动态规划

何为动态规划

基本思想： 动态规划算法通常用于求解具有某种最优性质的问题。在这类问题中， 可能会有很多可行解。每一个解都对应于一个值，我们希望找到具有最优值的解。动态规划算法与分治法类似，其基本思想也是将待求解问题分解为若干个子问题，先求解子问题，然后从这些子问题的解得到原问题的解。与分治法不同的是，适用于动态规划算法求解的问题，经分解得到的子问题往往不是互相独立的。若用分治法来解这类问题，则分解得到的子问题数目太多，有些子问题被重复计算很多次。如果我们能保存已解决子问题的答案，而在需要时再找出已求得的答案，这样就可以避免大量的重复计算，节省时间。我们可以用一个表来记录所有已解决的子问题的答案。不管该子问题以后是否被用到，只要它被计算过，就将其结果填入表中。这就是动态规划算法的基本思路。具体的动态规划算法多种多样，但它们具有相同的填表格式。

与分治法最大的差别是：适用于动态规划求解的问题，**经分解后得到的子问题往往不是互相独立的**（即下一个子阶段的求解是建立在上一个子阶段的解的基础上，进行进一步的求解）

应用场景： 适用于动态规划的问题必须满足最优化原理、无后效性和重叠性。

(1) 最优化原理（最优子结构性质）：一个最优化策略具有这样的性质，不论过去状态和决策如何，对前面的决策所形成的状态而言，余下的决策必须构成最优策略。简而言之，一个最优化策略的子策略总是最优的。一个问题满足最优化原理又称其具有最优子结构性质。

(2) 无后效性：将各阶段按照一定的次序排列好之后，对于某个给定的阶段状态，它以前各阶段的状态无法直接影响它未来的决策，而只能通过当前的这个状态。换句话说，每个状态都是过去历史的一个完整总结。这就是无后向性，又称无后效性。

(3) 子问题的重叠性：动态规划将原来具有指数级时间复杂度的搜索算法改进成了具有多项式时间复杂度的算法。其中的关键在于解决冗余，这就是动态规划算法的根本目的。动态规划实质上是一种以空间换时间的技术，它在实现的过程中，不得不存储产生过程中的各种状态，所以它的空间复杂度要大于其他算法。

背包问题

01背包：每个物体最多拿一次

- 为什么能用一维数组：对dp[i]由影响的只有dp[i-1]
- 为什么第二个循环要从后往前枚举：

  如果从前往后枚举，可以发现对于当前处理的物品i和当前状态dpi(j),在j>=wi时，dpi(j)是会被dpi(j-w[i])所影响的。这就相当于物品i可以多次放入背包，不符合题意（实际上这是完全背包问题的解法）

```
//maxw->总容量,w[i]->重量,v[i]->价值
for(int i=1;i<=n;i++)
    for(int j=maxw;j>=w[i];j--)/*注意从后往前枚举*/
        dp[j]=max(dp[j],dp[j-w[i]]+v[i])
```

完全背包：一个物品可以选无限次

```
//maxw->总容量,w[i]->重量,v[i]->价值
for(int i=1;i<=n;i++)
    for(int j=w[i];j<=maxw;j++)
        dp[j]=max(dp[j],dp[j-w[i]]+v[i])
```

多重背包：每种物品有k个

思路：多重背包 ——> 二进制拆分 ——> 01背包

我们可以通过「二进制分组」的方式使拆分方式更加优美。

具体地说就是令 $A_{i,j}\,(j \in [0, \lfloor \log_2(k_i+1) \rfloor - 1])$ 分别表示由 2^j 个单个物品「捆绑」而成的大物品。特殊地，若 k_i+1 不是 2 的整数次幂，则需要在最后添加一个由 $k_i - 2^{\lfloor \log_2(k_i+1) \rfloor - 1}$ 个单个物品「捆绑」而成的大物品用于补足。

举几个例子：

- $6 = 1 + 2 + 3$
- $8 = 1 + 2 + 4 + 1$
- $18 = 1 + 2 + 4 + 8 + 3$
- $31 = 1 + 2 + 4 + 8 + 16$

显然，通过上述拆分方式，可以表示任意 $\leq k_i$ 个物品的等效选择方式。将每种物品按照上述方式拆分后，使用 0-1 背包的方法解决即可。

时间复杂度 $O(W \sum_{i=1}^{n} \log_2 k_i)$

```
//index->第几个背包
for(int i=1;i<=n;i++)
{
    cin>>a>>b>>c//a->价值,b->重量,c->有几件
    for(int j=1;j<=c;j<<=1)//二进制拆分
    {
        index++;
        v[index]=j*a;
        w[index]=j*b;
        c-=j;
    }
    if(c>=0)//如果最后一个装不满
    {
        index++;
        v[index]=c*a;
        w[index]=c*b;
    }
}//后面用背包板子的话总数当作index
```

混合背包

```
for (循环物品种类)
{
  if (是 0 - 1 背包)
    套用 0 - 1 背包代码;
  else if (是完全背包)
    套用完全背包代码;
```

```
    else if (是多重背包)
      套用多重背包代码;
}//btw,01背包也可以算次数为1的多重背包
```

动态规划其他题型参考资料

动态规划（线性模型，区间模型）

分组背包

动态规划部分简介 - OI Wiki

状态压缩DP

例题

【T1】B3637 最长上升子序列

B3637 最长上升子序列

提交答案 加入题单

题目描述

复制Markdown 展开

这是一个简单的动规板子题。

给出一个由 $n(n \leq 5000)$ 个不超过 10^6 的正整数组成的序列。请输出这个序列的**最长上升子序列**的长度。

最长上升子序列是指，从原序列中**按顺序**取出一些数字排在一起，这些数字是**逐渐增大**的。

输入格式

第一行，一个整数 n，表示序列长度。

第二行有 n 个整数，表示这个序列。

输出格式

一个整数表示答案。

输入输出样例

输入 #1 复制

```
6
1 2 4 1 3 4
```

输出 #1 复制

```
4
```

说明/提示

分别取出 1、2、3、4 即可。

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=1e6;
int n;
int arr[N],dp[N];//dp[i]表示以a[i]结尾的最长子序列长度
int LIS()
{
    int ans=1;
    for(int i=0;i<n;i++)//枚举子序列终点，长度最短为自身，即1
    {
        dp[i]=1;
        for(int j=0;j<i;j++)//从头向终点检索每一个元素
        {
            if(arr[i]>arr[j])
                dp[i]=max(dp[i],dp[j]+1);/***状态转移***/
```

