DP初步

信智学院　陈琰宏

教学内容

01 动态规划原理

02

03 案例讲解

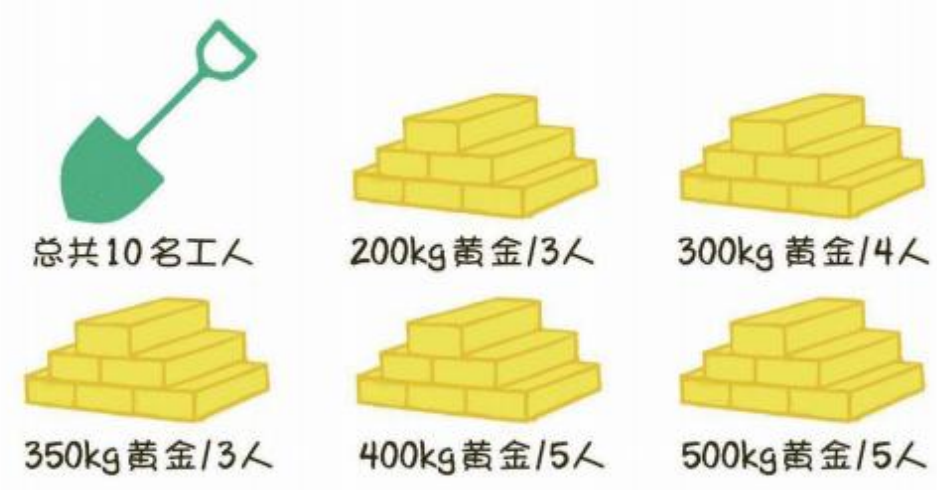
1 一个关于财富自由的问题

很久很久以前，有一位国王拥有5座金矿，每座金矿的黄金储量不同，需要参与挖掘的工人人数也不同。例如有的金矿储量是500kg黄金，需要5个工人来挖掘；有的金矿储量是200kg黄金，需要3个工人来挖掘……

如果参与挖矿的工人的总数是10。每座金矿要么全挖，要么不挖，不能派出一半人挖取一半的金矿。要求用程序求出，要想得到尽可能多的黄金，应该选择挖取哪几座金矿？

哇，要是我家也有5座金矿，我就财富自由了！

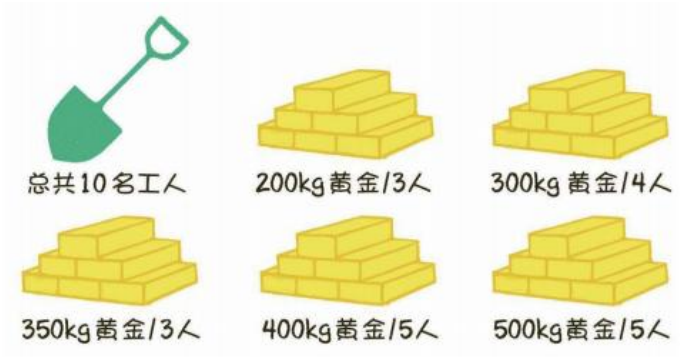
1 一个关于财富自由的问题



我想到了一个办法！我们可以按照金矿的性价比从高到低进行排序，优先选择性价比最高的金矿来挖掘，然后是性价比第2的……

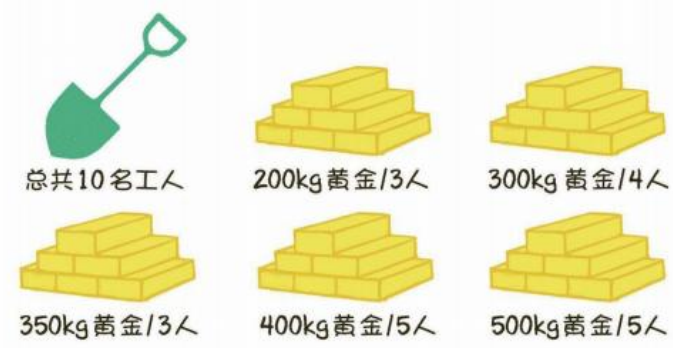
1 一个关于财富自由的问题

按照此思路，金矿按照性价比从高到低进行排序，排名结果如下。
第1名，350kg黄金/3人的金矿，人均产值约为116.6kg黄金。
第2名，500kg黄金/5人的金矿，人均产值为100kg黄金。
第3名，400kg黄金/5人的金矿，人均产值为80kg黄金。
第4名，300kg黄金/4人的金矿，人均产值为75kg黄金。
第5名，200kg黄金/3人的金矿，人均产值约为66.6kg黄金。
由于工人数量是10人，优先挖掘性价比排名为第1名和第2名的金矿之后，工人还剩下2人，不够再挖掘其他金矿了。
所以，**最后得出的最佳金矿收益是350+500即850kg黄金！**



1 一个关于财富自由的问题

如果放弃性价比最高的350kg黄金/3人的金矿，选择500kg黄金/5人和400kg黄金/5人的金矿，**加起来收益是900kg黄金**，是不是大于你得到的850kg黄金？



啊，还真是呢！

第1名，350kg黄金/3人的金矿，人均产值约为116.6kg黄金。
第2名，500kg黄金/5人的金矿，人均产值为100kg黄金。
第3名，400kg黄金/5人的金矿，人均产值为80kg黄金。

1.1 卡车运货问题

设卡车最大载重量是100，三种动物a、b、c的重量分别是40，50，70，其对应的总价值分别是80、100、150，如何运输才能使卡车运送的物品价值最大？

	a	b	c
重量	40	50	70
价值	80	100	150
Vi/Wi	2	2	2.14

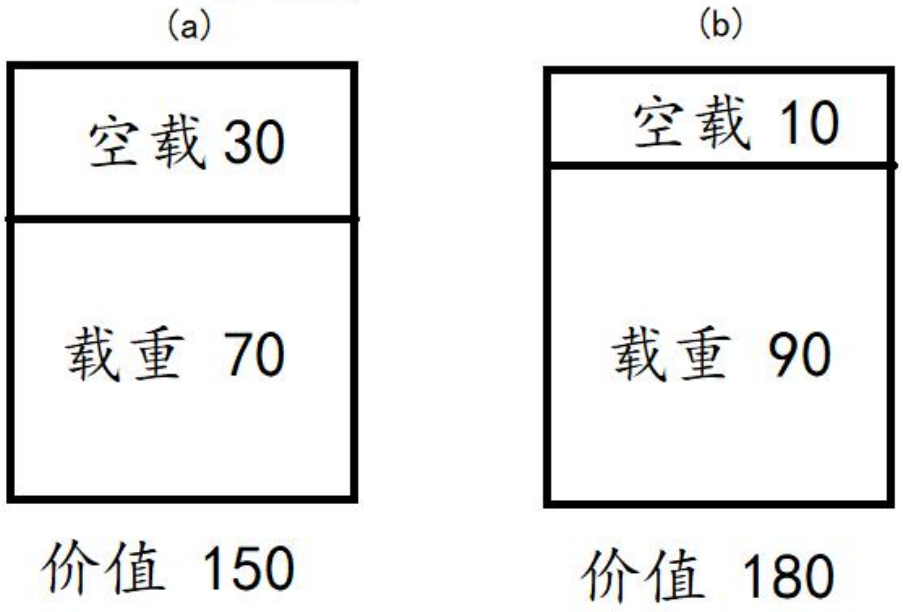
1.1 卡车运货问题

	a	b	c
重量	40	50	70
价值	80	100	150
Vi/Wi	2	2	2.14

情况a：首先选择动物c，得到价值150，然后任意选择a或b，**由于卡车最大载重为100，因此卡车不能装载其他动物。**

情况b：不按上述约束条件，直接选择a和b。可以得到价值80+100=180，卡车装载的重量为40+50=90。没有超过卡车的实际载重，因此也是一种可行解，显然，这种解比上一种解要优化。

问题出现在什么地方呢?

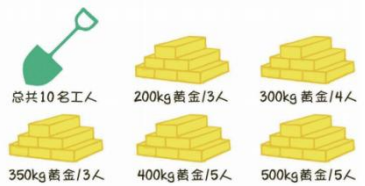


卡车装载货物情况分析

情况a，卡车的空载率比情况b高。也就是说，上面的分析，**只考虑了货物的价值质量比，而没有考虑到卡车的运营效率，**因此，**局部的最优化，不能导致全局的最优化**。

因此，贪心不能简单进行，而需要全面的考虑。

问题出现在什么地方呢?



