[一、 贪心算法 1](#_Toc452415130)

[1、 区间选点 1](#_Toc452415131)

[2、 区间覆盖 1](#_Toc452415132)

[3、 不相交区间 2](#_Toc452415133)

[4、 哈夫曼编码 2](#_Toc452415134)

[5、 最小值最大化、最大值最小化（二分查找） 2](#_Toc452415135)

[二、 动态规划 7](#_Toc452415136)

[1、 最长公共子序列（LCS） 7](#_Toc452415137)

[2、 最长上升公共子序列（LIS） 9](#_Toc452415138)

[3、 子段和 13](#_Toc452415139)

[4、 DAG上的动态规划 19](#_Toc452415140)

[5、 区间DP 25](#_Toc452415141)

[6、 状态压缩DP 35](#_Toc452415142)

[7、 双线DP 38](#_Toc452415143)

[8、 背包问题(见背包九讲) 41](#_Toc452415144)

[三、 数据结构 41](#_Toc452415145)

[1、 并查集 41](#_Toc452415146)

[2、 树状数组 43](#_Toc452415147)

[3、 （字符串）KMP匹配 45](#_Toc452415148)

[四、 最小生成树算法 48](#_Toc452415149)

[Prime核心算法 48](#_Toc452415150)

[Kruskal算法 53](#_Toc452415151)

[五、 单源最短路径 57](#_Toc452415152)

[Dijkstra核心算法 57](#_Toc452415153)

[Bellman\_Ford算法 61](#_Toc452415154)

[SPFA算法（Bellman\_Ford的队列实现） 64](#_Toc452415155)

[六、 二分图匹配 67](#_Toc452415156)

[1、 匈牙利算法 67](#_Toc452415157)

[七、 网络流 68](#_Toc452415158)

[1、 SAP算法 68](#_Toc452415159)

[2、 Dinic算法 70](#_Toc452415160)

## 贪心算法

### 区间问题

**区间选点**

选取尽量少的点覆盖所有的区间，是每个区间至少包含一个点。对区间右端点进行排序。

**区间覆盖**

选取尽量少的区间覆盖整个区域。对左端点进行排序。

**不相交区间**

选取尽量多的不相交区间。对区间右端点进行排序。

### 哈夫曼编码

### 最小值最大化、最大值最小化（二分查找）

NYOJ 疯牛问题（最小值最大化）

农夫 John 建造了一座很长的畜栏，它包括N (2 <= N <= 100,000)个隔间，这些小隔间依次编号为x1,...,xN (0 <= xi <= 1,000,000,000).  
但是，John的C (2 <= C <= N)头牛们并不喜欢这种布局，而且几头牛放在一个隔间里，他们就要发生争斗。为了不让牛互相伤害。John决定自己给牛分配隔间，使任意两头牛之间的最小距离尽可能的大，那么，这个**最大的最小距离是**什么呢？

#include <cstdio>

#include <iostream>

#include <algorithm>

using namespace std;

int n, c;

int pos[100005];

bool judge(int k)

{

int cnt = 1;

int st = pos[0];

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

{

if(pos[i] - st >= k)

{

++cnt;

if(cnt >= c)

return true;

st = pos[i];

}

}

return false;

}

int Binary\_search(int left, int right) /// 二分枚举满足条件的最大距离

{

while(left <= right)

{

int mid = (left + right) >> 1;

if(judge(mid)) /// 所求距离 >= mid，可以继续增大试探

left = mid+1;

else /// 所求距离 < mid,所以必须减小来试探

right = mid-1;

}

return left-1;

}

int main()

{

while(~scanf("%d%d", &n, &c))

{

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

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

sort(pos, pos+n);

printf("%d\n", Binary\_search(0, pos[n-1] - pos[0]));

}

return 0;

}

NYOJ 摘枇杷（最大值最小化）

理工学院的枇杷快熟了，ok，大家都懂得。而且大家都知道，学校的枇杷树都是一列一列的。现在小Y同学已经在筹划怎么摘枇杷了。现在我们假设有一列枇杷树，而且每棵枇杷树上枇杷果的数量小Y都已经知道了。

假设现在有n棵枇杷树，小Y可以把这n棵枇杷树分成m组，每组枇杷果的数量是这组内每棵枇杷树上枇杷果数量的和。注意，每组的枇杷树必须是连续的。（每组最少1棵树，最多n棵树）。小Y把枇杷往寝室拿的时候是一组一组拿的，所花费的力气等于这m组中枇杷果最多的那组枇杷果的数量。现在小Y想**花尽量少的力气把这些枇杷果拿回寝室**。

#include <stdio.h>

#include <string.h>

int n, m, a[1005];

bool judge(int x)

{

int s=0,count=0;

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

{

if(s+a[i]>x)

{

count++;

s=a[i];

if(count>m-1) ///当count==m时，而i<n, 则当每次运送x时，m组内运不完

return false;

}

else

s += a[i];

}

return true; ///每次运送x时，m组内就能运完

}

int Binary\_search(int l, int r)

{

while(l<=r)

{

int mid = (l+r)/2;

if(judge(mid)) r = mid-1;

else l = mid+1;

}

return l; ///当从while中退出时，最后一次操作是r=mid-1,原本的l<=r不再满足 ,所以l才是所要求的值

}

int main()

{

while(~scanf("%d%d", &n, &m))

{

memset(a,0,sizeof(a));

int Max=0, sum=0;

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

{

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

sum += a[i];

Max = Max>a[i]?Max:a[i];

}

printf("%d\n", Binary\_search(Max, sum));

}

return 0;

}

## 动态规划

最长公共子序列（LCS）

最长公共子序列也称作最长公共子串(不要求连续)，英文缩写为LCS（Longest Common Subsequence）。其定义是，一个序列 S ，如果分别是两个或多个已知序列的子序列，且是所有符合此条件序列中最长的，则 S 称为已知序列的最长公共子序列。

二维表：

for(int i=1; i<=n1; i++)

{

for(int j=1; j<=n2; j++)

{

if(a[i]==b[j]) c[i][j] = c[i-1][j-1] + 1;

else if(c[i-1][j]>=c[i][j-1]) c[i][j] = c[i-1][j];

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

}

}

空间优化（滚动数组）

#include <stdio.h>

#include <string.h>

int dp[1005];

int main()

{

int t;

scanf("%d", &t);

while(t--)

{

getchar();

char a[1005]={0},b[1005]={0};

memset(dp, 0, sizeof(dp));

scanf("%s%s", a+1,b+1);

int la = strlen(a+1);

int lb = strlen(b+1);

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

{

int old = 0; //old表示c[i-1][j-1]

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

{

int tmp = dp[j]; //注意

if(a[i]==b[j]) dp[j] = old+1;

else if(dp[j-1]>dp[j]) dp[j] = dp[j-1];

old = tmp;

}

}

printf("%d\n", dp[lb]);

}

return 0;

}

时间优化（转化为LIS 时间复杂度O（nlogn））见LIS

### 最长上升公共子序列（LIS）

最长递增子序列问题：在一列数中寻找一些数，这些数满足：任意两个数a[i]和a[j]，若i<j，必有a[i]<a[j]，这样最长的子序列称为最长递增子序列。

设dp[i]表示以i为结尾的最长递增子序列的长度，则状态转移方程为：dp[i] = max{dp[j]+1}, 1<=j<i,a[j]<a[i].

这样简单的复杂度为O(n^2)，其实还有更好的方法。

考虑两个数a[x]和a[y]，x<y且a[x]<a[y],且dp[x]=dp[y]，当a[t]要选择时，到底取哪一个构成最优的呢？显然选取a[x]更有潜力，因为可能存在a[x]<a[z]<a[y]，这样a[t]可以获得更优的值。在这里给我们一个启示，当dp[t]一样时，尽量选择更小的a[x].