```
        }
        ans=max(ans,dp[i]);//比较每一个dp[i]，最大的那个就是答案
    }
        return ans;
}
int main()
{
    cin>>n;
    for(int i=0;i<n;i++)
        cin>>arr[i];
    int ans=LIS();
    cout<<ans<<endl;
}
/*让我们举个例子：求 2 7 1 5 6 4 3 8 9 的最长上升子序列。我们定义d(i) (i∈[1,n])来表示前 i 个数以A[ i ]结尾的最长上升子序列长度。

    前1个数 d(1)=1 子序列为2;

    前2个数 7前面有2小于7 d(2)=d(1)+1=2 子序列为2 7

    前3个数 在1前面没有比1更小的，1自身组成长度为1的子序列 d(3)=1 子序列为1

    前4个数 5前面有2小于5 d(4)=d(1)+1=2 子序列为2 5

    前5个数 6前面有2 5小于6 d(5)=d(4)+1=3 子序列为2 5 6

    前6个数 4前面有2小于4 d(6)=d(1)+1=2 子序列为2 4

    前7个数 3前面有2小于3 d(3)=d(1)+1=2 子序列为2 3

    前8个数 8前面有2 5 6小于8 d(8)=d(5)+1=4 子序列为2 5 6 8

    前9个数 9前面有2 5 6 8小于9 d(9)=d(8)+1=5 子序列为2 5 6 8 9

    d(i)=max{d(1),d(2),……,d(i)} 我们可以看出这9个数的LIS为d(9)=5*/
```

【T2】LCS

题意翻译

题目描述：

给定一个字符串 s 和一个字符串 t，输出 s 和 t 的最长公共子序列。

输入格式：

两行，第一行输入 s，第二行输入 t。

输出格式：

输出 s 和 t 的最长公共子序列。如果有多种答案，输出任何一个都可以。

说明/提示：

数据保证 s 和 t 仅含英文小写字母，并且 s 和 t 的长度小于等于3000。

输入输出样例

输入 #1 复制

```
axyb
abyxb
```

输出 #1 复制

```
axb
```

输入 #2 复制

```
aa
xayaz
```

输出 #2 复制

```
aa
```

输入 #3 复制

```
a
z
```

输出 #3 复制

```

```

输入 #4 复制

```
abracadabra
avadakedavra
```

输出 #4 复制

```
aaadara
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int dp[1005][1005];
int len1,len2;
string s1,s2;
char ans[1005];
void lcs()//先求lcs的长度
{
    for(int i=1;i<=len1;i++)//要从1开始遍历，不然会出现下标为负数的情况
    {
        for(int j=1;j<=len2;j++)
        {
            if(s1[i-1]==s2[j-1]) dp[i][j]=dp[i-1][j-1]+1;//状态转移（注意下标）
            else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);//
```

```
        }
    }
}
/*
解析:
我们可以用dp[i][j]来表示第一个串的前i位，第二个串的前j位的LCS的长度，那么我们是很容易想到状态转移方程的:
 (1) 如果当前的a[ i ]和b[ j ]相同 (即是有新的公共元素) 那么 dp[ i ] [ j ] = max(dp[ i ] [ j ], dp[ i-1 ] [ j-1 ] + 1);
 (2) 如果不相同，即无法更新公共元素，考虑继承: dp[ i ] [ j ] = max(dp[ i-1 ][ j ] , dp[ i ][ j-1 ])
*/
void find()
{
    int i=len1,j=len2,k=0;//k用来记录答案个数
    while(dp[i][j]>0)
    {
        if(s1[i-1]==s2[j-1])//因为字符串第一个下表是0所以要-1
        {
           ans[k++]=s1[--i];//这样的k是从后往前找的，所以等下要反向输出
           j--;
        }
        else
        {
            if(dp[i-1][j]<dp[i][j-1]) j--;
            else i--;
        }
    }
    for(int i=k-1;i>=0;i--)//总共k个，所以从k-1开始
        cout<<ans[i];

}
int main()
{
    cin>>s1>>s2;
    len1=s1.size();
    len2=s2.size();
    lcs();
    find();
}
```

【T3】P1048 [NOIP2005 普及组] 采药

P1048 [NOIP2005 普及组] 采药

提交答案 加入题单

题目描述

复制Markdown 展开

辰辰是个天资聪颖的孩子，他的梦想是成为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到一个到处都是草药的山洞里对他说：“孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。”

如果你是辰辰，你能完成这个任务吗?

输入格式

第一行有 2 个整数 T（$1 \leq T \leq 1000$）和 M（$1 \leq M \leq 100$），用一个空格隔开，T 代表总共能够用来采药的时间，M 代表山洞里的草药的数目。

接下来的 M 行每行包括两个在 1 到 100 之间（包括 1 和 100）的整数，分别表示采摘某株草药的时间和这株草药的价值。

输出格式

输出在规定的时间内可以采到的草药的最大总价值。

输入输出样例

输入 #1 复制

```
70 3
71 100
69 1
1 2
```

输出 #1 复制

```
3
```

说明/提示

【数据范围】

- 对于 30% 的数据，$M \leq 10$;
- 对于全部的数据，$M \leq 100$。

【题目来源】

NOIP 2005 普及组第三题

