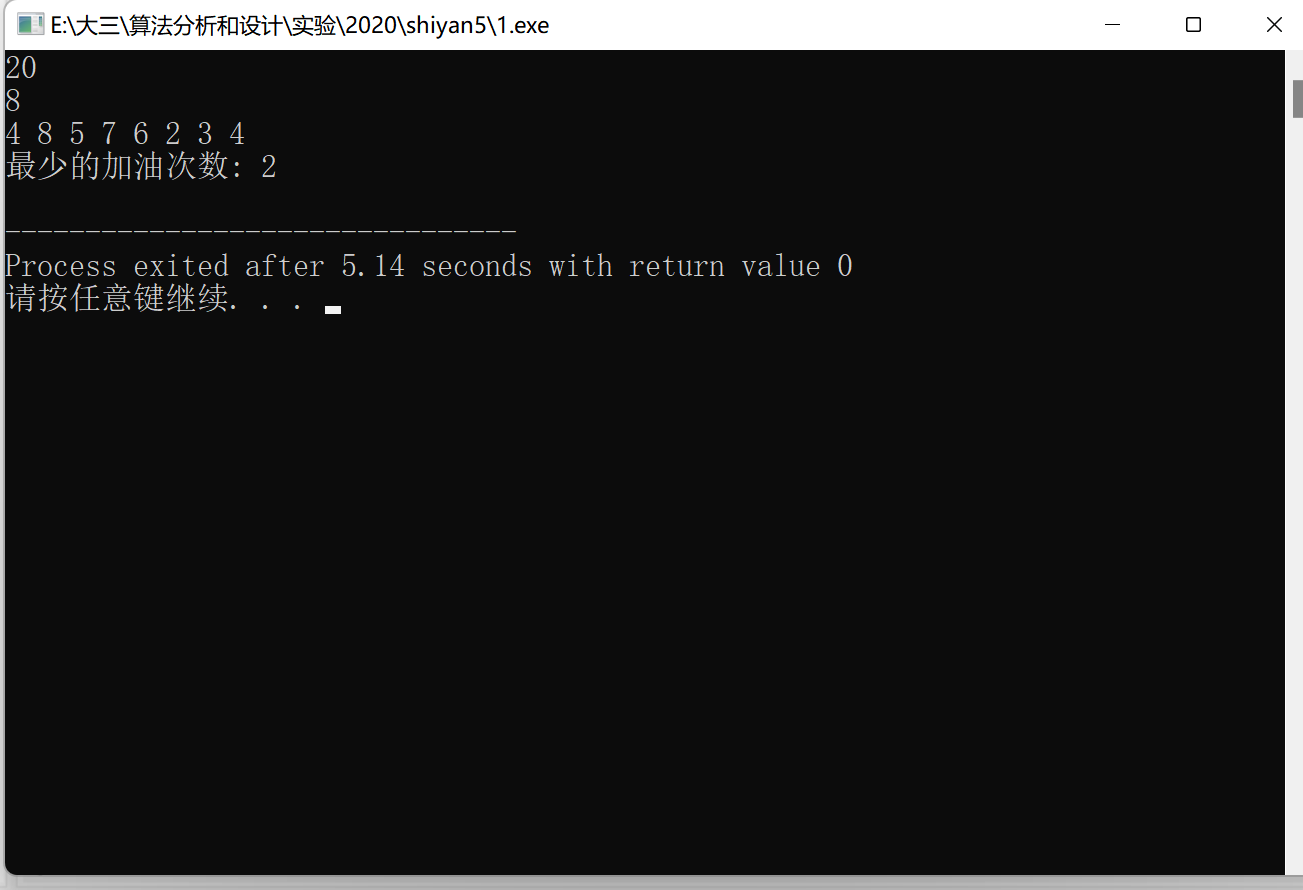
【算法设计】

汽车从起点出发,当前走过的距离用cur存储,判断到达下一个加油站的距离加上现在走过的距离cur之和与汽车加满油能走的总距离d相比,如果汽车能到达下一个加油站,则更新cur的值,继续判断考虑下一个加油站;如果到不了,则需要在当前的加油站加油,才能继续前进,更新答案.

