[acm 模板]

V2.0.0

作者:[qwb]

2017年4月14日

1.强连通分量有向图缩点

```
const int MX = 5e3 + 5;
const int INF = 0x3f3f3f3f;
struct Edge {
    int u, v, nxt;
} E[60005];
int Head[MX], erear;
void edge_init() {
    erear = 0;
    memset(Head, -1, sizeof(Head));
}
void edge_add(int u, int v) {
    E[erear].u = u;
    E[erear].v = v;
    E[erear].nxt = Head[u];
    Head[u] = erear++;
}
int n, m, IN[MX], cnt[MX], val[MX];
int bsz, ssz, dsz;
int Low[MX], DFN[MX];
int belong[MX], Stack[MX];
bool inStack[MX];
void Init_tarjan(int n) {
    bsz = ssz = dsz = 0;
    for(int i = 1; i <= n; ++i) Low[i] = DFN[i] = 0;
}
void Tarjan(int u) {
    Stack[++ssz] = u;
    inStack[u] = 1;
    Low[u] = DFN[u] = ++dsz;
    for(int i = Head[u]; ~i; i = E[i].nxt) {
        int v = E[i].v;
        if(!DFN[v]) {
            Tarjan(v);
            Low[u] = min( Low[v], Low[u]);
        } else if(inStack[v]) {
            Low[u] = min( Low[u], DFN[v]);
        }
    }
    if(Low[u] == DFN[u]) {
        ++bsz;
        int v;
        do {
            v = Stack[ssz--];
            inStack[v] = 0;
            belong[v] = bsz;
        } while(u != v);
    }
}
int solve(int n) {
    Init_tarjan(n);
    for (int i = 1; i <= n; i++) {
        if (!DFN[i]) Tarjan(i);
    }
    edge_init();
    for(int i = 0; i < m; i++) {
        int u = E[i].u, v = E[i].v;
        u = belong[u]; v = belong[v];
        if(u != v) {
```

```
            edge_add(u, v);
        }
    }
}
```

2.强连通分量无向图缩点

```
int DFN[MX], Low[MX], dsz, tim;
int Stack[MX], inStack[MX], Belong[MX], bsz, ssz;

void trajan(int u, int e) {
    inStack[u] = 1;
    Stack[++ssz] = u;
    DFN[u] = Low[u] = ++dsz;
    for(int i = Head[u]; ~i; i = E[i].nxt) {
        int v = E[i].v;
        if((i ^ 1) == e) continue;
        if(!DFN[v]) {
            trajan(v, i);
            Low[u] = min(Low[u], Low[v]);
        } else if(inStack[v]) {
            Low[u] = min(Low[u], Low[v]);
        }
    }
    if(DFN[u] == Low[u]) {
        bsz++; int v;
        do {
            v = Stack[ssz--];
            inStack[v] = 0;
            Belong[v] = bsz;
        } while(ssz && v != u);
    }
}
void tarjan_solve(int n) {
    dsz = bsz = ssz = 0;
    memset(DFN, 0, sizeof(DFN));
    for(int i = 1; i <= n; i++) {
        if(!DFN[i]) trajan(i, -1);
    }
    /*缩点*/
    edge_init();
    for(int i = 0; i < 2 * m; i += 2) {
        int u = E[i].u, v = E[i].v;
        u = Belong[u]; v = Belong[v];
        if(u == v) continue;
        edge_add(u, v);
        edge_add(v, u);
    }
}
```

3.凸多边形平面图转对偶图

```
/*
1->2->3->...n->1
然后在上面加了 m 条边
把这种图转换成对偶图
区域的个数就是 dsz
做模板的时候忘记初始化了，多组输入注意一下
*/
```

```
int n, m;
vector<int> point[MX];
map<pii, bool> vis;
map<pii, int> id;
int cur[MX], dsz;
int who[MX][2];
struct Data {
    int color;
    vector<int> num;
} data[MX];

int get_id(int u, int v) {
    if(u > v) swap(u, v);
    if(id.count(pii(u, v))) return id[pii(u, v)];
    return -1;
}
int get_next_pos(int pre, int u, int s) {
    if(pre != s) {//上一条边如果是 s，不能往回走的
        //看 u 是否有边直接连到了起点，有的话就要回起点
        auto p = lower_bound(point[u].begin(), point[u].end(), s);
        if(p != point[u].end() && *p == s) {
            return s;
        }
    }
    while(cur[u] >= 0) {
        int v = point[u][cur[u]];
        if(vis.count(pii(u, v))) {
            cur[u]--;
        }
        break;
    }
    // assert(cur[u] >= 0);
    return point[u][cur[u]];
}
void presolve() {
    for(int i = 1; i <= m; i++) {
        int u, v;//点的下标都减了 1
        scanf("%d%d", &u, &v); u--; v--;
        point[u].push_back(v);
        point[v].push_back(u);
        if(u > v) swap(u, v);
        id[pii(u, v)] = i;
    }
    for(int i = 0; i < n; i++) {
        point[i].push_back((i + 1) % n);
        point[i].push_back((i - 1 + n) % n);
        sort(point[i].begin(), point[i].end());
    }
    for(int i = 0; i < n; i++) {
        cur[i] = point[i].size() - 1;
    }

    cur[0]--;//减去 n-1 的
    for(int s = 0; s < n - 1; s++) {
        while(cur[s] >= 0) {
            int u = s, v = point[s][cur[s]];
            if(vis.count(pii(s, v))) {
                cur[s]--; continue;
            }

            dsz++;//这个时候一定会有区域
            int last = -1;
```

```
            while(true) {
                int v = get_next_pos(last, u, s);
                int id = get_id(u, v);//确定这条边是否是后来加的
                if(id == -1) {
                    vis[pii(u, v)] = vis[pii(v, u)] = 1;
                } else {
                    vis[pii(u, v)] = 1;
                    who[id][who[id][0] ? 1 : 0] = dsz;
                }
                data[dsz].num.push_back(v);
                if(v == s) break;
                last = u; u = v;
            }
        }
    }
}
```

4.差分约束

B-A<=C 转换成 A->B 的边权值为 C
求 B-A 最大值转换为求 A->B 最短路
求 B-A 最小值转换为求 B->A 最短路并取负号
如果存在负环，则无解
如果不存在最短路，则无数解

5.点分治

```
/*
记得打上 vis 标记。
点分治常用思路：
1.先找到重心
2.维护重心为根的树
3.若答案以重心为端点时的答案
4.若答案经过重心时的答案
5.上述两种情况，要注意两个端点不能同时属于一颗子树
6.递归
*/
int sum[MX], root, rtsum;
void Tree_G(int u) {
    int mx = 0;
    sum[u] = 1;
    for (int i = Head[u]; ~i; i = E[i].nxt) {
        int v = E[i].v;
        if (vis[v]) continue;
        Tree_G(v);
        sum[u] += sum[v];
        mx = max(mx, sum[v]);
    }
    mx = max(mx, sum[0] - sum[u]);
    if (mx < rtsum) rtsum = mx, root = u;
    vis[u] = 0;
}
```

6.匈牙利匹配

```
/*复杂度 O(VE)
最小点覆盖=最大匹配数
最小边覆盖=左右点数-最大匹配数
最小路径覆盖=点数-最大匹配数
```

```
最大独立集=点数-最大匹配数
*/
int match[MX];
bool vis[MX];
bool DFS(int u) {
    for(int i = Head[u]; ~i; i = E[i].nxt) {
        int v = E[i].v;
        if(!vis[v]) {
            vis[v] = 1;
            if(match[v] == -1 || DFS(match[v])) {
                match[v] = u;
                return 1;
            }
        }
    }
    return 0;
}
int BM(int n) {
    int res = 0;
    memset(match, -1, sizeof(match));
    for(int u = 1; u <= n; u++) {
        memset(vis, 0, sizeof(vis));
        if(DFS(u)) res++;
    }
    return res;
}
```

7.KM 二分图带权匹配

```
const int MX = 3e2 + 5;
/*  KM 算法
 *   复杂度 O(nx*nx*ny)
 *  求最大权匹配
 *   若求最小权匹配，可将权值取相反数，结果取相反数
 *  点的编号从 1 开始
 */
const int INF = 0x3f3f3f3f;
int nx, ny; //两边的点数
int G[MX][MX];//二分图描述
int linker[MX], lx[MX], ly[MX]; //y 中各点匹配状态，x,y 中的点标号
int slack[MX];
bool visx[MX], visy[MX];

bool DFS(int x) {
    visx[x] = 1;
    for(int y = 1; y <= ny; y++) {
        if(visy[y]) continue;
        int tmp = lx[x] + ly[y] - G[x][y];
        if(tmp == 0) {
            visy[y] = 1;
            if(linker[y] == -1 || DFS(linker[y])) {
                linker[y] = x;
                return 1;
            }
        } else if(slack[y] > tmp) {
            slack[y] = tmp;
        }
    }
    return 0;
}
int KM() {
    memset(linker, -1, sizeof(linker));
    memset(ly, 0, sizeof(ly));
```

```
    for(int i = 1; i <= nx; i++) {
        lx[i] = -INF;
        for(int j = 1; j <= ny; j++) {
            if(G[i][j] > lx[i]) lx[i] = G[i][j];
        }
    }
    for(int x = 1; x <= nx; x++) {
        for(int i = 1; i <= ny; i++) slack[i] = INF;
        while(true) {
            memset(visx, 0, sizeof(visx));
            memset(visy, 0, sizeof(visy));
            if(DFS(x)) break;
            int d = INF;
            for(int i = 1; i <= ny; i++) {
                if(!visy[i] && d > slack[i]) d = slack[i];
            }
            for(int i = 1; i <= nx; i++) {
                if(visx[i]) lx[i] -= d;
            }
            for(int i = 1; i <= ny; i++) {
                if(visy[i]) ly[i] += d;
                else slack[i] -= d;
            }
        }
    }
    int res = 0;
    for(int i = 1; i <= ny; i++) {
        if(linker[i] != -1) res += G[linker[i]][i];
    }
    return res;
}
```

8.欧拉回路 Fleury

```
/*删边要注意复杂度，尽量别用标记删除，而是直接删除
无向图满足欧拉回路：度为偶数，或者度为奇数的点个数为 2
有向图满足欧拉回路：入度全部等于出度，或者 1 个点入度-出度=1，一个点出度-入度=1，其他点入度等于出度
*/
void Fleury(int u) {
    for(int i = Head[u]; ~i; i = Head[u]) {
        Head[u] = E[i].nxt;
        if(!vis[i | 1]) {
            int v = E[i].v;
            vis[i | 1] = 1;
            Fleury(v);
        }
    }
    Path[++r] = u;
}
```

9.删 2 条边不连通

先搞一颗生成树，给每条非树边随机 hash
对于每条树边 hash 值=所有经过他的非树边的 hash 值 xor 和
如果存在两条边的 hash 值相等，则不连通

10.树中最长路

```
int solve(int u, int from, int &ans) {
    int Max1 = 0, Max2 = 0;
    for(int id = Head[u]; ~id; id = Next[id]) {
        int v = E[id].v;
        if(v == from) continue;
```

```
        int t = solve(v, u, ans) + 1;

        if(t > Max1) {
            Max2 = Max1;
            Max1 = t;
        } else if(t > Max2) Max2 = t;
    }

    ans = max(ans, Max1 + Max2);
    return Max1;
}
/*调用方法
int ans = 0;
solve(1, -1, ans);
*/
```

11.2sat

```
struct Edge {
    int v, nxt;
} E[MX << 1];
int Head[MX][2], erear;
void edge_init() {
    erear = 0;
    memset(Head, -1, sizeof(Head));
}
void edge_add(int z, int u, int v) {
    E[erear].v = v;
    E[erear].nxt = Head[u][z];
    Head[u][z] = erear++;
}
void edge_add(int u, int v) {
    edge_add(0, u, v);
    edge_add(1, v, u);
}
int Stack[MX], Belong[MX], vis[MX], ssz, bsz;
void DFS(int u, int s) {
    vis[u] = 1;
    if(s) Belong[u] = s;
    for(int i = Head[u][s > 0]; ~i; i = E[i].nxt) {
        int v = E[i].v;
        if(!vis[v]) DFS(v, s);
    }
    if(!s) Stack[++ssz] = u;
}
/*得到的 Belong 的拓扑序
把 u 拆成 2 个点，分别表示真和假
如果 a 和-a 在同一个连通分量里则无解
如果 Belong[a]>Belong[-a]，则 a 为 true
如果 Belong[-a]>Belong[a]，则为 false
tarjan 的 2sat 时，记得是 2 倍点数，切记 MX
A,B 不能同时取 <A,B'><B,A'>
A,B 必须取一个<A',B><B',A>
A,B 必须都取或者都不取 <A,B><B,A><A',B'><B',A'>
必须取 A <A',A>
*/
void tarjan(int n) {
    ssz = bsz = 0;
    for(int i = 1; i <= n; i++) vis[i] = 0;
    for(int i = 1; i <= n; i++) {
        if(!vis[i]) DFS(i, 0);
    }
    for(int i = 1; i <= n; i++) vis[i] = 0;
```

```
    for(int i = ssz; i >= 1; i--) {
        if(!vis[Stack[i]]) DFS(Stack[i], ++bsz);
    }
}
```

12.O(1)lca

```
struct ST_LCA {
    int loo[MX * 2];
    int first[MX], dfn_to_id[MX], dfn;
    int ST[MX * 2][20], st_len;
    int dist[MX];

    void presolve() {
        dfn = st_len = 0;
        loo[1] = 0; loo[2] = 1;
        for(int i = 3; i < MX * 2; i++) {
            loo[i] = loo[i / 2] + 1;
        }
    }
    void DFS(int u, int f, int d) {
        int now = ++dfn;

        dist[u] = d;
        dfn_to_id[now] = u;
        ST[++st_len][0] = now;
        first[u] = st_len;

        for(int i = Head[u]; ~i; i = E[i].nxt) {
            int v = E[i].v;
            if(v == f) continue;
            DFS(v, u, d + 1);
            ST[++st_len][0] = now;
        }
    }
    void MT_presolve() {
        DFS(1, -1, 0);
        for(int i = 1; (1 << i) <= st_len; i++) {
            for(int j = 1; j + (1 << i) - 1 <= st_len; j++) {
                ST[j][i] = min(ST[j][i - 1], ST[j + (1 << (i - 1))][i - 1]);
            }
        }
    }
    int query(int u, int v) {
        int l = first[u], r = first[v];
        if(l > r) swap(l, r);
        int i = loo[r - l + 1];
        return dfn_to_id[min(ST[l][i], ST[r - (1 << i) + 1][i])];
    }
    int distance(int u, int v) {
        int lca = query(u, v);
        return dist[u] + dist[v] - 2 * dist[lca];
    }
} lca;
```

13.lca 离线

```
const int MQ = 40000 + 5;
const int MX = 80000 + 5;

struct Edge {
    int v, d;
    Edge(int _v, int _d) {
        v = _v; d = _d;
```

```
    }
};
struct Que {
    int id, u, v;
    Que() {}
    Que(int _u, int _v, int _id) {
        u = _u; v = _v; id = _id;
    }
} A[MQ];

int D[MX];
struct LCA {
    int n, ans[MQ];//答案按照id保存在ans中
    int P[MX]; bool vis[MX];

    vector<Edge>E[MX];
    vector<Que>Q[MQ];

    void Init(int _n) {
        n = _n;
        memset(ans, -1, sizeof(ans));
        memset(vis, false, sizeof(vis));
        for(int i = 1; i <= n; i++) {
            E[i].clear();
            Q[i].clear();
            P[i] = i;
        }
    }

    void AddQue(int u, int v, int id) {
        Q[u].push_back(Que(u, v, id));
        Q[v].push_back(Que(v, u, id));
    }

    void AddEdge(int u, int v, int d) {
        E[u].push_back(Edge(v, d));
        E[v].push_back(Edge(u, d));
    }

    int Find(int x) {
        return P[x] == x ? x : (P[x] = Find(P[x]));
    }

    void Union(int u, int v) {
        int p1 = Find(u), p2 = Find(v);
        P[p1] = p2;
    }

    /*初始DFS(root,root)*/
    void DFS(int u, int f, int d) {
        D[u] = d;
        for(int i = 0; i < E[u].size(); i++) {
            int v = E[u][i].v, cost = E[u][i].d;
            if(v != f) DFS(v, u, d + cost);
        }

        vis[u] = 1;
        for(int i = 0; i < Q[u].size(); i++) {
            int v = Q[u][i].v, id = Q[u][i].id;
            if(vis[v]) {
                ans[id] = Find(v);
            }
        }
```

```
        Union(u, f);
    }
} lca;
```

14.lca 在线

```
const int M = 30;//n的log

int dep[MX], fa[MX][M], n;
void DFS(int u, int _dep, int _fa) {
    dep[u] = _dep; fa[u][0] = _fa;
    for(int i = Head[u]; ~i; i = E[i].nxt) {
        int v = E[i].v;
        if(v == u || v == _fa) continue;
        DFS(v, _dep + 1, u);
    }
}
/*记得构图后要初始化一遍*/
void presolve() {
    DFS(1, 0, 1);
    for(int i = 1; i < M; i++) {
        for(int j = 1; j <= n; j++) {
            fa[j][i] = fa[fa[j][i - 1]][i - 1];
        }
    }
}
/*倍增法要理解对2的次方的枚举顺序
如果是要走固定步数，那么顺序枚举与i位与为1就行
如果是要求一个临界位置，那么要从大到小枚举
*/
int LCA(int u, int v) {
    while(dep[u] != dep[v]) {
        if(dep[u] < dep[v]) swap(u, v);
        int d = dep[u] - dep[v];
        for(int i = 0; i < M; i++) {
            if(d >> i & 1) u = fa[u][i];
        }
    }
    if(u == v) return u;
    for(int i = M - 1; i >= 0; i--) {
        if(fa[u][i] != fa[v][i]) {
            u = fa[u][i];
            v = fa[v][i];
        }
    }
    return fa[u][0];
}
```

15.sap 网络流

```
const int MX = 300 + 5;
const int INF = 0x3f3f3f3f;
/*复杂度O(n^2*m)
下标从0开始
*/
int maze[MX][MX];
int gap[MX], dis[MX], pre[MX], cur[MX];

int sap(int start, int end, int nodenum) {
    memset(cur, 0, sizeof(cur));
    memset(dis, 0, sizeof(dis));
    memset(gap, 0, sizeof(gap));
    int u = pre[start] = start, maxflow = 0, aug = -1;
```

```
    gap[0] = nodenum;
    while(dis[start] < nodenum) {
loop:
        for(int v = cur[u]; v < nodenum; v++) {
            if(maze[u][v] && dis[u] == dis[v] + 1) {
                if(aug == -1 || aug > maze[u][v]) aug = maze[u][v];
                pre[v] = u; u = cur[u] = v;
                if(v == end) {
                    maxflow += aug;
                    for(u = pre[u]; v != start; v = u, u = pre[u]) {
                        maze[u][v] -= aug;
                        maze[v][u] += aug;
                    }
                    aug = -1;
                }
                goto loop;
            }
        }
        int mindis = nodenum - 1;
        for(int v = 0; v < nodenum; v++) {
            if(maze[u][v] && mindis > dis[v]) {
                cur[u] = v;
                mindis = dis[v];
            }
        }
        if((--gap[dis[u]]) == 0) break;
        gap[dis[u] = mindis + 1]++;
        u = pre[u];
    }
    return maxflow;
}
```