(a)

剩余 2人

价值 850

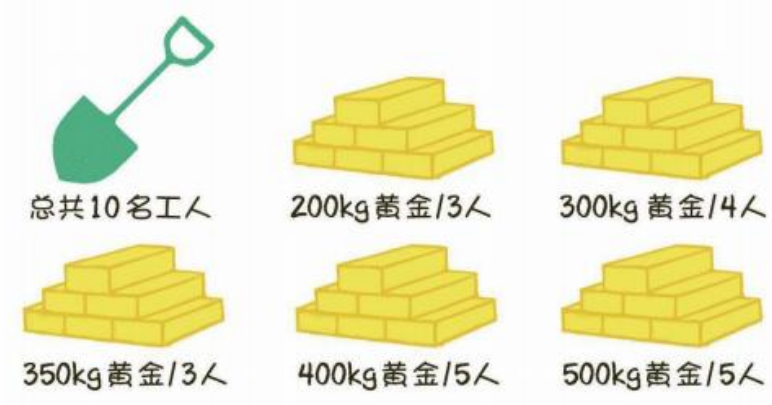
(b)

剩余 0人

价值 900

1.2 聪明国王的解决方法

国王首先来到了第5个金矿的所在地，他的臣子告诉他，如果要挖取第5个金矿的话就需要500个人，并且第5个金矿可以挖出50000公斤金子。听到这里国王哈哈大笑起来，因为原先他以为要知道5个金矿在仅有1000个人的情况下最多能挖出多少金子是一件很难思考的问题，但是，就在刚才听完他的臣子所说的那句话时，国王已经知道总共最多能挖出多少金子了，国王是如何在不了解其它金矿的情况下知道最多能挖出多少金子的呢？他的臣子们也不知道这个谜，因此他的臣子们就问他了：“最聪明的国王陛下，我们都没有告诉您其它金矿的情况，您是如何知道最终答案的呢？”



1.2 聪明国王的解决方法

得意的国王笑了笑，然后把他最得意的“左、右手”叫到跟前，说到：“我并不需要考虑最终要挖哪些金矿才能得到最多的金子，我只需要考虑我面前的这座金矿就可以了，对于我面前的这座金矿不外乎仅有两种选择，要么挖，要么不挖，对吧？”

“当然，当然”大臣们回答倒。

国王继续说道：“如果我挖取第5座金矿的话那么我现在就能获得50000公斤金子，而我将用去500个人，那么我还剩下500个人。我亲爱的左部下，如果你告诉我当我把所有剩下的500个人和所有剩下的其它金矿都交给你去开采你最多能给我挖出多少金子的话，那么我不就知道了在第5个金矿一定开采的情况下所能得到的最大金币数吗？”

1.2 聪明国王的解决方法

“是啊，是啊……如果第5座金矿一定开采的话……”大臣们点头说到。

国王笑着继续对着他的右部下说到：“亲爱的右部下，也许我并不打算开采这第5座金矿，那么我依然拥有1000个人，如果我把这1000个人和剩下的金矿都给你的话，你最多能给我挖出多少个金子呢？”

国王的右部下聪明地说道：“尊敬的国王陛下，我明白您的意思了，如果我回答最多能购开采出y个金子的话，那您就可以在y和x+50000之间选择一个较大者，而这个较大者就是最终我们能获得的最大金币数，您看我这样理解对吗？”

国王笑得更灿烂了，问他的左部下：“那么亲爱的左部下，我给你500个人和其余金矿的话你能告诉我最多能挖出多少金子吗？”

“请您放心，这个问题难不倒我”。左部下向国王打包票说到。

国王高兴地继续问他的右部下：“那右部下你呢，如果我给你1000个[illegible]和其余金矿的话你能告诉我最多能挖出多少金子吗？”

当然能了！交给我吧！”右部下同左部下一样自信地回答道。

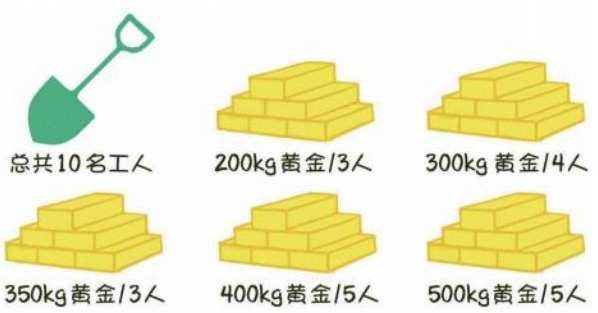
1.2 聪明国王的解决方法

国王走后，国王的左、右部下来到了第4座金矿，早已在那里等待他们的金矿勘测兵向两位大臣报道：“聪明的两位大臣，您们好，第4座金矿需要500个人才能开采，可以获得40000个金子”。

因为国王仅给他的左部下500个人，所以国王的左部下叫来了两个人，对着其中一个人问到：“如果我给你500个人和除了第5、第4的其它所有金矿的话，你能告诉我你最多能挖出多少金子吗？”

然后国王的左部下继续问另一个人：“如果我给你0个人（两个矿用完1000人）和除了第5、第4的其它所有金矿的话，你能告诉我你最多能挖出多少金子吗？”

国王的左部下在心里想着：“如果他们俩都能回答我的问题的话，那国王交给我的问题不就解决了吗？哈哈哈！”



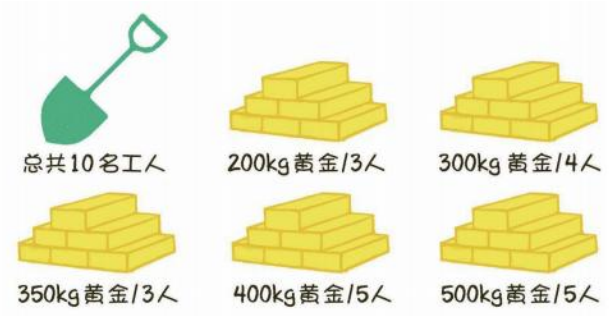
1.2 聪明国王的解决方法

因为国王给了他的右部下1000个人，所以国王的右部下同样也叫来了两个人，对着其中一个人问：“如果我给你1000个人和除了第5、第4的其它所有金矿的话，你能告诉我你最多能挖出多少金子吗？”

然后国王的右部下继续问他叫来的另一个人：“如果我给你500个人和除了第5、第4的其它所有金矿的话，你能告诉我你最多能挖出多少金子吗？”

此时，国王的右部下同左部下一样，他们都在为自己如此聪明而感到满足。

…… ……



1.2 聪明国王的解决方法

没用多少功夫，这个问题已经在全国传开了，更多人的人找到了更更多的人来解决这个问题，而有些人却不需要去另外找两个人帮他，哪些人不需要别人的帮助就可以回答他们的问题呢？

很明显，当被问到给你z个人和仅有第1座金矿时最多能挖出多少金子时，就不需要别人的帮助，因为你知道，如果z大于等于挖取第1座金矿所需要的人数的话，那么挖出来的最多金子数就是第1座金矿能够挖出来的金子数，如果这z个人不够开采第1座金矿，那么能挖出来的最多金子数就是1，因为这唯一的金矿不够人力去开采。

1.3 动态规划问题

故事讲到这里先暂停一下，我们重新分析这个故事属于什么问题？

典型的动态规划问题、著名的“背包问题”

所谓动态规划，就是把复杂的问题简化成规模较小的子问题，再从简单的子问题自底向上一步一步递推，最终得到复杂问题的最优解。

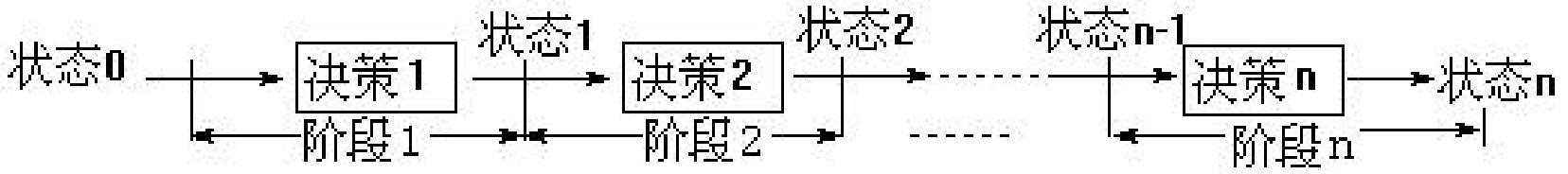
1.3.1 动态规划的基本思想

动态规划程序设计是对**解最优化问题**的一种途径、一种方法，而不是一种特殊算法，没有一个标准的数学表达式和明确清晰的解题方法。

动态规划程序设计往往是针对一种最优化问题，由于各种问题的性质不同，确定最优解的条件也互不相同，因而动态规划的设计方法对不同的问题，有各具特色的解题方法，而不存在一种万能的动态规划算法，可以解决各类最优化问题。

1.3.1 动态规划的基本思想

动态规划是把一个复杂问题分解成相对简单的“子问题”的方式求解，再组合出答案的方法。不过，分治法是将问题划分成一些“独立”的子问题，递归地求解各子问题，然后再合并子问题的解而得到原问题的解，如二分查找、快速排序等问题。与此不同，动态规划适用于子问题不是独立的情况，也就是子问题包含公共的“子子问题”。动态规划算法对每个子子问题只求解一次，并且立刻将其结果保存到一张“表”中，从而避免后面每次遇到子子问题时重复去计算。