【核心代码及解释】

int ans = 0;
  
 int curIndex = 1;
  
 bool hasArrived = false;
  
 while (true)
  
 {
  
 if (hasArrived == true)
  
 {
  
 break;
  
 }
  
 int cur = 0;
  
 for (; curIndex <= n; curIndex++)
  
 {
  
 cur += position[curIndex];
  
 if (d < cur)
  
 {
  
 ans++;
  
 break;
  
 }
  
 else if (d >= cur && curIndex == n)
  
 {
  
 hasArrived = true;
  
 }
  
 }
  
 }
  
 cout << "最少的加油次数: " << ans << endl;

【实验结果】





**项目2：汽车加油行驶问题**

【整体思路】

* 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

设计一个算法，求出汽车从起点出发到达终点所需要的最少费用。

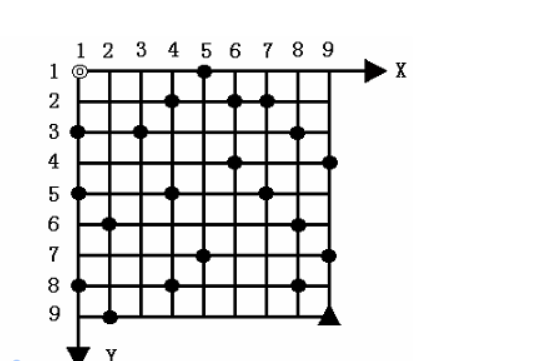
本题要求从左上角走到右下角，并且还会有强制消费，因此我们根据状态**递推**。为了保证最优解，我们是可以不理会它的强制消费转而向左或向上走，但是这样可能会有**时间不同步**的情况，不方便处理，所以我们试着把这个过程**分层**。类似广搜，实则**DP**，我们每次**只走一步**更新状态是最稳妥的。而对于每个状态取的都是**最小值**，所以引出了**DP**的思想。

【算法分析设计】

对于每个点来说，我们一旦出发,**时间轴**就会向前推进，为了保证状态正确，我们额外开一维使得状态只能来**自同时的相邻的四个点**，也就是**分层**的思想。那么一直推下去，理论最坏复杂度为N^5*N*5，路径长度可能达到N^2*N*2甚至更多，而根据考试策略，对于最大情况100×100控制在1s以内（最好是0.8s左右），来控制分层图的层数。于是我就定了分层为一个**固定的不会爆1s**的常数除以n的大小，这样既保证了小数据的正确性（此时n较小），又可以碰碰运气看较大数据是否能跑出正确解。

这个题还是有一些细节需要注意的，不仅是判断加油或绕路，而且还可以在**任何想建立**加油站的地方建立加油站，然后更新每个状态取最小值就可以了，只要**方程中**状态相同，从这里怎么转移出去都是一样的，就有点像**DP**的做法了。

实际上，由于最短路径长度为2n-1，加油费用与向上/左绕路在变动较大时可能是**加油更优**，那么我们基本可以推测到最短路径的长度在4n以内（虽然题目没给A,B,C的范围），考试时为了稳妥还是稍微卡一下时好了。转移方程也是比较清晰的，具体如下图所示。