16.dinic 邻接表网络流

```
const int MX = 1e3;
const int MS = 4e5 + 5;
const int INF = 0x3f3f3f3f;

template<class T>
struct Max_Flow {
    int n;
    int Q[MX], sign;
    int head[MX], level[MX], cur[MX], pre[MX];
    int nxt[MS], pnt[MS], E;
    T cap[MS];
    void Init(int n) {
        E = 0;
        this->n = n + 1;
        fill(head, head + this->n, -1);
    }
    void Add(int from, int to, T c, T rw = 0) {
        pnt[E] = to; cap[E] = c; nxt[E] = head[from]; head[from] = E++;
        pnt[E] = from; cap[E] = rw; nxt[E] = head[to]; head[to] = E++;
    }
    bool BFS(int s, int t) {
        sign = t;
        std::fill(level, level + n, -1);
        int *front = Q, *tail = Q;
        *tail++ = t; level[t] = 0;
        while (front < tail && level[s] == -1) {
            int u = *front++;
            for (int e = head[u]; e != -1; e = nxt[e]) {
```

```
            if (cap[e ^ 1] > 0 && level[pnt[e]] < 0) {
                level[pnt[e]] = level[u] + 1;
                *tail++ = pnt[e];
            }
        }
    }
    return level[s] != -1;
}
void Push(int t, T &flow) {
    T mi = INF;
    int p = pre[t];
    for (int p = pre[t]; p != -1; p = pre[pnt[p ^ 1]]) {
        mi = std::min(mi, cap[p]);
    }
    for (int p = pre[t]; p != -1; p = pre[pnt[p ^ 1]]) {
        cap[p] -= mi;
        if (!cap[p]) {
            sign = pnt[p ^ 1];
        }
        cap[p ^ 1] += mi;
    }
    flow += mi;
}
void DFS(int u, int t, T &flow) {
    if (u == t) {
        Push(t, flow);
        return;
    }
    for (int &e = cur[u]; e != -1; e = nxt[e]) {
        if (cap[e] > 0 && level[u] - 1 == level[pnt[e]]) {
            pre[pnt[e]] = e;
            DFS(pnt[e], t, flow);
            if (level[sign] > level[u]) {
                return;
            }
            sign = t;
        }
    }
}
T Dinic(int s, int t) {
    pre[s] = -1;
    T flow = 0;
    while (BFS(s, t)) {
        std::copy(head, head + n, cur);
        DFS(s, t, flow);
    }
    return flow;
}
};
Max_Flow<int>F;
```

17.zkw 费用流

```
namespace MCMF {
    int S, T;//源点，汇点
    int erear, n;
    int st, en, maxflow, mincost;
    bool vis[MX];
    int Head[MX], cur[MX], dis[MX];
    int roade[MX], roadv[MX], rsz; //用于打印路径
    const int ME = 4e5 + 5;//边的数量

    queue <int> Q;
```

```
struct Edge {
    int v, cap, cost, nxt, flow;
    Edge() {}
    Edge(int a, int b, int c, int d) {
        v = a, cap = b, cost = c, nxt = d, flow = 0;
    }
} E[ME], SE[ME];

void init(int _n) {
    n = _n, erear = 0;
    for(int i = 0; i <= n; i++) Head[i] = -1;
}
void edge_add(int u, int v, int cap, int cost) {
    E[erear] = Edge(v, cap, cost, Head[u]);
    Head[u] = erear++;
    E[erear] = Edge(u, 0, -cost, Head[v]);
    Head[v] = erear++;
}
bool adjust() {
    int v, min = INF;
    for(int i = 0; i <= n; i++) {
        if(!vis[i]) continue;
        for(int j = Head[i]; ~j; j = E[j].nxt) {
            v = E[j].v;
            if(E[j].cap - E[j].flow) {
                if(!vis[v] && dis[v] - dis[i] + E[j].cost < min) {
                    min = dis[v] - dis[i] + E[j].cost;
                }
            }
        }
    }
    if(min == INF) return false;
    for(int i = 0; i <= n; i++) {
        if(vis[i]) {
            cur[i] = Head[i];
            vis[i] = false;
            dis[i] += min;
        }
    }
    return true;
}
int augment(int i, int flow) {
    if(i == en) {
        mincost += dis[st] * flow;
        maxflow += flow;
        return flow;
    }
    vis[i] = true;
    for(int j = cur[i]; j != -1; j = E[j].nxt) {
        int v = E[j].v;
        if(E[j].cap == E[j].flow) continue;
        if(vis[v] || dis[v] + E[j].cost != dis[i]) continue;
        int delta = augment(v, std::min(flow, E[j].cap - E[j].flow));
        if(delta) {
            E[j].flow += delta;
            E[j ^ 1].flow -= delta;
            cur[i] = j;
            return delta;
        }
    }
    return 0;
}
void spfa() {
```

```
        int u, v;
        for(int i = 0; i <= n; i++) {
            vis[i] = false;
            dis[i] = INF;
        }
        Q.push(st);
        dis[st] = 0; vis[st] = true;
        while(!Q.empty()) {
            u = Q.front(), Q.pop(); vis[u] = false;
            for(int i = Head[u]; ~i; i = E[i].nxt) {
                v = E[i].v;
                if(E[i].cap == E[i].flow || dis[v] <= dis[u] + E[i].cost) continue;
                dis[v] = dis[u] + E[i].cost;
                if(!vis[v]) {
                    vis[v] = true;
                    Q.push(v);
                }
            }
        }
        for(int i = 0; i <= n; i++) {
            dis[i] = dis[en] - dis[i];
        }
        spfa_time_total++;
    }
    int zkw(int s, int t, int &ret_flow) {
        st = s, en = t;
        spfa();
        mincost = maxflow = 0;
        for(int i = 0; i <= n; i++) {
            vis[i] = false;
            cur[i] = Head[i];
        }
        do {
            while(augment(st, INF)) {
                memset(vis, false, n * sizeof(bool));
            }
        } while(adjust());
        ret_flow = maxflow;
        return mincost;
    }
}
```

18.普通费用流

```
const int MX = 400 + 5;//都开 4 倍把..
const int MM = 400 + 5;
const int INF = 0x3f3f3f3f;

struct Edge {
    int to, next, cap, flow, cost;
    Edge() {}
    Edge(int _to, int _next, int _cap, int _flow, int _cost) {
        to = _to; next = _next; cap = _cap; flow = _flow; cost = _cost;
    }
} E[MM];

int Head[MX], tol;
int pre[MX]; //储存前驱顶点
int dis[MX]; //储存到源点 s 的距离
bool vis[MX];
int N;//节点总个数，节点编号从 0~N-1

void init(int n) {
```

```
    tol = 0;
    N = n + 2;
    memset(Head, -1, sizeof(Head));
}
void edge_add(int u, int v, int cap, int cost) {
    E[tol] = Edge(v, Head[u], cap, 0, cost);
    Head[u] = tol++;

    E[tol] = Edge(u, Head[v], 0, 0, -cost);
    Head[v] = tol++;
}
bool spfa(int s, int t) {
    queue<int>q;
    for (int i = 0; i < N; i++) {
        dis[i] = INF;
        vis[i] = false;
        pre[i] = -1;
    }
    dis[s] = 0;
    vis[s] = true;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        vis[u] = false;
        for (int i = Head[u]; i != -1; i = E[i].next) {
            int v = E[i].to;
            if (E[i].cap > E[i].flow && dis[v] > dis[u] + E[i].cost) {
                dis[v] = dis[u] + E[i].cost;
                pre[v] = i;
                if (!vis[v]) {
                    vis[v] = true;
                    q.push(v);
                }
            }
        }
    }
    if (pre[t] == -1) return false;
    else return true;
}

//返回的是最大流， cost 存的是最小费用
int minCostMaxflow(int s, int t, int &cost) {
    int flow = 0;
    cost = 0;
    while (spfa(s, t)) {
        int Min = INF;
        for (int i = pre[t]; i != -1; i = pre[E[i ^ 1].to]) {
            if (Min > E[i].cap - E[i].flow)
                Min = E[i].cap - E[i].flow;
        }
        for (int i = pre[t]; i != -1; i = pre[E[i ^ 1].to]) {
            E[i].flow += Min;
            E[i ^ 1].flow -= Min;
            cost += E[i].cost * Min;
        }
        flow += Min;
    }
    return flow;
}
```

19.Dijstra

```
const int dij_v = 3e5;
const int dij_edge = 8e5;
template<class T>
struct Dijkstra {
    struct Edge {
        T w;
        int v, nxt;
    } E[dij_edge << 1];
    typedef pair<T, int> PII;
    int Head[dij_v], erear;
    T d[dij_v], INF;

    void init() {
        erear = 0;
        memset(Head, -1, sizeof(Head));
    }
    void add(int u, int v, T w) {
        E[erear].v = v;
        E[erear].w = w;
        E[erear].nxt = Head[u];
        Head[u] = erear++;
    }
    void run(int u) {
        memset(d, 0x3f, sizeof(d));
        INF = d[0];
        priority_queue<PII, vector<PII>, greater<PII> >Q;

        Q.push(PII(0, u)); d[u] = 0;
        Q.push(PII(A[u], u + n)); d[u + n] = A[u];
        while(!Q.empty()) {
            PII ftp = Q.top(); Q.pop();
            int u = ftp.second;
            if(ftp.first != d[u]) continue;
            for(int i = Head[u]; ~i; i = E[i].nxt) {
                int v = E[i].v; T w = E[i].w;
                if(d[u] + w < d[v]) {
                    d[v] = d[u] + w;
                    Q.push(PII(d[v], v));
                }
            }
        }
    }
};
Dijkstra<LL> dij;
```

20.floyd

```
void floyd(int n) {
    for(int k = 1; k <= n; k++) {
        for(int i = 1; i <= n; i++) {
            for(int j = 1; j <= n; j++) {
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
            }
        }
    }
}
```

21.普通 spfa

```
/*
spfa 的优化
SPFA 算法有两个优化算法 SLF 和 LLL： SLF：Small Label First 策略，设要加入的节点是 j，队首元素为 i，若 dist(j)<dist(i)，则将 j 插入队首，否则插入队尾。 LLL：Large Label Last 策略，设队首元素为 i，队
```

列中所有 dist 值的平均值为 x，若 dist(i)>x 则将 i 插入到队尾，查找下一元素，直到找到某一 i 使得 dist(i)<=x，则将 i 出对进行松弛操作。 SLF 可使速度提高 15 ~ 20%；SLF + LLL 可提高约 50%。 在实际的应用中 SPFA 的算法时间效率不是很稳定，为了避免最坏情况的出现，通常使用效率更加稳定的 Dijkstra 算法。
```
*/
void spfa(int s) {
    queue <int> q;
    for(int i = 1; i <= n; i++) {
        d[i] = INF; vis[i] = 0;
    }
    d[s] = 0; vis[s] = 1; q.push(s);
    while(!q.empty()) {
        int u = q.front(); q.pop(); vis[u] = 0;
        for(int i = Head[u]; ~i; i = E[i].nxt) {
            int v = E[i].v, cost = E[i].cost;
            if(d[u] + cost < d[v]) {
                d[v] = d[u] + cost;
                if(!vis[v]) {
                    vis[v] = 1;
                    q.push(v);
                }
            }
        }
    }
}
```

22.SLF 优化的 spfa

```
const int MX = 1e5 + 5;
const int MS = 1e5 + 5;
template<class T>
struct SPFA {
    struct Edge {
        T w;
        int v, nxt;
    } E[MS << 1];
    int Head[MX], erear;
    bool vis[MX];
    T d[MX], INF;
    deque<int> Q;

    void init() {
        erear = 0;
        memset(Head, -1, sizeof(Head));
    }
    void add(int u, int v, T w) {
        E[erear].v = v;
        E[erear].w = w;
        E[erear].nxt = Head[u];
        Head[u] = erear++;
    }

    inline void relax(int u, int v, T w) {
        if(d[u] + w < d[v]) {
            d[v] = d[u] + w;
            if(!vis[v]) {
                if(!Q.empty() && d[v] <= d[Q.front()]) {
                    Q.push_front(v);
                } else Q.push_back(v);
                vis[v] = 1;
            }
        }
    }
    void run(int u) {
```

```
        Q.clear();
        memset(d, 0x3f, sizeof(d)); INF = d[0];
        d[u] = 0; Q.push_back(u); vis[u] = 1;
        while(!Q.empty()) {
            int u = Q.front(); Q.pop_front(); vis[u] = 0;
            for(int i = Head[u]; ~i; i = E[i].nxt) {
                relax(u, E[i].v, E[i].w);
            }
        }
    }
};
SPFA<LL> spfa;
```

23.有负环的 spfa

```
/*
d[i]为-INF，说明路径上存在一个负环
d[i]为 INF，说明不连通
D[i]为其他值，表示普通的最短路
*/
int d[MX], IN[MX];
void spfa(int n, int op) {
    for(int i = 0; i <= n; i++) {
        d[i] = INF;
        vis[i] = IN[i] = 0;
    }
    queue<int>Q;
    Q.push(op); d[op] = 0;
    while(!Q.empty()) {
        int u = Q.front(); Q.pop();
        vis[u] = 0; dvis[u] = 1;
        for(int i = Head[u]; ~i; i = E[i].nxt) {
            int v = E[i].v, w = E[i].w;
            int s = d[u] == -INF ? -INF : w + d[u];
            if(s < d[v]) {
                d[v] = s;
                if(!vis[v]) {
                    vis[v] = 1; IN[v]++;
                    if(IN[v] > 10) d[v] = -INF;//最差情况为 n，根据需要修改
                    Q.push(v);
                }
            }
        }
    }
}
```

24.割顶和桥

```
int Low[MX], DFN[MX], dfs_clock;
int cut[MX];
void tarjan_init() {
    dfs_clock = 0;
    memset(DFN, 0, sizeof(DFN));
    memset(cut, 0, sizeof(cut));
}
/*
桥的性质 lowv>DFN[u]
割点的性质 lowv>=DFN[u]或者为有>=2 儿子的根节点
*/
int tarjan(int u, int e) {
    Low[u] = DFN[u] = ++dfs_clock;
    int child = 0;
    for(int id = Head[u]; ~id; id = Next[id]) {
        int v = E[id].v;
```

```
        if(!DFN[v]) {
            int lowv = tarjan(v, id | 1);
            Low[u] = min(Low[u], lowv);

            if(lowv >= DFN[u]) {
                cut[u] = 1;
            }
            if(lowv > DFN[u]) {
                E[id].sign = 1;
                E[id ^ 1].sign = 1;
            }
            child++;
        } else if((id | 1) != e && DFN[v] < DFN[u]) {
            Low[u] = min(Low[u], DFN[v]);
        }
    }
    if(e == -1 && child == 1) cut[u] = 0;
    return Low[u];
}
void tarjan_run() {
    for(int i = 1; i <= n; i++) {
        if(!DFN[i]) tarjan(i, -1);
    }
}
```

25.其他网络流

无源汇有上下界最大流
du[i]=in[i]-out[i]，入的总流减出的总流
du[i]>0,连一条边从 S 到 i，流量为 du[i]
du[i]<0,连一条边从 i 到 T，流量为-du[i]
最后看总流量事都等于所有 du[i](du[i]>0)之和

先增加一条边从 T 到 S 没有下界上界 INF
然后按无源汇有上下界最大流的方法建图，然后跑一次最大流
然后把 T 到 S 的边给拆掉
再跑一次从 S 到 T 的最大流
把两次得到的最大流相加就是答案

26.费用流可行流

方案 1：如果某一次的最短路的费用>=0 时，直接返回 false
方案 2：S->S’,费用 0，流 INF。S’->T，费用 0，流 INF。用 S’当作新的源点

27.最小割

最大流即是最小割。
最小割：删除某些边，使得 s 和 t 不连通，要求删除的边的容量和最小

28.最大权闭合图

能求一个闭合图，使得点权值之和最大。
最大权闭合图建图：
如果要选 a，就必须先选 b，c，d，那么就要连 a->b,a->c,a->d，容量为 INF
如果一个位置费用为正，那么 op->u，容量为费用
如果一个位置费用为负，那么 u->ed，容量为费用取反
最后的最大闭合权值答案就是正权之和-最大流
残余网络中的点，就是要被删除的点。

1. AC 自动机

```
/*基础代码*/
/*MX 为总长度*/
const int MX = 500000 + 5;

struct AC_machine {
    int rear, root;
    int Next[MX][26], Fail[MX], End[MX];

    void Init() {
        rear = 0;
        root = New();
    }

    int New() {
        rear++;
        End[rear] = 0;
        for(int i = 0; i < 26; i++) {
            Next[rear][i] = -1;
        }
        return rear;
    }

    void Add(char*A) {
        int n = strlen(A), now = root;
        for(int i = 0; i < n; i++) {
            int id = A[i] - 'a';
            if(Next[now][id] == -1) {
                Next[now][id] = New();
            }
            now = Next[now][id];
        }
        End[now]++;
    }
} AC;
/*状态自动机 build*/
void Build() {
    queue<int>Q;
    Fail[root] = root;
    for(int i = 0; i < 4; i++) {
        if(Next[root][i] == -1) {
            Next[root][i] = root;
        } else {
            Fail[Next[root][i]] = root;
            Q.push(Next[root][i]);
        }
    }

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

        //注意这一句话根据不同的情况修改
        if(End[Fail[u]]) End[u] = 1;
        for(int i = 0; i < 4; i++) {
            if(Next[u][i] == -1) {
                Next[u][i] = Next[Fail[u]][i];
            } else {
                Fail[Next[u][i]] = Next[Fail[u]][i];
```

```
                Q.push(Next[u][i]);
            }
        }
    }
}
/*字符串匹配 build*/
void Build() {
    queue<int>Q;
    Fail[root] = root;
    for(int i = 0; i < 26; i++) {
        if(Next[root][i] == -1) {
            Next[root][i] = root;
        } else {
            Fail[Next[root][i]] = root;
            Q.push(Next[root][i]);
        }
    }

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

        for(int i = 0; i < 26; i++) {
            if(Next[u][i] == -1) {
                Next[u][i] = Next[Fail[u]][i];
            } else {
                Fail[Next[u][i]] = Next[Fail[u]][i];
                Q.push(Next[u][i]);
            }
        }
    }
}
/*匹配串出现了几个*/
int Query(char *S) {
    int n = strlen(S), now = root, ret = 0;
    for(int i = 0; i < n; i++) {
        now = Next[now][S[i] - 'a'];
        int temp = now;

        while(temp != root) {
            ret += End[temp];
            //如果要多次匹配，下面改成 vis 标记
            End[temp] = 0;
            temp = Fail[temp];
        }
    }
    return ret;
}
/*每个字符串出现次数*/
void Query(char *S) {
    int n = strlen(S), now = root;
    for(int i = 0; i < n; i++) {
        now = Next[now][S[i] - 'a'];
        int temp = now;

        while(temp != root) {
            if(End[temp]) ans[End[temp]]++;
            temp = Fail[temp];
        }
    }
}
/*防重叠匹配出现次数*/
void Query(char *S) {
```

```
    int n = strlen(S), now = root, ret = 0;

    for(int i = 0; i < n; i++) {
        now = Next[now][S[i] - 'a'];
        int temp = now;

        while(temp != root) {
            if(End[temp] && (last[End[temp]] == -1 || last[End[temp]] + len[temp] <= i)) {
                ans[End[temp]]++;
                last[End[temp]] = i;
            }
            temp = Fail[temp];
        }
    }
}
```