1.3.2 动态规划的基本概念

1. 阶段和阶段变量

用动态规划求解一个问题时，需要将问题的全过程恰当地分成若干个相互联系的阶段，以便按一定的次序去求解。描述阶段的变量称为阶段变量，通常用K表示，阶段的划分一般是根据时间和空间的自然特征来划分，同时阶段的划分要便于把问题转化成多阶段决策过程，如例题1中，可将其划分成4个阶段，即K = 1，2，3，4。

2. 状态和状态变量

某一阶段的出发位置称为状态，通常一个阶段包含若干状态。一般地，状态可由变量来描述，用来描述状态的变量称为状态变量。如例题1中，C3是一个状态变量。

1.3.2 动态规划的基本概念

3. 决策、决策变量和决策允许集合

在对问题的处理中作出的**每种选择性的行动就是决策**。即从该阶段的每一个状态出发，通过一次选择性的行动转移至下一阶段的相应状态。在每一个阶段的每一个状态中都需要有一次决策，**决策也可以用变量来描述**，称这种变量为决策变量。在实际问题中，决策变量的取值往往限制在某一个范围之内，此范围称为允许决策集合。

4. 策略和最优策略

所有阶段依次排列构成问题的全过程。**全过程中各阶段决策变量所组成的有序总体称为策略**。在实际问题中，从决策允许集合中找出最优效果的策略成为最优策略。

1.3.2 动态规划的基本概念

5. **状态转移方程**

前一阶段的终点就是后一阶段的起点，对前一阶段的状态作出某种决策，产生后一阶段的状态，这种关系描述了由k阶段到k+1阶段状态的演变规律，称为状态转移方程。

1.3.3 动态规划的基本特点

子问题：

国王需要根据两个大臣的答案以及第5座金矿的信息才能判断出最多能够开采出多少金子。为了解决自己面临的问题，他需要给别人制造另外两个问题，这两个问题就是子问题。

思考动态规划的第一点----最优子结构：

国王相信，只要他的两个大臣能够回答出正确的答案，再加上他的聪明的判断就一定能得到最终的正确答案。我们把这种子问题最优时母问题通过优化选择后一定最优的情况叫做“最优子结构”。

1.3.2 动态规划的基本特点

思考动态规划的第二点----子问题重叠：

实际上国王也好，大臣也好，所有人面对的都是同样的问题，即给你一定数量的人，给你一定数量的金矿，让你求出能够开采出来的最多金子数。我们把这种母问题与子问题本质上是同一个问题的情况称为“子问题重叠”。

思考动态规划的第三点----边界：

想想如果不存在前面我们提到的那些底层劳动者的话这个问题能解决吗？永远都不可能！我们把这种子问题在一定时候就不再需要提出子子问题的情况叫做边界，没有边界就会出现死循环。

1.3.2 动态规划的基本特点

思考动态规划的第四点----子问题独立：

要知道，当国王的两个大臣在思考他们自己的问题时他们是不会关心对方是如何计算怎样开采金矿的，因为他们知道，国王只会选择两个人中的一个作为最后方案，另一个人的方案并不会得到实施，因此一个人的决定对另一个人的决定是没有影响的。我们把这种一个母问题在对子问题选择时，当前被选择的子问题两两互不影响的情况叫做“子问题独立”。

1.3.2 动态规划的基本特点

- 最优子结构
- 子问题重叠
- 边界
- 子问题独立

当你发现你正在思考的问题具备这四个性质的话，那么恭喜你，你基本上已经找到了动态规划的方法。

1.3.2 思考动态规划问题的核心

1. 怎么划分阶段
2. 如何描述状态
3. 目标状态是什么
4. 状态转移方程如何描述
5. 初始状态是什么

1 [7277] 国王的金矿

国王在他的国家发现了n座金矿，为了描述方便，我们给他们从1到n编号。对于第i个金矿，需要投入w(i）个的费用，能挖出来V(i)个单位的金子。

现在国王想开挖这些金矿，但是最多只有M个人力用于投入，问最多可以挖出来多少单位的金子。

输入第一行两个整数，分别为N和M；接下来N行每行两个整数，第i+1行为w(i)和v(i)。

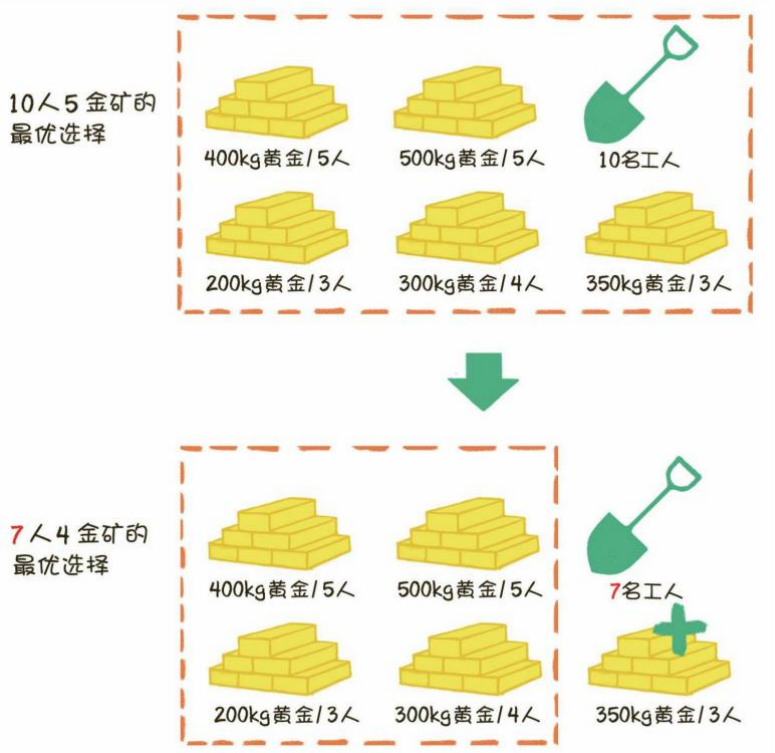
输出一行一个整数，为最多可以挖出来多少单位的金子。

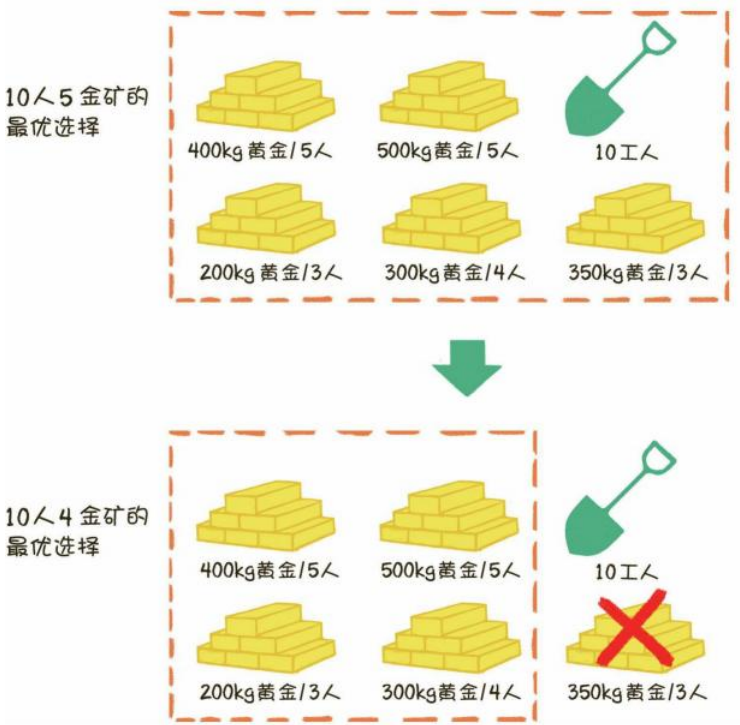
样例输入	样例输出
5 10	900
3 350	
5 500	
5 400	
4 300	
3 200	

分析

假设如果最后一个金矿注定不被挖掘，那么问题会转化成10个工人在前4个金矿中做出最优选择；

相应地，假设最后一个金矿一定会被挖掘，那么问题转化成7个工人在前4个金矿中做出最优选择（由于最后一个金矿消耗了3个工人）





分析

最后一个金矿到底该不该挖呢？ 那就要看10个工人在前4个金矿的收益，和7个工人在前4个金矿的收益+最后一个金矿的收益谁大谁小了。

设f[i][j]表示前i座金矿被j个人挖掘后，可以获得的最大价值，那么我们可以很容易分析得出 f[i][j]的计算方法：

f[5][10]=max(f[4][10],f[4][7]+350);

