# 起手板子

```
// #pragma GCC optimize(2)

#include <bits/stdc++.h>

#define fi first
#define se second
#define mkp(x, y) make_pair((x), (y))
#define all(x) (x).begin(), (x).end()

using namespace std;

typedef long long LL;
typedef double db;
typedef pair<int, int> PII;

void solve() {
    //
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout << fixed;  // << setprecision(20); // double
    // freopen("i.txt", "r", stdin);
    // freopen("o.txt", "w", stdout);
    // time_t t1 = clock();
    int Tcase = 1;
    // cin >> Tcase; // scanf("%d", &Tcase);
    while (Tcase--)
        solve();
    // cout << "time: " << 1000.0 * (1.*(clock() - t1) / CLOCKS_PER_SEC) <<
"ms\n";
    return 0;
}
```

# 动态规划

## LIS

```
vector<int> stk = {a[0]};
int ans = 0;
for(int i = 1; i < n; i ++) {
    if(stk.back() < a[i]) stk.push_back(a[i]);
    else *lower_bound(all(stk), a[i]) = a[i];
    // upper_bound(all(stk), a[i])-stk.begin(); 为当前长度。
}
```

# 图论

# 网络流

## 常见模型建图方式

### 上下界网络流

- 无源汇可行流

  先满足下界，由于斜对称，所以，流出 $x$ 等价于 流入 $-x$ 。

```
for (int i = 1; i <= cnt; i++) {
    w[u] -= low[i], w[v] += low[i];
    // mncost += low[i] * w[i]; 费用流
    mxflow.addFlow(u[i], v[i], limit[i]-low[i], 0);
}

int tot = 0;

for (int i = 1; i <= n; i++) {
    if (w[i] > 0) {
        mxflow.addFlow(S, i, w[i]), tot += w[i];
    }
    else if (w[i] < 0) {
        mxflow.addFlow(i, T, -w[i]);
    }
}
bool ok = mxflow.dinic() == tot;
// 若在G'跑出的最大流等于斜对称的值，即 ok == true ，则代表有解。下为每条边的
for (int i = 1; i <= m; i++)
    cout << low[i] + e[(i-1)*2] << "\n";
```

- 有源汇可行流

  连一条 $T \to S$ 容量为 $+\infty$ 的边，转化为**无源汇可行流**。再按照**无源汇**的方式建出超级源/汇点，SS 和 TT 。

  当前流量为 $T \to S$ 的反向边 $S \to T$的 流量。

- 有源汇最大流

  结论：给定**任意**一组**可行流**，对其运行最大流算法，我们总能得到正确的最大流。这是因为最大流算法本身**支持撤销**，即退流操作。所以，无论初始的流函数 f 如何，只要 f **合法**，就一定能求出最大流。

  先求出**有源汇可行流**，并且记录下此时的流量 $f_1$ ；

  再二次调整，在 G' 的 SS 、 TT 和 $T \to S$ 流量为 $+\infty$ 撤去，这样就得到在 原图 G 上的一组**有源汇可行流**。

  然后，在撤回后的 G' 以 S 和 T 上跑最大流，得到**新增流量** $f_2$ ，所求为 $f_1 + f_2$ 。

  - 不要在求完**有源汇可行流**后回到 G 上跑最大流，这样此时流量下界是没有保证的；而在 G' 上跑，只要 $f_1$ 合法，求出的最大流必然也合法。

```
// 最后加 t -> s, INF，删除。
f.addFlow(t, s, INF);
if(f.dinic(ss, tt) != tot) return -1;
U f1 = f.e[f.h[s]];
f.h[s] = f.ne[f.h[s]], f.h[t] = f.ne[f.h[t]];
return f1 + f.dinic(s, t);
```

- 有源汇最小流

  根据 $S \to T$ 的最小流等于 $T \to S$ 最大流的相反数，**有源汇可行流**减去调整后**残留网络的最大流**，即 $f_1 - f_2$ 。

  ```
  return f1 - f.dinic(t, s);
  ```

- 有源汇费用流

  将最大流换成费用流，所有 SS 、 TT 相关的连边代价为 0 。

  初始费用为 $\sum b(u,v)w(u,v)$ ，求**可行流**时，加上 $SS \to TT$ 的最小费用最大流的费用，二次调整时，再加上产生的费用。

**最小割点**

求 S 和 T 不连通的最少代价。

点边互化，点拆成**入点**和**出点**，原图的边 $u_{out} \to v_{in}$ ，因为删点不删边，边权 $+\infty$ 。

答案为 $S_{out} \to T_{in}$ 的最小割。

**集合划分**

把物品划分到两个集合中，放到 A 集合代价 $a_i$ ，放到 B 集合代价 $b_i$ ，若 u 和 v 之间有边相连，且 u 和 v 在不同集合，代价 $c_{u,v}$ 。

S 向 i 连容量为 $a_i$ ，i 向 T 连容量为 $b_i$ 的边，u 向 v 连容量 $c_{u,v}$ 的边，求最小割。

- $a_i$，$b_i$ 出现负值。

  $a_i$，$b_i$ 同时加上 $w_i$ ，由于每个点一定会删掉一个边，划到一个集合里，所以再在最后求出的最小割中减去 $\sum w_i$ 。

- $c_{u,v}$ 出现负值。

  只有 $c_{u,v}$ 均为负值且要求代价最大化，才可做。

  此时将**所有边权**（包括所有的 $a_i, b_i$）取反。把求**代价**最大转化为求**贡献**最小。

- 输出方案==

**最大权闭合子图**

问题：对于所有边 (u, v)，点 u 在子图中则 点v 必在子图中，求子图的点权和最大值。

$$
\begin{aligned}
w_i > 0 \quad &\Rightarrow \quad add(S, i, w_i) \\
w_i < 0 \quad &\Rightarrow \quad add(i, T, -w_i) \\
\text{原图边}(u,v) \quad &\Rightarrow \quad add(u, v, INF)
\end{aligned}
$$

闭合子图和 $S$ 构成 集合 $V_S$，其余点和 $T$ 构成 集合 $V_T$。

最大权闭合子图的权 $W = \sum_{i \in V_S} w_i[w_i > 0] - \sum_{i \in V_S} |w_i|[w_i < 0]$;

最小割一定是简单割，最小割的值为 $c(S,T) = \sum_{i \in V_T} w_i[w_i > 0] + \sum_{i \in V_S} |w_i|[w_i < 0]$;

则点权和最大值 $W = \sum w_i[w_i > 0] - c(S,T)$，即`正权点之和 - 最小割`。

不要被限制关系迷惑了，如限制关系，其实反过来考虑就是闭合子图。

直接上这种建图方式即可。注意排除一些题中一些性质不合法的情况，如有些题不能有环或被环连出的边。

总是转化为`选 a 必须选 b 的形式`。

#### 有负环的费用流

用**上下界网络流**将负权边流满。

```
// 加边 仅为示例，上下界的情况可能最好再开个结构体，记带有上下界的边。
void add(int _u, int _v, int w, int c) { // 流量，费用
    if(c < 0) {
      u[++e] = _u, v[e] = _v, lo[e] = w, hi[e] = w, cst[e] = c;
      u[++e] = _v, v[e] = _u, lo[e] = 0, hi[e] = w, cst[e] = -c;
    }
    else u[++e] = _u, v[e] = _v, lo[e] = 0, hi[e] = w, cst[e] = c;
  }
```

#### 最大费用最大流

将所有权值取相反数，转化成**最小费用最大流**，遇到有负环的需要用上个解法。

#### 有先后顺序 / 流的时间顺序的图

把每个点在每个时间都建出来，按照限制设置前一个时间点的点和当前时间点的流的容量。

优化：一个时间点里的操作，就在当层操作。有时需要控制点数和边数，把不必要的点 / 边不建出来，不如没有操作的。

## 值得注意的地方

- 如果一组割对应了题目的一种方案，在求解最大流之后，一定不能将所有流满的边视作割边，而是将两端所在集合不同的边视作割边。

  具体求解方式：由于我们维护了在残留网络中的 dis[] 。若 dis[u] 存在，则 $u \in V_S$ ，否则 $v \in V_T$ 。
- **最大流**是可以反悔的，所以在加边的时候，不必要把边都清空，可以在跑出来的基础上，加上新跑出来的结果；

  但**费用流**是基于最短路，所以需要重新跑一遍。

## 模板

#### 网络流 Dinic - $O(n^2m)$

可处理 $10^4 - 10^5$ 规模的网络。

```
template<class U>
struct MaxFlow {
    int S, T;
    int idx, h[NN], ne[MM], ver[MM];
    U mxflow, e[MM];
    int d[NN], cur[NN];
    void init() {
        idx = 0, memset(h, -1, sizeof h);
    }
    void add(int a, int b, U c) {
        ver[idx] = b, e[idx] = c, ne[idx] = h[a], h[a] = idx ++;
    }
    void addFlow(int a, int b, U c1, U c2 = 0) {
        add(a, b, c1), add(b, a, c2);
    }
    bool bfs() {
        queue<int> q; q.push(S);
```

```
        memset(d, -1, sizeof d);
        d[S] = 0, cur[S] = h[S];
        while(!q.empty()) {
            int u = q.front(); q.pop();
            for(int i = h[u]; ~i; i = ne[i]) {
                int v = ver[i];
                if(d[v] == -1 && e[i] > 0) {
                    d[v] = d[u]+1;
                    cur[v] = h[v];
                    if(v == T) return true;
                    q.push(v);
                }
            }
        }
        return false;
    }
    U update(int u, U limit) {
        if(u == T) return limit;
        U flow = 0;
        for(int i = cur[u]; ~i && flow < limit; i = ne[i]) {
            cur[u] = i;
            int v = ver[i];
            if(d[v] == d[u]+1 && e[i] > 0) {
                U t = update(v, min(e[i], limit-flow));
                if(!t) d[v] = -1;
                flow += t;
                e[i] -= t;
                e[i^1] += t;
            }
        }
        return flow;
    }
    U dinic(int s, int t) {
        S = s, T = t;
        U flow;
        mxflow = 0;
        while(bfs())
            while(flow = update(S, INF))
                mxflow += flow;
        return mxflow;
    }
};
MaxFlow<int> mxflow;
```

```
/* M是总边数，不要忘记乘以2之类的。*/
/*******临时加边/容量***************
int tx = h[x], ty = h[y];
for(int z = 0; z < idx; z += 2)
    e[z] += e[z^1], e[z^1] = 0;
add(x, y, C), add(y, x, 0);
LL tres = dinic();
idx -= 2;
h[x] = tx, h[y] = ty;
******也可以先记录下来，然后记得用bool判定只加了一次
bool ok = false;
for(int z = 0; z < idx; z += 2) { // .....
    e[z] = rec[z] + (!ok && ver[z] == y && ver[z^1] == x ? C : 0);
```

```
    e[z^1] = 0;
    if(ver[z] == y && ver[z^1] == x) ok = true;
}*/
```