2. 在线 AC 自 X 动机

```
/*下标从 1 开始*/
const int MG = 30;
const int MX = 6e5 + 5;
struct Trie_graph_online {
    int nxt[MX][26], Fail[MX], End[MX], sz;
    int root[MG], gsize[MG], gsz, ssz;
    string str[MX];
    int val[MX];

    void Init() {
        sz = gsz = ssz = 0;
    }
    int New() {
        End[++sz] = 0;
        for(int i = 0; i < 26; i++) nxt[sz][i] = 0;
        return sz;
    }
    void DealNxt(int root) {
        queue<int> Q;
        Fail[root] = root;
        for(int i = 0; i < 26; i++) {
            if(!nxt[root][i]) {
                nxt[root][i] = root;
            } else {
                Fail[nxt[root][i]] = root;
                Q.push(nxt[root][i]);
            }
        }
        while(!Q.empty()) {
            int u = Q.front(); Q.pop();
            End[u] += End[Fail[u]];
            for(int i = 0; i < 26; i++) {
                if(!nxt[u][i]) {
                    nxt[u][i] = nxt[Fail[u]][i];
                } else {
                    Fail[nxt[u][i]] = nxt[Fail[u]][i];
                    Q.push(nxt[u][i]);
                }
            }
        }
    }
    void Rebuild(int l, int r, int &root) {
        root = New();
        for(int i = l; i <= r; i++) {
            int len = str[i].length();
```

```
            int rt = root;
            for(int j = 0; j < len; j++) {
                int id = str[i][j] - 'a';
                if(!nxt[rt][id]) nxt[rt][id] = New();
                rt = nxt[rt][id];
            }
            End[rt] += val[i];
        }
        DealNxt(root);
    }
    void Add(char s[], int x) {
        str[++ssz] = string(s);
        val[ssz] = x;

        gsize[++gsz] = 1; root[gsz] = ++sz;
        while(gsz >= 2 && gsize[gsz] == gsize[gsz - 1]) {
            gsz--; gsize[gsz] *= 2;
        }
        sz = root[gsz] - 1;
        Rebuild(ssz - gsize[gsz] + 1, ssz, root[gsz]);
    }
    int Query_each(int root, char s[], int len) {
        int now = root, ret = 0;
        for(int i = 0; i < len; i++) {
            now = nxt[now][s[i] - 'a'];
            ret += End[now];
        }
        return ret;
    }
    int Query(char s[]) {
        int ret = 0, len = strlen(s);
        for(int i = 1; i <= gsz; i++) {
            ret += Query_each(root[i], s, len);
        }
        return ret;
    }
} AC;
```

3. 后缀数组

```
/*
复杂度 O(nlogn)
n 自动+1 无需再管，返回的 SA,R,H 的下标都是 0~n
其中多包括了一个空字符串
SA 后缀数组，R 名次数组，H 高度数组
H[i]表示 SA[i]和 SA[i-1]的 lcp
*/
char s[MX];
int SA[MX], R[MX], H[MX];
int wa[MX], wb[MX], wv[MX], wc[MX];
int cmp(int *r, int a, int b, int l) {
    return r[a] == r[b] && r[a + l] == r[b + l];
}
void Suffix(char *r, int m = 128) {
    int n = strlen(r) + 1;
    int i, j, p, *x = wa, *y = wb, *t;
    for(i = 0; i < m; i++) wc[i] = 0;
    for(i = 0; i < n; i++) wc[x[i] = r[i]]++;
    for(i = 1; i < m; i++) wc[i] += wc[i - 1];
    for(i = n - 1; i >= 0; i--) SA[--wc[x[i]]] = i;
    for(j = 1, p = 1; p < n; j *= 2, m = p) {
        for(p = 0, i = n - j; i < n; i++) y[p++] = i;
        for(i = 0; i < n; i++) if(SA[i] >= j) y[p++] = SA[i] - j;
```

```
        for(i = 0; i < n; i++) wv[i] = x[y[i]];
        for(i = 0; i < m; i++) wc[i] = 0;
        for(i = 0; i < n; i++) wc[wv[i]]++;
        for(i = 1; i < m; i++) wc[i] += wc[i - 1];
        for(i = n - 1; i >= 0; i--) SA[--wc[wv[i]]] = y[i];
        for(t = x, x = y, y = t, p = 1, x[SA[0]] = 0, i = 1; i < n; i++) {
            x[SA[i]] = cmp(y, SA[i - 1], SA[i], j) ? p - 1 : p++;
        }
    }
    int k = 0; n--;
    for(i = 0; i <= n; i++) R[SA[i]] = i;
    for(i = 0; i < n; i++) {
        if(k) k--;
        j = SA[R[i] - 1];
        while(r[i + k] == r[j + k]) k++;
        H[R[i]] = k;
    }
}
```

4. KMP

```
int Next[MX], n;
void GetNext() {
    Next[0] = 0;
    for(int i = 1; i < n; i++) {
        int j = Next[i - 1];
        while(j && S[i] != S[j]) j = Next[j - 1];
        Next[i] = S[i] == S[j] ? j + 1 : 0;
    }
}
/*求前缀 i 循环节最长长度*/
int GetCir(int p) {
    return (p + 1) % (p - Next[p] + 1) == 0 ? p - Next[p] + 1 : p + 1;
}

/*会有重叠部分*/
int Next[MX];

int KMP(char *A, char *B) {
    int m = strlen(A), n = strlen(B);

    Next[0] = 0;
    for(int i = 1; i < n; i++) {
        int k = Next[i - 1];
        while(B[i] != B[k] && k) k = Next[k - 1];
        Next[i] = B[i] == B[k] ? k + 1 : 0;
    }

    int ans = 0, j = 0;
    for(int i = 0; i < m; i++) {
        while(A[i] != B[j] && j) j = Next[j - 1];
        if(A[i] == B[j]) j++;
        if(j == n) ans++;
    }
    return ans;
}

/*不会有重叠部分*/
int Next[MX];

int KMP(char *A, char *B) {
    int m = strlen(A), n = strlen(B);
```

```
    Next[0] = 0;
    for(int i = 1; i < n; i++) {
        int k = Next[i - 1];
        while(B[i] != B[k] && k) k = Next[k - 1];
        Next[i] = B[i] == B[k] ? k + 1 : 0;
    }

    int ans = 0, j = 0;
    for(int i = 0; i < m; i++) {
        while(A[i] != B[j] && j) j = Next[j - 1];
        if(A[i] == B[j]) j++;
        if(j == n) ans++, j = Next[j - 1];
    }
    return ans;
}
```

5. Manacher

```
const int MAX = 110000 + 10;
char s[MAX * 2];
int p[MAX * 2];

/*
首先，i>=2 的 p 才有意义
p[i]-1 为以 i 为中心的回文长度
p[i]/2 表示回文半径
i%2==0 表示这个位置为字符，i/2-1 表示原字符串的位置
i%2==1 表示为字符中间，这两边的字符在原字符串的位置分别为 i/2-1 和 i/2
*/
int manacher(char *s){
    int len = strlen(s), id = 0, ans = 0;
    for(int i = len; i >= 0; i--) {
        s[i + i + 2] = s[i];
        s[i + i + 1] = '#';
    }
    s[0] = '*';
    for(int i = 2; i < 2 * len + 1; ++i) {
        if(p[id] + id > i) p[i] = min(p[2 * id - i], p[id] + id - i);
        else p[i] = 1;
        while(s[i - p[i]] == s[i + p[i]]) p[i]++;
        if(id + p[id] < i + p[i]) id = i;
        ans = max(ans, p[i] - 1);
    }
    return ans;
}
```

6. MT 定理

```
/*
url:http://www.spoj.com/problems/HIGH/
Matrix-Tree 定理的裸题
构造方法：C 矩阵=D 矩阵-G 矩阵，D[i][i]表示 i 的度，其他位置为 0
G[i][j]=1 表示 i 和 j 之间有一条边。
之后，我们用高斯消元去求 C 的其中一个余子式的行列式就行了。
为了方便，我们通常取(n-1,n-1)的余子式。
这里有几个要注意的地方：
1.如果最后的答案非常大，要取模，就把高斯消元的除法改成逆元
2.如果我们消元的那个位置 A[i][i]为 0，应该及时返回 0，否则就会除以 0
3.得多留意重边之类的处理。
*/
```

```
const int MX = 10 + 5;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
const double eps = 1e-8;

typedef double Matrix[MX][MX];

int n, m;
Matrix C;
int G[MX][MX], D[MX][MX];

double det(Matrix A, int n) {
    double ret = 1;
    int i, j, k, r;
    for(i = 0; i < n; i++) {
        r = i;
        for(j = i + 1; j < n; j++) {
            if(fabs(A[j][i]) > fabs(A[r][i])) r = j;
        }
        if(r != i) for(j = 0; j < n; j++) swap(A[r][j], A[i][j]);
        if(fabs(A[i][i]) < eps) return 0;

        for(k = i + 1; k < n; k++) {
            double f = A[k][i] / A[i][i];
            for(j = i; j < n; j++) A[k][j] -= f * A[i][j];
        }
        ret = ret * A[i][i];
    }
    return ret;
}
int main() {
    int T; //FIN;
    scanf("%d", &T);
    while(T--) {
        memset(D, 0, sizeof(D));
        memset(G, 0, sizeof(G));
        scanf("%d%d", &n, &m);
        for(int i = 1; i <= m; i++) {
            int u, v;
            scanf("%d%d", &u, &v);
            if(u == v) continue;
            u--; v--;
            G[u][v] = G[v][u] = 1;
            D[u][u]++; D[v][v]++;
        }
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                C[i][j] = D[i][j] - G[i][j];
            }
        }
        printf("%.0f\n", fabs(det(C, n - 1)));
    }
    return 0;
}
```

7. 二分求 lcp

```
const int seed = 131;
typedef unsigned long long ULL;
ULL fac[MX], pre[MX];
char A[MX];
```

```
void presolve() {
    fac[0] = 1;
    for(int i = 1; i < MX; i++) {
        fac[i] = fac[i - 1] * seed;
    }
}
bool check(int a, int b, int l) {
    ULL left = pre[a + l - 1] - pre[a - 1] * fac[l];
    ULL right = pre[b + l - 1] - pre[b - 1] * fac[l];
    return left == right;
}
int lcp(int n, int a, int b) {
    pre[0] = 0;
    for(int i = 1; i <= n; i++) {
        pre[i] = pre[i - 1] * seed + A[i];
    }

    int l = 0, r = n, m;
    while(l <= r) {
        m = (l + r) >> 1;
        if(check(a, b, m)) l = m + 1;
        else r = m - 1;
    }
    return l - 1;
}
```

8. 最小表示法

```
int solve(char *s, int l) {
    int i = 0, j = 1, k = 0, t;
    while(i < l && j < l && k < l) {
        t = s[(i + k) >= l ? i + k - 1 : i + k] - s[(j + k) >= l ? j + k - l : j + k];
        if(!t) k++;
        else {
            if(t > 0) i = i + k + 1;
            else j = j + k + 1;
            if(i == j) j++;
            k = 0;
        }
    }
    return min(i, j);
}
```

数论

1. 矩阵快速幂(vector 版)

```
typedef vector<int> vec;
typedef vector<vec> mat;
mat mat_mul(mat &A, mat &B) {
    mat C(A.size(), vec(B[0].size()));
    for(int i = 0; i < A.size(); i++) {
        for(int j = 0; j < B[0].size(); j++) {
            for(int k = 0; k < B.size(); k++) {
                C[i][j] = ((LL)A[i][k] * B[k][j] + C[i][j]) % mod;
            }
        }
    }
    return C;
}
mat mat_pow(mat A, LL n) {
    mat B(A.size(), vec(A.size()));
    for(int i = 0; i < A.size(); i++) B[i][i] = 1;
```

```
    while(n) {
        if(n & 1) B = mat_mul(B, A);
        A = mat_mul(A, A);
        n >>= 1;
    }
    return B;
}
/*初始化矩阵*/
mat A(n, vec(n));
```

2. 矩阵快速幂(更快)

```
const int matX = 1e2 + 5;
const int mod = 1e9 + 7;
struct Matrix {
    int n, m, s[matX][matX];
    Matrix(int n, int m): n(n), m(n) {
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) s[i][j] = 0;
        }
    }
    Matrix operator*(const Matrix &P)const {
        Matrix ret(n, P.m);
        for(int i = 0; i < n; i++) {
            for(int k = 0; k < m; k++) {
                if(s[i][k]) {
                    for(int j = 0; j < P.m; j++) {
                        ret.s[i][j] = ((LL)s[i][k] * P.s[k][j] + ret.s[i][j]) % mod;
                    }
                }
            }
        }
        return ret;
    }
    Matrix operator^(const LL &P)const {
        LL num = P;
        Matrix ret(n, m), tmp = *this;
        for(int i = 0; i < n; i++) ret.s[i][i] = 1;
        while(num) {
            if(num & 1) ret = ret * tmp;
            tmp = tmp * tmp;
            num >>= 1;
        }
        return ret;
    }
};
```

3. O(1)gcd

```
namespace qwb_gcd {
    const int MX = 33000 + 5;//最大值
    const int MP = 1e4 + 5;//1e6 时约 79000 个
    const int MK = 180;//sqrt(MX)

    int g[MK][MK];
    int prime[MP], prear;
    int pmin[MX], s[MX][3];
    bool not_prime[MP];

    void init() {
        for(int i = 1; i < MK; i++) {
            for(int j = 0; j < i; j++) {
                if(!j) g[i][j] = i;
                else g[i][j] = g[j][i % j];
```

```
            }
        }

        prear = 0;
        not_prime[1] = 1;
        for(int i = 2; i < MX; i++) {
            if(!not_prime[i]) {
                pmin[i] = i;
                prime[++prear] = i;
            }
            for(int j = 1; j <= prear && i * prime[j] < MX; j++) {
                not_prime[prime[j]*i] = 1;
                pmin[prime[j]*i] = prime[j];
                if(i % prime[j] == 0) break;
            }
        }

        s[1][0] = s[1][1] = s[1][2] = 1;
        for(int i = 2; i < MX; i++) {
            for(int j = 0; j < 3; j++) s[i][j] = s[i / pmin[i]][j];
            if(s[i][0]*pmin[i] < MK) s[i][0] *= pmin[i];
            else if(s[i][1]*pmin[i] < MK) s[i][1] *= pmin[i];
            else s[i][2] *= pmin[i];
        }
    }

    int gcd(int x, int y) {
        if(!x || !y) return x + y;
        if(x < MK && y < MK) return g[x][y % x];

        int ret = 1, d;
        for(int i = 0; i < 3; i++) {
            if(s[x][i] == 1) continue;
            if(s[x][i] < MK) d = g[s[x][i]][y % s[x][i]];
            else if(y % s[x][i] == 0) d = s[x][i];
            else d = 1;
            ret *= d; y /= d;
        }
        return ret;
    }
}
```

4. 线性基

```
/*复杂度 nlogn
能求出 A 数组的线性基，并保存到 p 中
要注意 A 数组的数据范围
*/
void Guass_base() {
    memset(P, 0, sizeof(P));
    for(int i = 1; i <= n; i++) {
        for(int j = 62; j >= 0; j--) {
            if(!(A[i] >> j & 1)) continue;
            if(!P[j]) {
                P[j] = A[i]; break;
            }
            A[i] ^= P[j];
        }
    }
}
```

5. k 次幂之和

```
/*
求(1^k+2^k+3^k+...+n^k)%mod
复杂度约为 O(klogMOD)
*/
const int MX = 1e6 + 10;
const int mod = 1e9 + 7;
struct Lagrange {
    short factor[MX];
    int P[MX], S[MX], ar[MX], inv[MX];

    inline LL power(LL a, LL b) {
        LL res = 1;
        while (b) {
            if (b & 1) res = res * a % mod;
            a = a * a % mod;
            b >>= 1;
        }
        return res;
    }
    int lagrange(LL n, int k) {
        if (!k) return n % mod;

        int i, j, x, res = 0;
        if (!inv[0]) {
            for (i = 2, x = 1; i < MX; i++) x = (long long)x * i % mod;
            inv[MX - 1] = power(x, mod - 2);
            for (i = MX - 2; i >= 0; i--) inv[i] = ((long long)inv[i + 1] * (i + 1)) % mod;
        }

        k++;
        for (i = 0; i <= k; i++) factor[i] = 0;
        for (i = 4; i <= k; i += 2) factor[i] = 2;
        for (i = 3; (i * i) <= k; i += 2) {
            if (!factor[i]) {
                for (j = (i * i), x = i << 1; j <= k; j += x) {
                    factor[j] = i;
                }
            }
        }

        for (ar[1] = 1, ar[0] = 0, i = 2; i <= k; i++) {
            if (!factor[i]) ar[i] = power(i, k - 1);
            else ar[i] = ((LL)ar[factor[i]] * ar[i / factor[i]]) % mod;
        }

        for (i = 1; i <= k; i++) {
            ar[i] += ar[i - 1];
            if (ar[i] >= mod) ar[i] -= mod;
        }
        if (n <= k) return ar[n];

        P[0] = 1, S[k] = 1;
        for (i = 1; i <= k; i++) P[i] = ((LL)P[i - 1] * ((n - i + 1) % mod)) % mod;
        for (i = k - 1; i >= 0; i--) S[i] = ((LL)S[i + 1] * ((n - i - 1) % mod)) % mod;

        for (i = 0; i <= k; i++) {
            x = (LL)ar[i] * P[i] % mod * S[i] % mod * inv[k - i] % mod * inv[i] % mod;
            if ((k - i) & 1) {
                res -= x;
                if (res < 0) res += mod;
```

```
            } else {
                res += x;
                if (res >= mod) res -= mod;
            }
        }
        return res % mod;
    }
} lgr;
```

6. 约瑟夫环

```
/*
F[n] = (F[n - 1] + m) % n, F[1] = 0
返回的下标从 0 开始，复杂度大约为 O(m)*/
int Joseph(int n, int m) {
    if(n == 1) return 0;
    if(m == 1) return n - 1;
    LL pre = 0; int now = 2;
    while(now <= n) {
        if(pre + m >= now) {
            pre = (pre + m) % now;
            now++;
        } else {
            int a = now - 1 - pre, b = m - 1;
            int k = a / b + (a % b != 0);
            if(now + k > n + 1) k = n + 1 - now;
            pre = (pre + (LL)m * k) % (now + k - 1);
            now += k;
        }
    }
    return pre;
}
```