设w[i]表示挖掘第i座金矿需要的人数，挖掘后能获得的价值是c[i]，则

f[i][j]=max{f[i-1][j],f[i-1][j-w[i]]+c[i]}。

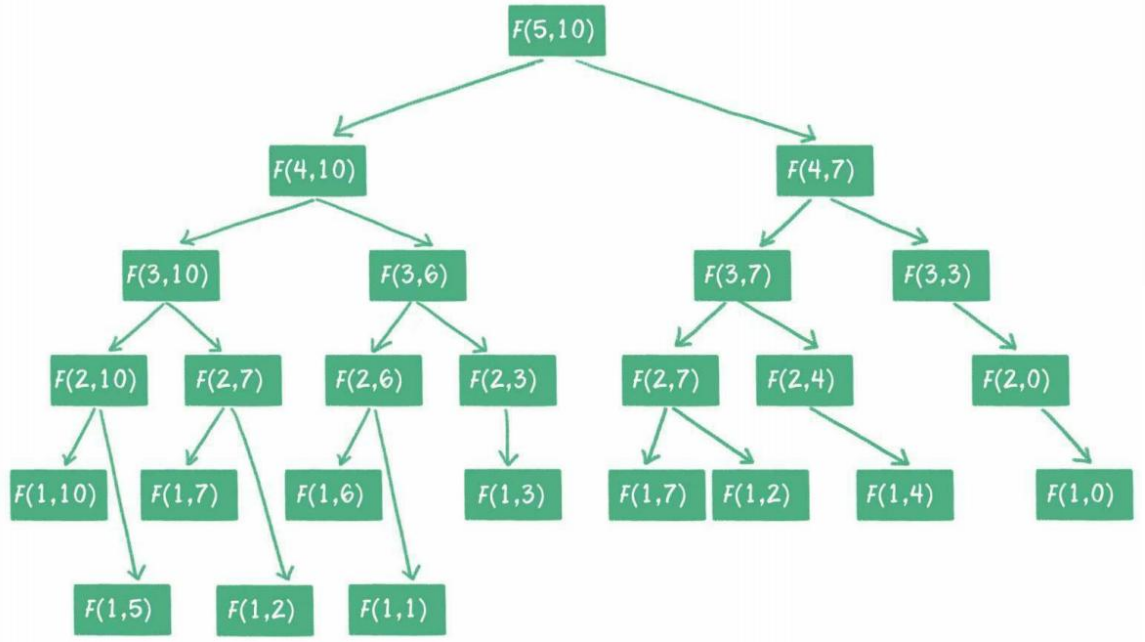
f[j]=max{f[j],f[j-w[i]]+c[i]}。

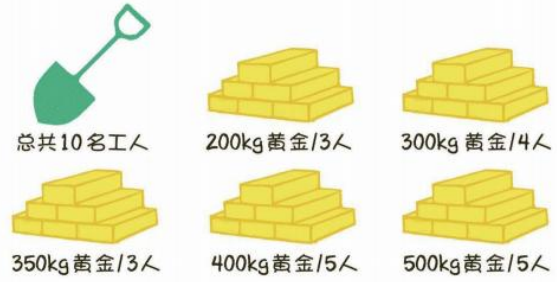
分析

同样的道理，对于前4个金矿的选择，我们还可以做进一步简化。

…… ……　　…… ……

就这样，问题一分为二，二分为四，一直把问题简化成在0个金矿或0个 工人时的最优选择，这个收益结果显然是0，也就是问题的边界。





思考

1. 怎么划分阶段

   对于每一件物品都有选和不选两种选择。将物品的选取当做阶段

2. 如何描述状态

   dp[i][j] 表示前i座金矿被j个人挖掘后，可以获得的最大价值

3. 目标状态是什么

   dp[m][n]

4. 状态转移方程如何描述

   dp[i][j]=max{dp[i-1][j],dp[i-1][j-w[i]]+c[i]}。

5. 初始状态是什么

   dp[i][j]=0

代码

```
#include<bits/stdc++.h>
using namespace std;
int f[205][3005],w[205],c[205];
int main(){
    int n,v;
    scanf("%d %d",&n,&v);
    for(int i=1;i<=n;i++){
        scanf("%d %d",&w[i],&c[i]);
    }
    memset(f,0,sizeof(f));
    for(int i=1;i<=n;i++){
        for(int j=1;j<=v;j++){
            if(w[i]<=j)
                f[i][j]=max(f[i-1][j],f[i-1][j-w[i]]+c[i]);
            else
                f[i][j]=f[i-1][j];
        }
    }
    printf("%d",f[n][v]);
    return 0;
}
```

1.3.3 动态规划分类

- 数字三角形
- 记忆化搜索
- 最长上升子序列
- 最长公共子序列
- 背包问题
- 区间DP
- 树形DP
- 状态压缩DP

[2757]移动路线

X桌子上有一个m行n列的方格矩阵，将每个方格用坐标表示，行坐标从下到上依次递增，列坐标从左至右依次递增，左下角方格的坐标为(1,1)，则右上角方格的坐标为(m,n)。

小明是个调皮的孩子，一天他捉来一只蚂蚁，不小心把蚂蚁的右脚弄伤了，于是蚂蚁只能向上或向右移动。小明把这只蚂蚁放在左下角的方格中，蚂蚁从

左下角的方格中移动到右上角的方格中，每步移动一个方格。蚂蚁始终在方格矩阵内移动，请计算出不同的移动路线的数目。

对于1行1列的方格矩阵，蚂蚁原地移动，移动路线数为1；对于1行2列（或2行1列）的方格矩阵，蚂蚁只需一次向右（或向上）移动，移动路线数也为1.....对于一个2行3列的方格矩阵，如下图所示：

(2,1)　(2,2)　(2,3)

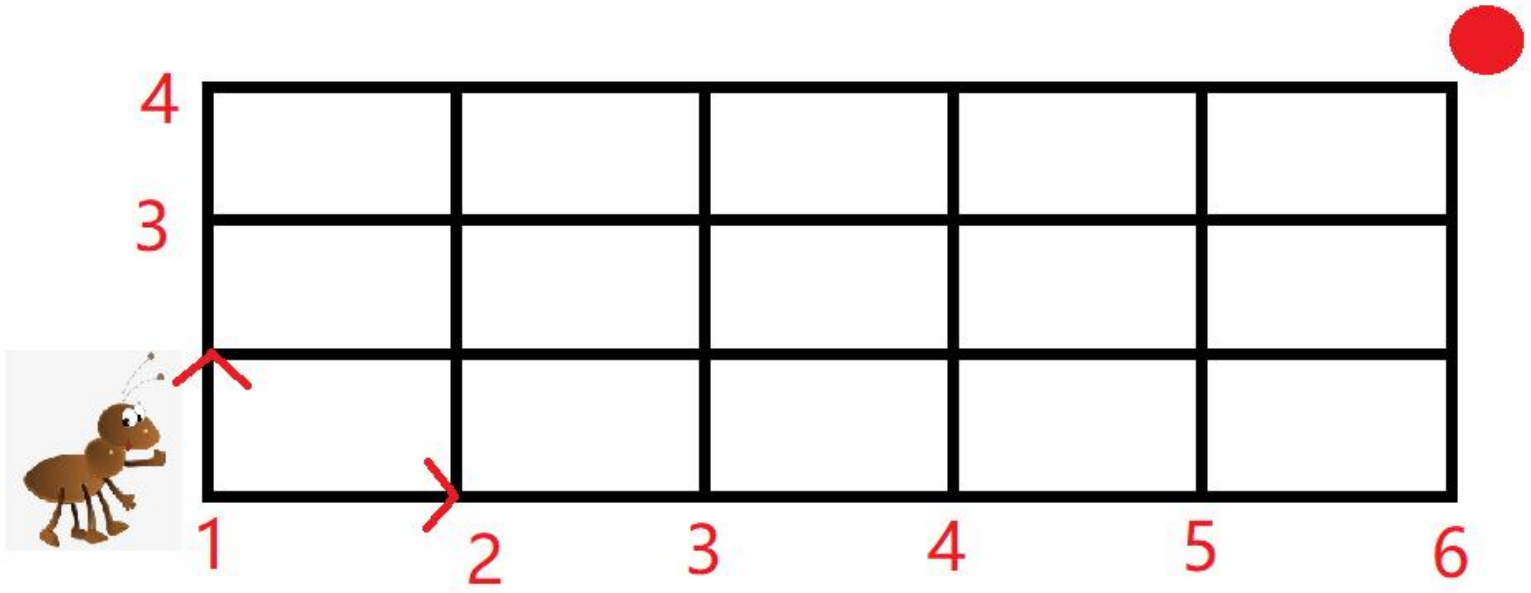
(1,1)　(1,2)　(1,3)

1 [2757]移动路线

蚂蚁共有3种移动路线：
路线1：(1,1) → (1,2) → (1,3) → (2,3)
路线2：(1,1) → (1,2) → (2,2) → (2,3)
路线3：(1,1) → (2,1) → (2,2) → (2,3)
输入
输入只有一行，包括两个整数m和n（$0 < m+n \leq 20$），代表方格矩阵的行数和列数，m、n之间用空格隔开。
输出
输出只有一行，为不同的移动路线的数目。
样例输入
2 3
样例输出
3

