ACM Standard Code Library

Zhang Zhipeng

Department of Computer Science and Technology

Harbin Institute of Technology at Weihai

May 23, 2013

Contents

Predefine && 精度模板

```
/*-------------------------------------------------------------------------------------------------------------*/

#define MID(x,y) ((x+y)>>1)
#define DMID(x,y) ((x+y)/2)
#define mem(a,b) memset(a,b,sizeof(a))

typedef long long LL;
const int sup = 0x7fffffff;
const int inf = -0x7fffffff;

//精度模板
const double eps = 1e-6;
bool dy(double x,double y)  {   return x > y + eps;}         // x > y
bool xy(double x,double y)  {   return x < y - eps;}         // x < y
bool dyd(double x,double y) {   return x > y - eps;}         // x >= y
bool xyd(double x,double y) {   return x < y + eps;}         // x <= y
bool dd(double x,double y)  {   return fabs( x - y ) < eps;}    // x == y
/*-------------------------------------------------------------------------------------------------------------*/
```

String

BKDRhash

```
/*-------------------------------------------------------------------------------------------------------------*/

unsigned int BKDRHash(char *str){
    unsigned int seed = 31;           // 31、131、1313、13131，……
    unsigned int hash = 0;
    while (*str)        hash = hash * seed + (*str++);
    return (hash & (0x7FFFFFFF));
}
//需要冲突处理：vector 模拟拉链法

/*-------------------------------------------------------------------------------------------------------------*/
```

KMP

```
/*-------------------------------------------------------------------------------------------------------------*/
```

```
const int N = 1000100;
string A, B;
int pi[N];
void get_pi(){
    memset(pi,0,sizeof(pi));
    int m = B.length();
    pi[0] = -1;
    int j = -1;
    for (int i = 1;i < m; i ++){
        while(j > -1 &&B[j+1] != B[i])  j = pi[j];
        if (B[j+1] == B[i]) j++;
        pi[i] = j;
    }
}
int KMP(){
    int res = 0;
    get_pi();
    int n = A.length();        int m = B.length();
    int j = -1;
    for (int i = 0; i < n; i ++){
        while(j > -1 && A[i] !=B[j+1])  j = pi[j];
        if (A[i] == B[j+1]) j++;
        if (j == m - 1){
            res ++;            j = pi[j];
        }
    }
    return res;
}
int main(){
    int tt;
    scanf("%d",&tt);
    while (tt--){
        cin>>B;cin>>A;
        cout<<KMP()<<endl;
    }
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------------*/
```

Manacher O(n)最长回文子串

```
/*-------------------------------------------------------------------------------------------------------------------*/

void Manacher(int *p,char *str,int n){                    //p数组记录以每个字符为中心的最长回文半径，
str是字符数组，为了把奇数偶数一块考虑，如果原数组是aabb，则str为$#a#a#b#b#,  n为str数组的长度
    int i;
    int mx = 0;
    int id;
    for(i=1; i<n; i++){
        if( mx > i )
            p[i] = min( p[2*id-i], mx-i );
```

```
        else
            p[i] = 1;
        for(; str[i+p[i]] == str[i-p[i]]; p[i]++);
        if( p[i] + i > mx ){
            mx = p[i] + i;
            id = i;
        }
    }
}
```

/*--*/

Common Function

/*--*/

```
/* 返回大于或者等于指定表达式的最小整数 */
double ceil(double x);

/* 查找完全匹配的子字符串 */
extern char *strstr( char *str1, char *str2 );
size_t string::find( const string& str, size_t pos = 0 ) const;
size_t string::find( char c, size_t pos = 0 ) const;


/* 提取从某个位置开始, 长度为1的子串 */
string substr ( size_t pos = 0, size_t n = npos ) const;

/* 把字符串中的某个子串完全替换成另一个子串 */
string& replace ( size_t pos1, size_t n1, const string& str );
       //用str替换this字符串从pos1位置开始的n1个字符
string& replace ( size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2 );
       //用str中pos2位置开始的n2个字符替换this字符串从pos1位置开始的n1个字符
```

/*--*/

DP

方格取数

/*--*/

```
//NOIP 2000 方格取数
//多线程DP
int n;
int f[31][11][11];
int a[11][11];
int main(){
    scanf("%d",&n);
```

```
    int x,y,c;
    while(scanf("%d%d%d",&x,&y,&c)){
        if (x == 0 && y == 0 && c == 0)       break;
        a[x][y] = c;
    }
    for (int k = 1; k < n + n; k ++)
        for (int i = 1; i <= min(n,k); i ++)
            for (int j = 1; j <= min(n,k); j ++){
                f[k][i][j] = max(f[k-1][i][j], f[k-1][i-1][j]);
                f[k][i][j] = max(f[k][i][j], f[k-1][i][j-1]);
                f[k][i][j] = max(f[k][i][j], f[k-1][i-1][j-1]);
                if (i == j)
                    f[k][i][j] += a[i][k+1-i];
                else    f[k][i][j] += a[i][k+1-i] + a[j][k+1-j];
            }
    printf("%d\n",f[n+n-1][n][n]);
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

构造回文串

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
POJ 1159，题目大意：给定一个字符串，问最少需要添加多少个字符(任意位置)才能构成一个回文串。

思路：答案就是：字符串长度 - 字符串和其反串的最长公共子串长度。 在求最长公共子串时，5000*5000的数组会爆内存。这里因为f[i][]只与f[i-1][]有关系，所以用一个f[2][]的滚动数组就可以完美解决了。

*/
int f[2][5005];
char s1[5005],s2[5005];
int main(){
    int n;
    scanf("%d",&n);
    scanf("%s",s1);
    reverse_copy(s1,s1+n,s2);   //反向复制
    memset(f,0,sizeof(f));
    for (int i = 1; i <= n; i ++){
        for (int j = 1; j <= n; j ++){
            if (s1[i-1] == s2[j-1]){
                f[i%2][j] = f[(i-1)%2][j-1] + 1;
            }
            else{
                f[i%2][j] = max(f[(i-1)%2][j] , f[i%2][j-1]);
            }
        }
    }
    printf("%d\n",n - f[n%2][n]);
    return 0;
}
```

/*--*/

背包

/*--*/

```
//01背包 --- O(VN)
void zero_one_pack(int cost, int weight){
	for (int v = V; v >= cost; v --){
		f[v] = max(f[v], f[v-cost] + weight);
	}
}
for (int i = 1; i <= N; i ++)
	zero_one_pack(c[i], w[i]);

//完全背包 --- O(VN)
void complete_pack(int cost, int weight){
	for (int v = cost; v <= V; v ++){
		f[v] = max(f[v], f[v-cost] + weight);
	}
}
for (int i = 1; i <= N; i ++)
	complete_pack(c[i], w[i]);

//多重背包+倍增优化 --- O(VNlogni)
void multiple_pack(int cost, int weight, int amount){
	if (cost * amount >= V){
		complete(cost, weight);
		return ;
	}
	int k = 1;
	while(k < amount){
		zero_one_pack(k*cost, k*weight);
		amount -= k;
		k <<= 1;
	}
	zero_one_pack(amount*cost, amount*weight);
}

//混合背包
for (int i = 1; i <= N; i ++){
	if 第i件物品属于01背包
		zero_one_pack(c[i], w[i]);
	else if 第i件物品属于完全背包
		complete_pack(c[i], w[i]);
	else if 第i件物品属于多重背包
		multiple_pack(c[i], w[i], n[i]);
}

//bool型背包, f[0] = 1.
void zero_one_pack(int V, int cost){
    for (int v = V; v >= cost; v --){
```

```
        if (f[v-cost])
            f[v] = 1;
    }
    return ;
}
void complete_pack(int V, int cost){
    for (int v = cost; v <= V; v ++){
        if (f[v-cost])
            f[v] = 1;
    }
}
void multiple_pack(int V, int cost, int amount){
    if (cost*amount >= V){
        complete_pack(V, cost);
        return ;
    }
    int k = 1;
    while(k < amount){
        zero_one_pack(V, k*cost);
        amount -= k;
        k <<= 1;
    }
    zero_one_pack(V, amount*cost);
    return ;
}

//分组背包（每组只能选一个）
for (int k = 1; k <= K; k ++)
       for (int v = V; v >= 0; v --)          //一维时必须倒着，二维无所谓
              for (i属于k)
                     dp[k][v] = max(dp[k][v], dp[k-1][v-cost[i]] + value[i]);     //可以省掉第一维

//分组背包变形（每组至少选一个）
for (int k = 1; k <= K; k ++)
       for (i属于k)
              for (int v = V; v >= 0; v --){
              //二维更直接简单，一维还要考虑不能从k-1继承状态，必须倒着
                     dp[k][v] = max(dp[k][v], dp[k][v-price[i]] + value[i]);             //大于1个
                     dp[k][v] = max(dp[k][v], dp[k-1][v-price[i]] + value[i]);
                     //选择1个，并且注意顺序，因为price可能为0
              }
/*-------------------------------------------------------------------------------------------------------------*/
```

枚举状态背包

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
POJ 1837，题目大意：有个天平，给定G个不同大小的力，C个天平两边的力臂，求把所有力放在力臂上后天平平衡的方案数.
```

很难从这道题中找到一般DP那样的最优子结构，所以这类问题也就需要我们来“枚举”所有的情况了.

一般情况下需要枚举状态的DP，我们会设bool f[i][j]表示某种状态存不存在。而这道题要求方案数，所以，

我们设f[i][j]表示前i种物品平衡度为j的方案数.并且因为存在负的平衡度，所以我们给他扩大一倍->15000.

初始f[0][7500] = 1(7500表示平衡度为0)

状态转移方程即为：f[i][j] += f[i-1][j-l[k]*w[i]]

```
*/
#define MID(x,y) ((x+y)>>1)
using namespace std;
typedef long long LL;
LL f[30][20000];
int l[30], w[30];
int main(){
    int C, G;
    scanf("%d%d", &C, &G);
    for (int i = 0; i < C; i ++)
        scanf("%d", &l[i]);
    for (int i = 0; i < G; i ++)
        scanf("%d", &w[i]);
    for (int i = 0; i < 30; i ++)
        for (int j = 0; j < 20000; j ++)
            f[i][j] = 0;
    f[0][7500] = 1;
    for (int i = 1; i <= G; i ++){
        for (int j = 0; j < 15000; j ++){
            for (int k = 0; k < C; k ++)
                if (j - l[k]*w[i] >= 0)
                    f[i][j] += f[i-1][j-l[k]*w[i-1]];
        }
    }
    printf("%I64d\n", f[G][7500]);
       return 0;
}
/*-------------------------------------------------------------------------------------------------------------------*/
```

石子归并

```
/*-------------------------------------------------------------------------------------------------------------------*/
//NOI 1995 石子合并
//区间划分DP

int n;
int a[205],s[205][205],t[205][205];
int num[205][205];
int calnum(){
    for (int i = 0; i < 2 * n - 1 ; i ++)
        num[i][i] = a[i];
```

```
    for (int p = 1; p < n; p ++)
        for (int i = 0; i < 2 * n - 1; i ++){
            if (i + p < 2 * n - 1) {
                int j = i + p;
                num[i][j] = num[i][i] + num[i + 1][j];
            }
        }
}
int calmin(){
    for (int i = 0; i < 205; i ++)
        for (int j = 0; j < 205; j ++)
            s[i][j] = 1e8;
    for (int i = 0; i < 2 * n - 1 ; i ++)
        s[i][i] = 0;
    for (int k = 1; k < n; k ++)
        for (int i = 0; i < 2 * n - 1; i ++){
            if (i + k < 2 * n - 1) {
                int j = i + k;
                for (int p = i; p < j; p ++)
                    s[i][j] = min(s[i][j], s[i][p] + s[p + 1][j] + num[i][p] + num[p + 1][j]);
            }
        }
    int res = 1e8;
    for (int i = 0; i < n; i ++)
        if (s[i][i + n - 1] < res)
            res = s[i][i + n - 1];
    return res;
}
int calmax(){
    for (int p = 1; p < n; p ++)
        for (int i = 0; i < 2 * n - 1; i ++){
            if (i + p < 2 * n - 1) {
                int j = i + p;
                for (int k = i; k < j; k ++)
                    t[i][j] = max(t[i][j], t[i][k] + t[k + 1][j] + num[i][k] + num[k + 1][j]);
            }
        }
    int res = 0;
    for (int i = 0; i < n; i ++)
        if (t[i][i + n - 1] > res)
            res = t[i][i + n - 1];
    return res;
}
int main(){
    scanf("%d",&n);
    for (int i = 0; i < n; i ++){
        scanf("%d",&a[i]);
        a[i + n] = a[i];
    }
    calnum();
    printf("%d\n%d\n",calmin(),calmax());
    return 0;
```

```
}
/*-------------------------------------------------------------------------------------------------------------*/
```

字符串编辑距离

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
POJ 3356，题目大意：给定字符串A和B。求A串转到B串所需的最少编辑操作次数。编辑操作包括：删除、添加一个字符、替换成任意字符。
思路：dp[i][j]，代表字符串1的前i子串和字符串2的前j子串的距离。初始化dp[0][0] = 0,dp[i][0] = i,dp[0][j] = j;
dp[i][j] = min {       dp[i][j-1] + 1     //添加(在i位置后面添加一个字符)
                       dp[i-1][j] + 1    //删除(删除第i个字符)
                       dp[i-1][j-1] + (A[i] == B[i]?0:1)    //替换        }
*/
int f[1010][1010];
char s1[1010],s2[1010];
int main(){
    int l1,l2;
    while(cin>>l1){
        cin>>s1;
        cin>>l2;
        cin>>s2;
        for (int i = 0; i < 1010; i ++){
            for (int j = 0; j < 1010; j ++){
                f[i][j] = INT_MAX;
            }
        }
        for (int i = 0; i <= l1; i ++){
            f[i][0] = i;
        }
        for (int i = 0; i <= l2; i ++){
            f[0][i] = i;
        }
        for (int i = 1; i <= l1; i ++){
            for (int j = 1; j <= l2; j ++){
                f[i][j] = min(f[i-1][j-1] + (s1[i-1]==s2[j-1]?0:1), f[i][j-1] + 1);
                f[i][j] = min(f[i][j], f[i-1][j] + 1);
            }
        }
        cout<<f[l1][l2]<<endl;
    }
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

最大连续子段和

```
/*-------------------------------------------------------------------------------------------------------------*/
```

```
/*
题目大意：给定一个数列，找到两个不相交的子序列，使他们的和最大。
思路：因为求两个不相交的子序列，所以就是找到一个i，使得他左区间最大连续子段和+右区间最大连续子段和最大。考虑求一边的就可以了，即经典的一维最大连续子段和问题。dp[i]表示[0..i]区间的最大连续子段和，b[i]表示以i结尾的最大连续子段和。
状态转移方程：b[i] = max{b[i-1] + a[i] , a[i]},     dp[i] = max{b[j] (j = 0..i) }
时间复杂度：O(N)DP预处理+O(N)查询
*/
const int N = 100010;
int dp1[N],dp2[N],b1[N],b2[N],a[N];
void go_dp(int n){
        //left
        b1[0] = a[0];
        dp1[0] = b1[0];
        int max = dp1[0];
        for (int i = 1; i < n; i ++){
                if (b1[i-1] > 0){
                        b1[i] = b1[i-1] + a[i];
                }
                else{
                        b1[i] = a[i];
                }
                if (b1[i] > max){
                        max = b1[i];
                }
                dp1[i] = max;
        }
        //right
        b2[n-1] = a[n-1];
        dp2[n-1] = b2[n-1];
        max = dp2[n-1];
        for (int i = n - 2; i >= 0; i --){
                if (b2[i+1] > 0){
                        b2[i] = b2[i+1] + a[i];
                }
                else{
                        b2[i] = a[i];
                }
                if (b2[i] > max){
                        max = b2[i];
                }
                dp2[i] = max;
        }
}

int ff(int n){
        int max = INT_MIN;
        for (int i = 0; i < n - 1; i ++){
                if (dp1[i] + dp2[i+1] > max){
                        max = dp1[i] + dp2[i+1];
                }
        }
```

```
        return max;
}
int main(){
        int n;
        while(scanf("%d",&n),n){
                for (int i = 0; i < n; i ++){
                        scanf("%d",&a[i]);
                }
                go_dp(n);
                printf("%d\n",ff(n));
        }
        //system("pause");
        return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

最大连续 M 子段和

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
题目大意：给定n个数，每次选连续的m个数，选3次且选的数不能重复，求选出的这些数的和的最大值.
思路：网上说是01背包，恕本菜没看出来= =……我倒觉得他像3个最大连续子段和，但是限制每次只能选m个数，所以姑且叫他3次最大连续m子段和吧……
设dp[i][j]表示在前i个数中选j次的最大和，首先预处理一下sum[i]表示以i为结尾的连续m个数的和，
则方程为：dp[i][j] = max(dp[i-1][j],  dp[i-m][j-1] + sum[i] )
*/
const int N = 50010;
int a[N], sum[N], f[N][4];
int main(){
    int t, n, m;
    scanf("%d",&t);
    while(t --){
        scanf("%d",&n);
        for (int i = 0; i < n; i ++){
            scanf("%d",&a[i]);
        }
        scanf("%d",&m);

        int s = 0;
        for (int i = 0; i < n; i ++){
            s += a[i];
            if (i > m - 1){
                s -= a[i - m];
                sum[i] = s;
            }
            else{
                sum[i] = s;
            }
        }
        memset(f, 0, sizeof(f));
        for (int i = m; i <= n; i ++){
```

```
            for (int j = 1; j <= 3; j ++){
                f[i][j] = f[i-1][j];
                if (f[i-m][j-1] + sum[i-1] > f[i][j]){
                        f[i][j] = f[i-m][j-1] + sum[i-1];
                }
            }
        }
        printf("%d\n", f[n][3]);
    }
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

最大子矩阵和

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
POJ 1050，思路：如果不是方阵而是一维数组，显然是最大连续子段和。如果已经确定这个矩形应该包含哪些行，而还不知道该包含哪些列，那么可以将每一列的各行元素相加，从而将矩阵转换为一维数组的最大连续子段和。因此，只要我们枚举矩阵应该从哪一行到哪一行，就可以将问题用最大连续子段和策略来求解了，O(N3)。
*/
int a[120][120];
int b[120],dp[120];
int main(){
    int maxn;
    int n;
    while(scanf("%d",&n) == 1){
        maxn = INT_MIN;
        for (int i = 0; i < n; i ++){
            for (int j = 0; j < n; j ++){
                scanf("%d",&a[i][j]);
                maxn = max(maxn, a[i][j]);
            }
        }
        for (int i = 0; i < n; i ++){
            memset(b,0,sizeof(b));
            for (int j = i; j < n; j ++){
                dp[0] = 0;
                for (int k = 0; k < n; k ++){
                    b[k] += a[j][k];
                    dp[k+1] = (dp[k] > 0 ? dp[k] : 0) + b[k];
                    maxn = max(maxn, dp[k+1]);
                }
            }
        }
        printf("%d\n",maxn);
    }
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