7. fft

```
const double PI = acos(-1.0);
struct complex {
    double r, i;
    complex(double _r = 0.0, double _i = 0.0) {
        r = _r; i = _i;
    }
    complex operator +(const complex &b) {
        return complex(r + b.r, i + b.i);
    }
    complex operator -(const complex &b) {
        return complex(r - b.r, i - b.i);
    }
    complex operator *(const complex &b) {
        return complex(r * b.r - i * b.i, r * b.i + i * b.r);
    }
};
void change(complex y[], int len) {
    int i, j, k;
    for(i = 1, j = len / 2; i < len - 1; i++) {
        if(i < j) swap(y[i], y[j]);
        k = len / 2;
        while(j >= k) {
            j -= k;
            k /= 2;
        }
        if(j < k) j += k;
    }
}
void fft(complex y[], int len, int on) {
```

```
    change(y, len);
    for(int h = 2; h <= len; h <<= 1) {
        complex wn(cos(on * 2 * PI / h), sin(on * 2 * PI / h));
        for(int j = 0; j < len; j += h) {
            complex w(1, 0);
            for(int k = j; k < j + h / 2; k++) {
                complex u = y[k];
                complex t = w * y[k + h / 2];
                y[k] = u + t;
                y[k + h / 2] = u - t;
                w = w * wn;
            }
        }
    }
    if(on == -1) {
        for(int i = 0; i < len; i++) {
            y[i].r /= len;
        }
    }
}
void solve(int n) {
    for(len = 1; len < 2 * n; len <<= 1);
    for(int i = 0; i < len; i++) {
        a[i] = 条件 ? complex(A[i], 0) : complex(0, 0);
        b[i] = 条件 ? complex(B[i], 0) : complex(0, 0);
    }
    fft(a, len, 1); fft(b, len, 1);
    for(int i = 0; i < len; i++) {
        a[i] = a[i] * b[i];
    }
    fft(a, len, -1);
    for(int i = 0; i < len; i++) {
        int t = a[i].r + 0.5;
        if(t) printf("[%d]%d\n", i, t);
    }
}
```

8. fwt

```
/*
复杂度 O(nlogn)，n 为区间长度
n 必须为 2 的 n 次幂
使用方法：
fwt(a,0,n-1);fwt(b,0,n-1);
for(int i=0;i<n-1;i++) a[i]=a[i]*b[i]%mod;
fwt(a,0,n-1);
之后 a 数组就是答案
模数只是为了防止爆 long long
如果答案不会爆 long long，可以不模数
*/
LL inv2 = power(2, mod - 2);
void fwt_xor(LL a[], int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    fwt_xor(a, l, mid);
    fwt_xor(a, mid + 1, r);
    int len = mid - l + 1;
    for (int i = l; i <= mid; ++i) {
        LL x1 = a[i];
        LL x2 = a[i + len];
        a[i] = (x1 + x2) % mod;
        a[i + len] = (x1 - x2 + mod) % mod;
```

```
    }
}
void ifwt_xor(LL a[], int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    int len = mid - l + 1;
    for (int i = l; i <= mid; ++i) {
        LL y1 = a[i];
        LL y2 = a[i + len];
        a[i] = (y1 + y2) * inv2 % mod;
        a[i + len] = ((y1 - y2 + mod) % mod * inv2) % mod;
    }
    ifwt_xor(a, l, mid);
    ifwt_xor(a, mid + 1, r);
}

void fwt_and(LL a[], int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    fwt_and(a, l, mid);
    fwt_and(a, mid + 1, r);
    int len = mid - l + 1;
    for (int i = l; i <= mid; ++i) {
        LL x1 = a[i];
        LL x2 = a[i + len];
        a[i] = (x1 + x2) % mod;
        a[i + len] = x2 % mod;
    }
}
void ifwt_and(LL a[], int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    int len = mid - l + 1;
    for (int i = l; i <= mid; ++i) {
        LL y1 = a[i];
        LL y2 = a[i + len];
        a[i] = (y1 - y2 + mod ) % mod;
        a[i + len] = y2 % mod;
    }
    ifwt_and(a, l, mid);
    ifwt_and(a, mid + 1, r);
}

void fwt_or(LL a[], int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    fwt_or(a, l, mid);
    fwt_or(a, mid + 1, r);
    int len = mid - l + 1;
    for (int i = l; i <= mid; ++i) {
        LL x1 = a[i];
        LL x2 = a[i + len];
        a[i] = x1 % mod;
        a[i + len] = (x2 + x1) % mod;
    }
}
void ifwt_or(LL a[], int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    int len = mid - l + 1;
    for (int i = l; i <= mid; ++i) {
        LL y1 = a[i];
        LL y2 = a[i + len];
```

```
        a[i] = y1 % mod;
        a[i + len] = (y2 - y1 + mod) % mod;
    }
    ifwt_or(a, l, mid);
    ifwt_or(a, mid + 1, r);
}
```

9. ntt

```
const int MX = 5e5;
const int g = 3;//3
const LL MOD = 40531930642382849LL;//(479<<21)+1


LL qp[40];
LL x1[MX], x2[MX];

LL multi (LL x , LL y, LL mod) {
    return (x * y - (LL)(x / (long double)mod * y + 1e-3) * mod + mod) % mod ;
}
LL power(LL x, LL y, LL P) {
    LL ans = 1;
    while(y > 0) {
        if(y & 1)ans = multi(ans, x, P) % P;
        x = multi(x, x, P) % P;
        y >>= 1;
    }
    return ans;
}
/*记得一定要 init()*/
void init() {
    for(int i = 0; i < 33; i++) {
        int t = 1 << i;
        qp[i] = power(g, (MOD - 1) / t, MOD);
    }
}
void rader(LL F[], int len) {
    int j = len / 2;
    for(int i = 1; i < len - 1; i++) {
        if(i < j)swap(F[i], F[j]);
        int k = len / 2;
        while(j >= k) {
            j -= k;
            k >>= 1;
        }
        if(j < k)j += k;
    }
}
void ntt(LL F[], int len, int t) {
    int id = 0;
    rader(F, len);
    for(int h = 2; h <= len; h <<= 1) {
        id++;
        for(int j = 0; j < len; j += h) {
            LL E = 1;
            for(int k = j; k < j + h / 2; k++) {
                LL u = F[k] % MOD;
                LL v = multi(E, F[k + h / 2], MOD);
                F[k] = (u + v) % MOD;
                F[k + h / 2] = ((u - v) + MOD) % MOD;
                E = multi(E, qp[id], MOD);
            }
        }
```

```
    }
    if(t == -1) {
        for(int i = 1; i < len / 2; i++)swap(F[i], F[len - i]);
        LL inv = power(len, MOD - 2, MOD);
        for(int i = 0; i < len; i++)F[i] = multi(F[i] % MOD, inv, MOD);
    }
}
void solve(int n) {
    //len 为长度,len1 为 2 的幂的长度
    int len = n, len1 = 1;
    while(len1 < 2 * len) len1 *= 2;
    ntt(x1, len1, 1); ntt(x2, len1, 1);
    for(int i = 0; i < len1; i++) {
        x1[i] = multi(x1[i], x2[i], MOD);
    }
    ntt(x1, len1, -1);
}
```

10.Lucas 定理

```
int fac[mod + 7];
void init() {
    fac[0] = 1;
    for (int i = 1; i <= mod; ++i) fac[i] = (LL)fac[i - 1] * i % mod;
}
LL power(LL a, LL b) {
    LL x = a % mod, ret = 1;
    while (b) {
        if (b & 1) ret = ret * x % mod;
        x = x * x % mod;
        b >>= 1;
    }
    return ret;
}
LL C(int n, int m, int mod) {
    return m > n ? 0 : fac[n] * power((LL)fac[m] * fac[n - m], mod - 2) % mod;
}
LL Lucas(LL n, LL m, int mod) {
    return m ? (LL)C(n % mod, m % mod, mod) * Lucas(n / mod, m / mod, mod) % mod : 1;
}
```

11.大质数判定

```
LL multi(LL a, LL b, LL mod) {
    LL ret = 0;
    while(b) {
        if(b & 1) ret = ret + a;
        if(ret >= mod) ret -= mod;

        a = a + a;
        if(a >= mod) a -= mod;
        b >>= 1;
    }
    return ret;
}

LL power(LL a, LL b, LL mod) {
    LL ret = 1;
    while(b) {
        if(b & 1) ret = multi(ret, a, mod);
        a = multi(a, a, mod);
        b >>= 1;
    }
```

```
    return ret;
}

bool Miller_Rabin(LL n) {
    LL u = n - 1, pre, x;
    int i, j, k = 0;
    if(n == 2 || n == 3 || n == 5 || n == 7 || n == 11)  return true;
    if(n == 1 || (!(n % 2)) || (!(n % 3)) || (!(n % 5)) || (!(n % 7)) || (!(n % 11)))   return
false;
    for(; !(u & 1); k++, u >>= 1);
    srand(time(NULL));
    for(i = 0; i < 5; i++) {
        x = rand() % (n - 2) + 2;
        x = power(x, u, n);
        pre = x;
        for(j = 0; j < k; j++) {
            x = multi(x, x, n);
            if(x == 1 && pre != 1 && pre != (n - 1))
                return false;
            pre = x;
        }
        if(x != 1)  return false;
    }
    return true;
}
```

12.康托展开

```
int F[100];

void init() {
    F[0] = 1;
    for(int i = 1; i <= 9; i++) F[i] = F[i - 1] * i;
}

/*下标从0开始,返回值也从0开始*/
int Contor(int A[], int n) {
    int ret = 0;
    for(int i = 0; i < n; i++) {
        int cnt = 0;
        for(int j = i + 1; j < n; j++) {
            if(A[i] > A[j]) cnt++;
        }
        ret += F[n - i - 1] * cnt;
    }
    return ret;
}
```

13.扩展欧几里德

```
/*可以得到x>=bound时的x和y，返回true表示有解
否则无解，我只想问这个模板无脑调用有木有~
但是不同的题目特判不同，有的地方记得还是特判，比如a和b的正负和是否为0~*/
LL exgcd(LL a, LL b, LL &x, LL &y) {
    if(b == 0) {
        x = 1; y = 0;
        return a;
    }
    LL r = exgcd(b, a % b, x, y);
    LL t = y;
    y = x - a / b * y;
    x = t;
    return r;
```

```
}
bool solve(LL a, LL b, LL c, LL bound, LL &x, LL &y) {
    LL xx, yy, d = exgcd(a, b, xx, yy);
    if(c % d) return false;

    xx = xx * c / d; yy = yy * c / d;
    LL t = (bound - xx) * d / b;

    x = xx + b / d * t;
    if(x < bound) {
        t++;
        x = xx + b / d * t;
    }
    y = yy - a / d * t;
    return true;
}
```

14.欧拉函数

```
/*单个点的欧拉函数 O(sqrt(n))*/
LL eular(LL n) {
    LL ans = n;
    for(int i = 2; (LL)i * i <= n; i++) {
        if(n % i == 0) {
            ans -= ans / i;
            while(n % i == 0) n /= i;
        }
    }
    if(n > 1) ans -= ans / n;
    return ans;
}

/*线性筛 O(nlogn)*/
void phi_init() {
    memset(phi, 0, sizeof(phi));
    phi[1] = 1;
    for(int i = 2; i < MX; i++) if(!phi[i]) {
            for(int j = i; j < MX; j += i) {
                if(!phi[j]) phi[j] = j;
                phi[j] = phi[j] / i * (i - 1);
            }
        }
}
```

15.求组合数

利用递推公式

```
const int MX = 1000;
LL C[MX][MX];

C[0][0] = 1;
for(int i = 1; i < MX; i++) {
    C[i][0] = C[i][i] = 1;
    for(int j = 1; j < i; j++) {
        C[i][j] = (C[i-1][j-1] + C[i-1][j]) % mod;
    }
}

/*利用费马小定理*/
const int MX = 1000000 + 5;
const int mod = 1e9 + 7;

LL F[MX], invF[MX];
```

```
LL power(LL a, LL b) {
    LL ret = 1;
    while(b) {
        if(b & 1) ret = (ret * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return ret;
}

void init() {
    F[0] = 1;
    for(int i = 1; i < MX; i++) {
        F[i] = (F[i - 1] * i) % mod;
    }
    invF[MX - 1] = power(F[MX - 1], mod - 2);
    for(int i = MX - 2; i >= 0; i--) {
        invF[i] = invF[i + 1] * (i + 1) % mod;
    }
}

LL C(int n, int m) {
    if(n < 0 || m < 0 || m > n) return 0;
    if(m == 0 || m == n)    return 1;
    return F[n] * invF[n - m] % mod * invF[m] % mod;
}

LL A(int n, int m) {
    if(n < 0 || m < 0 || m > n) return 0;
    return F[n] * invF[n - m] % mod;
}
```

16.高斯消元(浮点数)

```
const double exps = 1e-8;
typedef vector<double> vec;
typedef vector<vec> mat;

int dcmp(double x) {
    if(fabs(x) < exps) return 0;
    return x < 0 ? -1 : 1;
}

void guass(mat &A, int m, int n) {
    for(int i = 0; i < m; i++) {
        int pv = i, id;
        for(int j = 0; j <= n; j++) {
            for(int k = i + 1; k < m; k++) {
                if(fabs(A[k][j]) > fabs(A[pv][j])) {
                    pv = k;
                }
            }
            if(dcmp(A[pv][j])) break;
        }
        swap(A[i], A[pv]);

        for(id = 0; id <= n && !dcmp(A[i][id]); id++);
        if(id > n) return;

        for(int j = i + 1; j < m; j++) {
            if(!dcmp(A[j][id])) continue;

            double f = A[j][id] / A[i][id];
```

```
            for(int k = id + 1; k <= n; k++) A[j][k] -= A[i][k] * f;
            A[j][id] = 0;
        }
    }
}
/*-1 无解，0 多组解，1 唯一解*/
int solve(mat &A) {
    int m = A.size(), n = A[0].size() - 1;
    guass(A, m, n);

    int r1 = 0, r2 = 0;
    for(int i = 0; i < m; i++) {
        bool sign = true;
        for(int j = 0; j <= n; j++) {
            if(dcmp(A[i][j])) {
                r2++;
                if(j < n) r1++;
                sign = false;
                break;
            }
        }
        if(sign) break;
    }

    if(r1 != r2) return -1;
    if(r1 == r2 && r1 != n) return 0;
    for(int i = n - 1; i >= 0; i--) {
        A[i][n] /= A[i][i];
        for(int j = i - 1; j >= 0; j--) A[j][n] -= A[i][n] * A[j][i];
    }
    return 1;
}
```

17.高斯消元(任意模数)

```
///以下代码是用高斯消元求同余方程组
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if(!b) {
        x = 1;
        y = 0;
        return a;
    } else {
        ll r = exgcd(b, a % b, y, x);
        y -= x * (a / b);
        return r;
    }
}
ll lcm(ll a, ll b) {
    ll x = 0, y = 0;
    return a / exgcd(a, b, x, y) * b;
}
int A[MX][MX], free_x[MX], x[MX];
void Gauss(int n, int m) {
    int r, c;
    for(r = 0, c = 0; r < n && c < m; c++) {
        int maxr = r;
        for(int i = r + 1; i < n; i++) if(abs(A[i][c]) > abs(A[maxr][c])) maxr = i;
        if(maxr != r) for(int i = c; i <= m; i++) swap(A[r][i], A[maxr][i]);
        if(!A[r][c]) continue;
        for(int i = r + 1; i < n; i++) if(A[i][c]) {
                ///这里要保证运算都是整数，所以要求最小公倍数
                ///范围不大可以用 int，int 运算更快
                ll d = lcm(A[i][c], A[r][c]);
```

```
                ll t1 = d / A[i][c], t2 = d / A[r][c];
                for(int j = c; j <= m; j++)
                    A[i][j] = ((A[i][j] * t1 - A[r][j] * t2) % mod + mod) % mod;
            }
        r++;
    }
    for(int i = r; i < n; i++) if(A[i][m]) return ;
    ///这里保证是没有自由变元的情况下。
    ///有自由变元的时候，不一定是 x[i] 对应 A[i][m]应该找那一行最开始的一列不为 0 的那个
    for(int i = r - 1; i >= 0; i--) {
        x[i] = A[i][m];
        for(int j = i + 1; j < m; j++) {
            x[i] = ((x[i] - A[i][j] * x[j]) % mod + mod) % mod;
        }
        ll x1 = 0, y1 = 0;
        ///这里是用 exgcd 求逆元，也可以用费马小定理求，如果 mod 是素数
        ll d = exgcd(A[i][i], mod, x1, y1);
        x1 = ((x1 % mod) + mod) % mod;
        x[i] = x[i] * x1 % mod;
    }
}
void Gauss_init() {
    memset(A, 0, sizeof(A));
    memset(free_x, 0, sizeof(free));
    memset(x, 0, sizeof(x));
}
```

18.高斯消元(xor)

```
int gauss(int equ, int var) {
    int max_r, col, k;
    for(k = 0, col = 0; k < equ && col < var; k++, col++) {
        max_r = k;
        for(int i = k + 1; i < equ; i++) {
            if(A[i][col] > A[max_r][col]) {
                max_r = i;
            }
        }
        if(A[max_r][col] == 0) {
            k--;
            continue;
        }
        if(max_r != k) {
            for(int j = col; j < var + 1; j++) {
                swap(A[k][j], A[max_r][j]);
            }
        }
        for(int i = k + 1; i < equ; i++) {
            if(A[i][col] != 0) {
                for(int j = col; j < var + 1; j++) {
                    A[i][j] ^= A[k][j];
                }
            }
        }
    }
    for(int i = k; i < equ; i++) {
        if(A[i][col] != 0) return -1;
    }
    if(k < var) return var - k;

    for(int i = var - 1; i >= 0; i--) {
        for(int j = i + 1; j < var; j++) {
            A[i][var] ^= (A[i][j] && A[j][var]);
```

```
        }
    }
    return 0;
}
```

19.凸包

```
struct Node {
    double x, y;
    bool operator<(const Node&b) const {
        if(x == b.x) return y < b.y;
        return x < b.x;
    }
} P[MX], R[MX];

double cross(Node a, Node b, Node c) {
    return ((b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y));
}

int convex(int n) {
    int m = 0, k;
    sort(P, P + n);
    for(int i = 0; i < n; i++) {
        while(m > 1 && cross(R[m - 1], P[i], R[m - 2]) <= 0) m--;
        R[m++] = P[i];
    }

    k = m;
    for(int i = n - 2; i >= 0; i--) {
        while(m > k && cross(R[m - 1], P[i], R[m - 2]) <= 0) m--;
        R[m++] = P[i];
    }
    if(n > 1) m--;
    return m;
}
```