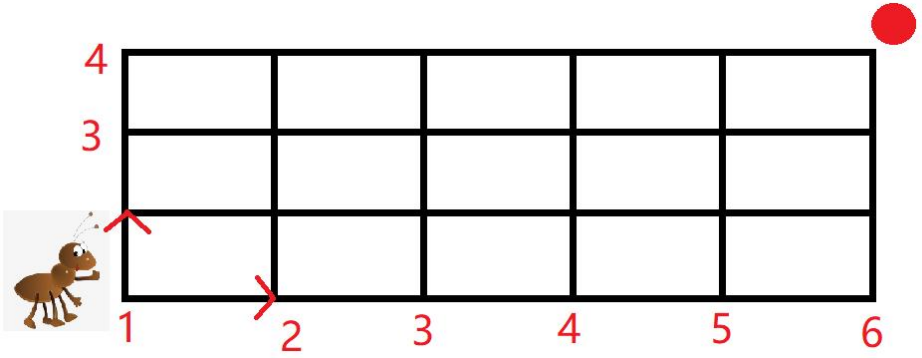
思考

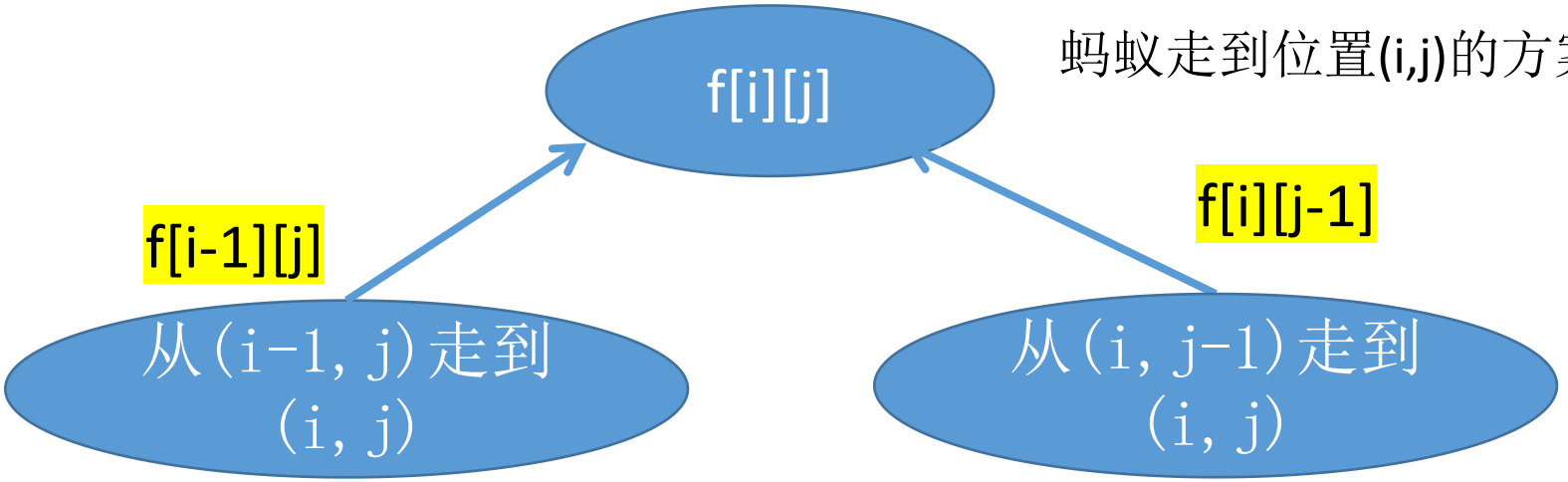
1.目标状态如何描述？
2. 递推式是什么？
3. 初始条件时什么？



思考

1.状态如何描述？
2. 状态如何计算？
3. 目标状态是什么？
4. 初始条件时什么？





$$f[i][j]=f[i-1][j]+f[i][j-1]$$

思考

1.状态如何描述？

f[i][j]表示蚂蚁走到位置(i,j)的方案数

2. 状态如何计算？

f[i][j]=f[i-1][j]+f[i][j-1]

3. 目标状态是什么？

f[m][n]

4. 初始条件时什么？

f[i][1]=1

f[1][i]=1

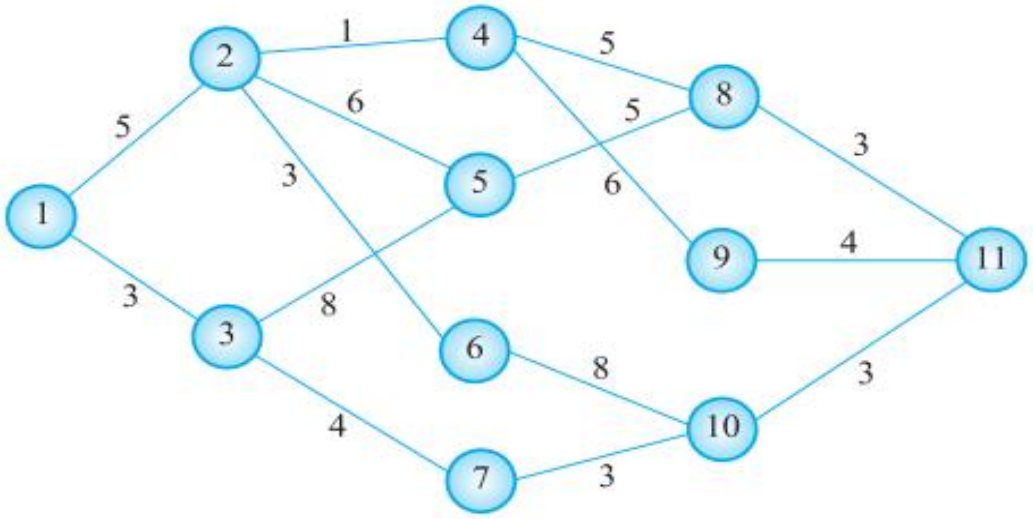
代码

```
1  #include<cstdio>
2  #include<cstring>
3  #include<iostream>
4  using namespace std;
5  int a[30][30];
6  int main(){
7      int m,n,i,j;
8      scanf("%d%d",&m,&n);
9      for(i=1;i<=m;i++)a[i][1]=1;
10     for(j=1;j<=n;j++)a[1][j]=1;
11     for(i=2;i<=m;i++)
12         for(j=2;j<=n;j++)
13             a[i][j]=a[i-1][j]+a[i][j-1];
14     printf("%d",a[m][n]);
15     return 0;
16 }
```

2 [4003]城市交通网络图

有 n 个城市，编号 1~n，有些城市之间有路相连，有些则没有，有路则会有一个距离。图 所示为一个含有 11 个城市的交通图，连线上的数（权）表示距离。

现在规定只能从编号小的城市到编号大的城市。问：从编号为 1 的城市到编号为n 的城市之间的最短距离是多少？



输入 第 1 行为 n，表示城市数，n≤100。下面的 n 行是一个 n×n 的邻接矩阵map[i,j]，其中 map[i,j]=0 表示城市 i和城市 j 之间没有路相连，否则为两者之间的距离。

输出一行一个数，表示最短距离。数据保证一定可以从城市 1 到城市 n。

输入样例

```
11
0 5 3 0 0 0 0 0 0 0 0
5 0 0 1 6 3 0 0 0 0 0
3 0 0 0 8 0 4 0 0 0 0
0 1 0 0 0 0 0 5 6 0 0
0 6 8 0 0 0 0 5 0 0 0
0 3 0 0 0 0 0 0 0 8 0
0 0 4 0 0 0 0 0 0 3 0
0 0 0 5 5 0 0 0 0 0 3
0 0 0 6 0 0 0 0 0 0 4
0 0 0 0 0 8 3 0 0 0 3
0 0 0 0 0 0 0 3 4 3 0
```