计算数列所有子区间异或不同结果

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*
CodeForces #150 div1 A，题目大意：给定一个数列{a1,a2,...,an}，定义f[l,r] = al | al+1 | al+2 | ... | ar 。统计所有f[l,r]的不同结果。
思路：  设dp[i][j]为以i为右区间，区间长度为j的f[]，即dp[i][j] = f[i - j, i]。那么有方程dp[i][] = dp[i-1][] | a[i].但时间复杂度会达到O(n^2)，无法承受。这里我们巧妙的用set来自动把前面的状态去重来减少不必要的操作，即设set <int> dp[i]为以i为右区间的所有f[*, i]。set去重的强大啊！~直接帮助我们忽略分析或运算性质的过程了。
*/
set <int> dp[N],ans;
int a[N];
int main(){
    int n;
    cin>>n;
    for (int i = 0; i < n; i ++){
        cin>>a[i];
        dp[i].clear();
    }
    ans.clear();
    for (int i = 0; i < n; i ++){
        dp[i].insert(a[i]);
        ans.insert(a[i]);
        if (i != 0){
            set <int> ::iterator it;
            for (it = dp[i-1].begin(); it != dp[i-1].end(); it ++){
                dp[i].insert(*it | a[i]);
            }
            for (it = dp[i].begin(); it != dp[i].end(); it ++){
                ans.insert(*it);
            }
        }
    }
    cout<<ans.size()<<endl;
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------------*/
```

树形 DP：树节点的最大独立子集

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*
POJ 3342 题目大意：给定一棵关系树，子节点和父节点不能同时选，最后最多能选多少点？并且判断最大解是否唯一。
思路：找最多能选几个点和上面那道题是一样的，这题的难点在于怎么判断解唯一。我们再设一个flag[i][2]，来表示dp[i][2]的方案是否唯一。
状态转移方程：flag的方程在代码中标注的地方
*/
const int N = 210;
map <string, int> m;
vector <int> v[N];
int dp[N][2];
```

```
bool flag[N][2],vis[N];
void tree_dp(int n){
    if (!vis[n]){
        vis[n] = true;
        for (int i = 0; i < v[n].size(); i ++){
            int p = v[n][i];
            if (!vis[p]){
                tree_dp(p);
                dp[n][0] += max(dp[p][0], dp[p][1]);
                //flag的状态转移方程
                if (dp[p][0] == dp[p][1]){
                    flag[n][0] = true;
                }
                else if (dp[p][0] > dp[p][1] && flag[p][0]){
                    flag[n][0] = true;
                }
                else if (dp[p][1] > dp[p][0] && flag[p][1]){
                    flag[n][0] = true;
                }
                dp[n][1] += dp[p][0];
                if (flag[p][0])
                    flag[n][1] = true;
            }
        }
        dp[n][1] ++;
    }
}
int main(){
    int n;
    while(cin>>n){
        if (n == 0){
            break;
        }
        int mnum = 1;
        m.clear();
        for (int i = 0; i < N; i ++){
            v[i].clear();
        }
        memset(vis,0,sizeof(vis));
        memset(dp,0,sizeof(dp));
        memset(flag,false,sizeof(flag));
        string s;
        cin>>s;
        m[s] = mnum ++;
        for (int i = 1; i < n; i ++){
            string s1,s2;
            cin>>s1>>s2;
            if (m.find(s1) == m.end())  m[s1] = mnum ++;
            if (m.find(s2) == m.end())  m[s2] = mnum ++;
            v[m[s2]].push_back(m[s1]);
        }
        tree_dp(1);
```

```
        if (dp[1][0] == dp[1][1]){
            cout<<dp[1][1]<<" No\n";
        }
        else{
            if (dp[1][0] > dp[1][1]){
                if (flag[1][0])
                    cout<<dp[1][0]<<" No\n";
                else{
                    cout<<dp[1][0]<<" Yes\n";
                }
            }
            else if (dp[1][0] < dp[1][1]){
                if (flag[1][1])
                    cout<<dp[1][1]<<" No\n";
                else{
                    cout<<dp[1][1]<<" Yes\n";
                }
            }
        }
    }
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

树形 DP：树上每点的最大权值路径

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
SGU 149 && HDU 2196
题目大意：给一棵树，每条树边都有权值，问从每个顶点出发，经过的路径权值最大为多少？每条树边都只能走一次，n
<= 10000,权值<=10^9
思路：以1为根固定树。然后两遍DFS，第一遍从子节点到根节点更新，求出每个节点在以它为根的子树上的最远距离和次
远距离（并且要保证他俩不在一条支路上，这个为第二次DFS做准备），第二遍DFS从根节点到子节点更新，因为每个节点
的最远路径不一定在他的子树上，所以要通过父亲的最远路经对节点进行一次更新。但有个问题就是万一父亲节点的最远
路经正好在该节点子树这条路上怎么办，则此时我们选择不在这个支路上的那条父亲节点的次长路来对节点更新。
*/
const int N = 10010;
vector <pair<int,int> > v[N];
int dp1[N],dp2[N],ndp1[N],ndp2[N];  //最远路和次远路
int vis[N];
void dfs(int n,int root){
    if (!vis[n]){
        vis[n] = 1;
        for (int i = 0; i < v[n].size(); i ++){
            int p = v[n][i].first;
            if (p == root)    continue;
            if (!vis[p]){
                dfs(p, n);
                //更新从子节点上来的信息，因为dp1和dp2不能在同一条支路上，所以只用对每个子节点的dp1来更
新。
                if (dp1[p] + v[n][i].second > dp1[n]){
```

```
                    dp2[n] = dp1[n];
                    ndp2[n] = ndp1[n];
                    dp1[n] = dp1[p] + v[n][i].second;
                    ndp1[n] = p;
                }
                else{
                    if (dp1[p] + v[n][i].second > dp2[n]){
                        dp2[n] = dp1[p] + v[n][i].second;
                        ndp2[n] = p;
                    }
                }
            }
        }
    }
}

void tree_dp(int n, int root){
    if (!vis[n]){
        vis[n] = 1;
        //先通过父节点更新节点
        for (int i = 0; i < v[n].size(); i ++){
            if (v[n][i].first == root){
                if (ndp1[root] != n){
                    dp2[n] = dp1[n];
                    ndp2[n] = ndp1[n];
                    dp1[n] = dp1[root] + v[n][i].second;
                    ndp1[n] = root;
                }
                else{
                    if (dp2[root] + v[n][i].second > dp1[n]){
                        dp2[n] = dp1[n];
                        ndp2[n] = ndp1[n];
                        dp1[n] = dp2[root] + v[n][i].second;
                        ndp1[n] = root;
                    }
                    else if (dp2[root] + v[n][i].second > dp2[n]){
                        dp2[n] = dp2[root] + v[n][i].second;
                        ndp2[n] = root;
                    }
                }
            }
        }
        //从节点向子节点传递信息
        for (int i = 0; i < v[n].size(); i ++){
            if (v[n][i].first != root){
                int p = v[n][i].first;
                tree_dp(p, n);
            }
        }
    }
}
int main(){
```

```
    int n;
    while(scanf("%d",&n)!=EOF){
        for (int i = 0; i <= n; i ++)
            v[i].clear();
        memset(vis,0,sizeof(vis));
        memset(dp1,0,sizeof(dp1));
        memset(dp2,0,sizeof(dp2));
        memset(ndp1,0,sizeof(ndp1));
        memset(ndp2,0,sizeof(ndp2));
        for (int i = 2; i <= n; i ++){
            int a,b;
            scanf("%d%d",&a,&b);
            v[i].push_back(make_pair(a,b));
            v[a].push_back(make_pair(i,b));
        }
        dfs(1,0);
        memset(vis,0,sizeof(vis));
        tree_dp(1,0);
        for (int i = 1; i <= n; i ++){
            printf("%d\n",dp1[i]);
        }
    }
    //system("pause");
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------------*/
```

状态压缩 DP：棋盘模型

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*
sgu 223 基本棋盘模型：在n*n(n<=10)的棋盘上放k个国王（可攻击相邻的8个格子），求使他们无法相互攻击的方案数.
这类题需要先将每一行的可行状态先DFS出来用二进制表示压缩到S[]数组中.然后以此为状态DP.
特征：相邻格子不能放
*/
int n, k;
const int NUM_OF_PLACEMENGT = 520;
int s[NUM_OF_PLACEMENGT], c[NUM_OF_PLACEMENGT], place_num;          //the placement of each line
long long dp[13][NUM_OF_PLACEMENGT][103];

void dfs(int p, int condition_num, int condition_one_amount){              //Store the placement of each
line
    if (p == n){
        s[++ place_num] = condition_num;
        //cout << place_num << " " << s[place_num] << endl;
        c[place_num] = condition_one_amount;
        return ;
    }
    dfs(p+1, condition_num << 1, condition_one_amount);
    if (!(condition_num & 1))
       //相邻格子不能放
```

```
        dfs(p+1, condition_num << 1 | 1, condition_one_amount + 1);
    return ;
}
bool ifok(int s1, int s2){      //decide whether tha current condition and the last condition are
comtradictry.
    if(s1 & s2) return false;          //和正上方判断
    if(s1 & (s2<<1))return false;      //和右上方判断
    if(s1 & (s2>>1))return false;      //和左上方判断
    return true;
}
int main(){
    while(scanf("%d %d", &n, &k) != EOF){
        //cout << n << " " << k << endl;
        mem(dp, 0);
        place_num = 0;
        dp[0][1][0] = 1;
        dfs(0, 0, 0);
        for (int i = 1; i <= n; i ++){
            for (int j1 = 1; j1 <= place_num; j1 ++){
                for (int j2 = 1; j2 <= place_num; j2 ++){
                    for (int w = 0; w <= k; w ++){
                        if (ifok(s[j1], s[j2]) && w-c[j1] >= 0)
                            dp[i][j1][w] += dp[i-1][j2][w-c[j1]];
                    }
                }
            }
        }
        long long res = 0;
        for (int j = 1; j <= place_num; j ++)
            res += dp[n][j][k];
        printf("%I64d\n", res);
    }
        return 0;
}

/*
HDU4239在n*m(n<=100,m<=10)的棋盘上放士兵，每个士兵的曼哈顿距离2的地方都不能放别的士兵。并且有些地方不能放
士兵。求最多能放几个士兵。
特征：曼哈顿距离为2的点不能放
*/

int n, m;
const int NUM_OF_PLACEMENGT = 500;
int s[NUM_OF_PLACEMENGT], c[NUM_OF_PLACEMENGT], place_num;          //the placement of each line
int dp[2][NUM_OF_PLACEMENGT][NUM_OF_PLACEMENGT];
vector <int> dd[NUM_OF_PLACEMENGT];
int no[105];

void dfs(int p, int condition_num, int condition_one_amount){              //Store the placement of each
line
    if (p == m){
        s[++ place_num] = condition_num;
```

```
        //cout << condition_num << endl;
        c[place_num] = condition_one_amount;
        return ;
    }
    dfs(p+1, condition_num << 1, condition_one_amount);
    if (!(condition_num & 2))
        //隔一个不能放
        dfs(p+1, condition_num << 1 | 1, condition_one_amount + 1);
    return ;
}
bool ifok(int s1, int s2){      //decide whether tha current condition and the last condition are
comtradictry.
    //if(s1 & s3)     return false;         //和正上方判断
    if(s1 & (s2<<1))    return false;        //和右上方判断
    if(s1 & (s2>>1))    return false;        //和左上方判断
    return true;
}
int main(){
    while(scanf("%d %d", &n, &m) == 2){
        mem(no, 0);
        place_num = 0;
        mem(dp, -1);
        dp[0][1][1] = 0;
        dfs(0, 0, 0);
        for (int i = 1; i <= n; i ++)
        for (int j = 1; j <= m; j ++){
            int tmp;
            scanf("%d", &tmp);
            if (tmp == 0)   no[i] += (1 << (m-j));
        }
        for (int i = 1; i <= n; i ++)
            for (int j1 = 1; j1 <= place_num; j1 ++){
                if(s[j1] & no[i])   continue;
                for (int j3 = 1; j3 <= place_num; j3 ++){
                    if (s[j1] & s[j3])  continue;
                    for (int j2 = 1; j2 <= place_num; j2 ++){
                        if (ifok(s[j1],s[j2]) && dp[(i-1)&1][j2][j3] != -1){
                            dp[i&1][j1][j2] = max(dp[i&1][j1][j2], dp[(i-1)&1][j2][j3] + c[j1]);;
                        }
                    }
                }
            }
        int res = 0;
        for (int j1 = 1; j1 <= place_num; j1 ++)
            for (int j2 = 1; j2 <= place_num; j2 ++)
                res = max(res, dp[n&1][j1][j2]);
        printf("%d\n", res);
    }
    return 0;
}
```

/*----------*/

Mathematics

Mathematical Formula

/*----------*/

错排公式：M(n)=(n-1)[M(n-2)+M(n-1)]

泰勒级数列表

下面我们给出了几个重要的泰勒级数。参数x为复数时它们依然成立。

- 指数函数和自然对数：

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \quad \forall x$$

$$\ln(1+x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} x^n \quad \forall x \in (-1,1]$$

- 几何级数：

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n \quad \forall x : |x| < 1$$

- 二项式定理：

$$(1+x)^\alpha = \sum_{n=0}^{\infty} C(\alpha, n) x^n \quad \forall x : |x| < 1, \forall \alpha \in \mathbb{C}$$

- 三角函数：

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \quad \forall x$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} \quad \forall x$$

$$\tan x = \sum_{n=1}^{\infty} \frac{B_{2n}(-4)^n(1-4^n)}{(2n)!} x^{2n-1} \quad \forall x : |x| < \frac{\pi}{2}$$

$$\sec x = \sum_{n=0}^{\infty} \frac{(-1)^n E_{2n}}{(2n)!} x^{2n} \quad \forall x : |x| < \frac{\pi}{2}$$

$$\arcsin x = \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} \quad \forall x : |x| < 1$$

$$\arctan x = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1} \quad \forall x : |x| < 1$$

$$\arctan x = \frac{\pi \operatorname{sgn} x}{2} - \frac{1}{x} + \sum_{k=1}^{\infty} \frac{(-1)^k}{(2k+1)\, x^{2k+1}} \quad \forall x : |x| > 1$$

/*----------*/

筛素数

```
/*-------------------------------------------------------------------------------------------------------------*/
/*-------------------厄拉多塞筛法-------------------*/
#define MAX 100000
bool noprime[MAX];
void Prime(int n){
    int k;
    noprime[0] = noprime[1] = 1;
    for (int i = 2; i * i <= n; i ++)
        if (!noprime[i]){
            k = i + i;
            while(k <= n){
                                noprime[k]=1;
                k += i;
            }
        }
}

/*--------------------线性筛素数--------------------*/
bool noprime[MAX];
vector <int> prime;
void Prime(int n){
    for (int i = 2; i <= n; i ++){
        if (!noprime[i]){
            prime.push_back(i);
        }
        for (int j = 0; j < prime.size() && prime[j] * i <= n; j ++){
            noprime[prime[j]*i] = 1;
            if (i % prime[j] == 0)  break;           //每个数只被他自身最小的素因子筛去
        }
    }
}
/*-------------------------------------------------------------------------------------------------------------*/
```

分解因子

```
/*-------------------------------------------------------------------------------------------------------------*/
/*-----------------试除法求n的素因子-------------------*/
vector <int> p;
void find_prime_factor(int n)
{
    for (int i = 2; i * i <= n; i ++){
        //枚举到sqrt(n)即可
        //这里有一个优化就是可以预处理筛出素数，然后上面i只用枚举素数即可
        if (n % i == 0){
            p.push_back(i);
            while(n % i == 0){
                n = n / i;
```

```
            }
        }
        if (n == 1) break;
    }
    if (n > 1){                                 //只可能有一个大于sqrt(n)的素因子，留在最后就行
        p.push_back(n);
    }
    return;
}

/*----------------试除法求n的所有因子-------------------*/
vector <int> factor;
void Factor(int n){
    factor.clear();
    factor.push_back(1);
    factor.push_back(n);
    for (int i = 2; i * i <= n; i ++){
        if (n % i == 0){
            factor.push_back(i);
            factor.push_back(n / i);
        }
    }
}
/*-------------------------------------------------------------------------------------------------------------*/
```

N!分解素因子 && 组合数分解素因子

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
        ①找出n!的素因子是很容易的，显然就是1~n的素数。
        ②关键在于怎么求出素因子的个数：

        以素因子2为例：
        N!      =1*2*3*4*5……N
                =(2*4*6……) * (1*3*5……)              //因为有N/2个偶数，所以偶数部分可以提出N/2个2，
                =2^(N/2) * (1*2*3*……N/2) * (1*3*5*……)
                =2^(N/2) * (N/2)! * (1*3*5*……)

        于是有递推公式 f(n,2) = f(n/2,2) + n/2，表示n! 中2的个数。
        用同样的方法可以推出 f(n,p) = f(n/p,p) + n/p，表示n! 中素数p的个数。
*/

/*-----------  n!中素因子p的个数：f(n,p) = f(n/p, p) + n/p ： O(log(p)(n)) ------------*/
long long f(int n, int p){
    long long res = 0;
    while(n){
        res += n/p;
        n /= p;
    }
    return res;
}
```

```
/*---- n!分解素因子: 复杂度O(n)（预先筛好素数O(p*logn)）,(n <= 10^6)----*/
vector <pair<int, long long> > prime;     //n!的素因子就是1~n中的素数,第一个记录素因子,第二个记录个数
bool noprime[1000010];
void factor(int n){
    prime.clear();
        //筛素数，要使用多次时可以放函数外预处理来优化时间
    for (int i = 2; i <= n; i ++){
        if (!noprime[i]){
            prime.push_back(make_pair(i, 0));
        }
        for (int j = 0; j < prime.size() && prime[j].first*i <= n; j ++){
            noprime[prime[j].first*i] = 1;
            if (i % prime[j].first == 0)
                break;
        }
    }
        //n!分解质因子过程，要用到上面的f(n, p)
    for (int i = 0; i < prime.size(); i ++){
        int p = prime[i].first;
        prime[i].second = f(n, p);
    }
}


/*---- 组合数分解素因子: C(n,k) = n!/(k!(n-k)!) ------*/
vector <pair<int, long long> > prime;     //组合数素因子表,第一个记录素因子,第二个记录个数
long long factor(int n, int k){

        prime.clear();
        //筛素数，要使用多次时可以放函数外预处理来优化时间
    for (int i = 2; i <= n; i ++){
        if (!noprime[i]){
            prime.push_back(make_pair(i, 0));
            for (int j = i; j <= n; j ++)
                ff[j][i] = f(j, i);
        }
        for (int j = 0; j < prime.size() && prime[j].first*i <= n; j ++){
            noprime[prime[j].first*i] = 1;
            if (i % prime[j].first == 0)
                break;
        }
    }
        //组合数分解素因子过程，要用到上面的f(n, p)
    long long res = 1;
    for (int i = 0; i < prime.size(); i ++){
        if (prime[i].first > n)
            break;
        int p = prime[i].first;
        prime[i].second = f(n, p);
        if (p <= k)
            prime[i].second -= f(k, p);
        if (p <= (n - k))
```

```
            prime[i].second -= f(n-k, p);
        res *= (prime[i].second + 1);
    }
    return res;
}

/*-----------------------------------------------------------------------------------------------------------*/
```

Miller-Rabin && Pollard-ρ

```
/*-----------------------------------------------------------------------------------------------------------*/
//POJ1811 判断一个数是不是素数，如果是合数，求出最小的非平凡因子（非1非本身的因子）

//return a * b % m
unsigned long long mul_mod(unsigned long long a, unsigned long long b, unsigned long long m){
    //为了防止long long型a * b溢出，有时需要把乘法变加法
    //且因为暴力加法会超时要使用二分快速乘法模（模仿二分快速幂模……）
    unsigned long long res = 0, tmp = a % m;
    while(b){
        if (b & 1)
        {
            res = res + tmp;
            res = (res >= m ? res - m : res);
        }
        b >>= 1;
        tmp <<= 1;
        tmp = (tmp >= m ? tmp - m : tmp);
    }
    return res;
}

//return a ^ b % m
long long exp_mod(long long a, long long b, long long m){
    long long res = 1 % m, tmp = a % m;
    while(b){
        if (b & 1){
            //如果m在int范围内直接用下一式乘就可以，否则需要用下二式把乘法化加法，用快速乘法模
            //res = (res * t) % m;
            res = mul_mod(res, tmp, m);
        }
        //同上
        //t = t * t % m;
        tmp = mul_mod(tmp, tmp, m);

        b >>= 1;
    }
    return res;
}

/*-------------Miller-Rabin 素数测试 部分(用到上面mul_mod和exp_mod   素数return true)--------------*/
```

```
bool Miller_Rabin(long long n){
    int a[5] = {2, 3, 7, 61, 24251};
    //一般Miller_Rabin素数测试是随机选择100个a,这样的错误率为0.25^100
    //但在OI&&ACM中，可以使用上面一组a，在这组底数下，10^16内唯一的强伪素数为46,856,248,255,981

    if (n == 2)
        return true;
    if (n == 1 || (n & 1) == 0)
        return false;

    long long b = n - 1;
    for (int i = 0; i < 5; i ++){
        if (a[i] >= n)
            break;
        while((b & 1) == 0)    b >>= 1;
        long long t = exp_mod(a[i], b, n);
        while(b != n - 1 && t != 1 && t != n - 1){
            t = mul_mod(t, t, n);
            b <<= 1;
        }
        if (t == n - 1 || (b & 1))
            continue;
        else
            return false;
    }
    return true;
}
/*-------------Miller-Rabin 素数测试 部分--------------*/

/*-------------pollard-rho 大整数n因子分解 部分(用到mul_mod()和Miller-Rabin测试)--------------*/
long long factor[100];      //存n的素因子
long long nfactor, minfactor;

long long gcd(long long a, long long b){
    return b ? gcd(b, a%b) : a;
}
void Factor(long long n);
void pollard_rho(long long n){
    if (n <= 1)
        return ;
    if (Miller_Rabin(n)){
        factor[nfactor ++] = n;
        if (n < minfactor)
            minfactor = n;
        return ;
    }
    long long x = 2 % n, y = x, k = 2, i = 1;
    long long d = 1;
    while(true){
        i ++;
        x = (mul_mod(x, x, n) + 1) % n;
        d = gcd((y - x + n) % n, n);
```

```
        if (d > 1 && d < n){
            pollard_rho(d);
            pollard_rho(n/d);
            return ;
        }
        if (y == x){
            Factor(n);
            return ;
        }
        if (i == k){
            y = x;
            k <<= 1;
        }
    }
}
void Factor(long long n){
    //有时候RP不好 or n太小(比如n==4就试不出来……)用下面的pollard_rho没弄出来，则暴力枚举特殊处理一下
    long long d = 2;
    while(n % d != 0 && d * d <= n)
        d ++;
    pollard_rho(d);
    pollard_rho(n/d);
}
/*-------------pollard-rho 大整数n因子分解 部分--------------*/

int main(){
    //srand(time(0));
    int t;
    cin >> t;
    for (int i = 0; i < t; i ++){
        nfactor = 0;
        minfactor = (1L << 63);
        long long n;
        cin >> n;
        pollard_rho(n);
        if (nfactor == 1 && factor[0] == n){
            cout << "Prime\n";
            continue;
        }
        sort(factor, factor+nfactor);
        cout << factor[0] << endl;
    }
    return 0;
}
/*-----------------------------------------------------------------------------------------------------------*/
```