20.极角排序

```
struct Point {
    LL x, y;
    Point(LL _x = 0, LL _y = 0) {
        x = _x; y = _y;
    }
    Point operator-(const Point &P) const {
        Point ret(x - P.x, y - P.y);
        return ret;
    }
    LL operator^(const Point &P) const {
        return x * P.y - y * P.x;
    }
    LL operator*(const Point &P) const {
        return x * P.x + y * P.y;
    }
} P[MX], W[MX], pc;
int n;

inline int get_seg(Point a) {
    if(a.x > 0 && a.y >= 0) return 1;
    if(a.x <= 0 && a.y > 0) return 2;
    if(a.x < 0 && a.y <= 0) return 3;
    return 4;
}
/*pc 为当前排序的中心*/
```

```
bool cmp(const Point &a, const Point &b) {
    if(a.x == pc.x && a.y == pc.y) return 1;
    if(b.x == pc.x && b.y == pc.y) return 0;
    int u = get_seg(a - pc), v = get_seg(b - pc);
    if(u == v) return ((a - pc) ^ (b - pc)) > 0;
    return u < v;
}
```

21.集合-莫比乌斯反演

```
/*
莫比乌斯反演复杂度 O(nlogn)
若原先为整个集合的所有子集的答案之和
通过反演后，就能得到本身的答案
*/
for(int i = 0; i < n; i++) {
    for(int s = 0; s < 1 << n; s++) {
        if(s >> i & 1) continue;
        dp[s | (1 << i)] -= dp[s];
    }
}
```

22.集合-莫比乌斯变换

```
/*
莫比乌斯变换复杂度 O(nlogn)
可以将子集的答案求和，加到自己上面
*/
for(int i = 0; i < n; i++) {
    for(int s = 0; s < 1 << n; s++) {
        if(s >> i & 1) continue;
        dp[s | (1 << i)] += dp[s];
    }
}
```

23.O(3^n)枚举子集

```
for(int i = s; i; i = (i - 1)&s) {

}
```

24.求 n 以内质数个数

```
/*
使用前，先 init()
n 可以等于 1e11
lehmer_pi(n)求 n 以内质数的个数
*/
const int N = 5e6 + 2;
const int M = 7;
const int PM = 2 * 3 * 5 * 7 * 11 * 13 * 17;
struct prime_cnt {
    bool np[N];
    int p[N], pi[N];
    int getprime() {
        int cnt = 0;
        np[0] = np[1] = true;
        pi[0] = pi[1] = 0;
        for(int i = 2; i < N; ++i) {
            if(!np[i]) p[++cnt] = i;
            pi[i] = cnt;
```

```
        for(int j = 1; j <= cnt && i * p[j] < N; ++j) {
            np[i * p[j]] = true;
            if(i % p[j] == 0)   break;
        }
    }
    return cnt;
}

int phi[PM + 1][M + 1], sz[M + 1];
void init() {
    getprime();
    sz[0] = 1;
    for(int i = 0; i <= PM; ++i)  phi[i][0] = i;
    for(int i = 1; i <= M; ++i) {
        sz[i] = p[i] * sz[i - 1];
        for(int j = 1; j <= PM; ++j) {
            phi[j][i] = phi[j][i - 1] - phi[j / p[i]][i - 1];
        }
    }
}
int sqrt2(LL x) {
    LL r = (LL)sqrt(x - 0.1);
    while(r * r <= x) ++r;
    return int(r - 1);
}
int sqrt3(LL x) {
    LL r = (LL)cbrt(x - 0.1);
    while(r * r * r <= x) ++r;
    return int(r - 1);
}
LL getphi(LL x, int s) {
    if(s == 0)  return x;
    if(s <= M)  return phi[x % sz[s]][s] + (x / sz[s]) * phi[sz[s]][s];
    if(x <= p[s]*p[s])   return pi[x] - s + 1;
    if(x <= p[s]*p[s]*p[s] && x < N) {
        int s2x = pi[sqrt2(x)];
        LL ans = pi[x] - (s2x + s - 2) * (s2x - s + 1) / 2;
        for(int i = s + 1; i <= s2x; ++i) {
            ans += pi[x / p[i]];
        }
        return ans;
    }
    return getphi(x, s - 1) - getphi(x / p[s], s - 1);
}
LL getpi(LL x) {
    if(x < N)   return pi[x];
    LL ans = getphi(x, pi[sqrt3(x)]) + pi[sqrt3(x)] - 1;
    for(int i = pi[sqrt3(x)] + 1, ed = pi[sqrt2(x)]; i <= ed; ++i) {
        ans -= getpi(x / p[i]) - i + 1;
    }
    return ans;
}
LL lehmer_pi(LL x) {
    if(x < N)   return pi[x];
    int a = (int)lehmer_pi(sqrt2(sqrt2(x)));
    int b = (int)lehmer_pi(sqrt2(x));
    int c = (int)lehmer_pi(sqrt3(x));
    LL sum = getphi(x, a) + LL(b + a - 2) * (b - a + 1) / 2;
    for (int i = a + 1; i <= b; i++) {
        LL w = x / p[i];
        sum -= lehmer_pi(w);
        if (i > c) continue;
        LL lim = lehmer_pi(sqrt2(w));
```

```
            for (int j = i; j <= lim; j++) {
                sum -= lehmer_pi(w / p[j]) - (j - 1);
            }
        }
        return sum;
    }
} prime;
```

25.simpson 定积分

```
double f(double x) {
    return 2 * x;
}
double simpson(double a, double b) {
    double c = a + (b - a) / 2;
    return (f(a) + 4 * f(c) + f(b)) * (b - a) / 6;
}

double asr(double a, double b, double eps, double A) {
    double c = a + (b - a) / 2;
    double L = simpson(a, c), R = simpson(c, b);
    if(fabs(L + R - A) <= 15 * eps) return L + R + (L + R - A) / 15.0;
    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
}
/*求区间[a,b]的定积分，精度为 eps*/
double asr(double a, double b, double eps) {
    return asr(a, b, eps, simpson(a, b));
}
```

26.中国剩余定理

```
LL exgcd(LL a, LL b, LL &x, LL &y) {
    if(b == 0) {
        x = 1; y = 0;
        return a;
    }
    LL r = exgcd(b, a % b, x, y);
    LL t = y;
    y = x - a / b * y;
    x = t;
    return r;
}
LL multi(LL a, LL b, LL mod) {
    LL ret = 0;
    while(b) {
        if(b & 1) {
            ret = ret + a;
            if(ret >= mod) ret -= mod;
        }
        a = a + a;
        if(a >= mod) a -= mod;
        b >>= 1;
    }
    return ret;
}
/*x % m = a*/
LL ex_crt(int n, LL m[], LL a[]) {
    LL M = 1, d, y, x = 0;
    for(int i = 0; i < n; i++) M *= m[i];
    for(int i = 0; i < n; i++) {
        LL w = M / m[i];
        d = exgcd(m[i], w, d, y);
        y = (y % m[i] + m[i]) % m[i];
        x = ((x + multi(multi(a[i], w, M), y, M)) % M + M) % M;
```

```
    }
    return x;
}
```

27.莫比乌斯函数筛法

```
bool vis[MX];
int prime[MX], mu[MX], tot;

void miu_init() {
    memset(vis, 0, sizeof(vis));
    mu[1] = 1; tot = 0;
    for(int i = 2; i < MX; i++) {
        if(!vis[i]) {
            prime[tot++] = i;
            mu[i] = -1;
        }
        for(int j = 0; j < tot; j++) {
            if(i * prime[j] >= MX) break;
            vis[i * prime[j]] = 1;
            if(i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
                break;
            } else {
                mu[i * prime[j]] = -mu[i];
            }
        }
    }
}
```

28.可不互质的中国剩余定理

```
int n;
LL m[MX], r[MX];
LL exgcd(LL a, LL b, LL &x, LL &y) {
    if(b == 0) {
        x = 1; y = 0;
        return a;
    }
    LL r = exgcd(b, a % b, x, y);
    LL t = y;
    y = x - a / b * y;
    x = t;
    return r;
}
pair<LL, LL> ex_crt() {
    LL M = m[1], R = r[1], x, y, d;
    for(int i = 2; i <= n; i++) {
        d = exgcd(M, m[i], x, y);
        if((r[i] - R) % d) return make_pair(-1, -1);
        x = (r[i] - R) / d * x % (m[i] / d);
        R = R + x * M;
        M = M / d * m[i];
        R %= M;
    }
    R = R > 0 ? R : R + M;
    return make_pair(R, M);
}
```

29.排列组合总结

1）球同，盒同，无空箱
dp[n-m][m],dp 同第 2 种情况,n>=m
0, n<m

2）球同，盒同，允许空箱
dp[n][m]=dp[n][m-1]+dp[n-m][m], n>=m
dp[n][m]=dp[n][m-1], n<m
边界 dp[k][1]=1,dp[1][k]=1,dp[0][k]=1

3）球同，盒不同，无空箱
C(n-1,m-1), n>=m
0, n<m

4)球同，盒不同，允许空箱 C(n+m-1,m-1)

5)球不同，盒相同，无空箱 第二类斯特林数 dp[n][m]
dp[n][m]=m*dp[n-1][m]+dp[n-1][m-1],1<=m<n
dp[k][k]=1,k>=0
dp[k][0]=0,k>=1
dp[n][m]=0,n<m

6）球不同，盒相同，允许空箱 sigma dp[n][i],1<=i<=m

7)球不同，盒不同，无空箱 dp[n][m]*fact[m]

8）球不同，盒不同，允许空箱 power(m,n)

30.奇怪的公式

GCD(a,b,c)=1,则必然有 ax+by+cz=1，与扩展欧几里德的原理是一样的
若有 GCD(x,n)=1,那么在一个圈中隔点报数必能全部报完

x<=1e9,则说明最多只会由 9 个质数组成
与 n 互质的所有数(<n)的和为 n*phi(n)/2，要注意 n=1 时

错排公式 F[i]=(i-1)*(F[i-1]+F[i-2])
其中边界 F[1]=0,F[2]=1

一些质数 999983

关于组合数和杨辉三角的性质
C(m+1,n)/C(m,n)=(m+1)/(m+1-n)
杨辉三角

```
                          1
                       1   1
                     1   2   1
                   1   3   3   1
                 1   4   6   4   1
               1   5   10  10  5   1
             1   6   15  20  15  6   1
           1   7   21  35  35  21  7   1
         1   8   28  56  70  56  28  8   1
       1   9   36  84  126 126 84  36  9   1
     1   10  45  120 210 252 210 120 45  10  1
   1   11  55  165 330 462 462 330 165 55  11  1
 1   12  66  220 495 792 924 792 495 220 66  12  1
```

初始值为 1，每次都是前面的数字之和，假如做 k 次，会得到一个矩阵

```
C(k-1,k-1)     0           0              0..
C(k,k-1)       C(k-1,k-1)   0              0..
C(k+1,k-1)     C(k,k-1)     C(k-1,k-1)      0..
.........
C(k+n-2,k-1)   C(k+n-3,k-1)........
```

卡特兰数定义：F(n)=C(2n,n)/(n+1)

F(n)=C(2n,n)-C(2n,n-1)
F(n)=F(0)*F(n-1)+F(1)*F(n-2)+...+F(n-1)*F(0),n >= 1, F(0)=(1)=1
递推公式 F(n)=F(n-1)*(4n-2)/(n+1),F(1)=1

斐波那契数列
F[1]=F[2]=1,F[n]=F[n-1]+F[n-2]
F[n]=1/sqrt(5)*(pow((1+sqrt(5))/2,n)-pow((1-sqrt(5))/2,n));
奇项求和 F[1]+F[3]+F[5]+...+F[2n-1]=F[2n]
偶项求和 F[2]+F[4]+F[6]+...+F[2n]=F[2n+1]-1
全部求和 F[1]+F[2]+...+F[n]=F[n+2]-1
平方求和 F[1]*F[1]+F[2]*F[2]+...+F[n]*F[n]=F[n]*F[n+1]
两倍关系 F[2*n]/F[n]=F[n-1]+F[n+1]
其他关系 F[n-1]*F[n+1]-F[n]*F[n]=(-1)^n
F[1]+2*F[2]+3*F[3]+...+n*F[n]=n*F[n+2]-F[n+3]+2
F[m]F[n]+F[m-1]F[n-1]=F[m+n-1]
F[m]F[n+1]+F[m-1]F[n]=F[m+n]
前一项/后一项=黄金分割数
杨辉三角每行相加等于斐波那契数列
斐波那契数列个位数每 60 一循环

平方剩余：存在一个整数 x 使得 x*x%p=a
如果 p 是奇质数，则 a 平方剩余当且仅当 power(a,(p-1)/2,p)==1
且在 1,2,...p-1 中恰好有(p-1)/2 个数是平方剩余的

对于一般的数论题，所以通常取一个比较小的质数 p，然后开始找规律

F=(a+sqrt(b))^n
可以写成递推式 F(n)=a*F(n-1)+(b+a*sqrt(b))*F(n-2)

海伦公式
p=(a+b+c)/2
S=sqrt(p*(p-a)*(p-b)*(p-c))

求一个 n 个数 Ai，有多少个数与 x 互质：
  对于 Ai,设 t 为 Ai 的约数，那么 cnt[t]++
  枚举 x 的约数记为 t，然后求 sigma cnt[t]*miu[t]，即与 x 互质的个数
  其中 miu 为莫比乌斯函数

数据结构

1. 左偏树

```
/*复杂度
取最小 O(1)
合并 O(logn)
这个是最小堆，求最大堆改 merge 即可
*/
const int MX = 1000 + 5;

struct Data {
    int l, r, key, dist;
} D[MX << 1];
int rear, root;

int lt_init() {
    rear = root = 0;
    D[0].dist = -1;
}
int lt_new(int _key = 0) {
    rear++;
    D[rear].l = D[rear].r = 0;
```

```
    D[rear].key = _key;
    D[rear].dist = 0;
    return rear;
}
int lt_merge(int r1, int r2) {
    if(!r1) return r2;
    if(!r2) return r1;
    if(D[r1].key > D[r2].key) {
        swap(r1, r2);
    }
    D[r1].r = lt_merge(D[r1].r, r2);
    if(D[D[r1].l].dist < D[D[r1].r].dist) {
        swap(D[r1].l, D[r1].r);
    }
    D[r1].dist = D[D[r1].r].dist + 1;
    return r1;
}
int lt_pop(int &rt) {
    int ret = D[rt].key;
    rt = lt_merge(D[rt].l, D[rt].r);
    return ret;
};
void lt_push(int &rt, int key) {
    rt = lt_merge(rt, lt_new(key));
}

/*使用的时候
lt_init();
lt_push(rt,1);
*/
```

2. ST 表

```
/*可以从 0 也可以从 1 开始,30 应该能使用 100W 以内的*/
int A[MX];
int MIN[MX][30], MAX[MX][30];

void RMQ_init(int n) {
    for(int i = 0; i < n + 1; i++) {
        MAX[i][0] = MIN[i][0] = A[i];
    }
    for(int j = 1; (1 << j) <= n + 1; j++) {
        for(int i = 0; i + (1 << j) - 1 < n + 1; i++) {
            MAX[i][j] = max(MAX[i][j - 1], MAX[i + (1 << (j - 1))][j - 1]);
            MIN[i][j] = min(MIN[i][j - 1], MIN[i + (1 << (j - 1))][j - 1]);
        }
    }
}

int RMQ_min(int L, int R) {
    int k = 0;
    while((1 << (k + 1)) <= R - L + 1) k++;
    return min(MIN[L][k], MIN[R - (1 << k) + 1][k]);
}

int RMQ_max(int L, int R) {
    int k = 0;
    while((1 << (k + 1)) <= R - L + 1) k++;
    return max(MAX[L][k], MAX[R - (1 << k) + 1][k]);
}
```

3. 二维 RMQ

```
/*
傻逼二维 RMQ 超级大常数
理论支持下标从 0 开始
n 是一维的大小，m 是二维的大小
P 是 MX 的 log 大小
*/
int n, m;
int dp[MX][MX][P][P];
void umax(int &a, int b) {
    a = max(a, b);
}
void RMQ_init() {
    for(int p = 0; (1 << p) <= n; p++) {
        for(int q = 0; (1 << q) <= m; q++) {
            int l1 = 1 << p, l2 = 1 << q;
            for(int i = 1; i + l1 - 1 <= n; i++) {
                for(int j = 1; j + l2 - 1 <= m; j++) {
                    if(!p && !q) continue;
                    int t1 = max(p - 1, 0), t2 = max(q - 1, 0);
                    umax(dp[i][j][p][q], dp[i][j][t1][t2]);
                    umax(dp[i][j][p][q], dp[i][j + l2 - (1 << t2)][t1][t2]);
                    umax(dp[i][j][p][q], dp[i + l1 - (1 << t1)][j][t1][t2]);
                    umax(dp[i][j][p][q], dp[i + l1 - (1 << t1)][j + l2 - (1 << t2)][t1][t2]);
                }
            }
        }
    }
}
int RMQ(int x1, int y1, int x2, int y2) {
    int l1 = x2 - x1 + 1, l2 = y2 - y1 + 1;
    int p = 0, q = 0, ret = 0;
    while((1 << (p + 1)) <= l1) p++;
    while((1 << (q + 1)) <= l2) q++;
    l1 = 1 << p, l2 = 1 << q;
    umax(ret, dp[x1][y1][p][q]);
    umax(ret, dp[x1][y2 - l2 + 1][p][q]);
    umax(ret, dp[x2 - l1 + 1][y1][p][q]);
    umax(ret, dp[x2 - l1 + 1][y2 - l2 + 1][p][q]);
    return ret;
}
```

4. 线段树缩空间

```
int ID(int l, int r) {
    return l + r | l != r;
}
```

5. 曼哈顿最小生成树

```
const int MS = 4000 + 5;
const int MID = 2000 + 2;
const int MX = 1e5 + 5;
const int INF = 0x3f3f3f3f;
#define lson l,m,rt<<1
#define rson m+1,r,rt<<1|1
int n, k, sz;
int MIN[MS << 2], id[MS << 2];
int P[MX];
int find(int x) {
    return P[x] == x ? x : (P[x] = find(P[x]));
```

```
}
struct point {
    int x, y, id;
    bool operator<(const point &P)const {
        if(x == P.x) return y - x < P.y - P.x;
        else return x < P.x;
    }
} A[MX];
struct Edge {
    int u, v, cost;
    bool operator<(const Edge &P)const {
        return cost < P.cost;
    }
} E[MX];
void push_up(int rt) {
    if(MIN[rt << 1] < MIN[rt]) {
        MIN[rt] = MIN[rt << 1];
        id[rt] = id[rt << 1];
    }
    if(MIN[rt << 1 | 1] < MIN[rt]) {
        MIN[rt] = MIN[rt << 1 | 1];
        id[rt] = id[rt << 1 | 1];
    }
}
void update(int p, int x, int uid, int l, int r, int rt) {
    if(l == r) {
        if(x < MIN[rt]) {
            MIN[rt] = x;
            id[rt] = uid;
        }
        return;
    }
    int m = (l + r) >> 1;
    if(p <= m) update(p, x, uid, lson);
    else update(p, x, uid, rson);
    push_up(rt);
}
PII query(int L, int R, int l, int r, int rt) {
    if(L <= l && r <= R) {
        return PII(MIN[rt], id[rt]);
    }
    int m = (l + r) >> 1; PII ret(INF, -1);
    if(L <= m) ret = min(ret, query(L, R, lson));
    if(R > m) ret = min(ret, query(L, R, rson));
    return ret;
}
/*读入到A数组中*/
void build() {
    sz = 0;
    for(int w = 0; w <= 3; w++) {
        if(w == 1 || w == 3) {
            for(int i = 1; i <= n; i++) {
                swap(A[i].x, A[i].y);
            }
        }
        if(w == 2) {
            for(int i = 1; i <= n; i++) {
                A[i].x = -A[i].x;
            }
        }
        sort(A + 1, A + 1 + n);
        memset(MIN, INF, sizeof(MIN));
        for(int i = n; i >= 1; i--) {
```

```
            PII p = query(A[i].y - A[i].x + MID, MS - 1, 1, MS - 1, 1);
            if(p.first != INF) {
                sz++;
                E[sz].u = A[i].id; E[sz].v = p.second;
                E[sz].cost = p.first - A[i].x - A[i].y;
            }
            update(A[i].y - A[i].x + MID, A[i].x + A[i].y, A[i].id, 1, MS - 1, 1);
        }
    }
}
int MST() {
    int cnt = 0, ret = 0;
    sort(E + 1, E + 1 + sz);
    for(int i = 1; i <= n; i++) P[i] = i;
    for(int i = 1; i <= sz; i++) {
        int u = E[i].u, v = E[i].v;
        int p1 = find(u), p2 = find(v);
        if(p1 != p2) {
            cnt++; P[p2] = p1;
            ret += E[i].cost;
        }
    }
}
```