```cpp
#include<bits/stdc++.h>//这好像直接空间优化了
using namespace std;
int dp[1005];//dp[t]:在t时间内采的最大价值
int t,m,cost,value;
```

```
int main()
{
    cin>>t>>m;
    for(int i=0;i<m;i++)//一个一个判断
    {
        cin>>cost>>value;
        for(int j=t;j>=0;j--)
        {
            if(j>=cost)
            dp[j]=max(dp[j],dp[j-cost]+value);
        }
    }
    cout<<dp[t];
}
```

【T4】P1060 [NOIP2006 普及组] 开心的金明

P1060 [NOIP2006 普及组] 开心的金明

提交答案 加入题单

题目描述

复制Markdown 展开

金明今天很开心，家里购置的新房就要领钥匙了，新房里有一间他自己专用的很宽敞的房间。更让他高兴的是，妈妈昨天对他说："你的房间需要购买哪些物品，怎么布置，你说了算，只要不超过N元钱就行"。今天一早金明就开始做预算，但是他想买的东西太多了，肯定会超过妈妈限定的N元。于是，他把每件物品规定了一个重要度，分为5等：用整数$1-5$表示，第5等最重要。他还从因特网上查到了每件物品的价格（都是整数元）。他希望在不超过N元（可以等于N元）的前提下，使每件物品的价格与重要度的乘积的总和最大。

设第j件物品的价格为$v[j]$，重要度为$w[j]$，共选中了k件物品，编号依次为$j_1,j_2,\ldots,j_k$，则所求的总和为：

$v[j_1]\times w[j_1]+v[j_2]\times w[j_2]+\ldots+v[j_k]\times w[j_k]$。

请你帮助金明设计一个满足要求的购物单。

输入格式

第一行，为2个正整数，用一个空格隔开：n,m（其中$N(<30000)$表示总钱数，$m(<25)$为希望购买物品的个数。）

从第2行到第$m+1$行，第j行给出了编号为$j-1$的物品的基本数据，每行有2个非负整数vp（其中v表示该物品的价格$(v\le 10000)$，p表示该物品的重要度$(1-5)$

输出格式

1个正整数，为不超过总钱数的物品的价格与重要度乘积的总和的最大值(<100000000)。

输入输出样例

输入 #1 复制

```
1000 5
800 2
400 5
300 5
400 3
200 2
```

输出 #1 复制

```
3900
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int dp[30][30000],value[30],cost[30];//dp[i][j]表示前i个物品，总钱数为j时的最大总和
int n,m;//m钱，n个数
int main()
{
    cin>>m>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>cost[i]>>value[i];
```

```
        value[i]*=cost[i];//value数组表示总收获（重要度*钱）
    }
    for(int i=1;i<=n;i++)
    {
        for(int j=0;j<=m;j++)
        {
            dp[i][j]=dp[i-1][j];
            if(j>=cost[i])
                dp[i][j]=max(dp[i][j],dp[i-1][j-cost[i]]+value[i]);
        }
    }
    cout<<dp[n][m];
}

#include<iostream>-----一维数组优化
#include<algorithm>
using namespace std;
int w[30],v[30],f[50000];//w数组为重要度，v数组为money，f是用来dp的数组
int n,m;//n是总物品个数，m是总钱数
int main()
{
    cin>>m>>n;//输入
    for(int i=1;i<=n;i++)
    {
        cin>>v[i]>>w[i];
        w[i]*=v[i];//w数组在这里意义变为总收获（重要度*money）
    }
       //01背包（参照第二类模板“一维数组优化”）
    for(int i=1;i<=n;i++)
    {
        for(int j=m;j>=v[i];j--)//注意从m开始
        {
            f[j]=max(f[j],f[j-v[i]]+w[i]);//dp
        }
    }
    cout<<f[m]<<endl;//背包大小为m时最大值
}
```

【T5】P1115 最大子段和

P1115 最大子段和

提交答案 加入题单

题目描述

复制Markdown 展开

给出一个长度为 n 的序列 a，选出其中连续且非空的一段使得这段和最大。

输入格式

第一行是一个整数，表示序列的长度 n。

第二行有 n 个整数，第 i 个整数表示序列的第 i 个数字 a_i。

输出格式

输出一行一个整数表示答案。

输入输出样例

输入 #1 复制

```
7
2 -4 3 -1 2 -4 3
```

输出 #1 复制

```
4
```

说明/提示

样例 1 解释

选取 $[3,5]$ 子段 $\{3,-1,2\}$，其和为 4。

数据规模与约定

- 对于 40% 的数据，保证 $n \le 2\times 10^3$。
- 对于 100% 的数据，保证 $1 \le n \le 2\times 10^5$，$-10^4 \le a_i \le 10^4$。

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=2*1e5;
int n,a[N],sum=0;
int maxn=-INT_MAX;
int main()
{
    cin>>n;
    for(int i=0;i<n;i++)
    {
        cin>>a[i];
        if(sum<0) sum=0;//表明从此时开始算子序列，sum的值为当前的数
        sum+=a[i];
```

```
            if(sum>maxn) maxn=sum;
        }
        cout<<maxn;
    }//动态更新
```

【T6】P1216 [USACO1.5][IOI1994]数字三角形 Number Triangles

题目描述

复制Markdown 展开

观察下面的数字金字塔。

写一个程序来查找从最高点到底部任意处结束的路径，使路径经过数字的和最大。每一步可以走到左下方的点也可以到达右下方的点。

```
        7
      3   8
    8   1   0
  2   7   4   4
4   5   2   6   5
```