A ^ B % M

```
/*-----------------------------------------------------------------------------------------------------------*/
/*
FZU 1752 (a,b,c <= 2^63 - 1)
```

```
由于m也很大，还需要快速乘法模，而且递归形式慢过不了此题……
*/

//return a * b % m
unsigned long long quick_add_mod(unsigned long long a, unsigned long long b, unsigned long long m){
    //为了防止long long型a * b溢出，有时需要把乘法变加法
    //且因为暴力加法会超时要使用二分快速乘法模（模仿二分快速幂模……）
    unsigned long long res = 0, tmp = a % m;
    while(b){
        if (b & 1){
            res = res + tmp;
            res = (res >= m ? res - m : res);                //用减法比用mod快
        }
        b >>= 1;
        tmp <<= 1;
        tmp = (tmp >= m ? tmp - m : tmp);
    }
    return res;
}

//return a ^ b % m
long long exp_mod(long long a, long long b, long long m){
    long long res = 1 % m, tmp = a % m;
    while(b){
        if (b & 1){
            //如果m在int范围内直接用下一式乘就可以，否则需要用下二式把乘法化加法，用快速乘法模
            //res = (res * t) % m;
            res = quick_add_mod(res, tmp, m);
        }
        //同上
        //t = t * t % m;
        tmp = quick_add_mod(tmp, tmp, m);

        b >>= 1;
    }
    return res;
}
int main(){
    long long a,b,c;
    while(scanf("%I64d%I64d%I64d", &a, &b, &c) != EOF){
        printf("%I64d\n", exp_mod(a,b,c));
    }
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

Matrix

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
        类似整数快速幂模.
```

```
        几点注意事项：
                ①矩阵乘法、快速幂是要注意保证乘法、加法不溢出(地方已标出)
                ②一定要把每个矩阵的row,col都设置清楚……(这是模版比较挫的地方之一= =……)
*/
const int MAX = 30;
struct Mat{
    int row, col;
    LL mat[MAX][MAX];
};
//initialize square matrix to unit matrix
Mat unit(int n){
    Mat A;
    A.row = A.col = n;
    memset(A.mat, 0, sizeof(A.mat));
    for (int i = 0; i < n; i ++)
        A.mat[i][i] = 1;
    return A;
}
//return T(A)
Mat transpose(Mat A){
    Mat T;
    T.row = A.col;
    T.col = A.row;
    for (int i = 0; i < A.col; i ++)
        for (int j = 0; j < A.row; j ++)
            T.mat[i][j] = A.mat[j][i];
    return T;
}
//return (A+B)%mod
Mat add(Mat A, Mat B, int mod){
    Mat C = A;
    for (int i = 0; i < C.row; i ++)
        for (int j = 0; j < C.col; j ++)
            C.mat[i][j] = (A.mat[i][j] + B.mat[i][j]) % mod;
    return C;
}
//return (A-B)%mod
Mat dec(Mat A, Mat B, int mod){
    Mat C = A;
    for (int i = 0; i < C.row; i ++)
        for (int j = 0; j < C.col; j ++)
            C.mat[i][j] = (A.mat[i][j] - B.mat[i][j] + mod) % mod;
    return C;
}
//return A*B%mod
Mat mul(Mat A, Mat B, int mod){
    Mat C;
    C.row = A.row;
    C.col = B.col;
    for (int i = 0; i < A.row; i ++){
        for (int j = 0; j < B.col; j ++){
            C.mat[i][j] = 0;
```

```
            for (int k = 0; k < A.col; k ++)
                //注意这里要保证乘法不溢出，否则还需要设计特殊的乘法模
                C.mat[i][j] += A.mat[i][k] * B.mat[k][j];
            C.mat[i][j] %= mod;
        }
    }
    return C;
};
//return A^n%mod
Mat exp_mod(Mat A, int n, int mod){
    Mat res = unit(A.row);
    while(n){
        if (n & 1){
            res = mul(res, A, mod);
        }
        A = mul(A, A, mod);
        n >>= 1;
    }
    return res;
}
//return S = A^1 + A^2 + …… + A^n   (binary search)
Mat Sum(Mat A, int n, int mod){
    if (n == 0){
        Mat zero = A;
        memset(zero.mat, 0, sizeof(zero.mat));
        return zero;
    }
        if (n == 1)
        return A;
    if (n & 1){
        return add(Sum(A, n-1, mod), exp_mod(A, n, mod), mod);
    }
    else{
        return mul(Sum(A, n/2, mod), add(exp_mod(A, n/2, mod), unit(A.col), mod), mod);
    }
}
/*-------------------------------------------------------------------------------------------------------------*/
```

容斥原理：[1..r]内与 n 互素的数的个数

```
/*-------------------------------------------------------------------------------------------------------------*/
int solve(int r,int n){
    int res = 0;
    vector <int> p;
    for (int i = 2; i * i <= n; i ++){
        if (n % i == 0){
            p.push_back(i);
            while(n % i == 0){
                n = n / i;
            }
        }
```

```
    }
    if (n > 1){
        p.push_back(n);
    }
    for (int msk = 1; msk < (1 << p.size()); msk ++){
        int mult = 1, bit = 0;
        for (int i = 0; i < p.size(); i ++){
            if (msk & (1 << i)){
                ++bit;
                mult *= p[i];
            }
        }
        int cur = r / mult;
        if (bit % 2 == 1){
            res += cur;
        }
        else    res -= cur;
    }
    return r - res;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

Cantor

```
/*-------------------------------------------------------------------------------------------------------------*/
int cantor[]={1,1,2,6,24,120,720,5040,40320,362880,3628800,39916800};          //=n!
//康托展开，求数列是全排列第几大数(复杂度O（n^2）)
//参数puzz[]为待展开之的集合的各元素,如需展开2134,则puzz[4]={2,1,3,4}.这里puzz也可以是string，char[]...etc
int Can(int puzz[],int psize)           //psize puzz集合的大小
{
    int ct = 0;
    for(int i = 0; i < psize-1; i++){
        int tmp = 0;
        for(int j = i + 1; j < psize; j++){
            if( puzz[j] < puzz[i]) tmp++;
        }
        ct += tmp * cantor[psize-i-1];
    }
    return ct;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

Euler's totient function

```
/*-------------------------------------------------------------------------------------------------------------*/
//由公式phi(i) = m(1-1/p1)(1-1/p2)……(1-1/pn) 求单个phi(i)
//只用一个phi值可以求2^50左右的phi值
long long phi(long long n){
    long long res = n;
```

```
    for (int i = 2; i * i <= n; i ++){
        if (n % i == 0){
            res = res / i * (i - 1);
            while(n % i == 0)
                n /= i;
        }
    }
    if (n > 1){
        res = res / n * (n - 1);
    }
    return res;
}

//但是如果要用所有phi(1)~phi(n)的值的话上面的方法就行不通了
//我们知道当m,n互素时，phi(mn) = phi(m)*phi(n) 且  phi(m) = phi(p1^q1) * phi(p2^q2) * …… * phi(pn^qn)
//那么当p是素数且是m的约数时，phi(m*pi) = …… * phi(pr^(qr+1)) * ……  = phi(m) * pi
//
//总结：p是素数：①p是m的约数 phi(m*p) = phi(m)*p;  ②p不是m的约数 phi(m*p) = phi(m)*phi(p) = phi(m)*(p-1)

/*----------利用线性筛素数的思想快速求1~n的欧拉函数----------*/
int phi[MAX];
bool noprime[MAX];
vector <int> prime;
void Euler(int n){
    phi[1] = 0;
    for (int i = 2; i <= n; i ++){
        if (!noprime[i]){
            prime.push_back(i);
            phi[i] = i - 1;
        }
        for (int j = 0; j < prime.size() && prime[j] * i <= n; j ++){
            noprime[prime[j]*i] = 1;
            if (i % prime[j] == 0){
                phi[prime[j]*i] = phi[i] * prime[j];
            }
            else{
                phi[prime[j]*i] = phi[i] * phi[prime[j]];
            }
        }
    }
}
/*-------------------------------------------------------------------------------------------------------------*/
```

ExGCD: AX+BY=C && AX=C(mod B) && 一般同余式组

```
/*-------------------------------------------------------------------------------------------------------------*/
/*      POJ 2891
整个过程求解同于方程组
x = a1 (mod m1)
x = a2 (mod m2)
……
```

```
x = ar (mod mr)
（m1 m2 …… mr不必互素， （互素直接用中国剩余定理即可） ）

函数indeterminate_equation()求解不定方程ax + by = c  ->  AX = C (mod B)
*/
void ext_gcd(long long a, long long b, long long &x, long long &y){
    if (b == 0){
        x = 1;
        y = 0;
        return ;
    }
    ext_gcd(b, a%b, x, y);
    long long tmp = x;
    x = y;
    y = tmp - a/b * y;
    return ;
}
long long gcd(long long a, long long b){
    return b ? gcd(b, a % b) : a;
}
long long lcm(long long a, long long b){
    return a / gcd(a, b) * b;
}

//求解不定方程ax + by = c  &&  模线性方程 AX = C (mod B)
//POJ 1061的过程，整合成一个函数
bool indeterminate_equation(long long a, long long b, long long c, long long &x, long long &y){
    int g = gcd(a, b);
    if (c % g != 0){
        return false;
    }
    a /= g;
    b /= g;
    c /= g;
    ext_gcd(a, b, x, y);
    x *= c;
    y *= c;
    //上面过程是求解出x ,y， 下面过程是求x的最小整数值
    long long tmp = abs(double(b));
    x = (x % tmp + tmp) % tmp;
    return true;
}

long long m[1010];
long long r[1010];
int main(){
    int k;
    while(cin >> k){
        long long mlcm = 1;
        int ok = 1;
        for (int i = 1; i <= k; i ++){
            cin >> m[i] >> r[i];
```

```
        }
        long long ans = r[1];
        for (int i = 2; i <= k; i ++){
            long long a = mlcm = lcm(mlcm, m[i-1]);
            long long b = m[i];
            long long c = r[i] - ans;
            long long x, y;
            if (indeterminate_equation(a, b, c, x, y)){
                ans = ans + x * mlcm;
            }
            else{
                cout << -1 << endl;
                ok = 0;
                break;
            }
        }
        if (ok)
            cout << ans << endl;
    }
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------------*/
```

ExGCD: AX+BY=C + 限制区间

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*
        sgu106: 给定a,b,c,x1,x2,y1,y2，求满足a*x+b*y+c = 0的解x满足x1<=x<=x2,y满足y1<=y<=y2.求满足条件的解的个数。
        我们先令c = -c，转化为我们熟悉的线性方程ax + by = c.然后：
        ①用扩展欧几里德找到一组解(x0,y0)，这个很简单。但也有需要注意的地方，比如a,b,c=0的情况需要特殊注意一下，虽然扩展欧几里德可以解出a,b,c为任何值的一组解，但是这些情况在寻求解的个数的时候还是和一般方程不太一样的。
        ②我们也知道方程的通解为：
        x = x0 + kb’  (1);
        y = y0 - ka’  (2) ;                    (b’ = b/gcd(a,b),   a’ = a / gcd(a,b) ).
        那么sgu 106就是求在范围内k能取几个：
        ③将x1,x2带入上面(1)方程求出k1,k2 (k1 < k2); 同理将y1, y2带入上面(2)方程求出k3, k4 (k3 < k4).则ans = min(k2, k4) - max(k1, k3).(这个min,max,等式一时半会儿也不好解释清楚，最好还是在纸上比划比划验证一下……).  而且，注意到我们是没把x0,y0解算进去的，于是当x0,y0也在范围内时，ans+1.
        ④细节：边界问题：例如，如果x0 = 3, x1 = 5, x2 =7, b = 2,   那么k2 - k1 = 2 - 1 = 1，实际上[x1..x2]内是可以取到两个k的~~应该看出什么了，当xo < x1 < x2时，计算k1时要把x1-1，同理当x1 < x2 < x0时，计算k2时要把x2+1；对y0, y1, y2同理。
*/
long long gcd(long long a, long long b){
    if (b == 0)
        return a;
    return gcd(b, a%b);
}
void ext_gcd(long long a, long long b, long long &x, long long &y){
    if (b == 0){
```

```
        x = 1;
        y = 0;
        return ;
    }
    ext_gcd(b, a%b, x, y);
    long long tmp = x;
    x = y;
    y = tmp - a / b * y;
    return ;
}
long long num_of_equation(long long x0, long long y0, long long x1, long long y1, long long x2, long long
y2, long long a, long long b){
    long long xx1 = x1, xx2 = x2, yy1 = y1, yy2 = y2;   //保留区间，最后判断x0,y0是否在区间内需要

    //边界处理：
    if (x0 > x2) x2 ++;
    if (x0 < x1) x1 --;
    if (y0 > y2) y2 ++;
    if (y0 < y1) y1 --;

    long long k1 = (x1 - x0)/b;
    long long k2 = (x2 - x0)/b;
    if (k1 > k2){
        long long tmp = k1;
        k1 = k2;
        k2 = tmp;
    }
    long long k3 = (y1 - y0)/-a;
    long long k4 = (y2 - y0)/-a;
    if (k3 > k4){
        long long tmp = k3;
        k3 = k4;
        k4 = tmp;
    }
    long long res = min(k2, k4) - max(k1, k3);
    if (x0 >= xx1 && x0 <= xx2 && y0 >= yy1 && y0 <= yy2)   res ++;
    return res < 0 ? 0 : res;
}
long long f(long long a, long long b, long long c, long long x1, long long x2, long long y1, long long
y2){
    c *= -1;    //把c移到等式右边构成ax+by=c

    //特殊处理a=b=0,a=0 b!=0,a!=0 b=0的情况
    if (a == 0 && b == 0){
        if (c == 0)
            return (x2 - x1 + 1) * (y2 - y1 + 1);
        else
            return 0;
    }
    if (a == 0){
        if (c % b == 0)
            if (c/b >= y1 && c/b <= y2)
```

```
                return (x2 - x1 + 1);
            else
                return 0;
        else
            return 0;
    }
    if (b == 0){
        if (c % a == 0)
            if (c/a >= x1 && c/a <= x2)
                return (y2 - y1 + 1);
            else
                return 0;
        else
            return 0;
    }

    //一般情况:
    long long x0, y0, flagx = 1, flagy = 1, g = gcd(a, b);
    if (c % g != 0)
        return 0;
    a /= g;
    if (a < 0)
        a = -1*a, flagx = -1;    //把a,b都变正数(不过感觉不变也行……囧)
    b /= g;
    if (b < 0)
        b = -1*b, flagy = -1;
    c /= g;
    ext_gcd(a, b, x0, y0);

    x0 *= (c * flagx);       //解出了一组特殊解x0, y0
    y0 *= (c * flagy);

    a *= flagx;
    b *= flagy;
    return num_of_equation(x0, y0, x1, y1, x2, y2, a, b);
}
int main(){
    long long a, b, c, x1, x2, y1, y2;
    cin >> a >> b >> c >> x1 >> x2 >> y1 >> y2;
    cout << f(a, b, c, x1, x2, y1, y2) << endl;
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