输出样例

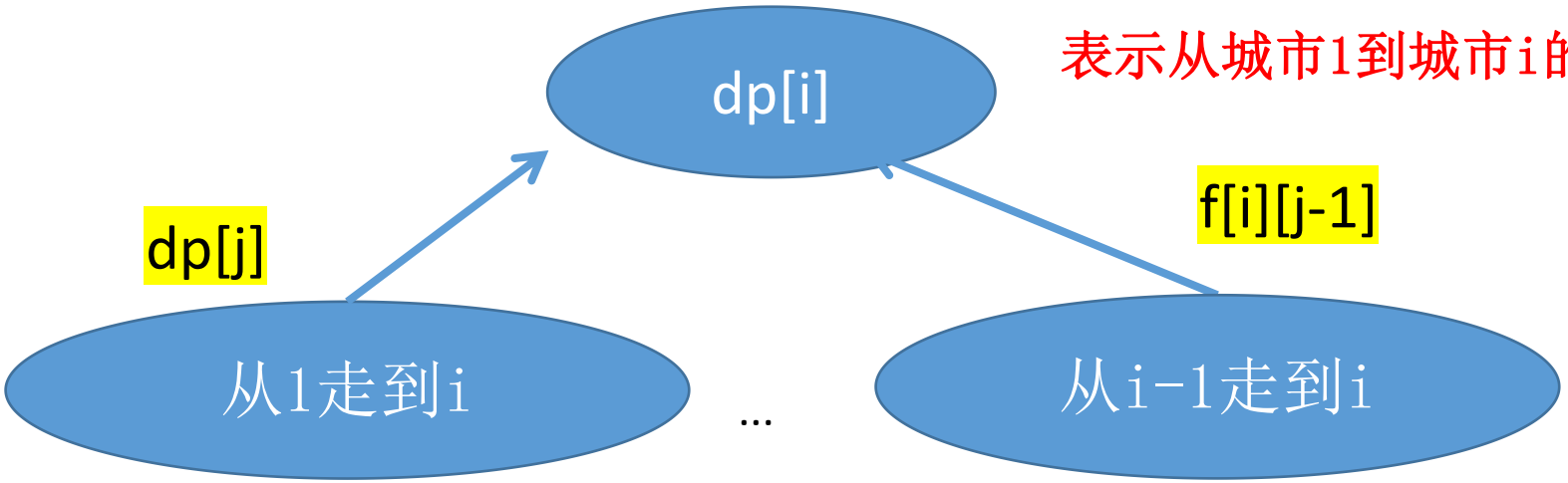
```
13
```

分析

逆向思考，现在想要从城市1到达城市11，则只能从城市8、9或10中转过去，如果知道了从城市1到城市8、9和10的最短距离，那么只要把这3个最短距离分别加上这3个城市与城市11之间的距离，再从中取一个最小值即可。这样一来，问题就变成了求城市1到城市8、9、10的最短距离，而这3个子问题与原问题是完全一致的，只是问题的规模缩小了一点。如何求城市1到城市8的最短距离呢？想法与刚才一样，如果知道了城市1到城市4和5的最短距离，那么到城市8的最短距离就是到城市4的最短距离加上5以及到城市5的最短距离加上5当中较小的那个值。而如何求城市4和5的最短距离呢？…如此下去，直到求城市1到城市2和3的最短距离，而这个值是已知的（因为城市1和2、3之间有直接的边相连）。这种解题思想就是“动态规划”。

状态转移

规定只能从编号小的城市到编号大的城市



dp[i]=min(dp[j]+a[j][i])

思考

1. 怎么划分阶段

   非常明显，每走一个城市就一个阶段

2. 如何描述状态

   dp[i] 表示从城市1到城市j的最短距离

3. 目标状态是什么

   dp[n]

4. 状态转移方程如何描述

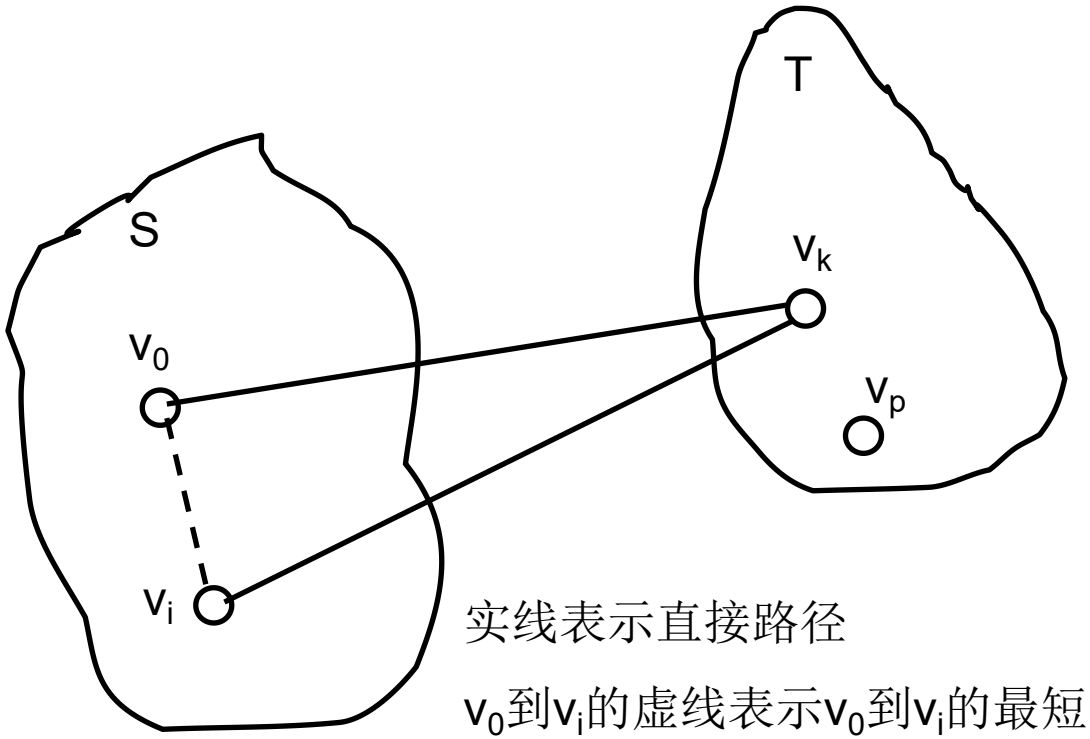
   dp[i]=min{dp[i],dp[j]+a[j][i])

5. 初始状态是什么

   dp[1]=0

分析

可以证明v_0到T中顶点v_k的最短路径，要么是从v_0到v_k的直接路径；要么是从v_0经S中某个顶点 v_i 再到v_k的路径。



实线表示直接路径

v_0到v_i的虚线表示v_0到v_i的最短路径

分析

假设dis[i]表示从**城市1到城市i的最短距离**，求解过程可以用下列式子归纳：

dis[1]=0

dis[i]=min{dis[i],dis[j]+map[j,i]}

（i>1，j从1到i-1且map[j,i]不等于0）

最后，只要输出dis[n]。

代码

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N = 110, INF = 1 << 30;//INF表示2^30
int a[N][N], dp[N], n;
int main(){
    scanf("%d", &n);
     for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++) scanf("%d", &a[i][j]);
    for(int i = 2; i <= n; i++) dp[i] = INF;
        //打擂台取最小值，所以初值赋一个很大的数INF
    for(int i = 2; i <= n; i++)
        for(int j = 1; j < i; j++)
            if(a[i][j]) dp[i] = min(dp[i], dp[j] + a[i][j]);
    printf("%d\n", dp[n]);
        return 0;
}
```

[2850] 糖果

由于在维护世界和平的事务中做出巨大贡献，Dzx被赠予糖果公司2010年5月23日当天无限量糖果免费优惠券。在这一天，Dzx可以从糖果公司的N件产品中任意选择若干件带回家享用。糖果公司的N件产品每件都包含数量不同的糖果。Dzx希望他选择的产品包含的糖果总数是K的整数倍，这样他才能平均地将糖果分给帮助他维护世界和平的伙伴们。当然，在满足这一条件的基础上，糖果总数越多越好。Dzx最多能带走多少糖果呢？

注意：Dzx只能将糖果公司的产品整件带走。

输入第一行包含两个整数N(1≤N≤10)和K(1≤K≤10)。以下N行每行1个整数，表示糖果公司该件产品中包含的糖果数目，糖果总数不超过100000。

输出符合要求的最多能达到的糖果总数，如果不能达到K的倍数这一要求，输出0。

输入样例	输出样例
5 7 1 2 3 4 5	14

数据范围与提示

Dzx的选择是2+3+4+5=14，这样糖果总数是7的倍数，并且是总数最多的选择。

思考动态规划问题的核心

1. 怎么划分阶段
2. 如何描述状态
3. 目标状态是什么
4. 状态转移方程如何描述
5. 初始状态是什么

[7519]瓶子涂色 DP

小猪上小学的时候，一度对颜色非常感兴趣，虽然他的美术非常糟糕。

有一次他喝完n瓶饮料把透明的瓶子排成一排，想把这些饮料瓶子都涂上颜色。他觉得如果所有相邻的两个瓶子颜色都不一样的话会比较有趣。

他现在只有红色(Red)、绿色(Green)和蓝色(Blue)这三种颜料。由于瓶子的大小和表面材质不同，在不同的瓶子上涂不同的颜色需要的花费都不一样。小猪统计了一下，把第i个瓶子染成红色需要Ri元钱，染成绿色需要Gi元钱，染成蓝色需要Bi元钱。 现在请你帮他计算出要使相邻两个瓶子的颜色都不一样，他至少需要多少花费。

[7519]瓶子涂色 DP

输入第一行只有一个整数n，表示共有n只瓶子。 第二行有n个正整数（以一个空格分隔）,第i个数Ri表示把第i个瓶子染成红色需要Ri元钱。 第三行有n个正整数（以一个空格分隔）,第i个数Gi表示把第i个瓶子染成绿色需要Gi元钱。第四行有n个正整数（以一个空格分隔）,第i个数Bi表示把第i个瓶子染成蓝色需要Bi元钱。1≤n≤100000，1≤Ri，Gi，Bi≤100。

输出仅有一行，该行只有一个整数，表示最小花费。

样例输入