按dp[t]=k来分类，只需保留dp[t]=k的所有a[t]中的最小值，设d[k]记录这个值，d[k]=min{a[t],dp[t]=k}。

这时注意到d的两个特点（重要）：

1. d[k]在计算过程中单调不升；

2. d数组是有序的，d[1]<d[2]<..d[n]。

利用这两个性质，可以很方便的求解：

1. 设当前已求出的最长上升子序列的长度为len（初始时为1），每次读入一个新元素x：

2. 若x>d[len]，则直接加入到d的末尾，且len++；（利用性质2）

否则，在d中二分查找，找到第一个比x小的数d[k]，并d[k+1]=x，在这里x<=d[k+1]一定成立（性质1,2）。

动态规划（O(n^2)）

#include <stdio.h>

#include <string.h>

int s[100005]={0};

int dp[100005]={0};

int main()

{

int n;

while(~scanf("%d", &n)&&n!=EOF)

{

int max=0;

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

{

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

dp[i] = 0;

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

if(s[j]<s[i] && dp[i]<dp[j]) dp[i]=dp[j];

dp[i]++;

if(max<dp[i]) max=dp[i];

}

printf("%d\n", max);

}

return 0;

}

动态规划+二分查找（O(nlogn)）

#include<stdio.h>

#include<string.h>

int dp[100010];

int Binarysearch(int key,int len)

{

int left=0,right=len-1,mid;

while(left<=right)

{

mid=(left+right)/2;

if(key==dp[mid])

return mid;

else if(key>dp[mid])

left=mid+1;

else

right=mid-1;

}

return left;

}

int main()

{

int n,i,j,k,len;

while(scanf("%d",&n)!=EOF)

{

scanf("%d",&k);

dp[0]=k;

len=1;

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

{

scanf("%d",&k);

j=Binarysearch(k,len);

dp[j]=k;

len=j+1>len?j+1:len;

}

printf("%d\n",len);

}

return 0;

}

### 子段和

**子段和：**

给定一整型数列{a1,a2...,an}，找出连续非空子串{ax,ax+1,...,ay}，使得该子序列的和最大，其中，1<=x<=y<=n。

**M段子段和**：

给你一个序列 S1, S2, S3, S4 ... Sx, ... Sn (1 ≤ x ≤ n ≤ 1,000,000, -32768 ≤ Sx ≤ 32767). 我们定义sum(i, j) = Si + ... + Sj (1 ≤ i ≤ j ≤ n).现在给你一个 m（8>m>0&&m<n）你的任务是计算sum(i1, j1) + sum(i2, j2) + sum(i3, j3) + ... + sum(im, jm) ；我们规定他是不相交的。请输出m段最大和，比如：m = 2,n = 6 ,{-1 4 -2 3 -2 4} 它的结果是 9;

#include <cstdio>

#include <cstring>

#define MAXN 1000010

#define INF 0x3f3f3f3f

#define max(a,b) (a>b?a:b)

int ch, ans, f, a[MAXN], dp[MAXN], last[MAXN];

int in()

{

if((ch = getchar()) == '-') f = -1;

else f = 1;

while(ch < '0' || '9' < ch) ch = getchar();

ans = ch-'0';

while((ch = getchar()) >= '0' && '9' >= ch) ans = ans\*10+ch-'0';

return ans\*f;

}

int main()

{

int t;

t = in();

while(t--)

{

int m = in(), n = in(), maxn;

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

a[i] = in();

memset(dp,-0x3f,sizeof(dp));

memset(last,0,sizeof(last));

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

{

maxn = -INF;

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

{

dp[j] = max(dp[j-1], last[j])+a[j];

last[j] = maxn;

maxn = max(maxn,dp[j]);

}

}

printf("%d\n", maxn);

}

return 0;

}

**首尾相连数组的最大子数组和**

给定一个由N个整数元素组成的数组arr，数组中有正数也有负数，这个数组不是一般的数组，其首尾是相连的。数组中一个或多个连续元素可以组成一个子数组，其中存在这样的子数组arr[i],…arr[n-1],arr[0],…,arr[j]，现在请你这个ACM\_Lover用一个最高效的方法帮忙找出所有连续子数组和的最大值（如果数组中的元素全部为负数，则最大和为0，即一个也没有选）。

**最大和**

给定一个由整数组成二维矩阵（r\*c），现在需要找出它的一个子矩阵，使得这个子矩阵内的所有元素之和最大，并把这个子矩阵称为最大子矩阵。

例子：

0 -2 -7 0

9 2 -6 2   
-4 1 -4 1   
-1 8 0 -2   
其最大子矩阵为：

9 2   
-4 1   
-1 8   
其元素总和为15。

#include<iostream>

#include<cstdio>

#include<climits>

#include<cstring>

using namespace std;

const int N = 110;

int row, col;

int matrix[N][N];

int temp[N][N]; ///辅助数组

void DP(void)

{

int thissum, maxsum;

maxsum = INT\_MIN;

for(int i = 1; i <= row; ++i) ///累加求和

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

temp[i][j] = temp[i - 1][j] + matrix[i][j];

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

for(int j = i; j <= row; ++j) ///枚举子矩阵

{

thissum = 0;

for(int k = 1; k <= col; ++k)

{

if(thissum > 0)

thissum += temp[j][k] - temp[i - 1][k];

else

thissum = temp[j][k] - temp[i - 1][k];

maxsum = max(maxsum, thissum);

}

}

printf("%d\n", maxsum);

}

int main(void)

{

int ncase;

scanf("%d", &ncase);

while(ncase--)

{

memset(temp, 0, sizeof(temp));

scanf("%d %d", &row, &col);

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

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

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

DP();

}

return 0;

}

### DAG上的动态规划

**Nyoj 矩形嵌套（一维无序单向搜索）**

有n个矩形，每个矩形可以用a,b来描述，表示长和宽。矩形X(a,b)可以嵌套在矩形Y(c,d)中当且仅当a<c,b<d或者b<c,a<d（相当于旋转X90度）。例如（1,5）可以嵌套在（6,2）内，但不能嵌套在（3,4）中。你的任务是选出尽可能多的矩形排成一行，使得除最后一个外，每一个矩形都可以嵌套在下一个矩形内。

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef struct Rec{

int a, b;

}Rec;

bool cmp(Rec r1, Rec r2)

{

if(r1.a<r2.a) return 1;

else if(r1.a==r2.a && r1.b<r2.b) return 1;

return 0;

}

Rec r[1005];

int d[1005], n;

int dfs(int i){

if(d[i] > 0) return d[i];

d[i] = 1;

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

{

int te = dfs(j)+1;

if(r[i].a<r[j].a && r[i].b<r[j].b){

if(d[i] < te) d[i] = te;

}

}

return d[i];

}

int main()

{

int t;

scanf("%d", &t);

while(t--)

{

//memset(G, 0, sizeof(G));

memset(d, 0, sizeof(d));

scanf("%d", &n);

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

int x, y;

scanf("%d%d", &x, &y);

r[i].a = x<y?x:y;

r[i].b = x>y?x:y;

}

sort(r, r+n, cmp);

int maxn=0;

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

{

int tem = dfs(i);

if(tem > maxn) maxn = tem;

}

printf("%d\n", maxn);

}

return 0;

}

**Nyoj 5 skiing （DAG上的最短路（二维有序四个方向搜索））**

Michael喜欢滑雪百这并不奇怪， 因为滑雪的确很刺激。可是为了获得速度，滑的区域必须向下倾斜，而且当你滑到坡底，你不得不再次走上坡或者等待升降机来载你。Michael想知道载一个区域中最长底滑坡。区域由一个二维数组给出。数组的每个数字代表点的高度。下面是一个例子

2 3 4 5

16 17 18 19 6

15 24 25 20 7

14 23 22 21 8

13 12 11 10 9

一个人可以从某个点滑向上下左右相邻四个点之一，当且仅当高度减小。在上面的例子中，一条可滑行的滑坡为24-17-16-1。当然25-24-23-...-3-2-1更长。事实上，这是最长的一条。

#include <stdio.h>

#include <string.h>

#include <algorithm>

using namespace std;

int Map[1005][1005];

int dp[1005][1005], m, n;

int dfs(int x, int y){

int ans = 1;

if(dp[x][y] > 1) return dp[x][y]; ///如果有当前最长，则直接返回

if(x-1>=0 && Map[x][y]>Map[x-1][y]){

int tem = dfs(x-1, y) + 1;

if(ans < tem) ans = tem;

}

if(x+1<m && Map[x][y]>Map[x+1][y]){

int tem = dfs(x+1, y) + 1;

if(ans < tem) ans = tem;

}

if(y-1>=0 && Map[x][y]>Map[x][y-1]){

int tem = dfs(x, y-1) + 1;

if(ans < tem) ans = tem;

}

if(y+1<n && Map[x][y]>Map[x][y+1]){

int tem = dfs(x, y+1) + 1;

if(ans < tem) ans = tem;

}

dp[x][y] = ans;

return ans;

}

int main()

{

int t;

scanf("%d", &t);

while(t--)

{

scanf("%d%d", &m, &n);

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

{

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

{

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

dp[i][j] = 1;

}

}

int maxn=0;

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

{

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

int tem = dfs(i, j);

if(tem > maxn) maxn = tem;

