一、图论

1.Dijkstra

时间复杂度 $O((n+m)logm)$

```
int d[N];
bool st[N];
void dijkstra(int root)
{
    memset(st, 0, sizeof st);
    memset(d, 0x3f, sizeof d);
    d[root] = 0;

    priority_queue<PII, vector<PII>, greater<PII>> heap;
    heap.push((PII){0, root});

    while(!heap.empty())
    {
        int u = heap.top().second;  heap.pop();

        if(st[u])  continue;
        st[u] = true;

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i], dist = d[u] + w[i];
            if(dist < d[j])
            {
                d[j] = dist;
                heap.push((PII){d[j], j});
            }
        }
    }
}
```

1.1严格次短路计数

```
int n, m, S, T;
int d[N][2], cnt[N][2];     //  d[u][0]最短路， d[u][1]次短路
bool st[N][2];
int dijkstra()
{
    for(int i = 1; i <= n; i ++)
        d[i][0] = d[i][1] = INF, cnt[i][0] = cnt[i][1] = st[i][0] = st[i][1] =
0;
    d[S][0] = 0;
    cnt[S][0] = 1;
```

```
    priority_queue<PII, vector<PII>, greater<PII>> heap;
    heap.push({0, S});

    while(heap.size())
    {
        int t = heap.top().first;
        int u = heap.top().second;  heap.pop();

        int k;
        if(t == d[u][0])  k = 0;
        else if(t == d[u][1])  k = 1;
        else  continue;

        if(st[u][k])  continue;
        st[u][k] = true;

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i], dist = d[u][k] + w[i];
            if(dist < d[j][0])
            {
                d[j][1] = d[j][0],  cnt[j][1] = cnt[j][0];
                d[j][0] = dist,     cnt[j][0] = cnt[u][k];
                heap.push({dist, j});
            }
            else if(dist == d[j][0])
            {
                cnt[j][0] += cnt[u][k];
            }
            else if(dist < d[j][1])
            {
                d[j][1] = dist,  cnt[j][1] = cnt[u][k];
                heap.push({dist, j});
            }
            else if(dist == d[j][1])
            {
                cnt[j][1] += cnt[u][k];
            }
        }
    }

    if(d[T][1] == d[T][0] + 1)  cnt[T][0] += cnt[T][1];
    return cnt[T][0];
}
```

2.SPFA

手写循环队列

时间复杂度 $O(nm)$

```
int d[N], q[N];
bool st[N];
void spfa(int root)
{
```

```
    memset(st, 0, sizeof st);
    memset(d, 0x3f, sizeof d);
    d[root] = 0;

    int hh = 0, tt = 0;
    q[tt ++] = root;
    st[root] = true;

    while(hh != tt)
    {
        int u = q[hh ++];
        if(hh == N)  hh = 0;
        st[u] = false;

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i], dist = d[u] + w[i];
            if(dist < d[j])
            {
                d[j] = dist;
                if(!st[j])
                {
                    q[tt ++] = j;
                    if(tt == N)  tt = 0;
                    st[j] = true;
                }
            }
        }
    }
}
```

2.1判断负环

队列换成栈

时间复杂度 $O(nm)$

```
int d[N], q[N], len[N];
bool st[N];
bool spfa()
{
    int tt = -1;
    for(int i = 1; i <= n; i ++)  q[ ++ tt] = i,  len[i] = 1, st[i] = true;

    //int qwq = 0;
    while(tt >= 0)
    {
        int u = q[tt --];
        st[u] = false;

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i], dist = d[u] + w[i];
            if(dist < d[j])
            {
                d[j] = dist;
                len[j] = len[u] + 1;
```

```
                if(len[j] >= n)  return true;
                //if( ++ qwq > 1000000)  return true;
                if(!st[j])
                {
                    q[ ++ tt] = j;
                    st[j] = true;
                }
            }
        }
    }
    return false;
}
```

2.2玄学优化

SLF优化，新节点入队后对比队头队尾元素，把最短路更小的元素交换到队头

```
int d[N], q[N];
bool st[N];
void spfa()
{
    memset(d, 0x3f, sizeof d);
    d[S] = 0;

    int hh = 0, tt = 0;
    q[tt ++] = S;
    st[S] = true;

    while(hh != tt)
    {
        int u = q[hh ++];
        if(hh == N)  hh = 0;
        st[u] = false;

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i], dist = d[u] + w[i];
            if(dist < d[j])
            {
                d[j] = dist;
                if(!st[j])
                {
                    q[tt] = j;
                    if(d[q[hh]] > d[q[tt]])  swap(q[hh], q[tt]);
                    if( ++ tt == N)  tt = 0;
                    st[j] = true;
                }
            }
        }
    }
}
```

3.Floyd

时间复杂度 $O(n^3)$

```
      for(int k = 1; k <= n; k ++)
          for(int i = 1; i <= n; i ++)
              for(int j = 1; j <= n; j ++)
                  g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
```

3.1邻接矩阵快速幂

求恰好经过 k 条路的最短路

时间复杂度 $O(n^3logk)$

```
int n, m;
int g[N][N];
int res[N][N];
void mul(int c[N][N], int a[N][N], int b[N][N])
{
    static int t[N][N];
    memset(t, 0x3f, sizeof t);

    for(int k = 1; k <= n; k ++)
        for(int i = 1; i <= n; i ++)
            for(int j = 1; j <= n; j ++)
                t[i][j] = min(t[i][j], a[i][k] + b[k][j]);

    memcpy(c, t, sizeof t);
}

void qmi(int k)
{
    memset(res, 0x3f, sizeof res);
    for(int i = 1; i <= n; i ++)  res[i][i] = 0;

    while(k)
    {
        if(k & 1)  mul(res, res, g);
        mul(g, g, g);
        k >>= 1;
    }
}
```

3.2传递闭包

给定 n 个变量和 m 个不等式，判断是否有唯一的顺序

输入样例

```
4 6
A<B
A<C
B<C
C<D
B<D
A<B
3 2
A<B
B<A
26 1
A<Z
0 0
```

输出样例

```
Sorted sequence determined after 4 relations: ABCD.
Inconsistency found after 2 relations.
Sorted sequence cannot be determined.
```

```
const int N = 26;

int g[N][N];   //g[a][b]为true表示a<b
int n, m;

void floyd()
{
    for(int k = 0; k < n; k ++)
        for(int i = 0; i < n; i ++)
            for(int j = 0; j < n; j ++)
                if(g[i][k] && g[k][j])  g[i][j] = true;
                //如果i < k且k < j，那么i < j
}

int check()
{
    for(int i = 0; i < n; i ++)
        if(g[i][i])  return 2;

    for(int i = 0; i < n; i ++)
        for(int j = i + 1; j < n; j ++)
            if(!g[i][j] && !g[j][i])  return 0;
    return 1;
}

bool st[N];
int get_min()
{
    int t = -1;
    for(int i = 0; i < n; i ++)
        if(!st[i] && (t == -1 || g[i][t]))  t = i;
    return t;
}
```

```
int main()
{
    while(scanf("%d%d", &n, &m), n || m)
    {
        memset(g, false, sizeof g);
        memset(st, false, sizeof st);

        int type = 0, t;
        //type：0顺序未确定，1确定，2有矛盾
        for(int i = 1; i <= m; i ++)
        {
            char s[5];
            scanf("%s", s);
            int a = s[0] - 'A', b = s[2] - 'A';

            if(!type)       //如果状态未确定
            {
                g[a][b] = true;

                floyd();
                type = check();
                if(type)  t = i;
            }
        }

        if(type == 2)  printf("Inconsistency found after %d relations.\n", t);
        else if(!type)  printf("Sorted sequence cannot be determined.\n");
        else{
            printf("Sorted sequence determined after %d relations: ", t);
            for(int i = 0; i < n; i ++)
            {
                int j = get_min();
                printf("%c", 'A' + j);
                st[j] = true;
            }
            puts(".");
        }
    }
    return 0;
}
```

3.3求带权无向图最小环

求带权无向图中至少包含 3 个点的边权之和最小的环，输出方案

输入数据

```
5 7
1 4 1
1 3 300
3 1 10
1 2 16
2 3 100
2 5 15
5 3 20
```

输出数据

```
1 3 5 2
```

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 110, M = 10010, INF = 0x3f3f3f3f;

int n, m;
int d[N][N], g[N][N];
int pos[N][N];
int path[N], cnt;

void get_path(int i, int j)
{
    if(!pos[i][j])  return;

    int k = pos[i][j];
    get_path(i, k);
    path[cnt ++] = k;
    get_path(k, j);
}

int main()
{
    memset(g, 0x3f, sizeof g);

    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i ++)  g[i][i] = 0;
    while(m --)
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        g[a][b] = g[b][a] = min(g[a][b], c);
    }

    memcpy(d, g, sizeof d);

    int res = INF;
    for(int k = 1; k <= n; k ++)
    {
        for(int i = 1; i < k; i ++)
            for(int j = i + 1; j < k; j ++)
                if((long long)d[i][j] + g[j][k] + g[k][i] < res)
                {
                    res = d[i][j] + g[j][k] + g[k][i];

                    cnt = 0;
                    path[cnt ++] = k;
                    path[cnt ++] = i;
                    get_path(i, j);
                    path[cnt ++] = j;
```

```
                }

        for(int i = 1; i <= n; i ++)
            for(int j = 1; j <= n; j ++)
                if(d[i][j] > d[i][k] + d[k][j])
                {
                    d[i][j] = d[i][k] + d[k][j];
                    pos[i][j] = k;
                }
    }

    if(res == INF)  puts("No solution.");
    else
    {
        for(int i = 0; i < cnt; i ++)  printf("%d ", path[i]);
    }
    return 0;
}
```

4.最小生成树

时间复杂度 $O(mlogm)$

```
struct Edge{
    int a, b, w;
    bool used;
    bool operator< (const struct Edge &E) const {
        return w < E.w;
    }
}e[M];

int n, m;
int p[N];
int find(int x)
{
    if(p[x] != x)  p[x] = find(p[x]);
    return p[x];
}

int kruskal()
{
    for(int i = 1; i <= n; i ++)  p[i] = i;

    sort(e, e + m);

    int res = 0, cnt = 1;
    for(int i = 0; i < m; i ++)
    {
        int a = find(e[i].a),  b = find(e[i].b),  w = e[i].w;
        if(a != b)
        {
            res += w;
            cnt ++;
            e[i].used = true;
            p[a] = b;
        }
```

```
    }
    if(cnt < n)  return -1;
    return res;
}
```

4.1每条边在最小生成树中的出现情况

给出一个 n 个点 m 条边的无向连通图，求问每条边在最小生成树中的出现情况。
如果不在任何最小生成树中出现，输出`none`
如果在至少一棵最小生成树中出现，输出`at least one`
如果再所有最小生成树中都出现，输出`any`
$n, m \leq 1e5$

输入样例

```
4 5
1 2 101
1 3 100
2 3 2
2 4 2
3 4 1
```

输出样例

```
none
any
at least one
at least one
any
```

$tarjan$ **求桥**

首先对所有边从小到大排序，然后同时处理边权相同的边

- 如果边连接的两个点已经在同一连通块中，那么这条边必然不在任何最小生成树中
- 剩下的边要么一定在最小生成树中，要么可能再最小生成树中，把这条边建出来

如果某条边一定在最小生成树中，断开这条边会导致连通块数量增加（截止到目前考虑到的这些边形成的图中）

而这与桥的定义相同，所以我们对于当前考虑的这些边涉及到的点，跑一下 $tarjan$ 给桥打上标记即可

最后再遍历一下这些边，把他们连接的点合并，并清空这些边

时间复杂度 $O(mlogm)$

C++ 代码

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<vector>

using namespace std;
```

```cpp
const int N = 100010, M = N << 1;

struct Edge{
    int a, b, w, type, id;
    bool operator< (const Edge &W) const {
        return w < W.w;
    }
}edges[N];

int h[N], e[M], num[M], ne[M], idx;
void add(int a, int b, int c)
{
    e[idx] = b, num[idx] = c, ne[idx] = h[a], h[a] = idx ++;
}

int p[N];
int find(int x)
{
    if(p[x] != x)  p[x] = find(p[x]);
    return p[x];
}

int dfn[N], low[N], tot;

void tarjan(int u, int fa)          //  fa为抵达u节点的边的编号
{
    dfn[u] = low[u] = ++ tot;
    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i], k = num[i];
        if(k == fa)  continue;
        if(!dfn[j])
        {
            tarjan(j, k);
            low[u] = min(low[u], low[j]);
            if(low[j] > dfn[u])  edges[k].type = 2;
        }
        else
            low[u] = min(low[u], dfn[j]);
    }
}

int res[N];

int main()
{
    memset(h, -1, sizeof h);

    int n, m;
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i++)  p[i] = i;

    for(int i = 0; i < m; i ++)
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        edges[i] = (Edge){a, b, c, 0, i};
    }
```

```
    sort(edges, edges + m);

    for(int i = 0, j; i < m; i = j + 1)
    {
        j = i;
        while(j + 1 < m and edges[j + 1].w == edges[i].w)  j ++;

        for(int k = i; k <= j; k ++)
        {
            int a = find(edges[k].a), b = find(edges[k].b);
            if(a != b)
            {
                add(a, b, k);
                add(b, a, k);
                edges[k].type = 1;
            }
        }
        for(int k = i; k <= j; k ++)
        {
            int a = find(edges[k].a), b = find(edges[k].b);
            if(a != b and !dfn[a])  tarjan(a, -1);
        }
        for(int k = i; k <= j; k ++)
        {
            int a = find(edges[k].a), b = find(edges[k].b);
            if(a != b)
            {
                h[a] = h[b] = -1;
                dfn[a] = dfn[b] = 0;
                p[a] = b;
            }
        }
    }

    for(int i = 0; i < m; i ++)  res[edges[i].id] = edges[i].type;

    for(int i = 0; i < m; i ++)
    {
        if(!res[i])  puts("none");
        else if(res[i] == 1)  puts("at least one");
        else  puts("any");
    }
    return 0;
}
```

4.2严格次小生成树

```
#include<iostream>
#include<algorithm>
#include<cstring>
#include<queue>

using namespace std;

typedef long long LL;

const int N = 100010, M = 600010, INF = 0x3f3f3f3f;
```

```
struct Edges{
    int a, b, w;
    bool used;
    bool operator< (const struct Edges &W) const{
        return w < W.w;
    }
}edges[M];

int n, m;
int f[N];

int find(int x)
{
    if(f[x] != x)  f[x] = find(f[x]);
    return f[x];
}

LL kruskal()
{
    for(int i = 1; i <= n; i ++)  f[i] = i;

    LL res = 0;
    for(int i = 0; i < m; i ++)
    {
        int a = find(edges[i].a), b = find(edges[i].b), w = edges[i].w;
        if(a != b)
        {
            f[a] = b;
            res += w;
            edges[i].used = true;
        }
    }
    return res;
}

int h[N], e[N * 2], w[N * 2], ne[N * 2], idx;
void add(int a, int b, int c)
{
    e[idx] = b;
    w[idx] = c;
    ne[idx] = h[a];
    h[a] = idx ++;
}

void build()
{
    memset(h, -1, sizeof h);
    for(int i = 0; i < m; i ++)
        if(edges[i].used)
        {
            int a = edges[i].a, b = edges[i].b, c = edges[i].w;
            add(a, b, c);
            add(b, a, c);
        }
}

int depth[N];
```

```cpp
int fa[N][17];                  //fa[u][i]为点u向上跳2^i步的位置
int d1[N][17], d2[N][17];       //d1,d2[u][i]分别为点u向上条2^i步的最大边和[严格]次大边（小于最大边的最大边）的边权

void bfs()
{
    depth[1] = 1;

    queue<int> q;
    q.push(1);

    while(!q.empty())
    {
        int u = q.front();  q.pop();

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i];
            if(!depth[j])
            {
                depth[j] = depth[u] + 1;
                q.push(j);

                fa[j][0] = u;
                d1[j][0] = w[i], d2[j][0] = -INF;

                for(int k = 1; k < 17; k ++)
                {
                    int t = fa[j][k - 1];
                    fa[j][k] = fa[t][k - 1];

                    int dist[4] = {d1[j][k - 1], d2[j][k - 1], d1[t][k - 1], d2[t][k - 1]};

                    d1[j][k] = d2[j][k] = -INF;
                    for(int l = 0; l < 4; l ++)
                    {
                        int d = dist[l];
                        if(d > d1[j][k])  d2[j][k] = d1[j][k], d1[j][k] = d;
                        else if(d < d1[j][k] && d > d2[j][k])  d2[j][k] = d;
                        //d严格>d1, d2 = d1, d1 = d, d1严格>d2
                        //d2 < d < d1, d2 = d, d1严格>d2
                    }
                }
            }
        }
    }
}

int lca(int a, int b, int w)            //返回a到b路径上小于w的最大边权
{
    int res = 0;

    if(depth[a] < depth[b])  swap(a, b);

    for(int i = 16; i >= 0; i --)
        if(depth[fa[a][i]] >= depth[b])
        {
```

```
            if(d1[a][i] < w)  res = max(res, d1[a][i]);
            else  res = max(res, d2[a][i]);
            a = fa[a][i];
        }

    if(a == b)  return res;

    for(int i = 16; i >= 0; i --)
        if(fa[a][i] != fa[b][i])
        {
            if(d1[a][i] < w)  res = max(res, d1[a][i]);
            else  res = max(res, d2[a][i]);
            if(d1[b][i] < w)  res = max(res, d1[b][i]);
            else  res = max(res, d2[b][i]);
            a = fa[a][i];
            b = fa[b][i];
        }
    if(d1[a][0] < w)  res = max(res, d1[a][0]);
    else  res = max(res, d2[a][0]);
    if(d1[b][0] < w)  res = max(res, d1[b][0]);
    else  res = max(res, d2[b][0]);

    return res;
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 0; i < m; i ++)  scanf("%d%d%d", &edges[i].a, &edges[i].b,
&edges[i].w);
    sort(edges, edges + m);

    LL sum = kruskal();
    build();
    bfs();

    LL res = 1e18;
    for(int i = 0; i < m; i ++)
        if(!edges[i].used)
        {
            int a = edges[i].a, b = edges[i].b, w = edges[i].w;
            res = min(res, sum + w - lca(a, b, w));
        }
    printf("%lld", res);
    return 0;
}
```

5.拓扑排序

时间复杂度 $O(n+m)$

```
int q[N], d[N];
void topsort()
{
    int hh = 0, tt = -1;
    for(int i = 1; i <= n; i ++)
```

```
        if(!d[i])  q[++ tt] = i;

    while(hh <= tt)
    {
        int u = q[hh ++];

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i];
            if(-- d[j] == 0)  q[++ tt] = j;
        }
    }
}
```

6.差分约束

求一组形如 $x_i \le x_j + c$ 的不等式组的可行解

从源点出发可以遍历到所有边，才能满足所有条件

令差值最大

$x_i \le x_j + c$

`add(j, i, c)`

求最短路，有负环无解

令差值最小

$x_i \ge x_j + c$

`add(j, i, c)`

求最长路，有正环无解

6.1区间

给定 n 个区间 $[a_i, b_i]$ 和 n 个整数 c_i。

你需要构造一个无重复的整数集合 Z，使得 $\forall i \in [1, n]$，Z 中满足$a_i \le x \le b_i$的整数 x 不少于 c_i 个。

求这样的整数集合 Z 最少包含多少个数

```
#include<iostream>
#include<algorithm>
#include<cstring>
#include<queue>

using namespace std;

const int N = 50010, M = 200010;

int h[N], e[M], w[M], ne[M], idx;
void add(int a, int b, int c)
{
    e[idx] = b;
```

```
    w[idx] = c;
    ne[idx] = h[a];
    h[a] = idx ++;
}

int d[N];
queue<int> q;
bool st[N];

void spfa()
{
    memset(d, -0x3f, sizeof d);
    d[0] = 0;

    q.push(0);
    st[0] = true;

    while(!q.empty())
    {
        int u = q.front();  q.pop();
        st[u] = false;
        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i], dist = d[u] + w[i];
            if(dist > d[j])
            {
                d[j] = dist;
                if(!st[j])
                {
                    q.push(j);
                    st[j] = true;
                }
            }
        }
    }
}

int main()
{
    memset(h, -1, sizeof h);

    for(int i = 1; i <= 50001; i ++)
    {
        add(i - 1, i, 0);
        add(i, i - 1, -1);
    }

    int n;
    scanf("%d", &n);
    while(n --)
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        a ++,  b ++;
        add(a - 1, b, c);
    }

    spfa();
```

```
    printf("%d", d[50001]);

    return 0;
}
```

7.倍增求LCA

时间复杂度 预处理$O(nlogn)$ 询问$O(logn)$

```
int dep[N], fa[N][16], q[N];
void bfs(int root)
{
    dep[0] = 0, dep[root] = 1;
    int hh = 0, tt = -1;
    q[ ++ tt] = root;

    while(hh <= tt)
    {
        int u = q[hh ++];

        for(int i = h[u]; ~i ; i = ne[i])
        {
            int j = e[i];
            if(!dep[j])
            {
                dep[j] = dep[u] + 1;
                q[ ++ tt] = j;
                fa[j][0] = u;
                for(int k = 1; k <= 15; k ++)
                    fa[j][k] = fa[fa[j][k - 1]][k - 1];
            }
        }
    }

}

int lca(int a, int b)
{
    if(dep[a] < dep[b])  swap(a, b);

    for(int i = 15; i >= 0; i --)
        if(dep[fa[a][i]] >= dep[b])  a = fa[a][i];

    if(a == b)  return a;

    for(int i = 15; i >= 0; i --)
        if(fa[a][i] != fa[b][i])
        {
            a = fa[a][i];
            b = fa[b][i];
        }
    return fa[a][0];
}
```

8.有向图的强联通分量

强联通分量的编号为拓扑序逆序

一个 DAG 想成为强联通分量至少要增加 $max(cnt[din == 1], cnt[dout == 1])$ 条边

时间复杂度 $O(n)$

```
int dfn[N], low[N], tot;
int id[N], cnt;
int q[N], tt = -1;
bool st[N];
void tarjan(int u)
{
    dfn[u] = low[u] = ++ tot;
    q[ ++ tt] = u;
    st[u] = true;

    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if(!dfn[j])
        {
            tarjan(j);
            low[u] = min(low[u], low[j]);
        }
        else if(st[j])
            low[u] = min(low[u], dfn[j]);
    }

    if(low[u] == dfn[u])
    {
        ++ cnt;
        int y;
        do{
            y = q[tt --];
            st[y] = false;
            id[y] = cnt;
        }while(y != u);
    }
}
```

8.1有源汇DP

求从1号点出发抵达n号点路径上的最大点权-最小点权

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 100010, M = 2e6 + 10, INF = 0x3f3f3f3f;

int h[N], hs[N], e[M], ne[M], idx;
void add(int h[], int a, int b)
```

```
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

int n, m;
int w[N];

int dfn[N], low[N], tot;
int id[N], wmin[N], wmax[N], cnt;
int q[N], tt = -1;
bool st[N];
void tarjan(int u)
{
    dfn[u] = low[u] = ++ tot;
    q[ ++ tt] = u;
    st[u] = true;

    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if(!dfn[j])
        {
            tarjan(j);
            low[u] = min(low[u], low[j]);
        }else if(st[j])
            low[u] = min(low[u], dfn[j]);
    }

    if(dfn[u] == low[u])
    {
        ++ cnt;
        int y;
        do{
            y = q[tt --];
            st[y] = false;
            id[y] = cnt;
            if(!wmin[cnt])
                wmin[cnt] = wmax[cnt] = w[y];
            else
            {
                wmin[cnt] = min(wmin[cnt], w[y]);
                wmax[cnt] = max(wmax[cnt], w[y]);
            }
        }while(y != u);
    }
}

int res, ed;
void dfs(int u, int low, int pro)       //  low为路径上的最小权重，pro为最大利润
{
    low = min(low, wmin[u]);
    pro = max(pro, wmax[u] - low);

    if(u == ed)
    {
        res = max(res, pro);
        return;
    }
```

```
    for(int i = hs[u]; ~i; i = ne[i])
    {
        int j = e[i];
        dfs(j, low, pro);
    }
}

int g[N], f[N];

int main()
{
    memset(h, -1, sizeof h);
    memset(hs, -1, sizeof hs);

    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i ++)  scanf("%d", &w[i]);
    while(m --)
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        add(h, a, b);
        if(c == 2)  add(h, b, a);
    }

    for(int i = 1; i <= n; i ++)
        if(!dfn[i])  tarjan(i);

    for(int i = 1; i <= n; i ++)
        for(int j = h[i]; ~j; j = ne[j])
        {
            int k = e[j];
            int a = id[i], b = id[k];
            if(a != b)  add(hs, a, b);
        }

    //for(int i = 1; i <= n; i ++)  printf("id[%d] = %d\n", i, id[i]);
    /*
    ed = id[n];
    dfs(id[1], 100, 0);
    printf("%d\n", res);
    */

    memset(g, 0x3f, sizeof g);
    memset(st, 0, sizeof st);           //  能否从1号点抵达
    st[id[1]] = true;

    int res = 0;
    for(int i = cnt; i; i --)
        if(st[i])
        {
            g[i] = min(g[i], wmin[i]);
            f[i] = max(f[i], wmax[i] - g[i]);

            //printf("g[%d] = %d    f[%d] = %d\n", i, g[i], i, f[i]);
            for(int j = hs[i]; ~j; j = ne[j])
            {
                int k = e[j];
```

```
                st[k] = true;
                g[k] = min(g[k], g[i]);
                f[k] = max(f[k], f[i]);
            }
        }
    printf("%d\n", f[id[n]]);
    return 0;
}
```

9.无向图的双联通分量

9.1边的双联通分量

一棵无根树想要成为双联通分量至少要增加 $(\text{叶子节点数量}+1)/2$ 条边

时间复杂度 $O(n)$

```
int dfn[N], low[N], tot;
int id[N], cnt;
int q[N], tt = -1;
bool st[M];     //  某条边是否是桥
void tarjan(int u, int fa)
{
    dfn[u] = low[u] = ++ tot;
    q[++ tt] = u;

    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if(!dfn[j])
        {
            tarjan(j, u);
            low[u] =  min(low[u], low[j]);
            if(dfn[u] < low[j])  st[i] = st[i ^ 1] = true;
        }else if(j != fa)
            low[u] = min(low[u], dfn[j]);
    }

    if(dfn[u] == low[u])
    {
        ++ cnt;
        int y;
        do{
            y = q[tt --];
            id[y] = cnt;
        }while(y != u);
    }
}
```

9.2点的双联通分量

时间复杂度 $O(n)$

9.2.1每个点删除后剩余多少连通块

```cpp
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 300010, M = N << 1;

int h[N], e[M], ne[M], idx;
void add(int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

int n, m;
int dfn[N], low[N], tot;
int root, res[N];

void tarjan(int u)
{
    dfn[u] = low[u] = ++ tot;

    int cnt = 0;          //  子树个数
    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if(!dfn[j])
        {
            tarjan(j);
            low[u] = min(low[u], low[j]);
            if(low[j] >= dfn[u])  cnt ++;
        }else
            low[u] = min(low[u], dfn[j]);
    }

    if(u != root)  cnt ++;
    res[u] = cnt;
}

int main()
{
    memset(h, -1, sizeof h);

    scanf("%d%d", &n, &m);
    while(m --)
    {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);
        add(b, a);
    }

    int cnt = 0;         //  连通块个数
    for(int i = 1; i <= n; i ++)
        if(!dfn[i])
        {
            cnt ++;
```