6. 主席树

```
int A[MX], B[MX], rear;
int S[MX << 2], ls[MX << 2], rs[MX << 2], o[MX], sz;
void push_up(int rt) {
    S[rt] = S[ls[rt]] + S[rs[rt]];
}
void build(int l, int r, int &rt) {
    rt = ++sz;
    if(l == r) {
        S[rt] = 0;
        return;
    }
    int m = (l + r) >> 1;
    build(l, m, ls[rt]); build(m + 1, r, rs[rt]);
    push_up(rt);
}
void update(int pos, int l, int r, int pre, int &rt) {
    rt = ++sz;
    if(l == r) {
        S[rt] = S[pre] + 1;
        return;
    }
    int m = (l + r) >> 1;
    ls[rt] = ls[pre]; rs[rt] = rs[pre];
    if(pos <= m) update(pos, l, m, ls[pre], ls[rt]);
    else update(pos, m + 1, r, rs[pre], rs[rt]);
    push_up(rt);
}
int query(int k, int l, int r, int pre, int rt) {
    if(l == r) return l;
    int m = (l + r) >> 1, num = S[ls[rt]] - S[ls[pre]];
    if(k <= num) return query(k, l, m, ls[pre], ls[rt]);
    else return query(k - num, m + 1, r, rs[pre], rs[rt]);
}
```

7. KD 树-子矩阵查询修改

```
/*通常关于曼哈顿距离的，都可以把图像旋转 45 度，之后就变成了矩阵了！*/
struct Node {
```

```
    int xy[2];
    int minx, maxx;
    int miny, maxy;
    int f, id, ls, rs;
    int val, sum;
} A[MX];
int kd_cmp, d;
int xl, xr, yl, yr;

bool cmp(const Node &a, const Node &b) {
    return a.xy[kd_cmp] < b.xy[kd_cmp];
}
inline void umax(int &a, int b) {
    a = max(a, b);
}
inline void umin(int &a, int b) {
    a = min(a, b);
}
void push_up(int rt) {
    A[rt].minx = A[rt].maxx = A[rt].xy[0];
    A[rt].miny = A[rt].maxy = A[rt].xy[1];
    if(A[rt].ls) {
        umin(A[rt].minx, A[A[rt].ls].minx);
        umax(A[rt].maxx, A[A[rt].ls].maxx);
        umin(A[rt].miny, A[A[rt].ls].miny);
        umax(A[rt].maxy, A[A[rt].ls].maxy);
    }
    if(A[rt].rs) {
        umin(A[rt].minx, A[A[rt].rs].minx);
        umax(A[rt].maxx, A[A[rt].rs].maxx);
        umin(A[rt].miny, A[A[rt].rs].miny);
        umax(A[rt].maxy, A[A[rt].rs].maxy);
    }
}
/*build(1,n,0,0);*/
int build(int l, int r, int w, int fa) {
    int m = (l + r) >> 1; kd_cmp = w;
    nth_element(A + l, A + m, A + r + 1, cmp);
    Rank[A[m].id] = m;
    A[m].val = A[m].sum = 0; A[m].f = fa;
    A[m].ls = l != m ? build(l, m - 1, !w, m) : 0;
    A[m].rs = r != m ? build(m + 1, r, !w, m) : 0;
    push_up(m);
    return m;
}
int query(int rt) {
    if(A[rt].minx > xr || A[rt].maxx < xl || A[rt].miny > yr || A[rt].maxy < yl)
        return 0;
    if(xl <= A[rt].minx && A[rt].maxx <= xr && yl <= A[rt].miny && A[rt].maxy <= yr)
        return A[rt].sum;
    int ret = 0;
    if(xl <= A[rt].xy[0] && A[rt].xy[0] <= xr && yl <= A[rt].xy[1] && A[rt].xy[1] <= yr)
        ret += A[rt].val;
    if(A[rt].ls) ret += query(A[rt].ls);
    if(A[rt].rs) ret += query(A[rt].rs);
    return ret;
}
void update(int rt, int x) {
    A[rt].val += x;
    while(rt) {
        A[rt].sum += x;
        rt = A[rt].f;
    }
```

```
}
```

8. KD 树

```
struct Point {
    int xy[2], l, r, id;
    void read(int i) {
        id = i;
        scanf("%d%d", &xy[0], &xy[1]);
    }
} P[MX];
int cmpw; LL ans;
int idx[MX];

bool cmp(const Point &a, const Point &b) {
    return a.xy[cmpw] < b.xy[cmpw];
}
int build(int l, int r, int w) {
    int m = (l + r) >> 1; cmpw = w;
    nth_element(P + l, P + m, P + 1 + r, cmp);
    idx[P[m].id] = m;
    P[m].l = l != m ? build(l, m - 1, !w) : 0;
    P[m].r = r != m ? build(m + 1, r, !w) : 0;
    return m;
}
LL dist(LL x, LL y = 0) {
    return x * x + y * y;
}
void query(int rt, int w, LL x, LL y) {
    LL temp = dist(x - P[rt].xy[0], y - P[rt].xy[1]);
    if(temp) ans = min(ans, temp);
    if(P[rt].l && P[rt].r) {
        bool sign = !w ? (x <= P[rt].xy[0]) : (y <= P[rt].xy[1]);
        LL d = !w ? dist(x - P[rt].xy[0]) : dist(y - P[rt].xy[1]);
        query(sign ? P[rt].l : P[rt].r, !w, x, y);
        if(d < ans) query(sign ? P[rt].r : P[rt].l, !w, x, y);
    } else if(P[rt].l) query(P[rt].l, !w, x, y);
    else if(P[rt].r) query(P[rt].r, !w, x, y);
}

int rt = build(1, n, 0);
for(int i = 1; i <= n; i++) {
    ans = 1e18;
    query(rt, 0, P[idx[i]].xy[0], P[idx[i]].xy[1]);
    printf("%I64d\n", ans);
}
```

9. 离线第 k 大带修改

```
const int MX = 4e5 + 5;
const int INF = 0x3f3f3f3f;

int sum[MX], flag[MX], n, DFN;
int val[MX], ans[MX], tmp[MX];
struct Data {
    int op, id, x, y, k, cnt;
    Data() {}
    Data(int _op, int _id, int _x, int _y, int _k) {
        op = _op; id = _id;
        x = _x; y = _y; k = _k;
    }
} A[MX], T1[MX], T2[MX];
void add(int x, int y, int id) {
    for(int i = x; i <= n; i += i & -i) {
```

```
        if(flag[i] != id) flag[i] = id, sum[i] = 0;
        sum[i] += y;
    }
}
int ask(int x, int id) {
    int ret = 0;
    for(int i = x; i; i -= i & -i) {
        if(flag[i] == id) ret += sum[i];
    }
    return ret;
}
void solve(int L, int R, int l, int r) {
    if(L > R) return;
    if(l == r) {
        for(int i = L; i <= R; i++) {
            if(A[i].op == 3) ans[A[i].id] = l;
        }
        return;
    }

    int m = (l + r) >> 1; DFN++;
    for(int i = L; i <= R; i++) {
        if(A[i].op == 1 && A[i].y <= m) add(A[i].x, 1, DFN);
        if(A[i].op == 2 && A[i].y <= m) add(A[i].x, -1, DFN);
        if(A[i].op == 3) tmp[i] = ask(A[i].y, DFN) - ask(A[i].x - 1, DFN);
    }
    int l1 = 0, l2 = 0;
    for(int i = L; i <= R; i++) {
        if(A[i].op == 3) {
            if(A[i].cnt + tmp[i] >= A[i].k) T1[++l1] = A[i];
            else A[i].cnt += tmp[i], T2[++l2] = A[i];
        } else if(A[i].y <= m) T1[++l1] = A[i];
        else T2[++l2] = A[i];
    }
    for(int i = 1; i <= l1; i++) A[L + i - 1] = T1[i];
    for(int i = 1; i <= l2; i++) A[L + l1 + i - 1] = T2[i];
    solve(L, L + l1 - 1, l, m);
    solve(L + l1, R, m + 1, r);
}

int main() {
    //FIN;
    while(~scanf("%d", &n)) {
        int sz = 0, Q, qsz = 0, Max = 0; DFN = 0;
        memset(sum, 0, sizeof(sum));

        for(int i = 1; i <= n; i++) {
            scanf("%d", &val[i]);
            A[++sz] = Data(1, -1, i, val[i], -1);
            Max = max(Max, val[i]);
        }
        scanf("%d", &Q);
        for(int i = 1; i <= Q; i++) {
            int op, x, y, k;
            scanf("%d%d%d", &op, &x, &y);
            if(op == 1) {
                A[++sz] = Data(2, -1, x, val[x], -1);
                A[++sz] = Data(1, -1, x, y, -1);
                Max = max(Max, y);
                val[x] = y;
            } else {
                qsz++;
                scanf("%d", &k);
```

```
                A[++sz] = Data(3, qsz, x, y, k);
                A[sz].cnt = 0;
            }
        }
        solve(1, sz, 0, Max);

        for(int i = 1; i <= qsz; i++) {
            printf("%d\n", ans[i]);
        }
    }
    return 0;

}
```

10.非旋转 Treap

```
/*这个是敌兵布阵，非旋转 treap 主要是运用 Cut 和 Merge*/
const int MX = 1e5 + 5;
struct Node {
    Node *ch[2];
    int val, r, sum, sz;
} MEMO[MX], *null, *root;
int tot = 0;
void push_up(Node *o) {
    if(o == null) return;
    o->sz = o->ch[0]->sz + o->ch[1]->sz + 1;
    o->sum = o->ch[0]->sum + o->ch[1]->sum + o->val;
}
void New(Node *&o, int val = 0) {
    o = &MEMO[tot++];
    o->ch[0] = o->ch[1] = null;
    o->sz = 1; o->r = rand();
    o->sum = o->val = val;
}
void Cut(Node *o, Node *&a, Node *&b, int p) {
    if(o->sz <= p) a = o, b = null;
    else if(p == 0) a = null, b = o;
    else {
        if(o->ch[0]->sz >= p) {
            b = o;
            Cut(o->ch[0], a, b->ch[0], p);
        } else {
            a = o;
            Cut(o->ch[1], a->ch[1], b, p - o->ch[0]->sz - 1);
        }
        push_up(o);
    }
}
void Merge(Node *&o, Node *a, Node *b) {
    if(a == null) o = b;
    else if(b == null) o = a;
    else {
        if(a->r > b->r) {
            o = a;
            Merge(o->ch[1], a->ch[1], b);
            push_up(o);
        } else {
            o = b;
            Merge(o->ch[0], a, b->ch[0]);
            push_up(o);
        }
    }
}
```

```
void Init() {
    srand(time(NULL));
    tot = 0;
    New(null);
    null->sz = 0;
    root = null;
}
void Insert(int p, int x) {
    Node *a, *b, *c;
    Cut(root, a, b, p);
    New(c, x);
    Merge(a, a, c);
    Merge(root, a, b);
}
void Update(int p, int x) {
    Node *a, *b, *c;
    Cut(root, a, b, p - 1);
    Cut(b, b, c, 1);
    b->val += x;
    push_up(b);
    Merge(b, b, c);
    Merge(root, a, b);
}
int Query(int L, int R) {
    Node *a, *b, *c;
    Cut(root, a, b, R);
    Cut(a, a, c, L - 1);
    int ans = c->sum;
    Merge(a, a, c);
    Merge(root, a, b);
    return ans;
}
int main() {
    //FIN;
    int T, n, ansk = 0;
    scanf("%d", &T);
    while(T--) {
        Init();
        printf("Case %d:\n", ++ansk);
        scanf("%d", &n);
        for(int i = 1; i <= n; i++) {
            int t; scanf("%d", &t);
            Insert(i - 1, t);
        }

        char op[10]; int x, y;
        while(scanf("%s", op), op[0] != 'E') {
            scanf("%d%d", &x, &y);
            if(op[0] == 'Q') printf("%d\n", Query(x, y));
            else if(op[0] == 'A') Update(x, y);
            else Update(x, -y);
        }
    }
    return 0;
}
```

11.平衡堆

```
int rear;
int S[MX << 2];
/*这个是最小堆*/
void push(int x) {
```

```
    int now = ++rear, pre = now >> 1;
    S[now] = x;
    while(pre && S[pre] > S[now]) {
        swap(S[now], S[pre]);
        now = pre, pre = now >> 1;
    }
}

void pop() {
    S[1] = S[rear--];

    int now = 1;
    while((now << 1) <= rear) {
        int lt = now << 1, rt = now << 1 | 1;
        if(rt <= rear) {
            if(S[lt] >= S[now] && S[rt] >= S[now]) break;
            if(S[now] >= S[lt] && S[rt] >= S[lt]) swap(S[now], S[lt]), now = lt;
            else swap(S[now], S[rt]), now = rt;
        } else {
            if(S[lt] > S[now]) break;
            swap(S[now], S[lt]), now = lt;
        }
    }
}
```

12.莫队算法

```
const int MX = 5e4 + 5;
const int MP = 1e6 + 5;
const int MQ = 2e5 + 5;

int n, unit, Qt;
LL ans[MQ];
int vis[MP], A[MX];

struct Que {
    int L, R, id;
    bool operator<(const Que &b)const {
        if(L / unit == b.L / unit) {
            if(R == b.R) return L < b.L;
            return R < b.R;
        }
        return L / unit < b.L / unit;
    }
} Q[MQ];

void solve() {
    LL sum = 0;
    int L = 1, R = 0, c = 1;
    while(c <= Qt) {
        while(Q[c].L < L) {
            vis[A[--L]]++;
            if(vis[A[L]] == 1) {
                sum += A[L];
            }
        }
        while(Q[c].R > R) {
            vis[A[++R]]++;
            if(vis[A[R]] == 1) {
                sum += A[R];
            }
        }
        while(Q[c].L > L) {
```

```
            vis[A[L]]--;
            if(!vis[A[L]]) {
                sum -= A[L];
            }
            L++;
        }
        while(Q[c].R < R) {
            vis[A[R]]--;
            if(!vis[A[R]]) {
                sum -= A[R];
            }
            R--;
        }
        ans[Q[c++].id] = sum;
    }
}

int main() {
    int T;
    scanf("%d", &T);
    while(T--) {
        memset(vis, 0, sizeof(vis));
        scanf("%d", &n);
        unit = sqrt(n + 0.5);

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

        scanf("%d", &Qt);
        for(int i = 1; i <= Qt; i++) {
            scanf("%d%d", &Q[i].L, &Q[i].R);
            Q[i].id = i;
        }
        sort(Q + 1, Q + 1 + Qt);

        solve();
        for(int i = 1; i <= Qt; i++) {
            printf("%I64d\n", ans[i]);
        }
    }
    return 0;
}
```

13.表达式树

```
const int MX = 1e5 + 5;

char op[MX];
int lch[MX], rch[MX], s[MX], r;

int build(char *S, int L, int R) {
    int c[] = { -1, -1}, p = 0, u;

    int sum = 0, sign = true;
    for(int i = L; i <= R; i++) {
        if(isdigit(S[i])) sum = sum * 10 + S[i] - '0';
        else {
            sign = false;
            break;
        }
    }
```

```
    if(sign) {
        u = ++r;
        op[u] = '.';
        s[u] = sum;
        return u;
    }

    for(int i = L; i <= R; i++) {
        switch(S[i]) {
        case '(': p++; break;
        case ')': p--; break;
        case '+': case '-': if(!p) c[0] = i; break;
        case '*': case '/': if(!p) c[1] = i; break;
        }
    }

    if(c[0] < 0) c[0] = c[1];
    if(c[0] < 0) u = build(S, L + 1, R - 1);
    else {
        u = ++r;
        op[u] = S[c[0]];
        lch[u] = build(S, L, c[0] - 1);
        rch[u] = build(S, c[0] + 1, R);
    }
    return u;
}

double solve(int u) {
    if(op[u] == '.') return s[u];
    double al = solve(lch[u]), ar = solve(rch[u]);
    switch(op[u]) {
    case '+': return al + ar;
    case '-': return al - ar;
    case '*': return al * ar;
    case '/': return al / ar;
    }
}
```