}

}

printf("%d\n", maxn);

}

return 0;

}

### 区间DP

定义： 区间DP是一类在区间上进行动态规划的最优问题，一般是根据问题设出一个表示状态的dp，可以是二维的也可以是三维的，一般情况下为二维。然后将问题划 分成两个子问题，也就是一段区间分成左右两个区间，然后将左右两个区间合并到整个区间，或者说局部最优解合并为全局最优解，然后得解。

解题步骤：（对应三个for循环）

1）枚举区间长度；

2）枚举左右端点

3）状态转移（分成左右两个区间，然后将左右两个区间合并到整个区间）

**例：Nyoj 石子合并（一）（对于任意的 i，j，合并Union(i, j) >?= Union(i,k)+Union(k+1,j)+cost(I,j)）**

有N堆石子排成一排，每堆石子有一定的数量。现要将N堆石子并成为一堆。合并的过程只能每次将相邻的两堆石子堆成一堆，每次合并花费的代价为这两堆石子的和，经过N-1次合并后成为一堆。求出总的代价最小值。

分析：

1）枚举区间长度：要求n个石子归并，我们根据dp的思想划分成子问题，先求出每两个合并的最小代价，然后每三个的最小代价，依次知道n个。

2）枚举左右端点：定义状态dp [ i ] [ j ]为从第i个石子到第j个石子的合并最小代价。

3）那么dp [ i ] [ j ] = min(dp [ i ] [ k ] + dp [ k+1 ] [ j ])

那么我们就可以从小到大依次枚举让石子合并，直到所有的石子都合并。

#include <cstdio>

using namespace std;

int main()

{

int n;

while(scanf("%d", &n)!=EOF)

{

int a[205]={0}, sum[205] = {0};

int m[205][205] = {0};

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

{

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

sum[i] = sum[i-1]+a[i];

}

for(int r = 2; r <= n; r++) ///枚举区间长度

{

for(int i = 1; i <= n-r+1; i++) ///枚举起点位置

{

int j = i + r - 1; ///终点

m[i][j] = m[i+1][j] + sum[j]-sum[i-1];

for(int k = i+1; k<j; k++) ///状态转移

{

int t = m[i][k] + m[k+1][j] + sum[j]-sum[i-1];

if(t < m[i][j]) m[i][j] = t;

}

}

}

printf("%d\n", m[1][n]);

}

return 0;

}

**Nyoj 110 剑客决斗（对于当前的x， 拆分成两个，分别放在首尾， 对于任意的 i， j， 决斗vs(i, j) = {vs(I,k)==1 || vs(k, j)==1}?1:0）**

在路易十三和红衣主教黎塞留当权的时代，发生了一场决斗。n个人站成一个圈，依次抽签。抽中的人和他右边的人决斗，负者出圈。这场决斗的最终结果关键取决于决斗的顺序。现书籍任意两决斗中谁能胜出的信息，但“A赢了B”这种关系没有传递性。例如，A比B强，B比C强，C比A强。如果A和B先决斗，C最终会赢，但如果B和C决斗在先，则最后A会赢。显然，他们三人中的第一场决斗直接影响最终结果。

假设现在n个人围成一个圈，按顺序编上编号1~n。一共进行n-1场决斗。第一场，其中一人（设i号）和他右边的人（即i+1号，若i=n，其右边人则为1号）。负者被淘汰出圈外，由他旁边的人补上他的位置。已知n个人之间的强弱关系（即任意两个人之间输赢关系）。如果存在一种抽签方式使第k个人可能胜出，则我们说第k人有可能胜出，我们的任务是根据n个人的强弱关系，判断可能胜出的人数。

#include <stdio.h>

#include <string.h>

#define N 505

bool beat[N][N],meet[N][N];

int main()

{

int z,n;

scanf("%d",&z);

while(z--)

{

memset(meet,false,sizeof(meet));

scanf("%d",&n);

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

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

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

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

meet[i][i+1] = true;

meet[n-1][0] = true;

for(int d=2; d<=n; d++) ///枚举区间长度 ，长度必须到n，否则遇不到自己

{

for(int i=0; i<n; i++) ///枚举起点位置

{

int j = i+d; ///终点

for(int k=i+1; k<j; k++) ///状态转移

{

if(meet[i][k%n] && meet[k%n][j%n] && (beat[i][k%n] || beat[j%n][k%n]))

{

meet[i][j%n] = true;

break;

}

}

}

}

int ans = 0;

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

if(meet[i][i])

++ans;

printf("%d\n",ans);

}

return 0;

}

**NYOJ 304 节能**

Dr.Kong设计的机器人卡多越来越聪明。最近市政公司交给卡多一项任务，每天早晨5：00开始，它负责关掉ZK大道右侧上所有的路灯。

卡多每到早晨5：00准会在ZK大道上某盏路灯的旁边，然后他开始关灯。每盏灯都有一定的功率，机器人卡多有着自觉的节能意识，它希望在关灯期间，ZK大道右侧上所有路灯的耗电量总数是最少的。

机器人卡多以1m/s的速度行走。假设关灯动作不需要花费额外的时间，因为当**它通过某盏路灯时就顺手将灯关掉**。

请你编写程序，计算在给定路灯设置，灯泡功率以及机器人卡多的起始位置的情况下，卡多关灯期间，ZK大道上所有灯耗费的最小能量。

输入

有多组测试数据，以EOF为输入结束的标志  
每组测试数据第一行： N 表示ZK大道右侧路灯的数量 （2≤ N ≤ 1000）   
第二行： V 表示机器人卡多开始关灯的路灯号码。 （1≤V≤N）  
接下来的N行中，每行包含两个用空格隔开的整数D和W，用来描述每盏灯的参数  
D表示该路灯与ZK大道起点的距离 (用米为单位来表示)，  
W表示灯泡的功率，即每秒该灯泡所消耗的能量数。路灯是按顺序给定的。  
（ 0≤D≤1000， 0≤W≤1000 ）

输出

输出一个整数，即消耗能量之和的最小值。注意结果小于200，000，000

样例输入

4

3

2 2

5 8

6 1

8 7

样例输出

56

**代码：**

#include <stdio.h>

#include <string.h>

#define LEN 1001

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

///DP[i][j][0] 代表已经关闭i到j之间的灯，并且此刻机器人在左i点时，此时已经消耗的最小电能

///DP[i][j][1] 代表已经关闭i到j之间的灯，并且此刻机器人在左j点时，此时已经消耗的最小电能

int DP[LEN][LEN][2];

int bw[LEN][LEN]; ///bw[i][j]表示第i个灯到第j个灯的功率和

int D[LEN];

int W[LEN];

int main()

{

int N,V,tw=0;

int i,j,t;

while(scanf("%d %d",&N,&V)!=EOF)

{

tw = 0;

memset(bw, 0, sizeof(bw));

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

{

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

tw += W[i];

}

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

{

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

{

bw[i][j] = bw[i][j-1] + W[j]; ///压缩

}

}

//初始化

for(i=V-1; i>0; --i) ///起点左边

{

DP[i][V][0] = DP[i+1][V][0]+(tw-bw[i+1][V])\*(D[i+1]-D[i]);

DP[i][V][1] = DP[i][V][0] + (tw-bw[i][V])\*(D[V]-D[i]);

}

for(j=V+1; j<=N; ++j) ///起点右边

{

DP[V][j][1] = DP[V][j-1][1] + (tw-bw[V][j-1])\*(D[j]-D[j-1]);

DP[V][j][0] = DP[V][j][1] + (tw-bw[V][j])\*(D[j]-D[V]);

}

//DP

for(i=V-1; i>0; i--)

{

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

{

DP[i][j][0] = min(DP[i+1][j][0]+(tw-bw[i+1][j])\*(D[i+1]-D[i]),

DP[i+1][j][1]+(tw-bw[i+1][j])\*(D[j]-D[i]));

DP[i][j][1] = min(DP[i][j-1][0]+(tw-bw[i][j-1])\*(D[j]-D[i]),

DP[i][j-1][1]+(tw-bw[i][j-1])\*(D[j]-D[j-1]));

}

}

printf("%d\n", min(DP[1][N][0], DP[1][N][1]));

}

return 0;

}

### 状态压缩DP

**HDUOJ 4359 郑厂长系列故事——排兵布阵**

一天，郑厂长带着他的军队来到了一个n\*m的平原准备布阵。

　　根据以往的战斗经验，每个士兵可以攻击到并且只能攻击到与之曼哈顿距离为2的位置以及士兵本身所在的位置。当然，一个士兵不能站在另外一个士兵所能攻击到的位置，同时因为地形的原因平原上也不是每一个位置都可以安排士兵。