```
            root = i;
            tarjan(root);
        }

    for(int i = 1; i <= n; i ++)  printf("%d ", res[i] + cnt - 1);

    return 0;
}
```

9.2.2矿场搭建

至少设置几个出口使得任意一个点被删除之后，其他点都可以抵达出口，以及设置出口的方案数

```
#include<iostream>
#include<algorithm>
#include<cstring>
#include<cstdio>
#include<vector>

using namespace std;

typedef unsigned long long ULL;

const int N = 1010;

int h[N], e[N], ne[N], idx;
void add(int a, int b)
{
    e[idx] = b;
    ne[idx] = h[a];
    h[a] = idx ++;
}

int dfn[N], low[N], tot;
int root, dcc_cnt;
vector<int> dcc[N];
bool cut[N];        //  是否是割点
int q[N], tt;

void tarjan(int u)
{
    dfn[u] = low[u] = ++ tot;
    q[++ tt] = u;

    if(u == root and h[u] == -1)
    {
        dcc[++ dcc_cnt].push_back(u);
        return;
    }

    int cnt = 0;
    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if(!dfn[j])
        {
            tarjan(j);
```

```
            low[u] = min(low[u], low[j]);
            if(dfn[u] <= low[j])
            {
                cnt ++;
                if(u != root or cnt > 1)  cut[u] = true;

                ++ dcc_cnt;
                int y;
                do{
                    y = q[tt --];
                    dcc[dcc_cnt].push_back(y);
                }while(y != j);
                dcc[dcc_cnt].push_back(u);
            }
        }else
            low[u] = min(low[u], dfn[j]);
    }
}

int n, m;

int main()
{
    int T = 1;
    while(scanf("%d", &m), m)
    {
        for(int i = 1; i <= dcc_cnt; i ++)  dcc[i].clear();
        n = idx = tot = dcc_cnt = 0;
        memset(h, -1, sizeof h);
        memset(dfn, 0, sizeof dfn);
        memset(cut, 0, sizeof cut);

        while(m --)
        {
            int a, b;
            scanf("%d%d", &a, &b);
            add(a, b),  add(b, a);
            n = max(n, a),  n = max(n, b);
        }

        tt = -1;
        for(int i = 1; i <= n; i ++)
            if(!dfn[i])
            {
                root = i;
                tarjan(i);
            }

        int res = 0;
        ULL num = 1;
        for(int i = 1; i <= dcc_cnt; i ++)
        {
            int cnt = 0, n = dcc[i].size();
            for(int j = 0; j < n; j ++)
                if(cut[dcc[i][j]])  cnt ++;

            if(cnt == 0)
            {
```

```
                if(n > 1)  res += 2,  num *= n * (n - 1) / 2;
                else  res ++;
            }
            else if(cnt == 1)  res ++,  num *= n - 1;
        }
        printf("Case %d: %d %llu\n", T ++, res, num);
    }
    return 0;
}
```

10.染色法判定二分图

时间复杂度 $O(n)$

```
bool flag = true;
int col[N];
void dfs(int u, int c)
{
    col[u] = c;
    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if(col[j] == col[u])  flag = false;
        else if(!col[j])  dfs(j, 3 - c);
    }
}

for(int i = 1; i <= n; i ++)
    if(!col[i])  dfs(i, 1);
```

11.二分图

11.1概念和性质

最小点覆盖：选取最少的点使得每条边的两个端点至少有一个被选中

最大独立集：选出最多的点值得任意两点之间都**没有边**

最大团：选出最多的点值得任意两点之间都**有边**

最小路径点覆盖：DAG上用最少的互不相交路径将所有的点覆盖

二分图的最大匹配 = 最小点覆盖 = 总点数 - 最大独立集 = 总点数 - 最小路径点覆盖

最大团 = 补图的最大独立集（补图：对所有边的存在性取反）

11.2匈牙利算法

时间复杂度 $O(nm)$

```
int st[N];
int match[N];
int find(int u)
{
    for(int i = h[u]; ~i; i = ne[i])
```

```
    {
        int j = e[i];
        if(!st[j])
        {
            st[j] = 1;
            if(!match[j] or find(match[j]))
            {
                match[j] = u;
                return j;
            }
        }
    }
    return 0;
}

int main()
{
    int n1, n2, m;
    scanf("%d%d%d", &n1, &n2, &m);

    memset(h, -1, sizeof h);
    while(m --)
    {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);
    }

    int cnt = 0;
    for(int i = 1; i <= n1; i ++)
    {
        memset(st, 0, sizeof st);
        if(find(i))  cnt ++;
    }

    printf("%d", cnt);
    return 0;
}
```

12.网络流

12.1Dinic求最大流

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 10010, M = 200010, INF = 0x3f3f3f3f;

int h[N], e[M], f[M], ne[M], idx;
void add(int a, int b, int c)
{
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx ++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx ++;
```

```
}

int n, m, S, T;
int hs[N], d[N], q[N];

bool bfs()
{
    memset(d, -1, sizeof d);
    d[S] = 0;

    int hh = 0, tt = -1;
    q[ ++ tt] = S;
    hs[S] = h[S];

    while(hh <= tt)
    {
        int u = q[hh ++];

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i];
            if(d[j] == -1 and f[i])
            {
                d[j] = d[u] + 1;
                hs[j] = h[j];
                if(j == T)  return true;
                q[ ++ tt] = j;
            }
        }
    }
    return false;
}

int find(int u, int limit)
{
    if(u == T)  return limit;

    int flow = 0;
    for(int i = hs[u]; ~i and flow < limit; i = ne[i])
    {
        hs[u] = i;
        int j = e[i];
        if(d[j] == d[u] + 1 and f[i])
        {
            int t = find(j, min(f[i], limit - flow));
            if(!t)  d[j] = -1;
            f[i] -= t, f[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}

int dinic()
{
    int res = 0, flow = 0;
    while(bfs())  while(flow = find(S, INF))  res += flow;
    return res;
}
```

```
int main()
{
    memset(h, -1, sizeof h);

    scanf("%d%d%d%d", &n, &m, &S, &T);
    while(m --)
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        add(a, b, c);
    }

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

    return 0;
}
```

12.2二分图的最大匹配

二分图两个集合 $[1,m]$ $[m+1,n]$，求最大匹配并输出方案

时间复杂度 $O(n^{1.5})$

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 110, M = 100010, INF = 0x3f3f3f3f;

int h[N], e[M], f[M], ne[M], idx;
void add(int a, int b, int c)
{
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx ++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx ++;
}

int m, n, S, T;
int hs[N], d[N], q[N];

bool bfs()
{
    memset(d, -1, sizeof d);
    d[S] = 0;

    int hh = 0, tt = -1;
    q[ ++ tt] = S;
    hs[S] = h[S];

    while(hh <= tt)
    {
        int u = q[hh ++];

        for(int i = h[u]; ~i; i = ne[i])
        {
```

```
            int j = e[i];
            if(d[j] == -1 and f[i])
            {
                d[j] = d[u] + 1;
                hs[j] = h[j];
                if(j == T)  return true;
                q[ ++ tt] = j;
            }
        }
    }
    return false;
}

int find(int u, int limit)
{
    if(u == T)  return limit;

    int flow = 0;
    for(int i = hs[u]; ~i and flow < limit; i = ne[i])
    {
        hs[u] = i;
        int j = e[i];
        if(d[j] == d[u] + 1 and f[i])
        {
            int t = find(j, min(f[i], limit - flow));
            if(!t)  d[j] = -1;
            f[i] -= t, f[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}

int dinic()
{
    int res = 0, flow;
    while(bfs())  while(flow = find(S, INF))  res += flow;
    return res;
}

int main()
{
    memset(h, -1, sizeof h);

    scanf("%d%d", &m, &n);
    S = 0, T = n + 1;

    int a, b;
    while(scanf("%d%d", &a, &b), ~a and ~b)  add(a, b, 1);
    for(int i = 1; i <= m; i ++)  add(S, i, 1);
    for(int i = m + 1; i <= n; i ++)  add(i, T, 1);
    printf("%d\n", dinic());

    for(int i = 0; i < idx; i += 2)
    {
        int a = e[i ^ 1], b = e[i];
        if(a >= 1 and a <= m and b > m and b <= n and !f[i])
            printf("%d %d\n", a, b);
    }
```

```
    return 0;
}
```

12.3无源汇上下界可行流

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 210, M = 100010, INF = 0x3f3f3f3f;

int h[N], e[M], f[M], l[M], ne[M], idx;
void add(int a, int b, int c, int d)
{
    e[idx] = b, f[idx] = d - c, l[idx] = c, ne[idx] = h[a], h[a] = idx ++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx ++;
}

int n, m, S, T, A[N];

int hs[N], d[N], q[N];
bool bfs()
{
    memset(d, -1, sizeof d);
    d[S] = 0;

    int hh = 0, tt = -1;
    q[ ++ tt] = S;
    hs[S] = h[S];

    while(hh <= tt)
    {
        int u = q[hh ++];

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i];
            if(d[j] == -1 and f[i])
            {
                d[j] = d[u] + 1;
                hs[j] = h[j];
                if(j == T)  return true;
                q[ ++ tt] = j;
            }
        }
    }
    return false;
}

int find(int u, int limit)
{
    if(u == T)  return limit;

    int flow = 0;
    for(int i = hs[u]; ~i and flow < limit; i = ne[i])
```

```
    {
        hs[u] = i;
        int j = e[i];
        if(d[j] == d[u] + 1 and f[i])
        {
            int t = find(j, min(f[i], limit - flow));
            if(!t)  d[j] = -1;
            f[i] -= t, f[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}

int dinic()
{
    int res = 0, flow;
    while(bfs())  while(flow = find(S, INF))  res += flow;
    return res;
}

int main()
{
    memset(h, -1, sizeof h);

    scanf("%d%d", &n, &m);
    S = 0, T = n + 1;
    for(int i = 0; i < m; i ++)
    {
        int a, b, c, d;
        scanf("%d%d%d%d", &a, &b, &c, &d);
        add(a, b, c, d);
        A[a] -= c, A[b] += c;
    }

    int sum = 0;
    for(int i = 1; i <= n; i ++)
        if(A[i] > 0)  add(S, i, 0, A[i]),  sum += A[i];
        else if(A[i] < 0)  add(i, T, 0, -A[i]);

    if(dinic() != sum)  puts("NO");
    else
    {
        puts("YES");
        for(int i = 0; i < m * 2; i += 2)
            printf("%d\n", f[i ^ 1] + l[i]);
    }
    return 0;
}
```

12.4有源汇上下界最大流

关建边为扩容之后流量可以增加的边

```
#include<cstdio>
#include<cstring>
#include<algorithm>
```

```cpp
using namespace std;

const int N = 510, M = 100010, INF = 0x3f3f3f3f;

int h[N], e[M], f[M], ne[M], idx;
void add(int a, int b, int c)
{
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx ++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx ++;
}

int n, m, S, T;
int A[N];

int hs[N], q[N], d[N];
bool bfs()
{
    memset(d, -1, sizeof d);
    d[S] = 0;

    int hh = 0, tt = -1;
    q[ ++ tt] = S;
    hs[S] = h[S];

    while(hh <= tt)
    {
        int u = q[hh ++];

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i];
            if(d[j] == -1 and f[i])
            {
                d[j] = d[u] + 1;
                hs[j] = h[j];
                if(j == T)  return true;
                q[ ++ tt] = j;
            }
        }
    }
    return false;
}

int find(int u, int limit)
{
    if(u == T)  return limit;

    int flow = 0;
    for(int i = hs[u]; ~i and flow < limit; i = ne[i])
    {
        hs[u] = i;
        int j = e[i];
        if(d[j] == d[u] + 1 and f[i])
        {
            int t = find(j, min(f[i], limit - flow));
            if(!t)  d[j] = -1;
            f[i] -= t, f[i ^ 1] += t, flow += t;
        }
```

```
    }
    return flow;
}

int dinic()
{
    int res = 0, flow;
    while(bfs())  while(flow = find(S, INF))  res += flow;
    return res;
}

int main()
{
    memset(h, -1, sizeof h);

    int s, t;
    scanf("%d%d%d%d", &n, &m, &s, &t);
    S = 0, T = n + 1;

    while(m --)
    {
        int a, b, c, d;
        scanf("%d%d%d%d", &a, &b, &c, &d);
        add(a, b, d - c);
        A[a] -= c, A[b] += c;
    }

    int sum = 0;
    for(int i = 1; i <= n; i ++)
        if(A[i] > 0)  add(S, i, A[i]),  sum += A[i];
        else if(A[i] < 0)  add(i, T, -A[i]);

    add(t, s, INF);
    if(dinic() != sum)  puts("No Solution");
    else
    {
        int res = f[idx - 1];
        f[idx - 2] = f[idx - 1] = 0;
        S = s, T = t;
        res += dinic();
        printf("%d\n", res);
    }
    return 0;
}
```

12.5最大流关建边

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 510, M = 100010, INF = 0x3f3f3f3f;

int h[N], e[M], f[M], ne[M], idx;
void add(int a, int b, int c)
```

```
{
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx ++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx ++;
}

int n, m, S, T;
int hs[N], d[N], q[N];

bool bfs()
{
    memset(d, -1, sizeof d);
    d[S] = 0;

    int hh = 0, tt = -1;
    q[ ++ tt] = S;
    hs[S] = h[S];

    while(hh <= tt)
    {
        int u = q[hh ++];

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i];
            if(d[j] == -1 and f[i])
            {
                d[j] = d[u] + 1;
                hs[j] = h[j];
                if(j == T)  return true;
                q[ ++ tt] = j;
            }
        }
    }
    return false;
}

int find(int u, int limit)
{
    if(u == T)  return limit;

    int flow = 0;
    for(int i = hs[u]; ~i and flow < limit; i = ne[i])
    {
        hs[u] = i;
        int j = e[i];
        if(d[j] == d[u] + 1 and f[i])
        {
            int t = find(j, min(f[i], limit - flow));
            if(!t)  d[j] = -1;
            f[i] -= t, f[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}

void dinic()
{
    while(bfs())  while(find(S, INF));
```

```
}

bool st0[N], st1[N];
void dfs(int u, bool st[N], int t)
{
    st[u] = true;
    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if(!st[j] and f[i ^ t])
            dfs(j, st, t);
    }
}

int main()
{
    memset(h, -1, sizeof h);

    scanf("%d%d", &n, &m);
    S = 0, T = n - 1;
    while(m --)
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        add(a, b, c);
    }

    dinic();
    dfs(S, st0, 0);
    dfs(T, st1, 1);

    int res = 0;
    for(int i = 0; i < idx; i += 2)
        if(!f[i] and st0[e[i ^ 1]] and st1[e[i]])  res ++;
    printf("%d\n", res);
    return 0;
}
```

12.6最小路径覆盖问题

问题：**有向图中选取最少的路径覆盖所有点**

原图中的每个点拆成两个点分别放入点集 A, B 中，原图中的每条边转化为从 A 到 B 的边

路径不相交的最小路径覆盖 = 点数 - 二分图的最大匹配数

路径可相交则要提前求一边传递闭包

12.7最大权闭合子图

问题：**带点权的有向图，求权值之和最大的点集，使得点集内的点所能抵达的点也必须在集合内**

源点向所有正权点建边，容量为正权点权重

所有负权点向汇点建边，容量为负权点权重的绝对值

原图中的边容量为正无穷

最大利润 = 正权点之和 - 最小割

```cpp
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 55010, M = 1e6 + 10, INF = 0x3f3f3f3f;

int h[N], e[M], f[M], ne[M], idx;
void add(int a, int b, int c)
{
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx ++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx ++;
}

int n, m, S, T;
int w[N];

int hs[N], d[N], q[N];
bool bfs()
{
    memset(d, -1, sizeof d);
    d[S] = 0;

    int hh = 0, tt = -1;
    q[ ++ tt] = S;
    hs[S] = h[S];

    while(hh <= tt)
    {
        int u = q[hh ++];

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i];
            if(d[j] == -1 and f[i])
            {
                d[j] = d[u] + 1;
                hs[j] = h[j];
                if(j == T)  return true;
                q[ ++ tt] = j;
            }
        }
    }
    return false;
}

int find(int u, int limit)
{
    if(u == T)  return limit;

    int flow = 0;
    for(int i = hs[u]; ~i and flow < limit; i = ne[i])
    {
        hs[u] = i;
        int j = e[i];
        if(d[j] == d[u] + 1 and f[i])
```

```
        {
            int t = find(j, min(f[i], limit - flow));
            if(!t)  d[j] = -1;
            f[i] -= t, f[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}

int dinic()
{
    int res = 0, flow;
    while(bfs())  while(flow = find(S, INF))  res += flow;
    return res;
}

int main()
{
    memset(h, -1, sizeof h);

    scanf("%d%d", &n, &m);
    S = 0, T = n + m + 1;
    for(int i = 1; i <= n; i ++)  scanf("%d", &w[i]),  add(i, T, w[i]);

    int sum = 0;
    for(int i = n + 1; i <= n + m; i ++)
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        sum += c;
        add(S, i, c);
        add(i, a, INF);
        add(i, b, INF);
    }

    printf("%d\n", sum - dinic());
    return 0;
}
```

11.8最大密度子图

01分数规划

12.9二分图的最小权点覆盖&&最大权独立集

最小权点覆盖：**选择点权最小的一个点集使得每条边连接的两个点至少有一个被选中**

最大权独立集：**选择一个点权之和最大的点集使得被选中的点两两之间没有边相连**

一般图的最小权点覆盖：NPC问题

二分图的最小权点覆盖：点权变为连向源点或汇点的边的容量，原图中的边容量设为正无穷，求最小割

最大权独立集：总点权 - 最小权点覆盖

```
#include<cstdio>
#include<cstring>
#include<algorithm>
```

```cpp
#include<vector>

using namespace std;

const int N = 210, M = 20010, INF = 0x3f3f3f3f;

int h[N], e[M], f[M], ne[M], idx;
void add(int a, int b, int c)
{
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx ++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx ++;
}

int n, m, S, T;
int hs[N], d[N], q[N];
bool bfs()
{
    memset(d, -1, sizeof d);
    d[S] = 0;

    int hh = 0, tt = -1;
    q[ ++ tt] = S;
    hs[S] = h[S];

    while(hh <= tt)
    {
        int u = q[hh ++];

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i];
            if(d[j] == -1 and f[i])
            {
                d[j] = d[u] + 1;
                hs[j] = h[j];
                if(j == T)  return true;
                q[ ++ tt] = j;
            }
        }
    }
    return false;
}

int find(int u, int limit)
{
    if(u == T)  return limit;

    int flow = 0;
    for(int i = hs[u]; ~i and flow < limit; i = ne[i])
    {
        hs[u] = i;
        int j = e[i];
        if(d[j] == d[u] + 1 and f[i])
        {
            int t = find(j, min(f[i], limit - flow));
            if(!t)  d[j] = -1;
            f[i] -= t, f[i ^ 1] += t, flow += t;
        }
```

```
    }
    return flow;
}

int dinic()
{
    int res = 0, flow;
    while(bfs())  while(flow = find(S, INF))  res += flow;
    return res;
}

bool st[N];
void dfs(int u)
{
    st[u] = true;
    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if(!st[j] and f[i])
            dfs(j);
    }
}

int main()
{
    memset(h, -1, sizeof h);

    scanf("%d%d", &n, &m);
    S = 0, T = n * 2 + 1;

    for(int i = 1; i <= n; i ++)
    {
        int w;
        scanf("%d", &w);
        add(S, i, w);
    }

    for(int i = 1; i <= n; i ++)
    {
        int w;
        scanf("%d", &w);
        add(n + i, T, w);
    }

    while(m --)
    {
        int a, b;
        scanf("%d%d", &a, &b);
        add(b, n + a, INF);
    }

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

    dfs(S);

    vector<int> res;
    for(int i = 0; i < idx; i += 2)
    {
```

```
        int a = e[i ^ 1], b = e[i];
        if(st[a] and !st[b])  res.push_back(i);
    }

    printf("%d\n", res.size());
    for(int i : res)
    {
        int a = e[i ^ 1], b = e[i];
        if(a == S)  printf("%d +\n", b);
        else if(b == T)  printf("%d -\n", a - n);
    }
    return 0;
}
```

12.10最小费用最大流

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 5010, M = 100010, INF = 0x3f3f3f3f;

int h[N], e[M], f[M], w[M], ne[M], idx;
void add(int a, int b, int c, int d)
{
    e[idx] = b, f[idx] = c, w[idx] = d, ne[idx] = h[a], h[a] = idx ++;
    e[idx] = a, f[idx] = 0, w[idx] = -d, ne[idx] = h[b], h[b] = idx ++;
}

int n, m, S, T;
int d[N], incf[N], q[N], pre[N];
bool st[N];

bool spfa()
{
    memset(d, 0x3f, sizeof d);
    memset(incf, 0, sizeof incf);
    d[S] = 0, incf[S] = INF;

    int hh = 0, tt = 0;
    q[tt ++] = S;
    st[S] = true;

    while(hh != tt)
    {
        int u = q[hh ++];
        if(hh == N)  hh = 0;
        st[u] = false;

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i], dist = d[u] + w[i];
            if(f[i] and dist < d[j])
            {
                d[j] = dist;
```

```
                incf[j] = min(f[i], incf[u]);
                pre[j] = i;
                if(!st[j])
                {
                    q[tt ++] = j;
                    if(tt == N)  tt = 0;
                    st[j] = true;
                }
            }
        }
    }
    return incf[T];
}

void EK(int &flow, int &cost)
{
    flow = cost = 0;
    while(spfa())
    {
        int t = incf[T];
        flow += t, cost += t * d[T];
        for(int i = T; i != S; i = e[pre[i] ^ 1])
        {
            f[pre[i]] -= t;
            f[pre[i] ^ 1] += t;
        }
    }
}

int main()
{
    memset(h, -1, sizeof h);

    scanf("%d%d%d%d", &n, &m, &S, &T);
    while(m --)
    {
        int a, b, c, d;
        scanf("%d%d%d%d", &a, &b, &c, &d);
        add(a, b, c, d);
    }

    int flow, cost;
    EK(flow, cost);
    printf("%d %d\n", flow, cost);

    return 0;
}
```

12.11最小费用上下界可行流

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 2010, M = 100010, INF = 0x3f3f3f3f;
```

```
int h[N], e[M], f[M], w[M], ne[M], idx;
void add(int a, int b, int c, int d)
{
    e[idx] = b, f[idx] = c, w[idx] = d, ne[idx] = h[a], h[a] = idx ++;
    e[idx] = a, f[idx] = 0, w[idx] = -d, ne[idx] = h[b], h[b] = idx ++;
}

int n, m, S, T;

int d[N], incf[N], pre[N], q[N];
bool st[N];

bool spfa()
{
    memset(d, 0x3f, sizeof d);
    memset(incf, 0, sizeof incf);
    d[S] = 0, incf[S] = INF;

    int hh = 0, tt = 0;
    q[tt ++] = S;
    st[S] = true;

    while(hh != tt)
    {
        int u = q[hh ++];
        if(hh == N)  hh = 0;
        st[u] = false;

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i], dist = d[u] + w[i];
            if(f[i] and dist < d[j])
            {
                d[j] = dist;
                incf[j] = min(f[i], incf[u]);
                pre[j] = i;
                if(!st[j])
                {
                    q[tt ++] = j;
                    if(tt == N)  tt = 0;
                    st[j] = true;
                }
            }
        }
    }
    return incf[T];
}

int EK()
{
    int res = 0;
    while(spfa())
    {
        int t = incf[T];
        res += t * d[T];
        for(int i = T; i != S; i = e[pre[i] ^ 1])
        {
```

```
            f[pre[i]] -= t;
            f[pre[i] ^ 1] += t;
        }
    }
    return res;
}

int main()
{
    memset(h, -1, sizeof h);

    scanf("%d%d", &n, &m);
    S = N - 2, T = N - 1;

    int last = 0;
    for(int i = 1; i <= n; i ++)
    {
        int c;
        scanf("%d", &c);
        if(last > c)  add(S, i, last - c, 0);
        else if(last < c)  add(i, T, c - last, 0);
        add(i, i + 1, INF - c, 0);
        last = c;
    }
    add(S, n + 1, last, 0);

    while(m --)
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        add(b + 1, a, INF, c);
    }

    printf("%d\n", EK());
    return 0;
}
```

12.12二分图的最大权匹配

有 n 件工作要分配给 n 个人做。

第 i 个人做第 j 件工作产生的效益为 c_{ij}。

试设计一个将 n 件工作分配给 nn 个人做的分配方案。

对于给定的 n 件工作和 n 个人，计算最优分配方案和最差分配方案。

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 110, M = 100010, INF = 0x3f3f3f3f;

int h[N], e[M], f[M], w[M], ne[M], idx;
void add(int a, int b, int c, int d)
{
```

```
    e[idx] = b, f[idx] = c, w[idx] = d, ne[idx] = h[a], h[a] = idx ++;
    e[idx] = a, f[idx] = 0, w[idx] = -d, ne[idx] = h[b], h[b] = idx ++;
}

int n, S, T;

int d[N], incf[N], pre[N], q[N];
bool st[N];

bool spfa()
{
    memset(d, 0x3f, sizeof d);
    memset(incf, 0, sizeof incf);
    d[S] = 0, incf[S] = INF;

    int hh = 0, tt = 0;
    q[tt ++] = S;
    st[S] = true;

    while(hh != tt)
    {
        int u = q[hh ++];
        if(hh == N)  hh = 0;
        st[u] = false;

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i], dist = d[u] + w[i];
            if(f[i] and dist < d[j])
            {
                d[j] = dist;
                incf[j] = min(f[i], incf[u]);
                pre[j] = i;
                if(!st[j])
                {
                    q[tt ++] = j;
                    if(tt == N)  tt = 0;
                    st[j] = true;
                }
            }
        }
    }
    return incf[T];
}

bool spfa2()
{
    memset(d, 0xcf, sizeof d);
    memset(incf, 0, sizeof incf);
    d[S] = 0, incf[S] = INF;

    int hh = 0, tt = 0;
    q[tt ++] = S;
    st[S] = true;

    while(hh != tt)
    {
        int u = q[hh ++];
```

```
        if(hh == N)  hh = 0;
        st[u] = false;

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i], dist = d[u] + w[i];
            if(f[i] and dist > d[j])
            {
                d[j] = dist;
                incf[j] = min(f[i], incf[u]);
                pre[j] = i;
                if(!st[j])
                {
                    q[tt ++] = j;
                    if(tt == N)  tt = 0;
                    st[j] = true;
                }
            }
        }
    }
    return incf[T];
}

void EK()
{
    int res = 0;
    while(spfa())
    {
        int t = incf[T];
        res += t * d[T];
        for(int i = T; i != S; i = e[pre[i] ^ 1])
        {
            f[pre[i]] -= t;
            f[pre[i] ^ 1] += t;
        }
    }
    printf("%d\n", res);

    for(int i = 0; i < idx; i += 2)
        f[i] += f[i ^ 1], f[i ^ 1] = 0;
    res = 0;

    while(spfa2())
    {
        int t = incf[T];
        res += t * d[T];
        for(int i = T; i != S; i = e[pre[i] ^ 1])
        {
            f[pre[i]] -= t;
            f[pre[i] ^ 1] += t;
        }
    }
    printf("%d\n", res);
}

int main()
{
    memset(h, -1, sizeof h);
```

```
    scanf("%d", &n);
    S = 0, T = N - 1;
    for(int i = 1; i <= n; i ++)
        for(int j = n + 1; j <= n * 2; j ++)
        {
            int w;
            scanf("%d", &w);
            add(i, j, 1, w);
        }
    for(int i = 1; i <= n; i ++)  add(S, i, 1, 0);
    for(int i = n + 1; i <= n * 2; i ++)  add(i, T, 1, 0);
    EK();
    return 0;
}
```