Gauss

```
/*-------------------------------------------------------------------------------------------------------------*/
int gcd(int a, int b){
    return b ? gcd(b, a%b) : a;
}
int lcm(int a, int b){
```

```
    return a / gcd(a,b) * b;
}

/*  Gauss解整数线性方程组(浮点数的求法类似，但是要在判断是否为0时，加入EPS，以消除精度问题。)  */
//O(N^3)
//定义方程结构体EQU
//函数gauss()即为高斯消元法过程。return 0表示有唯一解，解向量存在x[]中；  return -2表示有实数解但无整数解。

const int MAX_EQU = 230;
typedef struct  equation_set{
    int equ, var;                       //方程个数、变元个数，则增广矩阵有equ行、var+1列.
    int mat[MAX_EQU][MAX_EQU+1];        //增广矩阵
    int x[MAX_EQU];                             //方程解向量
    bool free_x[MAX_EQU];               //判断是否是不确定变元

    void init(int in_equ, int in_var);
    void debug();
    int gauss();
}EQU;
void equation_set::init(int in_equ, int in_var){
    equ = in_equ;
        var = in_var;
    memset(mat, 0, sizeof(mat));
    memset(x, 0, sizeof(x));
    memset(free_x, 1, sizeof(free_x));
}
void equation_set::debug(){
    for (int i = 0; i < equ; i ++){
        for (int j = 0; j < var; j ++)
            cout << mat[i][j] << " ";
        cout << mat[i][var] << endl;
    }
}
int equation_set::gauss(){
    int row = 0, col = 0;
    int max_row;
    while (row < equ && col < var){
        max_row = row;
        for (int i = row + 1; i < equ; i ++)
            if (abs(mat[i][col]) > abs(mat[max_row][col]))
                max_row = i;
        if (max_row != row){
            for (int j = 0; j <= var; j ++){
                int tmp = mat[row][j];
                mat[row][j] = mat[max_row][j];
                mat[max_row][j] = tmp;
            }
        }
        if (mat[row][col] == 0){
            col ++;
            continue;
        }
```

```
                //普通方程组行阶梯化:
        for (int i = row + 1; i < equ; i ++){
            if (mat[i][col] != 0){
                int LCM = lcm(abs(mat[i][col]), abs(mat[row][col]));
                int ta = LCM  / abs(mat[i][col]), tb = LCM / abs(mat[row][col]);
                if (mat[i][col] * mat[row][col] < 0)    tb = -tb;
                for (int j = col; j <= var; j ++)
                    mat[i][j] = mat[i][j] * ta - mat[row][j] * tb;
            }
        }

        row++, col++;
    }
    //debug();

    //1. 无解
    for (int i = row; i < equ; i++){
        if (mat[i][col] != 0) return -1;
    }
    //2.无穷解(这里只是计算出了确定变元，对于多解的计算，需要枚举自由变元的状态或者其他方法求解.)
    if (row < var){
                //计算确定变元，大多数题目不需要
        int free_x_num = 0;
        int free_index;
        for (int i = row - 1; i >= 0; i --){
            for (int j = i; j < var; j ++)
                if (mat[i][j] != 0 && free_x[j])    free_x_num ++, free_index = j;
            if (free_x_num > 1) continue;
            int tmp = mat[i][var];
            for (int j = 0; j < var; j ++)
                if (free_index != j)    tmp -= mat[i][j] * x[j];
            if (tmp % mat[i][free_index] != 0)  return -2;  //无整数解，看题目有没有需要吧。
            x[free_index] = tmp / mat[i][free_index];
            free_x[free_index] = 0;                                         //该列是确定变元
        }
        return var - row;
    }
    //3.唯一解
    for (int i = row - 1; i >= 0; i --){
        int tmp = mat[i][var];
        for (int j = var - 1; j > i; j --)
            tmp -= x[j] * mat[i][j];
        x[i] = tmp / mat[i][i];
    }
    return 0;
}

/* ----------------------- 高斯消元解异或方程组+枚举自由元求解 --- POJ 3185 ----------------------- */
#define MAX 25
typedef struct equation_set{
    int equ, var;
    int a[MAX][MAX];
```

```
    int x[MAX];

    void init(){
        equ = var = 20;
        memset(a, 0, sizeof(a));
        memset(x, 0, sizeof(x));
        for (int i = 0; i < 20; i ++){
            a[i][i] = 1;
            if (i > 0)
                a[i-1][i] = 1;
            if (i < 19)
                a[i+1][i] = 1;
        }
        return ;
    }
    void debug();
    int gauss();
    int dfs(int n, int row);
}EQU;
void EQU::debug(){
    for (int i = 0; i < 20; i ++){
        for (int j = 0; j < 20; j ++)
            cout << a[i][j] << " ";
        cout << a[i][20] << endl;
    }
}

//dfs枚举自由元并求出所有解
int EQU::dfs(int n, int row){
    if (n == var){
        for (int i = row - 1; i >= 0; i --){
            int tmp = a[i][var];
            for (int j = i + 1; j < var; j ++)
                tmp ^= (a[i][j] && x[j]);
            x[i] = tmp;
        }
        int ans = 0;
        for (int i = 0; i < var; i ++)
            if (x[i])   ans ++;
        return ans;
    }
    x[n] = 0;
    int p1 = dfs(n+1, row);
    x[n] = 1;
    int p2 = dfs(n+1, row);
    return min(p1, p2);
}
int EQU::gauss(){
    int row = 0, col = 0;
    while(row < equ && col < var){
        int max_row = row;
        for (int i = row + 1; i < equ; i ++)
```

```
            if (a[i][col] > a[row][col])    max_row = i;
        if (max_row != row){
            for (int j = 0; j <= var; j ++){
                int tmp = a[row][j];
                a[row][j] = a[max_row][j];
                a[max_row][j] = tmp;
            }
        }
        if (a[row][col] == 0){
            col ++;
            continue;
        }

        for (int i = row + 1; i < equ; i ++){
            if (a[i][col]){
                for (int j = col; j <= var; j ++)
                    a[i][j] = a[i][j] ^ a[row][j];
            }
        }

        row ++, col ++;
    }
    //debug();

        //无解
    for (int i = row; i < equ; i ++)
        if (a[i][col])
            return -1;

        //多解，dfs枚举自由元
    if (row < var){
        return dfs(row, row);
    }

        //唯一解
    for (int i = col - 1; i >= 0; i --){
        int tmp = a[i][var];
        for (int j = i+1; j < var; j ++)
            tmp ^= (a[i][j] && x[j]);
        x[i] = tmp;
    }

    int ans = 0;
    for (int i = 0; i < var; i ++)
        if (x[i])   ans ++;
    return ans;
}
int main(){
    EQU E;
    E.init();
    for (int i = 0; i < 20; i ++)
        cin >> E.a[i][20];
```

```
    //E.debug();
    cout << E.gauss() << endl;
        return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

Generating Function

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
        普通母函数的应用---整数拆分：
        HDU 1398 300以内完全平方数分值硬币：1元、4元、9元……有几种方法拼成n元(n <= 300)
        (1+x+x^2+x^3+x^4+x^5+……)(1+x^4+x^8+x^12+x^16+……)(1+x^9+x^18+x^27+……) = a1+a2*x+a3*x^2+……
+an*x^n+……
        x^n的系数即为所求
        res储存最后结果，每次乘上一个多项式pro
*/
int a[17] = {1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289};
/* generating_function */
const int MAX = 400;
struct generating_function{
    int res[MAX];
    int pro[MAX];
    int mmax;   //x^n --- max(n);
    void init(int n){
        memset(res, 0, sizeof(res));
        memset(pro, 0, sizeof(pro));
        res[0] = 1;
               mmax = n;
    }
    void cal(int p);
}gf;
void generating_function::cal(int p){
    int tmp[MAX];
    memset(tmp, 0, sizeof(tmp));
    for (int i = 0; i <= mmax; i ++){
        for (int j = 0; j <= mmax; j += a[p]){       //这里的范围可以根据当前pro优化一下.
            if (i + j <= mmax)
            tmp[i+j] +=  res[i] * pro[j];
        }
    }
        memset(res, 0, sizeof(res));
    for (int i = 0; i <= mmax; i ++){
        res[i] = tmp[i];
    }
    return ;
}
/* generating_function */
int main(){
    int n;
    while(scanf("%d", &n) == 1, n){
```

```
        gf.init(n);
        for (int i = 0; i < 17; i ++){
            memset(gf.pro, 0, sizeof(gf.pro));
            if (a[i] > gf.mmax)
                break;
            int p = 0;
            while(p <= gf.mmax){
                gf.pro[p] = 1;
                p += a[i];
            }
            gf.cal(i);
        }
        printf("%d\n", gf.res[gf.mmax]);
    }
    return 0;
}
/*
        HDU 2069 带限制条件的找硬币问题---整数划分问题：总数不能超过100
        方法：将母函数变形，加一维表示数量。
*/
int a[5] = {1, 5, 10, 25, 50};
const int N = 300;
struct generating_function{
    int c1[N][101], c2[N][101];
    int maxn;
    void init(int n){
        memset(c1, 0, sizeof(c1));
        memset(c2, 0, sizeof(c2));
        c1[0][0] = 1;
        maxn = n;
    }
    void cal(int p);
}ef;
void generating_function::cal(int p){
    int tmp[N][101];
    memset(tmp, 0, sizeof(tmp));
    for (int i = 0; i <= maxn; i ++){
        for (int j = 0; j <= maxn; j += a[p]){
            int k1 = j/a[p];
            if (i + j <= maxn && c2[j][k1])
                for (int k2 = 0; k1+k2 <= 100; k2 ++)
                    tmp[i+j][k1+k2] += c1[i][k2] * c2[j][k1];
        }
    }
    memset(c1, 0, sizeof(c1));
    for (int i = 0; i <= maxn; i ++){
        for (int k = 0; k <= 100; k ++){
            c1[i][k] = tmp[i][k];
        }
    }
    return ;
}
```

```
int main(){
    int n;
    while(scanf("%d", &n) == 1){
        ef.init(n);
        for (int i = 0; i < 5; i ++){
            memset(ef.c2, 0, sizeof(ef.c2));
            for (int j = 0; j <= n; j += a[i]){
                ef.c2[j][j/a[i]] = 1;
            }
            ef.cal(i);
        }

        int res = 0;
        for (int k = 0; k <= 100; k ++){
            res += ef.c1[n][k];
        }
        printf("%d\n", res);
    }
        return 0;
}
/*
        指数型母函数：
        HDU 2065 “红色病毒”问题 (指数母函数 && 泰勒级数)
        由4种字母组成，A和C只能出现偶数次。
        构造指数级生成函数：(1+x/1!+x^2/2!+x^3/3!……)^2*(1+x^2/2!+x^4/4!+x^6/6!……)^2.
        前面是B和D的情况，可以任意取，但是相同字母一样，所以要除去排列数。后者是A和C的情况，只能取偶数个情
况。
        根据泰勒展开，e^x在x0=0点的n阶泰勒多项式为 1+x/1!+x^2/2!+x^3/3!……
        而后者也可以进行调整，需要把奇数项去掉，则e^(-x)的展开式为1-x/1!+X^2/2!-X^3/3!……
        所以后者可以化简为(e^x+e^(-x))/2。则原式为 (e^x)^2   *  ((e^x*e^(-x))/2)^2
        整理得到1/4*(e^4x+2*e^2x+1)。
        又由上面的泰勒展开
        e^4x = 1 + (4x)/1! + (4x)^2/2! + (4x)^3/3! + ... + (4x)^n/n!;
        e^2x = 1 + (2x)/1! + (2x)^2/2! + (2x)^3/3! + ... + (2x)^n/n!;
        对于系数为n的系数为(4^n+2*2^n)/4=4^(n-1)+2^（n-1）;
        快速幂搞之。
*/
int PowMod(int a,LL b){
    int ret=1;
    while(b){
        if(b&1)
            ret=(ret*a)%MOD;
        a=(a*a)%MOD;
        b>>=1;
    }
    return ret;
}
int main(){
    int t;
    while(scanf("%d",&t)!=EOF&&t){
        int cas=0;
        LL n;
```

```
        while(t--){
            scanf("%I64d",&n);
            printf("Case %d: %d\n",++cas,(PowMod(4,n-1)+PowMod(2,n-1))%MOD);
        }
        printf("\n");
    }
    return 0;
}
/*-----------------------------------------------------------------------------------------------------------------*/
```

归并排序求逆序数

```
/*-----------------------------------------------------------------------------------------------------------------*/
/*
序列逆序数：由1，2，…，n组成的一个有序数组称为一个n级排列。在一个排列中如果一对数的前后位置与大小顺序相反，即大数排在小数的前面，则称它们为一个逆序。一个排列中所有逆序的总和称为该排列的逆序数。

归并求逆序简单原理：
        归并排序是分治的思想，具体原理自己去看书吧。利用归并求逆序是指在对子序列 s1和s2在归并时，若s1[i]>s2[j]（逆序状况），则逆序数加上s1.length-i,因为s1中i后面的数字对于s2[j]都是逆序的。
*/
const int N = 1001;
int a[N];
int res;
void mmerge(int a[], int p, int q, int r){
    int n1 = q - p + 1;
    int n2 = r - q;
    int n = r - p + 1;
    int tmp[n], k = 0;
    int left[n1], right[n2];
    for (int i = 0; i < n1; i ++)
        left[i] = a[p + i];
    for (int i = 0; i < n2; i ++)
        right[i] = a[q + 1 + i];
    int i1 = 0, i2 = 0;
    while(i1 < n1 && i2 < n2){
        if (left[i1] <= right[i2]){
            tmp[k ++] = left[i1 ++];
        }
        else if (left[i1] > right[i2]){
            tmp[k ++] = right[i2 ++];
            res += n1 - i1;
        }
    }
    while (i1 < n1){
        tmp[k ++] = left[i1 ++];
    }
    while (i2 < n2){
        tmp[k ++] = right[i2 ++];
    }
    for (int i = 0; i < n; i ++){
```

```
        a[p + i] = tmp[i];
    }
}
void merge_sort(int a[], int l, int r){
    if (r <= l)
        return ;
    int mid = MID(l, r);
    merge_sort(a, l, mid);
    merge_sort(a, mid+1, r);
    mmerge(a, l, mid, r);
    return ;
}
int main(){
    int t;
    scanf("%d\n", &t);
    for (int caseo = 1; caseo <= t; caseo ++){
        int n;
        scanf("%d", &n);
        mem(a, 0);
        for (int i = 0; i < n; i ++)
            scanf("%d", &a[i]);
        res = 0;
        merge_sort(a, 0, n - 1);
        printf("Scenario #%d:\n", caseo);
        printf("%d\n\n", res);
    }
        return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

科学计数法

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
        应用:  ①取一个数的前几位(double精度以内);
*/
//n是原数，可以是整数或实数;  _float是科学计数法的小数，_exp是10的指数
void scientific_notation(long long n, double &_float, int &_exp){
    double log10_n = log10(n);
    _exp = (int)log10_n;
    log10_n = log10_n - _exp;
    _float = pow(10, log10_n);
    return ;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

三分求函数极值

```
/*-------------------------------------------------------------------------------------------------------------*/
```

```
//如果一个解函数的二次导恒>0(or <0)，则该函数为凸函数.
//精度模板
const double eps = 1e-6;
bool dy(double x,double y)  {   return x > y + eps;}         // x > y
bool xy(double x,double y)  {   return x < y - eps;}         // x < y
bool dyd(double x,double y) {   return x > y - eps;}         // x >= y
bool xyd(double x,double y) {   return x < y + eps;}         // x <= y
bool dd(double x,double y)  {   return fabs( x - y ) < eps;}    // x == y

double cal(double pos){
    /* 根据题目的意思计算 */
}

double solve(double left, double right){
    while(xy(left, right)){
        double mid = DMID(left, right);
        double midmid = DMID(mid, right);
        double mid_area = cal(mid);
        double midmid_area = cal(midmid);
        if (xyd(mid_area, midmid_area))     //下凸函数，上凸函数用dyd(mid_area, midmid_area)
            right = midmid;
        else
            left = mid;
    }
    return left;
}
/*-------------------------------------------------------------------------------------------------------------------*/
```

Data Structure

Disjoint Sets

```
/*-------------------------------------------------------------------------------------------------------------------*/
//Common Disjoint Sets
const int MAXN = 505;
struct Disjoint_Sets{
    int father, ranks;
}S[MAXN];
void init(){
    for (int i = 0; i < MAXN; i ++)
        S[i].father = i, S[i].ranks = 0;
}
int Father(int x){
    if (S[x].father == x){
        return x;
    }
    else{
```

```
        S[x].father = Father(S[x].father);                    //Path compression
        return S[x].father;
    }
}
bool Union(int x, int y){
    int fx = Father(x), fy = Father(y);
    if (fx == fy){
        return false;
    }
    else{                                              //Rank merge
        if (S[fx].ranks > S[fy].ranks){
            S[fy].father = fx;
        }
        else{
            S[fx].father = fy;
            if (S[fx].ranks == S[fy].ranks){
                ++ S[fy].ranks;
            }
        }
        return true;
    }
}

//种类并查集
const int N=100005;
struct set
{
    int parent;        //记录父节点
    int rank;          //记录集合的节点数
    int relation;      //节点与根节点的关系
}elem[N];

int MAX;      //最大集的元素个数
void init()
{
    int i;
    for(i=0;i<=N;i++)
    {
        elem[i].parent=i;
        elem[i].rank=1;
        elem[i].relation=0;
    }
}
int Find(int x)
{
    int temp;
    if (elem[x].parent != x)                           //路径压缩
    {
        temp=elem[x].parent;
        elem[x].parent = Find(elem[x].parent);
        elem[x].relation=(elem[temp].relation + elem[x].relation)%3;
    }
```

```
    return elem[x].parent;
}
void Union(int d,int a,int b)                    //合并两个集合
{
    int x,y;
    x=Find(a);
    y=Find(b);
    elem[x].parent=y;
    elem[x].relation=(elem[b].relation-elem[a].relation+2+d)%3;
}
/*-------------------------------------------------------------------------------------------------------------------*/
```

单调栈：求 01 矩阵中最大的全 1 矩阵

```
/*-------------------------------------------------------------------------------------------------------------------*/
//POJ 3494 --- 求01矩阵中最大的全1矩阵
//参看09年国家队朱晨光论文《基本数据结构在信息学竞赛中的应用》
int height[2005], a[2005][2005];      //height[]为悬线
int left[2005];                                  //悬线左扩展
stack <int> S;
int main(){
    int n, m;
    while(scanf("%d %d", &n, &m) != EOF){
        int res = 0;
        memset(height, 0, sizeof(height));
        for (int i = 0; i < n; i ++)
            for (int j = 0; j < m; j ++)
                scanf("%d", &a[i][j]);
        for (int i = 0; i < n; i ++){
            while(!S.empty())
                S.pop();
            for (int j = 0; j <= m; j ++){
                if (j == m){
                    height[j] = -1;
                }
                else{
                    if (a[i][j] == 0)
                        height[j] = 0;
                    else
                        height[j] ++;
                }

                if (S.empty() || height[S.top()] < height[j]){
                    if (S.empty())
                        left[j] = -1;
                    else    left[j] = S.top();
                    S.push(j);
                }
                else{
                    int ok = 0;
                    while(!S.empty() && height[S.top()] >= height[j]){
                        if (height[S.top()] == height[j]){
```

```
                        left[j] = left[S.top()];
                        S.top() = j;
                        ok = 1;
                        break;
                    }
                    while(!S.empty() && height[S.top()] > height[j]){
                        res = max(res, height[S.top()] * (j - left[S.top()] - 1));
                        S.pop();
                    }
                    if (!ok){
                        if (S.empty())
                            left[j] = -1;
                        else    left[j] = S.top();
                        S.push(j);
                    }
                }
            }
        }
        printf("%d\n", res);
    }
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

Trie

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
字典树，又称单词查找树，Trie树，是一种树形结构，典型应用是用于统计，排序和保存大量的字符串，所以经常被搜索引擎系统用于文本词频统计。它的优点是：利用字符串的公共前缀来节约存储空间，最大限度的减少无谓的字符串比较，查询效率比哈希表高。

字典树的应用：
        字符串的快速检索
        哈希
        最长公共前缀
*/
int words_num;

struct Trie_node
{
    int Ch_Count;   //统计单词个数，如果为0表示没有该串。
    int hash;       //单词hash后的标识数
    Trie_node *next[26];//指向各个子树的指针,下标0-25代表26字符,如果是数字改成10就行了
    Trie_node()
    {
        Ch_Count=0;
        hash=-1;
        memset(next,NULL,sizeof(next));
    }
```

```
};

class Trie
{
public:
    Trie();
    void insert(char *word);                //插入新单词
    int search(char * word);                //返回单词个数，如果为0表示不存在该单词
private:
    Trie_node* root;
};

Trie::Trie()
{
    root = new Trie_node();
}

void Trie::insert(char * word)              //有时候可以插入查询一块儿做
{
    Trie_node *p = root;
    int len=strlen(word);
    if(len==0)return ;
    for (int i=0;i<len;i++)
    {
        if(p->next[word[i]-'a'] == NULL)      //如果不存在的话，我们就建立一个新的节点
        {
            Trie_node *tmp = new Trie_node();
            p->next[word[i]-'a'] = tmp;
            p = p->next[word[i]-'a'];         //每插入一步，相当于有一个新串经过，指针要向下移动
        }
        else                                 //如果这个节点之前就已经存在呃，我们只需要把统计次数加上1
            p=p->next[word[i]-'a'];
        //p->Ch_Count++;                       //这里是求所有前缀出现的次数，如果只求整个单词出现次数则用后
一个
    }
    p->Ch_Count++;                          //求整个单词的出现次数
    /*
    if (p->hash<0)
        p->hash=words_num++;
    */
    return ;
}

int Trie::search(char * word)               //返回单词个数，如果为0表示不存在该单词
{
    Trie_node *p = root;
    int len=strlen(word);
    if (len==0) return 0;
    for (int i=0;i<len;i++)
    {
        if (p->next[word[i]-'a']!=NULL)
            p=p->next[word[i]-'a'];
```

```
        else
            return 0;
    }
    return p->Ch_Count;
    //return p->hash;                       //返回单词的hash标识数，如果为0表示不存在该单词
}

int main()              //简单测试
{
    Trie t;
    t.insert("a");
    t.insert("abandon");
    string c= "abandoned";
    t.insert(c);
    t.insert("abashed");
    if(t.search("abashed"))
        printf("abashed true\n");
    if(t.search("a"))
        printf("a true\n");
    if(t.search("abandoned"))
        printf("abandoned true\n");
    if(t.search("aba"))                     //看前缀统计不统计
        printf("aba true\n");
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------------*/
```

SBT

```
/*-------------------------------------------------------------------------------------------------------------------*/
const int maxn= 100010;
class SBT{
public:
    void Clear(){
        memset(K,0,sizeof(K));
        memset(L,0,sizeof(L));
        memset(R,0,sizeof(R));
        memset(S,0,sizeof(S));
        RT= SZ= 0;
    }
    void MLR()  { printf("----------------\n"); MLR(RT);}    //前序遍历SBTree，每行输出（当前子树的根，~左子树，~右子树------输出方式可自己更改）
    void Insert(int key)    {Insert(RT,key);}               //在SBTree中插入键值为v的结点
    int Delete(int key)     {return Delete(RT,key);}        //在SBTree中删除键值为v的结点
    int Succ(int key)       {return Succ(RT,key);}          //返回SBTree中比v大的最小的键值
    int Pred(int key)       {return Pred(RT,key);}          //返回SBTree中比v小的最大的键值
    int Rank(int key)       {return Rank(RT,key);}          //返回SBTree中键值v的排名。也就是树中键值比v小的结点数+1
    int Find(int key)       {return Find(RT,key);}          //在SBTree中查找键值为v的结点
    int Select(int key)     {return Select(RT,key);}        //返回SBTree中排名为k的结点的值。同时该操作能够实现Get-min,Get-max,因为Get-min等于Select(1),Get-max等于Select(s[t])
```