在上面的样例中,从 $7 \to 3 \to 8 \to 7 \to 5$ 的路径产生了最大

输入格式

第一个行一个正整数 r ,表示行的数目。

后面每行为这个数字金字塔特定行包含的整数。

输出格式

单独的一行,包含那个可能得到的最大的和。

输入输出样例

输入 #1 复制

```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

输出 #1 复制

```
30
```

说明/提示

【数据范围】
对于 100% 的数据，$1 \leq r \leq 1000$，所有输入在 $[0, 100]$ 范围内。

题目翻译来自NOCOW。

USACO Training Section 1.5

IOI1994 Day1T1

```cpp
//从下往上搜(可以直接看出max)
#include<bits/stdc++.h>
using namespace std;
int r,a[1005][1005];
```

```cpp
int main()
{
    cin>>r;
    for(int i=0;i<r;i++)
    {
        for(int j=0;j<=i;j++)
            cin>>a[i][j];
    }
    for(int i=r-2;i>=0;i--)//下标志从n-2开始，因为最后一排不用去推
    {
        for(int j=0;j<=i;j++)
        a[i][j]+=max(a[i+1][j],a[i+1][j+1]);
    }
    cout<<a[0][0];
}
//从上往下搜
#include<bits/stdc++.h>
using namespace std;
int dp[1005][1005],maxx=0;
int main()
{
        int r;cin>>r;
        int a[r+1][r+1];
        memset(a,0,sizeof(a)); //边界都赋值为0，不然下面计算会炸
        for(int i=1;i<=r;i++)
                for(int j=1;j<=i;j++)
                        cin>>a[i][j];
        dp[1][1]=a[1][1]; //第一行为原来的a数组值
        for(int i=2;i<=r;i++)
        {
                for(int j=1;j<=i;j++)
                {//因为每次只能向下或者右下 所以为上一行同列或上一行前一列里最大的加上本
身
                        dp[i][j]=max(dp[i-1][j],dp[i-1][j-1])+a[i][j];
                        maxx=max(maxx,dp[i][j]); //刷新maxx
                }
        }
        cout<<maxx;
}
```

【T7】P1616 疯狂的采药

题目描述

LiYuxiang 是个天资聪颖的孩子，他的梦想是成为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到一个到处都是草药的山洞里对他说：“孩子，这个山洞里有一些不同种类的草药，采每一种都需要一些时间，每一种也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。”

如果你是 LiYuxiang，你能完成这个任务吗?

此题和原题的不同点：

1. 每种草药可以无限制地疯狂采摘。

2. 药的种类眼花缭乱，采药时间好长好长啊！师傅等得菊花都谢了！

输入格式

输入第一行有两个整数，分别代表总共能够用来采药的时间 t 和代表山洞里的草药的数目 m。

第 2 到第 $(m+1)$ 行，每行两个整数，第 $(i+1)$ 行的整数 a_i, b_i 分别表示采摘第 i 种草药的时间和该草药的价值。

输出格式

输出一行，这一行只包含一个整数，表示在规定的时间内，可以采到的草药的最大总价值。

输入输出样例

输入 #1 复制

```
70 3
71 100
69 1
1 2
```

输出 #1 复制

```
140
```

说明/提示

数据规模与约定

- 对于 30% 的数据，保证 $m \leq 10^3$ 。
- 对于 100% 的数据，保证 $1 \leq m \leq 10^4$，$1 \leq t \leq 10^7$，且 $1 \leq m \times t \leq 10^7$，$1 \leq a_i, b_i \leq 10^4$ 。

```cpp
#include<bits/stdc++.h>
using namespace std;
const long long N=1e7;
long long m;
long long t;
struct pick
{
    long long cost;
    long long value;
}arr[10005];
```

```
long long ans[N];
int main()
{
    cin>>t>>m;
    for(long long i=0;i<m;i++)
        cin>>arr[i].cost>>arr[i].value;
    for(long long i=0;i<m;i++)
    {
        for(long long j=arr[i].cost;j<=t;j++)
            ans[j]=max(ans[j],ans[j-arr[i].cost]+arr[i].value);
    }
    cout<<ans[t];
}
```

【T8】P2196 [NOIP1996 提高组] 挖地雷

P2196 [NOIP1996 提高组] 挖地雷

提交答案 加入题单

题目描述

复制Markdown 展开

在一个地图上有N个地窖$(N \leq 20)$，每个地窖中埋有一定数量的地雷。同时，给出地窖之间的连接路径。当地窖及其连接的数据给出之后，某人可以从任一处开始挖地雷，然后可以沿着指出的连接往下挖（仅能选择一条路径），当无连接时挖地雷工作结束。设计一个挖地雷的方案，使某人能挖到最多的地雷。

输入格式

有若干行。

第1行只有一个数字，表示地窖的个数N。

第2行有N个数，分别表示每个地窖中的地雷个数。

第3行至第$N+1$行表示地窖之间的连接情况：

第3行有$n-1$个数（0或1），表示第一个地窖至第2个、第3个、...、第n个地窖有否路径连接。如第3行为$11000\ldots0$，则表示第1个地窖至第2个地窖有路径，至第3个地窖有路径，至第4个地窖、第5个、...、第n个地窖没有路径。

第4行有$n-2$个数，表示第二个地窖至第3个、第4个、...、第n个地窖有否路径连接。

... ...

第$n+1$行有1个数，表示第$n-1$个地窖至第n个地窖有否路径连接。（为0表示没有路径，为1表示有路径）。

输出格式

有两行

第一行表示挖得最多地雷时的挖地雷的顺序，各地窖序号间以一个空格分隔，不得有多余的空格。

第二行只有一个数，表示能挖到的最多地雷数。

输入输出样例

输入 #1 复制