```
5
1 3 1 2 2
1 2 3 4 3
4 2 1 5 3
```

样例输出

```
9
```

分析

非常典型的动态规划问题。无论第i个瓶子涂什么颜色（但一定不能与第i-1个瓶子的颜色相同）。

f[i][j]表示第i个瓶子涂第j种颜色所花费的最小代价,从而得到状态转移方程：

f[i][j]=min{f[i][j],f[i-1][k]+g[i][j]}

f[i-1,k]表示第i-1个瓶子涂第k种颜色所花费的最小代价,保证jk最后输出的结果就是f[n,1],f[n,2],f[n,3]中的最小值算法时间复杂度O(9*n)

思考

1. 怎么划分阶段

   瓶子以一个个的涂，把涂一个瓶子当做一个阶段

2. 如何描述状态

   dp[i][j] 表示第i个瓶子涂第j种颜色所花费的最小代价

3. 目标状态是什么

   dp[n][0] dp[n][1] dp[n][2]

4. 状态转移方程如何描述

   dp[i][j]=min{dp[i][j],dp[i-1][k]+a[i][j]}

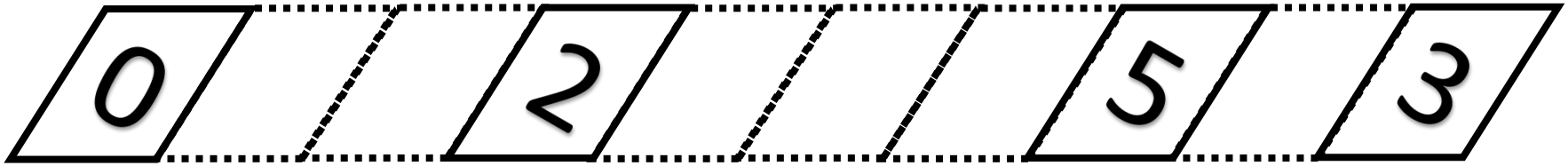
5. 初始状态是什么

   dp[1]=0

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,a[100001][3],dp[100001][3];
int main(){
    memset(dp,0x3f3f3f,sizeof(dp));
    cin>>n;
    for(int i=0;i<3;i++)
        for(int j=1;j<=n;j++)
            cin>>a[j][i];
    dp[0][1]=0;dp[0][2]=0;dp[0][0]=0;
    for(int i=1;i<=n;i++){
            dp[i][0]=min(dp[i-1][1]+a[i][0],dp[i-1][2]+a[i][0]);
            dp[i][1]=min(dp[i-1][0]+a[i][1],dp[i-1][2]+a[i][1]);
            dp[i][2]=min(dp[i-1][0]+a[i][2],dp[i-1][1]+a[i][2]);
        }
    cout<<min(dp[n][0],min(dp[n][1],dp[n][2]));
    return 0;
}
```

[3276] 跳房子

- 题目描述（part1）
  - 跳房子，也叫跳飞机，是一种世界性的儿童游戏，也是中国民间传统的体育游戏之一。
  - 跳房子的游戏规则如下：
  - 在地面上确定一个起点，然后在起点右侧画 n 个格子，这些格子都在同一条直线上。每个格子内有一个数字（整数），表示到达这个 格子能得到的分数。玩家第一次从起点开始向右跳，跳到起点右侧的一个格子内。第二次再从当前位置继续向右跳，依此类推。规则规定：
  - 玩家每次都必须跳到当前位置右侧的一个格子内。**玩家可以在任意时刻结束游戏，获得的分数为曾经到达过的格子中的数字之和。**



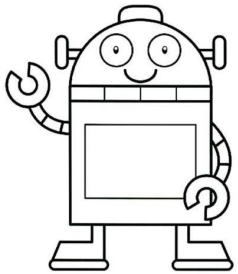
1 2 3 4 5 6 7 8 9

题目描述（part2）

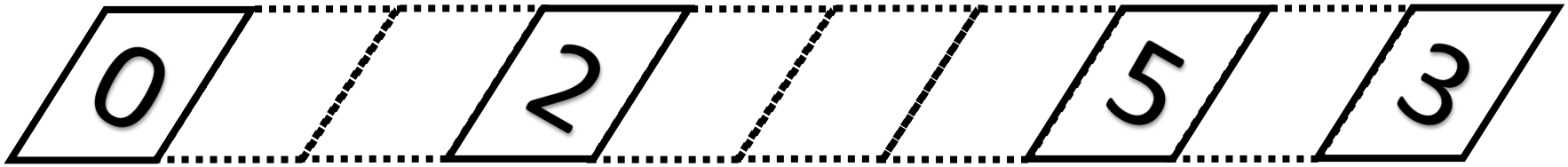
- 现在小R研发了一款弹跳机器人来参加这个游戏。但是这个机器人有一个非常严重的缺陷，它**每次向右弹跳的距离只能为固定的 d** 。小R希望改进他的机器人，如果他花 g 个金币改进他的机器人，那么他的机器人灵活性就能增加 g ，但是需要注意的是，每 次弹跳的距离至少为1。具体而言（当 g<d 时）他的机器人每次可以选择向右弹跳的距离为 d−g, d−g+1, d−g+2, …, d+g−2, d+g−1, d+g；否则（当 g≥d 时），他的机器人每次可以选择向右弹跳的距离为 1, 2, 3, …， d+g−2, d+g−1, d+g。

- 现在小R希望获得**至少 k 分**，请问他**至少要花多少金币**来改造他的机器人。

d= { 2,3,4,5,6 }



最终得分：10



1 2 3 4 5 6 7 8 9

输入输出

- **输入格式**
  - 第一行三个正整数 n ， d ， k ，分别表示格子的数目，改进前机器人弹跳的固定距离，以及希望至少获得的分数。相邻两个数 之间用一个空格隔开。
  - 接下来 n 行，每行两个整数 x_i ， s_i，分别表示起点到第 i 个格子的距离以及第 i 个格子的分数。两个数之间用一个空格隔开。保证 x_i 按递增顺序输入。
- **输出格式**
  - 共一行，一个整数，**表示至少要花多少金币来改造他的机器人。若无论如何他都无法获得至少 k 分，输出 -1。**

数据规模

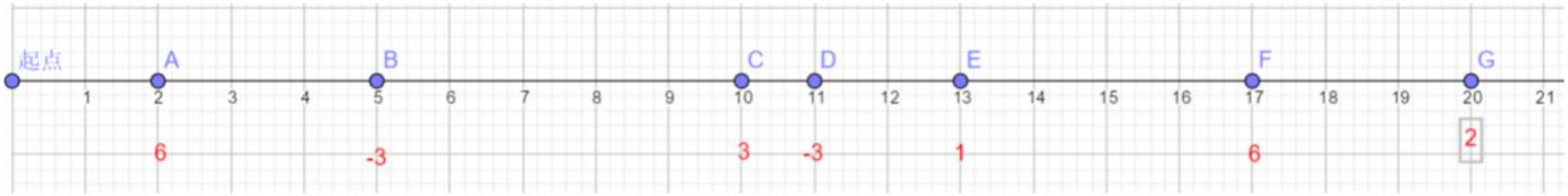
- 对于第 1，2 组测试数据，n ⩽ 10
- 对于第 3，4，5 组测试数据，n ⩽ 500
- 对于第 6，7，8 组测试数据，d = 1
- （数据保证输出的结果不大于1000）

  1 ⩽ n ⩽ 500000，1 ⩽ d ⩽2000，

  1 ⩽ x_i ，k ⩽ 10^9，| s_i | < 10^5。

分析

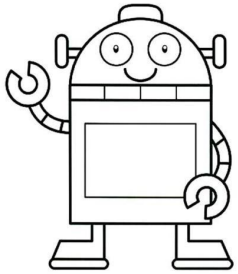
7 4 10　n　d　k 2 6　　dis[i]　w[i] 5 -3 10 3 11 -3 13 1 17 6 20 2	**2**