**上下界可行流 - Dinic**

```
// +dinic板子
template<class U>
struct BoundedFlow {
    struct Edge {
        int u, v; U low, high;
        Edge (int u_, int v_, U lo_, U hi_): u(u_), v(v_), low(lo_), high(hi_)
{}
    };
    U w[NN];
    vector<Edge> edges;
    void add(int a, int b, U lo, U hi) {
        edges.emplace_back(a, b, lo, hi);
    }
    int bf(int n, int s, int t, int ss, int tt) { // 无源汇去掉s和t
        memset(w, 0, sizeof w);
        auto &f = mxflow;
        for(auto [u, v, lo, hi]: edges) {
            w[u] -= lo, w[v] += lo;
            f.addFlow(u, v, hi-lo);
        }
        U tot = 0;
        for(int i = 1; i <= n; i ++) {
            if(w[i] > 0) {
                f.addFlow(ss, i, w[i]);
                tot += w[i];
            }
            else if(w[i] < 0) {
                f.addFlow(i, tt, -w[i]);
            }
        }
        f.addFlow(t, s, INF);
        if(f.dinic(ss, tt) != tot) return -1;
        U f1 = f.e[f.h[s]];
        f.h[s] = f.ne[f.h[s]], f.h[t] = f.ne[f.h[t]];
        return f1 + f.dinic(s, t); // 求最小流，f1 - f.dinic(t, s);
    }
};
BoundedFlow<int> bflow;

void solve() {
    int n, m, S, T; cin >> n >> m >> S >> T;
    for(int i = 0; i < m; i ++) {
        int a, b, c, d;
        cin >> a >> b >> c >> d;
        bflow.add(a, b, c, d);
    }
    mxflow.init();
    int res = bflow.bf(n, S, T, 0, n+1);
    if(res != -1) {
        cout << res << '\n';
    }
```

```
    else {
        cout << "please go home to sleep\n";
    }
}
```

```
// 可行流
for(int i = 0; i < m; i ++) {
    cout << bflow.edges[i].low + mxflow.e[i*2+1] << '\n';
}
```

```
// +费用流dinic板子
template<class U>
struct BoundedFlow {
    struct Edge {
        int u, v; U low, high, w;
        Edge (int u_, int v_, U lo_, U hi_, U w_): u(u_), v(v_), low(lo_),
high(hi_), w(w_) {}
    };
    U flow, cost, w[NN];
    vector<Edge> edges;
    void add(int a, int b, U lo, U hi, U ew) {
        edges.emplace_back(a, b, lo, hi, ew);
    }
    pair<U, U> bf(int n, int s, int t, int ss, int tt) {
        memset(w, 0, sizeof w);
        auto &f = mcmf;
        flow = 0, cost = 0;
        for(auto [u, v, lo, hi, ew]: edges) {
            w[u] -= lo, w[v] += lo;
            cost += lo * ew;
            f.addFlow(u, v, ew, hi-lo);
        }
        U tot = 0;
        for(int i = 1; i <= n; i ++) {
            if(w[i] > 0) {
                f.addFlow(ss, i, 0, w[i]);
                tot += w[i];
            }
            else if(w[i] < 0) {
                f.addFlow(i, tt, 0, -w[i]);
            }
        }
        f.addFlow(t, s, 0, INF);
        auto [f1, c1] = f.dinic(ss, tt);
        // if(f1 != tot) return {-1, -1};
        flow = f.e[f.h[s]], cost += c1;
        f.h[s] = f.ne[f.h[s]], f.h[t] = f.ne[f.h[t]];
        auto [f2, c2] = f.dinic(s, t);
        flow += f2, cost += c2;
        return {flow, cost};
    }
};
BoundedFlow<LL> bflow;

void solve() {
    int n, m, S, T; cin >> n >> m >> S >> T;
```

```
    for(int i = 0; i < m; i ++) {
        int a, b; LL c, d;
        cin >> a >> b >> c >> d;
        if(d > 0) bflow.add(a, b, 0, c, d);
        else {
            bflow.add(a, b, c, c, d);
            bflow.add(b, a, 0, c, -d);
        }
    }
    mcmf.init();
    auto [flow, cost] = bflow.bf(n, S, T, 0, n+1);
    cout << flow << ' ' << cost << '\n';
}
```

**费用流 SSP Dinic&SPFA - $O(nmf)$**

f 为最大流。

- 最小费用流：若 d[T] 为负则继续增广。
- 最小费用最大流：若 d[T] 不为 INF 则继续增广

SSP 在图中有负环的时候，要先消去负环。

```
template<class U>
struct MinCostMaxFlow { // By Dinic
    int S, T;
    int idx, h[NN], ne[MM], ver[MM];
    U mxflow, mncost, e[MM], w[MM], d[NN]; // d[]可以int，但注意INF的值是否爆int
    U cur[NN];
    bool vis[NN];
    void init() {
        idx = 0, memset(h, -1, sizeof h);
    }
    void add(int a, int b, U c, U d) {
        ver[idx] = b, e[idx] = c, w[idx] = d, ne[idx] = h[a], h[a] = idx ++;
    }
    void addFlow(int a, int b, U d, U c1, U c2 = 0) {
        add(a, b, c1, d), add(b, a, c2, -d);
    }
    bool spfa() {
        queue<int> q;
        q.push(S);
        memset(d, 0x3f, sizeof d);
        d[S] = 0, vis[S] = true;
        while(!q.empty()) {
            int u = q.front(); q.pop();
            cur[u] = h[u]; vis[u] = false;
            for(int i = h[u]; ~i; i = ne[i]) {
                int v = ver[i];
                if(d[v] > d[u] + w[i] && e[i] > 0) {
                    d[v] = d[u]+w[i];
                    if(!vis[v]) {
                        q.push(v);
                        vis[v] = true;
                    }
                }
            }
        }
```

```
        return d[T] < INF;
    }
    U update(int u, U limit) {
        if(u == T) return limit;
        vis[u] = true;
        U flow = 0;
        for(int i = cur[u]; ~i && flow < limit; i = ne[i]) {
            cur[u] = i;
            int v = ver[i];
            if(!vis[v] && d[v] == d[u]+w[i] && e[i] > 0) {
                U t = update(v, min(e[i], limit-flow));
                if(!t) d[v] = INF; // INF是否爆d[]，mark
                mncost += t * w[i];
                flow += t;
                e[i] -= t;
                e[i^1] += t;
            }
        }
        vis[u] = false;
        return flow;
    }
    pair<U, U> dinic(int s, int t) {
        S = s, T = t;
        U flow;
        mxflow = mncost = 0;
        while(spfa())
            while(flow = update(S, INF))
                mxflow += flow;
        return {mxflow, mncost};
    }
};
MinCostMaxFlow<int> mcmf;
```

```
// spfa trick: SLF swap
if(d[q.front()] > d[q.back()]) {
    swap(q.front(), q.back());
}
```

**费用流 Dinic&Dijkstra -** $O(m^2logUlogm)$

U 为边的最大容量，基于 Capacity Scaling 。

**费用流 Dinic&SPFA -** $O(nm^2logU)$

U 为边的最大容量，基于 Capacity Scaling 。

## Tarjan求强连通分量

```
void tarjan(int u) {
    dfn[u] = low[u] = ++tms;
    stk[++top] = u, inStk[u] = true;
    for(int i = h[u]; ~i; i = ne[i]) {
        int v = ver[i];
        if(!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }
```

```
        else if(inStk[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if(dfn[u] == low[u]) {
        ++sccCnt; int v;
        do {
            v = stk[top--];
            inStk[v] = false;
            id[v] = sccCnt;
        } while(u != v);
    }
}
```

```
for 所有的点
    if(!dfn[i]) tarjan(i);
```

```
// 缩点 -> DAG
for(int i = 1; i <= n; i ++)
    for i 所有邻点
        if(i 和 v不在一个SCC)
            add(id(i), id(v)); // 连通块相连（有时不用真的建图，可能只是入读和初度）
```

## 2-SAT

将 $A$ 点拆为 $A$ 和 $A'$ 。

1. 若选 $A$ 必选 $B$，那么从 $A$ 向 $B$ 连一条边。
2. tarjan 缩点。（时间复杂度 $O(nm) \Rightarrow O(n)$ ）
3. 判断 $A$ 和 $A'$ 是否在同一强连通分量中；在同一个强连通分量中，则无解。

需要输出方案。

4. 根据缩点后的图，建出反图。
5. 对反图进行拓扑排序。
6. 按照拓扑排序的顺序标记答案。

模型建边方式：

- $A$， $B$ 不能同时取。
  - 选 $A$ 只能选 $B'$，选 $B$ 只能选 $A'$。
  - 连边： $A \to B'$， $B \to A'$。
- $A$， $B$ 不能同时不取。
  - 选 $A'$ 只能选 $B$，选 $B'$ 只能选 $A$。
  - 连边： $A' \to B$， $B' \to A$。
- $A$， $B$ 要么都取，要么都不取。
  - 选 $A$ 只能选 $B$，选 $B$ 只能选 $A$，选 $A'$ 只能选 $B'$，选 $B'$ 只能选 $A'$。
  - 连边： $A \to B$， $B \to A$， $A' \to B'$， $B' \to A'$。
- $A$， $A'$ 必取 $A$。
  - 连边： $A' \to A$。

任意一组解，拓扑序在后面的为真，tarjan 求出的拓扑序为逆序。

```
while(m--) {
```

```
    int i, a, j, b;
    cin >> i >> a >> j >> b;
    add(i + (!a)*n, j + b*n);
    add(j + (!b)*n, i + a*n);
}
for(int i = 1; i <= nn; i ++) {
    if(!dfn[i]) tarjan(i);
}
for(int i = 1; i <= n; i ++) {
    if(id[i] == id[i+n]) {
        cout << "IMPOSSIBLE\n";
        return;
    }
}
cout << "POSSIBLE\n";
for(int i = 1; i <= n; i ++) {
    cout << (id[i] > id[i+n]) << " \n"[i==n];
}
```

## 欧拉路径

是否有欧拉通路，即是否图中只有0个或者2个度数为奇数的点。

## 图的匹配

最小点覆盖 = 最大匹配

最大独立集 = n - 最小点覆盖 = n - 最大匹配

### 二分图最大匹配

**增广路算法 Augmenting Path Algorithm - $O(nm)$**

```
vector<int> e[N];
int n1, n2, m;
int mch[N], st[N]; // match 的点不能有 0，所以从 1 开始。 pos(x, y) x*m + y + 1

int find(int u) {
    for(auto v : e[u]) {
        if(vis[v]) continue;
        vis[v] = true;
        if(!mch[v] || find(mch[v])) {
            mch[v] = u;
            return 1;
        }
    }
    return 0;
}
//
int res = 0;
for(int i = 1; i <= n1; i ++) {
    memset(vis, 0, sizeof vis);
    res += find(i);
}
```