```
private:
    int K[maxn];                //键值
    int L[maxn];                //左子树结点
    int R[maxn];                //右子树结点
    int S[maxn];                //子树结点个数
    int RT, SZ;
    void LeftRotate(int &x){
        int k= R[x];
        R[x]= L[k];
        L[k]=  x;
        S[k]= S[x];
        S[x]= S[L[x]]+S[R[x]]+1;
        x= k;
    }
    void RightRotate(int &x){
        int k= L[x];
        L[x]= R[k];
        R[k]= x;
        S[k]= S[x];
        S[x]= S[L[x]]+S[R[x]]+1;
        x= k;
    }
     void MaintainFat(int &t){
        if (S[L[L[t]]]>S[R[t]]){
            RightRotate(t);
            MaintainFat(R[t]);
            MaintainFat(t);
            return;
        }
        if (S[R[L[t]]]>S[R[t]]){
            LeftRotate(L[t]);
            RightRotate(t);
            MaintainFat(L[t]);
            MaintainFat(R[t]);
            MaintainFat(t);
            return;
        }
        if (S[R[R[t]]]>S[L[t]]){
            LeftRotate(t);
            MaintainFat(L[t]);
            MaintainFat(t);
            return;
        }
        if (S[L[R[t]]]>S[L[t]]){
            RightRotate(R[t]);
            LeftRotate(t);
            MaintainFat(L[t]);
            MaintainFat(R[t]);
            MaintainFat(t);
            return;
        }
```

```
}
void Maintain(int &t, int flag){
    if (!flag){
        if (S[L[L[t]]] >S[R[t]])
            RightRotate(t);
        else if (S[R[L[t]]] >S[R[t]]){
            LeftRotate(L[t]);
            RightRotate(t);
        }
        else
            return;
    }
    else{
        if (S[R[R[t]]] >S[L[t]])
            LeftRotate(t);
        else if (S[L[R[t]]] >S[L[t]]){
            RightRotate(R[t]);
            LeftRotate(t);
        }
        else
            return;
    }
    Maintain(L[t], false);
    Maintain(R[t], true);
    Maintain(t, true);
    Maintain(t, false);
}
void Insert(int &t, int key){
    if (t==0){
        t= ++SZ;
        K[t]= key;
        S[t]= 1;
        return;
    }
    S[t]++;
    if (key<K[t])
        Insert(L[t],key);
    else
        Insert(R[t],key);
    Maintain(t,key>K[t]);

}
int Delete(int &t, int key){
    S[t]--;
    if ((key==K[t])||(key<K[t]&&L[t]==0)||(key>K[t]&&R[t]==0)){
        int ret= K[t];
        if (L[t]==0 || R[t]==0)
            t= L[t]+R[t];       // T change to his Leftson or Rightson
        else
            K[t]= Delete(L[t], K[t]+1); // Not find then delete the last find Point
        return ret;
```

```
        }
        else{
            if (key<K[t])
                return Delete(L[t],key);
            else
                return Delete(R[t],key);
        }

    }
    int Find(int t, int key){     //return root point
        if (t==0 || key==K[t])
            return t;
        if (key<K[t])
            return Find(L[t],key);
        else
            return Find(R[t],key);

    }
    int Select(int t, int k){       // return K-th int tree
        int num= S[L[t]]+1;
        if (k==num)
            return K[t];
        else if (k<num)
            return Select(L[t],k);
        else
            return Select(R[t],k-num);

    }
    int Succ(int t, int key){
        if (t==0)
            return key;
        if (key>=K[t])                  //
            return Succ(R[t], key);
        else{
            int r= Succ(L[t], key);
            if (r==key)
                return K[t];
            else
                return r;
        }

    }
    int Pred(int t, int key){
        if (t==0)
            return key;
        if (key<=K[t])                  //
            return Pred(L[t], key);
        else {
            int r= Pred(R[t], key);
            if (r==key)
                return K[t];
            else
```

```
                return r;
        }

    }
    int Rank(int t, int key){
        if (t==0)
            return 1;
        if (key<=K[t])           //
            return Rank(L[t], key);
        else if (key>K[t])
            return S[L[t]]+1+Rank(R[t],key);

    }
    void MLR(int t){
        if (t==0)   return;
        printf(" %d  % d  %d\n", K[t], K[L[t]], K[R[t]] );
        MLR(L[t]);
        MLR(R[t]);
    }


};
/*-------------------------------------------------------------------------------------------------------------*/
```

K 路归并

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
        k路归并：即给定N个有序的序列，要求把这N个有序的序列合并成一个有序的序列。
        分析:
                数据很大，但是只要求求出前N个，所以可以想到用堆优化
                每个表的元素都是从左到右移入新表
                把每个表的当前元素放入二叉堆中，每次删除最小值并放入新表中，然后加入此序列的下一个元素
                每次操作需要logk时间，因此总共需要nlogk的时间
                可以把这些和看成n个有序表:
                A[1]+B[1] <= A[1]+B[2] <= A[1]+B[3] <=…
                A[2]+B[1] <= A[2]+B[2] <= A[2]+B[3] <=…
                …
                A[n]+B[1] <= A[n]+B[2] <= A[n]+B[3] <=…

        综上所述，可以采用K路归并：

        POJ 2442 将M个列表合并取新表前N个元素
        方法：将M个列表不断的用两个列表K路归并的方法求。
*/
int m,n;
int a[2005], b[2005], tm[2005];

struct condition{
    int a, b;   //表示当前key是由A[a] + B[b]得来
    int key;
```

```
    friend bool operator < (condition n1, condition n2){
        return n2.key < n1.key;
    }
};

void k_merge(int A[], int B[], int n){      //K路归并,筛出前n个小的元素放回A[],A[]、B[]也只留n个即可
    priority_queue <condition , vector <condition> > Q;
    for (int i = 0; i < n; i ++){
        condition tmp;
        tmp.key = A[i] + B[0];
        tmp.a = i;
        tmp.b = 0;
        Q.push(tmp);
    }
    mem(tm, 0);
    for (int i = 0; i < n; i ++){
        condition t = Q.top();
        tm[i] = t.key;
        Q.pop();
        t.b ++;
        t.key = A[t.a] + B[t.b];
        Q.push(t);
    }
    for (int i = 0; i < n; i ++)
        A[i] = tm[i];
}

int main(){
    int t;
    scanf("%d", &t);
    while(t --){
        scanf("%d %d", &m, &n);
        for (int i = 0; i < n; i ++)
            scanf("%d", &a[i]);
        sort(a, a+n);
        for (int i = 1; i < m; i ++){
            for (int j = 0; j < n; j ++)
                scanf("%d", &b[j]);
            sort(b, b+n);
            k_merge(a, b, n);
        }
        for (int i = 0; i < n - 1; i ++)
            printf("%d ", a[i]);
        printf("%d\n", a[n-1]);
    }

    return 0;
}
/*--------------------------------------------------------------------------------------------------------------*/
```

LCA Tarjan

```
/*-----------------------------------------------------------------------------------------------------------*/
/*
        LCA Tarjan模板 --- POJ 1330.
        调用方法:离线把树的邻接表读进V.询问读进Q(Q[u],Q[v]两边都要存进去).然后对树根root调用LCA(root)即可得出这颗树的所有答案。当存在多个询问时可以考虑用map< pair<int ,int> int > ans;存储答案，不过同时也要注意pair<u,v>和pair<v,u>都要存。
*/
const int maxn = 10005;
struct Disjoint_Sets
{
    int parent;
    int rank;
}elem[maxn];
void init()
{
    for(int i = 0;i < maxn; i ++)
        elem[i].parent = i,    elem[i].rank = 1;
}
int Find(int x)
{
    if (elem[x].parent != x)                          //路径压缩
        elem[x].parent = Find(elem[x].parent);
    return elem[x].parent;
}
void Union(int a,int b)   //合并两个集合
{
    int x,y;
    x = Find(a),  y = Find(b);
    if(elem[x].rank >= elem[y].rank)
    {
        elem[y].parent = elem[x].parent;
        elem[x].rank += elem[y].rank;
    }
    else
    {
        elem[x].parent = elem[y].parent;
        elem[y].rank += elem[x].rank;
    }
}
//LCA
int ancestor[maxn];
bool vis[maxn];
vector <int> Q[maxn];
vector <int> V[maxn];
struct Query
{
    int u,v;
}que[maxn];
void initLCA()
{
```

```
    memset(ancestor,0,sizeof(ancestor));
    memset(vis,0,sizeof(vis));
    for (int i = 0; i < maxn; i ++)
    {
        V[i].clear();
        Q[i].clear();
    }
}
void LCA(int u)
{
    ancestor[u] = u;
    for (size_t i = 0; i < V[u].size(); i ++)
    {
        int v = V[u][i];
        LCA(v);
        Union(u,v);
        ancestor[Find(u)] = u;
    }
    vis[u] = 1;
    for (size_t i = 0; i < Q[u].size(); i ++)
        if (vis[Q[u][i]])
        {
            printf("%d\n",ancestor[Find(Q[u][i])]);
            return;
        }
}
int indegree[maxn];
int main()
{
    //freopen("data.txt","r+",stdin);
    int t;
    scanf("%d",&t);
    int n;
    while(t--)
    {
        init();
        initLCA();
        memset(indegree,0,sizeof(indegree));
        scanf("%d",&n);
        for (int i = 0; i < n - 1; i ++)
        {
            int s,t;
            scanf("%d%d",&s,&t);
            V[s].push_back(t);
            indegree[t] ++;
        }
        int s,t;
        scanf("%d%d",&s,&t);
        Q[s].push_back(t);
        Q[t].push_back(s);
        for (int i = 1; i <= n; i ++)
            if (indegree[i] == 0)
```

```
            {
                LCA(i);
                break;
            }
    }
    return 0;
}
/*-----------------------------------------------------------------------------------------------------------*/
```

函数式线段树（不带修改的区间第 K 小值）

```
/*-----------------------------------------------------------------------------------------------------------*/
const int MAXN = 100010;
struct tree
{
    int l,r;
    int ls,rs;
    int sum;
}t[MAXN*20];
int tot,root[MAXN];

int build(int l, int r)
{
    int k = ++ tot;
    t[k].l = l, t[k].r = r;
    t[k].sum = 0;
    if (l == r) return k;
    int mid = MID(l,r);
    t[k].ls = build(l,mid);
    t[k].rs = build(mid+1,r);
    return k;
}
int change(int o, int x, int v)
{
    int k = ++ tot;
    t[k] = t[o];
    t[k].sum += v;
    if (t[o].l == x && t[o].r == x)
        return k;
    int mid = MID(t[o].l, t[o].r);
    if (x <= mid)   t[k].ls = change(t[o].ls, x, v);
    else    t[k].rs = change(t[o].rs, x, v);
    return k;
}
int query(int n,int o,int k)  //询问区间[t1,t2]第k小
{
    if (t[n].l == t[n].r) return t[n].l;
    int res = t[t[n].ls].sum-t[t[o].ls].sum;
    if (k <= res)
        return query(t[n].ls,t[o].ls,k);
    else return query(t[n].rs,t[o].rs,k-res);
```

```
}

int b[MAXN],sortb[MAXN];
int q;
int main()
{
    //freopen("test.in","r+",stdin);
    int n,m;
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        for (int i = 1; i <= n; i ++)
            scanf("%d",&b[i]),sortb[i]=b[i];
        sort(sortb+1, sortb+n+1);
        int i;
        for (q = 1, i = 2; i <= n; ++ i)    //去重
            if (sortb[q] != sortb[i])
                sortb[++q] = sortb[i];
        root[0] = build(1,q);
        for (int i = 1; i <= n; i ++)
        {
            int p = lower_bound(sortb+1, sortb+n+1, b[i])-sortb;
            root[i] = change(root[i-1], p, 1);
        }

        for (int i = 0; i < m; i ++)
        {
            int a,b,k;
            scanf("%d%d%d",&a,&b,&k);
            printf("%d\n",sortb[query(root[b],root[a-1],k)]);
        }

    }
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------------*/
```

KD-Tree

```
/*-------------------------------------------------------------------------------------------------------------------*/
const int N=100005;
LL res;
struct Point
{
        int x, y;               //点是二维的，此时是2D-Tree
};

LL dist2(const Point &a, const Point &b)                    //距离的平方
{
        return LL(a.x - b.x) * LL(a.x - b.x) + LL(a.y - b.y) * LL(a.y - b.y);
}
```

```
bool cmpX(const Point &a, const Point &b)
{
    return a.x < b.x;
}
bool cmpY(const Point &a, const Point &b)
{
    return a.y < b.y;
}

struct KDTree          //很崇拜这种销魂的建树方法啊~0.0~很抽象很强大------p数组已经代表了KD-Tree了，神马左右子树全省了，OOOOOrrz!!!……
{
        Point p[N];            //空间内的点
        int Div[N];            //记录区间是按什么方式划分（分割线平行于x轴还是y轴, ==1平行y轴切;==0平行x轴切）

        void build(int l, int r)                  //记得先把p备份一下。
        {
                if (l > r)      return;
                int mid=MID(l, r);
                int minX, minY, maxX, maxY;
                minX = min_element(p + l, p + r + 1, cmpX)->x;
                minY = min_element(p + l, p + r + 1, cmpY)->y;
                maxX = max_element(p + l, p + r + 1, cmpX)->x;
                maxY = max_element(p + l, p + r + 1, cmpY)->y;
                Div[mid] = (maxX - minX >= maxY - minY);
                nth_element(p + l, p + mid, p + r + 1, Div[mid] ? cmpX : cmpY);
                build(l, mid - 1);
                build(mid+1, r);
        }

        void find(int l, int r, Point a)                              //查找最近点的平方距离
        {
                if (l > r)      return;
                int mid = MID(l, r);
                LL dist = dist2(a, p[mid]);
                if (dist > 0)   //如果有重点不能这么判断
                        res = min(res, dist);
                LL d = Div[mid] ? (a.x - p[mid].x) : (a.y - p[mid].y);
                int l1, l2, r1, r2;
                l1 = l , l2 = mid + 1;
                r1 = mid - 1, r2 = r;
                if (d > 0)
                        swap(l1, l2), swap(r1, r2);
                find(l1, r1, a);
                if (d * d < res)
                        find(l2, r2, a);
        }
};

Point pp[N];
KDTree kd;
```

```
int main()
{
    //freopen("data.txt","r+",stdin);
    int t;
    scanf("%d", &t);
    while(t--)
    {
        int n;
        scanf("%d", &n);
        for (int i = 0; i < n; i++)
        {
            scanf("%d%d", &pp[i].x, &pp[i].y);
            kd.p[i] = pp[i];
        }
        kd.build(0, n-1);
        for (int i = 0; i < n; i++)
        {
            res = 9223372036854775807LL;
            kd.find(0, n - 1, pp[i]);
            printf("%I64d\n", res);
        }
    }
       return 0;
}
/*-------------------------------------------------------------------------------------------------------------------*/
```

DLX 数独

```
/*-------------------------------------------------------------------------------------------------------------------*/
//Dancing Links X解数独

//列(N+N+N)*N+N*N=4*N*N.
#define N 750   //>9*9*9
#define M 350   //>4*9*9
const int P=9;  //p阶数独
int h;
int L[N*M],R[N*M],U[N*M],D[N*M],S[N*M],C[N*M],O[N*M];

int n,m;
void remove (const int &c)
{
    L[R[c]]=L[c];
    R[L[c]]=R[c];
    for (int i=D[c];i!=c;i=D[i])
        for (int j=R[i];j!=i;j=R[j])
        {
            U[D[j]]=U[j];
            D[U[j]]=D[j];
            --S[C[j]];
        }
```

```
}

void resume (const int &c)
{
    for (int i=U[c];i!=c;i=U[i])
        for (int j=L[i];j!=i;j=L[j])
        {
            U[D[j]]=j;
            D[U[j]]=j;
            ++S[C[j]];
        }
    L[R[c]]=c;
    R[L[c]]=c;
}

bool Dance(int k)
{
    if (R[h]==h)
    {
        sort(O,O+k);
        for (int i=0;i<k;i++)
            printf("%d",(O[i]-1)/m%P==0?P:((O[i]-1)/m%P));
        printf("\n");

        return true;
    }

    int c,ss=INT_MAX;

    for (int i=R[h];i!=h;i=R[i])
        if (S[i]<ss)
        {
            ss=S[i];
            c=i;
        }

    remove(c);
    for (int i=D[c];i!=c;i=D[i])
    {
        O[k]=i;
        for (int j=R[i];j!=i;j=R[j])
            remove(C[j]);
        if (Dance(k+1))
            return true;
        for (int j=L[i];j!=i;j=L[j])
            resume(C[j]);
    }
    resume(c);

    return false;
}
```

```
//Initialize
void Link(char s[])
{
    int d[N][M];
    memset(d,0,sizeof(d));
    memset(0,0,sizeof(0));

    //preprocess the sudoku 数独转换精确覆盖问题矩阵形式
    int q=0;
    for (int i=1;i<=P;i++)
        for (int j=1;j<=P;j++)
        {
            if (s[q]=='.')
            {
                for (int k=1;k<=P;k++)
                {
                    int rr=((i-1)*P+j-1)*P+k;                    //行。(9*9*9)行 表示数独第i行第j列数填k
                    d[rr][(i-1)*P+k]=1;                          //列。前(9*9)列 表示数独第i行有数k
                    d[rr][(j-1)*P+k+P*P]=1;                      //列。(9*9)列 表示数独第j行有数k
                    d[rr][((i-1)/3*3+(j-1)/3)*P+k+2*P*P]=1;      //列。(9*9)列 表示数独第p个九宫格有数k
                    d[rr][(i-1)*P+j+3*P*P]=1;                    //列。(9*9)列 表示数独第i行第j行有一个数
（防止一个格子填多个数）
                }
            }
            else
            {
                int num=s[q]-'0';
                int rr=((i-1)*P+j-1)*P+num;                  //行。(9*9*9)行 表示数独第i行第j列数填k
                d[rr][(i-1)*P+num]=1;                        //列。前(9*9)列 表示数独第i行有数k
                d[rr][(j-1)*P+num+81]=1;                     //列。(9*9)列 表示数独第j行有数k
                d[rr][((i-1)/3*3+(j-1)/3)*P+num+162]=1;      //列。(9*9)列 表示数独第p个九宫格有数k
                d[rr][(i-1)*P+j+243]=1;                      //列。(9*9)列 表示数独第i行第j行有一个数（防止
一个格子填多个数）
            }

            q++;
        }

    //Initialize the all matrix to list.

    int x[N],row[N],col[M];          //x表示当前行中的第一个链,row表示当前行中上一个插入的链、col表示当前
列中上一个插入的链。
    h=0;
    for (int i=1;i<=m;i++)
    {
        R[i-1]=i;
        L[i]=i-1;
        S[i]=0;
        col[i]=i;
    }
    col[0]=0;
    L[h]=m;
```

```
    R[m]=h;

    for (int i=1;i<=n;i++)
    {
        x[i]=0;                                 //行第一个链表
        for (int j=1;j<=m;j++)
            if (d[i][j])
            {
                int index=i*(4*P*P)+j;          //带插入的列表下标。
                if (!x[i])
                {
                    row[i]=x[i]=index;
                }
                else
                {
                    R[row[i]]=index;
                    L[index]=row[i];
                }
                D[col[j]]=index;
                U[index]=col[j];
                row[i]=col[j]=index;
                C[index]=j;
                S[j]++;
            }
    }

    for (int i=1;i<=n;i++)
        if (x[i])
        {
            L[x[i]]=row[i];
            R[row[i]]=x[i];
        }
    for (int j=1;j<=m;j++)
    {
        D[col[j]]=j;
        U[j]=col[j];
    }
}

int main()
{
    //freopen("test.in","r+",stdin);
    //freopen("test.out","w+",stdout);
    char s[100];
    n=P*P*P;
    m=4*P*P;
    while(~scanf("%s",s))
    {
        if (!strcmp(s,"end"))
            return 0;
        Link(s);
        Dance(0);
```

```
    }
    return 0;
}
/*-----------------------------------------------------------------------------------------------------------*/
```