　　现在，已知n,m 以及平原阵地的具体地形，请你帮助郑厂长计算该阵地,最多能安排多少个士兵。

Input

输入包含多组测试数据；

每组数据的第一行包含2个整数n和m (n <= 100, m <= 10 )，之间用空格隔开；

接下来的n行，每行m个数，表示n\*m的矩形阵地，其中1表示该位置可以安排士兵，0表示该地形不允许安排士兵。

Output

请为每组数据计算并输出最多能安排的士兵数量，每组数据输出一行。

Sample Input

6 6

0 0 0 0 0 0

0 0 0 0 0 0

0 0 1 1 0 0

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

Sample Output

2

代码：

#include <stdio.h>

#include <string.h>

#include <algorithm>

using namespace std;

int map[105][15];

int dp[1<<10][1<<10],tem[1<<10][1<<10];

int now[1<<10],pre[1<<10],prepre[1<<10],ans[1<<10];

int l\_now,l\_pre,l\_prepre;

int n,m;

void dfs(int id,int k,int p,int sum)

{

if(k>=m)

{

now[++l\_now] = p;

ans[l\_now] = sum;

return;

}

if(k>=2 && map[id][k] && !(p&(1<<(k-2))))//这格可以放，而左边第二格不能放，那么在这个位置放下

dfs(id,k+1,p|(1<<k),sum+1);

else if(k<2 && map[id][k])//如果是前两列，这格可以放就先放下

dfs(id,k+1,p|(1<<k),sum+1);

dfs(id,k+1,p,sum);//这格不放

}

void solve()

{

int i,j,k,t;

tem[1][1] = pre[1] = prepre[1] = 0;

l\_now = l\_pre = l\_prepre = 1;

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

{

l\_now = 0;

dfs(k,0,0,0);

for(i = 1; i<=l\_now; i++)

for(j = 1; j<=l\_pre; j++)

dp[i][j] = 0;

for(i = 1; i<=l\_now; i++)

{

for(j = 1; j<=l\_pre; j++)

{

for(t = 1; t<=l\_prepre; t++)

{

if(now[i] & prepre[t]) continue;

if(now[i] & (pre[j]>>1)) continue;

if(now[i] & (pre[j]<<1)) continue;

dp[i][j] = max(dp[i][j],tem[j][t]+ans[i]);

}

}

}

for(i = 1; i<=l\_now; i++)

for(j = 1; j<=l\_pre; j++)

tem[i][j] = dp[i][j];

for(i = 1; i<=l\_pre; i++)

prepre[i] = pre[i];

for(i = 1; i<=l\_now; i++)

pre[i] = now[i];

l\_prepre = l\_pre;

l\_pre = l\_now;

}

}

int main()

{

int i,j;

while(~scanf("%d%d",&n,&m))

{

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

for(j = 0; j<m; j++)

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

solve();

int maxn = 0;

for(i = 1; i<=l\_pre; i++)

for(j = 1; j<=l\_prepre; j++)

maxn = max(maxn,tem[i][j]);

printf("%d\n",maxn);

}

return 0;

}

**NYOJ 81 炮兵阵地**

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

int legal[60];

int num, a[110][15];

int dp[110][60][60];

int map[110];

void Initial(int m)

{

int i, j;

num = 0;

for(i = 0;i < (1<<m);i++)

{

if(i & (i<<1) || i & (i>>1) || i & (i<<2) || i & (i>>2)) //判断二进制有没有相邻的1 或者相隔2位

continue;

legal[num++] = i;

}

}

int get(int i, int j, int k)

{

int sum = 0;

while(j)

{

if(j%2)

sum += a[i][k];

k--;

j/=2;

}

return sum;

}

int main()

{

int n, m, ncase;

scanf("%d", &ncase);

while(ncase--)

{

scanf("%d %d", &n, &m);

int ans = 0;

Initial(m);

memset(dp,0,sizeof(0));

memset(map,0,sizeof(map));

memset(a,0,sizeof(a));

getchar();

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

{

for(int j = 0;j < m;j++)

{

char c;

scanf("%c",&c);

a[i][j] = (c=='P'); ///P赋值为1，H赋值为0

if(a[i][j] == 0) ///如果是H(山地)，

map[i] |= (1<<(m-1-j));

}

getchar();

}

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

{

if(map[0] & legal[i]) continue;

dp[0][i][0] = get(0,legal[i],m-1);

ans = max(ans,dp[0][i][0]);

}

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

{

if(map[1] & legal[i]) continue;

for(int j = 0;j < num;j++)

{

if(map[0] & legal[j]) continue;

if(legal[i] & legal[j]) continue;

dp[1][i][j] = dp[0][j][0] + get(1,legal[i],m-1);

ans = max(ans,dp[1][i][j]);

}

}

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

{

for(int j = 0;j < num;j++)

{

if(map[i] & legal[j]) continue;

for(int l = 0; l < num; l++)

{

if(map[i-1] & legal[l]) continue;

if(legal[j] & legal[l]) continue;

int temp = -1;

for(int k = 0; k<num; k++)

{

if(map[i-2] & legal[k]) continue;

if(legal[j] & legal[k]) continue;

if(legal[k] & legal[l]) continue;

temp = max(temp, dp[i-1][l][k]);

}

dp[i][j][l] = temp + get(i, legal[j], m-1);

ans = max(ans,dp[i][j][l]);

}

}

}

printf("%d\n",ans);

}

}

### 双线DP

对于一个给定的二维地图，求从左上角到右下角两条不相交的路径所获得的价值总和（所需的花费总和）。

探寻宝藏：

传说HMH大沙漠中有一个M\*N迷宫，里面藏有许多宝物。某天，Dr.Kong找到了迷宫的地图，他发现迷宫内处处有宝物，最珍贵的宝物就藏在右下角，迷宫的进出口在左上角。当然，迷宫中的通路不是平坦的，到处都是陷阱。Dr.Kong决定让他的机器人卡多去探险。

但机器人卡多从左上角走到右下角时，只会向下走或者向右走。从右下角往回走到左上角时，只会向上走或者向左走，而且卡多不走回头路。（即：一个点最多经过一次）。当然卡多顺手也拿走沿路的每个宝物。

Dr.Kong希望他的机器人卡多尽量多地带出宝物。请你编写程序，帮助Dr.Kong计算一下，卡多最多能带出多少宝物。

#include <stdio.h>

int n,m;

int min(int a,int b){return a<b?a:b;}

int max(int a,int b){return a>b?a:b;}

int Map[51][51];

int f[102][51][51];

int main(){

int t;

scanf("%d", &t);

while(t--)

{

scanf("%d%d",&n,&m);

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

{

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

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

}

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

{

for (int x1=1;x1<=min(i,n);x1++)

{

int y1=i-x1+1;

if(y1>m) continue;

for (int x2=x1+1;x2<=min(i,n);x2++)

{

int y2=i-x2+1;

int temp=0;

temp=max(f[i-1][x1][x2],f[i-1][x1-1][x2-1]);

temp=max(temp,f[i-1][x1-1][x2]);

temp=max(temp,f[i-1][x1][x2-1]);

f[i][x1][x2]=temp+Map[x1][y1]+Map[x2][y2];

}

}

}

int ans = 0;

if(ans < f[m+n-2][n-1][n]) ans = f[m+n-2][n-1][n];

if(ans < f[m+n-2][n][n-1]) ans = f[m+n-2][n][n-1];

printf("%d\n",ans+Map[n][m]);

}

return 0;

}

### 背包问题(见背包九讲)

## 数据结构

### 并查集

**NYOJ 608 畅通工程**

某省调查城镇交通状况，得到现有城镇道路统计表，表中列出了每条道路直接连通的城镇。省政府“畅通工程”的目标是使全省任何两个城镇间都可以实现交通（但不一定有直接的道路相连，只要互相间接通过道路可达即可）。问最少还需要建设多少条道路？

输入

测试输入包含若干测试用例。每个测试用例的第1行给出两个正整数，分别是城镇数目N ( < 1000 )和道路数目M；随后的M行对应M条道路，每行给出一对正整数，分别是该条道路直接连通的两个城镇的编号。为简单起见，城镇从1到N编号。   
注意:两个城市之间可以有多条道路相通,也就是说  
3 3  
1 2  
1 2  
2 1  
这种输入也是合法的  
当N为0时，输入结束，该用例不被处理。