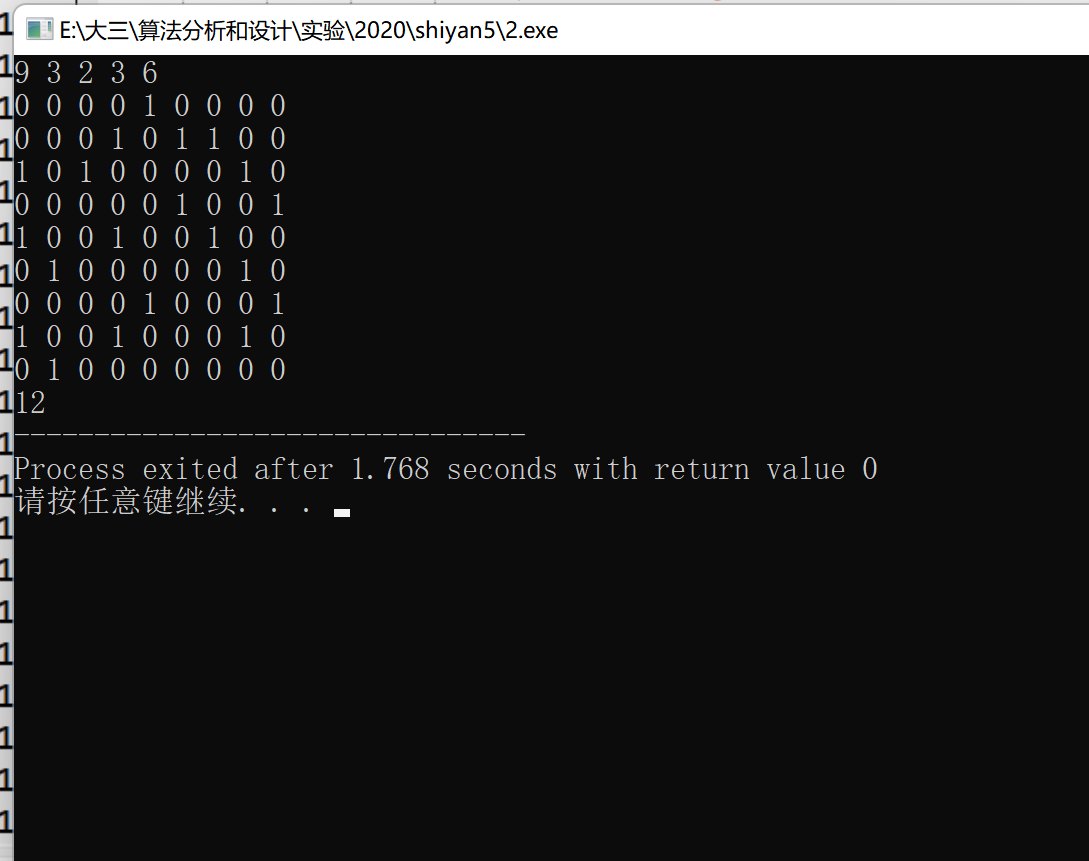
【核心代码及解释】：

使用一个三维的数组dp[x][y][fuel]来存储走到(x,y)且剩余fuel燃料时的最小花销.

之后根据广度优先搜索+条件判断来更新数组的值

// 初始化第一个点(1,1),到达这点不需要耗费能量
  
 for (int i = 0; i <= K; i++)
  
 {
  
 dp[1][1][i] = 0;
  
 }
  
 q.push(node{ 1,1,K }); //将初始点加入列表
  
 node top; //队列中第一个元素额
  
 node next;//考虑的下一个元素
  
 int cost; //花销
  
 while (!q.empty())
  
 {
  
 top = q.front();
  
 q.pop();
  
 // 往右走或者往下走
  
 for (int i = 0; i <= 1; i++)
  
 {
  
 next = node{ top.x + dir[i][0],top.y + dir[i][1],top.fuel - 1 };
  
 if (check(next.x, next.y))
  
 {
  
 // 更新一下当前的花销
  
 cost = dp[top.x][top.y][top.fuel];
  
 // 考虑一下next的情况
  
 if (map[next.x][next.y] == 1)
  
 { //加油了
  
 cost += A;
  
 next.fuel = K;
  
 }
  
 if ((!next.fuel) && ((next.x != n) || (next.y != n)))
  
 { //没油了,也没到终点
  
 cost += A + C;
  
 next.fuel = K;
  
 }
  
 if (cost >= dp[next.x][next.y][next.fuel])
  