### Hopcraft–Karp (HK) - $O(m\sqrt{n})$

复杂度和直接 Dinic 是同级，但是常数据说小一点。

```
struct HopcraftKarp {
    vector<int> e[N];
    int n1, n2, m, dx[N], dy[N], mchx[N], mchy[N]; // dis_x, dis_y, x_match,
y_match
    bitset<N> ismchx, vis; // x_is_match

    void init(int n1_, int n2_, int m_) {
        n1 = n1_, n2 = n2_, m = m_;
    }
    void add(int a, int b) {
        e[a].push_back(b);
    }
    bool bfs() {
        vector<int> q;
        memset(dx, -1, sizeof dx);
        memset(dy, -1, sizeof dy);
        for(int i = 1; i <= n1; i ++) {
            if(!ismchx[i]) {
                dx[i] = 0;
                q.push_back(i);
            }
        }
        int dT = INF;
        while(!q.empty()) {
            int u = q.front(); q.erase(q.begin());
            if(dx[u] >= dT) break; // dis[u] >= dis[T]
            for(auto v: e[u]) {
                if(~dy[v]) continue;
                dy[v] = dx[u] + 1;
                if(!mchy[v]) dT = dy[v] + 1;
                else {
                    dx[mchy[v]] = dy[v] + 1;
                    q.push_back(mchy[v]);
                }
            }
        }
        return dT != INF;
    }
    int find(int u) {
        for(auto v: e[u]) {
            if(vis[v] || dx[u]+1 != dy[v]) continue;
            vis[v] = true;
            if(!mchy[v] || find(mchy[v])) {
                ismchx[u] = true;
                mchy[v] = u, mchx[u] = v;
                return 1;
            }
        }
        return 0;
    }
    int hk() {
        int ret = 0;
        while(bfs()) {
```

```
            vis.reset();
            for(int i = 1; i <= n; i ++) {
                if(!ismchx[i]) ret += find(i);
            }
        }
        return ret;
    }
}hk;
```

## 一般图最大匹配

### 带花树算法（Blossom Tree Algorithm）- $O(n^3)$

常数小，基本跑不满。

```
struct BlossomTree {
    vector<int> e[N];
    int n, m;
    int fa[N], mch[N], pre[N], col[N], st[N]; // 并查集，匹配点，未匹配点，颜色，标记
    vector<int> q;
    void init(int n_) {
        n = n_;
    }
    void add(int a, int b) {
        e[a].push_back(b), e[b].push_back(a);
    }
    int leader(int x) {
        return (x == fa[x] ? x : fa[x] = leader(fa[x]));
    }
    bool same(int x, int y) {
        return leader(x) == leader(y);
    }
    int lca(int x, int y) {
        static int t = 0; t++;
        x = leader(x), y = leader(y);
        while(st[x] != t) {
            st[x] = t;
            x = leader(pre[mch[x]]);
            if(y) swap(x, y);
        }
        return x;
    }
    void shrink(int x, int y, int anc) {
        while(leader(x) != anc) {
            pre[x] = y, y = mch[x]; // 更新pre，y当临时变量，交替沿着x-anc向上跳
            fa[y] = fa[x] = anc; // 缩花
            if(col[y] == 2) col[y] = 1, q.push_back(y); // 路径上颜色不为1的染成1，并
加入队列
            x = pre[y]; // x交替向上跳
        }
    }
    void rev(int x) {
        // if(x) {
        //     rev(mch[pre[x]]);
        //     mch[x] = pre[x], mch[pre[x]] = x;
        // }
        for(int y; x; x = y) { // 一个点对、一个点对地更新
```

```
            y = mch[pre[x]];
            mch[x] = pre[x], mch[pre[x]] = x;
        }
    }
    int find(int x) {
        for(int i = 1; i <= n; i ++) {
            fa[i] = i, pre[i] = col[i] = 0;
        }
        q = {x};
        col[x] = 1;
        while(!q.empty()) {
            auto u = q.front(); q.erase(q.begin());
            assert(col[u] == 1);
            for(auto v: e[u]) {
                if(col[v] == 2 || same(u, v)) continue; // 偶环或缩过的奇环
                if(col[v] == 1) {
                    int anc = lca(u, v);
                    shrink(u, v, anc);
                    shrink(v, u, anc);
                }
                else if(!col[v]) {
                    col[v] = 2;
                    pre[v] = u;
                    if(!mch[v]) { // 找到增广
                        rev(v); // 修改匹配
                        return 1;
                    }
                    col[mch[v]] = 1;
                    q.push_back(mch[v]); // 继续找
                }
            }
        }
        return 0;
    }
    int ba() {
        int ret = 0;
        for(int i = 1; i <= n; i ++) {
            if(!mch[i]) ret += find(i);
        }
        return ret;
    }
}bt;
```

#### 基于高斯消元的一般图匹配算法 - $O(n^3)$

常数大，基本跑满。好写、好理解，便于解决**最大匹配中的必须点**等问题。

## 二分图最大权匹配

#### KM (Kuhn–Munkres Algorithm) - $O(n^3)$

KM 实际上是用来求**最大权完美匹配**的，所以对二分图需要先进行处理。

- 若求最大权匹配，补点使两部分点相等，不存在的边补 0。
- 若边权能为负，则将不存在的边视为 $-\infty$。

```
const int N = 510;
const LL INF = 0x3f3f3f3f3f3f3f3f;
```

```cpp
int n1, n2, m;

struct KuhnMunkres {
     int n;
     int pre[N], mchy[N], mchx[N];
     LL lx[N], ly[N], slack[N], w[N][N];
     bitset<N> visy;
     void init(int n_, int w_init = 0) { // max(n1, n2)
        n = n_;
        memset(mchx, 0, sizeof mchx);
        memset(mchy, 0, sizeof mchy);
        memset(lx, 0, sizeof lx);
        memset(ly, 0, sizeof ly);
        memset(w, w_init, sizeof w);
     }
     void add(int a, int b, LL c) {
        w[a][b] = c;
        // lx[a] = max(lx[a], c);
     }
     void find(int p) {
        visy.reset();
        memset(pre, 0, sizeof pre);
        memset(slack, 0x3f, sizeof slack);
        int x = mchy[0] = p, y = 0;
        while(true) {
            visy[y] = true;
            LL d = INF;
            int yy = 0;
            for(int i = 1; i <= n; i ++) {
                if(visy[i]) continue;
                LL t = lx[x]+ly[i] - w[x][i];
                if(slack[i] > t) slack[i] = t, pre[i] = y;
                if(slack[i] < d) d = slack[i], yy = i;
            }
            for(int i = 0; i <= n; i ++) {
                if(visy[i]) lx[mchy[i]] -= d, ly[i] += d;
                else slack[i] -= d;
            }
            y = yy;
            if(!mchy[y]) break;
            x = mchy[y];
        }
        while(y) mchy[y] = mchy[pre[y]], mchx[mchy[y]] = y, y = pre[y];
     }
     LL km() {
        for(int i = 1; i <= n; i ++) find(i);
        LL ret = 0;
        for(int i = 1; i <= n; i ++) {
            ret += lx[i] + ly[i];
        }
        return ret;
     }
}km;
```

```
// 未匹配为 0
for(int i = 1; i <= n1; i ++) {
    cout << (km.w[i][km.mchx[i]] ? km.mchx[i] : 0) << " \n"[i==n1];
}
```

### 一般图最大权匹配

#### 带花树上KM - $O(n^3)$

```
```

## 优化连边

### 线段树优化连边

```
vector<array<int, 2>> e[N<<3|1];
int n, q, S, Nid;
int nid[2][N<<2];
LL d[N<<3|1]; // 所有的距离
bitset<N<<3|1> vis;

void add(int a, int b, int c) {
    e[a].push_back({b, c});
}
struct Info {
    int l, r;
};
struct SegmentTree {
    Info tr[2][N<<2];
    // 0 - in fa->son, 1 - out son->fa
    vector<int> ve;
    #define ls(u) ((u)<<1)
    #define rs(u) ((u)<<1|1)
    void build(int u, int l, int r, int ty) {
        nid[ty][u] = ++Nid;
        tr[ty][u] = {l, r};
        if(l == r) {
            if(ty == 0) add(nid[0][u], r, 0);
            else add(r, nid[1][u], 0);
            return;
        }
        int mid = l+r >> 1;
        build(ls(u), l, mid, ty), build(rs(u), mid+1, r, ty);
        if(ty == 0) {
            add(nid[0][u], nid[0][ls(u)], 0);
            add(nid[0][u], nid[0][rs(u)], 0);
        }
        else {
            add(nid[1][ls(u)], nid[1][u], 0);
            add(nid[1][rs(u)], nid[1][u], 0);
        }
    }
    void ask(int u, int l, int r, int ty) {
        assert(l <= r); // 在外面判好
        auto [ll, rr] = tr[ty][u];
        if(rr < l || r < ll) return;
```

```
        if(l <= ll && rr <= r) {
            ve.push_back(nid[ty][u]);
            return;
        }
        ask(ls(u), l, r, ty), ask(rs(u), l, r, ty);
    }
    void addx2lr(int from, int l, int r, int c) {
        assert(l <= r); // 在外面判好
        ve = vector<int>();
        ask(1, l, r);
        for(auto v: ve) addFlow(from, v, c, 0);
    }
    #undef ls
    #undef rs
}sgt;

void solve() {
    cin >> n >> q >> S;
    Nid = n+1;
    sgt.build(1, 1, n, 0);
    sgt.build(1, 1, n, 1);
    while(q--) {
        int op, u, l, r, w;
        cin >> op;
        if(op == 1) {
            cin >> l >> r >> w;
            add(l, r, w);
        }
        else {
            cin >> u >> l >> r >> w;
            sgt.ve.clear();
            sgt.ask(1, l, r, op-2);
            if(op == 2) {
                for(auto v: sgt.ve) add(u, v, w);
            }
            else {
                for(auto v: sgt.ve) add(v, u, w);
            }
        }
    }
}
```

## 树上优化建边

### 树链剖分

配合线段树，和上方思想一致，按照需求看要建一棵还是两棵线段树。

注意在线段树里的下标代表树链剖分内部的 dfn。

一般建法需要 $log^2n$ 条边。

使用重链的前缀优化可以只建 $logn$ 条边，原理是往上跳的过程中只有最后一段可能跳的不是重链的前缀。

具体前缀优化建边方式为：

对于每一个点，我们拆出一个虚点，原点向虚点连长度为 0 的边，然后每个虚点向下一个虚点连长度为 0 的边。这样在前缀对应的最后一个点向外连边就等价于在原图上对于前缀上的所有点连出一条边了。