13. 2-SAT问题

$a \vee b \iff \neg a \to b \iff \neg b \to a$

`a or b <==> 当a为0时b必定为1 <==> 当b为0时a必定为1`

对于每一个 $a \vee b$ 的关系，建 `!a -> b` 和 `!b -> a` 两条边
建图关系为当某个点成立时，那么所有它能抵达的点都要成立

当且仅当存在 u，使得 `u` 和 `!u` 在同一强联通分量内时，无解

若 `u` 与 `!u` 不在同一强联通分量内，则他们中拓扑序靠前的点**有可能**可以抵达靠后的点
我们令其中拓扑序靠后的点成立（若可以抵达，则当前者成立时，后者也需成立，矛盾）
由于 $tarjan$ 所求强联通分量编号为拓扑序逆序，所以令所在强联通分量编号较小的点成立即可

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 2e6 + 10, M = N;

int h[N], e[M], ne[M], idx;
void add(int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

int dfn[N], low[N], tot;
int id[N], cnt;
int q[N], tt = -1;
bool st[N];
void tarjan(int u)
{
    dfn[u] = low[u] = ++ tot;
    q[ ++ tt] = u;
    st[u] = true;

    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
```

```
        if(!dfn[j])
        {
            tarjan(j);
            low[u] = min(low[u], low[j]);
        }else if(st[j])
            low[u] = min(low[u], dfn[j]);
    }

    if(dfn[u] == low[u])
    {
        ++ cnt;
        int y;
        do{
            y = q[tt --];
            st[y] = false;
            id[y] = cnt;
        }while(y != u);
    }
}

int n, m;
int get(int i, int a)
{
    return i + a * n;
}

int main()
{
    memset(h, -1, sizeof h);

    scanf("%d%d", &n, &m);
    while(m --)
    {
        int a, b, i, j;
        scanf("%d%d%d%d", &i, &a, &j, &b);
        add(get(i, !a), get(j, b));
        add(get(j, !b), get(i, a));
    }

    for(int i = 1; i <= n * 2; i ++)
        if(!dfn[i])  tarjan(i);

    for(int i = 1; i <= n; i ++)
        if(id[i] == id[n + i])  return 0 * printf("IMPOSSIBLE");

    puts("POSSIBLE");
    for(int i = 1; i <= n; i ++)
        if(id[i] < id[n + i])  printf("0 ");
        else  printf("1 ");
    return 0;
}
```

14.朱刘算法

```
#include<cstdio>
#include<cstring>
```

```cpp
#include<algorithm>
#include<cmath>

#define x first
#define y second

using namespace std;

typedef pair<double, double> PDD;

const int N = 110, INF = 0x3f3f3f3f;

int n, m;
PDD p[N];
bool g[N][N];
double d[N][N], bd[N][N];
int pre[N], bpre[N];
bool st[N];

void dfs(int u)
{
    st[u] = true;
    for(int i = 1; i <= n; i ++)
        if(g[u][i] and !st[i])  dfs(i);
}

bool check()
{
    memset(st, 0, sizeof st);
    dfs(1);
    for(int i = 1; i <= n; i ++)
        if(!st[i])  return false;
    return true;
}

double dist(int i, int j)
{
    double dx = p[i].x - p[j].x, dy = p[i].y - p[j].y;
    return sqrt(dx * dx + dy * dy);
}

int dfn[N], low[N], tot;
int id[N], cnt;
int q[N], tt = -1;

void tarjan(int u)
{
    dfn[u] = low[u] = ++ tot;
    q[ ++ tt] = u;
    st[u] = true;

    int j = pre[u];
    if(!dfn[j])
    {
        tarjan(j);
        low[u] = min(low[u], low[j]);
    }else if(st[j])
        low[u] = min(low[u], dfn[j]);
```

```
    if(dfn[u] == low[u])
    {
        ++ cnt;
        int y;
        do{
            y = q[tt --];
            st[y] = false;
            id[y] = cnt;
        }while(y != u);
    }
}

double work()
{
    double res = 0;
    for(int i = 1; i <= n; i ++)
        for(int j = 1; j <= n; j ++)
            if(g[i][j])  d[i][j] = dist(i, j);
            else  d[i][j] = INF;

    while(true)
    {
        for(int i = 1; i <= n; i ++)
        {
            pre[i] = i;
            for(int j = 1; j <= n; j ++)
                if(d[j][i] < d[pre[i]][i])  pre[i] = j;
        }

        memset(dfn, 0, sizeof dfn);
        memset(st, 0, sizeof st);
        tot = cnt = 0;

        for(int i = 1; i <= n; i ++)
            if(!dfn[i])  tarjan(i);

        if(cnt == n)
        {
            for(int i = 2; i <= n; i ++)  res += d[pre[i]][i];
            break;
        }

        for(int i = 2; i <= n; i ++)
            if(id[pre[i]] == id[i])  res += d[pre[i]][i];

        for(int i = 1; i <= cnt; i ++)
            for(int j = 1; j <= cnt; j ++)
                bd[i][j] = INF;

        for(int i = 1; i <= n; i ++)
            for(int j = 1; j <= n; j ++)
                if(d[i][j] < INF and id[i] != id[j])
                {
                    int a = id[i], b = id[j];
                    if(id[pre[j]] == id[j])  bd[a][b] = min(bd[a][b], d[i][j] -
d[pre[j]][j]);
                    else  bd[a][b] = min(bd[a][b], d[i][j]);
```

```
                }

            n = cnt;
            memcpy(d, bd, sizeof d);
        }
        return res;
}

int main()
{
    while(~scanf("%d%d", &n, &m))
    {
        memset(g, 0, sizeof g);

        for(int i = 1; i <= n; i ++)  scanf("%lf%lf", &p[i].x, &p[i].y);
        while(m --)
        {
            int a, b;
            scanf("%d%d", &a, &b);
            if(a != b and b != 1)  g[a][b] = true;
        }

        if(!check())  puts("poor snoopy");
        else  printf("%.2lf\n", work());
    }
    return 0;
}
```

二、数据结构

1.树状数组

单点修改，维护前缀和

时间复杂度 $O(logn)$

```
int tr[N], n;

void add(int x, int c)
{
    for(int i = x; i <= n; i += i & -i)  tr[i] += c;
}

int sum(int x)
{
    int res = 0;
    for(int i = x; i; i -= i & -i)  res += tr[i];
    return res;
}
```

2.线段树

2.1区间加/区间乘/区间求和取模

```
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

typedef long long LL;

const int N = 100010;

struct Node{
    int l, r;
    LL sum, add, mul;
}tr[N * 4];

int n, mod;
int a[N];

void pushup(int u)
{
    tr[u].sum = (tr[u << 1].sum + tr[u << 1 | 1].sum) % mod;
}

void build(int u, int l, int r)
{
    tr[u] = {l, r, 0, 0, 1};
    if(l == r)
    {
        tr[u].sum = a[l] % mod;
        return;
    }
    int mid = l + r >> 1;
    build(u << 1, l, mid),  build(u << 1 | 1, mid + 1, r);
    pushup(u);
}

void eval(int u, int add, int mul)
{
    tr[u].sum = (tr[u].sum * mul + (LL)(tr[u].r - tr[u].l + 1) * add) % mod;
    tr[u].mul = (tr[u].mul * mul) % mod;
    tr[u].add = (tr[u].add * mul + add) % mod;
}

void pushdown(int u)
{
    eval(u << 1, tr[u].add, tr[u].mul);
    eval(u << 1 | 1, tr[u].add, tr[u].mul);
    tr[u].add = 0;
    tr[u].mul = 1;
}

LL query(int u, int l, int r)
{
    if(tr[u].l >= l && tr[u].r <= r)  return tr[u].sum;
```

```
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    LL sum = 0;
    if(l <= mid)  sum = query(u << 1, l, r);
    if(r > mid)  sum = (sum + query(u << 1 | 1, l, r)) % mod;
    return sum;
}

void modify(int u, int l, int r, int add, int mul)
{
    if(tr[u].l >= l && tr[u].r <= r)
    {
        eval(u, add, mul);
        return;
    }

    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    if(l <= mid)  modify(u << 1, l, r, add, mul);
    if(r > mid)  modify(u << 1 | 1, l, r, add, mul);
    pushup(u);
}

int main()
{
    scanf("%d%d", &n, &mod);
    for(int i = 1; i <= n; i ++)  scanf("%d", &a[i]);
    build(1, 1, n);

    int m;
    scanf("%d", &m);
    while(m --)
    {
        int t, l, r, c;
        scanf("%d%d%d", &t, &l, &r);
        if(t == 3)  printf("%lld\n", query(1, l, r));
        else{
            scanf("%d", &c);
            if(t == 2)  modify(1, l, r, c, 1);
            else  modify(1, l, r, 0, c);
        }
    }
    return 0;
}
```

2.2区间加/区间gcd

```
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

typedef long long LL;

const int N = 500010;
```

```
LL gcd(LL a, LL b)
{
    return b ? gcd(b, a % b) : a;
}

struct Node{
    int l, r;
    LL sum, d;
}tr[N * 4];

int n, m;
LL a[N];

void pushup(Node &u, Node &l, Node &r)
{
    u.sum = l.sum + r.sum;
    u.d = gcd(l.d, r.d);
}

void pushup(int u)
{
    pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
}

void build(int u, int l, int r)
{
    tr[u] = {l, r};
    if(l == r)
    {
        tr[u].sum = tr[u].d = a[l] - a[l - 1];
        return;
    }
    int mid = l + r >> 1;
    build(u << 1, l, mid),  build(u << 1 | 1, mid + 1, r);
    pushup(u);
}

void modify(int u, int x, LL c)
{
    if(tr[u].l == x && tr[u].r == x)
    {
        tr[u].sum += c;
        tr[u].d += c;
        return;
    }

    int mid = tr[u].l + tr[u].r >> 1;
    if(x <= mid)  modify(u << 1, x, c);
    else  modify(u << 1 | 1, x, c);
    pushup(u);
}

Node query(int u, int l, int r)
{
    if(tr[u].l >= l && tr[u].r <= r)  return tr[u];

    int mid = tr[u].l + tr[u].r >> 1;
    if(r <= mid)  return query(u << 1, l, r);
```

```
    else if(l > mid)  return query(u << 1 | 1, l, r);
    else{
        Node res;
        Node left = query(u << 1, l, r);
        Node right = query(u << 1 | 1, l, r);
        pushup(res, left, right);
        return res;
    }
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i ++)  scanf("%lld", &a[i]);

    build(1, 1, n);

    while(m --)
    {
        char s[3];
        int l, r;
        scanf("%s%d%d", s, &l, &r);
        if(s[0] == 'C')
        {
            LL c;
            scanf("%lld", &c);
            modify(1, l, c);
            if(r + 1 <= n)  modify(1, r + 1, -c);
        }else{
            Node left = query(1, 1, l);
            Node right = query(1, l + 1, r);
            printf("%lld\n", abs(gcd(left.sum, right.d)));
        }
    }
    return 0;
}
```

2.3扫描线

```
#include<iostream>
#include<algorithm>
#include<cstring>
#include<vector>

using namespace std;

const int N = 100010;

struct Segment{
    double x;
    double y1, y2;
    int w;
    bool operator< (const struct Segment &W) const{
        return x < W.x;
    }
}seg[N * 2];
```

```
struct Node{
    int l, r;
    int cnt;
    double len;
}tr[N * 8];

vector<double> ys;

int find(double y)
{
    return lower_bound(ys.begin(), ys.end(), y) - ys.begin();
}

void build(int u, int l, int r)
{
    tr[u] = {l, r, 0, 0};

    if(l == r)  return;

    int mid = l + r >> 1;
    build(u << 1, l, mid),  build(u << 1 | 1, mid + 1, r);
}

void pushup(int u)
{
    if(tr[u].cnt)  tr[u].len = ys[tr[u].r + 1] - ys[tr[u].l];
    else if(tr[u].l != tr[u].r)  tr[u].len = tr[u << 1].len + tr[u << 1 |
1].len;
    else tr[u].len = 0;
}

void modify(int u, int l, int r, int w)
{
    if(tr[u].l >= l && tr[u].r <= r)
    {
        tr[u].cnt += w;
        pushup(u);
        return;
    }

    int mid = tr[u].l + tr[u].r >> 1;
    if(l <= mid)  modify(u << 1, l, r, w);
    if(r > mid)  modify(u << 1 | 1, l, r, w);
    pushup(u);
}

int main()
{
    int n, t = 1;
    while(scanf("%d", &n), n)
    {
        ys.clear();
        for(int i = 0; i < n; i ++)
        {
            double x1, y1, x2, y2;
            scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
            seg[i * 2] = {x1, y1, y2, 1};
            seg[i * 2 + 1] = {x2, y1, y2, -1};
```

```
            ys.push_back(y1),  ys.push_back(y2);
        }
        sort(ys.begin(), ys.end());
        ys.erase(unique(ys.begin(), ys.end()), ys.end());
        build(1, 0, ys.size() - 2);

        sort(seg, seg + n * 2);

        double res = 0;
        for(int i = 0; i < n * 2; i ++)
        {
            if(i)  res += tr[1].len * (seg[i].x - seg[i - 1].x);
            modify(1, find(seg[i].y1), find(seg[i].y2) - 1, seg[i].w);
        }
        printf("Test case #%d\nTotal explored area: %.2lf\n\n", t ++, res);
    }
    return 0;
}
```

2.4线段树维护字符串哈希

给出 n 个单词，再给出一个字符串，有 m 次操作，每次修改字符串中的某一个字母，或询问字符串的某一个区间是否是一个单词。T 组测试数据。
$n \leq 10^4$
$lensum(\text{单词}) \leq 2 \cdot 10^6$
$len(\text{字符串}) \leq 10^5$

首先数据范围要求我们必须在 $mlogn$ 的时间复杂度内解决问题，
这样我们必然不可能每次查询都去遍历字符串，所以要用线段树维护字符串哈希来解决这个问题。

线段树维护区间哈希值，pushup可以**用左子节点的哈希值乘以右子节点的长度加上右子节点的哈希值**
大概这样 `root.hash = left.hash * right.len + right.hash`

单点修改就很好办了，但是查询中有一个很容易错的地方。

当我们的查询区间横跨左右两个子节点的时候，
递归到左子节点查询的时候需要把查询区间的右端点改为 mid
`return query(u << 1, l, ` mid `) * p[r - mid] + query(u << 1 | 1, l, r);`

```
#include<cstring>
#include<cstdio>
#include<set>

using namespace std;

const int N = 100010, P = 131;

typedef unsigned long long LL;

struct note
{
    int l,r;
    LL h;
}tr[N * 4];

char s[N];
LL p[N];
```

```
int n;

LL haxi(char s[])
{
    LL res = 0;
    for(int i = 0; s[i]; i ++)  res = res * P + s[i];
    return res;
}

void build(int u, int l, int r)
{
    tr[u] = {l, r};
    if(l == r)
    {
        tr[u].h = s[l];
        return;
    }
    int mid = l + r >> 1;
    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);

    tr[u].h = tr[u << 1].h * p[r - mid] + tr[u << 1 | 1].h;
}

void modify(int u, int x, char c)
{
    if(tr[u].l == tr[u].r)
    {
        tr[u].h = c;
        return;
    }

    int mid = tr[u].l + tr[u].r >> 1;
    if(x <= mid)  modify(u << 1, x, c);
    else  modify(u << 1 | 1, x, c);

    tr[u].h = tr[u << 1].h * p[tr[u].r - mid] + tr[u << 1 | 1].h;
}

LL query(int u, int l, int r)
{
    if(tr[u].l >= l and tr[u].r <= r)  return tr[u].h;

    int mid = tr[u].l + tr[u].r >> 1;
    if(r <= mid)  return query(u << 1, l, r);
    else if(l > mid)  return query(u << 1 | 1, l, r);
    return query(u << 1, l, mid) * p[r - mid] + query(u << 1 | 1, l, r);
}

set<LL> S;
int main()
{
    p[0] = 1;
    for(int i = 1; i < N; i ++)  p[i] = p[i - 1] * P;

    int T;
    scanf("%d", &T);
    for(int t = 1; t <= T; t ++)
    {
```

```
        memset(tr, 0, sizeof tr);
        S.clear();

        printf("Case #%d:\n", t);

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

        while(n --)
        {
            scanf("%s", s);
            S.insert(haxi(s));
        }

        scanf("%s", s);
        n = strlen(s);
        build(1, 0, n - 1);

        int m;
        scanf("%d", &m);
        while(m --)
        {
            char op[2];
            scanf("%s", op);

            if(op[0] == 'Q')
            {
                int l, r;
                scanf("%d%d", &l, &r);
                if(S.count(query(1, l, r)))  puts("Yes");
                else  puts("No");
            }
            else
            {
                int x;
                char c[2];
                scanf("%d%s", &x, c);
                modify(1, x, c[0]);
            }
        }
    }
    return 0;
}
```

2.5三次方和

区间赋值，累加，累乘维护三次方和

```
struct Node{
    int l, r;
    int lazy, add, mul;
    int sum, sum2, sum3;
}tr[N << 2];

void pushup(int u)
{
    tr[u].sum = (tr[u << 1].sum + tr[u << 1 | 1].sum) % mod;
```

```
    tr[u].sum2 = (tr[u << 1].sum2 + tr[u << 1 | 1].sum2) % mod;
    tr[u].sum3 = (tr[u << 1].sum3 + tr[u << 1 | 1].sum3) % mod;
}

void build(int u, int l, int r)
{
    tr[u] = {l, r, -1, 0, 1};
    if(l == r)
    {
        int c = w[rk[l]];
        tr[u].sum = c;
        tr[u].sum2 = (LL)c * c % mod;
        tr[u].sum3 = (LL)tr[u].sum2 * c % mod;
        return;
    }
    int mid = l + r >> 1;
    build(u << 1, l, mid),  build(u << 1 | 1, mid + 1, r);
    pushup(u);
}

void cover(int u, int w)        //  区间赋值
{
    int len = tr[u].r - tr[u].l + 1;
    tr[u].sum  = (LL)len * w % mod;
    tr[u].sum2 = (LL)len * w % mod * w % mod;
    tr[u].sum3 = (LL)len * w % mod * w % mod * w % mod;
    tr[u].lazy = w;
    tr[u].add = 0;
    tr[u].mul = 1;
}

void eval(int u, int add, int mul)      //  区间加/乘
{
    int len = tr[u].r - tr[u].l + 1;
    if(mul != 1)
    {
        tr[u].sum  = (LL)tr[u].sum  * mul % mod;
        tr[u].sum2 = (LL)tr[u].sum2 * mul % mod * mul % mod;
        tr[u].sum3 = (LL)tr[u].sum3 * mul % mod * mul % mod * mul % mod;
    }
    if(add)
    {
        tr[u].sum3 = (tr[u].sum3 + 3ll * tr[u].sum2 * add + 3ll * tr[u].sum *
add % mod * add + (LL)add * add % mod * add % mod * len) % mod;
        tr[u].sum2 = (tr[u].sum2 + 2ll * tr[u].sum * add + (LL)add * add % mod *
len) % mod;
        tr[u].sum  = (tr[u].sum + (LL)add * len) % mod;
    }
    tr[u].mul = (LL)tr[u].mul * mul % mod;
    tr[u].add = ((LL)tr[u].add * mul + add) % mod;
}

void pushdown(int u)
{
    if(tr[u].lazy != -1)
    {
        cover(u << 1, tr[u].lazy);
        cover(u << 1 | 1, tr[u].lazy);
```

```
        tr[u].lazy = -1;
    }
    eval(u << 1, tr[u].add, tr[u].mul);
    eval(u << 1 | 1, tr[u].add, tr[u].mul);
    tr[u].add = 0;
    tr[u].mul = 1;
}

void modify(int u, int l, int r, int op, int w)
{
    if(tr[u].l >= l and tr[u].r <= r)
    {
        if(op == 1)  cover(u, w);
        else
        {
            int add = 0, mul = 1;
            if(op == 2)  add = w;
            else  mul = w;
            eval(u, add, mul);
        }
        return;
    }

    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    if(l <= mid)  modify(u << 1, l, r, op, w);
    if(r > mid)  modify(u << 1 | 1, l, r, op, w);
    pushup(u);
}

int query(int u, int l, int r)
{
    if(tr[u].l >= l and tr[u].r <= r)  return tr[u].sum3;

    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1, res = 0;
    if(l <= mid)  res = query(u << 1, l, r);
    if(r > mid)  res = (res + query(u << 1 | 1, l, r)) % mod;
    return res;
}
```

2.6线段树合并

给出一个 n 个点 m 条边的有点权有边权的无向图，有 q 个询问，
每次询问求从点 v 出发只经过边权小于等于 x 的边，能抵达的第 k 大的点的点权

$n \leq 10^5$
$m, q \leq 5 \times 10^5$
点权，边权 $\leq 10^9$

首先把点权离散化，在每个点上建一棵线段树，维护点权区间的 $size$

然后把边按照边权排序，把询问按照边权 x 排序

由于询问是递增的，我们就可以把边权小于等于 x 的每一条边所连接的点合并
并查集合并连通块，同时合并线段树

对于每个询问我们在合并之后的线段树内二分即可找到第 k 大的点

时间复杂度 $O((n+m)logn)$

```
#include<cstdio>
#include<cstring>
#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;

const int N = 100010, M = 500010;

int n, m, k;
int h[N];

vector<int> hs;
int find(int x)
{
    return lower_bound(hs.begin(), hs.end(), x) - hs.begin();
}

struct Edge{
    int a, b, w;
    bool used;
    bool operator< (const struct Edge &W) const {
        return w < W.w;
    }
}e[M];

int p[N];
int fand(int x)
{
    if(x != p[x])  p[x] = fand(p[x]);
    return p[x];
}

struct Query{
    int root, x, k, num;
    bool operator< (const struct Query &W) const {
        return x < W.x;
    }
}q[M];

struct Node{
    int l, r;
    int sz;
}tr[N * 18];

int root[N], tot;

void insert(int &u, int l, int r, int x)
{
    if(!u)  u = ++ tot;
    tr[u].sz ++;
    if(l == r)  return;
    int mid = l + r >> 1;
    if(x <= mid)  insert(tr[u].l, l, mid, x);
```

```
    else  insert(tr[u].r, mid + 1, r, x);
}

int merge(int x, int y, int l, int r)
{
    if(!x or !y)  return x + y;

    int u = x;
    tr[u].sz = tr[x].sz + tr[y].sz;
    if(l == r)  return u;
    int mid = l + r >> 1;
    if(tr[x].l or tr[y].l)  tr[u].l = merge(tr[x].l, tr[y].l, l, mid);
    if(tr[x].r or tr[y].r)  tr[u].r = merge(tr[x].r, tr[y].r, mid + 1, r);
    return u;
}

int query(int u, int l, int r, int k)
{
    if(tr[u].sz < k)  return -1;
    if(l == r)  return hs[l];

    int mid = l + r >> 1;
    if(tr[tr[u].r].sz >= k)  return query(tr[u].r, mid + 1, r, k);
    return query(tr[u].l, l, mid, k - tr[tr[u].r].sz);
}

int res[M];

int main()
{
    scanf("%d%d%d", &n, &m, &k);
    for(int i = 1; i <= n; i ++)  scanf("%d", &h[i]),  hs.push_back(h[i]), p[i]
= i;
    sort(hs.begin(), hs.end());
    hs.erase(unique(hs.begin(), hs.end()), hs.end());

    for(int i = 0; i < m; i ++)  scanf("%d%d%d", &e[i].a, &e[i].b, &e[i].w);
    sort(e, e + m);

    for(int i = 0; i < k; i ++)
    {
        scanf("%d%d%d", &q[i].root, &q[i].x, &q[i].k);
        q[i].num = i;
    }
    sort(q, q + k);

    for(int i = 1; i <= n; i ++)  insert(root[i], 0, hs.size() - 1, find(h[i]));

    for(int i = 0, j = 0; i < k; i ++)
    {
        while(j < m and e[j].w <= q[i].x)
        {
            int a = e[j].a, b = e[j].b;
            int pa = fand(a), pb = fand(b);
            if(pa != pb)
            {
                root[pb] = merge(root[pa], root[pb], 0, hs.size() - 1);
                p[pa] = pb;
```