输出

对每个测试用例，在1行里输出最少还需要建设的道路数目。

样例输入

4 2

1 3

4 3

3 3

1 2

1 3

2 3

5 2

1 2

3 5

999 0

0

样例输出

1

0

2

998

代码：

#include <cstdio>

int Father[1005];

int count;

int Find(int u)

{

if(Father[u] != u) Father[u] = Find(Father[u]);

return Father[u];

}

void Union(int u, int v)

{

u = Find(u);

v = Find(v);

if(u != v) {

Father[v] = u;

count++;

}

}

int main()

{

int m, n;

while(scanf("%d", &m), m)

{

scanf("%d", &n);

count = 0;

for(int i=0; i<1005; i++)

{

Father[i] = i;

}

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

{

int u, v;

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

Union(u, v);

}

printf("%d\n", m-count-1);

}

return 0;

}

### 树状数组

**NYOJ 士兵杀敌（二）**

南将军手下有N个士兵，分别编号1到N，这些士兵的杀敌数都是已知的。小工是南将军手下的军师，南将军经常想知道第m号到第n号士兵的总杀敌数，请你帮助小工来回答南将军吧。南将军的某次询问之后士兵i可能又杀敌q人，之后南将军再询问的时候，需要考虑到新增的杀敌数。

输入

只有一组测试数据

第一行是两个整数N,M，其中N表示士兵的个数(1<N<1000000)，M表示指令的条数。(1<M<100000)

随后的一行是N个整数，ai表示第i号士兵杀敌数目。(0<=ai<=100)

随后的M行每行是一条指令，这条指令包含了一个字符串和两个整数，首先是一个字符串，如果是字符串QUERY则表示南将军进行了查询操作，后面的两个整数 m,n，表示查询的起始与终止士兵编号；如果是字符串ADD则后面跟的两个整数I,A(1<=I<=N,1<=A<=100), 表示第I个士兵新增杀敌数为A.

输出

对于每次查询，输出一个整数R表示第m号士兵到第n号士兵的总杀敌数，每组输出占一行

样例输入

5 6

1 2 3 4 5

QUERY 1 3

ADD 1 2

QUERY 1 3

ADD 2 3

QUERY 1 2

QUERY 1 5

样例输出

6

8

8

20

代码：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define N 1000050

int num[N] = {0}; // 数据数组

int c[N] = {0}; // 树状数组

int Lowbit(int t) // 取出一个数二进制中的 最后一个1及后面的0； 比如 lowbit(8) = 8, lowbit(12) = 4

{

return t & (-t);

}

int Build(int Size) // 创建树状数组

{

for(int i = 1; i <= Size; i ++){ // 树状数组元素下标

for(int j = (i - Lowbit(i) + 1); j <= i; j++){ // 该 树状数组元素 所包含的子元素和

c[i] += num[j];

}

}

return 0;

}

void Add(int i, int v, int Size) // i元素增大v

{

while(i <= Size){

c[i] += v;

i += Lowbit(i); // i元素增加后，后面与其相关的 数组数组元素值 也相应的增加

}

}

int Sum(int n) // 求前n项和

{

int s = 0;

while(n > 0){

s += c[n];

n -= Lowbit(n); // 求在 树状数组 上的上一个有关元素下标

}

return s;

}

int main()

{

char order[6];

int soldier\_num, order\_num;

int start, end;

scanf("%d%d", &soldier\_num, &order\_num);

for(int i = 1; i <= soldier\_num; i ++){

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

}

Build(soldier\_num); // 创建树状数组

while(order\_num --){

scanf(" %s%d%d", order, &start, &end);

if( !strcmp(order, "QUERY") ){ // 求区域和

printf("%d\n", ( Sum(end) - Sum(start - 1) )); // 两端相减

}

else{

Add(start, end, soldier\_num); // 某个士兵杀敌数增加

}

}

return 0;

}

### （字符串）KMP匹配

**例：对于一个模式串p，求其在源串s中出现的次数，p<=10,s<=1000**

#include <stdio.h>

#include <string.h>

char s[1020],p[20];

int next[1020], sl, pl;

void get\_next()

{

int k=-1,j=0;

next[0]=-1;

while(j<pl)

{

if(k==-1||p[j]==p[k])

{

j++;k++;

next[j]=k;

}

else

k=next[k];

}

}

int KMP()

{

int i=0, j=0, sum = 0;

get\_next();

while(j<pl&&i<sl)

{

if(j==-1||s[i]==p[j])

{

i++;j++;

}

else j=next[j];

if(j==pl)

{

sum++;j=next[j];

}

}

return sum;

}

int main(){

int n;

scanf("%d",&n);

while(n--)

{

scanf("%s%s", p, s);

pl = strlen(p);

sl = strlen(s);

printf("%d\n",KMP());

}

return 0;

}

**例：NYOJ 327 亲和串**

最近zyc遇到了一个很棘手的问题：判断亲和串，以前判断亲和串的时候直接可以看出来，但现在不同了，现在给出的两字符串都非常的大，看的zyc头都晕 了。于是zyc希望大家能帮他想一个办法来快速判断亲和串。亲和串定义：给定两个字符串s1和s2，如果能通过s1循环移动，使s2包含在s1中，那么我 们就说s2是s1的亲和串。

输入

本题有多组测试数据，每组数据的第一行包含输入字符串s1,第二行包含输入字符串s2，s1与s2的长度均小于100000。

输出

如果s2是s1的亲和串，则输出"yes"，反之，输出"no"。每组测试的输出占一行。

样例输入

AABCD

CDAA

ASD

ASDF

样例输出

yes

no

CODE：

#include <stdio.h>

#include <string.h>

#define maxn 100000 + 5

char str1[2 \* maxn], str2[maxn];

int next[maxn], len1, len2;

void getNext(){

int j = -1, i = 0;

next[0] = -1;

while(i < len2){

if(j == -1 || str2[i] == str2[j]){

++i; ++j;

if(str2[i] != str2[j]) next[i] = j;

else next[i] = next[j];

}else j = next[j];

}

}

bool KMP(){

getNext();

int i = 0, j = 0;

while(i < len1 && j < len2){

if(j == -1 || str1[i] == str2[j]) ++j, ++i;

else j = next[j];

}

return j == len2;

}

int main(){

while(scanf("%s%s", str1, str2) == 2){

len1 = strlen(str1);

len2 = strlen(str2);

if(len1 < len2){

printf("no\n");

continue;

}

memcpy(str1 + len1, str1, len1);

len1 \*= 2;

if(KMP()) printf("yes\n");

else printf("no\n");

}

return 0;

}

## 最小生成树算法

### Prime核心算法

#define INF 0x7fffffff

#define N 110

int Map[N][N],dist[N];

bool vis[N];

///当前位置cur，一共有m个点，数据从1开始存储

void prime(int cur, int m)

{

int sum = 0;

memset(vis, false, sizeof(vis)); ///初始化vis

vis[cur] = true;

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

dist[i] = Map[cur][i];

}

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

{

int mincost = INF, index;

for(int j = 1; j <= m; j++)///从1开始存储

{

if(!vis[j] && dist[j] < mincost){

mincost = dist[j];

index = j;

}

}

vis[index] = true;

sum += mincost;

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

{

if(!vis[j] && Map[index][j] < dist[j]){

dist[j] = Map[index][j];

}

}

}

printf("%d\n", sum);

}

**例：nyoj筹建工程：**

省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通(但不一定有直接的公路相连,只要能间接通过公路可达即可)。经过调查评估,得到的统计表中列出了有可能建设公路的若干条道路的成本(道路是双向的)。现请你编写程序,计算出全省畅通需要的最低成本。

输入

测试输入包含若干测试用例。第一行一个整数 T (T <= 5) 表示测试用例数量,每个测试用例的第 1 行给出评估的道路条数 N、村庄数目 M ( 1 <= M < 100,0 <= N <= M \*( M-1) /2),随后的 N 行对应村庄间道路的成本,每行给出三个正整数,依次是两个村庄的编号(每对村庄至多出现一次),以及此两村庄间道路的成本( 也是正整数 )。为简单起见,村庄从 1 到 M 编号。

输出

对每个测试用例,在 1 行里输出全省畅通需要的最低成本。若统计数据不足以保证畅通,则输出 No solution。

#include <stdio.h>

#include <string.h>

#define INF 0x3f3f3f3f

#define N 110

int Map[N][N],dist[N];

bool vis[N];

void prime(int cur, int m)///当前位置cur， 一共有m个点

{

int sum = 0;

memset(vis, false, sizeof(vis)); ///初始化vis

vis[cur] = true;