14.树链剖分

```
/*点更新*/
int fa[MX], top[MX], siz[MX], son[MX], dep[MX], id[MX], rear;
/*fa 父节点，top 重链开头起点，siz 子树大小，son 重儿子，dep 深度，id 新编号*/

/*第一次 DFS 找到重边，并维护好 siz,son,fa,dep*/
void DFS1(int u, int f, int d) {
    fa[u] = f; dep[u] = d;
    son[u] = 0; siz[u] = 1;
    for(int i = Head[u]; ~i; i = E[i].nxt) {
        int v = E[i].v;
        if(v == f) continue;
        DFS1(v, u, d + 1);
        siz[u] += siz[v];
        if(siz[son[u]] < siz[v]) {
            son[u] = v;
        }
    }
}
/*将重边编号好，维护 id*/
void DFS2(int u, int tp) {
    top[u] = tp;
    id[u] = ++rear;
    if(son[u]) DFS2(son[u], tp);
```

```
    for(int i = Head[u]; ~i; i = E[i].nxt) {
        int v = E[i].v;
        if(v == fa[u] || v == son[u]) continue;
        DFS2(v, v);
    }
}
/*用来给点编号，以及建立线段树，要用 id 编号建树*/
void HLD_presolve() {
    rear = 0;
    DFS1(1, 0, 1);
    DFS2(1, 1);
    for(int i = 1; i <= rear; i++) {
        A[id[i]] = B[i];
    }
    build(1, rear, 1);
}
/*修改，要注意使用 id 编号修改*/
void HLD_update(int x, int d) {
    update(id[x], d, 1, rear, 1);
}
/*路径查询*/
int HLD_query(int u, int v) {
    int tp1 = top[u], tp2 = top[v];
    int sum = 0;
    while(tp1 != tp2) {
        if(dep[tp1] < dep[tp2]) {
            swap(u, v);
            swap(tp1, tp2);
        }
        sum += query(id[tp1], id[u], 1, rear, 1);
        u = fa[tp1]; tp1 = top[u];
    }

    if(dep[u] > dep[v]) swap(u, v);
    sum += query(id[u], id[v], 1, rear, 1);
    return sum;
}


/*边更新*/

/*节点 1 不使用，建树要小心
一般边使用更深的那个点的 id 编号来表示
*/
void HLD_presolve() {
    rear = 0;
    DFS1(1, 0, 1);
    DFS2(1, 1);
    for(int i = 0; i < 2 * (rear - 1); i += 2) {
        int u = E[i].u, v = E[i].v;
        if(dep[u] < dep[v]) swap(u, v);
        A[id[u]] = E[i].cost;
    }
    A[1] = -INF;
    build(1, rear, 1);
}
/*找到对应边的更深的点的 id 编号*/
void HLD_update(int x, int d) {
    x = (x - 1) * 2;
    int u = E[x].u, v = E[x].v;
    if(dep[u] < dep[v]) swap(u, v);
    update(id[u], d, 1, rear, 1);
}
```

```
/*注意最后一个查询与单点更新的区别以及 u==v 就需要返回 x*/
int HLD_query(int u, int v) {
    int tp1 = top[u], tp2 = top[v], ans = -INF;
    while(tp1 != tp2) {
        if(dep[tp1] < dep[tp2]) {
            swap(u, v);
            swap(tp1, tp2);
        }
        ans = max(ans, query(id[tp1], id[u], 1, rear, 1));
        u = fa[tp1]; tp1 = top[u];
    }
    if(u == v) return ans;
    if(dep[u] > dep[v]) swap(u, v);
    ans = max(ans, query(id[son[u]], id[v], 1, rear, 1));
    return ans;
}

/*路径合并更新*/
int HLD_query(int u, int v) {
    int tp1 = top[u], tp2 = top[v], ret = 0;
    int lastla[2] = { -1, -1}, lastra[2] = { -1, -1}, la[2] = { -1, -1}, ra[2] = { -1, -1};
    while(tp1 != tp2) {
        if(dep[tp1] >= dep[tp2]) {
            ret += query(id[tp1], id[u], 1, rear, 1, la[0], ra[0]);
            if(ra[0] == lastla[0]) ret--;
            lastla[0] = la[0]; lastra[0] = ra[0];
            u = fa[tp1]; tp1 = top[u];
        } else {
            ret += query(id[tp2], id[v], 1, rear, 1, la[1], ra[1]);
            if(ra[1] == lastla[1]) ret--;
            lastla[1] = la[1]; lastra[1] = ra[1];
            v = fa[tp2]; tp2 = top[v];
        }
    }
    if(dep[u] <= dep[v]) {
        ret += query(id[u], id[v], 1, rear, 1, la[1], ra[1]);
        if(ra[1] == lastla[1]) ret--;
        if(la[1] == lastla[0]) ret--;
    } else {
        ret += query(id[v], id[u], 1, rear, 1, la[0], ra[0]);
        if(ra[0] == lastla[0]) ret--;
        if(la[0] == lastla[1]) ret--;
    }
    return ret;
}
```

15.线段树扫描线

```
/*基本模板*/
int const MX = 1e3 + 5;

int rear, cnt[MX << 2];
double A[MX], S[MX << 2];

struct Que {
    int d;
    double top, L, R;
    Que() {}
    Que(double _top, double _L, double _R, int _d) {
        top = _top; L = _L; R = _R; d = _d;
    }
    bool operator<(const Que &b)const {
        return top < b.top;
```

```
    }
} Q[MX];

int BS(double x) {
    int L = 1, R = rear, m;
    while(L <= R) {
        m = (L + R) >> 1;
        if(A[m] == x) return m;
        if(A[m] > x) R = m - 1;
        else L = m + 1;
    }
    return -1;
}

void push_up(int l, int r, int rt) {
    if(cnt[rt]) S[rt] = A[r + 1] - A[l];
    else if(l == r) S[rt] = 0;
    else S[rt] = S[rt << 1] + S[rt << 1 | 1];
}

void update(int L, int R, int d, int l, int r, int rt) {
    if(L <= l && r <= R) {
        cnt[rt] += d;
        push_up(l, r, rt);
        return;
    }

    int m = (l + r) >> 1;
    if(L <= m) update(L, R, d, lson);
    if(R > m) update(L, R, d, rson);
    push_up(l, r, rt);
}

int main() {
    int n, ansk = 0;
    //freopen("input.txt", "r", stdin);
    while(~scanf("%d", &n), n) {
        rear = 0;
        memset(cnt, 0, sizeof(cnt));
        memset(S, 0, sizeof(S));

        for(int i = 1; i <= n; i++) {
            double x1, y1, x2, y2;
            scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
            Q[i] = Que(y1, x1, x2, 1);
            Q[i + n] = Que(y2, x1, x2, -1);

            A[++rear] = x1; A[++rear] = x2;
        }
        sort(Q + 1, Q + 1 + 2 * n);
        sort(A + 1, A + 1 + rear);
        rear = unique(A + 1, A + 1 + rear) - A - 1;

        double ans = 0, last = 0;
        for(int i = 1; i <= 2 * n; i++) {
            ans += (Q[i].top - last) * S[1];
            update(BS(Q[i].L), BS(Q[i].R) - 1, Q[i].d, root);
            last = Q[i].top;
        }
        printf("Test case #%d\n", ++ansk);
        printf("Total explored area: %.2lf\n\n", ans);
    }
    return 0;
```

```
}

/*只覆盖一次*/
void push_up(int l, int r, int rt) {
    if(cnt[rt]) {
        S1[rt] = A[r + 1] - A[l];
        if(cnt[rt] == 1) S2[rt] = S1[rt] - S1[rt << 1] - S1[rt << 1 | 1];
        else S2[rt] = 0;
    } else if(l == r) S1[rt] = S2[rt] = 0;
    else {
        S1[rt] = S1[rt << 1] + S1[rt << 1 | 1];
        S2[rt] = S2[rt << 1] + S2[rt << 1 | 1];
    }
}

/*覆盖次数>=2*/
void push_up(int l, int r, int rt) {
    if(cnt[rt]) {
        S1[rt] = A[r + 1] - A[l];
        if(cnt[rt] == 1) {
            S2[rt] = S1[rt << 1] + S1[rt << 1 | 1];
        } else S2[rt] = S1[rt];
    } else if(l == r) S1[rt] = S2[rt] = 0;
    else {
        S1[rt] = S1[rt << 1] + S1[rt << 1 | 1];
        S2[rt] = S2[rt << 1] + S2[rt << 1 | 1];
    }
}

/*按优先级覆盖*/
void push_up(int l, int r, int rt) {
    if(cnt[rt][3]) {
        S[rt][1] = S[rt][2] = 0;
        S[rt][3] = A[r + 1] - A[l];
    } else if(cnt[rt][2]) {
        S[rt][3] = S[rt << 1][3] + S[rt << 1 | 1][3];
        S[rt][2] = A[r + 1] - A[l] - S[rt][3];
        S[rt][1] = 0;
    } else if(cnt[rt][1]) {
        S[rt][3] = S[rt << 1][3] + S[rt << 1 | 1][3];
        S[rt][2] = S[rt << 1][2] + S[rt << 1 | 1][2];
        S[rt][1] = A[r + 1] - A[l] - S[rt][3] - S[rt][2];
    } else if(l == r) S[rt][1] = S[rt][2] = S[rt][3] = 0;
    else {
        S[rt][1] = S[rt << 1][1] + S[rt << 1 | 1][1];
        S[rt][2] = S[rt << 1][2] + S[rt << 1 | 1][2];
        S[rt][3] = S[rt << 1][3] + S[rt << 1 | 1][3];
    }
}
```

16.树状数组

```
struct BIT {
    int cid[MX], tim;
    int cnt[MX], n;
    void init(int _n) {
        n = _n; tim++;
    }
    void clear() {
        tim++;
    }
    void update(int p, int x) {
        for(; p <= n; p += p & -p) {
```

```
            if(cid[p] != tim) {
                cid[p] = tim;
                cnt[p] = 0;
            }
            cnt[p] += x;
        }
    }
    int sum(int p) {
        int ret = 0;
        for(; p; p -= p & -p) {
            if(cid[p] == tim) {
                ret += cnt[p];
            }
        }
        return ret;
    }
    int query(int l, int r) {
        if(l > r) return 0;
        return sum(r) - sum(l - 1);
    }
} bit;
```

17.树状数组第 k 大

```
int siz[MX], n;
void add(int x) {
    for(; x <= n; x += x & -x) {
        siz[x]++;
    }
}
int kth(int k) {
    int cur = 0;
    for(int i = 1 << 20; i; i >>= 1) {
        if(cur + i <= n && k - siz[cur + i] > 0) {
            k -= siz[cur + i];
            cur += i;
        }
    }
    return cur + 1;
}
```

18.Splay

```
int size[MX];
int num[MX], col[MX], n, m;
int son[MX][2], fa[MX], root, sz;
void Link(int x, int y, int c) {
    fa[x] = y; son[y][c] = x;
}
void push_up(int rt) {
    size[rt] = size[son[rt][0]] + size[son[rt][1]] + 1;
}
void push_down(int rt) {
    if(col[rt]) {
        col[son[rt][0]] ^= 1;
        col[son[rt][1]] ^= 1;
        swap(son[rt][0], son[rt][1]);
        col[rt] = 0;
    }
}
void Rotate(int x, int c) {
    int y = fa[x];
    push_down(y); push_down(x);
    Link(x, fa[y], son[fa[y]][1] == y);
```

```
    Link(son[x][!c], y, c);
    Link(y, x, !c);
    push_up(y);
}
/*把节点 x 旋转到 g 的下面*/
void Splay(int x, int g) {
    push_down(x);
    while(fa[x] != g) {
        int y = fa[x], cx = son[y][1] == x, cy = son[fa[y]][1] == y;
        if(fa[y] == g) Rotate(x, cx);
        else {
            if(cx == cy) Rotate(y, cy);
            else Rotate(x, cx);
            Rotate(x, cy);
        }
    }
    push_up(x);
    if(!g) root = x;
}
void NewNode(int f, int &rt) {
    rt = ++sz;
    fa[rt] = f, size[rt] = 1;
    son[rt][0] = son[rt][1] = col[rt] = 0;
}
/*把第 k 个找出来，放到 g 的下面*/
int Select(int k, int g) {
    int rt = root;
    while(size[son[rt][0]] != k) {
        if(size[son[rt][0]] > k) rt = son[rt][0];
        else k -= size[son[rt][0]] + 1, rt = son[rt][1];
        push_down(rt);
    }
    Splay(rt, g);
    return rt;
}
void Build(int l, int r, int &rt, int f) {
    if(l > r) return;
    int m = (l + r) >> 1, t;
    NewNode(f, rt); num[rt] = m;
    Build(l, m - 1, son[rt][0], rt);
    Build(m + 1, r, son[rt][1], rt);
    push_up(rt);
}
void Prepare(int n) {
    sz = 0;
    NewNode(0, root); num[1] = 0;
    NewNode(root, son[root][1]); num[2] = 0;
    Build(1, n, son[2][0], 2);
    Splay(3, 0);
}
void Print(int rt, int &DFN){
    if(!rt) return;
    push_down(rt);
    Print(son[rt][0], DFN);
    if(num[rt]) printf("%d%c", num[rt], ++DFN == n ? '\n' : ' ');
    Print(son[rt][1], DFN);
}
void Flip(int l, int r){
    Select(l - 1, 0);
    Select(r + 1, root);
    col[son[son[root][1]][0]] ^= 1;
}
/*剪断[a,b]放到 c 后面*/
```

```
void Cut(int a, int b, int c){
    Select(a - 1, 0);
    Select(b + 1, root);
    int w = son[son[root][1]][0];
    son[son[root][1]][0] = 0;
    Splay(son[root][1], 0);
    Select(c, 0);
    Select(c + 1, root);
    son[son[root][1]][0] = w;
    Splay(son[root][1], 0);
}
/*平衡树操作*/

void NewNode(int f, int x, int &rt) {
    rt = ++sz;
    fa[rt] = f, size[rt] = 1;
    son[rt][0] = son[rt][1] = 0;
    num[rt] = x;
}
int Kth(int k) {
    int rt = root;
    while(size[son[rt][0]] != k) {
        if(size[son[rt][0]] > k) rt = son[rt][0];
        else k -= size[son[rt][0]] + 1, rt = son[rt][1];
    }
    Splay(rt, 0);
    return num[rt];
}
void Prepare(int n) {
    sz = 0;
    NewNode(0, -INF, root);
    NewNode(root, INF, son[root][1]);
    push_up(root);
}
void Insert(int x) {
    int rt = root;
    while(true) {
        int nxt = x > num[rt];
        if(!son[rt][nxt]) {
            NewNode(rt, x, son[rt][nxt]);
            Splay(sz, 0); return;
        }
        rt = son[rt][nxt];
    }
}
```

19.DLX 覆盖

```
/*精确覆盖*/
struct DLX {
    int m, n;
    int H[MX], S[MX];
    int Row[MN], Col[MN], rear;
    int L[MN], R[MN], U[MN], D[MN];

    void Init(int _m, int _n) {
        m = _m; n = _n;
        rear = n;
        for(int i = 0; i <= n; i++) {
            S[i] = 0;
            L[i] = i - 1;
            R[i] = i + 1;
            U[i] = D[i] = i;
```

```
    }
    L[0] = n; R[n] = 0;
    for(int i = 1; i <= m; i++) {
        H[i] = -1;
    }
}

void Link(int r, int c) {
    int rt = ++rear;
    Row[rt] = r; Col[rt] = c; S[c]++;

    D[rt] = D[c]; U[D[c]] = rt;
    U[rt] = c; D[c] = rt;
    if(H[r] == -1) {
        H[r] = L[rt] = R[rt] = rt;
    } else {
        int id = H[r];
        R[rt] = R[id]; L[R[id]] = rt;
        L[rt] = id; R[id] = rt;
    }
}

void Remove(int c) {
    R[L[c]] = R[c]; L[R[c]] = L[c];
    for(int i = D[c]; i != c; i = D[i]) {
        for(int j = R[i]; j != i; j = R[j]) {
            D[U[j]] = D[j]; U[D[j]] = U[j];
            S[Col[j]]--;
        }
    }
}

void Resume(int c) {
    for(int i = U[c]; i != c; i = U[i]) {
        for(int j = L[i]; j != i; j = L[j]) {
            D[U[j]] = U[D[j]] = j;
            S[Col[j]]++;
        }
    }
    R[L[c]] = L[R[c]] = c;
}

bool Dance(int cnt) {
    if(R[0] == 0) return true;

    int c = R[0];
    for(int i = R[0]; i != 0; i = R[i]) {
        if(S[i] < S[c]) c = i;
    }

    Remove(c);
    for(int i = D[c]; i != c; i = D[i]) {
        for(int j = R[i]; j != i; j = R[j]) Remove(Col[j]);

        int r = Row[i];
        /*保存方案*/
        if(Dance(cnt + 1)) return true;

        for(int j = L[i]; j != i; j = L[j]) Resume(Col[j]);
    }
    Resume(c);
    return false;
}
```

```
} G;


/*重复覆盖*/
void Remove(int c) {
    for(int i = D[c]; i != c; i = D[i]) {
        R[L[i]] = R[i]; L[R[i]] = L[i];
    }
}

void Resume(int c) {
    for(int i = U[c]; i != c; i = U[i]) {
        R[L[i]] = L[R[i]] = i;
    }
}

int h() {
    int ret = 0;
    memset(vis, 0, sizeof(vis));
    for(int c = R[0]; c != 0; c = R[c]) {
        if(!vis[c]) {
            ret++;
            vis[c] = 1;
            for(int i = D[c]; i != c; i = D[i]) {
                for(int j = R[i]; j != i; j = R[j]) {
                    vis[Col[j]] = 1;
                }
            }
        }
    }
    return ret;
}

void Dance(int cnt) {
    if(cnt + h() >= ans) return;
    if(R[0] == 0) {
        ans = min(ans, cnt);
        return;
    }

    int c = R[0];
    for(int i = R[0]; i != 0; i = R[i]) {
        if(S[i] < S[c]) c = i;
    }

    for(int i = D[c]; i != c; i = D[i]) {
        Remove(i);
        for(int j = R[i]; j != i; j = R[j]) Remove(j);
        Dance(cnt + 1);
        for(int j = L[i]; j != i; j = L[j]) Resume(j);
        Resume(i);
    }
```

动态规划

1. TSP

```
//W 是距离,n 是除了起点以外的数量，0 为原点
int TSP() {
    memset(dp, 0x3f, sizeof(dp));
```

```
    for(int S = 0; S <= (1 << n) - 1; S++) {
        for(int i = 1; i <= n; i++) {
            if(S & (1 << (i - 1))) {
                if(S == (1 << (i - 1))) dp[i][S] = W[0][i];
                else for(int j = 1; j <= n; j++) {
                        if(S & (1 << (j - 1)) && j != i) {
                            dp[i][S] = min(dp[i][S], dp[j][S ^ (1 << (i - 1))] + W[j][i]);
                        }
                    }
            }
        }
    }

    int ret = INF;
    for(int i = 1; i <= n; i++) {
        ret = min(ret, dp[i][(1 << n) - 1] + W[0][i]);
    }

    /*
    若不需要回到起点，只需要全部走完，那么直接这样写
    int ret=INF;
    for(int i=1;i<=n;i++){
        ret=min(ret,dp[i][(1<<n)-1]);
    }
    也就是说不需要加上了那 W[0][i]而已
    */
    return ret;
}
```

2. 四边形不等式优化的石子合并

```
S[0] = 0;
for(int i = 1; i <= n; i++) {
    scanf("%d", &t);
    dp[i][i] = 0;
    K[i][i] = i;
    S[i] = S[i - 1] + t;
}

for(int l = 2; l <= n; l++) {
    for(int i = 1; i <= n - l + 1; i++) {
        for(int j = K[i][i + l - 2]; j <= K[i + 1][i + l - 1]; j++) {
            int temp = dp[i][j] + dp[j + 1][i + l - 1] + S[i + l - 1] - S[i - 1];
            if(temp < dp[i][i + l - 1]) {
                dp[i][i + l - 1] = temp;
                K[i][i + l - 1] = j;
            }
        }
    }
}
printf("%d\n", dp[1][n]);
```

3. 斜率优化(凸包)