```
5
10 8 4 7 6
1 1 1 0
0 0 0
1 1
1
```

输出 #1 复制

```
1 3 4 5
27
```

```cpp
#include<bits/stdc++.h>
using namespace std;
bool mp[25][25];//是否为联通路
int cnt[25],dp[1005],pre[25];//cnt：地雷数目，pre：前驱
int n,pos,ans=0;
void dfs(int x)//倒着输出
{
  if(pre[x])
    dfs(pre[x]);//如果x有前驱，就继续搜
  cout<<x<<' ';
}
int main()
{
  cin>>n;
  for(int i=1;i<=n;i++)
    cin>>cnt[i];
  for(int i=1;i<=n-1;i++)
  {
    for(int j=i+1;j<=n;j++)
      cin>>mp[i][j];
  }
  dp[1]=cnt[1];//初始化
  for(int i=2;i<=n;i++)
  {
    dp[i]=cnt[i];
    for(int j=i-1;j>0;j--)//枚举寻找最大值
    {
      if(mp[j][i]&&dp[i]<dp[j]+cnt[i])//有路径且大于目前最大值
      {
        dp[i]=dp[j]+cnt[i];//更新
        pre[i]=j;//记录i的前驱为j
      }
    }
    if(ans<dp[i])//跟新答案
    {
      ans=dp[i];
      pos=i;//记录取点的位置
    }
  }
  dfs(pos);
  cout<<endl<<ans;
}
//dp做法
#include<bits/stdc++.h>
#define ll long long
using namespace std;
ll dp[25];//以i结尾挖到的地雷数为dp[i]
ll mp[25][25],n,a[25],path[25];
void dfs(ll x){
        if(path[x])dfs(path[x]);
        cout<<x<<" ";
}
int main(){
        cin>>n;
        for(int i=1;i<=n;i++){
                cin>>a[i];
```

```
        }
        for(int i=1;i<=n;i++){
                for(int j=i+1;j<=n;j++){
                        cin>>mp[i][j];
                }
        }
        for(int i=1;i<=n;i++){
            dp[i]=a[i];//初始化
            for(int j=i-1;j>0;j--){
                        if(mp[j][i]&&dp[i]<dp[j]+a[i]){
                    //当j可以通向i且这样挖到的地雷数更多时更新路径和dp数组
                                dp[i]=dp[j]+a[i];
                                path[i]=j;
                        }
                }
        }
        ll maxx=-1,pos=0;
for(int i=1;i<=n;i++){
        if(maxx<dp[i]){
                pos=i;//找到以哪个地窖结尾地雷数最大
                maxx=dp[i];//找地雷数的最大值
        }
}
        dfs(pos);//递归输出
        cout<<endl;
        cout<<maxx;
        return 0;
}
```

【T9】P8697 [蓝桥杯 2019 国 C] 最长子序列

P8697 [蓝桥杯 2019 国 C] 最长子序列

提交答案 加入题单

题目描述

复制Markdown 展开

我们称一个字符串 S 包含字符串 T 是指 T 是 S 的一个子序列，即可以从字符串 S 中抽出若干个字符，它们按原来的顺序组合成一个新的字符串与 T 完全一样。给定两个字符串 S 和 T，请问 T 中从第一个字符开始最长连续多少个字符被 S 包含?

输入格式

输入两行，每行一个字符串。第一行的字符串为 S，第二行的字符串为 T。两个字符串均非空而且只包含大写英文字母。

输出格式

输出一个整数，表示答案。

输入输出样例

输入 #1 复制

```
ABCDEABCD
AABZ
```

输出 #1 复制

```
3
```

说明/提示

对于 20% 的评测用例，$1 \leq |T| \leq |S| \leq 20$;

对于 40% 的评测用例，$1 \leq |T| \leq |S| \leq 100$;

对于所有评测用例，$1 \leq |T| \leq |S| \leq 1000$。

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    string s1, s2;
    cin >> s1 >> s2;
    int len = 0;
    for(int i = 0, j = 0; i < s1.size(); i++)
    {
        if(s1[i] == s2[j])
        {
            j++;
            len++;
        }
    }
    cout << len;
```

```
    return 0;
}//双指针解法
```

【T10】P8742 [蓝桥杯 2021 省 AB] 砝码称重

P8742 [蓝桥杯 2021 省 AB] 砝码称重

提交答案 加入题单

题目描述

复制Markdown 展开

你有一架天平和 N 个砝码, 这 N 个砝码重量依次是 $W_1, W_2, \cdots, W_N$。请你计算一共可以称出多少种不同的重量?

注意砝码可以放在天平两边。

输入格式

输入的第一行包含一个整数 N。

第二行包含 N 个整数: $W_1, W_2, W_3, \cdots, W_N$。

输出格式

输出一个整数代表答案。

输入输出样例

输入 #1 复制

```
3
1 4 6
```

输出 #1 复制

```
10
```

说明/提示

【样例说明】

能称出的 10 种重量是: 1、2、3、4、5、6、7、9、10、11。

$$
\begin{aligned}
1 &= 1 \\
2 &= 6 - 4 (\text{天平一边放 } 6, \text{ 另一边放 } 4) \\
3 &= 4 - 1 \\
4 &= 4 \\
5 &= 6 - 1 \\
6 &= 6 \\
7 &= 1 + 6 \\
9 &= 4 + 6 - 1 \\
10 &= 4 + 6 \\
11 &= 1 + 4 + 6
\end{aligned}
$$