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

dist[i] = Map[cur][i];

}

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

{

int mincost = INF, index;

for(int j = 1; j <= m; j++)///从1开始存储

{

if(!vis[j] && dist[j] < mincost){

mincost = dist[j];

index = j;

}

}

if(mincost==INF){

printf("No solution\n");

return;

}

vis[index] = true;

sum += mincost;

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

{

if(!vis[j] && Map[index][j] < dist[j]){

dist[j] = Map[index][j];

}

}

}

printf("%d\n", sum);

}

int main()

{

int t;

scanf("%d", &t);

while(t--)

{

int m, n;

scanf("%d%d", &n, &m);

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

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

if(i==j) Map[i][j] = 0;

else Map[i][j] = INF;

}

}

int u, v, b;

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

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

Map[u][v] = b;

Map[v][u] = b;

}

prime(1,m);

}

return 0;

}

### Kruskal算法

#include <stdio.h>

#include <queue>

#define N 100

using namespace std;

struct Node{

int start;

int end;

int price;

friend bool operator < (const Node& a, const Node& b){

return a.price > b.price;

}

};

priority\_queue<Node>q;

int v, l; // v为顶点数, l为边数

int father[N];

int get\_father(int cur)

{

return cur == father[cur] ? cur : father[cur] = get\_father(father[cur]); // 路径压缩

}

int join(int start, int end)

{

int root1 = get\_father(start);

int root2 = get\_father(end);

if(root1 == root2)

return 0;

father[root1] = root2;

return 1;

}

int kruskal()

{

int ans = 0;

Node cur;

while(!q.empty()){ // 此处可记录边数，以优化

cur = q.top();

q.pop();

if(join(cur.start, cur.end)){

printf("%d -> %d\n", cur.start, cur.end); // 输出路径

ans += cur.price;

}

}

return ans;

}

int main()

{

int i;

Node a;

scanf("%d%d", &v, &l);

for(i = 0; i < v; i ++){

father[i] = i;

}

for(i = 0; i < l; i ++){

scanf("%d%d%d", &a.start, &a.end, &a.price);

q.push(a);

}

printf("\nRoad: \n");

int min = kruskal(); // 最小生成树

printf("LowLoad: %d\n", min);

printf("\nFather: \n"); // 输出father数组

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

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

printf("\n");

return 0;

}

**例：HDUOJ 1233 还是畅通工程**

某省调查乡村交通状况，得到的统计表中列出了任意两村庄间的距离。省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要能间接通过公路可达即可），并要求铺设的公路总长度为最小。请计算最小的公路总长度。

Input

测试输入包含若干测试用例。每个测试用例的第1行给出村庄数目N ( < 100 )；随后的N(N-1)/2行对应村庄间的距离，每行给出一对正整数，分别是两个村庄的编号，以及此两村庄间的距离。为简单起见，村庄从1到N编号。  
当N为0时，输入结束，该用例不被处理。

Output

对每个测试用例，在1行里输出最小的公路总长度。

Sample Input

3

1 2 1

1 3 2

2 3 4

4

1 2 1

1 3 4

1 4 1

2 3 3

2 4 2

3 4 5

0

Sample Output

3

5

代码：

#include <stdio.h>

#include <queue>

#define N 110

using namespace std;

struct Node{

int start;

int end;

int len;

friend bool operator < (const Node& a, const Node& b){

return a.len > b.len;

}

};

int n;

priority\_queue<Node>q;

int Father[N];

void Init(){

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

Father[i] = i;

}

}

int GetFather(int cur){

return Father[cur] == cur ? cur : Father[cur] = GetFather(Father[cur]);

}

bool Join(int start, int end){

int root1 = GetFather(start);

int root2 = GetFather(end);

if(root1 == root2){

return 0;

}

else{

Father[root1] = root2;

return 1;

}

}

int Kruskal(){

int ans = 0;

while(!q.empty()){

Node cur = q.top();

q.pop();

if(Join(cur.start, cur.end)){

ans += cur.len;

}

}

return ans;

}

int main()

{

Node nw;

int start, end, len;

while(scanf("%d", &n), n){

Init();

for(int i = 0; i < n \* (n - 1) / 2; i ++){

scanf("%d%d%d", &start, &end, &len);

nw.start = start;

nw.end = end;

nw.len = len;

q.push(nw);

}

int ans = Kruskal();

printf("%d\n", ans);

}

return 0;

}

## 单源最短路径

### Dijkstra核心算法

#define INF 0x7fffffff

#define N 110

int Map[N][N],dist[N];

bool vis[N];

void Dijkstra(int s, int N) ///从原点s到其他N-1个点的距离

{

memset(vis,false,sizeof(vis));///初始化

vis[s]=true;///s到各点距离，所以s已经被访问

for(int i=1; i<=N; i++) ///初始化距离数组

dis[i]=Map[s][i];

for(int i=1; i<=N-1; i++) ///迪杰斯特拉核心语句

{

int minn = INF,u;///辅助变量

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

{

if(vis[j]==0&&dis[j]<minn)

{

minn=dis[j];

u=j;

}

}

vis[u]=true///为已访问

for(int v=1; v<=N; v++)

{

if(Map[u][v]<INF)

{

if(dis[v]>dis[u]+Map[u][v])

dis[v]=dis[u]+Map[u][v];///松弛完成

}

}

}

}

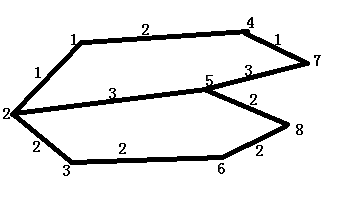
**城市平乱：**

南将军统领着N个部队，这N个部队分别驻扎在N个不同的城市。

他在用这N个部队维护着M个城市的治安，这M个城市分别编号从1到M。

现在，小工军师告诉南将军，第K号城市发生了暴乱，南将军从各个部队都派遣了一个分队沿最近路去往暴乱城市平乱。

现在已知在任意两个城市之间的路行军所需的时间，你作为南将军麾下最厉害的程序员，请你编写一个程序来告诉南将军第一个分队到达叛乱城市所需的时间。



注意，两个城市之间可能不只一条路。

分析：最终都要到达暴乱城市，所以可以以暴乱城市为起点进行搜索。

#include<string.h>

#include<stdio.h>

#define INF 9999999

using namespace std;

int Map[1005][1005];///存储地图数据

int dis[1005];///存储距离

int vis[1005];///已走过的点标记为1，初始化为0

void Dijsktra(int s, int N)

{

for(int i=1; i<=N; i++) ///初始化距离数组

dis[i]=Map[s][i];

vis[s]=1;///s到各点距离，所以s已经被访问

int minn,u;///辅助变量

for(int i=1; i<=N-1; i++) ///迪杰斯特拉核心语句

{

minn=INF;///找到距离1最近的顶点

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

{

if(vis[j]==0&&dis[j]<minn)

{

minn=dis[j];

u=j;

}

}

vis[u]=1;///标记为已访问

for(int v=1; v<=N; v++)

{

if(Map[u][v]<INF)

{

if(dis[v]>dis[u]+Map[u][v])

dis[v]=dis[u]+Map[u][v];///松弛完成

}

}

}

}

int main()

{

int K;

scanf("%d",&K);

while(K--)

{

int n, m, p, q;

int army[105];

scanf("%d%d%d%d", &n, &m, &p, &q);

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

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

for(int i=0; i<1005; i++) ///初始化

{

for(int j=0; j<1005; j++)

{

if(i==j) Map[i][j] = 0;

else Map[i][j] = INF;

}

}

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

{

int a, b, t;

scanf("%d%d%d", &a, &b, &t);

if(t<Map[a][b]) {

Map[a][b] = t;

Map[b][a] = t;

}

}

memset(vis,0,sizeof(vis));///已访问的标记为1

Dijsktra(q, m);

int minx = INF;

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

{

if(dis[army[i]]<minx) minx = dis[army[i]];

}

printf("%d\n", minx);

}

return 0;

}

### Bellman\_Ford算法

Bellman－Ford算法可以大致分为三个部分

第一，初始化所有点。每一个点保存一个值，表示从原点到达这个点的距离，将原点的值设为0，其它的点的值设为无穷大（表示不可达）。

第二，进行循环，循环下标为从1到n－1（n等于图中点的个数）。在循环内部，遍历所有的边，进行松弛计算。

第三，遍历途中所有的边（edge（u，v）），判断是否存在这样情况：

d（v） > d (u) + w(u,v)