```
            }
            j ++;
        }
        res[q[i].num] = query(root[fand(q[i].root)], 0, hs.size() - 1, q[i].k);
    }

    for(int i = 0; i < k; i ++)  printf("%d\n", res[i]);
    return 0;
}
```

3.树链剖分

3.1轻重链剖分

树上路径赋值，路径最大子段和

时间复杂度 $O((n+m)log^2n)$

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 100010, M = N << 1, INF = 0x3f3f3f3f;

int h[N], e[M], ne[M], idx;
void add(int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

int a[N];

int sz[N], dep[N], son[N], f[N];
void dfs(int u, int fa)
{
    sz[u] = 1,  dep[u] = dep[fa] + 1,  f[u] = fa;
    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if(j == fa)  continue;
        dfs(j, u);
        sz[u] += sz[j];
        if(sz[j] > sz[son[u]])  son[u] = j;
    }
}

int id[N], rk[N], top[N], tot;
void dfs2(int u, int t)
{
    id[u] = ++ tot;  rk[tot] = u,  top[u] = t;
    if(son[u])  dfs2(son[u], t);
    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
```

```
        if(j == f[u] or j == son[u])  continue;
        dfs2(j, j);
    }
}

struct Node{
    int l, r;
    int d, ld, rd, sum, lazy;
}tr[N << 2];

void pushup(Node &u, Node left, Node right)
{
    u.sum = left.sum + right.sum;
    u.ld = max(left.ld, left.sum + right.ld);
    u.rd = max(right.rd, right.sum + left.rd);
    u.d = max(max(left.d, right.d), left.rd + right.ld);
}

void pushup(int u)
{
    pushup(tr[u], tr[u << 1], tr[u << 1 | 1]);
}

void build(int u, int l, int r)
{
    tr[u] = {l, r};
    tr[u].lazy = INF;
    if(l == r)
    {
        tr[u].sum = a[rk[l]];
        tr[u].d = tr[u].ld = tr[u].rd = max(0, a[rk[l]]);
        return;
    }
    int mid = l + r >> 1;
    build(u << 1, l, mid),  build(u << 1 | 1, mid + 1, r);
    pushup(u);
}

void pushdown(int u)
{
    if(tr[u].lazy != INF)
    {
        Node &left = tr[u << 1], &right = tr[u << 1 | 1];
        left.sum = tr[u].lazy * (left.r - left.l + 1);
        left.d = left.ld = left.rd = max(0, left.sum);
        right.sum = tr[u].lazy * (right.r - right.l + 1);
        right.d = right.ld = right.rd = max(0, right.sum);
        left.lazy = right.lazy = tr[u].lazy;
        tr[u].lazy = INF;
    }
}

void modify(int u, int l, int r, int c)
{
    if(tr[u].l >= l and tr[u].r <= r)
    {
        tr[u].sum = c * (tr[u].r - tr[u].l + 1);
        tr[u].d = tr[u].ld = tr[u].rd = max(0, tr[u].sum);
```

```
        tr[u].lazy = c;
        return;
    }

    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    if(l <= mid)  modify(u << 1, l, r, c);
    if(r > mid)  modify(u << 1 | 1, l, r, c);
    pushup(u);
}

void update(int a, int b, int c)
{
    while(top[a] != top[b])
    {
        if(dep[top[a]] < dep[top[b]])  swap(a, b);
        modify(1, id[top[a]], id[a], c);
        a = f[top[a]];
    }
    if(dep[a] > dep[b])  swap(a, b);
    modify(1, id[a], id[b], c);
}

Node query(int u, int l, int r)
{
    if(tr[u].l >= l and tr[u].r <= r)  return tr[u];

    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    if(r <= mid)  return query(u << 1, l, r);
    if(l > mid)  return query(u << 1 | 1, l, r);

    Node left = query(u << 1, l, r),  right = query(u << 1 | 1, l, r),  res;
    pushup(res, left, right);
    pushup(u);
    return res;
}

int ask(int a, int b)
{
    Node res = {0, 0, 0, 0, 0, 0, 0};
    Node left = res, right = res;
    while(top[a] != top[b])
    {
        if(dep[top[a]] > dep[top[b]])
        {
            Node q = query(1, id[top[a]], id[a]);
            pushup(left, q, left);
            a = f[top[a]];
        }
        else
        {
            Node q = query(1, id[top[b]], id[b]);
            pushup(right, q, right);
            b = f[top[b]];
        }
    }
    if(dep[a] > dep[b])  swap(a, b),  swap(left, right);
```

```
    swap(left.ld, left.rd);

    Node q = query(1, id[a], id[b]);
    pushup(left, left, q);
    pushup(res, left, right);
    return res.d;
}

int main()
{
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n; i ++)  scanf("%d", &a[i]);

    memset(h, -1, sizeof h);
    for(int i = 1; i < n; i ++)
    {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b),  add(b, a);
    }

    dfs(1, 0);
    dfs2(1, 1);
    build(1, 1, n);

    //for(int i = 1; i <= n; i ++)  printf("query(%d) = %d\n", i, query(1, i, i).d);
    //for(int i = 1; i <= n; i ++)  printf("id[%d] = %d   top[%d] = %d\n", i, id[i], i, top[i]);

    int m;
    scanf("%d", &m);
    while(m --)
    {
        int op, a, b, c;
        scanf("%d%d%d", &op, &a, &b);
        if(op == 1)
        {
            printf("%d\n", ask(a, b));
        }
        else
        {
            scanf("%d", &c);
            update(a, b, c);
        }
    }
    return 0;
}
```

一个调试用的树

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

4.平衡树

4.1无旋treap

```cpp
#include<cstdio>
#include<cstdlib>

using namespace std;

const int N = 100010;

struct Node{
    int l, r;
    int val, key;
    int sz;
}tr[N];

int root, idx;

int get_node(int val)
{
    int u = ++ idx;
    tr[u].val = val;
    tr[u].key = rand();
    tr[u].sz = 1;
    return u;
}

void update(int u)
{
    tr[u].sz = tr[tr[u].l].sz + tr[tr[u].r].sz + 1;
}

void split(int u, int val, int &x, int &y)
{
    if(!u)  x = y = 0;
    else
    {
        if(tr[u].val <= val)
        {
            x = u;
            split(tr[u].r, val, tr[u].r, y);
        }
        else
        {
```

```
            y = u;
            split(tr[u].l, val, x, tr[u].l);
        }
        update(u);
    }
}

int merge(int x, int y)
{
    if(!x or !y)  return x + y;
    if(tr[x].key < tr[y].key)
    {
        tr[x].r = merge(tr[x].r, y);
        update(x);
        return x;
    }
    else
    {
        tr[y].l = merge(x, tr[y].l);
        update(y);
        return y;
    }
}

int x, y, z;
void insert(int val)
{
    split(root, val, x, y);
    root = merge(merge(x, get_node(val)), y);
}

void dele(int val)
{
    split(root, val, x, y);
    split(x, val - 1, x, z);
    z = merge(tr[z].l, tr[z].r);
    root = merge(merge(x, z), y);
}

int get_rank(int val)
{
    split(root, val - 1, x, y);
    int res = tr[x].sz + 1;
    root = merge(x, y);
    return res;
}

int get_val(int rank)
{
    int u = root;
    while(u)
    {
        if(tr[tr[u].l].sz + 1 == rank)  break;
        else if(tr[tr[u].l].sz >= rank)  u = tr[u].l;
        else
        {
            rank -= tr[tr[u].l].sz + 1;
            u = tr[u].r;
```

```
        }
    }
    return tr[u].val;
}

int get_prev(int val)
{
    split(root, val - 1, x, y);
    int u = x;
    while(tr[u].r)  u = tr[u].r;
    root = merge(x, y);
    return tr[u].val;
}

int get_next(int val)
{
    split(root, val, x, y);
    int u = y;
    while(tr[u].l)  u = tr[u].l;
    root = merge(x, y);
    return tr[u].val;
}

int main()
{
    int m;
    scanf("%d", &m);
    while(m --)
    {
        int op, x;
        scanf("%d%d", &op, &x);
        if(op == 1)  insert(x);
        else if(op == 2)  dele(x);
        else if(op == 3)  printf("%d\n", get_rank(x));
        else if(op == 4)  printf("%d\n", get_val(x));
        else if(op == 5)  printf("%d\n", get_prev(x));
        else  printf("%d\n", get_next(x));
    }
    return 0;
}
```

4.2链表合并

题目描述

初始状态有 n 个糖，每个糖单独一堆，进行 m 次操作

- C a b 合并 a 所在的堆和 b 所在的堆
- D a 吃掉 a 所在的堆的所有糖
- Q k 询问糖的数量第 k 大的堆有多少块糖
  $n \leq 5 \times 10^5$
  $m \leq 5 \times 10^4$

输入样例

```
20 10
C 1 2
C 3 4
Q 2
Q 7
C 1 5
C 2 5
Q 1
D 5
Q 2
Q 1
```

输出样例

```
2
1
3
1
2
```

并查集 链表 平衡树

平衡树存每堆糖的数量，链表存每堆糖都有哪些糖

首先初始化并查集，往平衡树里放 n 个 1，再整 n 个链表，第 i 个链表放个 i 进去

合并操作 合并并查集，合并链表（记得更新 $tail$），平衡树删除之前两堆的数量，insert一个两堆之和

删除操作 找到这个糖所在的集合，给链表内所有元素打上删除标记，在平衡树中删除这堆糖的数量

询问操作 平衡树基本操作，根据 $rank$ 查找 val

时间复杂度 $O((n+m)logn)$

C++ 代码

```cpp
#include<cstdio>
#include<cstdlib>

const int N = 550010;

struct Node{
    int l, r;
    int val, key, sz;
}tr[N];

int root, tot;

int get_node(int val)
{
    int u = ++ tot;
    tr[u].val = val;
    tr[u].key = rand();
    tr[u].sz = 1;
    return u;
}
```

```
void pushup(int u)
{
    tr[u].sz = tr[tr[u].l].sz + tr[tr[u].r].sz + 1;
}

void split(int u, int val, int &x, int &y)
{
    if(!u)  x = y = 0;
    else
    {
        if(tr[u].val <= val)
        {
            x = u;
            split(tr[u].r, val, tr[u].r, y);
        }
        else
        {
            y = u;
            split(tr[u].l, val, x, tr[u].l);
        }
        pushup(u);
    }
}

int merge(int x, int y)
{
    if(!x or !y)  return x + y;
    if(tr[x].key > tr[y].key)
    {
        tr[x].r = merge(tr[x].r, y);
        pushup(x);
        return x;
    }
    else
    {
        tr[y].l = merge(x, tr[y].l);
        pushup(y);
        return y;
    }
}

int x, y, z;
void insert(int val)
{
    split(root, val, x, y);
    root = merge(merge(x, get_node(val)), y);
}

void dele(int val)
{
    split(root, val, x, y);
    split(x, val - 1, x, z);
    z = merge(tr[z].l, tr[z].r);
    root = merge(merge(x, z), y);
}

int get_val(int k)      //  第k大
{
```

```
    if(tr[root].sz < k)  return 0;

    int u = root;
    while(u)
    {
        if(tr[tr[u].r].sz + 1 == k)  break;
        if(tr[tr[u].r].sz >= k)  u = tr[u].r;
        else
        {
            k -= tr[tr[u].r].sz + 1;
            u = tr[u].l;
        }
    }
    return tr[u].val;
}

int n, m;
int p[N], sz[N];
int find(int x)
{
    if(p[x] != x)  return p[x] = find(p[x]);
    return p[x];
}

bool del[N];
int h[N], tail[N], e[N], ne[N], idx;

int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i ++)
    {
        insert(1);
        p[i] = i,  sz[i] = 1;

        e[idx] = i;
        h[i] = tail[i] = idx;
        ne[idx ++] = -1;
    }

    while(m --)
    {
        char op[2];
        int a, b, k;
        scanf("%s", op);
        if(op[0] == 'C')
        {
            scanf("%d%d", &a, &b);
            int pa = find(a), pb = find(b);
            if(del[a] or del[b] or pa == pb)  continue;

            dele(sz[pa]);
            dele(sz[pb]);

            sz[pb] += sz[pa];
            p[pa] = pb;

            ne[tail[pb]] = h[pa];
```

```
            tail[pb] = tail[pa];

            insert(sz[pb]);
        }
        else if(op[0] == 'D')
        {
            scanf("%d", &a);
            if(del[a])  continue;
            int pa = find(a);
            for(int i = h[pa]; ~i; i = ne[i])  del[e[i]] = true;
            dele(sz[pa]);
        }
        else
        {
            scanf("%d", &k);
            printf("%d\n", get_val(k));
        }
    }
    return 0;
}
```

4.3维护序列

插入删除，区间赋值，区间翻转，区间求和，整体求最大子段和

```
#include<cstdio>
#include<cstring>
#include<cstdlib>
#include<algorithm>

using namespace std;

typedef long long LL;

const int N = 4e6 + 10, INF = 0x3f3f3f3f;

struct Node{
    int l, r;
    int val, key, sz;
    LL sum, ld, rd, d;
    int lazy;
    bool re;
}tr[N];

int root, idx;
int get_node(int val)
{
    int u = ++ idx;
    tr[u].val = tr[u].sum = tr[u].d = val;
    tr[u].ld = tr[u].rd = max(0, val);
    tr[u].key = rand();
    tr[u].sz = 1;
    tr[u].lazy = INF;
    return u;
}
```

```
void pushup(int u)
{
    tr[u].sz = tr[tr[u].l].sz + tr[tr[u].r].sz + 1;
    tr[u].sum = tr[tr[u].l].sum + tr[tr[u].r].sum + tr[u].val;
    tr[u].d = max(tr[tr[u].l].d, tr[tr[u].r].d);
    tr[u].d = max(tr[u].d, tr[tr[u].l].rd + tr[u].val + tr[tr[u].r].ld);
    tr[u].ld = max(tr[tr[u].l].ld, tr[tr[u].l].sum + tr[u].val +
tr[tr[u].r].ld);
    tr[u].rd = max(tr[tr[u].r].rd, tr[tr[u].r].sum + tr[u].val +
tr[tr[u].l].rd);
}

//  区间赋值
void cover(int u, int c)
{
    if(!u)  return;
    tr[u].val = tr[u].lazy = c;
    tr[u].sum = (LL)tr[u].sz * c;
    tr[u].d = max((LL)c, tr[u].sum);
    tr[u].ld = tr[u].rd = max(0ll, tr[u].sum);
    tr[u].re = 0;
}

//  区间翻转
void rever(int u)
{
    if(!u)  return;
    tr[u].re ^= 1;
    swap(tr[u].l, tr[u].r);
    swap(tr[u].ld, tr[u].rd);
}

void pushdown(int u)
{
    if(tr[u].lazy != INF)
    {
        cover(tr[u].l, tr[u].lazy);
        cover(tr[u].r, tr[u].lazy);
        tr[u].lazy = INF;
    }
    if(tr[u].re)
    {
        rever(tr[u].l);
        rever(tr[u].r);
        tr[u].re = 0;
    }
}

void split(int u, int sz, int &x, int &y)
{
    if(!u)  x = y = 0;
    else
    {
        pushdown(u);
        if(tr[tr[u].l].sz < sz)
        {
            x = u;
            split(tr[u].r, sz - tr[tr[u].l].sz - 1, tr[u].r, y);
```

```
        }
        else
        {
            y = u;
            split(tr[u].l, sz, x, tr[u].l);
        }
        pushup(u);
    }
}

int merge(int x, int y)
{
    if(!x or !y)  return x | y;
    if(tr[x].key > tr[y].key)
    {
        pushdown(x);
        tr[x].r = merge(tr[x].r, y);
        pushup(x);
        return x;
    }
    else
    {
        pushdown(y);
        tr[y].l = merge(x, tr[y].l);
        pushup(y);
        return y;
    }
}

int n, m, x, y, z;
int w[N];
int main()
{
    tr[0].d = -1e18;

    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i ++)
    {
        scanf("%d", &x);
        root = merge(root, get_node(x));
    }

    while(m --)
    {
        int p, cnt, c;
        char op[20];
        scanf("%s", op);
        if(!strcmp(op, "INSERT"))
        {
            scanf("%d%d", &p, &cnt);
            for(int i = 0; i < cnt; i ++)  scanf("%d", &w[i]);
            if(!p)
            {
                for(int i = cnt - 1; i >= 0; i --)
                    root = merge(get_node(w[i]), root);
            }
            else
            {
```

```
                split(root, p, x, y);
                for(int i = 0; i < cnt; i ++)  x = merge(x, get_node(w[i]));
                root = merge(x, y);
            }
        }
        else if(!strcmp(op, "DELETE"))
        {
            scanf("%d%d", &p, &cnt);
            split(root, p - 1, x, y);
            split(y, cnt, y, z);
            root = merge(x, z);
        }
        else if(!strcmp(op, "MAKE-SAME"))
        {
            scanf("%d%d%d", &p, &cnt, &c);
            split(root, p - 1, x, y);
            split(y, cnt, y, z);
            cover(y, c);
            root = merge(merge(x, y), z);
        }
        else if(!strcmp(op, "REVERSE"))
        {
            scanf("%d%d", &p, &cnt);
            split(root, p - 1, x, y);
            split(y, cnt, y, z);
            rever(y);
            root = merge(merge(x, y), z);
        }
        else if(!strcmp(op, "GET-SUM"))
        {
            scanf("%d%d", &p, &cnt);
            split(root, p - 1, x, y);
            split(y, cnt, y, z);
            printf("%lld\n", tr[y].sum);
            root = merge(merge(x, y), z);
        }
        else
            printf("%lld\n", tr[root].d);
    }
    return 0;
}
```

4.4静态去重的区间最大子段和

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<map>

using namespace std;

typedef long long LL;

const int N = 100010;

int a[N], n;
```

```
struct Query{
    int l, r, num;
    bool operator< (const struct Query &Q) const {
        return r < Q.r;
    }
}q[N];

struct Node{
    int l, r;
    LL sum, add;           //  j \in [l, r]最大的sum[j, i]
    LL max, maxadd;        //  历史最大的sum，历史最大的add
}tr[N << 2];

void build(int u, int l, int r)
{
    tr[u] = {l, r};
    if(l == r)  return;
    int mid = l + r >> 1;
    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
}

void pushup(int u)
{
    tr[u].sum = max(tr[u << 1].sum, tr[u << 1 | 1].sum);
    tr[u].max = max(tr[u << 1].max, tr[u << 1 | 1].max);
}

void pushdown(int u)
{
    if(tr[u].add or tr[u].maxadd)
    {
        Node &left = tr[u << 1], &right = tr[u << 1 | 1];

        left.max = max(left.max, left.sum + tr[u].maxadd);
        right.max = max(right.max, right.sum + tr[u].maxadd);

        left.maxadd = max(left.maxadd, left.add + tr[u].maxadd);
        right.maxadd = max(right.maxadd, right.add + tr[u].maxadd);

        left.sum += tr[u].add,  right.sum += tr[u].add;
        left.add += tr[u].add,  right.add += tr[u].add;

        tr[u].add = tr[u].maxadd = 0;
    }
}

void modify(int u, int l, int r, int add)
{
    if(tr[u].l >= l and tr[u].r <= r)
    {
        tr[u].sum += add;
        tr[u].add += add;
        tr[u].max = max(tr[u].max, tr[u].sum);
        tr[u].maxadd = max(tr[u].maxadd, tr[u].add);
        //printf("[%d, %d]  sum = %d  max = %d\n", tr[u].l, tr[u].r, tr[u].sum,
tr[u].max);
        return;
    }
```

```
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    if(l <= mid)  modify(u << 1, l, r, add);
    if(r > mid)  modify(u << 1 | 1, l, r, add);
    pushup(u);
    //printf("[%d, %d]  sum = %d  max = %d\n", tr[u].l, tr[u].r, tr[u].sum,
tr[u].max);
}

LL query(int u, int l, int r)
{
    if(tr[u].l >= l and tr[u].r <= r)  return tr[u].max;

    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    LL res = 0;
    if(l <= mid)  res = query(u << 1, l, r);
    if(r > mid)  res = max(res, query(u << 1 | 1, l, r));
    pushup(u);
    return res;
}

map<int, int> p;
LL res[N];

int main()
{
    scanf("%d", &n);
    for(int i = 1; i <= n; i ++)  scanf("%d", &a[i]);

    int m;
    scanf("%d", &m);
    for(int i = 0; i < m; i ++)  scanf("%d%d", &q[i].l, &q[i].r),  q[i].num = i;

    sort(q, q + m);

    build(1, 1, n);

    for(int i = 0, j = 0; i < m; i ++)
    {
        while(j < q[i].r)
        {
            ++ j;
            //printf("i = %d    [%d, %d]\n", i, p[a[j]] + 1, j);
            if(p[a[j]] + 1 <= j)  modify(1, p[a[j]] + 1, j, a[j]);
            p[a[j]] = j;
        }

        res[q[i].num] = query(1, q[i].l, q[i].r);
    }

    for(int i = 0; i < m; i ++)  printf("%d\n", res[i]);
    return 0;
}
```

5.可持久化数据结构

5.1主席树

区间第 k 小

```
#include<cstdio>
#include<algorithm>
#include<vector>

using namespace std;

const int N = 100010;

struct Node{
    int l, r;
    int sz;
}tr[N * 4 + N * 17];

int root[N], tot;

int n, m;
int a[N];

vector<int> v;
int find(int x)
{
    return lower_bound(v.begin(), v.end(), x) - v.begin();
}

void build(int &u, int l, int r)
{
    u = ++ tot;
    if(l == r)  return;
    int mid = l + r >> 1;
    build(tr[u].l, l, mid),  build(tr[u].r, mid + 1, r);
}

void insert(int v, int &u, int l, int r, int x)
{
    u = ++ tot;
    tr[u] = tr[v];
    tr[u].sz ++;
    if(l == r)  return;
    int mid = l + r >> 1;
    if(x <= mid)  insert(tr[v].l, tr[u].l, l, mid, x);
    else  insert(tr[v].r, tr[u].r, mid + 1, r, x);
}

int query(int v, int u, int l, int r, int k)
{
    if(l == r)  return l;

    int mid = l + r >> 1;
    if(tr[tr[u].l].sz - tr[tr[v].l].sz >= k)  return query(tr[v].l, tr[u].l, l,
mid, k);
```

```
    return query(tr[v].r, tr[u].r, mid + 1, r, k - (tr[tr[u].l].sz -
tr[tr[v].l].sz));
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i ++)
    {
        scanf("%d", &a[i]);
        v.push_back(a[i]);
    }
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());

    build(root[0], 0, v.size() - 1);
    for(int i = 1; i <= n; i ++)
        insert(root[i - 1], root[i], 0, v.size() - 1, find(a[i]));

    while(m --)
    {
        int l, r, k;
        scanf("%d%d%d", &l, &r, &k);
        printf("%d\n", v[query(root[l - 1], root[r], 0, v.size() - 1, k)]);
    }
    return 0;
}
```

5.2可持久化01Trie

最大异或和

给定一个非负整数序列 a，初始长度为 N。

有 M 个操作，有以下两种操作类型：

1、”A x”：添加操作，表示在序列末尾添加一个数 x，序列的长度 N 增大1。
2、”Q l r x”：询问操作，你需要找到一个位置 p，满足l≤p≤rl≤p≤r，使得：a[p] xor a[p+1] xor … xor a[N] xor x 最大，输出这个最大值。

```
#include<cstdio>
#include<cstring>

const int N = 600010, M = N * 25;

int tr[M][2], id[M], tot;
int sum[N], root[N];

void insert(int v, int u, int num, int x)
{
    id[u] = num;
    for(int i = 23; i >= 0; i --)
    {
        int j = x >> i & 1;
        if(v)  tr[u][j ^ 1] = tr[v][j ^ 1];
        tr[u][j] = ++ tot;
        v = tr[v][j], u = tr[u][j];
```

```
        id[u] = num;
    }
}

int query(int l, int u, int x)
{
    int res = 0;
    for(int i = 23; i >= 0; i --)
    {
        int j = x >> i & 1;
        if(id[tr[u][j ^ 1]] >= l)  u = tr[u][j ^ 1], res |= 1 << i;
        else  u = tr[u][j];
    }
    return res;
}

int main()
{
    id[0] = -1;
    root[0] = ++ tot;
    insert(0, root[0], 0, 0);

    int n, m;
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i ++)
    {
        scanf("%d", &sum[i]);
        sum[i] ^= sum[i - 1];
        root[i] = ++ tot;
        insert(root[i - 1], root[i], i, sum[i]);
    }

    while(m --)
    {
        char op[2];
        int l, r, x;
        scanf("%s", op);
        if(op[0] == 'A')
        {
            ++ n;
            scanf("%d", &sum[n]);
            sum[n] ^= sum[n - 1];
            root[n] = ++ tot;
            insert(root[n - 1], root[n], n, sum[n]);
        }
        else
        {
            scanf("%d%d%d", &l, &r, &x);
            printf("%d\n", query(l - 1, root[r - 1], sum[n] ^ x));
        }
    }
    return 0;
}
```

6.ST表

时间复杂度 $O(nlogn)$ 预处理 $O(1)$ 询问区间最值

```
int a[N], n;
int f[N][19];

void init()
{
    for(int j = 0; j < 18; j ++)
        for(int i = 1; i + (1 << j) - 1 <= n; i ++)
            if(!j)  f[i][j] = a[i];
            else  f[i][j] = max(f[i][j - 1], f[i + (1 << j - 1)][j - 1]);
}

int query(int l, int r)
{
    int k = log2(r - l + 1);
    return max(f[l][k], f[r - (1 << k) + 1][k]);
}
```

7.分块

区间累加区间求和

```
typedef long long LL;
const int N = 100010, M = 400;
int n, m, w[N];
int id[N], l[M], r[M], block;
LL add[M], sum[M];
void build()
{
    block = sqrt(n);
    id[n] = n / block;  if(n % block)  id[n] ++;
    for(int i = 1; i <= id[n]; i ++)
        l[i] = (i - 1) * block + 1,  r[i] = i * block;
    r[id[n]] = n;

    for(int i = 1; i <= n; i ++)
    {
        id[i] = (i - 1) / block + 1;
        sum[id[i]] += w[i];
    }
}

void modify(int x, int y, int c)
{
    if(id[x] == id[y])
    {
        for(int i = x; i <= y; i ++)  w[i] += c, sum[id[i]] += c;
    }
    else
    {
        for(int i = x; i <= r[id[x]]; i ++)  w[i] += c, sum[id[i]] += c;
        for(int i = l[id[y]]; i <= y; i ++)  w[i] += c, sum[id[i]] += c;
        for(int i = id[x] + 1; i < id[y]; i ++)  add[i] += c, sum[i] += c *
block;
    }
}
```

```
LL query(int x, int y)
{
    LL res = 0;
    if(id[x] == id[y])
    {
        for(int i = x; i <= y; i ++)  res += w[i] + add[id[i]];
    }
    else
    {
        for(int i = x; i <= r[id[x]]; i ++)  res += w[i] + add[id[i]];
        for(int i = l[id[y]]; i <= y; i ++)  res += w[i] + add[id[i]];
        for(int i = id[x] + 1; i < id[y]; i ++)  res += sum[i];
    }
    return res;
}
```

8.莫队

8.1普通莫队

询问区间数字出现次数的平方和

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cmath>

using namespace std;

typedef long long LL;

const int N = 50010;

int n, m, k;
int a[N], cnt[N];

int pos[N];
struct Query{
    int l, r, num;
    bool operator< (const struct Query &W) const {
        if(pos[l] != pos[W.l])  return pos[l] < pos[W.l];
        return r < W.r;
    }
}q[N];

LL ans[N], res;
int l, r;

void add(int x)
{
    res -= (LL)cnt[a[x]] * cnt[a[x]];
    cnt[a[x]] ++;
    res += (LL)cnt[a[x]] * cnt[a[x]];
}

void sub(int x)
```

```
{
    res -= (LL)cnt[a[x]] * cnt[a[x]];
    cnt[a[x]] --;
    res += (LL)cnt[a[x]] * cnt[a[x]];
}

int main()
{
    scanf("%d%d%d", &n, &m, &k);

    int sz = sqrt(n);
    for(int i = 1; i <= n; i ++)
    {
        scanf("%d", &a[i]);
        pos[i] = i / sz;
    }

    for(int i = 0; i < m; i ++)
    {
        scanf("%d%d", &q[i].l, &q[i].r);
        q[i].num = i;
    }
    sort(q, q + m);

    l = 1, r = 0;
    for(int i = 0; i < m; i ++)
    {
        while(l > q[i].l)  add(-- l);
        while(r < q[i].r)  add(++ r);
        while(l < q[i].l)  sub(l ++);
        while(r > q[i].r)  sub(r --);
        ans[q[i].num] = res;
    }

    for(int i = 0; i < m; i ++)  printf("%lld\n", ans[i]);
    return 0;
}
```