 { //如果当前的花销比接下来考虑的下一个节点的花销大, 那它肯定不是最优的
  
 continue;
  
 }
  
 else
  
 { //接纳当前情况为最优点
  
 dp[next.x][next.y][next.fuel] = cost;
  
 for (int j = next.fuel - 1; j >= 0; j--)
  
 {
  
 if (dp[next.x][next.y][j] > cost)
  
 {
  
 dp[next.x][next.y][j] = cost;
  
 }
  
 else
  
 {
  
 break;
  
 }
  
 }
  
 q.push(next);
  
 }
  
 }
  
 }
  
 // 往左走或者往上走
  
 for (int i = 0; i <= 1; i++)
  
 { //基本同上
  
 next = node{ top.x - dir[i][0],top.y - dir[i][1],top.fuel - 1 }; // 往左走或者往上走
  
 if (check(next.x, next.y))
  
 {
  
 cost = dp[top.x][top.y][top.fuel] + B;
  
 if (map[next.x][next.y] == 1)
  
 {
  
 cost += A;
  
 next.fuel = K;
  
 }
  
 if ((!next.fuel) && ((next.x != n) || (next.y != n)))
  
 {
  
 cost += A + C;
  
 next.fuel = K;
  
 }
  
 if (cost >= dp[next.x][next.y][next.fuel])
  
 {
  
 continue;
  
 }
  
 // 接纳当前情况
  
 dp[next.x][next.y][next.fuel] = cost;
  
 for (int j = next.fuel - 1; j >= 0; --j)
  
 {
  
 if (dp[next.x][next.y][j] > cost)
  
 {
  
 dp[next.x][next.y][j] = cost;
  
 }
  
 else
  
 {
  
 break;
  
 }
  
 }
  
 q.push(next);
  
 }
  
 }
  
 }
  
 cout << dp[n][n][0];

【实验结果】



**源代码**

## **项目一**

#include<bits/stdc++.h>
  
#define debug(a) cout<<#a<<"="<<a<<endl
  
#define ll long long
  
  
/\*S:全程长度\*/
  
using namespace std;
  
//int x[] = { 10,20,35,40,50,65,75,85,100 }; /\*数组x:加油站位置(距离起点的位置)\*/
  
int main()
  
{
  
 int d=0, n=0;
  
 cin >> d;
  
 cin >> n;
  
 int position[n + 1];/\*数组position:加油点的位置\*/
  
 position[0] = 0;
  
 for (int i = 1; i <= n; i++)
  
 {
  
 cin >> position[i];
  
 }
  
 int ans = 0;
  
 int curIndex = 1;
  
 bool hasArrived = false;
  
 while (true)
  
 {
  
 if (hasArrived == true)
  
 {
  
 break;
  
 }
  
 int cur = 0;
  
 for (; curIndex <= n; curIndex++)
  
 {
  
 cur += position[curIndex];
  
 if (d < cur)
  
 {
  
 ans++;
  
 break;
  
 }
  
 else if (d >= cur && curIndex == n)
  
 {
  
 hasArrived = true;
  
 }
  
 }
  
 }
  
 cout << "最少的加油次数: " << ans << endl;
  
 return 0;
  
}
  
//10 20 35 40 50 65 75 85 100
  
  
//8
  
//4 8 5 7 6 2 3 4

## **项目二**

#include<iostream>
  
#include<queue>
  
  
using namespace std;
  
  
const int dir[2][2] = { {1,0},{0,1} };
  
int n, K, A, B, C, map[200][200];
  
int dp[200][200][50]; //到达点(x,y)且剩余燃料为p的情况下的最小开销
  
  
struct node
  
{
  
 int x;
  
 int y;
  
 int fuel;
  
};
  
queue<node> q;
  
  
int check(int x, int y)
  
{
  
 if (x<1 || x >n || y<1 || y>n)
  
 {
  
 return 0;
  
 }
  
 return 1;
  
}
  
  
int main()
  