则返回false，表示途中存在从源点可达的权为负的回路。

///边

typedef struct Edge{

int u, v; ///起点， 终点

int weight; ///权值

}Edge;

Edge edge[2\*maxn]; ///双向边，保存边的权值

int dist[maxn]; ///节点到原点的最小距离

int edgenum; ///边数

///插入边

void insert(int u, int v, int w)

{

edge[edgenum].u = u;

edge[edgenum].v = v;

edge[edgenum++].weight = w;

}

bool Bellman\_Ford(int source, int nodenum) ///原点和结点个数

{

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

dist[i] = INF;

dist[source] = 0;

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

for(int j=0; j < edgenum; ++j)

{

if(dist[edge[j].v] > dist[edge[j].u] + edge[j].weight)//松弛计算,求最小,求最大，用<

dist[edge[j].v] = dist[edge[j].u] + edge[j].weight;

}

}

bool flag = false;

/// 判断是否有负权环

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

if(dist[edge[i].v] > dist[edge[i].u] + edge[i].weight)

{

flag = true; ///有负权环

break;

}

}

return flag;

}

**例：赚钱啦**

某国家里有N个城市，分别编号为0~N-1，一个精明的商人准备从0号城市旅行到N-1号城市，在旅行的过程中，从一个城市移动到另外一个城市需要有一定 的花费，并且从A城市移动到B城市的花费和B城市移动到A城市的花费相同，但是，从A城市移动到B城市能赚取的钱和从B城市移动到A城市赚的钱不一定相 同。现在，已知各个城市之间移动的花费和城市之间交易可赚取的金钱，求该商人在从0号城市移动到N-1号城市的过程中最多能赚取多少钱？

输入

第一行是一个整数T(T<=10)表示测试数据的组数，

每组测试数据的第一行是两个整数N,M表示，共有N个城市(1<N<=1000),M条路(1<=M<=1000)  
随后的M行，每行有5个正整数，前两个数a,b(0<=a,b<N)表示两个城市的编号。后面的三个数c,u,v分别表示在a,b城市之间移动的花费，a城市移动到b城市可赚取的资金，b城市移动到a城市可赚取的资金。(0<=c,u,v<=1000)

输出

如果商人能够在旅行过程中赚取无限多的资金，则输出$$$否则输出他在移动过程中最多能赚取的资金数量如果只会赔钱的话就输出一个负数，表示最少赔的钱数。

分析：这个题和上面Bellman\_ford标准的求最短路正好相反，求得赚取的最小花费，所以初始化和松弛计算都有一点变化，dist[]应该初始化为负无穷，松弛计算应该取较大值。

#include <cstdio>

#define maxn 1002

#define INF -0x3fffffff

///边

typedef struct Edge{

int u, v; ///起点， 终点

int weight; ///权值

}Edge;

Edge edge[2\*maxn]; ///双向边，保存边的权值

int dist[maxn]; ///节点到原点的最小距离

int edgenum; ///边数

///插入边

void insert(int u, int v, int w)

{

edge[edgenum].u = u;

edge[edgenum].v = v;

edge[edgenum++].weight = w;

}

bool Bellman\_Ford(int source, int nodenum) ///原点和结点个数

{

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

dist[i] = INF;

dist[source] = 0;

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

for(int j=0; j < edgenum; ++j)

{

if(dist[edge[j].v] < dist[edge[j].u] + edge[j].weight)///松弛计算

dist[edge[j].v] = dist[edge[j].u] + edge[j].weight;

}

}

bool flag = false;

/// 判断是否有负权环

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

if(dist[edge[i].v] < dist[edge[i].u] + edge[i].weight)

{

flag = true; ///有负权环

break;

}

}

return flag;

}

int main()

{

int T;

scanf("%d", &T);

while(T--)

{

int n, m, flag=0;

scanf("%d%d", &n, &m);

edgenum = 0;

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

{

int a, b, c, u, v;

scanf("%d%d%d%d%d", &a, &b, &c, &u, &v);

if((u-c>0 && v-c >=0)||(u-c>=0 && v-c >0))

{

flag = 1;

continue;

}

insert(a, b, u-c);

insert(b, a, v-c);

}

if(flag == 1 || Bellman\_Ford(0, n))

printf("$$$\n");

else

printf("%d\n", dist[n-1]);

}

return 0;

}

### SPFA算法（Bellman\_Ford的队列实现）

**HDUOJ 1874 畅通工程续**

某省自从实行了很多年的畅通工程计划后，终于修建了很多路。不过路多了也不好，每次要从一个城镇到另一个城镇时，都有许多种道路方案可以选择，而某些方案要比另一些方案行走的距离要短很多。这让行人很困扰。现在，已知起点和终点，请你计算出要从起点到终点，最短需要行走多少距离。

Input

本题目包含多组数据，请处理到文件结束。  
每组数据第一行包含两个正整数N和M(0<N<200,0<M<1000)，分别代表现有城镇的数目和已修建的道路的数目。城镇分别以0～N-1编号。  
接下来是M行道路信息。每一行有三个整数A,B,X(0<=A,B<N,A!=B,0<X<10000),表示城镇A和城镇B之间有一条长度为X的双向道路。  
再接下一行有两个整数S,T(0<=S,T<N)，分别代表起点和终点。

Output

对于每组数据，请在一行里输出最短需要行走的距离。如果不存在从S到T的路线，就输出-1.

Sample Input

3 3

0 1 1

0 2 3

1 2 1

0 2

3 1

0 1 1

1 2

Sample Output

2

-1

代码：

#include <stdio.h>

#include <string.h>

#include <queue>

#define N 220

using namespace std;

int CityNum, RoadNum;

int map[N][N], dis[N];

bool vis[N];

void Init(){

memset(vis, 0, sizeof(vis));

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

dis[i] = 0xfffffff;

for(int j = 0; j < CityNum; j ++){

map[i][j] = 0xfffffff;

}

}

}

void SPFA(int start){

queue<int>q;

q.push(start);

dis[start] = 0;

while(!q.empty()){

int cur = q.front();

q.pop();

vis[cur] = 0;

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

if(dis[i] > map[cur][i] + dis[cur]){

dis[i] = map[cur][i] + dis[cur];

if(!vis[i]){

q.push(i);

vis[i] = 1;

}

}

}

}

}

int main()

{

int start, end, len;

while(scanf("%d%d", &CityNum, & RoadNum) != EOF){

Init();

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

scanf("%d%d%d", &start, &end, &len);

if(map[start][end] > len){

map[start][end] = map[end][start] = len;

}

}

scanf("%d%d", &start, &end);

SPFA(start);

if(dis[end] == 0xfffffff){

printf("-1\n");

}

else{

printf("%d\n", dis[end]);

}

}

return 0;

}

## 二分图匹配

（1）在二分图中求最少的点，让每条边都至少和其中的一个点关联，这就是

二分图的“最小顶点覆盖”。

**二分图的最小顶点覆盖数 　=　二分图的最大匹配数**

（2）用尽量少的不相交简单路径覆盖有向无环图(DAG)G的所有顶点，这就是DAG图的最小路径覆盖问题。

**DAG图的最小路径覆盖数　=　节点数（n）- 最大匹配数（m）**

（3）独立集指的是两两顶点之间没有边的顶点集合。顶点最多的独立集就是最大独

立集。

**二分图的最大独立集数　=　节点数（n）-　最大匹配数（m）**

**最大独立集＝补图的最大团**

(4) 图的团就是一个两两之间有边的顶点集合。如果一个团不被其他任一团所包含，即它不是其他任一团的真子集，则称该团为图G的极大团。顶点最多的极大团，称之为图G的最大团。

**最大团＝补图的最大独立集**

### 匈牙利算法

题意：奶牛们只愿意在特定的某几个圈中产奶，一个圈只容一只奶牛，问你最多能有几只奶牛产奶？

思路：基础二分图最大匹配，匈牙利算法实现，

源代码：(412K，0MS)

#include<iostream>

using namespace std;

const int Max = 405;

int n, m, ans;

int link[Max];

bool map[Max][Max], vis[Max]; // map[i][j]表示奶牛i能否匹配圈j。

bool dfs(int u){ // dfs实现，u表示现在在寻求匹配圈的奶牛u。

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

if(!vis[i] && map[u][i]){

vis[i] = true;

if(link[i] == -1 || dfs(link[i])){ // 条件：圈i还没匹配，或者link[i]找到新的匹配。

link[i] = u;

return true;

}

}

return false;