8.2带修莫队

时间复杂度 $O(\sqrt[3]{nt})$ （t是修改次数，n是数组长度、询问次数）

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cmath>

using namespace std;

const int N = 10010, M = 1e6 + 10;

int n, m, sz, idx, t;
int col[N];

int get(int x)
{
    return x / sz;
```

```
}

struct Query{
    int id, l, r, t;
    bool operator< (const struct Query &Q) const {
        if(get(l) != get(Q.l))  return get(l) < get(Q.l);
        if(get(r) != get(Q.r))  return get(r) < get(Q.r);
        return t < Q.t;
    }
}q[N];

struct Modify{
    int p, c;
}mo[N];

int ans[N], res;
int cnt[M];

void add(int c)
{
    if(cnt[c] ++ == 0)  res ++;
}

void sub(int c)
{
    if(cnt[c] -- == 1)  res --;
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i ++)  scanf("%d", &col[i]);

    for(int i = 0; i < m; i ++)
    {
        char op[2];
        int x, y;
        scanf("%s%d%d", op, &x, &y);
        if(op[0] == 'Q')  idx ++, q[idx] = {idx, x, y, t};
        else  mo[ ++ t] = {x, y};
    }

    sz = cbrt((double)n * t) + 1;

    sort(q + 1, q + idx + 1);

    for(int i = 1, l = 1, r = 0, t = 0; i <= idx; i ++)
    {
        while(l > q[i].l)  add(col[ -- l]);
        while(r < q[i].r)  add(col[ ++ r]);
        while(l < q[i].l)  sub(col[l ++ ]);
        while(r > q[i].r)  sub(col[r -- ]);
        while(t < q[i].t)
        {
            t ++;
            int p = mo[t].p;
            if(p >= q[i].l and p <= q[i].r)
            {
```

```
                sub(col[p]);
                add(mo[t].c);
            }
            swap(col[p], mo[t].c);
        }
        while(t > q[i].t)
        {
            int p = mo[t].p;
            if(p >= q[i].l and p <= q[i].r)
            {
                sub(col[p]);
                add(mo[t].c);
            }
            swap(col[p], mo[t].c);
            t --;
        }
        ans[q[i].id] = res;
    }
    for(int i = 1; i <= idx; i ++)  printf("%d\n", ans[i]);
    return 0;
}
```

8.3回滚莫队

给定一个序列，每次询问一个区间中 $cnt[c] * c$ 的最大值

时间复杂度 $O(n\sqrt{n})$

```
#pragma GCC optimize(2)
#include<cstdio>
#include<algorithm>
#include<map>
#include<cmath>

using namespace std;

typedef long long LL;

const int N = 100010;

int n, m, sz;
int col[N];
vector<int> v;
int find(int x)
{
    return lower_bound(v.begin(), v.end(), x) - v.begin();
}

int get(int x)
{
    return x / sz;
}

struct Query{
    int l, r, id;
    bool operator< (const struct Query &T) const {
        if(get(l) != get(T.l))  return get(l) <= get(T.l);
```

```
        return r < T.r;
    }
}q[N];

int cnt[N];
LL ans[N], res, last;

void add(int c)
{
    res = max(res, (LL)( ++ cnt[c]) * v[c]);
}

void sub(int c)
{
    cnt[c] --;
}

int main()
{
    scanf("%d%d", &n, &m);
    sz = sqrt(n);

    for(int i = 1; i <= n; i ++)  scanf("%d", &col[i]),  v.push_back(col[i]);

    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
    for(int i = 1; i <= n; i ++)  col[i] = find(col[i]);

    for(int i = 0; i < m; i ++)  scanf("%d%d", &q[i].l, &q[i].r), q[i].id = i;

    sort(q, q + m);

    for(int x = 0, y; x < m; x ++)
    {
        y = x;
        while(y + 1 < m and get(q[y + 1].l) == get(q[x].l))  y ++;

        res = last = 0;
        int right = get(q[x].l) * sz + sz;
        int l, r = right - 1;

        for(int i = x; i <= y; i ++)
        {
            res = last;
            while(r < q[i].r)  add(col[ ++ r]);
            last = res;

            l = min(right, q[i].r + 1);
            while(l > q[i].l)  add(col[ -- l]);

            ans[q[i].id] = res;
            while(l < right and l < q[i].r + 1)  sub(col[l ++ ]);
        }
        while(r >= right)  sub(col[r -- ]);
        x = y;
    }

    for(int i = 0; i < m; i ++)  printf("%lld\n", ans[i]);
```

```
    return 0;
}
```

8.4树上莫队

树上 n 个点每个点有个颜色，m 次询问每次问一条路径上有多少种不同的颜色

时间复杂度 $O(n\sqrt{n})$

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<vector>
#include<cmath>

using namespace std;

const int N = 100010;

int h[N], e[N], ne[N], idx;
void add(int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

int n, m;
int col[N];
vector<int> v;
int find(int x)
{
    return lower_bound(v.begin(), v.end(), x) - v.begin();
}

int fa[N][16], dep[N];
void bfs()
{
    int q[N], hh = 0, tt = -1;

    q[ ++ tt] = 1;
    dep[1] = 1;

    while(hh <= tt)
    {
        int u = q[hh ++];

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i];
            if(dep[j])  continue;

            dep[j] = dep[u] + 1;
            fa[j][0] = u;
            q[ ++ tt] = j;

            for(int k = 1; k < 16; k ++)
                fa[j][k] = fa[fa[j][k - 1]][k - 1];
        }
```

```
    }
}

int lca(int a, int b)
{
    if(dep[a] < dep[b])  swap(a, b);

    for(int i = 15; i >= 0; i --)
        if(dep[fa[a][i]] >= dep[b])
            a = fa[a][i];

    if(a == b)  return a;

    for(int i = 15; i >= 0; i --)
        if(fa[a][i] != fa[b][i])
        {
            a = fa[a][i];
            b = fa[b][i];
        }
    return fa[a][0];
}

int euler[N], tot;
int first[N], last[N];

void dfs(int u)
{
    euler[ ++ tot] = u;
    first[u] = tot;
    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if(j == fa[u][0])  continue;
        dfs(j);
    }
    euler[ ++ tot] = u;
    last[u] = tot;
}

int sz;
int get(int x)
{
    return x / sz;
}

struct Query{
    int l, r, id, p;
    bool operator< (const struct Query &T) const {
        if(get(l) != get(T.l))  return get(l) < get(T.l);
        return r < T.r;
    }
}q[N];

int cnt[N], ans[N], res;
int st[N];

void add(int u)
{
```

```
    st[u] ^= 1;
    if(!st[u])
    {
        if( -- cnt[col[u]] == 0)  res --;
    }
    else
    {
        if( ++ cnt[col[u]] == 1)  res ++;
    }
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i ++)  scanf("%d", &col[i]),  v.push_back(col[i]);

    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
    for(int i = 1; i <= n; i ++)  col[i] = find(col[i]);

    memset(h, -1, sizeof h);
    for(int i = 1; i < n; i ++)
    {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);
        add(b, a);
    }

    bfs();
    dfs(1);

    sz = sqrt(n * 2);
    for(int i = 0; i < m; i ++)
    {
        int a, b;
        scanf("%d%d", &a, &b);
        if(first[a] > first[b])  swap(a, b);

        int p = lca(a, b);
        if(p == a)  q[i] = {first[a], first[b], i, 0};
        else  q[i] = {last[a], first[b], i, p};
    }

    sort(q, q + m);

    for(int i = 0, l = 1, r = 0; i < m; i ++)
    {
        while(l > q[i].l)  add(euler[ -- l]);
        while(r < q[i].r)  add(euler[ ++ r]);
        while(l < q[i].l)  add(euler[l ++ ]);
        while(r > q[i].r)  add(euler[r -- ]);
        if(q[i].p)  add(q[i].p);
        ans[q[i].id] = res;
        if(q[i].p)  add(q[i].p);
    }

    for(int i = 0; i < m; i ++)  printf("%d\n", ans[i]);
```

```
    return 0;
}
```

9.树套树

9.1二逼平衡树

1.区间给val查询rank

2.区间给rank查询val

3.单点修改

4.区间查询前驱后继

```
#include<cstdio>
#include<cstdlib>
#include<algorithm>

using namespace std;

const int N = 50010, INF = 0x3f3f3f3f;

struct FHQ{
    int l, r;
    int val, key, sz;
}fhq[N * 40];

int tot;

int get_node(int val)
{
    int u = ++ tot;
    fhq[u].l = fhq[u].r = 0;
    fhq[u].val = val;
    fhq[u].key = rand();
    fhq[u].sz = 1;
    return u;
}

void pushup(int u)
{
    fhq[u].sz = fhq[fhq[u].l].sz + fhq[fhq[u].r].sz + 1;
}

void split(int u, int val, int &x, int &y)
{
    if(!u)  x = y = 0;
    else
    {
        if(fhq[u].val <= val)
        {
            x = u;
            split(fhq[u].r, val, fhq[u].r, y);
        }
        else
        {
```

```
            y = u;
            split(fhq[u].l, val, x, fhq[u].l);
        }
        pushup(u);
    }
}

int merge(int x, int y)
{
    if(!x or !y)  return x | y;

    if(fhq[x].key > fhq[y].key)
    {
        fhq[x].r = merge(fhq[x].r, y);
        pushup(x);
        return x;
    }
    else
    {
        fhq[y].l = merge(x, fhq[y].l);
        pushup(y);
        return y;
    }
}

int x, y, z;
void insert(int &root, int val)
{
    split(root, val, x, y);
    root = merge(merge(x, get_node(val)), y);
}

void dele(int &root, int val)
{
    split(root, val - 1, x, y);
    split(y, val, y, z);
    y = merge(fhq[y].l, fhq[y].r);
    root = merge(merge(x, y), z);
}

int get_rank(int &root, int val)
{
    split(root, val - 1, x, y);
    int res = fhq[x].sz;
    root = merge(x, y);
    return res;
}

int get_val(int &root, int k)
{
    int u = root;
    while(u)
    {
        if(fhq[fhq[u].l].sz + 1 == k)  break;
        if(fhq[fhq[u].l].sz >= k)  u = fhq[u].l;
        else
        {
            k -= fhq[fhq[u].l].sz + 1;
```

```
            u = fhq[u].r;
        }
    }
    return fhq[u].val;
}

int get_prev(int &root, int val)
{
    split(root, val - 1, x, y);
    int u = x;
    while(fhq[u].r)  u = fhq[u].r;
    int res = fhq[u].val;
    if(!x)  res = -INF;
    root = merge(x, y);
    return res;
}

int get_next(int &root, int val)
{
    split(root, val, x, y);
    int u = y;
    while(fhq[u].l)  u = fhq[u].l;
    int res = fhq[u].val;
    if(!y)  res = INF;
    root = merge(x, y);
    return res;
}

int n, m;
int w[N];

struct Node{
    int l, r;
    int root;
}tr[N << 2];

void build(int u, int l, int r)
{
    tr[u] = {l, r};
    for(int i = l; i <= r; i ++)  insert(tr[u].root, w[i]);
    if(l == r)  return;
    int mid = l + r >> 1;
    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
}

int query_rank(int u, int l, int r, int val)
{
    if(tr[u].l >= l and tr[u].r <= r)  return get_rank(tr[u].root, val);

    int mid = tr[u].l + tr[u].r >> 1, res = 0;
    if(l <= mid)  res = query_rank(u << 1, l, r, val);
    if(r > mid)  res += query_rank(u << 1 | 1, l, r, val);
    return res;
}

int query_val(int x, int y, int k)
{
    int l = 0, r = 1e8;
```

```
    while(l < r)
    {
        int mid = l + r + 1 >> 1;
        if(query_rank(1, x, y, mid) < k)  l = mid;
        else  r = mid -  1;
    }
    return l;
}

void modify(int u, int x, int val)
{
    dele(tr[u].root, w[x]);
    insert(tr[u].root, val);

    if(tr[u].l == tr[u].r)  return;
    int mid = tr[u].l + tr[u].r >> 1;
    if(x <= mid)  modify(u << 1, x, val);
    else  modify(u << 1 | 1, x, val);
}

int query_prev(int u, int l, int r, int val)
{
    if(tr[u].l >= l and tr[u].r <= r)  return get_prev(tr[u].root, val);

    int mid = tr[u].l + tr[u].r >> 1, res = -INF;
    if(l <= mid)  res = query_prev(u << 1, l, r, val);
    if(r > mid)  res = max(res, query_prev(u << 1 | 1, l, r, val));
    return res;
}

int query_next(int u, int l, int r, int val)
{
    if(tr[u].l >= l and tr[u].r <= r)  return get_next(tr[u].root, val);

    int mid = tr[u].l + tr[u].r >> 1, res = INF;
    if(l <= mid)  res = query_next(u << 1, l, r, val);
    if(r > mid)  res = min(res, query_next(u << 1 | 1, l, r, val));
    return res;
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i ++)  scanf("%d", &w[i]);

    build(1, 1, n);

    while(m --)
    {
        int op, l, r, x, k, val;
        scanf("%d", &op);
        if(op == 1)
        {
            scanf("%d%d%d", &l, &r, &val);
            printf("%d\n", query_rank(1, l, r, val) + 1);
        }
        else if(op == 2)
        {
```

```
            scanf("%d%d%d", &l, &r, &k);
            printf("%d\n", query_val(l, r, k));
        }
        else if(op == 3)
        {
            scanf("%d%d", &x, &val);
            modify(1, x, val);
            w[x] = val;
        }
        else if(op == 4)
        {
            scanf("%d%d%d", &l, &r, &val);
            printf("%d\n", query_prev(1, l, r, val));
        }
        else
        {
            scanf("%d%d%d", &l, &r, &val);
            printf("%d\n", query_next(1, l, r, val));
        }
    }
    return 0;
}
```

10.整体二分

题目支持离线做法

把修改变成删除插入，和询问全部视为操作

单点修改，区间查询第k小

时间复杂度 $O(nlog^2n)$

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 300010;

struct Node{
    int op;
    int x, y, k;
    int id;
}q[N], q1[N], q2[N];

int n, m;
int a[N];
int res[N];

int tr[N];
void add(int x, int c)
{
    for(int i = x; i <= n; i += i & -i)  tr[i] += c;
}
```

```
int sum(int x)
{
    int res = 0;
    for(int i = x; i; i -= i & -i)  res += tr[i];
    return res;
}

//  在[x, y]的值域里解决[l, r]这些操作
void solve(int x, int y, int l, int r)
{
    if(l > r)  return;
    if(x == y)
    {
        for(int i = l; i <= r; i ++)
            if(q[i].op == 2)  res[q[i].id] = x;
        return;
    }

    int mid = x + y >> 1;
    int t1 = -1, t2 = -1;
    for(int i = l; i <= r; i ++)
        if(q[i].op != 2)
        {
            if(q[i].y <= mid)  add(q[i].x, q[i].op),  q1[ ++ t1] = q[i];
            else  q2[ ++ t2] = q[i];
        }
        else
        {
            int t = sum(q[i].y) - sum(q[i].x - 1);
            if(q[i].k <= t)  q1[ ++ t1] = q[i];
            else
            {
                q[i].k -= t;
                q2[ ++ t2] = q[i];
            }
        }

    for(int i = 0; i <= t1; i ++)
        if(q1[i].op != 2)  add(q1[i].x, -q1[i].op);

    for(int i = 0; i <= t1; i ++)  q[l + i] = q1[i];
    for(int i = 0; i <= t2; i ++)  q[l + t1 + 1 + i] = q2[i];

    solve(x, mid, l, l + t1);
    solve(mid + 1, y, l + t1 + 1, r);
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i ++)
    {
        scanf("%d", &a[i]);
        q[i] = {1, i, a[i]};
    }

    int cnt = n, idx = 0;
```

```
    while(m --)
    {
        char op[2];
        int x, y, k;
        scanf("%s%d%d", op, &x, &y);
        if(op[0] == 'Q')
        {
            scanf("%d", &k);
            q[ ++ cnt] = (Node){2, x, y, k, ++ idx};
        }
        else
        {
            q[ ++ cnt] = (Node){-1, x, a[x]};
            q[ ++ cnt] = (Node){1, x, y};
            a[x] = y;
        }
    }

    solve(-1e9, 1e9, 1, cnt);

    for(int i = 1; i <= idx; i ++)
        printf("%d\n", res[i]);

    return 0;
}
```

11.CDQ分治

给定 n 个元素每个元素包含 a, b, c 三种属性

设 $f(i)$ 为满足 $a_j \leq a_i \wedge b_j \leq b_i \wedge c_j \leq c_i \wedge j! = i$ 的 j 的数量

对于 $d \in [0, n)$ 求 $f(i) = d$ 的 i 的数量

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 200010;

struct Data{
    int a, b, c;
    int cnt, res;
    bool operator< (const struct Data &T) const {
        if(a != T.a)  return a < T.a;
        if(b != T.b)  return b < T.b;
        return c < T.c;
    }
    bool operator== (const struct Data &T) const {
        return a == T.a and b == T.b and c == T.c;
    }
}q[N], tmp[N];

int n, m;
int tr[N];
```

```
void add(int x, int c)
{
    for(int i = x; i < N; i += i & -i)  tr[i] += c;
}

int sum(int x)
{
    int res = 0;
    for(int i = x; i; i -= i & -i)  res += tr[i];
    return res;
}

void merge_sort(int l, int r)
{
    if(l >= r)  return;

    int mid = l + r >> 1;
    merge_sort(l, mid),  merge_sort(mid + 1, r);

    int i = l, j = mid + 1, k = 0;
    while(i <= mid and j <= r)
        if(q[i].b <= q[j].b)  add(q[i].c, q[i].cnt),  tmp[k ++] = q[i ++];
        else q[j].res += sum(q[j].c),  tmp[k ++] = q[j ++];
    while(i <= mid)  add(q[i].c, q[i].cnt),  tmp[k ++] = q[i ++];
    while(j <= r)  q[j].res += sum(q[j].c),  tmp[k ++] = q[j ++];

    for(int i = l; i <= mid; i ++)  add(q[i].c, -q[i].cnt);
    for(int i = l, j = 0; i <= r; i ++, j ++)  q[i] = tmp[j];
}

int ans[N];

int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 0; i < n; i ++)
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        q[i] = {a, b, c, 1};
    }

    sort(q, q + n);

    int j = 0;
    for(int i = 1; i < n; i ++)
        if(q[i] == q[j])  q[j].cnt ++;
        else  q[ ++ j] = q[i];

    merge_sort(0, j);
    for(int i = 0; i <= j; i ++)
        ans[q[i].res + q[i].cnt - 1] += q[i].cnt;

    for(int i = 0; i < n; i ++)  printf("%d\n", ans[i]);
    return 0;
}
```

12.Dancing Links

一个 01 稀疏矩阵，选取其中的一些行使得每一列有且仅有一个 1

```
const int N = 5510;      //  节点数(1的数量)

int n, m;
int l[N], r[N], u[N], d[N], sz[N], row[N], col[N], idx;

void init()
{
    for(int i = 0; i <= m; i ++)
    {
        l[i] = i - 1, r[i] = i + 1;
        u[i] = d[i] = i;
    }
    l[0] = m, r[m] = 0;
    idx = m + 1;
}

void add(int &hh, int &tt, int x, int y)
{
    row[idx] = x, col[idx] = y, sz[y] ++;
    u[idx] = y, d[idx] = d[y], u[d[y]] = idx, d[y] = idx;
    l[idx] = hh, r[idx] = tt, r[hh] = l[tt] = idx;
    tt = idx ++;
}

//  删除第p列
void remove(int p)
{
    r[l[p]] = r[p], l[r[p]] = l[p];
    for(int i = d[p]; i != p; i = d[i])
        for(int j = r[i]; j != i; j = r[j])
        {
            sz[col[j]] --;
            d[u[j]] = d[j], u[d[j]] = u[j];
        }
}

//  恢复第p列
void resume(int p)
{
    for(int i = u[p]; i != p; i = u[i])
        for(int j = l[i]; j != i; j = l[j])
        {
            d[u[j]] = j, u[d[j]] = j;
            sz[col[j]] ++;
        }
    r[l[p]] = l[r[p]] = p;
}

vector<int> res;
bool dfs()
{
    if(!r[0])  return true;
```

```
    //  找到1个个数最少的一列
    int p = r[0];
    for(int i = r[0]; i; i = r[i])
        if(sz[i] < sz[p])  p = i;

    remove(p);
    for(int i = d[p]; i != p; i = d[i])
    {
        res.push_back(row[i]);
        for(int j = r[i]; j != i; j = r[j])  remove(col[j]);
        if(dfs())  return true;
        for(int j = l[i]; j != i; j = l[j])  resume(col[j]);
        res.pop_back();
    }
    resume(p);
    return false;
}

int main()
{
    scanf("%d%d", &n, &m);

    init();

    for(int i = 1; i <= n; i ++)
    {
        int hh = idx, tt = idx;
        for(int j = 1; j <= m; j ++)
        {
            int x;
            scanf("%d", &x);
            if(x)  add(hh, tt, i, j);
        }
    }

    if(dfs())  for(int i : res)  printf("%d ", i);
    else  puts("No Solution!");
    return 0;
}
```

13.点分治

求树中距离不超过 m 的点对数量

时间复杂度 $O(nlog^2n)$

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

typedef long long LL;

const int N = 10010, M = N << 1;

int h[N], e[M], w[M], ne[M], idx;
```

```
void add(int a, int b, int c)
{
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++;
}

int n, m;
int sz[N], ms[N], root;
bool dele[N];
void getrt(int u, int fa)
{
    sz[u] = 1, ms[u] = 0;
    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if(j == fa or dele[j])  continue;
        getrt(j, u);
        ms[u] = max(ms[u], sz[j]);
        sz[u] += sz[j];
    }
    ms[u] = max(ms[u], n - sz[u]);
    if(ms[u] < ms[root])  root = u;
}

int d[N], q[N], tt;
void getd(int u, int fa)
{
    if(d[u] <= m)  q[tt ++ ] = d[u];
    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if(j == fa or dele[j])  continue;
        d[j] = d[u] + w[i];
        getd(j, u);
    }
}

//  求数组里的i < j and a[i] + a[j] <= m的数量
LL get(int a[], int n)
{
    LL res = 0;

    sort(a, a + n);
    for(int i = 0, j = n - 1; i < j; i ++)
    {
        while(i < j and a[i] + a[j] > m)  j --;
        res += j - i;
    }
    return res;
}

LL res;
void solve(int u)
{
    tt = 0;
    q[tt ++ ] = 0;
    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
```

```
        if(dele[j])  continue;

        int last = tt;
        d[j] = w[i];
        getd(j, u);

        res -= get(q + last, tt - last);
    }
    res += get(q, tt);
}

void divide(int u)
{
    dele[u] = true;
    solve(u);

    for(int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if(dele[j])  continue;

        n = sz[j];
        ms[root = n] = N;
        getrt(j, -1);
        getrt(root, -1);

        divide(root);
    }
}

int main()
{
    while(scanf("%d%d", &n, &m), n or m)
    {
        for(int i = 0; i < n; i ++)  h[i] = -1,  dele[i] = 0;
        res = idx = 0;

        for(int i = 1; i < n; i ++)
        {
            int a, b, c;
            scanf("%d%d%d", &a, &b, &c);
            add(a, b, c);
            add(b, a, c);
        }

        ms[root = n] = N;
        getrt(0, -1);
        getrt(root, -1);

        divide(root);

        printf("%lld\n", res);
    }
    return 0;
}
```

三、字符串

1.KMP

最小循环节长度：$j - ne[j]$

```
#include<cstdio>
#include<cstring>


const int N = 1000010;

char p[N], s[N];
int ne[N];
int n, m;

int main()
{
    scanf("%d%s%d%s", &m, p + 1, &n, s + 1);

    for(int i = 2, j = 0; i <= m; i ++)
    {
        while(j and p[i] != p[j + 1])  j = ne[j];
        if(p[i] == p[j + 1])  j ++;
        ne[i] = j;
    }

    for(int i = 1, j = 0; i <= n; i ++)
    {
        while(j and s[i] != p[j + 1])  j = ne[j];
        if(s[i] == p[j + 1])
        {
            j ++;
            if(j == m)
            {
                printf("%d ", i - j);
                j = ne[j];
            }
        }
    }
}
```

2.Trie

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 100010;

int tr[N][26], cnt[N], idx;
```

```
char s[N];
void insert(char s[])
{
    int p = 0;
    for(int i = 0; s[i]; i ++)
    {
        int u = s[i] - 'a';
        if(!tr[p][u])  tr[p][u] = ++ idx;
        p = tr[p][u];
    }
    cnt[p] ++;
}

int query(char s[])
{
    int p = 0;
    for(int i = 0; s[i]; i ++)
    {
        int u = s[i] - 'a';
        if(!tr[p][u])  return 0;
        p = tr[p][u];
    }
    return cnt[p];
}

int main()
{
    int n;
    scanf("%d", &n);
    while(n --)
    {
        char op[3];
        scanf("%s%s", op, s);
        if(op[0] == 'I')  insert(s);
        else  printf("%d\n", query(s));
    }
    return 0;
}
```

3.AC自动机

3.1在文中出现过的单词数量

```
#include<cstdio>
#include<cstring>


const int N = 10010 * 55, M = 1e6 + 10;

int tr[N][26], cnt[N], ne[N], idx;

char s[M];

void insert()
{
    int u = 0;
```

```
    for(int i = 0; s[i]; i ++)
    {
        int k = s[i] - 'a';
        if(!tr[u][k])  tr[u][k] = ++ idx;
        u = tr[u][k];
    }
    cnt[u] ++;
}

int q[M];

void build()
{
    int hh = 0, tt = -1;
    for(int i = 0; i < 26; i ++)
        if(tr[0][i])  q[ ++ tt] = tr[0][i];

    while(hh <= tt)
    {
        int u = q[hh ++];

        for(int k = 0; k < 26; k ++)
        {
            int j = tr[u][k];
            if(!j)  tr[u][k] = tr[ne[u]][k];
            else
            {
                ne[j] = tr[ne[u]][k];
                q[ ++ tt] = j;
            }
        }
    }
}

int main()
{
    int _;
    scanf("%d", &_);
    while(_ --)
    {
        memset(tr, 0, sizeof tr);
        memset(cnt, 0, sizeof cnt);
        memset(ne, 0, sizeof ne);
        idx = 0;

        int n;
        scanf("%d", &n);
        while(n --)
        {
            scanf("%s", s);
            insert();
        }

        build();

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

        int res = 0;
```

```
        for(int i = 1, j = 0; s[i]; i ++)
        {
            int k = s[i] - 'a';
            j = tr[j][k];
            for(int p = j; p and ~cnt[p]; p = ne[p])
                res += cnt[p],  cnt[p] = -1;
        }
        printf("%d\n", res);
    }
    return 0;
}
```

3.2单词在文中的出现次数

```
#include<cstdio>
#include<cstring>

const int N = 210, M = 1e6 + 10;

int id[N];
int tr[M][26], cnt[M], ne[M], idx;
char s[M];

void insert(int num)
{
    int u = 0;
    for(int i = 0; s[i]; i ++)
    {
        int k = s[i] - 'a';
        if(!tr[u][k])  tr[u][k] = ++ idx;
        u = tr[u][k];
        cnt[u] ++;
    }
    id[num] = u;
}

int q[M];
void bfs()
{
    int hh = 0, tt = -1;
    for(int i = 0; i < 26; i ++)
        if(tr[0][i])  q[ ++ tt] = tr[0][i];

    while(hh <= tt)
    {
        int u = q[hh ++];

        for(int i = 0; i < 26; i ++)
        {
            int j = tr[u][i];
            if(!j)  tr[u][i] = tr[ne[u]][i];
            else
            {
                ne[j] = tr[ne[u]][i];
                q[++ tt] = j;
            }
        }
```

```
    }
}

int main()
{
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n; i ++)
    {
        scanf("%s", s);
        insert(i);
    }

    bfs();

    for(int i = idx - 1; i >= 0; i --)  cnt[ne[q[i]]] += cnt[q[i]];

    for(int i = 1; i <= n; i ++)  printf("%d\n", cnt[id[i]]);
    return 0;
}
```