【评测用例规模与约定】

对于 50% 的评测用例, $1 \leq N \leq 15$。

对于所有评测用例, $1 < N < 100$, N 个砝码总重不超过 10^5。

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;
int n,a[N],dp[105][N];//a[i]->砝码
int main()
{
    cin>>n;
    for(int i=0;i<n;i++)
    {
        cin>>a[i];
        if(!i)//考虑i=0（就是放第一个砝码的情况）
        {
            dp[i][a[i]]=1;
            dp[i][0]=1;
        }
        else
        {
            for(int j=0;j<N;j++)
                if(dp[i-1][j])//如果上一个状态的那些砝码能称出j重量的话
                {
                    dp[i][j]=1;//不放第i个
                    dp[i][j+a[i]]=1;//把第i个放右边
                    dp[i][abs(j-a[i])]=1;//把第i个放左边
                }
        }
    }
    int ans=0;
    for(int i=1;i<N;i++)
        ans+=dp[n-1][i];//dp[n-1][i]表示用这n个砝码能否称出i重量（能的话就+1）
    cout<<ans;
}
```

【T11】P1064 [NOIP2006 提高组] 金明的预算方案

P1064 [NOIP2006 提高组] 金明的预算方案

提交答案 加入题单

题目描述

复制Markdown 展开

金明今天很开心，家里购置的新房就要领钥匙了，新房里有一间金明自己专用的很宽敞的房间。更让他高兴的是，妈妈昨天对他说：“你的房间需要购买哪些物品，怎么布置，你说了算，只要不超过 n 元钱就行”。今天一早，金明就开始做预算了，他把想买的物品分为两类：主件与附件，附件是从属于某个主件的，下表就是一些主件与附件的例子：

主件	附件
电脑	打印机，扫描仪
书柜	图书
书桌	台灯，文具
工作椅	无

如果要买归类为附件的物品，必须先买该附件所属的主件。每个主件可以有 0 个、1 个或 2 个附件。每个附件对应一个主件，附件不再有从属于自己的附件。金明想买的东西很多，肯定会超过妈妈限定的 n 元。于是，他把每件物品规定了一个重要度，分为 5 等：用整数 $1 \sim 5$ 表示，第 5 等最重要。他还从因特网上查到了每件物品的价格（都是 10 元的整数倍）。他希望在不超过 n 元的前提下，使每件物品的价格与重要度的乘积的总和最大。

设第 j 件物品的价格为 v_j，重要度为w_j，共选中了 k 件物品，编号依次为 $j_1, j_2, \ldots, j_k$，则所求的总和为：

$v_{j_1} \times w_{j_1} + v_{j_2} \times w_{j_2} + \cdots + v_{j_k} \times w_{j_k}$。

请你帮助金明设计一个满足要求的购物单。

输入格式

第一行有两个整数，分别表示总钱数 n 和希望购买的物品个数 m。

第 2 到第 $(m+1)$ 行，每行三个整数，第 $(i+1)$ 行的整数 v_i，p_i，q_i 分别表示第 i 件物品的价格、重要度以及它对应的的主件。如果 $q_i = 0$，表示该物品本身是主件。

输出格式

输出一行一个整数表示答案。

输入输出样例

输入 #1 复制

```
1000 5
800 2 0
400 5 1
300 5 1
400 3 0
```

输出 #1 复制

```
2200
```

```
500 2 0
```

```
#include<bits/stdc++.h>
using namespace std;
const int N=32005;
int main_v[65],main_c[65];//主件
int attach_v[65][3],attach_c[65][3];//附件,_v->value;_c->cost
int dp[N];
int n,m;
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        int a,b,c;
        cin>>a>>b>>c;
        if(c!=0)
        {
            attach_c[c][0]++;//第二维第0个数用来判断是第几个附件
            attach_c[c][attach_c[c][0]]=a;
            attach_v[c][attach_c[c][0]]=a*b;//初始化这个附件的参数(这个方法不错)
        }
        else
        {
            main_c[i]=a;//主件
            main_v[i]=a*b;
        }
    }
    for(int i=1;i<=m;i++)
    {
        for(int j=n;main_c[i]!=0&&j>=main_c[i];j--)//main_c[i]!=0保证一定先选主件再考
虑选附件
        {                                          //从大到小遍历
            dp[j]=max(dp[j],dp[j-main_c[i]]+main_v[i]);
            if(j>=main_c[i]+attach_c[i][1])//是否选第一个附件
                dp[j]=max(dp[j],dp[j-main_c[i]-attach_c[i]
[1]]+main_v[i]+attach_v[i][1]);
            if(j>=main_c[i]+attach_c[i][2])//是否选第二个附件
                dp[j]=max(dp[j],dp[j-main_c[i]-attach_c[i]
[2]]+main_v[i]+attach_v[i][2]);
             if(j>=main_c[i]+attach_c[i][2]+attach_c[i][1])//是否都选
                dp[j]=max(dp[j],dp[j-main_c[i]-attach_c[i][1]-attach_c[i]
[2]]+main_v[i]+attach_v[i][1]+attach_v[i][2]);
        }
    }
    cout<<dp[n];
}
```

【T12】P1077 [NOIP2012 普及组] 摆花

P1077 [NOIP2012 普及组] 摆花

提交答案 加入题单

题目描述

复制Markdown 展开

小明的花店新开张，为了吸引顾客，他想在花店的门口摆上一排花，共 m 盆。通过调查顾客的喜好，小明列出了顾客最喜欢的 n 种花，从 1 到 n 标号。为了在门口展出更多种花，规定第 i 种花不能超过 a_i 盆，摆花时同一种花放在一起，且不同种类的花需按标号的从小到大的顺序依次摆列。

试编程计算，一共有多少种不同的摆花方案。

输入格式

第一行包含两个正整数 n 和 m，中间用一个空格隔开。

第二行有 n 个整数，每两个整数之间用一个空格隔开，依次表示 $a_1, a_2, \cdots, a_n$。

输出格式

一个整数，表示有多少种方案。注意：因为方案数可能很多，请输出方案数对 $10^6 + 7$ 取模的结果。

输入输出样例

输入 #1 复制

```
2 4
3 2
```

输出 #1 复制

```
2
```

说明/提示

【数据范围】

对于 20% 数据，有 $0 < n \leq 8, 0 < m \leq 8, 0 \leq a_i \leq 8$。

对于 50% 数据，有 $0 < n \leq 20, 0 < m \leq 20, 0 \leq a_i \leq 20$。

对于 100% 数据，有 $0 < n \leq 100, 0 < m \leq 100, 0 \leq a_i \leq 100$。

NOIP 2012 普及组 第三题