{
  
 cin >> n >> K >> A >> B >> C;
  
 // 初始化dp数组
  
 for (int i = 0; i <= n; i++)
  
 {
  
 for (int j = 0; j <= n; j++)
  
 {
  
 for (int k = 0; k <= K; k++)
  
 {
  
 dp[i][j][k] = INT\_MAX;
  
 }
  
 }
  
 }
  
 // 录入地图
  
 for (int i = 1; i <= n; i++)
  
 {
  
 for (int j = 1; j <= n; j++)
  
 {
  
 cin >> map[i][j];
  
 }
  
 }
  
 // 初始化第一个点(1,1),到达这点不需要耗费能量
  
 for (int i = 0; i <= K; i++)
  
 {
  
 dp[1][1][i] = 0;
  
 }
  
 q.push(node{ 1,1,K }); //将初始点加入列表
  
 node top; //队列中第一个元素额
  
 node next;//考虑的下一个元素
  
 int cost; //花销
  
 while (!q.empty())
  
 {
  
 top = q.front();
  
 q.pop();
  
 // 往右走或者往下走
  
 for (int i = 0; i <= 1; i++)
  
 {
  
 next = node{ top.x + dir[i][0],top.y + dir[i][1],top.fuel - 1 };
  
 if (check(next.x, next.y))
  
 {
  
 // 更新一下当前的花销
  
 cost = dp[top.x][top.y][top.fuel];
  
 // 考虑一下next的情况
  
 if (map[next.x][next.y] == 1)
  
 { //加油了
  
 cost += A;
  
 next.fuel = K;
  
 }
  
 if ((!next.fuel) && ((next.x != n) || (next.y != n)))
  
 { //没油了,也没到终点
  
 cost += A + C;
  
 next.fuel = K;
  
 }
  
 if (cost >= dp[next.x][next.y][next.fuel])
  
 { //如果当前的花销比接下来考虑的下一个节点的花销大, 那它肯定不是最优的
  
 continue;
  
 }
  
 else
  
 { //接纳当前情况为最优点
  
 dp[next.x][next.y][next.fuel] = cost;
  
 for (int j = next.fuel - 1; j >= 0; j--)
  
 {
  
 if (dp[next.x][next.y][j] > cost)
  
 {
  
 dp[next.x][next.y][j] = cost;
  
 }
  
 else
  
 {
  
 break;
  
 }
  
 }
  
 q.push(next);
  
 }
  
 }
  
 }
  
 // 往左走或者往上走
  
 for (int i = 0; i <= 1; i++)
  
 { //基本同上
  
 next = node{ top.x - dir[i][0],top.y - dir[i][1],top.fuel - 1 }; // 往左走或者往上走
  
 if (check(next.x, next.y))
  
 {
  
 cost = dp[top.x][top.y][top.fuel] + B;
  
 if (map[next.x][next.y] == 1)
  
 {
  
 cost += A;
  
 next.fuel = K;
  
 }
  
 if ((!next.fuel) && ((next.x != n) || (next.y != n)))
  
 {
  
 cost += A + C;
  
 next.fuel = K;
  
 }
  
 if (cost >= dp[next.x][next.y][next.fuel])
  
 {
  
 continue;
  
 }
  
 // 接纳当前情况
  
 dp[next.x][next.y][next.fuel] = cost;
  
 for (int j = next.fuel - 1; j >= 0; --j)
  
 {
  
 if (dp[next.x][next.y][j] > cost)
  
 {
  
 dp[next.x][next.y][j] = cost;
  
 }
  
 else
  
 {
  
 break;
  
 }
  
 }
  
 q.push(next);
  
 }
  
 }
  
 }
  
 cout << dp[n][n][0];
  
 return 0;
  
}
  
//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

**实验心得**

通过本次实验，我掌握了汽车加油行驶问题的动态规划法求解，本问题的难

点在于设计目标函数和约束条件。此外也学习了贪心法，关键在于思路的设计。