3.3.计数DP

输入包含多组测试数据。

每组数据第一行包含整数N，表示致病DNA片段的数量。

接下来N行，每行包含一个长度不超过20的非空字符串，字符串中仅包含字符'A', 'G' , 'C' , 'T'，用以表示致病DNA片段。

再一行，包含一个长度不超过1000的非空字符串，字符串中仅包含字符'A', 'G' , 'C' , 'T'，用以表示待修复DNA片段。

最后一组测试数据后面跟一行，包含一个0，表示输入结束。

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 1010, INF = 0x3f3f3f3f;

int mp(char c)
{
    if(c == 'A')  return 0;
    if(c == 'G')  return 1;
    if(c == 'C')  return 2;
    return 3;
}

char s[N];
int tr[N][4], ne[N], cnt[N], idx;

void insert()
{
    int u = 0;
    for(int i = 0; s[i]; i ++)
```

```
    {
        int k = mp(s[i]);
        if(!tr[u][k])  tr[u][k] = ++ idx;
        u = tr[u][k];
    }
    cnt[u] ++;
}

int q[N];
void bfs()
{
    int hh = 0, tt = -1;
    for(int i = 0; i < 4; i ++)
        if(tr[0][i])  q[ ++ tt] = tr[0][i];

    while(hh <= tt)
    {
        int u = q[hh ++];

        for(int k = 0; k < 4; k ++)
        {
            int j = tr[u][k];
            if(!j)  tr[u][k] = tr[ne[u]][k];
            else
            {
                ne[j] = tr[ne[u]][k];
                q[++ tt] = j;
                cnt[j] |= cnt[ne[j]];
            }
        }
    }
}

int f[N][N];

int main()
{
    int n, t = 0;
    while(scanf("%d", &n), n)
    {
        memset(tr, 0, sizeof tr);
        memset(cnt, 0, sizeof cnt);
        memset(ne, 0, sizeof ne);
        idx = 0;

        while(n --)
        {
            scanf("%s", s);
            insert();
        }

        bfs();

        scanf("%s", s + 1);
        n = strlen(s + 1);

        memset(f, 0x3f, sizeof f);
        f[0][0] = 0;
```

```
        for(int i = 0; i < n; i ++)
            for(int j = 0; j <= idx; j ++)
                for(int k = 0; k < 4; k ++)
                {
                    int c = mp(s[i + 1]) != k;
                    int p = tr[j][k];
                    if(!cnt[p])  f[i + 1][p] = min(f[i + 1][p], f[i][j] + c);
                }
        int res = INF;
        for(int i = 0; i <= idx; i ++)  res = min(res, f[n][i]);
        if(res == INF)  res = -1;
        printf("Case %d: %d\n", ++ t, res);
    }
    return 0;
}
```

4.字符串哈希

```
#include<iostream>

using namespace std;

typedef unsigned long long ULL;

const int N = 100010, P = 131;

char s[N];
ULL h[N], p[N];

int n, m;

ULL get(int l, int r)
{
    return h[r] - h[l - 1] * p[r - l + 1];
}

int main()
{
    cin >> n >> m;
    p[0] = 1;
    cin >> s + 1;
    for(int i = 1; i <= n; i ++)
    {
        p[i] = p[i - 1] * P;
        h[i] = h[i - 1] * P + s[i];
    }

    while(m --)
    {
        int i, j, k, l;
        cin >> i  >> j >> k >> l;
        if(get(i, j) == get(k, l))  puts("Yes");
        else  puts("No");
    }
}
```

5.循环同构串的最小表示法

```
//  返回串的起始位值
int get_min(char a[N], int n)
{
    static char b[N * 2];

    for(int i = 0; i < n; i ++)  b[i] = b[i + n] = a[i];

    int i = 0, j = 1, k = 0;
    while(i < n and j < n)
    {
        for(k = 0; k < n and b[i + k] == b[j + k]; k ++);
        if(k == n)  break;

        if(b[i + k] > b[j + k])
        {
            i += k + 1;
            if(i == j)  i ++;
        }
        else
        {
            j += k + 1;
            if(i == j)  j ++;
        }
    }
    k = min(i, j);

    for(int i = 0; i < n; i ++)  a[i] = b[k + i];
    a[i + n] = 0;

    return k;
}
```

6.马拉车算法

```
//  返回最长回文子串
string Manacher(string s) {
    // Insert '#'
    string t = "$#";
    for (int i = 0; i < s.size(); ++i) {
        t += s[i];
        t += "#";
    }
    // Process t
    vector<int> p(t.size(), 0);
    int mx = 0, id = 0, resLen = 0, resCenter = 0;
    for (int i = 1; i < t.size(); ++i) {
        p[i] = mx > i ? min(p[2 * id - i], mx - i) : 1;
        while (t[i + p[i]] == t[i - p[i]]) ++p[i];
        if (mx < i + p[i]) {
            mx = i + p[i];
            id = i;
        }
        if (resLen < p[i]) {
```

```
            resLen = p[i];
            resCenter = i;
        }
    }
    return s.substr((resCenter - resLen) / 2, resLen - 1);
}
```

7.后缀数组

$sa[i]$ 排名第 i 位的是第几个后缀

$rk[i]$ 第 i 个后缀的排名是多少

$height[i]$ 为 $sa[i]$ 和 $sa[i-1]$ 的最长公共前缀

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 1e6 + 10;

int n, m;
char s[N];
int sa[N], x[N], y[N], c[N], rk[N], height[N];

void get_sa()
{
    for(int i = 1; i <= n; i ++)  c[x[i] = s[i]] ++;
    for(int i = 2; i <= m; i ++)  c[i] += c[i - 1];
    for(int i = n; i; i --)  sa[c[x[i]] --] = i;

    for(int k = 1; k <= n; k <<= 1)
    {
        int num = 0;
        for(int i = n - k + 1; i <= n; i ++)  y[ ++ num] = i;
        for(int i = 1; i <= n; i ++)
            if(sa[i] > k)  y[ ++ num] = sa[i] - k;

        for(int i = 1; i <= m; i ++)  c[i] = 0;
        for(int i = 1; i <= n; i ++)  c[x[i]] ++;
        for(int i = 2; i <= m; i ++)  c[i] += c[i - 1];
        for(int i = n; i; i --)  sa[c[x[y[i]]] --] = y[i], y[i] = 0;
        swap(x, y);

        x[sa[1]] = 1, num = 1;
        for(int i = 2; i <= n; i ++)
            x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] and y[sa[i] + k] == y[sa[i - 1]
+ k]) ? num : ++ num;
        if(num == n)  break;
        m = num;
    }
}

void get_height()
{
```

```
    for(int i = 1; i <= n; i ++)  rk[sa[i]] = i;
    for(int i = 1, k = 0; i <= n; i ++)
    {
        if(rk[i] == 1)  continue;
        if(k)  k --;
        int j = sa[rk[i] - 1];
        while(i + k <= n and j + k <= n and s[i + k] == s[j + k])  k ++;
        height[rk[i]] = k;
    }
}

int main()
{
    scanf("%s", s + 1);
    n = strlen(s + 1), m = 122;
    get_sa();
    get_height();

    for(int i = 1; i <= n; i ++)  printf("%d ", sa[i]);  puts("");
    for(int i = 1; i <= n; i ++)  printf("%d ", height[i]);  puts("");
    return 0;
}
```

四、数学

1.筛素数

1.1线性筛

时间复杂度 $O(n)$

```
int primes[N], cnt;
bool st[N];
void init(int n)
{
    for(int i = 2; i <= n; i ++)
    {
        if(!st[i])  primes[cnt ++] = i;
        for(int j = 0; primes[j] <= n / i; j ++)
        {
            st[i * primes[j]] = true;
            if(i % primes[j] == 0)  break;
        }
    }
}
```

1.2埃氏筛

时间复杂度 $O(nloglogn)$

```
int primes[N], cnt;
bool st[N];
void init(int n)
{
```

```
    for(int i = 2; i <= n; i ++)
    {
        if(!st[i])
        {
            primes[cnt ++] = i;
            for(int j = i * 2; j <= n; j += i)
                st[j] = true;
        }
    }
}
```

1.3分段筛

给定两个整数L和U，你需要在闭区间[L,U]内找到距离最接近的两个相邻质数C1和C2（即C2-C1是最小的），如果存在相同距离的其他相邻质数对，则输出第一对。

同时，你还需要找到距离最远的两个相邻质数D1和D2（即D1-D2是最大的），如果存在相同距离的其他相邻质数对，则输出第一对。

输入格式

每行输入两个整数L和U，其中L和U的差值不会超过1000000。

输出格式

对于每个L和U，输出一个结果，结果占一行。

结果包括距离最近的相邻质数对和距离最远的相邻质数对。（具体格式参照样例）

如果L和U之间不存在质数对，则输出“There are no adjacent primes.”。

输入样例：

```
2 17
14 17
```

输出样例：

```
2,3 are closest, 7,11 are most distant.
There are no adjacent primes.
```

```
#include<iostream>
#include<algorithm>
#include<cstring>
#include<vector>

using namespace std;

typedef long long LL;

const int N = 1e6 + 10, M = 50010;

bool st[N];
int primes[M], cnt;

int main()
{
```

```
    for(int i = 2; i <= M; i ++)
    {
        if(!st[i])  primes[cnt ++] = i;
        for(int j = 0; primes[j] <= M / i; j ++)
        {
            st[i * primes[j]] = true;
            if(i % primes[j] == 0)  break;
        }
    }

    int l, r;
    while(~scanf("%d%d", &l, &r))
    {
        memset(st, 0, sizeof st);

        for(int i = 0; i < cnt; i ++)
        {
            int p = primes[i];
            for(LL j = max(p * 2ll, ((LL)l + p - 1) / p * p); j <= r; j += p)
                st[j - l] = true;
        }

        vector<int> a;
        for(int i = 0; i <= r - l; i ++)
            if(i + l >= 2 && !st[i])  a.push_back(i + l);

        if(a.size() < 2)  puts("There are no adjacent primes.");
        else{
            int l1, r1, l2, r2, x1 = 1e9, x2 = 0;
            for(int i = 1; i < a.size(); i ++)
            {
                if(a[i] - a[i - 1] < x1)  l1 = a[i - 1], r1 = a[i], x1 = a[i] -
a[i - 1];
                if(a[i] - a[i - 1] > x2)  l2 = a[i - 1], r2 = a[i], x2 = a[i] -
a[i - 1];
            }
            printf("%d,%d are closest, %d,%d are most distant.\n", l1, r1, l2,
r2);
        }
    }
    return 0;
}
```

2.约数个数

由算数基本定理（每个数都可以唯一的表示成质数的乘积）得：

约数个数为不同质因子的不同次幂的组合数

每种质因子的不同组合都可以对应唯一的一个约数

每个互不相同的质因子都可以取[0,指数]次，所以取法为指数+1

所以**约数个数为所有质因子的指数+1的乘积**

```
#include<cstdio>
```

```
#include<algorithm>
#include<map>

using namespace std;

typedef long long LL;

const int mod = 1e9 + 7;

map<int, int> cnt;

int main()
{
    int n;
    scanf("%d", &n);
    while(n --)
    {
        int x;
        scanf("%d", &x);
        for(int i = 2; i <= x / i; i ++)
            while(x % i == 0)
            {
                cnt[i] ++;
                x /= i;
            }
        if(x > 1)  cnt[x] ++;
    }
    int res = 1;
    for(auto p : cnt)  res = (LL)res * (p.second + 1) % mod;
    printf("%d\n", res);
    return 0;
}
```

3.约数之和

3.1 int范围求约数之和 $O(\sqrt{n})$

由算数基本定理得
质因子的指数的不同组合可以和约数一一对应

设p为质因子，a为质因子的指数，约数之和可以表示成如下形式

$$(p_1{}^0 + p_1{}^1 + p_1{}^2 + \ldots\ldots + p_1{}^{a_1})(p_2{}^0 + p_2{}^1 + \ldots\ldots + p_2{}^{a_2}) \ldots\ldots (p_n{}^0 + p_n{}^1 + \ldots\ldots + p_n{}^{a_n})$$

在计算这个式子时，首先要计算的是

$$(p_1{}^0 + p_1{}^1 + p_1{}^2 + \ldots\ldots + p_1{}^{a_1})$$

这种指数求和 $\sum_{i=0}^{a} p^i$ 可以用如下代码来算

```
int t = 1;
while(a --)
    t = t * p + 1;
```

t的值依次为

```
1                                   1
p + 1                               p + 1
(p + 1) * p + 1                     p^2 + p + 1
((p + 1) * p + 1) + 1               p^3 + p^2 + p + 1
(((p + 1) * p + 1) * p + 1) + 1     p^4 + p^3 + p^2 + p + 1
……
```

```cpp
#include <iostream>
#include <algorithm>
#include <unordered_map>
#include <vector>

using namespace std;

typedef long long LL;

const int N = 110, mod = 1e9 + 7;

int main()
{
    int n;
    cin >> n;

    unordered_map<int, int> primes;

    while (n -- )
    {
        int x;
        cin >> x;

        for (int i = 2; i <= x / i; i ++ )
            while (x % i == 0)
            {
                x /= i;
                primes[i] ++ ;
            }

        if (x > 1) primes[x] ++ ;
    }

    LL res = 1;
    for (auto p : primes)
    {
        LL a = p.first, b = p.second;
        LL t = 1;
        while (b -- ) t = (t * a + 1) % mod;
        res = res * t % mod;
    }

    cout << res << endl;

    return 0;
}
```

3.2 求 A^B 的约数之和

```cpp
#include<cstdio>
#include<algorithm>

using namespace std;

const int mod = 9901;

int qmi(int a, int k)
{
    a %= mod;
    int res = 1;
    while(k)
    {
        if(k & 1)  res = res * a % mod;
        k >>= 1;
        a = a * a % mod;
    }
    return res;
}

int sum(int a, int k)                //  求首项为1, 公比为a的等比数列的前k + 1项和
{
    if(!k)  return 1;
    if(k % 2 == 0)  return (1 + a % mod * sum(a, k - 1)) % mod;
    return sum(a, k >> 1) % mod * (1 + qmi(a, k / 2 + 1)) % mod;
}

int main()
{
    int a, b;
    scanf("%d%d", &a, &b);

    if(!a)  return 0 * printf("0");

    int res = 1;
    for(int i = 2; i <= a; i ++)
        if(a % i == 0)
        {
            int cnt = 0;
            while(a % i == 0)  a /= i,  cnt ++;
            res = res * sum(i, cnt * b) % mod;
        }
    if(a > 1)  res = res * sum(a, b) % mod;
    printf("%d", res);
    return 0;
}
```

4.欧拉函数

$phi(n)$ 为 $1 \sim n$ 中与 n 互质的数的个数

4.1欧拉函数

时间复杂度 $O(\sqrt{n})$

```
int phi(int n)
{
    int res = n;
    for(int i = 2; i <= n / i; 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;
}
```

4.2线性筛求欧拉函数

时间复杂度 $O(n)$

```
int primes[N], cnt;
bool st[N];

int phi[N];

void init(int n)
{
    phi[1] = 1;
    for(int i = 2; i <= n; i ++)
    {
        if(!st[i])  primes[cnt ++] = i,  phi[i] = i - 1;
        for(int j = 0; primes[j] <= n / i; j ++)
        {
            st[i * primes[j]] = true;
            if(i % primes[j] == 0)
            {
                phi[i * primes[j]] = phi[i] * primes[j];
                break;
            }
            phi[i * primes[j]] = phi[i] * (primes[j] - 1);
        }
    }
}
```

4.3求gcd为素数的对数

给定整数N，求1<=x,y<=N且GCD(x,y)为素数的数对(x,y)有多少对。

```
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

typedef long long LL;

const int N = 1e7 + 10;
```

```
int primes[N], cnt;
bool st[N];

int phi[N];
LL sum[N];

void init(int n)
{
    for(int i = 2; i <= n; i ++)
    {
        if(!st[i])
        {
            primes[cnt ++] = i;
            phi[i] = i - 1;
        }
        for(int j = 0; primes[j] <= n / i; j ++)
        {
            st[i * primes[j]] = true;
            if(i % primes[j] == 0)
            {
                phi[i * primes[j]] = phi[i] * primes[j];
                break;
            }
            phi[i * primes[j]] = phi[i] * (primes[j] - 1);
        }
    }
}

int main()
{
    int n;
    scanf("%d", &n);

    init(n);

    for(int i = 1; i <= n; i ++)  sum[i] = sum[i - 1] + phi[i];

    LL res = 0;
    for(int i = 0; i < cnt; i ++)  res += sum[n / primes[i]] * 2 + 1;
    printf("%lld", res);
    return 0;
}
```

5.欧几里得算法

5.1gcd

时间复杂度 $O(logn)$

```
int gcd(int a, int b)
{
    return b ? gcd(b, a % b) : a;
}
```

5.2exgcd

构造一组 x, y 使得 $ax + by = gcd(a, b)$

```
int exgcd(int a, int b, int &x, int &y)
{
    if(!b)
    {
        x = 1, y = 0;
        return a;
    }

    int d = exgcd(b, a % b, y, x);

    y -= a / b * x;

    return d;
}
```

5.3exgcd求解线性同余方程组

如果 $a \cdot x \equiv b(mod\, m)$

那么存在 $y \in Z$ 使得 $a \cdot x + m \cdot y = b$

令$d = gcd(a, m)$

$exgcd(a, m, x_0, y_0)$ 得$a \cdot x_0 + m \cdot y_0 = d$

`if(b % d == 0)` 则有解

等式两边同乘 $\frac{b}{d}$ 得

$\frac{b}{d}(a \cdot x_0 + m \cdot y_0) = b$

$a \cdot (\frac{b}{d} \cdot x_0) + m \cdot (\frac{b}{d} \cdot y_0) = b$

$x = b/d \cdot x_0$

对于每组 a, b, m 构造一个 x 使得 $ax \equiv b(mod\, m)$

```
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

typedef long long LL;

int exgcd(int a, int b, int &x, int &y)
{
    if(!b)
    {
        x = 1, y = 0;
        return a;
    }
```

```
    int d = exgcd(b, a % b, y, x);

    y -= a / b * x;

    return d;
}

int main()
{
    int n;
    scanf("%d", &n);
    while(n --)
    {
        int a, b, m;
        scanf("%d%d%d", &a, &b, &m);

        int x, y;
        int d = exgcd(a, m, x, y);

        if(b % d)  puts("impossible");
        else  printf("%lld\n", (LL)x * b / d % m);
    }
    return 0;
}
```

6.中国剩余定理

6.1中国剩余定理

给出 n 组 a, m，保证 m 之间互质

求一个最小的非负整数 x，要求对于任何一组a, m 满足 $x \equiv a(mod\ m)$

```
#include<cstdio>

typedef long long LL;

const int N = 1010;

LL a[N], m[N];

LL exgcd(LL a, LL b, LL &x, LL &y)
{
    if(!b)
    {
        x = 1, y = 0;
        return a;
    }
    LL d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}

int main()
{
    int n;
```

```
    while(~scanf("%d", &n))
    {
        LL M = 1;
        for(int i = 0; i < n; i ++)
        {
            scanf("%lld%lld", &a[i], &m[i]);
            M *= m[i];
        }

        LL res = 0;
        for(int i = 0; i < n; i ++)
        {
            LL Mi = M / m[i];
            LL x, y;
            exgcd(Mi, m[i], x, y);
            res += a[i] * Mi * x;
        }
        printf("%lld\n", (res % M + M) % M);
    }
    return 0;
}
```

6.2扩展中国剩余定理

给出 n 组 a, m

求一个最小的非负整数 x， 要求对于任何一组a, m 满足 $x \equiv a(mod\ m)$

```
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

typedef long long LL;

LL exgcd(LL a, LL b, LL &x, LL &y)
{
    if(!b)
    {
        x = 1, y = 0;
        return a;
    }

    LL d = exgcd(b, a % b, y, x);

    y -= a / b * x;

    return d;
}

LL mod(LL a, LL b)
{
    return (a % b + b) % b;
}

int main()
```

```
{
    int n;
    scanf("%d", &n);

    bool flag = true;
    LL a1, m1, a2, m2;
    scanf("%lld%lld", &a1, &m1);
    for(int i = 1; i < n; i ++)
    {
        scanf("%lld%lld", &a2, &m2);

        LL k1, k2;
        LL d = exgcd(a1, -a2, k1, k2);

        if((m2 - m1) % d)
        {
            flag = false;
            break;
        }

        k1 = k1 * (m2 - m1) / d;
        k1 = mod(k1, a2 / d);
        m1 += k1 * a1;
        a1 = abs(a1 / d * a2);
    }

    if(flag)  printf("%lld", mod(m1, a1));
    else  printf("-1");
    return 0;
}
```

7.高斯消元

7.1高斯消元解线性方程组

求解这样的方程组

```
M[1][1]x[1] * M[1][2]x[2] * … * M[1][n]x[n] = B[1]
M[2][1]x[1] * M[2][2]x[2] * … * M[2][n]x[n] = B[2]
…
M[n][1]x[1] * M[n][2]x[2] * … * M[n][n]x[n] = B[n]
```

```
#include<cstdio>
#include<algorithm>
#include<cmath>

using namespace std;

const int N = 110;
const double eps = 1e-6;

double a[N][N];
```

```
int n;
int gauss()
{
    int r, c;
    for(r = 0, c = 0; c < n; c ++)
    {
        if(fabs(a[r][c]) < eps)
        {
            int t = r;
            for(int i = r + 1; i < n; i ++)
                if(fabs(a[i][c]) > eps)
                {
                    t = i;
                    break;
                }

            if(t == r)  continue;

            for(int i = 0; i <= n; i ++)  swap(a[r][i], a[t][i]);
        }

        for(int i = n; i >= c; i --)  a[r][i] /= a[r][c];

        for(int i = r + 1; i < n; i ++)
            if(fabs(a[i][c]) > eps)
                for(int j = n; j >= c; j --)
                    a[i][j] -= a[i][c] * a[r][j];

        r ++;
    }

    if(r < n)
    {
        for(int i = r; i < n; i ++)
            if(fabs(a[i][n]) > eps)  return 1;        //无解

        return 2;           //多解
    }

    for(int i = n - 1; i; i --)
        for(int j = 0; j < i; j ++)
            a[j][n] -= a[i][n] * a[j][i];

    return 0;
}

int main()
{
    scanf("%d", &n);
    for(int i = 0; i < n; i ++)
        for(int j = 0; j <= n; j ++)  scanf("%lf", &a[i][j]);

    int t = gauss();

    if(!t)
        for(int i = 0; i < n; i ++)  printf("%.2lf\n", a[i][n]);
    else if(t == 1)
        printf("No solution");
```

```
    else
        printf("Infinite group solutions");

    return 0;
}
```

7.2高斯消元解异或线性方程组

```
M[1][1]x[1] ^ M[1][2]x[2] ^ … ^ M[1][n]x[n] = B[1]
M[2][1]x[1] ^ M[2][2]x[2] ^ … ^ M[2][n]x[n] = B[2]
…
M[n][1]x[1] ^ M[n][2]x[2] ^ … ^ M[n][n]x[n] = B[n]
```

```
#include<cstdio>
#include<algorithm>

using namespace std;

const int N = 110;

int a[N][N], n;
int gauss()
{
    int r, c;
    for(r = 0, c = 0; c < n; c ++)
    {
        if(!a[r][c])
        {
            int t = r;
            for(int i = r + 1; i < n; i ++)
                if(a[i][c])
                {
                    t = i;
                    break;
                }

            if(t == r)  continue;

            for(int i = c; i <= n; i ++)  swap(a[r][i], a[t][i]);
        }

        for(int i = r + 1; i < n; i ++)
            if(a[i][c])
                for(int j = n; j >= c; j --)
                    a[i][j] ^= a[r][j];

        r ++;
    }

    if(r < n)
    {
        for(int i = r; i < n; i ++)
            if(a[i][n])  return 1;      //无解
```

```
        return 2;                       //多解
    }

    for(int i = n - 1; i; i --)
        for(int j = 0; j < i; j ++)
            a[j][n] ^= a[i][n] & a[j][i];

    return 0;
}

int main()
{
    scanf("%d", &n);
    for(int i = 0; i < n; i ++)
        for(int j = 0; j <= n; j ++)
            scanf("%d", &a[i][j]);

    int t = gauss();
    if(!t)
        for(int i = 0; i < n; i ++)  printf("%d\n", a[i][n]);
    else if(t == 1)
        printf("No solution");
    else
        printf("Multiple sets of solutions");

    return 0;
}
```

8.求组合数

8.1杨辉三角 $O(n^2)$ 预处理

```
int c[N][N];
void init(int n)
{
    for(int i = 0; i <= n; i ++)
        for(int j = 0; j <= i; j ++)
            if(!j)  c[i][j] = 1;
            else  c[i][j] = (c[i - 1][j - 1] + c[i - 1][j]) % mod;
}
```

8.2 $O(n)$ 预处理阶乘

```
int qmi(int a, int k)
{
    int res = 1;
    while(k)
    {
        if(k & 1)  res = (LL)res * a % mod;
        k >>= 1;
        a = (LL)a * a % mod;
    }
    return res;
}
```

```
int fact[N], infact[N];

void init(int n)
{
    fact[0] = infact[0] = 1;
    for(int i = 1; i <= n; i ++)
    {
        fact[i] = (LL)fact[i - 1] * i % mod;
        infact[i] = qmi(fact[i], mod - 2);
    }
}

int C(int a, int b)
{
    if(a < b)  return 0;
    return (LL)fact[a] * infact[a - b] % mod * infact[b] % mod;
}
```

8.3卢卡斯定理

$$C_a^b \ mod \ p = C_{a \ mod \ p}^{b \ mod \ p} \cdot C_{a/p}^{b/p} \ mod \ p$$

```
int qmi(int a, int k)
{
    int res = 1;
    while(k)
    {
        if(k & 1)  res = (LL)res * a % p;
        k >>= 1;
        a = (LL)a * a % p;
    }
    return res;
}

int C(int a, int b)
{
    if(a < b)  return 0;

    int up = 1, down = 1;
    for(int i = a, j = 1; j <= b; i --, j ++)
    {
        up = (LL)up * i % p;
        down = (LL)down * j % p;
    }
    return (LL)up * qmi(down, p - 2) % p;
}

int Lucas(LL a, LL b)
{
    if(a < p && b < p)  return C(a, b);
    else  return (LL)C(a % p, b % p) * Lucas(a / p, b / p) % p;
}
```

8.4高精度

```cpp
#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;

const int N = 5010;

int primes[N], cnt;
int sum[N];
bool st[N];

void get_primes(int n)
{
    for(int i = 2; i <= n; i ++)
    {
        if(!st[i])  primes[cnt ++] = i;
        for(int j = 0; primes[j] <= n / i; j ++)
        {
            st[primes[j] * i] = true;
            if(i % primes[j] == 0)  break;
        }
    }
}

int get(int n, int p)
{
    int res = 0;
    while(n)
    {
        res += n / p;
        n /= p;
    }
    return res;
}

vector<int> mul(vector<int> a, int b)
{
    vector<int> c;
    int t = 0;
    for(int i = 0; i < a.size(); i ++)
    {
        t += a[i] * b;
        c.push_back(t % 10);
        t /= 10;
    }
    while(t)
    {
        c.push_back(t % 10);
        t /= 10;
    }
    return c;
}

int main()
{
    int a, b;
    scanf("%d%d", &a, &b);
```

```
    get_primes(a);

    for(int i = 0; i < cnt; i ++)
    {
        int p = primes[i];
        sum[i] = get(a, p) - get(a - b, p) - get(b, p);
    }

    vector<int> res;
    res.push_back(1);

    for(int i = 0; i < cnt; i ++)
        for(int j = 0; j < sum[i]; j ++)
            res = mul(res, primes[i]);

    for(int i = res.size() - 1; i >= 0; i --)  printf("%d", res[i]);

    return 0;
}
```

8.5卡特兰数

求 n 对括号的合法方案数

```
#include<iostream>
#include<algorithm>

using namespace std;

typedef long long LL;

const int mod = 1e9 + 7;

int qmi(int a, int k)
{
    int res = 1;
    while(k)
    {
        if(k & 1)  res = (LL)res * a % mod;
        k >>= 1;
        a = (LL)a * a % mod;
    }
    return res;
}

int C(int a, int b)
{
    if(a < b)  return 0;

    int up = 1, down = 1;
    for(int i = a, j = 1; j <= b; i --, j ++)
    {
        up = (LL)up * i % mod;
        down = (LL)down * j % mod;
    }
    return (LL)up * qmi(down, mod - 2) % mod;
```

```
}

int main()
{
    int n;
    scanf("%d", &n);
    printf("%d", (C(2 * n, n) - C(2 * n, n - 1) + mod) % mod);
    return 0;
}
```

8.6大施罗德数

在组合数学中，施罗德数用来描述从 $(0,0)$ 到 (n,n) 的网格中，只能使用 $(1,0)$、$(0,1)$、$(1,1)$ 三种移动方式，始终位于对角线下方且不越过对角线的路径数。

施罗德数的前几项(从第 0 项算起)为 $1,2,6,22,90,394,1806,8558,41586,206098$.....