线段树（区间更新、区间查询）

```
/*-----------------------------------------------------------------------------------------------------------*/
const int N=100005;                          //结点个数

int M;
__int64 sum[N*4];
__int64 add[N*4];

void PushUp(int rt)                          //统计汇总，rt为当前节点
{
    sum[rt] = sum[rt<<1] + sum[rt<<1|1];     //如果是区间最值则 sum[n]=max(sum[n*2],sum[n*2+1])
}

void BuildTree(int n)
{
    mem(sum, 0);
    mem(add, 0);
}

void PushDown(int rt,int l)           //标记下放，rt为当前节点，l为修改区间长度
{
    if (add[rt])
    {
        add[rt<<1]+=add[rt];
        add[rt<<1|1]+=add[rt];
        sum[rt<<1]+=add[rt]*(l-(l>>1));
        sum[rt<<1|1]+=add[rt]*(l>>1);
        add[rt]=0;
    }
}

void Update(int s,int t,int v,int l,int r,int rt)
{
    if (s<=l && r<=t)
    {
        add[rt]+=v;
        sum[rt]+=v*(r-l+1);
        return ;
    }
    PushDown(rt,r-l+1);
    int m=(l+r)>>1;
    if (s<=m)   Update(s,t,v,l,m,rt<<1);
    if (m<t)    Update(s,t,v,m+1,r,rt<<1|1);
    PushUp(rt);
}
```

```
__int64 Query(int s,int t,int l,int r,int rt)
{
    if (s<=l && r<=t)
        return sum[rt];
    PushDown(rt,r-l+1);
    __int64 ans=0;
    int m=(l+r)>>1;
    if (s<=m)    ans+=Query(s,t,l,m,rt<<1);
    if (m<t)     ans+=Query(s,t,m+1,r,rt<<1|1);
    return ans;
}

int main()
{
    int n,q;
    scanf("%d%d",&n,&q);
    BuildTree(n);
    for (int i=0;i<q;i++)
    {
        char c;
        int a,b,d;
        scanf("%*c%c",&c);
        if (c=='Q')
        {
            scanf("%d%d",&a,&b);
            printf("%I64d\n",Query(a,b,1,M,1));
        }
        else
        {
            scanf("%d%d%d",&a,&b,&d);
            Update(a,b,d,1,M,1);
        }
    }
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

分组线段树

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
HDU 4288 Coder ★(2012 ACM/ICPC Asia Regional Chengdu Online)

问题抽象：分组线段树求和。

思路:离线(离散化+排序)维护5颗线段树。sum[rt][5]的每棵树表示区间的数以该区间左端为起点mod 5的余数，cnt[rt]表示区间数的数目。一开始不知道怎么动态地维护插入、删除数据的位置的模5的余数，比如一开始插入1、3、5，5是要求的，但要是再插入个2变成1、2、3、5，那么就变成3了。。。这个让我想了好久，后来经过一些提示终于想到了思路:每个叶节点的值都附在sum[rt][0]里，即上面说的，sum[rt][i]表示以该区间左端点为起点mod 5的余数。那么在向上统计汇总时怎么转化呢？
```

答案是：sum[结点][i]=sum[左儿子][i]+sum[右儿子][((i+5)-cnt[左儿子]%5)%5]。

什么意思呢？从sum的意义出发，左儿子的区间左端点和父节点是一样的，所以他们的余数等价；然而需要把右儿子的左端点与父节点等价起来。设父区间左端点为a，则右儿子区间左端点即为a+cnt[左儿子]。若右儿子(pos-a)%5==i，则把它放到父区间(pos-a-cnt[])%5== i-cnt[]%5== (保证大于等于0小于5) ((i+5)-cnt[]%5)%5。

另外要注意这里离散化的方法~~~lower_bound()函数可以很方便的找到数在原数组中的位置。（或者自己写一个二分也可以。。。）

```
*/
const int N=100005;
int M;
LL sum[N<<2][5];
int cnt[N<<2];

void BuildTree(int n){
    for (M=1;M<=n+2;M<<=1);
    for (int i=1;i<N<<2;i++){
        for (int j=0;j<5;j++)
            sum[i][j]=0;
        cnt[i]=0;
    }
}
void PushUp(int rt){
    cnt[rt]=cnt[rt<<1]+cnt[rt<<1|1];
    for (int i=0;i<5;i++)
         sum[rt][i]=sum[rt<<1][i]+sum[rt<<1|1][((i+5)-cnt[rt<<1]%5)%5];
}

void Update(int s,int num,int v,int l,int r,int rt){
    if (l==s && r==s){
        sum[rt][0]+=v*num;
        cnt[rt]+=num;
        return;
    }
    int mid=MID(l,r);
    if (s<=mid) Update(s,num,v,l,mid,rt<<1);
    else Update(s,num,v,mid+1,r,rt<<1|1);
    PushUp(rt);
}

char str[100005][5];
int pri[100005];
int a[100005];

int main(){
    //freopen("test.in","r+",stdin);
    int n;
    while(scanf("%d",&n)!=EOF){
        int tot=0;
        for (int i=0;i<n;i++){
```

```
            scanf("%s",str[i]);
            if (str[i][0]!='s'){
                scanf("%d",&pri[i]);
                a[tot++]=pri[i];
            }
        }
        sort(a,a+tot);
        BuildTree(tot);
        for (int i=0;i<n;i++){
            if (str[i][0]=='a'){
                int x=pri[i];
                int pos=lower_bound(a,a+tot,x)-a+1;
                Update(pos,1,x,1,M,1);
            }
            else if (str[i][0]=='d'){
                int x=pri[i];
                int pos=lower_bound(a,a+tot,x)-a+1;
                Update(pos,-1,x,1,M,1);
            }
            else{
                printf("%I64d\n",sum[1][2]);
            }
        }
    }
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

分组异或线段树

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
CodeForces Round #149 E XOR on Segment ★(区间异或&&分组线段树)
题目大意：序列a有n个数。实现两个操作：①求[l,r]区间和    ②对某个区间[l,r]所有数异或一个数x。
思路：比较容易想到是用线段树，因为每个数都小于10^6，所以把每一位拆成20位储存，这样就用20颗线段树。那么问题就转化为区间01异或问题：0异或任何数不变，不用修改，1异或一个数x结果是1-x。。（详细理解还是看代码吧。。
*/
const int N = 100010;
long long sum[22][N<<2],cnt[N<<2],ans[22];

void pushup(int rt){
    for (int i = 0; i < 20; i ++){
        sum[i][rt] = sum[i][rt<<1] + sum[i][rt<<1|1];
    }
}
void pushdown(int rt,int w){
    if (cnt[rt]){
        cnt[rt<<1] ^= cnt[rt];
        cnt[rt<<1|1] ^= cnt[rt];
        for (int i = 0; i < 20; i ++){
            if (cnt[rt] >> i & 1){
```

```
                sum[i][rt<<1] = w - (w >> 1) - sum[i][rt<<1];
                sum[i][rt<<1|1] = (w >> 1) - sum[i][rt<<1|1];
            }
        }
        cnt[rt] = 0;
    }
}
void build(int l,int r,int rt){
    if (l == r){
        long long x;
        cin>>x;
        for (int i = 0; i < 20; i ++){
            sum[i][rt] = x >> i & 1;
        }
        return ;
    }
    int mid = MID(l,r);
    build(l,mid,rt<<1);
    build(mid+1,r,rt<<1|1);
    pushup(rt);
    return ;
}

void update(int s,int t,int c,int l,int r,int rt){
    if (s <= l && r <= t){
        cnt[rt] ^= c;
        for (int i = 0; i < 20; i ++){
            if (c >> i & 1){
                sum[i][rt] = r - l + 1 - sum[i][rt];
            }
        }
        return ;
    }
    pushdown(rt,r-l+1);
    int mid = MID(l,r);
    if (s <= mid)   update(s,t,c,l,mid,rt<<1);
    if (mid < t)    update(s,t,c,mid+1,r,rt<<1|1);
    pushup(rt);
}

void query(int s,int t,int l,int r,int rt){
    if (s <= l && r <= t){
        for (int i = 0; i < 20; i ++){
            ans[i] += sum[i][rt];
        }
        return ;
    }
    pushdown(rt,r-l+1);
    int mid = MID(l,r);
    if (s <= mid)   query(s,t,l,mid,rt<<1);
    if (mid < t)    query(s,t,mid+1,r,rt<<1|1);
}
```

```
long long getquery(int s,int t,int l,int r){
    memset(ans,0,sizeof(ans));
    query(s,t,l,r,1);
    long long res = 0;
    for (int i = 0; i < 20; i ++){
        res += (1LL << i) * ans[i];
    }
    return res;
}
int main(){
    int n;
    scanf("%d",&n);
    memset(sum,0,sizeof(sum));
    memset(cnt,0,sizeof(cnt));
    build(1,n,1);
    int q;
    scanf("%d",&q);
    for (int i = 0; i < q; i ++){
        int w;
        scanf("%d",&w);
        if (w == 1){
            int a,b;
            cin>>a>>b;
            cout<<getquery(a,b,1,n)<<endl;
        }
        else{
            int a,b,c;
            cin>>a>>b>>c;
            update(a,b,c,1,n,1);
        }
    }
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

线段树区间合并

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
POJ 3667 Hotel （区间合并入门）
题目大意：区间内最长连续房间数。
问题出来了，对于二叉树，或许某子树根的左孩子的右边跟右孩子的左边连续着呢，怎么办？
于是，我们开出三个数组 lsum[] rsum[] 和 sum[]。对于区间 [L, R]，lsum[rt]表示以 L为开头的最长连续房间数，
rsum[rt]表示以R为结尾的最长连续房间数，sum[]表示[L,R]内的最长连续房间。
继续分析：当 lsum[rt<<1]等于左孩子区间总长度时，lsum[rt<<1]和lsum[rt<<1|1] (即左孩子的lsum和右孩子的lsum)
是相连的；
同理得， 当 rsum[rt<<1|1]等于右孩子总长度时，rsum[rt<<1|1]和rsum[rt<<1]（即右孩子的rsum和左孩子的rsum）是
相连的。
而对于一个 sum[rt]= max{ rsum[rt<<1]+lsum[rt<<1|1], sum[rt<<1], sum[rt<<1|1 }。
*/
```

```
const int N=50005;
int mmax[N<<2],lmax[N<<2],rmax[N<<2],cov[N<<2];

void BuildTree(int l,int r,int rt)
{
    mmax[rt]=lmax[rt]=rmax[rt]=r-l+1;
    cov[rt]=-1;
    if (l==r) return;
    int mid=MID(l,r);
    BuildTree(l,mid,rt<<1);
    BuildTree(mid+1,r,rt<<1|1);
}

void PushUp(int rt,int w)
{
    lmax[rt]=lmax[rt<<1];
    rmax[rt]=rmax[rt<<1|1];
    if (lmax[rt]==w-(w>>1)) lmax[rt]+=lmax[rt<<1|1];
    if (rmax[rt]==(w>>1))     rmax[rt]+=rmax[rt<<1];
    mmax[rt]=max(rmax[rt<<1]+lmax[rt<<1|1],max(mmax[rt<<1],mmax[rt<<1|1]));
}

void PushDown(int rt,int w)
{
    if (cov[rt]!=-1)        //cov的作用只是向下传递标记时区别是清空还是住满该区间。
    {                       //cov在区间修改时决定，而那时的Update区间意味着要么住满要么清空
        cov[rt<<1]=cov[rt<<1|1]=cov[rt];
        mmax[rt<<1]=lmax[rt<<1]=rmax[rt<<1]=cov[rt]?0:w-(w>>1); //cov=1，则mmax,lmax,rmax=0，表示住满
        mmax[rt<<1|1]=lmax[rt<<1|1]=rmax[rt<<1|1]=cov[rt]?0:(w>>1); //cov=0，则mmax,lmax,rmax=区间长度，表
示清空
        cov[rt]=-1;         //一定要注意下放标记后要重置!
    }
}

void Update(int s,int t,int c,int l,int r,int rt)   //c=1表示要住进，c=0表示要清空
{
    if (s<=l && r<=t)
    {
        cov[rt]=c;
        mmax[rt]=lmax[rt]=rmax[rt]=c?0:(r-l)+1;
        return ;
    }

    PushDown(rt,r-l+1);
    int mid=MID(l,r);
    if (s<=mid) Update(s,t,c,l,mid,rt<<1);
    if (mid<t)  Update(s,t,c,mid+1,r,rt<<1|1);
    PushUp(rt,r-l+1);
}

int Query(int L,int l,int r,int rt)
{
```

```
    if (l==r)
        return l;
    PushDown(rt,r-l+1);
    int mid=MID(l,r);
    if (mmax[rt<<1]>=L)  return Query(L,l,mid,rt<<1);       //按从左向右的顺序选
    else if (rmax[rt<<1]+lmax[rt<<1|1]>=L) return mid-rmax[rt<<1]+1;
    else return Query(L,mid+1,r,rt<<1|1);
}

int main()
{
    //freopen("test.in","r+",stdin);
    int n,m;
    scanf("%d%d",&n,&m);
    BuildTree(1,n,1);
    for (int i=0;i<m;i++)
    {
        int p;
        scanf("%d",&p);
        if (p==1)
        {
            int L;
            scanf("%d",&L);
            if (mmax[1]<L)  printf("%d\n",0);
            else
            {
                int a=Query(L,1,n,1);
                printf("%d\n",a);
                Update(a,a+L-1,1,1,n,1);
            }
        }
        else
        {
            int a,b;
            scanf("%d%d",&a,&b);
            Update(a,a+b-1,0,1,n,1);
        }
    }
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------------*/
```

线段树二次合并

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*
HDU 4351 Digital root ★★(2012 Multi-University Training Contest 6) (线段树值得研究的一道好题)
区间合并->二次合并(pushup)->更新时要合并左右子区间,询问时要合并左右询问子区间（这是不一样的，比如在[1,8]中
询问[1,6]，那么左右子区间分别是[1,4],[5,8]，而左右询问子区间是[1,4],[5,6]）。
预备知识：一个数的数字根就是这个数mod 9的余数（余数为0则取9，当然如果这个数本身是0那就是0了……）
思路：因为题目中所查询的区间，是所有的子区间的结果的并。所以区间合并就很必要了。
```

```
每个结点，记录这个区间的和的数根， 记录本区间内从左端点为端点往右的连续区间出现的数根，从右端点为端点往左的连续区间出现的数根，以及整个区间能出现的数根。然后合并什么的。。。
*/
const int maxn = 100100;
int mmax[maxn<<2],lsum[maxn<<2][10],rsum[maxn<<2][10],ssum[maxn<<2][10];
int mul[10][10];    //合并i,j打表优化

struct ANS
{
    int ls[10], rs[10];
    int res[10];
    int all;
    ANS()
    {
        memset(ls,0,sizeof(ls));
        memset(rs,0,sizeof(rs));
        memset(res,0,sizeof(res));
        all = 0;
    }
};

void initial()
{
    for (int i = 0; i < 10; i ++)
        for (int j = 0; j < 10; j ++)
        {
            int k = i + j;
            while (k > 9)
                k -= 9;
            mul[i][j] = k;
        }
}
//外挂优化。。。
inline void scanf_(int &num)
{
    char in;
    bool neg = false;
    while(((in = getchar()) > '9' || in <'0') && in != '-') ;
    if(in == '-')
    {
        neg = true;
        while((in = getchar()) >'9' || in < '0');
    }
    num = in - '0';
    while(in = getchar(),in >= '0' && in <= '9')
        num *= 10, num += in - '0';
    if(neg)
        num = 0 - num;
}
inline void printf_(int num)
{
    if(num < 0)
```

```
    {
        putchar('-');
        num =- num;
    }
    int ans[10],top = 0;
    while(num != 0)
    {
        ans[top ++] = num % 10;
        num /= 10;
    }
    if(top == 0)
        putchar('0');
    for(int i = top - 1; i >= 0; i --)
    {
        char ch = ans[i] + '0';
        putchar(ch);
    }
    //putchar('\n');
}

//线段树部分
inline void pushUp(int rt)
{
    for (int i = 0; i < 10; i ++)
    {
        lsum[rt][i] = lsum[rt<<1][i];
        rsum[rt][i] = rsum[rt<<1|1][i];
        ssum[rt][i] = (ssum[rt<<1][i] == 1)?1:ssum[rt<<1|1][i];
    }
    //修改区间和数字根
    int k = mul[mmax[rt<<1]][mmax[rt<<1|1]];
    mmax[rt] = k,  ssum[rt][k] = 1;

    for (int i = 0; i < 10; i ++)
    {
        //右前缀+区间和扩展前缀数字根和子区间数字根
        if (lsum[rt<<1|1][i])
        {
            int k = mul[mmax[rt<<1]][i];
            lsum[rt][k] = 1,   ssum[rt][k] = 1;
        }
        //左后缀+区间和扩展后缀数字根和子区间数字根
        if (rsum[rt<<1][i])
        {
            int k = mul[mmax[rt<<1|1]][i];
            rsum[rt][k] = 1,   ssum[rt][k] = 1;

            //左后缀+右前缀扩展子区间数字根
            for (int j = 0; j < 10; j ++)
                if (lsum[rt<<1|1][j])
                {
                    int k = mul[i][j];
```

```
                    ssum[rt][k] = 1;
                }
            }
        }
    }

inline void pushUpAns(int rt,ANS &p,ANS &lp,ANS &rp)
{
    //查询时也要合并左右区间...
    for (int i = 0; i < 10; i ++)
    {
        p.ls[i] = lp.ls[i];
        p.rs[i] = rp.rs[i];
        p.res[i] = (lp.res[i] == 1)?1:rp.res[i];
    }
    for (int i = 0; i < 10; i ++)
    {
        if (rp.ls[i])
        {
            int k = mul[lp.all][i];
            p.ls[k] = 1,    p.res[k] = 1;
        }
        if (lp.rs[i])
        {
            int k = mul[rp.all][i];
            p.rs[k] = 1,    p.res[k] = 1;
        }
        if (lp.rs[i])
            for (int j = 0; j < 10; j ++)
                if (rp.ls[j])
                {
                    int k = mul[i][j];
                    p.res[k] = 1;
                }
    }
}

void build(int l,int r,int rt)
{
    if (l == r)
    {
        mmax[rt] = 0;
        for (int i = 0; i < 10; i ++)
        {
            lsum[rt][i] = 0;
            rsum[rt][i] = 0;
            ssum[rt][i] = 0;
        }
        int a;
        scanf_(a);
        //scanf("%d",&a);
        if (!a) {   mmax[rt] = 0; lsum[rt][0] = 1;   rsum[rt][0] = 1;   ssum[rt][0] = 1;   }
```

```
        else
        {
            a = a % 9;
            if (!a) {   mmax[rt] = 9; lsum[rt][9] = 1;   rsum[rt][9] = 1;   ssum[rt][9] = 1;  }
            else {  mmax[rt] = a; lsum[rt][a] = 1;   rsum[rt][a] = 1;    ssum[rt][a] = 1; }
        }
        return ;
    }
    int mid = MID(l,r);
    build(l, mid, rt<<1);
    build(mid+1, r, rt<<1|1);
    pushUp(rt);
}

int query(int s,int t,int l,int r,int rt,ANS &p)
{
    if (s <= l && r <= t)
    {
        for (int i = 9; i >= 0; i --)
        {
            if (ssum[rt][i])
                p.res[i] = 1;
            if (lsum[rt][i])
                p.ls[i] = 1;
            if (rsum[rt][i])
                p.rs[i] = 1;
        }
        p.all = mmax[rt];
        while (p.all > 9)   p.all -= 9;
        return p.all;
    }
    int res = 0;
    int mid = MID(l,r);
    ANS lp,rp;
    if (s <= mid)   res += query(s,t,l,mid,rt<<1,lp);
    if (mid < t)    res += query(s,t,mid+1,r,rt<<1|1,rp);
    p.all = res;
    while (p.all > 9)   p.all -= 9;
    pushUpAns(rt,p,lp,rp);
    return res;
}

int main()
{
//    freopen("data.txt","r+",stdin);
//    freopen("ans.txt","w+",stdout);
    initial();
    int t, caseo = 1;
    scanf_(t);
    //scanf("%d",&t);
    while(t--)
    {
```