```cpp
#include<bits/stdc++.h>
using namespace std;
const int mod=1e6+7;
int n,m;
int amt[105];
long long dp[105][105];//dp[i][j]前i种取j朵
int main()              //dp[i][j]=dp[i-1][j]+dp[i-1][j-1]+....dp[i-1][j-amt[i]]
{                       //dp[0][0]=1(啥都不取也算一种)
```

```
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>amt[i];
    dp[0][0]=1;
    for(int i=1;i<=n;i++)
    {
        for(int j=0;j<=m;j++)
        {
            for(int k=0;k<=min(j,amt[i]);k++)//多了一次循环
                dp[i][j]=(dp[i][j]+dp[i-1][j-k])%mod;//每一次都除mod就行
        }
    }
    cout<<dp[n][m];
}
//滚动数组:
#include<bits/stdc++.h>
using namespace std;
const int maxn=105, mod = 1000007;
int n, m, a[maxn], f[2][maxn], t;
int main()
{
    cin>>n>>m;
    for(int i=1; i<=n; i++) cin>>a[i];
    f[0][0] = 1;
    for(int i=1; i<=n; i++)
    {
        t = 1-t; //滚动
        for(int j=0; j<=m; j++)
        {
            f[t][j] = 0; //注意初始化
            for(int k=0; k<=min(j, a[i]); k++)
              f[t][j] = (f[t][j] + f[1-t][j-k])%mod;
        }
    }
    cout<<f[t][m]<<endl;
    return 0;
}
//一维01背包
#include<bits/stdc++.h>
using namespace std;
const int maxn=105, mod = 1000007;
int n, m, a[maxn], f[maxn];
int main()
{
    cin>>n>>m;
    for(int i=1; i<=n; i++) cin>>a[i];
    f[0] = 1;
    for(int i=1; i<=n; i++)
        for(int j=m; j>=0; j--) //注意，是01背包
            for(int k=1; k<=min(a[i], j); k++)
              f[j] = (f[j] + f[j-k])%mod;
    cout<<f[m]<<endl;
    return 0;
}
//dfs+记忆化搜索
#include<bits/stdc++.h>
using namespace std;
const int maxn=105, mod = 1000007;
```

```cpp
int n, m, a[maxn], rmb[maxn][maxn];
int dfs(int x,int k)
{
    if(k > m) return 0;
    if(k == m) return 1;
    if(x == n+1) return 0;
    if(rmb[x][k]) return rmb[x][k]; //搜过了就返回
    int ans = 0;
    for(int i=0; i<=a[x]; i++) ans = (ans + dfs(x+1, k+i))%mod;
    rmb[x][k] = ans; //记录当前状态的结果
    return ans;
}
int main()
{
    cin>>n>>m;
    for(int i=1; i<=n; i++) cin>>a[i];
    cout<<dfs(1,0)<<endl;
    return 0;
}
```

【T13】P1833 樱花

P1833 樱花

提交答案 加入题单

题目背景

复制Markdown 展开

《爱与愁的故事第四弹·plant》第一章。

题目描述

爱与愁大神后院里种了 n 棵樱花树，每棵都有美学值 $C_i(0 \le C_i \le 200)$。爱与愁大神在每天上学前都会来赏花。爱与愁大神可是生物学霸，他懂得如何欣赏樱花：一种樱花树看一遍过，一种樱花树最多看 $A_i(0 \le A_i \le 100)$ 遍，一种樱花树可以看无数遍。但是看每棵樱花树都有一定的时间 $T_i(0 \le T_i \le 100)$。爱与愁大神离去上学的时间只剩下一小会儿了。求解看哪几棵樱花树能使美学值最高且爱与愁大神能准时（或提早）去上学。

输入格式

共 $n+1$ 行：

第 1 行：现在时间 T_s（几时：几分），去上学的时间 T_e（几时：几分），爱与愁大神院子里有几棵樱花树 n。这里的 T_s，T_e 格式为：`hh:mm`，其中 $0 \le hh \le 23$，$0 \le mm \le 59$，且 hh, mm, n 均为正整数。

第 2 行到第 $n+1$ 行，每行三个正整数：看完第 i 棵树的耗费时间 T_i，第 i 棵树的美学值 C_i，看第 i 棵树的次数 P_i（$P_i=0$ 表示无数次，P_i 是其他数字表示最多可看的次数 P_i）。

输出格式

只有一个整数，表示最大美学值。

输入输出样例

输入 #1 复制

```
6:50 7:00 3
2 1 0
3 3 1
4 5 4
```

输出 #1 复制

```
11
```

说明/提示

100% 数据：$T_e - T_s \le 1000$（即开始时间距离结束时间不超过 1000 分钟），$n \le 10000$。保证 T_e, T_s 为同一天内的时间。

样例解释：赏第一棵樱花树一次，赏第三棵樱花树 2 次。