有递推式

$(i+1)F_i=(6n-3)F_{i-1}-(i-2)F_{i-2}$

使得 $F_0=S_0$，对于任意的 $i\geq 1$ 都有 $2F_i=S_i$

```
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

typedef long long LL;

const int N = 100010, mod = 1e9 + 7;

int qmi(int a, int k)
{
    int res = 1;
    while(k)
    {
        if(k & 1)  res = (LL)res * a % mod;
        k >>= 1;
        a = (LL)a * a % mod;
    }
    return res;
}

LL f[N], s[N];
void init(int n)
{
    f[0] = s[0] = f[1] = 1;
    s[1] = 2;
    for(int i = 2; i <= n; i ++)
    {
        f[i] = ((6 * i - 3) * f[i - 1] - (i - 2) * f[i - 2]) % mod * qmi(i + 1,
mod - 2) % mod;
        s[i] = f[i] * 2 % mod;
```

```
    }
}

int main()
{
    init(N - 1);

    for(int i = 0; i < 10; i ++)  printf("%d\n", s[i]);
    return 0;
}
```

9.容斥原理

求一些集合的并集的元素个数

$|S1 \cup S2 \cup S3 \cup ..... \cup Sn|$

$= |S_1| + |S_2| + |S_3| + ... + |S_n|$

$-|S_1 \cup S_2| - |S_1 \cup S_3| - ..... - |S_{n-1} \cup S_n|$

$+|S_1 \cup S_2 \cup S_3| + ... + |S_{n-2} \cup S_{n-1} \cup S_n| .....$

给出 m 个不同的素数 p_1 ~ p_m

求 1 ~ n 中能被其中至少一个素数整除的数有多少个

```
#include<cstdio>

typedef long long LL;

const int N = 20;

int p[N];

int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    for(int i = 0; i < m; i ++)  scanf("%d", &p[i]);

    int res = 0;
    for(int i = 1; i < 1 << m; i ++)
    {
        int x = 1, cnt = 0;
        for(int j = 0; j < m; j ++)
            if(i >> j & 1)
            {
                if((LL)x * p[j] > n)
                {
                    x = -1;
                    break;
                }
                x *= p[j];
                cnt ++;
            }
        if(~x)
```

```
        {
            if(cnt & 1)  res += n / x;
            else  res -= n / x;
        }
    }
    printf("%d\n", res);
    return 0;
}
```

10.快速幂

时间复杂度 $O(logk)$

```
int qmi(int a, int k)
{
    int res = 1;
    while(k)
    {
        if(k & 1)  res = (LL)res * a % mod;
        k >>= 1;
        a = (LL)a * a % mod;
    }
    return res;
}
```

10.1快速幂求逆元

当 mod 为素数时，$qmi(a, mod - 2)$ 为 a 在模 mod 下的乘法逆元

10.2龟速乘

```
LL mul(LL a, LL k)
{
    LL res = 0;
    while(k)
    {
        if(k & 1)  res = (res + a) % mod;
        k >>= 1;
        a = (a + a) % mod;
    }
    return res;
}
```

11.矩阵乘法

11.1斐波那契数列前n项和

```
#include<cstdio>
#include<cstring>

typedef long long LL;

const int N = 3;
```

```
int k, mod;

void mul(int c[N], int a[N][N], int b[N])
{
    static int t[N];
    memset(t, 0, sizeof t);

    for(int i = 0; i < N; i ++)
        for(int j = 0; j < N; j ++)
            t[i] = (t[i] + (LL)a[i][j] * b[j]) % mod;
    memcpy(c, t, sizeof t);
}

void mul(int c[N][N], int a[N][N], int b[N][N])
{
    static int t[N][N];
    memset(t, 0, sizeof t);

    for(int i = 0; i < N; i ++)
        for(int j = 0; j < N; j ++)
            for(int k = 0; k < N; k ++)
                t[i][j] = (t[i][j] + (LL)a[i][k] * b[k][j]) % mod;
    memcpy(c, t, sizeof t);
}

int main()
{
    int f[N] = {0, 1, 0};
    int a[N][N] = {
        {0, 1, 0},
        {1, 1, 0},
        {0 ,1, 1}
    };

    scanf("%d%d", &k, &mod);

    while(k)
    {
        if(k & 1)  mul(f, a, f);
        k >>= 1;
        mul(a, a, a);
    }
    printf("%d", f[2]);
    return 0;
}
```

四、搜索

1.双向搜索

1.1双向BFS

已知有两个字串 A, B 及一组字串变换的规则（至多6个规则）：

A1 -> B1

A2 -> B2

...

规则的含义为：在 A 中的子串 A1 可以变换为 B1、A2 可以变换为 B2 ...。

例如：A = 'abcd' B = 'xyz'

变换规则为：

'abc'->'xu' 'ud'->'y' 'y'->'yz'

则此时，A 可以经过一系列的变换变为 B，其变换的过程为：

'abcd'->'xud'->'xy'->'xyz'

共进行了三次变换，使得 A 变换为B。

输入样例：

```
abcd xyz
abc xu
ud y
y yz
```

输出样例：

```
3
```

```
#include<iostream>
#include<algorithm>
#include<unordered_map>
#include<queue>

using namespace std;

const int N = 6;

string a[N], b[N];
int n;

typedef unordered_map<string, int> HASH;

queue<string> qa, qb;
HASH da, db;

int extend(queue<string> &q, HASH &da, HASH &db, string a[N], string b[N])
{
    string t = q.front();
    q.pop();

    for(int i = 0; i < t.size(); i ++)
        for(int j = 0; j < n; j ++)
            if(t.substr(i, a[j].size()) == a[j])
            {
                string s = t.substr(0, i) + b[j] + t.substr(i + a[j].size());
                //cout << t << "->" << s << endl;
                if(db.count(s))  return da[t] + 1 + db[s];
```

```
                if(da.count(s))  continue;
                da[s] = da[t] + 1;
                q.push(s);
            }
    return 11;
}

int bfs(string start, string end)
{
    da[start] = 0, db[end] = 0;
    qa.push(start),  qb.push(end);

    while(qa.size() && qb.size())
    {
        int t;
        if(qa.size() <= qb.size())  t = extend(qa, da, db, a, b);
        else  t = extend(qb, db, da, b, a);

        if(t <= 10)  return t;
    }
    return 11;
}

int main()
{
    string start, end;
    cin >> start >> end;
    while(cin >> a[n] >> b[n])  n ++;

    int step = bfs(start, end);
    if(step > 10)  cout << "NO ANSWER!" << endl;
    else  cout << step << endl;
    return 0;
}
```

1.2双向DFS

达达帮翰翰给女生送礼物，翰翰一共准备了N个礼物，其中第i个礼物的重量是G[i]。

达达的力气很大，他一次可以搬动重量之和不超过W的任意多个物品。

达达希望一次搬掉尽量重的一些物品，请你告诉达达在他的力气范围内一次性能搬动的最大重量是多少。

输入格式

第一行两个整数，分别代表W和N。

以后N行，每行一个正整数表示G[i]。

输出格式

仅一个整数，表示达达在他的力气范围内一次性能搬动的最大重量。

数据范围

$1 \leq N \leq 46$
$1 \leq W, G[i] \leq 2^{31} - 1$

输入样例：

```
20 5
7
5
4
18
1
```

输出样例：

```
19
```

```cpp
#include<iostream>
#include<algorithm>

using namespace std;

typedef long long LL;

const int N = 46, M = 1 << 25;

int n, m, k;
int w[N], ans;
int weight[M], cnt = 1;

void dfs1(int u, int sum)
{
    if(u == k)
    {
        weight[cnt ++] = sum;
        return;
    }

    dfs1(u + 1, sum);
    if((LL)sum + w[u] <= m)  dfs1(u + 1, sum + w[u]);
}

void dfs2(int u, int sum)
{
    if(u == n)
    {
        int l = 0, r = cnt;
        while(l < r)
        {
            int mid = l + r + 1 >> 1;
            if((LL)sum + weight[mid] <= m)  l = mid;
            else  r = mid - 1;
        }
        ans = max(ans, sum + weight[l]);
        return;
    }

    dfs2(u + 1, sum);
    if((LL)sum + w[u] <= m)  dfs2(u + 1, sum + w[u]);
}
```

```
int main()
{
    scanf("%d%d", &m, &n);
    for(int i = 0; i < n; i ++)  scanf("%d", &w[i]);

    sort(w, w + n);
    reverse(w, w + n);

    k = n / 2 + 2;
    dfs1(0, 0);

    sort(weight, weight + cnt);
    int j = 0;
    for(int i = 1; i < cnt; i ++)
        if(weight[i] != weight[j])
            weight[++ j] = weight[i];
    cnt = j;

    dfs2(k, 0);
    printf("%d", ans);
    return 0;
}
```

2.迭代加深

2.1加成序列

满足如下条件的序列X（序列中元素被标号为1、2、3...m）被称为“加成序列”：

1、X[1]=1

2、X[m]=n

3、X[1]<X[2]<...<X[m-1]<X[m]

4、对于每个 k（$2 \le k \le m$）都存在两个整数 i 和 j （$1 \le i,j \le k-1$，i 和 j 可相等），使得 X[k]=X[i]+X[j]。

你的任务是：给定一个整数n，找出符合上述条件的长度m最小的“加成序列”。

如果有多个满足要求的答案，只需要找出任意一个可行解。

```
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 110;

int a[N], n;
bool dfs(int num, int cnt)
{
    if(num == cnt)  return a[cnt - 1] == n;

    bool st[N] = {0};
```

```
    for(int i = num - 1; i >= 0; i --)
        for(int j = i; j >= 0; j --)
        {
            int sum = a[i] + a[j];
            if(st[sum] || sum > n || sum <= a[num - 1])  continue;

            st[sum] = true;
            a[num] = sum;
            if(dfs(num + 1, cnt))  return true;
        }
    return false;
}

int main()
{
    a[0] = 1;
    while(scanf("%d", &n), n)
    {
        int cnt = 1;
        while(!dfs(1, cnt))  cnt ++;
        for(int i = 0; i < cnt; i ++)  printf("%d ", a[i]);  printf("\n");
    }
    return 0;
}
```

3.爆搜剪枝

3.1小猫爬山

翰翰和达达饲养了N只小猫，这天，小猫们要去爬山。

经历了千辛万苦，小猫们终于爬上了山顶，但是疲倦的它们再也不想徒步走下山了（呜咕>_<）。

翰翰和达达只好花钱让它们坐索道下山。

索道上的缆车最大承重量为W，而N只小猫的重量分别是C1、C2……CNC1、C2……CN。

当然，每辆缆车上的小猫的重量之和不能超过W。

每租用一辆缆车，翰翰和达达就要付1美元，所以他们想知道，最少需要付多少美元才能把这N只小猫都运送下山?

输入格式

第1行：包含两个用空格隔开的整数，N和W。

第2..N+1行：每行一个整数，其中第i+1行的整数表示第i只小猫的重量 C_i。

输出格式

输出一个整数，表示最少需要多少美元，也就是最少需要多少辆缆车。

数据范围

$1 \leq N \leq 18$
$1 \leq C_i \leq W \leq 10^8$

输入样例：

```
5 1996
1
2
1994
12
29
```

输出样例：

```
2
```

```cpp
#include<cstdio>
#include<algorithm>

using namespace std;

const int N = 20;

int n, m;
int w[N];
int sum[N];
int res = 1e9;

void dfs(int u, int cnt)
{
    if(cnt > res)  return;
    if(u >= n)
    {
        res = min(res, cnt);
        return;
    }

    for(int i = 0; i < cnt; i ++)
        if(sum[i] + w[u] <= m)
        {
            sum[i] += w[u];
            dfs(u + 1, cnt);
            sum[i] -= w[u];
        }
    sum[cnt] += w[u];
    dfs(u + 1, cnt + 1);
    sum[cnt] -= w[u];
}


int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 0; i < n; i ++)  scanf("%d", &w[i]);
    sort(w, w + n);
    reverse(w, w + n);

    dfs(0, 0);

    printf("%d", res);
```

```
    return 0;
}
```

3.2数独

```
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 9, M = 1 << N;

int ones[M], map[M];
int row[N], col[N], cell[3][3];
//记录状态，第t位为1则可填t + 1

char str[100];

//is = true在(i, j)处填上t + 1，否则把(i, j)处的t + 1删掉
void draw(int i, int j, int t, bool is)
{
    if(is)  str[i * N + j] = '1' + t;
    else  str[i * N + j] = '.';

    int v = 1 << t;
    row[i] ^= v;
    col[j] ^= v;
    cell[i / 3][j / 3] ^= v;
}

int lowbit(int x)   //返回x的最低位1
{
    return x & -x;
}

int get(int i, int j)
{
    return row[i] & col[j] & cell[i / 3][j / 3];
}

bool dfs(int cnt)
{
    if(!cnt)  return true;

    int minv = 10, x, y;
    for(int i = 0; i < N; i ++)
        for(int j = 0; j < N; j ++)
            if(str[i * N + j] == '.')
            {
                int state = get(i, j);
                if(ones[state] < minv)
                {
                    minv = ones[state];     //找到可填数最少的一个格
                    x = i, y = j;
                }
            }
```

```
    int state = get(x, y);
    for(int i = state; i; i -= lowbit(i))
    {
        int t = map[lowbit(i)];

        draw(x, y, t, true);
        if(dfs(cnt - 1))  return true;
        draw(x, y, t, false);
    }

    return false;
}

int main()
{
    for(int i = 0; i < N; i ++)  map[1 << i] = i;
    for(int i = 0; i < M; i ++)
        for(int j = 0; j < N; j ++)
            if(i >> j & 1)  ones[i] ++;

    while(gets(str), str[0] != 'e')
    {
        for(int i = 0; i < N; i ++)  row[i] = col[i] = M - 1;
        for(int i = 0; i < 3; i ++)
            for(int j = 0; j < 3; j ++)  cell[i][j] = M - 1;

        int cnt = 0;
        for(int i = 0; i < N; i ++)
            for(int j = 0; j < N; j ++)
                if(str[i * N + j] != '.')
                {
                    int t = str[i * N + j] - '1';
                    draw(i, j, t, true);
                }else
                    cnt ++;

        dfs(cnt);
        puts(str);
    }
    return 0;
}
```

3.3小木棍

乔治拿来一组等长的木棒，将它们随机地砍断，使得每一节木棍的长度都不超过50个长度单位。

然后他又想把这些木棍恢复到为裁截前的状态，但忘记了初始时有多少木棒以及木棒的初始长度。

请你设计一个程序，帮助乔治计算木棒的可能最小长度。

每一节木棍的长度都用大于零的整数表示。

输入格式

输入包含多组数据，每组数据包括两行。

第一行是一个不超过64的整数，表示砍断之后共有多少节木棍。

第二行是截断以后，所得到的各节木棍的长度。

在最后一组数据之后，是一个零。

输出格式

为每组数据，分别输出原始木棒的可能最小长度，每组数据占一行。

数据范围

数据保证每一节木棍的长度均不大于50。

输入样例：

```
9
5 2 1 5 2 1 5 2 1
4
1 2 3 4
0
```

输出样例：

```
6
5
```

```cpp
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 64;

int w[N], n, sum, length;

bool st[N];

bool dfs(int num, int len, int start)
{
    if(num * length == sum)  return true;
    if(len == length)  return dfs(num + 1, 0, 0);

    for(int i = start; i < n; i ++)
    {
        if(st[i] || len + w[i] > length)  continue;

        st[i] = true;
        if(dfs(num, len + w[i], i + 1))  return true;
        st[i] = false;

        if(len == 0 || len + w[i] == length)  return false;

        int j = i + 1;
        while(j < n && w[j] == w[i])  j ++;
        i = j - 1;
    }
    return false;
}
```

```
int main()
{
    while(scanf("%d", &n), n)
    {
        memset(st, false, sizeof st);
        sum = 0;
        for(int i = 0; i < n; i ++)  scanf("%d", &w[i]),  sum += w[i];

        sort(w, w + n);
        reverse(w, w + n);

        length = w[0];
        while(length <= sum)
        {
            if(sum % length == 0 && dfs(0, 0, 0))
            {
                printf("%d\n", length);
                break;
            }
            length ++;
        }
    }
    return 0;
}
```

3.4生日蛋糕

7月17日是Mr.W的生日，ACM-THU为此要制作一个体积为 $N\pi$的 M 层生日蛋糕，每层都是一个圆柱体。

设从下往上数第i层蛋糕是半径iRi, 高度为HiHi的圆柱。

当$i < M$时，要求$R_i > R_{i+1}$且 $H_i > H_{i+1}$。

由于要在蛋糕上抹奶油，为尽可能节约经费，我们希望蛋糕外表面（最下一层的下底面除外）的面积 Q 最小。

令 $Q = S\pi$ ，请编程对给出的 N 和 M，找出蛋糕的制作方案（适当的 R_i和 H_i 的值），使 S 最小。

除 Q外，以上所有数据皆为正整数 。

输入格式

输入包含两行，第一行为整数N（$N <= 10000$），表示待制作的蛋糕的体积为 $N\pi$。

第二行为整数$M(M <= 20)$，表示蛋糕的层数为 M。

输出格式

输出仅一行，是一个正整数S（若无解则$S = 0$）。

数据范围

$1 \leq N \leq 10000$
$1 \leq M \leq 20$

输入样例:

```
100
2
```

输出样例：

```
68
```

```cpp
#include<cstdio>
#include<algorithm>
#include<cmath>

using namespace std;

const int N = 22, INF = 2e9;

int n, m;
int minv[N], mins[N];
int R[N], H[N];
int res = INF;

void dfs(int u, int v, int s)
{
    if(v + minv[u] > n)  return;
    if(s + mins[u] >= res)  return;
    if(s + 2 * (n - v) / R[u + 1] >= res)  return;

    if(!u)
    {
        if(v == n)  res = s;
        return;
    }

    for(int r = min(R[u + 1] - 1, (int)sqrt(n - v)); r >= u; r --)
        for(int h = min(H[u + 1] - 1, (n - v) / r / r); h >= u; h --)
        {
            int t = 0;
            if(u == m)  t = r * r;
            R[u] = r, H[u] = h;
            dfs(u - 1, v + r * r * h, s + 2 * r * h + t);
        }
}

int main()
{
    scanf("%d%d", &n, &m);

    for(int i = 1; i <= m; i ++)
    {
        minv[i] = minv[i - 1] + i * i * i;
        mins[i] = mins[i - 1] + 2 * i * i;
    }
    R[m + 1] = H[m + 1] = INF;

    dfs(m, 0, 0);
```

```
    if(res == INF)  res = 0;
    printf("%d", res);
    return 0;
}
```

4.A*

4.1第k短路

给定一张N个点（编号1,2...N），M条边的有向图，求从起点S到终点T的第K短路的长度，路径允许重复经过点或边。

```
#include<iostream>
#include<algorithm>
#include<cstring>
#include<queue>

using namespace std;

typedef pair<int, int> PII;
typedef pair<int, PII> PIII;

const int N = 1010, M = 200010;

int h[N], rh[N], e[M], w[M], ne[M], idx;
void add(int h[], int a, int b, int c)
{
    e[idx] = b;
    w[idx] = c;
    ne[idx] = h[a];
    h[a] = idx ++;
}

int n, m, S, T, k;
int d[N];           //到终点的最短距离
bool st[N];
void dijkstra()     //求到终点的最短距离
{
    memset(d, 0x3f, sizeof d);
    d[T] = 0;

    priority_queue<PII, vector<PII>, greater<PII> > heap;
    heap.push((PII){0, T});

    while(heap.size())
    {
        PII t = heap.top();  heap.pop();

        int u = t.second;
        if(st[u])  continue;
        st[u] = true;

        for(int i = rh[u]; ~i; i = ne[i])
        {
            int j = e[i], dist = t.first + w[i];
            if(dist < d[j])
```

```
            {
                d[j] = dist;
                heap.push((PII){d[j], j});
            }
        }
    }
}

int astar()
{
    priority_queue<PIII, vector<PIII>, greater<PIII> > heap;
    heap.push((PIII){d[S], (PII){0, S}});

    int cnt = 0;        //终点被更新次数
    while(heap.size())
    {
        PIII t = heap.top();  heap.pop();

        int u = t.second.second;
        if(u == T)  cnt ++;
        if(cnt == k)  return t.second.first;

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i], dist = t.second.first + w[i];
            heap.push((PIII){dist + d[j], (PII){dist, j}});
        }
    }
    return -1;
}

int main()
{
    memset(h, -1, sizeof h);
    memset(rh, -1, sizeof rh);

    scanf("%d%d", &n, &m);
    while(m --)
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        add(h, a, b, c);
        add(rh, b, a, c);
    }
    scanf("%d%d%d", &S, &T, &k);
    if(S == T)  k ++;

    dijkstra();
    printf("%d", astar());
    return 0;
}
```

4.2八数码

```
#include <cstring>
#include <iostream>
#include <algorithm>
```

```cpp
#include <queue>
#include <unordered_map>

using namespace std;

int f(string s)
{
    int res = 0;
    for(int i = 0; i < s.size(); i ++)
    {
        if(s[i] == 'x')  continue;

        int u = s[i] - '1';
        res += abs(u / 3 - i / 3) + abs(u % 3 - i % 3);
    }
    return res;
}

typedef struct PSI{
    string s;
    int val;
    string path;

    bool operator< (const struct PSI &W) const{
        return val < W.val;
    }
    bool operator> (const struct PSI &W) const{
        return val > W.val;
    }
}PSI;

unordered_map<string, int> dist;

string bfs(string start)
{
    int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1};
    char op[4] = {'u', 'r', 'd', 'l'};

    string end = "12345678x";

    priority_queue<PSI, vector<PSI>, greater<PSI>> heap;
    heap.push((PSI){start, f(start), ""});

    dist[start] = 0;

    while (heap.size())
    {
        auto t = heap.top();
        heap.pop();

        string state = t.s, path = t.path;

        if (state == end)  return path;

        int step = dist[state];
        int x, y;
        for (int i = 0; i < state.size(); i ++ )
            if (state[i] == 'x')
```

```
            {
                x = i / 3, y = i % 3;
                break;
            }

        for (int i = 0; i < 4; i ++ )
        {
            int a = x + dx[i], b = y + dy[i];
            if (a >= 0 && a < 3 && b >= 0 && b < 3)
            {
                swap(state[x * 3 + y], state[a * 3 + b]);
                if (!dist.count(state) || dist[state] > step + 1)
                {
                    dist[state] = step + 1;
                    heap.push((PSI){state, dist[state] + f(state), path +
op[i]});
                }
                swap(state[x * 3 + y], state[a * 3 + b]);
            }
        }
    }

    return "unsolvable";
}

int main()
{
    string start, str;
    for(int i = 0; i < 9; i ++)
    {
        cin >> str;
        start += str[0];
    }
    int t = 0;
    for(int i = 0; i < start.size(); i ++)
    {
        if(start[i] == 'x')  continue;
        for(int j = i + 1; j < start.size(); j ++)
            if(start[i] > start[j])  t ++;
    }
    if(t % 2)  cout << "unsolvable";
    else  cout << bfs(start);
    return 0;
}
```

5.IDA*

5.1排书

给定n本书，编号为1-n。

在初始状态下，书是任意排列的。

在每一次操作中，可以抽取其中连续的一段，再把这段插入到其他某个位置。

我们的目标状态是把书按照1-n的顺序依次排列。

求最少需要多少次操作。

```cpp
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 15;

int n, a[N];
int w[5][N];

int f()
{
    int tot = 0;
    for(int i = 0; i + 1 < n; i ++)
        if(a[i + 1] != a[i] + 1)  tot ++;
    return (tot + 2) / 3;
}

bool dfs(int num, int depth)
{
    int val = f();
    if(num + val > depth)  return false;
    if(!val)  return true;

    for(int len = 1; len < n; len ++)
        for(int l = 0; l + len - 1 < n; l ++)
        {
            int r = l + len - 1;
            for(int k = r + 1; k < n; k ++)
            {
                memcpy(w[num], a, sizeof a);
                int i, j;
                for(i = l, j = r + 1; j <= k; i ++, j ++)  a[i] = w[num][j];
                for(j = l; j <= r; i ++, j ++)  a[i] = w[num][j];

                //for(int i = l, j = k; i <= r; i ++, j ++)  swap(a[i], a[j]);
                if(dfs(num + 1, depth))  return true;
                memcpy(a, w[num], sizeof a);
            }
        }
    return false;
}

int main()
{
    int t;
    scanf("%d", &t);
    while(t --)
    {
        scanf("%d", &n);
        for(int i = 0; i < n; i ++)  scanf("%d", &a[i]);

        int depth = 0;
        while(depth < 5 && !dfs(0, depth))  depth ++;
        if(depth >= 5)  printf("5 or more\n");
```

```
        else  printf("%d\n", depth);
    }
    return 0;
}
```