```
        printf("Case #%d:\n",caseo ++);
        int n;
        scanf_(n);
        //scanf("%d",&n);
        build(1,n,1);
        int Q;
        scanf_(Q);
        //scanf("%d",&Q);
        while(Q--)
        {
            ANS ans;
            int l,r;
            scanf_(l),scanf_(r);
            //scanf("%d%d",&l,&r);
            query(l,r,1,n,1,ans);
            int tot = 0;
            for (int i = 9; i >= 0; i --)
                if (ans.res[i] && tot < 5)
                {
                    tot ++;
                    //tot == 5?printf("%d\n",i):printf("%d ",i);
                    if (tot == 5)
                        printf_(i),putchar('\n');
                    else    printf_(i),putchar(' ');
                }
            while(tot < 5)
            {
                tot ++;
                //tot == 5?printf("-1\n"):printf("-1 ");
                if (tot == 5)
                    puts("-1");
                else
                {
                    putchar('-');
                    putchar('1');
                    putchar(' ');
                }
            }
        }
        if (t)  printf("\n");
    }
    return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

线段树矩形面积并

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
HDU 1542 Atlantis (矩形面积并入门题)
问题抽象：矩形面积并
```

思路：浮点数先要离散化；对于一个矩形(x1, y1, x2, y2)，只取上边和下边，结构体ss[] 记录一条边的左右点坐标和距坐标轴的高度，再拿一个变量 ss[].s记录这条边是上边还是下边，下边记录为1，上边记录为-1。按高度排序，对横轴建树，从低到高扫描一遍。若下边，则加入线段树；若上边，则从线段树上去掉。用cnt表示该区间下边比上边多几条线段,sum代表该区间内被覆盖的长度总和。
这里线段树的一个结点并非是线段的一个端点,而是该端点和下一个端点间的线段,所以题目中r+1,r-1的地方要好好的琢磨一下。
*/

```
const int maxn = 500;
double X[maxn];
double sum[maxn<<2];
int cnt[maxn<<2];

struct Seg
{
    double l,r,h;
    int s;
    Seg(){}
    Seg(double a,double b,double c,int d):l(a),r(b),h(c),s(d){}
    bool operator < (const Seg &cmp) const
    {
        return h < cmp.h;
    }
}ss[maxn];

void pushup(int rt,int l,int r)
{
    if (cnt[rt])    sum[rt] = X[r+1] - X[l];
    else if (l == r)    sum[rt] = 0;
    else sum[rt] = sum[rt<<1] + sum[rt<<1|1];
}
void update(int s,int t,int v,int l,int r,int rt)
{
    if (s <= l && r <= t)
    {
        cnt[rt] += v;
        pushup(rt,l,r);
        return ;
    }
    int mid = MID(l,r);
    if (s <= mid)   update(s,t,v,l,mid,rt<<1);
    if (mid < t)    update(s,t,v,mid+1,r,rt<<1|1);
    pushup(rt,l,r);
}
int binfind(double x,int n,double X[])
{
    int head = 0, tail = n - 1;
    while(head <= tail)
    {
        int mid = MID(head,tail);
        if (X[mid] == x)    return mid;
        if (X[mid] < x)     head = mid + 1;
        else    tail = mid - 1;
```

```
    }
    return -1;
}
int main()
{
    //freopen("data.txt","r+",stdin);

    int cas = 1;
    int n;
    while(~scanf("%d",&n) && n)
    {
        int tot = 0;
        while(n--)
        {
            double a,b,c,d;
            scanf("%lf%lf%lf%lf",&a,&b,&c,&d);
            X[tot] = a;
            ss[tot ++] = Seg(a,c,b,1);
            X[tot] = c;
            ss[tot ++] = Seg(a,c,d,-1);
        }
        sort(X,X+tot);
        sort(ss,ss+tot);
        int ptot = 1;
        for (int i = 1; i < tot ; i ++)
            if (X[i]!=X[i-1])   X[ptot ++] = X[i];

        //build(1,ptot,1);
        memset(cnt , 0 , sizeof(cnt));
        memset(sum , 0 , sizeof(sum));
        double res = 0;
        for (int i = 0; i < tot - 1; i ++)
        {
            int l = binfind(ss[i].l, ptot, X);
            int r = binfind(ss[i].r, ptot, X) - 1;
            if (l <= r) update(l, r, ss[i].s, 0, ptot-1 , 1);
            res += sum[1] * (ss[i+1].h - ss[i].h);
        }
        printf("Test case #%d\nTotal explored area: %.2lf\n\n",cas++ , res);
    }

    return 0;
}
/*-------------------------------------------------------------------------------------------------------------------------*/
```

二维线段树

```
/*-------------------------------------------------------------------------------------------------------------------------*/
/*
HDU 1823 Luck and Love (二维线段树入门)
因为有两个限制范围，所以需要二维的线段树。也没什么好说的地方。
```

```
需要注意几个问题：1.初始化sum = -1，不然所有的返回结果都是>=0的数，查找不到的情况不好判断。
2.坐标左小右大（因为这个白白贡献几次WA擦。。。）
3.有可能存在同一身高、活泼度的人，所以更新时不是直接赋值而是选个缘分值最大的。
*/
const int maxn1 = 105;
const int maxn2 = 1050;
int sum[maxn1<<2][maxn2<<2];
int n1 = 100, n2 = 1000;

void build()
{
    memset(sum,-1,sizeof(sum));
}
void update2(int p,int c,int l,int r,int rt1,int rt2)
{
    if (l == p && r == p)
    {
        sum[rt1][rt2] = max(sum[rt1][rt2], c);      //重要！：有可能存在身高、活泼度一样的美女，要选缘分值
高的
        return ;
    }

    int mid = MID(l,r);
    if (p <= mid)   update2(p,c,l,mid,rt1,rt2<<1);
    else    update2(p,c,mid+1,r,rt1,rt2<<1|1);
    sum[rt1][rt2] = max(sum[rt1][rt2<<1],sum[rt1][rt2<<1|1]);
}
void update1(int h,int p,int c,int l,int r,int rt)  //h身高，p活泼度
{
    update2(p,c,0,n2,rt,1);
    if (l == h && r == h)
        return ;
    int mid = MID(l,r);
    if (h <= mid)   update1(h,p,c,l,mid,rt<<1);
    else    update1(h,p,c,mid+1,r,rt<<1|1);
}
int query2(int p1,int p2,int l,int r,int rt1,int rt2)
{
    if (p1 <= l && r <= p2)
    {
        return sum[rt1][rt2];
    }
    int mid = MID(l,r);
    int res = -1;
    if (p1 <= mid)   res = max(res, query2(p1,p2,l,mid,rt1,rt2<<1));
    if (mid < p2)    res = max(res, query2(p1,p2,mid+1,r,rt1,rt2<<1|1));
    return res;
}
int query1(int h1,int h2,int p1,int p2,int l,int r,int rt)
{
    if (h1 <= l && r <= h2)
        return query2(p1,p2,0,n2,rt,1);
```

```
    int mid = MID(l,r);
    int res = -1;
    if (h1 <= mid)  res = max(res, query1(h1,h2,p1,p2,l,mid,rt<<1));
    if (mid < h2)   res = max(res, query1(h1,h2,p1,p2,mid+1,r,rt<<1|1));
    return res;
}
int main()
{
    //freopen("data.txt","r+",stdin);

    int m;
    while(scanf("%d",&m),m)
    {
        build();
        while(m--)
        {
            char s[2];
            scanf("%s",s);
            if (s[0] == 'I')
            {
                int h;
                double p,num;
                scanf("%d%lf%lf",&h,&p,&num);
                update1(h,int(p*10),int(num*10),n1,n1+100,1);
            }
            else
            {
                int h1,h2;
                double p1,p2;
                scanf("%d%d%lf%lf",&h1,&h2,&p1,&p2);
                if (h1 > h2)    swap(h1,h2);
                if (p1 > p2)    swap(p1,p2);
                int res = query1(h1,h2,int(p1*10),int(p2*10),n1,n1+100,1);
                if (res < 0)    printf("-1\n");
                else printf("%.1lf\n",res*1.0/10.0);
            }
        }
    }

    return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

Graph Theory

链式前向星

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
        head[i]表示以i为起点的第一条边在edge[]中的下标;
        edge[k]表示边的信息：edge[k].next表示与这条边同起点的下一条边的位置；edge[k].u、edge[k].v、
edge[k].w分别表示边的起点、终点、权值。
*/
const int MAXE = 1000;
struct node{
        int u, v, w;
        int next;
}edge[MAXE];
int cnt, head[MAXE];

void add(int u, int v, int w){//添加边
        edge[cnt].u = u;
        edge[cnt].v = v;
        edge[cnt].w = w;
        edge[cnt].next = head[u];
        head[u] = cnt ++;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

Eulerian Path

```
/*-------------------------------------------------------------------------------------------------------------*/
//求欧拉回路或欧拉路,邻接阵形式,复杂度O(n^2)
//返回路径长度,path返回路径(有向图时得到的是反向路径)
//传入图的大小n和邻接阵mat,不相邻点边权0
//可以有自环与重边,分为无向图和有向图
#define MAXN 100
void find_path_u(int n,int mat[][MAXN],int now,int& step,int* path){
        int i;
        for (i=n-1;i>=0;i--)
                while (mat[now][i]){
                        mat[now][i]--,mat[i][now]--;
                        find_path_u(n,mat,i,step,path);
                }
        path[step++]=now;
}

void find_path_d(int n,int mat[][MAXN],int now,int& step,int* path){
        int i;
        for (i=n-1;i>=0;i--)
                while (mat[now][i]){
                        mat[now][i]--;
                        find_path_d(n,mat,i,step,path);
                }
        path[step++]=now;
```

```
}

int euclid_path(int n,int mat[][MAXN],int start,int* path){
        int ret=0;
        find_path_u(n,mat,start,ret,path);
//      find_path_d(n,mat,start,ret,path);
        return ret;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

Topological Sorting

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
HDU 1285，拓扑排序，并且保证顶点标号字典序最小(BFS优先队列实现)
*/
const int MAXV = 503;
const int MAXE = 250003;
struct node{
    int u, v;
    int next;
}edge[MAXE];
int head[MAXV], cnt;
void init(){
    cnt = 0;
    mem(head, -1);
}
void add(int u, int v){
    edge[cnt].u = u;
    edge[cnt].v = v;
    edge[cnt].next = head[u];
    head[u] = cnt ++;
}
/* ---------------- Topological sorting ------------------- */
int ind[MAXV];
vector <int> top_res;
priority_queue <int , vector<int> , greater<int> > PQ;
void cal_indegree(){
    mem(ind, 0);
    for (int i = 0; i < cnt; i ++){
        ind[edge[i].v] ++;
    }
    return ;
}
void topsort(int n){
    while(!PQ.empty())
        PQ.pop();
    top_res.clear();
    cal_indegree();
    for (int i = 1; i <= n; i ++)
        if (ind[i] == 0)
```

```
            PQ.push(i);
    while(!PQ.empty()){
        int u = PQ.top();
        PQ.pop();
        top_res.push_back(u);
        for (int i = head[u]; i != -1; i = edge[i].next){
            if ((-- ind[edge[i].v]) == 0)
                PQ.push(edge[i].v);
        }
    }
}
/* ---------------- Topological sorting -------------------- */

int main(){
    int n, m;
    while(scanf("%d %d", &n, &m) != EOF){
        init();
        for (int i = 0; i < m; i ++){
            int a, b;
            scanf("%d %d", &a, &b);
            add(a, b);
        }
        topsort(n);
        for (int i = 0; i < (int)top_res.size() - 1; i ++)
            printf("%d ", top_res[i]);
        printf("%d\n", top_res[top_res.size()-1]);
    }
        return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

SCC:Kosaraju

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
        过程(证明算导P339):
                ①对图G做一遍DFS，计算出每个节点的结束时间戳f[u];
                ②计算图G的转置GT;
                ③对GT进行DFS，主循环中按f[u]降序的顺序访问节点，遍历得到的森林就是SCC的集合
                order[]是结束时间戳为i的节点标号，这样就不用再对f[]排序了.
        该算法的优点在于，最后得到的节点是按照拓扑序组织好的，在求解2-SAT的过程中十分方便。
        结果:
                scc[]存储每个节点所属的强连通分支编号,scc_num为强连通分支总数
*/
const int MAXV = 105;
const int MAXE = 20005;
/* ----------- 强联通分量Kosaraju算法 ----------- */
struct node{
    int u, v;
    int next;
}arc[MAXE], t_arc[MAXE];        //arc是G的弧,t_arc是GT的弧，加边时可以一块儿加.
```

```
int cnt, head[MAXV], t_head[MAXV];
int d[MAXV], f[MAXV];       //深搜时间戳(其实这里没用)
int order[MAXV];            //结束时间戳为i的节点标号
int scc[MAXV], scc_num;     //每个节点所属强连通分量编号，强连通分量总数
void init(){
    cnt = 0;
    mem(head, -1);
    mem(t_head, -1);
}
void add(int u, int v){
    arc[cnt].u = u;
    arc[cnt].v = v;
    arc[cnt].next = head[u];
    head[u] = cnt;

    t_arc[cnt].u = v;
    t_arc[cnt].v = u;
    t_arc[cnt].next = t_head[v];
    t_head[v] = cnt ++;
    return ;
}
bool vis[MAXV];
int id, fid;
void dfs(int u){
    vis[u] = 1;
    d[u] = id ++;
    for (int i = head[u]; i != -1; i = arc[i].next){
        int v = arc[i].v;
        if (!vis[v]){
            dfs(v);
        }
    }
    f[u] = id;
    order[fid ++] = u;
    return ;
}
void dfs_t(int u){
    vis[u] = 1;
    scc[u] = scc_num;
    for (int i = t_head[u]; i != -1; i = t_arc[i].next){
        int v = t_arc[i].v;
        if (!vis[v]){
            dfs_t(v);
        }
    }
    return ;
}
void Kosaraju(int n){
    //init
    scc_num = 0;
    mem(scc, -1);
    mem(d, -1);
```

```
    mem(f, -1);
    mem(order, -1);

    mem(vis, 0);
    id = fid = 0;
    for (int u = 1; u <= n; u ++){   //注意图中节点编号从几开始
        if (!vis[u]){
            dfs(u);
        }
    }
    mem(vis, 0);
    for (int i = n - 1; i >= 0; i --){
        int u = order[i];
        if (!vis[u]){
            scc_num ++;
            dfs_t(u);
        }
    }
}
/* ----------- 强联通分量Kosaraju算法 ----------- */
int main(){
    int n;
    scanf("%d", &n);
    init();
    for (int i = 1; i <= n; i ++){
        int j;
        while(scanf("%d", &j), j){
            add(i, j);
        }
    }
    Kosaraju(n);
    printf("%d\n", scc_num);
       return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

Shortest Path Problem(Bellman-Ford,Dijkstra,Floyd)

```
/*-------------------------------------------------------------------------------------------------------------*/
/* Bellman-Ford, 支持负权、返回值判断是否有负环 */
struct node{
    int u;
    int v;
    int w;
}edge[maxm];
int dis[maxn];
int cnt;
void init(){
    cnt = 0;
}
void add(int u, int v, int w){
```

```
    edge[cnt].u = u;
    edge[cnt].v = v;
    edge[cnt++].w = w;
}
bool bellman_ford(int s){
    for(int i = 0; i < n; i++){
        dis[i] = oo;
    }
    dis[s] = 0;
    for(int i = 1; i < n; i ++){
              bool relax = false;           //当过程不再松弛时便退出，一个很大的优化，效率可逼近Dijkstra
        for(int j = 0; j < cnt; j++){
            int u = edge[j].u;
            int v = edge[j].v;
            int w = edge[j].w;
            if(dis[v] > dis[u] + w){
                dis[v] = dis[u] + w;
                              relax = true;
            }
        }
               if (!relax)
                      break;
    }
       //判断是否有负环
    for(int j = 0; j < cnt; j++){
        int u = edge[j].u;
        int v = edge[j].v;
        int w = edge[j].w;
        if(dis[v] > dis[u] + w){
            return false;
        }
    }
    return true;
}

/* STL堆+Dijkstra: O(ElogV) 链式前向星实现 */
struct node{
    int u, v, w;
    int next;
}edge[MAXE];
int cnt, head[MAXV];
void init(){
    mem(head, -1);
    mem(edge, -1);
    cnt = 0;
}
void add(int u, int v, int w){
    edge[cnt].u = u;
    edge[cnt].v = v;
    edge[cnt].w = w;
    edge[cnt].next = head[u];
    head[u] = cnt ++;
```

```
}
priority_queue <pair<int ,int>,vector<pair<int ,int> >,greater<pair<int,int> > > PQ;
void Dijkstra(int s,int n)
{
    for (int i=0;i<n;i++)
        dist[i]=oo;

    int vis[MAXV];
    memset(vis,0,sizeof(vis));

    dist[s]=0;
    while(!PQ.empty())
        PQ.pop();
    PQ.push(make_pair(dist[s],s));
    while(!PQ.empty())
    {
        int k=PQ.top().second;
        PQ.pop();
        if (!vis[k])
        {
            vis[k]=1;
            for (int i = head[u]; i != -1 ; i = edge[i].next){
                        int v = edge[i].v;
                        if (dis[v] > dis[u] + edge[i].w){
                                dis[v] = dis[u] + edge[i].w;
                                PQ.push(make_pair(dis[v], v));
                        }
            }
        }
    }
}

/* Floyd-Warshall */
/*
        Floyd-Warshall算法（Floyd-Warshall algorithm）是解决任意两点间的最短路径的一种算法，可以正确处理有向图或负权的最短路径问题，同时也被用于计算有向图的传递闭包。
        Floyd-Warshall算法的原理是动态规划。
                设D(i,j,k)为从i到j的只以(1..k)集合中的节点为中间节点的最短路径的长度。
                若最短路径经过点k，则D(i,j,k) = D(i,k,k-1) + D(k,j,k-1);
                若最短路径不经过点k，则D(i,j,k) = D(i,j,k-1);
                因此，D(i,j,k) = min(D(i,j,k-1), D(i,k,k-1) + D(k,j,k-1))。
        在实际算法中，为了节约空间，可以直接在原来空间上进行迭代，这样空间可降至二维。
*/
void Floyd(int n){
        for (int k = 1; k <= n; k ++){
        for (int i = 1; i <= n; i ++){
            for (int j = 1; j <= n; j ++){
                dis[i][j] = min(dis[i][j] ,dis[i][k] + dis[k][j]);
            }
        }
    }
}
```

/*--*/

SPFA

/*--*/

```
#define arraysize 501
int maxData = 0x7fffffff;
typedef struct edge
{
   int to;
   int w;
}edge;
vector<edge> adjmap[arraysize]; //vector实现邻接表

int d[arraysize];
bool final[arraysize];          //记录顶点是否在队列中，SPFA算法可以入队列多次
int cnt[arraysize];             //记录顶点入队列次数

bool SPFA(int s)
{
     queue<int> myqueue;
     int i,j;
     for(i=0;i<n+1;++i)
     {
         d[i] = maxData;        //将除源点以外的其余点的距离设置为无穷大
     }
     memset(final,0,sizeof(final));
     memset(cnt,0,sizeof(cnt));
     d[s]=0;                    //源点的距离为0
     final[s] = true;
     cnt[s]++;                  //源点的入队列次数增加
     myqueue.push(s);
     int topint;
     while(!myqueue.empty())
     {
         topint = myqueue.front();
                myqueue.pop();
         final[topint] = false;
         for(i=0;i<adjmap[topint].size();++i)
         {
             int to = adjmap[topint][i].to;
             if(d[topint]<maxData && d[to]>d[topint]+ adjmap[topint][i].w)
             {
                  d[to] = d[topint]+ adjmap[topint][i].w;
                  if(!final[to])
                  {
                      final[to] = true;
                      cnt[to]++;
                      if(cnt[to]>=n)   //当一个点入队的次数>=n时就证明出现了负环。
                         return true;
                      myqueue.push(to);
```