#### 倍增

倍增的同时，连边，具体代码在倍增LCA注释的地方。

# 数据结构

## DSU

```
struct DSU {
    vector<int> fa, siz;

    void init(int n) {
        fa.resize(n+1);
        siz.resize(n+1);
        for(int i = 0; i <= n; i ++)
            fa[i] = i, siz[i] = 1;
    }
    int leader(int x, bool compress = true) {
        if(x == fa[x]) return x;
        int root = leader(fa[x]);
        return compress ? fa[x] = root : root;
    }
    bool same(int x, int y) {
        return leader(x) == leader(y);
    }
    bool merge(int x, int y, bool strict = true) {
        x = leader(x), y = leader(y);
        if(x == y) return false;
        if(!strict && siz[x] < siz[y]) swap(x, y);
        siz[x] += siz[y];
        fa[y] = x;
        return true;
    }
    int size(int x) {
        return siz[leader(x)];
    }
}dsu;
```

## Fenwick Tree

```
template<typename T>
struct FenwickTree {
    int n;
    vector<T> a;
    void init(int n) {
        this->n = n;
        a.assign(n+1, T());
    }
    void clear(int x) { // for Max
        for( ; x <= n; x += lowbit(x)) {
            a[x] = T();
        }
    }
    int lowbit(int x) {
        return x & -x;
    }
```

```
    void add(int x, T v) {
        for( ; x <= n; x += lowbit(x))
            a[x] += v;
    }
    T sum(int x) {
        auto ans = T();
        for( ; x; x -= lowbit(x))
            ans += a[x];
        return ans;
    }
    T sum(int l, int r) {
        return sum(r) - sum(l-1);
    }
    int kth(T k) {
        assert(k > 0);
        int x = 1;
        for (int i = 1<<__lg(n); i; i /= 2) {
            if(x+i <= n && k > a[x+i-1]) {
                x += i;
                k -= a[x-1];
            }
        }
        return x;
    }
};
FenwickTree<int> fen;
```

维护最大值（如要清空，请注意）

```
constexpr int INF = 1e9;
struct Max {
    int x;
    Max(int x = -INF) : x(x) {}
    Max &operator += (const Max &a) {
        if (a.x > x) x = a.x;
        return *this;
    }
};
FenwickTree<Max> fen;
```

维护最小值

```
struct Min {
    int x;
    Min(int x = INF) : x(x) {}
    Min &operator += (Min a) {
        if (a.x < x) x = a.x;
        return *this;
    }
};
```

# 线段树

```
vector<int> a, b;

struct Info {
    int l, r;
    LL max, addTag;
};
struct SegmentTree {
    Info tr[N<<2];
    #define ls(u) (u<<1)
    #define rs(u) (u<<1|1)
    void update(Info &u, const Info &l, const Info &r) {
        u.max = max(l.max, r.max);
    }
    void pushUp(int u) {
        update(tr[u], tr[ls(u)], tr[rs(u)]);
    }
    void lazydown(int u) {
        if(tr[u].addTag) {
            auto &t = tr[u].addTag;
            tr[ls(u)].max += t;
            tr[rs(u)].max += t;
            tr[ls(u)].addTag += t;
            tr[rs(u)].addTag += t;
            t = 0;
        }
    }
    void build(int u, int l, int r) {
        if(l == r) {
            tr[u] = {l, r, 0, 0};
            return;
        }
        tr[u] = {l, r};
        int mid = l+r >> 1;
        build(ls(u), l, mid), build(rs(u), mid+1, r);
        pushUp(u);
    }
    void modify(int u, int l, int r, int c) {
        int ll = tr[u].l, rr = tr[u].r;
        if(l <= ll && rr <= r) {
            tr[u].max += c;
            tr[u].addTag += c;
            return;
        }
        lazydown(u);
        int mid = ll+rr >> 1;
        if(l <= mid) modify(ls(u), l, r, c);
        if(r > mid) modify(rs(u), l, r, c);
        pushUp(u);
    }
    LL query(int u, int l, int r) {
        int ll = tr[u].l, rr = tr[u].r;
        if(l <= ll && rr <= r) {
            return tr[u].max;
        }
        lazydown(u);
```

```
        int mid = ll+rr >> 1;
        if(r <= mid) return query(ls(u), l, r);
        if(l > mid) return query(rs(u), l, r);
        LL lhs = query(ls(u), l, r), rhs = query(rs(u), l, r);
        LL ret = max(lhs, rhs);
        return ret;
    }
    #undef ls
    #undef rs
}sgt;
```

## 线段树分治

线段树区间修改时间维，把其他数据结构的删除操作改为增加/撤回。

```
const int N = 1e5+10, M = 2e5+10;

struct Backtrace {
    int x, y, siz;
}stkBack[M];
int top;

int n, m, k;
array<int, 2> rec[M];
vector<int> pos[M];

struct DSU {
    vector<int> fa, siz, d;

    void init(int n) {
        fa.resize(n+1);
        siz.resize(n+1);
        for(int i = 0; i <= n; i ++)
            fa[i] = i, siz[i] = 1;
    }
    int leader(int x, bool compress = false) {
        if(x == fa[x]) return x;
        int root = leader(fa[x]);
        return compress ? fa[x] = root : root;
    }
    bool same(int x, int y) {
        return leader(x) == leader(y);
    }
    bool merge(int x, int y, bool strict = true) {
        x = leader(x), y = leader(y);
        if(x == y) return false;
        if(!strict && siz[x] < siz[y]) swap(x, y);

        stkBack[++top] = {x, y, siz[x]};

        siz[x] += siz[y];
        fa[y] = x;
        return true;
    }
    int size(int x) {
        return siz[leader(x)];
    }
```

```
}dsu;

struct Info {
    int l, r;
    vector<int> id;
};

struct SegmentTree {
    Info tr[N<<2];
    #define ls(u) (u<<1)
    #define rs(u) (u<<1|1)
    void build(int u, int l, int r) {
        if(l == r) {
            tr[u] = {l, r};
            return;
        }
        tr[u] = {l, r};
        int mid = l+r >> 1;
        build(ls(u), l, mid), build(rs(u), mid+1, r);
    }
    void modify(int u, int l, int r, int x) {
        int ll = tr[u].l, rr = tr[u].r;
        if(l <= ll && rr <= r) {
            tr[u].id.push_back(x);
            return;
        }
        int mid = ll+rr >> 1;
        if(l <= mid) modify(ls(u), l, r, x);
        if(r > mid) modify(rs(u), l, r, x);
    }
    #undef ls
    #undef rs
}sgt;

void work(int u) {
    int ttop = top;
    bool ok = false;
    auto p = sgt.tr[u];
    int l = p.l, r = p.r;
    for(auto id : p.id) {
        auto [x, y] = rec[id];
        dsu.merge(x, y, false);
        if(dsu.size(x) == n) {
            for(int i = l; i <= r; i ++) {
                cout << "Connected\n";
            }
            ok = true;
            break;
        }
    }
    if(!ok) {
        if(l == r) {
            cout << "Disconnected\n";
        }
        else {
            work(u<<1), work(u<<1|1);
        }
    }
```

```
    while(top > ttop) {
        auto [x, y, siz] = stkBack[top--];
        dsu.siz[x] = siz, dsu.fa[y] = y;
    }
}
void solve() {
    cin >> n >> m;
    dsu.init(n);
    for(int i = 0; i < m; i ++) {
        auto &[x, y] = rec[i];
        cin >> x >> y;
        pos[i].push_back(0);
    }
    cin >> k;
    sgt.build(1, 1, k);
    for(int i = 0; i < k; i ++) {
        int c;
        cin >> c;
        while(c --) {
            int x;
            cin >> x;
            pos[--x].push_back(i+1);
        }
    }
    for(int i = 0; i < m; i ++) {
        pos[i].push_back(k+1);
    }
    for(int i = 0; i < m; i ++) {
        auto v = pos[i];
        for(int j = 1; j < v.size(); j ++) {
            if(v[j-1] + 1 <= v[j] - 1) {
                sgt.modify(1, v[j-1]+1, v[j]-1, i);
            }
        }
    }
    work(1);
}
```