5.2回转游戏

如下图所示，有一个“#”形的棋盘，上面有1,2,3三种数字各8个。

给定8种操作，分别为图中的A~H。

这些操作会按照图中字母和箭头所指明的方向，把一条长为8的序列循环移动1个单位。

例如下图最左边的“#”形棋盘执行操作A后，会变为下图中间的“#”形棋盘，再执行操作C后会变成下图最右边的“#”形棋盘。

给定一个初始状态，请使用最少的操作次数，使“#”形棋盘最中间的8个格子里的数字相同。

```
/*        A0    B1

          0     1
          2     3
H7  4  5  6  7  8  9  10  C2
          11    12
G6  13 14 15 16 17 18 19  D3
          20    21
          22    23

          F5    E4          */
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 24;

int array[8][7] = {
    0, 2, 6, 11, 15, 20, 22,        //A
    1, 3, 8, 12, 17, 21, 23,        //B
    10, 9, 8, 7, 6, 5, 4,           //C
    19, 18, 17, 16, 15, 14, 13      //D
};

int a[N], path[1024];

void move(int k)
{
    int t = a[array[k][0]];
    for(int i = 0; i < 6; i ++)  a[array[k][i]] = a[array[k][i + 1]];
    a[array[k][6]] = t;
}

int center[8] = {6, 7, 8, 11, 12, 15, 16, 17};
int f()
{
    int sum[4] = {0};
```

```
    for(int i = 0; i < 8; i ++)
        sum[a[center[i]]] ++;

    int s = 0;
    for(int i = 1; i <= 3; i ++)  s = max(s, sum[i]);

    return 8 - s;
}

bool check()
{
    for(int i = 1; i < 8; i ++)
        if(a[center[i]] != a[center[0]])  return false;
    return true;
}

int back[8] = {5, 4, 7, 6, 1, 0, 3, 2};     //back[i]是i的反操作
bool dfs(int num, int depth, int last)
{
    if(check())  return true;
    if(num + f() > depth)  return false;

    for(int i = 0; i < 8; i ++)
    {
        if(back[i] == last)  continue;      //如果是上一步的反操作跳过

        move(i);
        path[num] = i;
        if(dfs(num + 1, depth, i))  return true;
        move(back[i]);
    }
    return false;
}

int main()
{

    for(int i = 4; i < 8; i ++)
        for(int j = 0; j < 7; j ++)
            array[i][j] = array[back[i]][6 - j];

    while(scanf("%d", &a[0]), a[0])
    {
        for(int i = 1; i < N; i ++)  scanf("%d", &a[i]);

        int depth = 0;
        while(!dfs(0, depth, -1))  depth ++;

        if(!depth)  puts("No moves needed");
        else{
            for(int i = 0; i < depth; i ++)  printf("%c", 'A' + path[i]);
            printf("\n");
        }

        printf("%d\n", a[6]);
    }
    return 0;
}
```

五、动态规划

1.最长上升子序列

1.1最长上升子序列

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 100010;

int a[N], n;
int g[N], len;

int main()
{
    scanf("%d", &n);

    g[0] = -2e9;
    for(int i = 1; i <= n; i ++)
    {
        scanf("%d", &a[i]);

        int l = 0, r = len;
        while(l < r)
        {
            int mid = l + r + 1 >> 1;
            if(g[mid] < a[i])  l = mid;
            else  r = mid - 1;
        }
        if(l + 1 > len)  len ++;
        g[l + 1] = a[i];
    }
    printf("%d\n", len);
    return 0;
}
```

1.2拦截导弹

求最长上升子序列、最少上升子序列覆盖

```
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 100010;

int a[N];
int f[N];       //  以i结尾的最长不升子序列
```

```
int g[N];       //  第j个最长不升子序列的末尾的值

int main()
{
    int n = 0;
    while(~scanf("%d", &a[n]))  n ++;

    int res = 0, cnt = 0;
    for(int i = 0; i < n; i ++)
    {
        f[i] = 1;
        for(int j = 0; j < i; j ++)
            if(a[j] >= a[i])  f[i] = max(f[i], f[j] + 1);
        res = max(res, f[i]);

        int j = 0;
        while(a[i] > g[j] and j < cnt)  j ++;
        g[j] = a[i];
        if(j == cnt)  cnt ++;
    }
    printf("%d\n%d", res, cnt);
    return 0;
}
```

1.3最长公共上升子序列

对于两个数列A和B，如果它们都包含一段位置不一定连续的数，且数值是严格递增的，那么称这一段数是两个数列的公共上升子序列，而所有的公共上升子序列中最长的就是最长公共上升子序列

```
#include<cstdio>
#include<algorithm>

using namespace std;

const int N = 3010;

int a[N], b[N], f[N][N];        //  f[i][j]为a[1~i]和b[1~j]中以b[j]结尾的最长公共上升子序列的长度

int main()
{
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n; i ++)  scanf("%d", &a[i]);
    for(int i = 1; i <= n; i ++)  scanf("%d", &b[i]);

    for(int i = 1; i <= n; i ++)
    {
        int maxv = 1;
        for(int j = 1; j <= n; j ++)
        {
            f[i][j] = f[i - 1][j];
            if(a[i] == b[j])  f[i][j] = max(f[i][j], maxv);
            if(a[i] > b[j])  maxv = max(maxv, f[i - 1][j] + 1);
        }
    }
```

```
    int res = 0;
    for(int i = 1; i <= n; i ++)  res = max(res, f[n][i]);
    printf("%d\n", res);
    return 0;
}
```

2.单调队列优化DP

2.1多重背包3

```
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 1010, M = 20010;

int v[N], w[N], s[N];
int f[N][M];
int q[M];

int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i ++)  scanf("%d%d%d", &v[i], &w[i], &s[i]);

    for(int i = 1; i <= n; i ++)
    {
        scanf("%d%d%d", &v[i], &w[i], &s[i]);
        for(int k = 0; k < v[i]; k ++)
        {
            int hh = 0, tt = -1;
            for(int j = k; j <= m; j += v[i])
            {
                if(hh <= tt and q[hh] + v[i] * s[i] < j)  hh ++;
                f[i][j] = f[i - 1][j];
                if(hh <= tt)  f[i][j] = max(f[i][j], f[i - 1][q[hh]] + (j -
q[hh]) / v[i] * w[i]);
                while(hh <= tt and f[i - 1][q[tt]] - q[tt] / v[i] * w[i] <= f[i
- 1][j] - j / v[i] * w[i])  tt --;
                q[++ tt] = j;
            }
        }
    }
    printf("%d", f[n][m]);
    return 0;
}
```

2.2理想的正方形

有一个 $a \times b$ 的整数组成的矩阵，现请你从中找出一个 $n \times n$ 的正方形区域，使得该区域所有数中的最大值和最小值的差最小。

```cpp
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1010, INF = 1e9;

int n, m, k;
int w[N][N];
int row_min[N][N], row_max[N][N];
int q[N];

void get_min(int a[], int b[], int tot)
{
    int hh = 0, tt = -1;
    for (int i = 1; i <= tot; i ++ )
    {
        if (hh <= tt && q[hh] <= i - k) hh ++ ;
        while (hh <= tt && a[q[tt]] >= a[i]) tt -- ;
        q[ ++ tt] = i;
        b[i] = a[q[hh]];
    }
}

void get_max(int a[], int b[], int tot)
{
    int hh = 0, tt = -1;
    for (int i = 1; i <= tot; i ++ )
    {
        if (hh <= tt && q[hh] <= i - k) hh ++ ;
        while (hh <= tt && a[q[tt]] <= a[i]) tt -- ;
        q[ ++ tt] = i;
        b[i] = a[q[hh]];
    }
}

int main()
{
    scanf("%d%d%d", &n, &m, &k);

    for (int i = 1; i <= n; i ++ )
        for (int j = 1; j <= m; j ++ )
            scanf("%d", &w[i][j]);

    for (int i = 1; i <= n; i ++ )
    {
        get_min(w[i], row_min[i], m);
        get_max(w[i], row_max[i], m);
    }

    int res = INF;
    int a[N], b[N], c[N];
    for (int i = k; i <= m; i ++ )
    {
        for (int j = 1; j <= n; j ++ ) a[j] = row_min[j][i];
        get_min(a, b, n);
```

```
        for (int j = 1; j <= n; j ++ ) a[j] = row_max[j][i];
        get_max(a, c, n);

        for (int j = k; j <= n; j ++ ) res = min(res, c[j] - b[j]);
    }
    printf("%d\n", res);
    return 0;
}
```

3.状态压缩DP

3.1小国王

在 $n \times n$ 的棋盘上放 k 个国王，国王可以攻击相邻的 8 个格子， 求方案数

```
#include<cstdio>
#include<vector>

using namespace std;

typedef long long LL;

const int N = 12, M = 1 << 10, K = 110;

int n, m;
vector<int> state;      //  存放合法状态
int cnt[M];             //  存放每种状态中的国王数
vector<int> head[M];    //  head[i][j]表示能从状态i转移到j的状态
LL f[N][K][M];          //  集合 - 前i行已经摆完，并且使用了j个国王，然后最后一行的状态是s的方案集合

bool check(int state)
{
    for (int i = 0; i < n; i++)
    {
        if((state >> i & 1) && (state >> i + 1 & 1))    //  相邻两位不能为1
        {
            return false;
        }
    }
    return true;
}

int count(int x)
{
    int res = 0;
    while(x)  res ++, x -= x & -x;
    return res;
}

int main()
{
    scanf("%d %d", &n, &m);

    //  预处理合法方案
    for(int i = 0; i < 1 << n; i ++)
```

```
    {
        if(check(i))
        {
            state.push_back(i);
            cnt[i] = count(i);
        }
    }

    //  在所有合法状态中找到满足状态b->a的,存入g数组中
    for (int i = 0; i < state.size(); i++)
    {
        for (int j = 0; j < state.size(); j++)
        {
            int a = state[i], b = state[j];
            if((a & b) == 0 && check(a | b))
            {
                head[a].push_back(b);
            }
        }
    }

    f[0][0][0] = 1;
    for (int i = 1; i <= n + 1; i ++)    //  枚举每一行, n + 1 有利于最后输出
    {
        for (int j = 0; j <= m; j ++)    //  枚举国王数
        {
            for(int k = 0; k < state.size(); k ++)  //  枚举上一行状态
            {
                int a = state[k];
                for (int l = 0; l < head[a].size(); l ++)   //  枚举当前行的状态
                {
                    int b = head[a][l];     //  a为上一行状态,b从head数组里面挑的,保证了状态a->状态b合法
                    if(j >= cnt[a])         //  如果国王总数都小于该状态的国王数,则不合法
                    {
                        f[i][j][b] += f[i - 1][j - cnt[a]][a];
                    }
                }
            }
        }
    }

    printf("%lld", f[n + 1][m][0]);
    return 0;
}
```

3.2蒙德里安的梦想

求把 $N * M$ 的棋盘分割成若干个 $1 * 2$ 的的长方形，有多少种方案。

```
#include<iostream>
#include<cstring>

using namespace std;

const int N = 12, M = 1 << N;
```

```
bool st[2][M];
long long f[N][M];

void load(int num, int n)
{
    for(int i = 0; i < M; i ++)
    {
        st[num][i] = true;

        int cnt = 0;
        for(int j = 0; j < n; j ++)
            if(i >> j & 1)
            {
                if(cnt & 1)  break;
                cnt = 0;
            }else
                cnt ++;
        if(cnt & 1)  st[num][i] = false;
    }
}

int main()
{
    load(0, N), load(1, N - 1);

    int n, m;
    while(scanf("%d%d", &n, &m), n | m)
    {
        if(n * m & 1)
        {
            printf("0\n");
            continue;
        }

        memset(f, 0, sizeof f);
        f[0][0] = 1;
        for(int i = 1; i <= m; i ++)
            for(int j = 0; j < 1 << n; j ++)
                for(int k = 0; k < 1 << n; k ++)
                    if(!(j & k) && st[n % 2][j | k])
                        f[i][j] += f[i - 1][k];

        printf("%lld\n", f[m][0]);
    }

    return 0;
}
```

4.区间DP

4.1棋盘分割

将一个 8 * 8 的棋盘进行如下分割：将原棋盘割下一块矩形棋盘并使剩下部分也是矩形，再将剩下的部分继续如此分割，这样割了(n-1)次后，连同最后剩下的矩形棋盘共有n块矩形棋盘。

```cpp
#include<cstring>
#include<algorithm>
#include<cstring>
#include<cmath>

using namespace std;

const int N = 15, M = 9;
const double INF = 4e18;

int n, m = 8;
int s[M][M];
double f[M][M][M][M][N];
double X;

double get(int x1, int y1, int x2, int y2)
{
    double sum = s[x2][y2] - s[x1 - 1][y2] - s[x2][y1 - 1] + s[x1 - 1][y1 - 1] -
X;
    return sum * sum / n;
}

double dfs(int x1, int y1, int x2, int y2, int k)
{
    double &v = f[x1][y1][x2][y2][k];
    if(v >= 0)  return v;
    if(k == 1)  return v = get(x1, y1, x2, y2);

    v = INF;
    for(int i = x1; i < x2; i ++)
    {
        v = min(v, dfs(x1, y1, i, y2, k - 1) + get(i + 1, y1, x2, y2));
        v = min(v, dfs(i + 1, y1, x2, y2, k - 1) + get(x1, y1, i, y2));
    }
    for(int i = y1; i < y2; i ++)
    {
        v = min(v, dfs(x1, y1, x2, i, k - 1) + get(x1, i + 1, x2, y2));
        v = min(v, dfs(x1, i + 1, x2, y2, k - 1) + get(x1, y1, x2, i));
    }
    return v;
}

int main()
{
    scanf("%d", &n);
    for(int i = 1; i <= m; i ++)
        for(int j = 1; j <= m; j ++)
        {
            scanf("%d", &s[i][j]);
            s[i][j] += s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1];
        }

    memset(f, -1, sizeof f);
    X = (double)s[m][m] / n;

    printf("%.3lf", sqrt(dfs(1, 1, 8, 8, n)));
    return 0;
}
```

5.树形DP

6.数位DP

6.1数位统计

给定两个整数 a 和 b，求 a 和 b 之间的所有数字中0~9的出现次数。

```
#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;

const int N = 10;

int get(vector<int> &nums, int l, int r)
{
    int res = 0;
    for(int i = l; i >= r; i --)  res = res * 10 + nums[i];
    return res;
}

int pow10(int n)
{
    int x = 1;
    while(n --)  x *= 10;
    return x;
}

int dp(int n, int x)
{
    if(!n)  return 0;

    vector<int> nums;
    while(n)  nums.push_back(n % 10),  n /= 10;

    n = nums.size();
    int res = 0;
    for(int i = n - 1 - !x; i >= 0; i --)
    {
        if(i < n - 1)
        {
            res += get(nums, n - 1, i + 1) * pow10(i);
            if(!x)  res -= pow10(i);
        }

        if(nums[i] == x)  res += get(nums, i - 1, 0) + 1;
        else if(nums[i] > x)  res += pow10(i);
    }
    return res;
}

int main()
{
```

```
    int l, r;
    while(scanf("%d%d", &l, &r),  l | r)
    {
        if(l > r)  swap(l, r);
        for(int i = 0; i <= 9; i ++)
            printf("%d ", dp(r, i) - dp(l - 1, i));
        printf("\n");
    }
    return 0;
}
```

6.2不降数

从高位到低位非递减的数的个数

```
#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;

const int N = 12;

int f[N][N];            //f[i][j]从j开始的长度为i的不降数序列个数

int dp(int n)
{
    if(!n)  return 1;

    vector<int> nums;
    while(n)  nums.push_back(n % 10),  n /= 10;

    int res = 0, last = 0;
    for(int i = nums.size() - 1; i >= 0; i --)
    {
        int x = nums[i];
        for(int j = last; j < x; j ++)  res += f[i + 1][j];

        if(x < last)  break;
        last = x;

        if(!i)  res ++;
    }
    return res;
}

int main()
{
    for(int i = 0; i <= 9; i ++)  f[1][i] = 1;
    for(int i = 2; i < N; i ++)
        for(int j = 9; j >= 0; j --)
            for(int k = j; k <= 9; k ++)
                f[i][j] += f[i - 1][k];

    int l, r;
    while(~scanf("%d%d", &l, &r))  printf("%d\n", dp(r) - dp(l - 1));
    return 0;
```

```
}
```

6.3数位之和能被n整除的数

求各位数字之和 mod N 为 0 的数的个数

```
#include<iostream>
#include<algorithm>
#include<cstring>
#include<vector>

using namespace std;

const int N = 15, M = 110;

int f[N][N][M];         //长度为i的以j开头的%p = k的有多少个

int p;

void init()
{
    memset(f, 0, sizeof f);

    for(int i = 0; i <= 9; i ++)  f[1][i][i % p] ++;
    for(int i = 2; i < N; i ++)
        for(int j = 0; j <= 9; j ++)
            for(int k = 0; k < p; k ++)
                for(int x = 0; x <= 9; x ++)
                    f[i][j][(k + j) % p] += f[i - 1][x][k];
}

int dp(int n)
{
    if(!n)  return 1;

    vector<int> nums;
    while(n)  nums.push_back(n % 10),  n /= 10;
    int res = 0, last = 0;
    for(int i = nums.size() - 1; i >= 0; i --)
    {
        int x = nums[i];
        for(int j = 0; j < x; j ++)
            for(int k = 0; k < p; k ++)
                if((last + k) % p == 0)
                    res += f[i + 1][j][k];
        last += x;
        if(!i && last % p == 0)  res ++;
    }
    return res;
}

int main()
{
    int l, r;
    while(~scanf("%d%d%d", &l, &r, &p))
    {
        init();
```

```
        printf("%d\n", dp(r) - dp(l - 1));
    }
    return 0;
}
```

6.4不要62

求不含4且不含62的数的个数

```
#include<iostream>
#include<algorithm>
#include<cstring>
#include<vector>

using namespace std;

const int N = 10;

int f[N][N];        //长度为i的，以j打头的合法方案数

int dp(int n)
{
    if(!n)  return 1;

    vector<int> nums;
    while(n)  nums.push_back(n % 10),  n /= 10;

    int res = 0, last = -1;
    for(int i = nums.size() - 1; i >= 0; i --)
    {
        int x = nums[i];

        for(int j = 0; j < x; j ++)
        {
            if(last == 6 && j == 2)  continue;
            res += f[i + 1][j];
        }

        if(x == 4 || last == 6 && x == 2)  break;
        last = x;

        if(!i)  res ++;
    }
    return res;
}

int main()
{
    for(int i = 0; i <= 9; i ++)
        if(i != 4)  f[1][i] = 1;
    for(int i = 2; i < N; i ++)
        for(int j = 0; j <= 9; j ++)
        {
            if(j == 4)  continue;
            for(int k = 0; k <= 9; k ++)
            {
                if(k == 4 || j == 6 && k == 2)  continue;
```

```
                f[i][j] += f[i - 1][k];
            }
        }

    int l, r;
    while(scanf("%d%d", &l, &r), l | r)
    {
        printf("%d\n", dp(r) - dp(l - 1));
    }
    return 0;
}
```

7.斜率优化DP

8.约瑟夫环

```
f[1] = 0;
for(int i = 2; i <= n; i ++)
    f[i] = (f[i - 1] + m) % i;
printf("%d\n", f[n] + 1);
```

六、杂项

1.对拍

- 一个数据生成器代码 `data.cpp` 写好之后编译成 `data.exe`
- 一个自己写的代码 `my.cpp` 编译成 `my.cpp`
- 一个保证正确性的代码 `std.cpp` 编译成 `std.exe`
- 然后再如下写一个 `对拍.txt` 修改后缀为 `对拍.bat`

```
@echo off
:loop
    data.exe>data.in
    my.exe<data.in>my.out
    std.exe<data.in>std.out
    fc my.out std.out
if not errorlevel 1 goto loop
pause
goto loop
```

2.高精度

2.1高精度加法

```
#include<iostream>
#include<algorithm>
#include<vector>
```

```
using namespace std;

vector<int> add(vector<int> &a, vector<int> &b)
{
    if(a.size() < b.size())  return add(b, a);

    vector<int> res;
    int t = 0, i;
    for(i = 0; i < b.size(); i ++)
    {
        res.push_back((a[i] + b[i] + t) % 10);
        t = (a[i] + b[i] + t) / 10;
    }
    for(; i < a.size(); i ++)
    {
        res.push_back((a[i] + t) % 10);
        t = (a[i] + t) / 10;
    }
    if(t)  res.push_back(t);
    return res;
}

int main()
{
    string a, b;
    cin >> a >> b;

    vector<int> A, B;
    for(int i = a.size() - 1; i >= 0; i --)  A.push_back(a[i] - '0');
    for(int i = b.size() - 1; i >= 0; i --)  B.push_back(b[i] - '0');

    vector<int> C = add(A, B);

    for(int i = C.size() - 1; i >= 0; i --)  printf("%d", C[i]);
    return 0;
}
```

2.2高精度减法

```
#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;

bool upper(string &a, string &b)
{
    if(a.size() > b.size())  return true;
    else if(a.size() < b.size())  return false;

    for(int i = 0; i < a.size(); i ++)
        if(a[i] > b[i])  return true;
        else if(a[i] < b[i])  return false;
    return true;
}
```

```
vector<int> sub(vector<int> a, vector<int> &b)
{
    vector<int> res;
    int t = 0, i;
    for(i = 0; i < b.size(); i ++)
    {
        if(t)  a[i] --;
        if(a[i] < b[i])  a[i] += 10,  t = 1;
        else  t = 0;
        res.push_back(a[i] - b[i]);
    }
    for(; i < a.size(); i ++)
    {
        if(t)  a[i] --;
        if(a[i] < 0)  a[i] += 10,  t = 1;
        else  t = 0;
        res.push_back(a[i]);
    }
    while(res.size() > 1 && res[res.size() - 1] == 0)  res.pop_back();
    return res;
}

int main()
{
    string a, b;
    cin >> a >> b;
    vector<int> A, B, res;

    for(int i = a.size() - 1; i >= 0; i --)  A.push_back(a[i] - '0');
    for(int i = b.size() - 1; i >= 0; i --)  B.push_back(b[i] - '0');
    if(upper(a, b))  res = sub(A, B);
    else  res = sub(B, A),  printf("-");

    for(int i = res.size() - 1; i >= 0; i --)  printf("%d", res[i]);
    return 0;
}
```

2.3高精度乘法

```
#include<iostream>
#include<vector>

using namespace std;

vector<int> mul(vector<int> &a, int b)
{
    vector<int> res;
    int t = 0;
    for(int i = 0; i < a.size(); i ++)
    {
        t += a[i] * b;
        res.push_back(t % 10);
        t /= 10;
    }
    while(t)  res.push_back(t % 10),  t /= 10;
    while(res.size() > 1 and res.back() == 0)  res.pop_back();
    return res;
```

```
}

int main()
{
    string a;
    int b;
    cin >> a >> b;
    vector<int> A, res;
    for(int i = a.size() - 1; i >= 0; i --)  A.push_back(a[i] - '0');
    res = mul(A, b);
    for(int i = res.size() - 1; i >= 0; i --)  printf("%d", res[i]);
    return 0;
}
```

2.4高精度除法

```
#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;

int div(vector<int> &a, int b, vector<int> &res)
{
    int r = 0;
    for(int i = a.size() - 1; i >= 0; i --)
    {
        r = r * 10 + a[i];
        res.push_back(r / b);
        r %= b;
    }
    reverse(res.begin(), res.end());

    while(res.size() > 1 && res.back() == 0)  res.pop_back();
    return r;
}

int main()
{
    string a;
    int b;

    cin >> a >> b;
    vector<int> A, res;

    for(int i = a.size() - 1; i >= 0; i --)  A.push_back(a[i] - '0');
    int r = div(A, b, res);

    for(int i = res.size() - 1; i >= 0; i --)  printf("%d", res[i]);
    printf("\n%d", r);
    return 0;
}
```

3.快读

```
int read()
{
    int x=0,f=1;
    char c=getchar();
    while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}
    while(c>='0'&&c<='9'){x=x*10+c-'0';c=getchar();}
    return f*x;
}
```

4.指令优化

```
#pragma GCC optimize(2)
ios::sync_with_stdio(false);
#pragma comment(linker, "/STACK:1024000000,1024000000")
```

5.矩阵

5.1矩阵旋转

```
void revolve()
{
    static int t[N][N];
    for(int i = 1; i <= n; i ++)
        for(int j = 1; j <= n; j ++)
            t[i][j] = a[n - j + 1][i];

    for(int i = 1; i <= n; i ++)
        for(int j = 1; j <= n; j ++)
            a[i][j] = t[i][j];
}
```

5.2蛇形矩阵

```
#include<cstdio>

const int N = 110;

int s[N][N];

int dx[4] = {0, 1, 0, -1};
int dy[4] = {1, 0, -1, 0};
int main()
{
    int n, m;
    scanf("%d%d", &n, &m);

    int x = 1, y = 1, d = 0;

    for(int i = 1; i <= n * m; i ++)
    {
        s[x][y] = i;

        int a = x + dx[d], b = y + dy[d];
```

```
        if(a > n or a < 1 or b > m or b < 1 or s[a][b])  d = (d + 1) % 4;
        x += dx[d];
        y += dy[d];
    }

    for(int i = 1; i <= n; i ++)
    {
        for(int j = 1; j <= m; j ++)
            printf("%d ", s[i][j]);
        puts("");
    }
}
```

5.3菱形前缀和

```
int get(int x, int y, int r)
{
    if(r <= 0)  return 0;
    return  sum[x + r - 1][y] - sum[x - 1][y - r + 1] - left[x - 1][y - r] -
            sum[x - 1][y + r - 1] - right[x - 1][y + r] + sum[x - r][y];
}

sum[i][j] = sum[i - 1][j] + left[i - 1][j - 1] + right[i - 1][j + 1] + c;
left[i][j] = left[i - 1][j - 1] + c;
right[i][j] = right[i - 1][j + 1] + c;
```

6.01分数规划

题意：每个元素有两个属性 a, b 选取至少 m 个元素使得 $\frac{\sum a}{\sum b}$ 最大

二分答案，判断是否存在 $\frac{\sum a}{\sum b} > mid$

$\Longrightarrow \sum a > \sum b \cdot mid$

$\Longrightarrow \sum a - \sum b \cdot mid > 0$

$\Longrightarrow \sum a - b \cdot mid > 0$

最优比率环

点权值和除以边权之和最大的环

```
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 1010, M = 5010;

int h[N], e[M], t[M], ne[M], idx;
void add(int a, int b, int c)
{
    e[idx] = b, t[idx] = c, ne[idx] = h[a], h[a] = idx ++;
}
```

```
int n, m;
int f[N];
double d[N];
int q[N], cnt[N];
bool st[N];
bool check(double mid)
{
    int hh = 0, tt = 0;
    for(int i = 1; i <= n; i ++)
    {
        d[i] = 0;
        q[tt ++ ] = i;
        st[i] = true;
        cnt[i] = 1;
    }

    while(hh != tt)
    {
        int u = q[hh ++];
        if(hh == N)  hh = 0;
        st[u] = false;

        for(int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i];
            double dist = d[u] + f[u] - t[i] * mid;
            if(dist > d[j])
            {
                d[j] = dist;
                cnt[j] = cnt[u] + 1;
                if(cnt[u] > n)  return true;
                if(!st[j])
                {
                    q[tt ++] = j;
                    if(tt == N)  tt = 0;
                    st[j] = true;
                }
            }
        }
    }
    return false;
}

int main()
{
    memset(h, -1, sizeof h);

    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i ++)  scanf("%d", &f[i]);

    while(m --)
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        add(a, b, c);
    }
```

```
    double l = 1, r = 1000;
    while(r - l > 1e-4)
    {
        double mid = (l + r) / 2;
        if(check(mid))  l = mid;
        else  r = mid;
    }
    printf("%.2lf\n", l);
    return 0;
}
```