```
                }
            }
        }
    }
    return false;
}

int main()
{
    for(i=1;i<n+1;++i)        //此处特别注意对邻接表清空
        adjmap[i].clear();
    for(i=0;i<m;++i)          //双向
    {
        cin>>s>>e>>w;
        temp.to = e;
        temp.w = w;
        adjmap[s].push_back(temp);
        temp.to = s;
        adjmap[e].push_back(temp);
    }
}
/*-------------------------------------------------------------------------------------------------------------*/
```

Second Shortest Path Problem

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
二维Dijkstra求次短路(k很小时的k短路也适用)：

我们知道Dijkstra就是不断地用已经确定最短路的节点(黑色)去松弛未确定的点(白色)，用数学归纳法很容易证明这是正确的。它的核心思想就是某个节点的最短路一定是由它前驱节点的最短路扩展来的。那么对于次短路也可以类似的看：一个节点的次短路一定是由它前驱节点的最短路 or 次短路扩展而来的。那么我们就可以把节点分成两层处理：一层处理、存储最短路，另一层处理、存储次短路。这样，用于记录状态的数组变成了二维，放进堆中的状态也必须是"二维"的，这里的"二维"并不是要你开个二维数组，而是需要在放入堆中的结构体里多加一个标记变量，用于标识到底是最短路还是次短路，当然，用于标记已经确定最短路、次短路的点的closed表同样要变成二维的。循环结束条件是堆为空，因为要计数，就必须遍历所有情况.

具体在做状态转移的时候，拿到当前状态，扩展下一状态，设当前状态长度为d，下一状态最短路和次短路状态分别是d0和d1，则：

d小于d0，则d1=d0,d0=d，计数都重置为所赋的值对应的计数.
d等于d0，则累加最短路计数.
d小于d1，则d1=d，重置次短路计数为所赋的值对应的计数.
d等于d1，则累加次短路计数.
*/
const int sup = (1 << 30);
const int inf = -0x7fffffff;

const int MAXV = 1003;
const int MAXE = 10003;
struct Status{
```

```
    int v;
    int dis;
    int type;
    friend bool operator < (Status n1, Status n2){
        return n2.dis < n1.dis;
    }
};
struct node{
    int u, v, w;
    int next;
}edge[MAXE];
int n, m;
int s, f;
int cnt, head[MAXV];
int dis[MAXV][2], ans[MAXV][2];
bool vis[MAXV][2];
void add(int u, int v, int w){
    edge[cnt].u = u;
    edge[cnt].v = v;
    edge[cnt].w = w;
    edge[cnt].next = head[u];
    head[u] = cnt ++;
    return ;
}
void init(int n, int m){
    mem(head, -1);
    for (int i = 0; i <= n; i ++)
        dis[i][0] = dis[i][1] = sup;
    mem(ans, 0);
    mem(vis, 0);
    cnt = 0;
}
priority_queue <Status, vector<Status> > PQ;
void Dij(int s, int f){
    while(!PQ.empty())
        PQ.pop();
    Status tmp;
    dis[s][0] = 0;
    ans[s][0] = 1;
    tmp.v = s;
    tmp.type = 0;
    tmp.dis = 0;
    PQ.push(tmp);
    while(!PQ.empty()){
        Status t = PQ.top();
        PQ.pop();
        if (vis[t.v][t.type])   continue;
        vis[t.v][t.type] = 1;
        int u = t.v;
        int type = t.type;
        for (int i = head[u]; i != -1; i = edge[i].next){
            int v = edge[i].v;
```

```
            int w = dis[u][type] + edge[i].w;
            if (dis[v][0] > w){
                dis[v][1] = dis[v][0];
                ans[v][1] = ans[v][0];
                Status t0;
                t0.dis = dis[v][1];
                t0.type = 1;
                t0.v = v;
                PQ.push(t0);

                dis[v][0] = w;
                ans[v][0] = ans[u][type];
                t0.dis = dis[v][0];
                t0.type = 0;
                t0.v = v;
                PQ.push(t0);
            }
            else if (dis[v][0] == w){
                ans[v][0] += ans[u][type];
            }
            else if (dis[v][1] > w){
                dis[v][1] = w;
                ans[v][1] = ans[u][type];
                Status t0;
                t0.dis = dis[v][1];
                t0.type = 1;
                t0.v = v;
                PQ.push(t0);
            }
            else if (dis[v][1] == w){
                ans[v][1] += ans[u][type];
            }
        }
    }
    int res = ans[f][0];
    if (dis[f][0] == dis[f][1] - 1)
        res += ans[f][1];
    printf("%d\n", res);
}
int main(){
    int te;
    scanf("%d", &te);
    while (te --){
        scanf("%d %d", &n, &m);
        init(n, m);
        while (m --){
            int a, b, l;
            scanf("%d %d %d", &a, &b, &l);
            add(a, b, l);
        }
        scanf("%d %d", &s, &f);
        Dij(s, f);
```

```
    }
        return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

经过 N 条边的最短路（Floyd+二分矩阵快速幂）

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
这个问题和求两点间经过N条边的路径数很相似，而我们知道如果用图的邻接矩阵A存储图的话，二分矩阵快速幂A^N即为所求。路径数能用矩阵乘法求是因为它的状态方程正好和矩阵乘法一样：设dp[i][j][p]表示i到j点经过p条边的路径数，则dp[i][j][p] = sigma(dp[i][k][p-1]*dp[k][j][1])，即A=B*C（把dp[p]看成A，dp[p-1]看成B……）；

但显然最短路的方程不是这样的。按照Floyd的方程它应该是dp[i][j] = min(dp[i][j], dp[i][k]+dp[k][j])。

那这样的方程能用类似上面的方法求么？答案是肯定的，只要修改一下“乘法”即可。显然，只要矩阵运算符合结合律，那么它就能用二分矩阵快速幂做。关于上面Floyd方程符合结合率的证明，俞华程的论文《矩阵乘法在信息学中的应用》有证明，不过他前面的我没看懂。。。

那么只要我们重新定义下矩阵“乘法”为：C[i][j] = min(C[i][j], A[i][k]+B[k][j])，问题迎刃而解。

假如我们设图的邻接矩阵为VE，则按上面的定义，VE * VE 显然就是两点到达经过两条边所需要的最短路径。然后同理VE ^ N就是到达经过N条边所需要的最短路径。为什么这个很类似Floyd的式子求出来的是确定了经过边数的最短距离？因为这个更新和Floyd不同的是更新到一个新的矩阵上去了而不是直接像Floyd的自己更新自己。所以在一更新时，不会出现自己刚更新的值又来继续更新。
*/
int hash[1000003], cnt;
struct mat{
    long long map[200][200];
    void init(){
        mem(map, -1);
    }
    void make_head(){
        mem(map, -1);
        for (int i = 0; i < cnt; i ++)
            map[i][i] = 0;
    }
}A;
int n, t, s, e;
mat floyd(mat &A, mat &B){
    mat res;
    res.init();
    for (int k = 0; k < cnt; k ++){
        for (int i = 0; i < cnt; i ++){
            if (A.map[i][k] != -1){
                for (int j = 0; j < cnt; j ++){
                    if(B.map[k][j] != -1){
                        if (res.map[i][j] == -1){
                            res.map[i][j] = A.map[i][k] + B.map[k][j];
                        }
```

```
                        else{
                            res.map[i][j] = min(res.map[i][j], A.map[i][k] + B.map[k][j]);
                        }
                    }
                }
            }
        }
    }
    return res;
}
long long work(mat &A, int n){
    mat res;
    res.make_head();
    while(n){
        if (n & 1){
            res = floyd(res, A);
        }
        n >>= 1;
        A = floyd(A, A);
    }
    return res.map[hash[s]][hash[e]];
}
int main(){
    mem(hash, -1);
    A.init();
    cnt = 0;
    scanf("%d %d %d %d", &n, &t, &s, &e);
    if (hash[s] == -1)
        hash[s] = cnt ++;
    if (hash[e] == -1)
        hash[e] = cnt ++;
    for (int i = 0; i < t; i ++){
        int l, a, b;
        scanf("%d %d %d", &l, &a, &b);
        if (hash[a] == -1)
            hash[a] = cnt ++;
        if (hash[b] == -1)
            hash[b] = cnt ++;
        A.map[hash[a]][hash[b]] = l;
        A.map[hash[b]][hash[a]] = l;
    }
    printf("%I64d\n", work(A, n));
        return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

Transitive Closure

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
```

POJ 3660 题目大意：有N头奶牛，现在告诉你他们两头牛之间比赛的胜负结果M组，要你据此求出有多少头奶牛的排名可以确定下来。
解题思路：对于某个人而言，如果比他成绩好的人有lose（你输了），比他成绩差的有win（被你打败），如果lose+win的人数比总人数少一个（他自己），他的排名就可以确定了。所以只需要应用Floyd传递关系，最后计算出win的人数和lose的人数即可！
*/

```
/* ------------------------------求传递闭包模板部分------------------------------ */
/* calculate Transitive Closure By Warshall */
const int N = 103;
int R[N][N];
int TC[N][N];
void Warshall_init(){
    mem(TC, 0);
}
void Warshall(int n){
    Warshall_init();
    //To copy the original matrix to the matrix that will be calculated.
    for (int i = 0; i < n; i ++)
        for (int j = 0; j < n; j ++)
            TC[i][j] = R[i][j];

    for (int k = 0; k < n; k ++)
        for (int i = 0; i < n; i ++)
            for (int j = 0; j < n; j ++){
                if (TC[i][j])   continue;
                TC[i][j] = TC[i][k] && TC[k][j];
            }
}

/* ------------------------------求传递闭包模板部分------------------------------ */

int main(){
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; i ++){
        int a, b;
        scanf("%d %d", &a, &b);
        R[a-1][b-1] = 1;
    }
    Warshall(n);
    int ans = 0;
    for (int i = 0; i < n; i ++){
        int win = 0, fail = 0;
        for (int j = 0; j < n; j ++){
            if (TC[i][j])
                win ++;
            else if (TC[j][i])
                fail ++;
        }
        if (win + fail == n-1)
            ans ++;
```

```
    }
    printf("%d\n", ans);
        return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

Minimum Spanning Tree(Kruskal)

```
/*-------------------------------------------------------------------------------------------------------------*/
/*
        Kruskal.
        算法描述：克鲁斯卡尔算法需要对图的边进行访问，所以克鲁斯卡尔算法的时间复杂度只和边又关系，可以证明其时间复杂度为O(ElogE)。
        算法过程：
                1.将图各边按照权值进行排序
                2.将图遍历一次，找出权值最小的边，（条件：此次找出的边不能和已加入最小生成树集合的边构成环），若符合条件，则加入最小生成树的集合中。不符合条件则继续遍历图，寻找下一个最小权值的边。
                3.递归重复步骤1，直到找出n-1条边为止（设图有n个结点，则最小生成树的边数应为n-1条），算法结束。得到的就是此图的最小生成树。
*/
const int MAXN = 505;
const int MAXE = 250005;
int res, maxe;
struct Disjoint_Sets{
    int father, ranks;
}S[MAXN];
void init(){
    res = 0;
    maxe = 0;
    for (int i = 0; i < MAXN; i ++)
        S[i].father = i, S[i].ranks = 0;
}
int Father(int x){
    if (S[x].father == x){
        return x;
    }
    else{
        S[x].father = Father(S[x].father);     //Path compression
        return S[x].father;
    }
}
bool Union(int x, int y, int w){
    int fx = Father(x), fy = Father(y);
    if (fx == fy){
        return false;
    }
    else{                                       //Rank merge
        res += w;                                           //return sum of the edges of the MST.
        maxe = max(maxe, w);                  //return max edge of the edges of the MST.
        if (S[fx].ranks > S[fy].ranks){
            S[fy].father = fx;
```

```
        }
        else{
            S[fx].father = fy;
            if (S[fx].ranks == S[fy].ranks){
                ++ S[fy].ranks;
            }
        }
        return true;
    }
}
struct node{
    int u, v, w;
}edge[MAXE];
int cnt;
bool cmp (const node n1, const node n2){
    return n1.w < n2.w;
}
void add(int u, int v, int w){
    edge[cnt].u = u;
    edge[cnt].v = v;
    edge[cnt ++].w = w;
    return ;
}
int main(){
    int t, n;
    scanf("%d", &t);
    while(t --){
        cnt = 0;
        init();
        scanf("%d", &n);
        for (int i = 0; i < n; i ++){
            for (int j = 0; j < n; j ++){
                int tmp;
                scanf("%d", &tmp);
                if (i > j)
                add(i, j, tmp);
            }
        }
        sort(edge, edge+cnt, cmp);
               int num = 0;
        for (int i = 0; i < cnt && num < n - 1; i ++){
            if (Union(edge[i].u, edge[i].v, edge[i].w)){
                num ++;
            }
        }
               //Strictly , only when (num == n - 1) dose the gragh contain a MST.
        printf("%d\n", maxe);
    }
       return 0;
}
/*--------------------------------------------------------------------------------------------------------------*/
```

Second Minimum Spanning Tree

```
/*-------------------------------------------------------------------------------------------------------------*/
/*  次小生成树(The second MST, 2-MST)

基本思想：首先求出最小生成树，记录权值之和为MST_NUM。然后枚举添加边(u,v)，加上以后一定形成一个环，找到环上非(u,v)边的权值最大的边，把它删掉，计算当前生成树的权值之和，取所有枚举加边后生成树权值之和的最小值，就是次小生成树。

算法：
1.求出最小生成树T及其权值和MST_NUM，并标注在最小生成树上的边。
2.从每个顶点i为根，DFS遍历最小生成树，求出从i到j的路径上最大边的权值P(i, j)。
3.遍历每条不在最小生成树中的边(i,j)，加上这条边，并删除环上最大边(P(i,j))，新的生成树权值之和为MST_NUM + w(i,j) - P[i][j]，记录其最小值即可，时间复杂度为O(N^2)。求最小生成树可以用最简单的Prim即可，算法的瓶颈在第二步DFS遍历求路径上最大边需要O(n^2)，用更好的算法是没有意义的。(PS：对于每个边，实际上可以用LCA和预处理来得到环内最大边，这样复杂度就是O(ElogV)，然后配合Kruskal复杂度可以降到O(ElogV)，留待以后学习。)

------------------------------------------------------------
POJ 1679 The Unique MST  (MST是否唯一)
求出次小生成树，然后判断权值和是否等于最小生成树即可。
*/
const int MAXN = 103;
const int MAXE = 30003;
/* 链式前向星 */
struct node{        //gragh node
    int u, v, w;
    int op;         //无向边拆成两个有向边对应的另一个编号
    bool in_MST;    //是否在最小生成树中;
    int next;
}edge[MAXE];
int cnt, head[MAXN];
void Init(){
    cnt = 0;
    mem(head, -1);
}
void Add_edge(int u, int v, int w){     //添加无向边
    edge[cnt].in_MST = false;
    edge[cnt].u = u;
    edge[cnt].v = v;
    edge[cnt].w = w;
    edge[cnt].next = head[u];
    edge[cnt].op = cnt + 1;
    head[u] = cnt ++;

    edge[cnt].in_MST = false;
    edge[cnt].u = v;
    edge[cnt].v = u;
    edge[cnt].w = w;
    edge[cnt].next = head[v];
    edge[cnt].op = cnt - 1;
```

```
    head[v] = cnt ++;
}
/* 链式前向星 */

struct prim_node{   //prim node
    int t;      //状态节点标号
    int id;     //第几个边
    int w;      //权值
    friend bool operator < (prim_node n1, prim_node n2){
        return n1.w > n2.w;
    }
    void Init(int n){
        id = -1;
        t = n;
        w = sup;
    }
}dist[MAXN];
bool vis[MAXN];
priority_queue <prim_node, vector<prim_node> > Q;
int MST_NUM;
void Prim(int start, int n){          //初始MST_NUM=0，则对于不连通的图MST_NUM=0;
    mem(vis, 0);
    for (int i = 0; i <= n; i ++)
        dist[i].Init(i);
    while(!Q.empty())
        Q.pop();
    dist[start].w = 0;
    Q.push(dist[start]);
    while(!Q.empty()){
        prim_node tmp = Q.top();
        Q.pop();
        int u = tmp.t;
        int w = tmp.w;
        int id = tmp.id;
        if (vis[u]) continue;
        vis[u] = true;
        if (id != -1){      //确定最小生成树的边和权值和
            edge[id].in_MST = true;
            edge[edge[id].op].in_MST = true;
            MST_NUM += w;
        }
        for (int i = head[u]; i != -1; i = edge[i].next){
            int v = edge[i].v;
            int cost = edge[i].w;
            if (!vis[v] && dist[v].w > cost){
                dist[v].w = cost;
                dist[v].id = i;
                Q.push(dist[v]);
            }
        }
    }
}
```

```
int max_cost_on_MST[MAXN][MAXN];
bool used[MAXN];
void dfs(int s, int t, int maxnum){
    max_cost_on_MST[s][t] = maxnum;
    used[t] = 1;
    for (int i = head[s]; i != -1; i = edge[i].next){
        if (edge[i].in_MST == false)
            continue;
        int v = edge[i].v;
        if (!used[v])   dfs(s, v, max(maxnum, edge[i].w));
    }
}
int n, m;
void Second_MST(){
    Prim(1, n);
    int res = sup;
    //枚举不在最小生成树中的边
    for (int i = 1; i <= n; i ++){
        mem(used, 0);
        dfs(i, i, 0);
        for (int j = head[i]; j != -1; j = edge[j].next){
            int v = edge[j].v;
            if (edge[j].in_MST == false)
                res = min(res, MST_NUM + edge[j].w - max_cost_on_MST[i][v]);
        }
    }
    if (res == MST_NUM){
        printf("Not Unique!\n");
    }
    else{
        printf("%d\n", MST_NUM);
    }
}
int main(){
    int t;
    scanf("%d", &t);
    while(t --){
        MST_NUM = 0;
        Init();
        scanf("%d %d", &n, &m);
        for (int i = 0; i < m; i ++){
            int a, b, l;
            scanf("%d %d %d", &a, &b, &l);
            Add_edge(a, b, l);
        }
        Second_MST();
    }
        return 0;
}
/*-------------------------------------------------------------------------------------------------------------*/
```

最大流(SAP)

```
/*-----------------------------------------------------------------------------------------------------------------*/
const int MAXEDGE=20400;
const int MAXN=400;
const int inf=0x3fffffff;
struct edges
{
    int cap,to,next,flow;
} edge[MAXEDGE+100];
struct nodes
{
    int head,label,pre,cur;
} node[MAXN+100];
int L,N;
int gap[MAXN+100];
void init(int n)
{
    L=0;
    N=n;
    for (int i=0; i<N; i++)
        node[i].head=-1;
}
void add_edge(int x,int y,int z,int w)
{
    edge[L].cap=z;
    edge[L].flow=0;
    edge[L].to=y;
    edge[L].next=node[x].head;
    node[x].head=L++;
    edge[L].cap=w;
    edge[L].flow=0;
    edge[L].to=x;
    edge[L].next=node[y].head;
    node[y].head=L++;
}
int maxflow(int s,int t)
{
    memset(gap,0,sizeof(gap));
    gap[0]=N;
    int u,ans=0;
    for (int i=0; i<N; i++)
    {
        node[i].cur=node[i].head;
        node[i].label=0;
    }
    u=s;
    node[u].pre=-1;
    while (node[s].label<N)
    {
        if (u==t)
        {
```

```
            int min=inf;
            for (int i=node[u].pre; i!=-1; i=node[edge[i^1].to].pre)
                if (min>edge[i].cap-edge[i].flow)
                    min=edge[i].cap-edge[i].flow;
            for (int i=node[u].pre; i!=-1; i=node[edge[i^1].to].pre)
            {
                edge[i].flow+=min;
                edge[i^1].flow-=min;
            }
            u=s;
            ans+=min;
            continue;
        }
        bool flag=false;
        int v;
        for (int i=node[u].cur; i!=-1; i=edge[i].next)
        {
            v=edge[i].to;
            if (edge[i].cap-edge[i].flow && node[v].label+1==node[u].label)
            {
                flag=true;
                node[u].cur=node[v].pre=i;
                break;
            }
        }
        if (flag)
        {
            u=v;
            continue;
        }
        node[u].cur=node[u].head;
        int min=N;
        for (int i=node[u].head; i!=-1; i=edge[i].next)
            if (edge[i].cap-edge[i].flow && node[edge[i].to].label<min)
                min=node[edge[i].to].label;
        gap[node[u].label]--;
        if (!gap[node[u].label]) return ans;
        node[u].label=min+1;
        gap[node[u].label]++;
        if (u!=s) u=edge[node[u].pre^1].to;
    }
    return ans;
}
/*-------------------------------------------------------------------------------------------------------------------*/
```