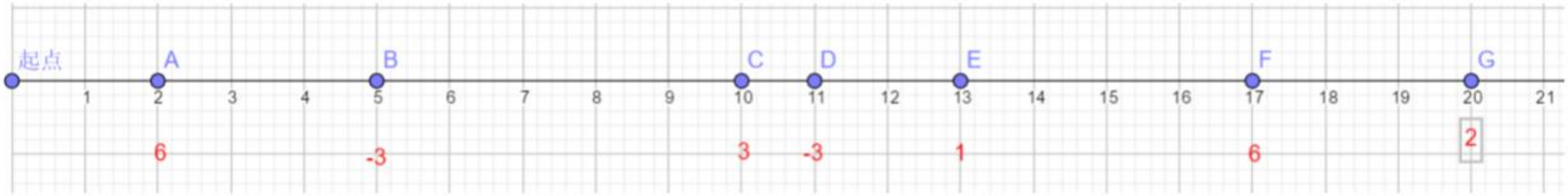


A-G分别表示1号到7号格子，红色文字表示该格子对应的分值

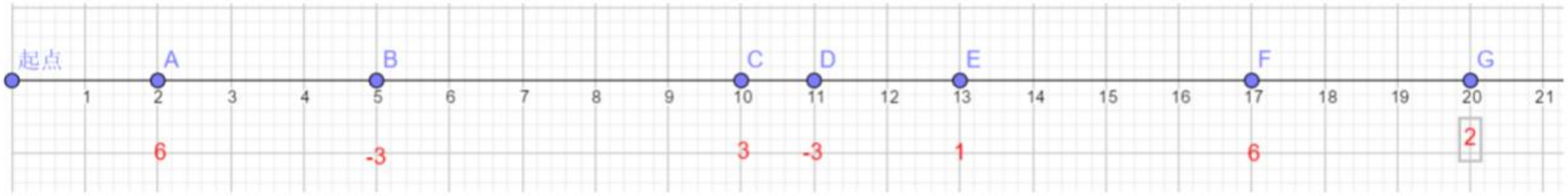
问题1：求解到达位置i的最大分数

d={ 2,3,4,5,6}





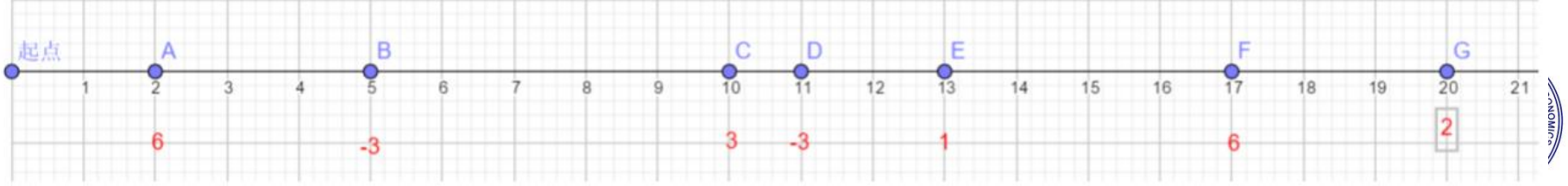
问题1：求解到达位置i的最大分数



初始状态，当d = 4时，无法由起点跳到其它点，此时需要花费2金币改造，改造后机器人的移动范围变为[2,6]，此时：

1. 机器人跳到A点，得6分，总分6分
2. 机器人跳到B点，得-3分，总分3分
3. 机器人跳到C点，得3分，总分6分
4. 机器人跳到E点，得1分，总分7分
5. 机器人跳到F点，得6分，总分13分

所以，当花费2金币进行改造时，得分不低于10分。



类似城市交通网络图，设f[i]表示在花费g个金币进行改造后，跳过前i个格子得到的最高得分

$$f[i]= \max(f[j],f[j]+w[i])$$

其中$\max(1,d-g)\le dist[i]-dist[j]\le d+g$

```
 8  void check(int gold){
 9      int L=max(d-gold,1);
10      int R=d+gold;
11      memset(f,0x80,sizeof(f));
12      f[0]=0;
13      for(int i=1;i<=n;i++){
14          for(int j=i-1;j>=0;j--){
15              if(dis[i]-dis[j]>=L&&dis[i]-dis[j]<=R)
16                  f[i]=max(f[i],f[j]+w[i]);
17          }
18      }
19  }
```

问题1：求解到达位置i的最大分数

```
 8  void check(int gold){
 9      int L=max(d-gold,1);
10      int R=d+gold;
11      memset(f,0x80,sizeof(f));
12      f[0]=0;
13      for(int i=1;i<=n;i++){
14          for(int j=i-1;j>=0;j--){
15              if(dis[i]-dis[j]>=L&&dis[i]-dis[j]<=R)
16                  f[i]=max(f[i],f[j]+w[i]);
17          }
18      }
19  }
```

问题2：判断小R已经满足获得至少 k 分

```
 8  int check(int gold){
 9      int L=max(d-gold,1);
10      int R=d+gold;
11      memset(f,0x80,sizeof(f));
12      f[0]=0;
13      for(int i=1;i<=n;i++){
14          for(int j=i-1;j>=0;j--){
15              if(dis[i]-dis[j]>=L&&dis[i]-dis[j]<=R)
16                  f[i]=max(f[i],f[j]+w[i]);
17              if(f[i]>=k)//这个循环可以放在里面
18                  return 1;
19
20          }
21      }
22      return 0;
23  }
```

剪枝----重要

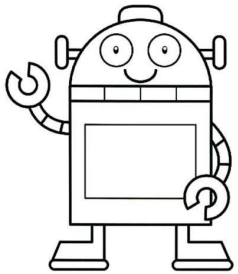
再次考虑时间复杂度 n^2，$1 \leqslant n \leqslant 500000$，必须考虑剪枝

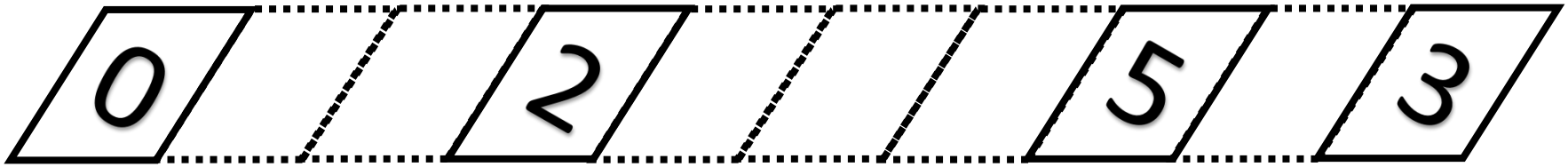
```
for(int i=1;i<=n;i++){
    for(int j=i-1;j>=0;j--){
        if(dis[i]-dis[j]>=L&&dis[i]-dis[j]<=R)
            f[i]=max(f[i],f[j]+w[i]);
    }
}
for(int i=1;i<=n;i++){
    for(int j=i-1;j>=0;j--){
        if(dis[j] + R < dis[i]) break;//1 强剪枝
        if(dis[i]-dis[j]>=L&&dis[i]-dis[j]<=R)
            f[i]=max(f[i],f[j]+w[i]);
    }
}
```

问题3：求解至少要花多少金币

d={ 1,2,3,4,5,6}

k_{max}=7





问题3：求解至少要花多少金币

- 问题转化为：
  - 在一个递增（不递减）的序列中，至少要花多少金币可以满足要求。

使用_____二分查找_____算法

二分查找

```
29  int main()
30  {
31      scanf("%d%d%d",&n,&d,&k);
32      for(int i=1;i<=n;i++)
33          scanf("%lld%lld",&dis[i],&w[i]);
34      int left=0,right=2000,ans=-1;//d的最大
35      while(left<right){//值为2000 right<=d
36          int mid=(left+right)>>1;
37          if(check(mid)){
38              ans=mid;
39              right=mid;
40          }
41          else
42              left=mid+1;
43      }
44      printf("%d\n",ans);
45      return 0;
46  }
```

DP代码

```
 8  int check(int gold){
 9      int L=max(d-gold,1);
10      int R=d+gold;
11      memset(f,0x80,sizeof(f));
12      f[0]=0;
13      for(int i=1;i<=n;i++){
14          for(int j=i-1;j>=0;j--){
15              if(dis[i]-dis[j]>=L&&dis[i]-dis[j]<=R)
16                  f[i]=max(f[i],f[j]+w[i]);
17          }
18      }
19      return 0;
20  }
```

```
1  include <bits/stdc++.h>
2  using namespace std;
3  const int N= 500050;
4  typedef long long LL;
5  LL dis[N],w[N],f[N];
6  int n,d,k;
```

单调队列优化解法

f[i]= max(f[j],f[j]+w[i])
其中max(1,d-g)≤dist[i]-dist[j]≤d+g

可以写成：
f(i)=max (f(j)) +w[i]

可以用滑动窗口的方法来维护f(j)的值,即题目可以转换在在滑动窗口[max(1,d-g),d+g]中求最大值问题。

要注意一下,w[i]可能是负数, 所以对于每个不能走到的格子我们都要设成负无穷。

问题1：求解到达位置i的最大分数

类似城市交通网络图，设f[i]表示在花费g个金币进行改造后，跳过前i个格子得到的最高得分

```
 8  void check(int gold){
 9      int L=max(d-gold,1);
10      int R=d+gold;
11      memset(f,0x80,sizeof(f));
12      f[0]=0;
13      for(int i=1;i<=n;i++){
14          for(int j=i-1;j>=0;j--){
15              if(dis[i]-dis[j]>=L&&dis[i]-dis[j]<=R)
16                  f[i]=max(f[i],f[j]+w[i]);
17          }
18      }
19  }
```

今天的课程结束啦……

下课了…
同学们再见！