## 吉如一（吉司机）线段树（seg-beats）

最值修改、维护历史最值。

```
const int N = 5e5+10;
const LL INF = 1e18;

int a[N];

struct Info {
    int l, r;
    // 分为最大值和非最大值，因为分为最大值和全部不太好求区间和。
    LL mx, mx2, hmx, sum, mxcnt;
    LL add, addmx, mxadd, mxaddmx;
};
struct SegmentTree {
    Info tr[N<<2];
    #define ls(u) (u<<1)
    #define rs(u) (u<<1|1)
```

```
    void update(Info &p, Info l, Info r) {
        auto &[_1, _2, mx, mx2, hmx, sum, mxcnt, _3, _4, _5, _6] = p;
        sum = l.sum + r.sum;
        hmx = max(l.hmx, r.hmx);
        if(l.mx == r.mx) {
            mx = r.mx;
            mx2 = max(l.mx2, r.mx2);
            mxcnt = l.mxcnt + r.mxcnt;
        }
        else if(l.mx < r.mx) {
            mx = r.mx;
            mx2 = max(l.mx, r.mx2);
            mxcnt = r.mxcnt;
        }
        else {
            mx = l.mx;
            mx2 = max(l.mx2, r.mx);
            mxcnt = l.mxcnt;
        }
    }
    void update(int u, LL c1, LL c1_, LL c2, LL c2_) {
        // 非最大值 和 最大值
        auto &[l, r, mx, mx2, hmx, sum, mxcnt, add, addmx, mxadd, mxaddmx] =
tr[u];
        sum += c1 * (r-l+1 - mxcnt) + c2 * mxcnt;
        hmx = max(hmx, mx + c2_);
        mx += c2;
        if(mx2 != -INF) mx2 += c1;
        addmx = max(addmx, add + c1_);
        add += c1;
        mxaddmx = max(mxaddmx, mxadd + c2_);
        mxadd += c2;
    }
    void lazydown(int u) {
        auto &[_1, _2, _3, _4, _5, _6, _7, add, addmx, mxadd, mxaddmx] = tr[u];
        Info &l = tr[ls(u)], &r = tr[rs(u)];
        LL tmx = max(l.mx, r.mx);
        if(l.mx == tmx) update(ls(u), add, addmx, mxadd, mxaddmx);
        else update(ls(u), add, addmx, add, addmx);
        if(r.mx == tmx) update(rs(u), add, addmx, mxadd, mxaddmx);
        else update(rs(u), add, addmx, add, addmx);
        add = addmx = mxadd = mxaddmx = 0;
    }
    void pushup(int u) {
        update(tr[u], tr[ls(u)], tr[rs(u)]);
    }
    void build(int u, int l, int r) {
        tr[u] = {l, r};
        if(l == r) {
            tr[u].mx = tr[u].hmx = tr[u].sum = a[r];
            tr[u].mx2 = -INF, tr[u].mxcnt = 1;
            return;
        }
        int mid = l+r >> 1;
        build(ls(u), l, mid), build(rs(u), mid+1, r);
        pushup(u);
    }
    void modifyAdd(int u, int l, int r, int c) {
```

```
        assert(l <= r);
        int ll = tr[u].l, rr = tr[u].r;
        if(rr < l || r < ll) return;
        if(l <= ll && rr <= r) {
            update(u, c, c, c, c);
            return;
        }
        lazydown(u);
        modifyAdd(ls(u), l, r, c), modifyAdd(rs(u), l, r, c);
        pushup(u);
    }
    void modifyMin(int u, int l, int r, int c) {
        assert(l <= r);
        int ll = tr[u].l, rr = tr[u].r;
        if(rr < l || r < ll || tr[u].mx <= c) return; // 保证了mx2一直为严格次大值。
        if(l <= ll && rr <= r && tr[u].mx2 < c) {
            auto t = tr[u].mx - c;
            update(u, 0, 0, -t, -t);
            return;
        }
        lazydown(u);
        modifyMin(ls(u), l, r, c), modifyMin(rs(u), l, r, c);
        pushup(u);
    }
    LL querySum(int u, int l, int r) {
        assert(l <= r);
        int ll = tr[u].l, rr = tr[u].r;
        if(rr < l || r < ll) return 0;
        if(l <= ll && rr <= r) return tr[u].sum;
        lazydown(u);
        return querySum(ls(u), l, r) + querySum(rs(u), l, r);
    }
    LL queryMax(int u, int l, int r) {
        assert(l <= r);
        int ll = tr[u].l, rr = tr[u].r;
        if(rr < l || r < ll) return -INF;
        if(l <= ll && rr <= r) return tr[u].mx;
        lazydown(u);
        return max(queryMax(ls(u), l, r), queryMax(rs(u), l, r));
    }
    LL queryHistoryMax(int u, int l, int r) {
        assert(l <= r);
        int ll = tr[u].l, rr = tr[u].r;
        if(rr < l || r < ll) return -INF;
        if(l <= ll && rr <= r) return tr[u].hmx;
        lazydown(u);
        return max(queryHistoryMax(ls(u), l, r), queryHistoryMax(rs(u), l, r));
    }
    #undef ls
    #undef rs
}sgt;

void solve() {
    int n, m;
    cin >> n >> m;
    for(int i = 1; i <= n; i ++) cin >> a[i];
    sgt.build(1, 1, n);
    while(m --) {
```

```
        int op, l, r, x;
        cin >> op >> l >> r;
        if(op == 1) {
            cin >> x;
            sgt.modifyAdd(1, l, r, x);
        }
        else if(op == 2) {
            cin >> x;
            sgt.modifyMin(1, l, r, x);
        }
        else if(op == 3) {
            cout << sgt.querySum(1, l, r) << '\n';
        }
        else if(op == 4) {
            cout << sgt.queryMax(1, l, r) << '\n';
        }
        else {
            cout << sgt.queryHistoryMax(1, l, r) << '\n';
        }
    }
}
```

### 李超线段树 - $O(nlog^2V)$

标记永久化。二维平面，插入线段 $log^2V$ ，查询 $x = x_0$ ，最大的 y 是多少。（V 为横坐标范围）

```
```

## 倍增LCA

```
struct MultiplyLCA {
    static const int _K = 20;
    vector<vector<array<int, 2>>> e;
    vector<int> dep;
    vector<array<int, _K>> fa;
    void init(int n) {
        dep.assign(n+1, 0);
        e.assign(n+1, {});
        fa.assign(n+1, {});
    }
    void add(int a, int b, int c) {
        e[a].push_back({b, c});
        e[b].push_back({a, c});
    }
    void build(int root = 1) {
        bfs(root);
    }
    void bfs(int root) {
        queue<int> q;
        dep[root] = 1, q.push(root);
        while(q.size()) {
            int u = q.front();
            q.pop();
            for(auto [v, c] : e[u]) {
                if(dep[v]) continue;
                dep[v] = dep[u] + 1;
```

```
                fa[v][0] = u;
                // nid[v][0] = ++Nid;
                // addFlow(nid[v][0], T, c, 0);
                for(int j = 1; j < _K; j ++) {
                    fa[v][j] = fa[fa[v][j - 1]][j - 1];
                    nid[v][j] = ++Nid;
                    // addFlow(nid[v][j], nid[v][j-1], INF, 0);
                    // addFlow(nid[v][j], nid[fa[v][j-1]][j-1], INF, 0);
                }
                q.push(v);
            }
        }
    }
    int LCA(int x, int y) {
        if(dep[x] < dep[y]) swap(x, y);
        for(int k = _K-1; k >= 0; k --)
            if(dep[fa[x][k]] >= dep[y])
                x = fa[x][k];
        if(x == y) return x;
        for(int k = _K-1; k >= 0; k --)
            if(fa[x][k] != fa[y][k])
                x = fa[x][k], y = fa[y][k];
        return fa[x][0];
    }
    /*void addx2lr(int from, int l, int r, int c) {
        if(dep[l] < dep[r]) swap(l, r);
        for(int k = _K-1; k >= 0; k --) {
            if(dep[fa[l][k]] >= dep[r]) {
                // addFlow(from, nid[l][k], c, 0);
                l = fa[l][k];
            }
        }
        if(l == r) return;
        for(int k = _K-1; k >= 0; k --) {
            if(fa[l][k] != fa[r][k]) {
                addFlow(from, nid[l][k], c, 0);
                addFlow(from, nid[r][k], c, 0);
                l = fa[l][k], r = fa[r][k];
            }
        }
        addFlow(from, nid[l][0], c, 0);
        addFlow(from, nid[r][0], c, 0);
    }*/
}mlca;
```

## 树链剖分

```
struct HeavyLightDecompotion {
    int n, tms;
    vector<int> siz, top, dep, fa, dfni, dfno, rnk;
    vector<vector<int>> e;

    void init(int n) {
        this->n = n;
        siz.resize(n+1);
        top.resize(n+1);
        dep.resize(n+1);
```

```
        fa.resize(n+1);
        dfni.resize(n+1);
        dfno.resize(n+1);
        rnk.resize(n+1);
        tms = 0;
        e.assign(n+1, {});
    }
    void addEdge(int u, int v) {
        e[u].push_back(v);
        e[v].push_back(u);
    }
    void build(int root = 1) {
        top[root] = root;
        dep[root] = 1;
        fa[root] = -1; // 不要直接访问fa[i]...
        dfs1(root);
        dfs2(root);
    }
    void dfs1(int u) {
        if (fa[u] != -1) {
            e[u].erase(find(all(e[u]), fa[u]));
        }
        siz[u] = 1;
        for (auto &v : e[u]) {
            fa[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            siz[u] += siz[v];
            if (siz[v] > siz[e[u][0]]) {
                swap(v, e[u][0]); // 保留引用
            }
        }
    }
    void dfs2(int u) {
        dfni[u] = ++tms;
        rnk[dfni[u]] = u;
        for (auto v : e[u]) {
            top[v] = (v == e[u][0]) ? top[u] : v;
            dfs2(v);
        }
        dfno[u] = tms+1;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = fa[top[u]];
            }
            else {
                v = fa[top[v]];
            }
        }
        return (dep[u] < dep[v]) ? u : v;
    }
    int getDist(int u, int v) {
        return dep[u] + dep[v] - 2 * dep[lca(u, v)];
    }
    int jump(int u, int k) {
        if (dep[u] < k) {
```

```
            return -1;
        }
        int d = dep[u] - k;
        while (dep[top[u]] > d) {
            u = fa[top[u]];
        }
        return rnk[dfni[u] - dep[u] + d];
    }
}hld;
```

```
// 路径查询和维护
/**********************************
void add_x2y(int x, int y, int c) {
    while(top[x] != top[y]) {
        if(dep[top[x]] < dep[top[y]]) swap(x, y);
        modify/query/ ... (1, dfn[top[x]], dfn[x], c);
        x = fa[top[x]];
    }
    if(dep[x] < dep[y]) swap(x, y);
    modify/query/ ... (1, dfn[y], dfn[x], c);
    // 不到公共祖先 ... (1, dfn[y]+1, dfn[x], c); (里面或外面判一下两个点是否相等)
}
**********************************/
// 优化建边
/**********************************
void addEdge_x2y(int u, int x, int y, int c) {
    while(top[x] != top[y]) {
        if(dep[top[x]] < dep[top[y]]) swap(x, y);

        // sgt.addx2lr(u, dfni[top[x]], dfni[x], c);
        addFlow(u, vir[x], c, 0);
        x = fa[top[x]];
    }
    if(dep[x] < dep[y]) swap(x, y);
    if(x != y) sgt.addx2lr(u, dfni[y]+1, dfni[x], c);
}
**********************************/
// 重链前缀优化建边
/**********************************
void prefix() {
    for(int i = 1; i <= n; i ++) { // 在流里面已经是以dfni为标。
        vir[i] = ++Nid;
        addFlow(vir[i], dfni[i], INF, 0);
    }
    for(int i = 1; i <= n; i ++) {
        if(i != top[i]) {
            addFlow(vir[i], vir[fa[i]], INF, 0);
        }
    }
}
**********************************/
```

感觉树链剖分重`点`，倍增`点`和`边`都重。

## 珂朵莉树 ODT

```
struct ODTree {
    struct Node {
        LL l, r;
        mutable LL v;
        Node(LL _l, LL _r = 0, LL _v = 0) : l(_l), r(_r), v(_v) {}
        bool operator < (const Node &rhs) const {
            return l < rhs.l;
        }
    };
    struct Rank {
        LL val, cnt;
        Rank(LL _val, LL _cnt) : val(_val), cnt(_cnt) {}
        bool operator < (const Rank &rhs) const {
            return val < rhs.val;
        }
    };

    set<Node> s;

    void insert(LL l, LL r, LL v) {
        s.insert(Node(l, r, v));
    }
    set<Node>::iterator split(LL pos) { // [l, pos), [pos, r]
        auto it = s.lower_bound(Node(pos));
        if(it != s.end() && it->l == pos) {
            return it;
        }
        it --;
        if(it->r < pos) return s.end();
        LL l = it->l, r = it->r, v = it->v;
        s.erase(it);
        s.insert(Node(l, pos-1, v));
        return s.insert(Node(pos, r, v)).first;
    }
    void assign(LL l, LL r, LL v) {
        auto itr = split(r+1);
        auto itl = split(l);
        s.erase(itl, itr);
        s.insert(Node(l, r, v));
    }
    void add(LL l, LL r, LL v) {
        auto itr = split(r+1);
        auto itl = split(l);
        for(auto it = itl; it != itr; it ++) {
            it->v += v;
        }
    }
    LL kth(LL l, LL r, LL k) {
        auto itr = split(r+1);
        auto itl = split(l);
        vector<Rank> v;
        for(auto it = itl; it != itr; it ++) {
            v.push_back(Rank(it->v, it->r - it->l + 1));
        }
        sort(all(v));
```

```
        for(auto [val, cnt] : v) {
            if(cnt < k) {
                k -= cnt;
            }
            else {
                return val;
            }
        }
        return -1;
    }
}odt;
```