}

int main(){

int i, num, u;

while(cin >> n >> m){

memset(map, 0, sizeof(map));

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

cin >> num;

while(num --){

cin >> u;

map[i][u] = 1;

}

}

ans = 0;

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

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

memset(vis, 0, sizeof(vis));

if(dfs(i)) ans ++;

}

cout << ans << endl;

}

return 0;

}

## 网络流

**例 NYOJ 323：**

在农夫约翰的农场上，每逢下雨，贝茜最喜欢的三叶草地就积聚了一潭水。这意味着草地被水淹没了，并且小草要继续生长还要花相当长一段时间。因此，农夫约翰 修建了一套排水系统来使贝茜的草地免除被大水淹没的烦恼（不用担心，雨水会流向附近的一条小溪）。作为一名一流的技师，农夫约翰已经在每条排水沟的一端安 上了控制器，这样他可以控制流入排水沟的水流量。  
  
农夫约翰知道每一条排水沟每分钟可以流过的水量，和排水系统的准确布局（起点为水潭而终点为小溪的一张网）。需要注意的是，有些时候从一处到另一处不只有一条排水沟。  
  
根据这些信息，计算从水潭排水到小溪的最大流量。对于给出的每条排水沟，雨水只能沿着一个方向流动，注意可能会出现雨水环形流动的情形。  
第1行: 两个用空格分开的整数N (0 <= N <= 200) 和 M (2 <= M <= 200)。N是农夫约翰已经挖好的排水沟的数量，M是排水沟交叉点的数量。交点1是水潭，交点M是小溪。  
第二行到第N+1行: 每行有三个整数，Si, Ei, 和 Ci。Si 和 Ei (1 <= Si, Ei <= M) 指明排水沟两端的交点，雨水从Si 流向Ei。Ci (0 <= Ci <= 10,000,000)是这条排水沟的最大容量。

### SAP算法

在SAP算法中，我们定义每个顶点的距离标号(Distance Labels)，即残留网络中这个点到汇点的距离。然后，我们只在距离标号相邻的点间寻找增广路径。如果从一个点出发没有容许边，就对重标记并回溯（类似预流推进）。

首先介绍一些定义：

残留容量：容量－流量。即，残留容量r[i,j]=c[i,j]-f[i,j]。如果只求流值，则在网络流算法中只需记录残量，而不必记录容量和流量。

残留边、残留网络：有残留容量的边成为残留边，由网络的点集、残留边集构成残留网络。

距离标号 ：对于每个点i，定义其在当前残留网络中到汇点的距离为d[i]。如d[t]=0。

容许边：当且仅当r[i,j]>0且d[i]=d[j]+1时，边(i,j)是容许边。我们的算法只考虑容许边。

算法的一开始，我们从汇点进行一次BFS求出距离标号。然后从源点开始递归操作。对于一个点i的操作，如果存在容许边(i,j)，当j是汇点时增广并从源点开始递归，否则就令i是j的前驱，并递归j。否则对i重标号，使得d[i]=min{d[j]}+1 ( r[i,j] )>0，并回溯。当d[s]>=n时算法停止，此时不存在任何一条增广路。

#include <cstdio>

#include <cstring>

#include <queue>

using namespace std;

const int N = 100010;

const int M = 400010;

const int inf = 1<<30;

struct Edge{

int v, next, w; ///起点、终点、权重

}e[M];

int head[N], cnt;

int n, m, ans;

int s, t;

int pre[N], cur[N], dis[N], gap[N];

int q[N], open, tail;

void addedge(int u,int v,int w)

{

e[cnt].v = v;

e[cnt].w = w;

e[cnt].next = head[u];

head[u] = cnt++;

e[cnt].v = u;

e[cnt].w = 0;

e[cnt].next = head[v];

head[v] = cnt++;

}

void BFS()

{

memset(gap,0,sizeof(gap));

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

open = tail = 0;

q[open] = t;

dis[t] = 0;

while (open <= tail)

{

int u = q[open++];

for(int i=head[u]; i!=-1; i=e[i].next)

{

int v = e[i].v;

if(e[i].w!=0 || dis[v]!=-1) continue;

q[++tail] = v;

++gap[dis[v] = dis[u]+1];

}

}

}

int sap(int n)

{

int v,u,flow = 0,aug = inf;

BFS();

gap[0]=1;

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

cur[i]=head[i];

u=pre[s]=s;

while (dis[s]<n)

{

int flag=0;

for (int j=cur[u]; j!=-1; j=e[j].next)

{

v=e[j].v;

if (e[j].w>0&&dis[u]==dis[v]+1)

{

flag=1;

if (e[j].w<aug) aug=e[j].w;

pre[v] = u;

u = v;

if (u == t)

{

flow += aug;

while (u != s)

{

u = pre[u];

e[cur[u]].w -= aug;

e[cur[u]^1].w += aug;

}

aug = inf;

}

break;

}

cur[u] = e[j].next;

}

if (flag) continue;

int mindis = n;

for (int j=head[u]; j!=-1; j=e[j].next)

{

v = e[j].v;

if (e[j].w>0 && mindis>dis[v])

{

mindis = dis[v];

cur[u] = j;

}

}

if (--gap[dis[u]]==0) break;

++gap[dis[u] = mindis+1];

u = pre[u];

}

return flow;

}

int main()

{

while(~scanf("%d%d", &m, &n))

{

cnt = 0;

s = 1;

t = n;

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

memset(pre,0,sizeof(pre));

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

{

int u, v, w;

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

addedge(u, v, w);

}

//ans = sap(n);

printf("%d\n", sap(n));

}

return 0;

}

### Dinic算法

Dinic 算法的基本步骤为:

1) 计算残余网络的层次图。我们定义 h[i] 为顶点 i 距离源 S 所经过到最小边数，求出所有顶点的 h 值，h[] 值相同的顶点属于同一层，这就是网络的层次图。

2) 在层次图上进行 BFS 增广，直到不存在增广路径。这时求得的增广路径上顶点是分层的，路径上不可能存在两个顶点属于同一层，即 h[i]== h[j] (i!= j )。同时，求得层次图后，我们可以在层次图上进行多次增广。

3) 重复 1 和 2。直到不存在增广路径。

最小割：对于图中的两个点(一般为源点和汇点)来说，如果把图中的一些边去掉，如果它们之间无法连通的话，则这些边组成的集合就叫为割了。如果这些边有权值，最小割就是指权值之和最小的一个割。

最大流最小割：应用于网络中，指总流量不超过链路可承载的最大值，且在每条子路径上取尽可能少的流量。对任意一个只有一个源点一个汇点的图来说，从源点到汇点的最大流等于最小割。

#include <cstdio>

#include <cstring>

#include <queue>

#define MAX 65535

struct node

{

int e;

int w;

int fro;

}eg[400];

int head[400]; //正儿八经的前向星

using namespace std;

int cont;

void add(int s,int e,int w) //加边

{

eg[cont].e = e; //前向弧

eg[cont].w = w;

eg[cont].fro = head[s];

head[s] = cont++;

eg[cont].e = s; //反向弧

eg[cont].w = 0;

eg[cont].fro = head[e];

head[e] = cont++;

return ;

}

int dis[400];

int BFS(int s,int e) //BFS找增光路 并且分层次

{

int now,tmp;

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

queue<int> que;

que.push(s);

dis[s] = 0;

while (!que.empty())

{

now = que.front();

que.pop();

for (int i = head[now];i != -1;i = eg[i].fro)

{

int v = eg[i].e;

if (dis[v] == -1 && eg[i].w > 0)

{

dis[v] = dis[now] + 1; //分层

que.push(v);

}

}

}

if (dis[e] != -1)

return 1;

return 0;

}

int dinic(int s,int e,int t)

{

if (s == e)

return t;

int tmp = t;

for (int i = head[s];i != -1;i = eg[i].fro)

{

int v = eg[i].e;

if(dis[v] == dis[s] + 1 && eg[i].w > 0)

{

int imin = dinic(v,e,min(t,eg[i].w)); //递归找最小容量，其实就是个DFS

eg[i].w -= imin;

eg[i ^ 1].w += imin;

t -= imin;

}

}

return tmp - t;

}

int main()

{

int n,m;

while (~scanf ("%d%d",&n,&m))

{

int i;

cont = 0;

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

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

{

int a,b,c;

scanf ("%d%d%d",&a,&b,&c);

add(a,b,c);

}

int ans = 0;

while (BFS(1,m))

ans += dinic(1,m,MAX);

printf ("%d\n",ans);

}

return 0;

}