```
#include<bits/stdc++.h>//二进制拆分->01背包
const int N=1e6+5;
```

```
int total_time,n,start_h,start_m,end_h,end_m;
int cost[10005],value[10005],chance[10005];//cost->花费时间 chance->观看次数 value->
价值
int dp[1005];
int tp,co[N],v[N];
using namespace std;
void binary_split()//进行二进制拆分
{
    for(int i=1;i<=n;i++)
    {
        int t=1;
        while(chance[i]!=0)//只要次数不为0就一直拆分下去
        {
            tp++;//tp表示这个大背包拆分成的第几个子背包
            co[tp]=t*cost[i];//这个子背包的cost->个数*单个物品cost
            v[tp]=t*value[i];//这个子背包的value->同上
            chance[i]-=t;//表示用掉了t次选择机会
            t*=2;//从前往后拆出来的背包容量差两倍
            if(chance[i]<t)//如果剩下的不能再拆就直接放一起
            {
                tp++;//开一个子背包给剩余的物品
                co[tp]=cost[i]*chance[i];//剩余物品总cost
                v[tp]=value[i]*chance[i];//...总value
                break;//---别漏
            }
        }
    }
}
int main()
{
    scanf("%d:%d%d:%d",&start_h,&start_m,&end_h,&end_m);
    /*也可以这样输入时间
    for(int i = 1 ; i <= 2 ; i ++)//输两遍
    {
        cin >> a >> ch >> b ;//输入
        pre[t] = a * 60 + b ;//时间
        t ++ ;
    }
    */
    cin>>n;
    total_time=(end_h*60+end_m)-(start_h*60+start_m);//总共有的时间
    for(int i=1;i<=n;i++)                          //ps.第一次做把开始的-结束的...找了
半天没发现哪错了
    {
        cin>>cost[i]>>value[i]>>chance[i];
        if(chance[i]==0)
            chance[i]=999999;//不要太大，一般百万就足够了（开INT_MAX会炸）
    }
    binary_split();
    for(int i=1;i<=tp;i++)//考虑每个拆出来的物品
    {
        for(int j=total_time;j>=co[i];j--)//01背包板子
            dp[j]=max(dp[j],dp[j-co[i]]+v[i]);
    }
    cout<<dp[total_time];
}
//碰到这种多重背包（01背包也可以看作只能拿一次的多重背包）和完全背包混在一起的可以这样写：
```

```
//但是这一题这样会超时
for(int i=1;i<=n;i++)
        {
                if(a[i]==0)//如果为完全背包
                {
                        for(int j=t[i];j<=tz;j++) dp[j]=max(dp[j],dp[j-
t[i]]+c[i]);//记得是正序
                }
                else
                {
                        for(int l=1;l<=a[i];l++)//重复a[i]次01背包的结果就相当于最多取
a[i]个的多重背包
                        for(int j=tz;j>=t[i];j--) //01背包,倒序
                        {
                                dp[j]=max(dp[j],dp[j-t[i]]+c[i]);
                        }
                }
        }
--------这种方法可以优化-----------
二.优化
我们可以发现当第ii个朵花,重复第kk次01背包时,对于第ii朵花
dp[k*t[i]]dp[k*t[i]]前(不包括本身)的值都是已经确定了的,
由于前面都是确定的就不要再循环到了

解释
我们先看第一次01背包时,dp[t[i]]dp[t[i]]之前的都是可以确定对于第ii朵花,
是一定为0的(也就他们的贡献为0),因为消耗那么多时间根本不够看第ii朵花

所以在第一次01背包中:dp[2*t[i]-1]dp[2*t[i]-1]
由已经确定的dp[t[i]-1]dp[t[i]-1]得来,
dp[2*t[i]-2]dp[2*t[i]-2]由已经确定的dp[t[i]-2]dp[t[i]-2]得来
...dp[t[i]]dp[t[i]]由已经确定的dp[0]dp[0]得来,
所以他们在之后的第二次01背包中也是确定的

第二次01背包与第一次一样由dp[2*t[i]]dp[2*t[i]]之前都是确定的
可以得出dp[3*t[i]]dp[3*t[i]]之前都是确定的给第三次01用,
这样重复直到次数的上限结束
for(int i=1;i<=n;i++)
        {
                if(a[i]==0)//完全背包和前面一样
                {
                        for(int j=t[i];j<=tz;j++) dp[j]=max(dp[j],dp[j-t[i]]+c[i]);
                }
                else
                {
                    for(int l=1;l<=a[i];l++)
    ------>for(int j=tz;j>=l*t[i];j--) //倒序,前面确定的不要循环到
                        {
                                dp[j]=max(dp[j],dp[j-t[i]]+c[i]);
                        }
                }
        }
```

【T14】P8707 [蓝桥杯 2020 省 AB1] 走方格

P8707 [蓝桥杯 2020 省 AB1] 走方格

提交答案 加入题单

题目描述

复制Markdown 展开

在平面上有一些二维的点阵。

这些点的编号就像二维数组的编号一样，从上到下依次为第 1 至第 n 行，从左到右依次为第 1 至第 m 列，每一个点可以用行号和列号来表示。

现在有个人站在第 1 行第 1 列，要走到第 n 行第 m 列。只能向右或者向下走。

注意，如果行号和列数都是偶数，不能走入这一格中。

问有多少种方案。

输入格式

输入一行包含两个整数 n，m。

输出格式

输出一个整数，表示答案。

输入输出样例

输入 #1 复制

```
3 4
```

输出 #1 复制

```
2
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int dp[35][35];//dp[i][j]表示走到i，j的路径数
int n,m;       //由于只能向下或向右走，故有dp[i][j]=dp[i-1][j]+dp[i][j-1]
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)dp[i][1]=1;
    for(int i=1;i<=m;i++)dp[1][i]=1;//把初始条件初始化
    for(int i=2;i<=n;i++)
    {
        for(int j=2;j<=m;j++)
        {
            if(i%2==0&&j%2==0)dp[i][j]=0;
            else dp[i][j]=dp[i-1][j]+dp[i][j-1];//转移
        }
    }
    cout<<dp[n][m];
}
```

```cpp
#include<bits/stdc++.h>//dfs做法--会TLE
using namespace std;
int n,m,ans=0;
int x_plus[4]={1,0};
int y_plus[4]={0,1};
bool mp[35][35];//默认初始化为false
void dfs(int x,int y)
{
    if(x==n-1&&y==m-1)
    {
        ans++;
        return;
    }
    mp[x][y]=true;
    for(int i=0;i<2;i++)
    {
        int x_next=x+x_plus[i];
        int y_next=y+y_plus[i];
        if(!mp[x_next][y_next]&&x_next<n&&y_next<m&&((x_next%2)==0||(y_next%2)==0))
        {
            dfs(x_next,y_next);
            mp[x_next][y_next]=false;
        }

    }
}
int main()
{
    cin>>n>>m;
    dfs(0,0);
    cout<<ans;
}
```