# 字符串

## 字符串双哈希

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

const LL mod1=1000000007;
const LL mod2=1000000009;

hashv operator + (hashv a, hashv b) {
    int c1 = a.fi+b.fi,c2 = a.se+b.se;
    if (c1 >= mod1) c1 -= mod1;
    if (c2>=mod2) c2 -= mod2;
    return {c1, c2};
}
hashv operator - (hashv a, hashv b) {
    int c1 = a.fi-b.fi,c2 = a.se-b.se;
    if (c1 < 0) c1 += mod1;
    if (c2 < 0) c2 += mod2;
    return {c1, c2};
}
hashv operator * (hashv a, hashv b) {
    return {1LL*a.fi * b.fi % mod1, 1LL*a.se*b.se%mod2};
}

hashv pw[N], Hx[N], base = {13331, 23333};

pw[0] = {1, 1};
for (int i=1;i<=n;i++)
    pw[i] = pw[i-1]*base, Hx[i] = Hx[i-1]*base+mkp(s[i],s[i]);

// 前面的才是高位，需要抵消掉
hashv hx1 = Hx[r]-Hx[l-1]*pw[r-l+1];
hashv hx2 = Hx[x]-Hx[x-len]*pw[len];

if (s1==s2) {
    // 双哈希相等
}
```

## 最小表示法

```
int MinimalString(string s) {
    int n = s.size();
    int p = 0, q = 1, k = 0;
    while(p < n && q < n && k < n) {
        char ch1 = s[(p+k) % n], ch2 = s[(q+k) % n];
        if(ch1 == ch2) k++;
        else {
            if(ch1 > ch2) p = p+k+1;
            else q = q+k+1;
            if(p == q) q++; // 保证两者不同
            k = 0;
        }
    }
    p = min(p, q); // 取未到末尾的
    return p;
}
// s的最小表示为 s[p..(p+m)%m]
```

## Manacher（马拉车）

```
int p[N<<1];
char s[N], t[N<<1];
void Manacher(char s[]) {
    int n = 0;
    t[n++] = '$', t[n++] = '#';
    for(int i = 0; s[i]; i ++) {
        t[n++] = s[i];
        t[n++] = '#';
    }
    t[n++] = '\0';
    int r = 0, mid = 0;
    for(int i = 0; i < n; i ++) {
        p[i] = r > i ? min(p[2*mid - i], r - i) : 1;
        while(t[i+p[i]] == t[i-p[i]]) p[i]++;
        if(r < i+p[i]) {
            r = i+p[i];
            mid = i;
        }
    }
}
```

```
/* p 数组里的值 - 1 为回文串的长度，非#位的是奇数，#位是偶数。
用void，用全局变量记录 p 数组和其他处理结果，效率高一点。
string += ，效率和这种写法比慢1/3。*/
```

## Trie树（字典树）

```
int idx, son[N*32][2], id[N*32]; // id, cnt之类的信息按需要记。
int k;

void insert(int x, int u) {
    int p = 0;
```

```
    for(int i = k-1; i >= 0; i --) {
        int t = x>>i & 1;
        if(!son[p][t]) son[p][t] = ++idx;
        p = son[p][t];
    }
    id[p] = u;
}
int query(int x, int u) { // query内部操作根据需要找
    int p = 0, ret = 0;
    for(int i = k-1; i >= 0; i --) {
        int t = x>>i & 1;
        if(son[p][t]) {
            ret = (ret<<1) + 1;
            p = son[p][t];
        }
        else if(son[p][!t]) {
            ret = (ret<<1);
            p = son[p][!t];
        }
    }
    /*
    if(ret > ans) {
        ans = ret;
        pp = id[p];
        qq = u;
    }
    */
    return ret;
}
void init() {
    for(int i = 0; i <= idx; i ++) {
        id[i] = 0;
        for(auto x: {0, 1}) {
            son[i][x] = 0;
        }
    }
    idx = 0;
}
```

```
/* 有些时候找异或最小，或从高位到低位尽可能匹配的，
可以直接 sort，相邻的就是异或值最小的。 */
```

# 数学

## 数论结论

## 分解因数

```cpp
vector<int> divs;
for(int i = 1; i <= c/i; i ++) {
    if(c % i == 0) {
        divs.push_back(i);
        if(c/i != i) divs.push_back(c/i); // 不重复
    }
}
```

## 快速幂 & ExGcd 求逆元 & 线性求逆元

```cpp
int qpm(int a, int b, const int &c) { // int/LL
    int ans = 1 % c;
    while(b) {
        if(b & 1) ans = 1LL*ans*a % c;
        a = 1LL*a*a % c;
        b >>= 1;
    }
    return ans;
}
template<typename T>
T ExGcd(T a, T b, T &x, T &y) {
    if(!b) {
        x = 1, y = 0;
        return a;
    }
    T d = ExGcd(b, a%b, x, y);
    T t = x;
    x = y, y = t-a/b*y;
    return d;
}
// a关于b的逆元，gcd(a, b) != 1时，逆元不存在。
int ExGcdInv(int a,  int b) {
    int x, y;
    int d = ExGcd(a, b, x, y);
    assert(d != 1);
    x = (x%b + b) % b;
    return x;
}
vector<int> inv;
void GetInvs(int n) {
    inv.resize(n+1);
    inv[1] = 1;
    for(int i = 2; i <= n; i ++)
        inv[i] = (LL)(MO - MO / i) * inv[MO % i] % MO;
}
```

## 组合数

二项式定理：$(y+1)^n = \sum_{i=0}^{n} \binom{n}{i} \cdot y^i$

一个普通恒等式：$k \cdot \binom{n}{k} = n \cdot \binom{n-1}{k-1}$

范德蒙德卷积：$\sum_{k=0}^{a} \binom{n}{k} \cdot \binom{m}{a-k} = \binom{n+m}{a}$

注意利用组合数的对称性等性质，使用范德蒙德卷积。

```
const int MO = 998244353;
using Z = MInt<MO>;

struct Comb {
    int n;
    vector<Z> _fac, _facInv, _inv;

    Comb() : n{0}, _fac{1}, _facInv{1}, _inv{0} {}
    Comb(int n) : Comb() {
        init(n);
    }

    void init(int m) {
        if (m <= n) return;
        _fac.resize(m+1);
        _facInv.resize(m+1);
        _inv.resize(m+1);

        for (int i = n+1; i <= m; i++) {
            _fac[i] = _fac[i-1] * i;
        }
        _facInv[m] = _fac[m].inv();
        for (int i = m; i > n; i--) {
            _facInv[i-1] = _facInv[i] * i;
            _inv[i] = _facInv[i] * _fac[i-1];
        }
        n = m;
    }

    Z fac(int m) {
        if (m > n) init(2*m);
        return _fac[m];
    }
    Z facInv(int m) {
        if (m > n) init(2*m);
        return _facInv[m];
    }
    Z inv(int m) {
        if (m > n) init(2*m);
        return _inv[m];
    }
    Z permu(int n, int m) {
        if(n < m || m < 0) return 0;
        return fac(n) * facInv(n-m);
    }
    Z binom(int n, int m) {
        if (n < m || m < 0) return 0;
        return fac(n) * facInv(m) * facInv(n - m);
    }
}comb;
```

## 中国剩余定理 - Crt

```
template<typename T>
T Crt(vector<T>& r, vector<T>& mod) {
    T n = 1, ans = 0;
    int k = r.size();
    for (int i = 0; i < k; i++) n = n * mod[i];
    for (int i = 0; i < k; i++) {
        T m = n / mod[i], b, y;
        ExGcd(m, mod[i], b, y);
        ans = (ans + r[i] * m * b % n) % n;
    }
    return (ans % n + n) % n;
}
```

### 扩展中国剩余定理 - ExCrt

```
void ExGcd(LL a, LL b, LL &d, LL &x, LL &y) {
    if (!b) d = a, x = 1, y = 0;
    else ExGcd(b, a % b, d, y, x), y -= x * (a / b);
}
PLL ExCrt(PLL a, PLL b) {
    auto[r1, m1] = a;
    auto[r2, m2] = b;
    if (r1 == -1 || r2 == -1) return {-1, -1};
    LL d, l1, l2;
    ExGcd(m1, m2, d, l1, l2);
    if ((r2 - r1) % d) return {-1, -1};
    LL M = m1 * m2 / d; // (__int128)m1 * m2 / d;
    LL R = ((r1 + (r2 - r1) / d * l1 % M * m1) % M + M) % M;
    //     (((__int128)r1 + (__int128)(r2 - r1) / d * l1 % M * m1) % M + M) % M;
    return {R, M};
}
```

## 线性基（子集第k大异或和）

### 正常版

```
struct LinearBasis {
    int n;
    vector<LL> a;
    void init(int _n = 64) {
        n = _n;
        a.assign(n, 0);
    }
    bool insert(LL x) {
        for(int i = n-1; i >= 0; i --) {
            if(!x) return false;
            if(!(x>>i & 1)) continue;
            if(a[i]) x ^= a[i];
            else {
                for(int j = 0; j < i; j ++) {
                    if(x>>j & 1) {
                        x ^= a[j];
                    }
```

```
                }
                for(int j = i+1; j < n; j ++) {
                    if(a[j]>>i & 1) {
                        a[j] ^= x;
                    }
                }
                a[i] = x;
                break;
            }
        }
        if(!x) return false;
        return true;
    }
    LL max(LL x = 0) {
        LL ret = x;
        for(int i = n-1; i >= 0; i --) {
            if((ret ^ a[i]) > ret) {
                ret ^= a[i];
            }
        }
        return ret;
    }
    LL kth(LL k) { // 能表示出来的第k大，不重复。
        LL ret = 0;
        for(int i = 0; i < n; i ++) {
            if(a[i]) {
                if(k & 1) { // 不为0的位才算个数
                    ret ^= a[i];
                }
                k >>= 1;
            }
        }
        return ret;
    }
    int count() {
        int ret = 0;
        for(int i = 0; i < n; i ++) {
            if(a[i]) ret++;
        }
        return ret;
    }
    void mergeFrom(const LinearBasis &rhs) {
        for(int i = 0; i < rhs.n; i ++) {
            insert(rhs.a[i]);
        }
    }
    LinearBasis merge(const LinearBasis &lhs, const LinearBasis &rhs) {
        LinearBasis res = lhs;
        res.mergeFrom(rhs);
        return res;
    }
}lb;
```

### bitset版