如果最后的表达式中，得到 k > s，k 表示斜率，s 为某个数
那么我们就维护上凸包。
从左往右的上凸包

```
struct Point {
    LL x, y;
    Point() {}
    Point(LL _x, LL _y) {
```

```
        x = _x; y = _y;
    }
    Point operator-(const Point &P)const {
        return Point(x - P.x, y - P.y);
    }
    LL operator*(const Point &P)const {
        return x * P.y - y * P.x;
    }
} P[MX], W[MX];
LL A[MX];
int n, sz;
LL solve() {
    LL ret = 0; sz = 0;
    for(int i = 1; i <= n; i++) {
        while(sz >= 2 && (P[i] - W[sz]) * (W[sz] - W[sz - 1]) <= 0) sz--;
        W[++sz] = P[i];
        int l = 1, r = sz, m1, m2;
        while(l < r) {
            m1 = (2 * l + r) / 3;
            m2 = (l + 2 * r + 2) / 3;
            if(f(i, W[m1].x) < f(i, W[m2].x)) l = m1 + 1;
            else r = m2 - 1;
        }
        ret = max(ret, f(i, W[l].x));
    }
}
```

从右往左的上凸包

```
for(int i = n; i >= 1; i--) {
        while(sz >= 2 && (P[i] - W[sz]) * (W[sz] - W[sz - 1]) >= 0) sz--;
```

如果最后的表达式中，得到 k < s，k 表示斜率，s 为某个数
那么我们就维护下凸包。
从左往右的下凸包

```
for(int i = 1; i <= n; i++) {
        while(sz >= 2 && (P[i] - W[sz]) * (W[sz] - W[sz - 1]) >= 0) sz--;
```

从右往左的下凸包

```
for(int i = n; i >= 1; i--) {
        while(sz >= 2 && (P[i] - W[sz]) * (W[sz] - W[sz - 1]) <= 0) sz--;
```

对于是处理前缀的情况，假如题目要求得到

$$max(S1[r]-S1[l-1]-(l-1)*(S2[r]-S2[l-1])$$

设$l1<l2$,令$f(l1)<(l2)$，可以得到

$$S2[r]<\frac{((l2-1)S2[l2]-S1[l2-1])-((l1-1)S2[l1]-S1[l1-1])}{l2-l1}$$

如果我们把点当作$(i,(i-1)*S2[i-1]-S1[i-1])$，那么其实表达的意思就是，这个位置是我们选择的左区间l位置。

如果我们把点当作$(i,i*S2[i]-S1[i])$,那么这个位置i代表的是$l-1$位置。

（终于能无脑写斜率优化了hhhh

4. 斜率优化(单调队列)

通常斜率优化的代价都是某个平方之类的
假设 k<j<i，假如 j 比 k 更优，列出式子然后能拆成(f(j)-f(k))/(g(j)-f(k))<h(i)的形式
只要是这个形式，且 h(i)函数单调递增，就能用斜率优化
最后单调队列中，左下角是队列首，右上角是队列尾
从首到尾组成的点斜率越来越大

```
int Q[MX], c, r;
/*分子*/
LL getup(int i, int j) {
```

```
    return dp[j - 1] + A[j] * A[j] - (dp[i - 1] + A[i] * A[i]);
}
/*分母*/
LL getdown(int i, int j) {
    return A[j] - A[i];
}
/*计算 dp 的值*/
LL getdp(int i, int j) {
    return dp[j - 1] + (A[i] - A[j]) * (A[i] - A[j]) + w;
}
for(int i = 1; i <= n; i++) {
    while(r - c + 1 >= 2 && getup(Q[c], Q[c + 1]) <= 2 * A[i]*getdown(Q[c], Q[c + 1])) c++;
    dp[i] = min(getdp(i, Q[c]), dp[i - 1] + w);
    while(r - c + 1 >= 2 && getup(Q[r], i)*getdown(Q[r - 1], Q[r]) <= getdown(Q[r], i)*getup(Q[r
- 1], Q[r])) r--;
    Q[++r] = i;
}
```

5. 往子集传递值

设 dp[w][s]表示对于低位的 w 位，1 必须是 1，0 可以是其他的，对于另外 20-w 位，1 就是 1，0 就是 0。
那么我们可以得到一个转移方程，如果 s>>(w-1)&1,那么等于 dp[w-1][s]
否则，就等于 dp[w-1][s|(1<<(w-1))]

```
/*初始值存在 dp[0][t]中*/
for(int i = 1; i <= 20; i++) {
    for(int s = 0; s <= w; s++) {//w 所有的二进制状态
        if(s >> (i - 1) & 1) dp[i][s] = dp[i - 1][s];
        else dp[i][s] = dp[i - 1][s] + dp[i - 1][s + (1 << (i - 1))];
    }
}
/*答案存在 dp[20][t]中*/
```

6. 数位 dp

```
inline int func(int s, int x) {
    x = 9 - x;
    for(int i = 0; i <= 9; i++) {
        if(i >= x && (s >> i & 1)) {
            return s ^ (1 << i) ^ (1 << x);
        }
    }
    return s ^ (1 << x);
}
LL DFS(int p, int s, bool limits) {
    if(p == 1) return __builtin_popcount(s);//通常返回 1
    if(!limits && dp[p][s] != -1) return dp[p][s];
    p--; LL ret = 0;
    int bound = limits ? A[p] : 9;
    for(int i = 0; i <= bound; i++) {
        ret += DFS(p, func(s, i), limits & (i == A[p]));
    }
    if(!limits) dp[p + 1][s] = ret;
    return ret;
}
void presolve(LL n) {
    w = 0;
    while(n) {
        A[++w] = n % 10;
        n /= 10;
    }
}
LL solve(LL n) {
    if(n == 0) return 0;
    presolve(n);
```

```
    LL ret = 0;
    for(int i = 1; i <= w; i++) {
        int ed = (i == w ? A[i] : 9);
        for(int j = 1; j <= ed; j++) {
            ret += DFS(i, func(0, j), (i == w && j == A[i]));
        }
    }
    return ret;
}
```

7. 区间内 2 个数位异或等于特定值

```
/*复杂度 O(60*2*2)
可以求出 t1 属于[1,a],t2 属于[1,b]
t1^t2=x 的(t1,t2)的点对数
*/
const int MX = 1e2;
LL dp[MX][2][2];
int na[MX], nb[MX], nx[MX];

LL S(LL a, LL b, LL x) {
    memset(dp, 0, sizeof(dp));
    for(int i = 63; i >= 0; i--) {
        na[i] = a >> i & 1;
        nb[i] = b >> i & 1;
        nx[i] = x >> i & 1;
    }

    dp[63][1][1] = 1;
    for(int i = 62; i >= 0; i--) {
        if(na[i] ^ nb[i] == nx[i]) {
            dp[i][1][1] += dp[i + 1][1][1];
        }

        dp[i][1][0] += dp[i + 1][1][0];
        if(nb[i] && na[i] == nx[i]) dp[i][1][0] += dp[i + 1][1][1];

        dp[i][0][1] += dp[i + 1][0][1];
        if(na[i] && nb[i] == nx[i]) dp[i][0][1] += dp[i + 1][1][1];

        dp[i][0][0] += dp[i + 1][0][0] * 2;
        if(na[i]) dp[i][0][0] += dp[i + 1][1][0];
        if(nb[i]) dp[i][0][0] += dp[i + 1][0][1];
        if(na[i] && nb[i] && !nx[i]) dp[i][0][0] += dp[i + 1][1][1];
    }
    return dp[0][0][0] + dp[0][0][1] + dp[0][1][0] + dp[0][1][1];
}
```

博弈

1. 斐波那契博弈

题意：1 堆石子 n 个，第一个人可以取任意个数但不能全部取完，以后每次拿的个数不能超过上一次对手拿的个数的 2 倍，轮流拿石子，问先手是否必赢

思路：斐波那契博弈，后手赢的情况的数字会呈现斐波那契数列。

2. 威佐夫博弈

题意：轮流取石子。1.在一堆中取任意个数.2.在两堆中取相同个数。最后取完的人胜利，问先手是否必赢

思路：威佐夫博弈博弈，满足黄金分割，且每个数字只会出现一次。

```
if(a >= b) swap(a, b);
int k = b - a;
```

```
int x = (sqrt(5.0) + 1) / 2 * k, y = x + k;
if(a == x && b == y) printf("0\n");
else printf("1\n");
```

3. 巴什博弈

题意：两人竞拍，每次加价的价格在[1,n]范围内，第一次>=m 的赢
思路：巴什博弈，当 m%(n+1)!=0 时，先手赢，否则后手赢

4. Anti-num 博弈

SG 函数的求法一模一样，最后如果只有一堆，也能用 SJ 定理
如果为 Anti-Nim 游戏，如下情况先手胜
SG 异或和为 0，且单个游戏的 SG 全部<=1
SG 异或不为 0，且存在单个游戏的 SG>1,即<=1 的个数不等于独立游戏个数

5. Nim 博弈

Nim 游戏相当于把独立游戏分开计算 SG 函数，然后再用位异或
Sg[u]=Mex({后继的集合})相当于取出最小的集合中不存在的数字，可以发现 mex 的值总是比后继的个数要少
而且 vis 数组通常都是开在函数内部，不开在全局变量中，防止冲突。

其他杂类

6. 大数

```
const int MX = 2500;
const int MAXN = 9999;
const int DLEN = 4;

/*已重载>+-*/%和 print*/
class Big {
public:
    int a[MX], len;
    Big(const int b = 0) {
        int c, d = b;
        len = 0;
        memset(a, 0, sizeof(a));
        while(d > MAXN) {
            c = d - (d / (MAXN + 1)) * (MAXN + 1);
            d = d / (MAXN + 1);
            a[len++] = c;
        }
        a[len++] = d;
    }
    Big(const char *s) {
        int t, k, index, L, i;
        memset(a, 0, sizeof(a));
        L = strlen(s);
        len = L / DLEN;
        if(L % DLEN) len++;
        index = 0;
        for(i = L - 1; i >= 0; i -= DLEN) {
            t = 0;
            k = i - DLEN + 1;
            if(k < 0) k = 0;
            for(int j = k; j <= i; j++) {
                t = t * 10 + s[j] - '0';
            }
            a[index++] = t;
        }
    }
```

```
    Big operator/(const int &b)const {
        Big ret;
        int i, down = 0;
        for(int i = len - 1; i >= 0; i--) {
            ret.a[i] = (a[i] + down * (MAXN + 1)) / b;
            down = a[i] + down * (MAXN + 1) - ret.a[i] * b;
        }
        ret.len = len;
        while(ret.a[ret.len - 1] == 0 && ret.len > 1) ret.len--;
        return ret;
    }
    bool operator>(const Big &T)const {
        int ln;
        if(len > T.len) return true;
        else if(len == T.len) {
            ln = len - 1;
            while(a[ln] == T.a[ln] && ln >= 0) ln--;
            if(ln >= 0 && a[ln] > T.a[ln]) return true;
            else return false;
        } else return false;
    }
    Big operator+(const Big &T)const {
        Big t(*this);
        int i, big;
        big = T.len > len ? T.len : len;
        for(i = 0; i < big; i++) {
            t.a[i] += T.a[i];
            if(t.a[i] > MAXN) {
                t.a[i + 1]++;
                t.a[i] -= MAXN + 1;
            }
        }
        if(t.a[big] != 0) t.len = big + 1;
        else t.len = big;
        return t;
    }
    Big operator-(const Big &T)const {
        int i, j, big;
        bool flag;
        Big t1, t2;
        if(*this > T) {
            t1 = *this;
            t2 = T;
            flag = 0;
        } else {
            t1 = T;
            t2 = *this;
            flag = 1;
        }
        big = t1.len;
        for(i = 0; i < big; i++) {
            if(t1.a[i] < t2.a[i]) {
                j = i + 1;
                while(t1.a[j] == 0) j++;
                t1.a[j--]--;
                while(j > i) t1.a[j--] += MAXN;
                t1.a[i] += MAXN + 1 - t2.a[i];
            } else t1.a[i] -= t2.a[i];
        }
        t1.len = big;
        while(t1.a[t1.len - 1] == 0 && t1.len > 1) {
            t1.len--;
            big--;
```

```
        }
        if(flag) t1.a[big - 1] = 0 - t1.a[big - 1];
        return t1;
    }
    int operator%(const int &b)const {
        int i, d = 0;
        for(int i = len - 1; i >= 0; i--) {
            d = ((d * (MAXN + 1)) % b + a[i]) % b;
        }
        return d;
    }
    Big operator*(const Big &T) const {
        Big ret;
        int i, j, up, temp, temp1;
        for(i = 0; i < len; i++) {
            up = 0;
            for(j = 0; j < T.len; j++) {
                temp = a[i] * T.a[j] + ret.a[i + j] + up;
                if(temp > MAXN) {
                    temp1 = temp - temp / (MAXN + 1) * (MAXN + 1);
                    up = temp / (MAXN + 1);
                    ret.a[i + j] = temp1;
                } else {
                    up = 0;
                    ret.a[i + j] = temp;
                }
            }
            if(up != 0) {
                ret.a[i + j] = up;
            }
        }
        ret.len = i + j;
        while(ret.a[ret.len - 1] == 0 && ret.len > 1) ret.len--;
        return ret;
    }
    void print() {
        printf("%d", a[len - 1]);
        for(int i = len - 2; i >= 0; i--) printf("%04d", a[i]);
    }
};
```

7. pb_ds 大法

```
//可并堆测试
#include <ext/pb_ds/priority_queue.hpp>
void ceshi_1() {
    //binary_heap_tag 一般比 std::priority_queue 快
    //pairing_heap_tag 和 std::priority_queue 启发式合并时，速度差不多
    __gnu_pbds::priority_queue<int, less<int>, __gnu_pbds::pairing_heap_tag> Q1, Q2;
    Q1.push(1); Q1.push(2); Q1.push(3);
    Q2.push(1); Q2.push(2); Q2.push(3);
    Q1.join(Q2);//pairing_heap_tag 配对堆，zici O(1)合并
    while(!Q1.empty()) {
        std::printf("%d\n", Q1.top());
        Q1.pop();
    }
}

//平衡树测试
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
void ceshi_2() {
    //支持 join 和 split
```

```
    typedef __gnu_pbds::tree<int,
            __gnu_pbds::null_type,
            less<int>,
            __gnu_pbds::rb_tree_tag,
            __gnu_pbds::tree_order_statistics_node_update>
            qwb_set;
    qwb_set w;
    w.insert(1); w.insert(6); w.insert(3); w.insert(100);
    auto t = w.find_by_order(1);//查找第 x+1 小的值
    int sum = w.order_of_key(101); //比 x 小的有多少个元素
}
```

8. bitset

内存占 size/8 字节
bitset<128>s;定义 s 变量
s=100;//可以直接赋值
s="100010";//可以赋值字符串
s.set(p);//设置 p 位为 1
s.reset(p);//设置 p 位为 0
s.set();//全部位设置为 1
s.reset();//全部位设置为 0
s.count();//1 的个数
s.flip();//0 变 1，1 变 0，相当于~
可以直接使用~|^&符号

9. 蔡勒公式

```
Week=(Day + 2*Month + 3*(Month+1)/5 + Year + Year/4 - Year/100 + Year/400) % 7 + 1
```

i. 该公式中要把 1 月和 2 月分别当成上一年的 13 月和 14 月处理。
例如：2008 年 1 月 4 日要换成 2007 年 13 月 4 日带入公式。
“1”为星期 1，……，“7”为星期日。

10.第 k 小

```
LL l = 1, r = 4e18, m;
while(l <= r) {
    m = (l + r) >> 1;
    if(check(m) >= k) r = m - 1;
    else l = m + 1;
}
```
check(m)来求<=m 的个数

11.三分整数

```
while(l < r) {
    int m1 = (2 * l + r) / 3, m2 = (l + 2 * r + 2) / 3;
    if(f(m1) < f(m2)) l = m1 + 1;
    else r = m2 - 1;
}
```

12.求阶乘后缀 0 个数

```
int get(int n) {
  int z = 0;
  while (n > 0) {
    n /= 5;
    z += n;
  }
  return z;
}
```

13.DFS 构造矩阵

```
/*骨牌覆盖的构造方法*/
void DFS(int a, int b, int l) {
    if(l == m) {
        A[b][a] = 1;//a 对 b 的影响
        return;
    }
    DFS(a << 1, b << 1 | 1, l + 1);//往下放
    DFS(a << 1 | 1, b << 1, l + 1);//不放
    if(l + 2 <= m) DFS(a << 2 | 3, b << 2 | 3, l + 2);//往右放
}
```

14.手动扩栈

```
C++扩栈
#pragma comment(linker, "/STACK:102400000,102400000")
G++扩栈貌似不可以？
int Size = 256 << 20;
char *p = (char*)malloc(Size) + Size;
__asm__("movl %0, %%esp\n" :: "r"(p));
```

15.正常的读入挂

```
inline int read() {
    int ret = 0, c, f = 1;
    for(c = getchar(); !(isdigit(c) || c == '-'); c = getchar());
    if(c == '-') f = -1, c = getchar();
    for(; isdigit(c); c = getchar()) ret = ret * 10 + c - '0';
    if(f < 0) ret = -ret;
    return ret;
}
```

16.fread 读入挂

```
namespace IO {
    const int MX = 1e7; //1e7 占用内存 11000kb
    char buf[MX]; int c, sz;
    void begin() {
        c = 0;
        sz = fread(buf, 1, MX, stdin);
    }
    inline bool read(int &t) {
        while(c < sz && buf[c] != '-' && (buf[c] < '0' || buf[c] > '9')) c++;
        if(c >= sz) return false;
        bool flag = 0; if(buf[c] == '-') flag = 1, c++;
        for(t = 0; c < sz && '0' <= buf[c] && buf[c] <= '9'; c++) t = t * 10 + buf[c] - '0';
        if(flag) t = -t;
        return true;
    }
}
```

17.测试系统类型

```
void test_system() {
#ifdef __linux__
    for(;;);
#else
    int a = 0, b = 1 / a;
#endif
}
```

18.方阵的循环节

$$w(n,p) = \prod_{i=0}^{n-1}(p^n - p^i)$$

$$A^t = A^{t\%w(n,p)}$$

只有 p 为质数时候才成立。

19.字符串分割读入的

```
/*再乱写就剁手*/
bool read(int &cur, int len, bool sign = 0) {
    for(; cur < len && !check(buf[cur]); cur++);
    if(cur == len) return false;

    int tt = 0;
    for(; cur < len && check(buf[cur]); cur++) {
        if(sign) tmp[tt++] = to_lower(buf[cur]);
        else tmp[tt++] = buf[cur];
    }
    tmp[tt] = 0;
    return true;
}

int cur = 0, len = strlen(buf);
while(read(cur, len)) {
    //tmp 就是已经读入进来的
}
```

20.区间随机数生成

```
int Rand(int L, int R) {//区间内随机数生成函数
    return (LL)rand() * rand() % (R - L + 1) + L;
}
```

21.祖传头文件

```
#include <map>
#include <set>
#include <cmath>
#include <ctime>
#include <stack>
#include <queue>
#include <cstdio>
#include <cctype>
#include <bitset>
#include <string>
#include <vector>
#include <cstring>
#include <iostream>
#include <algorithm>
#include <functional>
#define fuck(x) cout<<"["<<x<<"]";
#define FIN freopen("input.txt","r",stdin);
#define FOUT freopen("output.txt","w+",stdout);
//#pragma comment(linker, "/STACK:102400000,102400000")
using namespace std;
typedef long long LL;
```

```
typedef pair<int, int> PII;
```