```
```

## 复数

```
struct Complex{
    double x, y;
    Complex(double xx = 0.0, double yy = 0.0) {
        x = xx, y = yy;
    }
    Complex operator + (const Complex &o) const {
        return Complex(x+o.x, y+o.y);
    }
    Complex operator - (const Complex &o) const {
        return Complex(x-o.x, y-o.y);
    }
    Complex operator * (const Complex &o) const {
        return Complex(x*o.x-y*o.y, x*o.y+y*o.x);
    }
};
```

## FFT

```
const double Pi = acos(-1.0);

struct Complex{
    double x, y;
    Complex(double xx = 0.0, double yy = 0.0) {
        x = xx, y = yy;
    }
    Complex operator + (const Complex &o) const {
        return Complex(x+o.x, y+o.y);
    }
    Complex operator - (const Complex &o) const {
        return Complex(x-o.x, y-o.y);
    }
    Complex operator * (const Complex &o) const {
        return Complex(x*o.x-y*o.y, x*o.y+y*o.x);
    }
};

int n, m, limit = 1;
Complex a[N], b[N];

// 递归版本
void FFT(int tlimit, Complex a[], int on) {
    if(tlimit == 1) return;
    Complex a1[tlimit>>1], a2[tlimit>>1];
    for(int i = 0; i < tlimit; i += 2)
        a1[i>>1] = a[i], a2[i>>1] = a[i+1];
    FFT(tlimit>>1, a1, on), FFT(tlimit>>1, a2, on);
    Complex Wn(cos(2.0*Pi/tlimit), on*sin(2.0*Pi/tlimit)), w(1, 0);
    for(int i = 0; i < (tlimit>>1); i ++, w = w*Wn) {
        auto t = w * a2[i];
```

```
        a[i] = a1[i] + t;
        a[i+(tlimit>>1)] = a1[i] - t;
    }
}
// 迭代版本 Faster
void FFT(Complex a[], int on) {
    for(int i = 0; i < limit; i ++)
        if(i < R[i]) swap(a[i], a[R[i]]);
    for(int mid = 1; mid < limit; mid <<= 1) {
        Complex Wn(cos(Pi/mid), on*sin(Pi/mid));
        for(int j = 0, r = mid<<1; j < limit; j += r) {
            Complex w(1, 0);
            for(int k = 0; k < mid; k ++, w = w*Wn) {
                Complex x = a[j+k], y = w*a[j+mid+k];
                a[j+k] = x+y;
                a[j+mid+k] = x-y;
            }
        }
    }
}

void solve() {
    cin >> n >> m;
    for(int i = 0; i <= n; i ++)
        cin >> a[i].x;
    for(int i = 0; i <= m; i ++)
        cin >> b[i].x;
    limit = 1;
    while(limit <= n+m) limit <<= 1;
    FFT(limit, a, 1), FFT(limit, b, 1);
    for(int i = 0; i < limit; i ++)
        a[i] = a[i]*b[i];
    FFT(limit, a, -1);
    for(int i = 0; i <= n+m; i ++){
        cout << (int)(a[i].x/limit+0.5) << " \n"[i == n+m];
    }
}
```

## NTT

```
const int N = 4e6+10, MO = 998244353, G = 3, Gi = 332748118;

int n, m, limit = 1, rr;
int R[N];
int a[N], b[N];

int qpm(int a, int b, const int &c = MO) {
    int ans = 1;
    while(b) {
        if(b & 1) ans = 1LL*ans*a%MO;
        a = 1LL*a*a%MO;
        b >>= 1;
    }
    return ans;
}
void NTT(int a[], int on) {
    for(int i = 0; i < limit; i ++)
```

```
        if(i < R[i]) swap(a[i], a[R[i]]);
    for(int mid = 1; mid < limit; mid <<= 1) {
        int Wn = qpm((on == 1 ? G : Gi), (MO-1)/(mid<<1));
        for(int j = 0, r = mid<<1; j < limit; j += r) {
            int w = 1;
            for(int k = 0; k < mid; k ++, w = 1LL*w*Wn%MO) {
                int x = a[j+k], y = 1LL*w*a[j+mid+k]%MO;
                a[j+k] = (1LL*x+y)%MO;
                a[j+mid+k] = (1LL*x-y)%MO;
            }
        }
    }
}
void solve() {
    cin >> n >> m;
    for(int i = 0; i <= n; i ++) {
        cin >> a[i];
        a[i] %= MO;
    }
    for(int i = 0; i <= m; i ++) {
        cin >> b[i];
        b[i] %= MO;
    }
    limit = 1, rr = 0;
    while(limit <= n+m) limit <<= 1, rr ++;
    for(int i = 0; i < limit; i ++)
        R[i] = (R[i>>1]>>1) | ((i&1)<<(rr-1));
    NTT(a, 1), NTT(b, 1);
    for(int i = 0; i <= limit; i ++)
        a[i] = 1LL*a[i]*b[i]%MO;
    NTT(a, -1);
    int inv = qpm(limit, MO-2);
    for(int i = 0; i <= n+m; i ++){
        a[i] = (1LL*a[i]*inv%MO+MO)%MO;
    }
}
```

## 线性筛

```
vector<int> primes, minp, phi;

void Sieve(int n) {
    primes.clear();
    minp.assign(n+1, 0);
    phi.resize(n+1);
    phi[1] = 0;
    for(int i = 2; i <= n; i ++) {
        if(!minp[i]) {
            minp[i] = i;
            primes.push_back(i);
            phi[i] = i-1;
        }
        for(auto p : primes) {
            if(p > n/i) {
                break;
            }
            minp[i*p] = p;
```

```
            if(p == minp[i]) {
                phi[t] = phi[i]*p;
                break;
            }
            phi[t] = phi[i]*(p-1);
        }
    }
}
```

# 博弈论

## 二分图博弈

### 第一类

情形：在二分图上某点放置一枚棋子，双方轮流移动棋子，且不能将棋子移动到已经走过的点。

（不能经过重复状态，如果是二分图）

结论：如果初始点**不必定**在最大匹配上，则先手必败。

做法：

1. 判断初始点 H 是否必定在最大匹配上，用 Dinic，在建图时把涉及 H 点的边存下来，跑完第一次 Dinic后再建这些边，第二次 Dinic 看有没有增加流量。

   适合于单起点，多起点时间复杂度较高。

2. 判断哪些点不一定在最大匹配中，则剩余点一定在最大匹配中。

### 第二类

情形：在n*m的矩阵上有两个相同的棋子，有些格子是不能走的。现在两个人开始轮流对棋子操作：每次只能将其中一颗棋子向四个方向移动一格，不能移出边界，谁将两个棋子重合谁将胜利；同时移动时不能出现前面移动时出现过的局面，不能移动的就算是失败了。

## 树上删边游戏

情形：给定一棵 n 个节点的有根树，每次可以删一个子树。

结论：则叶子节点的 SG 值为 0，非叶子节点的 SG 值为其所有子节点的 (SG值 + 1) 的异或和，即 $SG(u) = Xor_{v \in son(u)} SG(v) + 1$ 。

## 无向图删边游戏

把奇环缩成一个点加一条新边，把偶环缩成一个点，不影响 SG，然后套用树上删边游戏。

```
```

# 计算几何

```
typedef double db; // long double?

const db eps = 1e-8;
const db PI = acos(-1);

// Remind: int, double? LL in Point/abs2/dot/cross?
```

```cpp
int sign(db a) {
    return (a < -eps ? -1 : (a > eps));
}
int dcmp(db a, db b) {
    return sign(a-b);
}
// Point
struct Point {
    db x, y;
    Point(db x = 0, db y = 0) : x(x), y(y) {}
    Point operator - (const Point &p) {
        return Point(-p.x, -p.y);
    }
    friend Point operator + (const Point &a, const Point &b) {
        return Point(a.x+b.x, a.y+b.y);
    }
    friend Point operator - (const Point &a, const Point &b) {
        return Point(a.x-b.x, a.y-b.y);
    }
    Point operator * (const db &v) const {
        return Point(x*v, y*v);
    }
    Point operator / (const db &v) const {
        assert(v != 0);
        return Point(x/v, y/v);
    }
    bool operator < (const Point &o) const {
        int c = dcmp(x, o.x);
        if(c) return c == -1;
        return dcmp(y, o.y) == -1;
    }
    bool operator == (const Point &o) const {
        return dcmp(x, o.x) == 0 && dcmp(y, o.y) == 0;
    }
    db dist(const Point &o) {
        return (*this-o).abs();
    }
    db alpha() {
        return atan2(y, x);
    }
    void read() {
        cin >> x >> y;
    }
    void write() {
        cout << '(' << x << ',' << y << ')' << '\n';
    }
    db abs() {
        return sqrt(abs2());
    }
    db abs2() {
        return x*x+y*y;
    }
    Point rot90() {
        return Point(-y, x);
    }
    Point unit() {
        return *this/abs();
    }
```

```
};
LL dot(const Point &a, const Point &b) {
    return 1LL*a.x*b.x + 1LL*a.y*b.y;
}
LL cross(const Point &a, const Point &b) {
    return 1LL*a.x*b.y - 1LL*a.y*b.x;
}
// Line
struct Line {
    // a -> b
    Point a, b;
    Line(Point _a = Point(), Point _b = Point()) : a(_a), b(_b) {}
};
bool PointOnLineLeft(const Point &p, const Line &l) {
    return cross(l.b-l.a, p-l.a) > 0;
}
bool PointOnLineLeftNonStrict(const Point &p, const Line &l) {
    return cross(l.b-l.a, p-l.a) >= 0;
}
Point LineIntersection(const Line &l1, const Line &l2) {
    return l1.a + (l1.b-l1.a) * (cross(l2.b-l2.a, l1.a-l2.a) / cross(l2.b-l2.a,
l1.a-l1.b));
}
bool PointOnSegment(const Point &p, const Line &l) {
    return sign(cross(p-l.a, l.b-l.a)) == 0 && min(l.a.x, l.b.x) <= p.x && p.x
<= max(l.a.x, l.b.x)
        && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y);
}
// Polygon
db perimeter(const vector<Point> &poly) {
    int n = poly.size();
    db ret = 0;
    for(int i = 0; i < n; i ++) {
        ret += (poly[i]-poly[(i+1)%n]).abs();
    }
    return ret;
}
db area(const vector<Point> &poly) {
    db ret = 0;
    int n = poly.size();
    for(int i = 0; i < n; i ++) {
        ret += cross(poly[i], poly[(i+1)%n]);
    }
    return ret / 2;
}
// 射线法
int PointInPolygonLine(const Point &p, const vector<Point> &poly) {
    // -1: out, 0: on, 1: in
    int n = poly.size(), ret = 0;
    for(int i = 0; i < n; i ++) {
        Point u = poly[i], v = poly[(i+1)%n];
        if(PointOnSegment(p, Line(u, v))) {
            return 0;
        }
        if(dcmp(u.y, v.y) <= 0) {
            swap(u, v);
        }
        if(dcmp(p.y, u.y) > 0 || dcmp(p.y, v.y) <= 0) {
```

```
            continue;
        }
        ret ^= PointOnLineLeft(p, Line(u, v));
    }
    if(ret == 1) return 1;
    return -1;
}
// 转角法（优化）
int PointInPolygonAngle(const Point &p, const vector<Point> &poly) {
    // -1: out, 0: on, 1: in
    int n = poly.size(), wn = 0;
    for(int i = 0; i < n; i++) {
        Point u = poly[i], v = poly[(i+1)%n];
        if(PointOnSegment(p, Line(u, v))) {
            return 0;
        }
        bool k = PointOnLineLeft(p, Line(u, v));
        int d1 = sign(u.y - p.y);
        int d2 = sign(v.y - p.y);
        if(!k && d1 > 0 && d2 <= 0) wn ++;
        if(k && d1 <= 0 && d2 > 0) wn --;
    }
    if(wn != 0) return 1;
    return -1;
}
// ConvexHull
vector<Point> ConvexHull(vector<Point> p) {
    int n = p.size();
    if(n <= 1) return p;
    sort(all(p));
    vector<Point> q(n<<1);
    int k = 0;
    for(int i = 0; i < n; q[k ++] = p[i ++]) {
        while(k > 1 && !PointOnLineLeft(p[i], Line(q[k-2], q[k-1]))) {
            k --;
        }
    }
    for(int i = n-2, t = k; i >= 0; q[k ++] = p[i --]) {
        while(k > t && !PointOnLineLeft(p[i], Line(q[k-2], q[k-1]))) {
            k --;
        }
    }
    q.resize(k-1);
    return q;
}
vector<Point> ConvexHullNonStrict(vector<Point> p) {
    // 有时有精度误差，已知周长，慎用
    int n = p.size();
    if(n <= 1) return p;
    sort(all(p)); // assert unique!
    p.resize(unique(all(p)) - p.begin());
    n = p.size();
    vector<Point> q(n<<1);
    int k = 0;
    for(int i = 0; i < n; q[k ++] = p[i ++]) {
        while(k > 1 && !PointOnLineLeftNonStrict(p[i], Line(q[k-2], q[k-1]))) {
            k --;
        }
```

```
    }
    for(int i = n-2, t = k; i >= 0; q[k ++] = p[i --]) {
        while(k > t && !PointOnLineLeftNonStrict(p[i], Line(q[k-2], q[k-1]))) {
            k --;
        }
    }
    q.resize(k-1);
    return q;
}
int PointInConvex(const Point &p, const vector<Point> &poly) {
    // counter-clockwise
    // -1: out, 0: on, 1: in
    if(PointOnLineLeft(p, Line(poly[1], poly[0])) ||
        PointOnLineLeft(p, Line(poly[0], poly.back())))
        return -1;
    if(PointOnSegment(p, Line(poly[0], poly.back()))) return 0;
    int n = poly.size();
    int l = 1, r = n-1;
    while(l < r) {
        int mid = l+r >> 1;
        if(PointOnLineLeft(p, Line(poly[0], poly[mid]))) l = mid+1;
        else r = mid;
    }
    if(PointOnSegment(p, Line(poly[r-1], poly[r]))) return 0;
    if(PointOnLineLeft(p, Line(poly[r-1], poly[r]))) return 1;
    return -1;
}
```

三维中直线的交点，先看两个坐标系的能不能交，然后再用算出的交点，判断是不是真的能交。

## 两球体积的交

```
double GetVol(Point a, Point b) {
    double dist2 = GetDist2(a, b);
    if(dist2 >= a.r+b.r) return 0;

    if(dist2+a.r <= b.r) return 4.0/3.0*PI*a.r*a.r*a.r;
    if(dist2+b.r <= a.r) return 4.0/3.0*PI*b.r*b.r*b.r;

    /*
    double dx = (a.r*a.r-b.r*b.r)/dist2;
    double L = (dist2+dx)/2.0, l = dist2-L;
    double x1 = a.r - L, x2 = b.r - l;
    double res = PI*x1*x1*(a.r - x1 / 3.0);
    res += PI*x2*x2*(b.r - x2 / 3.0);
    */

    double cosA = (a.r*a.r+dist2*dist2-b.r*b.r)/(2.0*dist2*a.r);
    double hA = a.r*(1.0-cosA);
    double res = PI*hA*hA*(3.0*a.r-hA)/3.0;

    double cosB = (b.r*b.r+dist2*dist2-a.r*a.r)/(2.0*dist2*b.r);
    double hB = b.r*(1.0-cosB);
    res += PI*hB*hB*(3.0*b.r-hB)/3.0;

    return res;
}
```

## 矩形中最大的正三角形

```
// 二分长度，用alpha角算出最宽多么宽，判断是否合法
double a, b;
cin >> a >> b;
if(a < b) swap(a, b);
if(b * 2 <= a * sqrt(3.0)) {
    cout << b*2/sqrt(3.0) << '\n';
    return;
}
double l = b, r = b*2/sqrt(3.0);
while(r-l > 1e-12) {
    double mid = (l+r) / 2;
    double alpha = acos(b / mid);
    alpha = PI/6 - alpha;
    if(mid*cos(alpha) <= a) l = mid;
    else r = mid;
}
cout << r << '\n';
// 数学 - O(1)
int a, b;
cin >> a >> b;
if(a > b) swap(a, b);
if(b >= a*2/sqrt(3)) {
    cout << a*2/sqrt(3);
    return;
}
cout << 2 * sqrt(a*a+b*b-sqrt(3)*a*b) << '\n';
```

## 扫描线

```
// 面积并
const int N = 1e5+10;

struct ScanLine {
    int l, r, h, val;
    bool operator < (const ScanLine &rhs) const {
        if(h == rhs.h) return val > rhs.val;
        return h < rhs.h;
    }
};
struct Info {
    int l, r, sum, len;
};

int n, nn, tot;
vector<ScanLine> lines;
vector<int> xx;

struct SegmentTree {
    Info tr[N<<3];
    #define ls(u) (u<<1)
    #define rs(u) (u<<1|1)

    void pushUp(int u, bool in) {
```

```
        int l = tr[u].l, r = tr[u].r;
        if(tr[u].sum) {
            tr[u].len = xx[r+1] - xx[l];
        }
        else if(in) {
            tr[u].len = tr[ls(u)].len + tr[rs(u)].len;
        }
        else {
            tr[u].len = 0;
        }
    }
    void update(int u) {
        pushUp(u, tr[u].l != tr[u].r);
    }
    void build(int u, int l, int r) {
        tr[u] = {l, r, 0, 0};
        if(l == r) return;
        int mid = l+r >> 1;
        build(ls(u), l, mid), build(rs(u), mid+1, r);
        update(u);
    }
    void modify(int u, int l, int r, int c) {
        int ll = tr[u].l, rr = tr[u].r;
        if(xx[rr+1] <= l || r <= xx[ll]) return;
        if(l <= xx[ll] && xx[rr+1] <= r) {
            tr[u].sum += c;
            update(u);
            return;
        }
        int mid = ll+rr >> 1;
        if(l <= xx[mid+1]) modify(ls(u), l, r, c);
        if(r > xx[mid+1]) modify(rs(u), l, r, c);
        update(u);
    }
    #undef ls(u) (u<<1)
    #undef rs(u) (u<<1|1)
}sgt;


void solve() {
    cin >> n;
    nn = n << 1;
    xx.resize(nn+1), lines.resize(nn);
    for(int i = 0; i < n; i ++) {
        int a1, b1, a2, b2;
        cin >> a1 >> b1 >> a2 >> b2;
        xx[(i<<1)+1] = a1, xx[(i<<1|1)+1] = a2;
        lines[i<<1] = {a1, a2, b1, 1};
        lines[i<<1|1] = {a1, a2, b2, -1};
    }
    sort(1+all(xx));
    xx.resize(unique(1+all(xx)) - xx.begin());
    tot = xx.size()-1;
    sort(all(lines));

    sgt.build(1, 1, tot-1);
    int lh = lines[0].h;
    LL ans = 0;
```

```
    for(auto [l, r, h, val] : lines) {
        ans += 1LL*sgt.tr[1].len * (h - lh);
        lh = h;
        sgt.modify(1, l, r, val);
    }
    cout << ans << '\n';
}
```

```
// 周长并
const int N = 5010;

struct ScanLine {
    int l, r, h, val;
    bool operator < (const ScanLine &rhs) const {
        if(h == rhs.h) return val > rhs.val; // 先加后减
        return h < rhs.h;
    }
};
struct Info {
    int l, r, sum, len, segc;
    bool lc, rc;
};

int n, nn, tot;
vector<ScanLine> lines;
vector<int> xx;

struct SegmentTree {
    // 与求面积并相同
    void pushUp(int u, bool in) {
        int l = tr[u].l, r = tr[u].r;
        if(tr[u].sum) {
            tr[u].len = xx[r+1] - xx[l];
            tr[u].lc = tr[u].rc = true;
            tr[u].segc = 1;
        }
        else if(in) {
            tr[u].len = tr[ls(u)].len + tr[rs(u)].len;
            tr[u].lc = tr[ls(u)].lc, tr[u].rc = tr[rs(u)].rc;
            tr[u].segc = tr[ls(u)].segc + tr[rs(u)].segc;
            tr[u].segc -= tr[ls(u)].rc && tr[rs(u)].lc;
        }
        else {
            tr[u].len = 0;
            tr[u].lc = tr[u].rc = false;
            tr[u].segc = 0;
        }
    }
    void update(int u) {
        // 与求面积并相同
    }
    void build(int u, int l, int r) {
        tr[u] = {l, r, 0, 0, 0, false, false};
        // 与求面积并相同
    }
    void modify(int u, int l, int r, int c) {
        // 与求面积并相同
```

```
    }
    // 与求面积并相同
}sgt;


void solve() {
    // 与求面积并相同
    int lh = lines[0].h, llen = 0;
    int ans = 0;
    for(auto [l, r, h, val] : lines) {
        ans += 2 * sgt.tr[1].segc * (h-lh);
        sgt.modify(1, l, r, val);
        ans += abs(sgt.tr[1].len - llen);
        lh = h, llen = sgt.tr[1].len;
    }
    cout << ans << '\n';
}
```

# 杂

## 不怎么用，但是需要用的语法

```
// 赋值函数
// void的递归auto有问题。有引用，带引用
std::function<void(int, int, int &)>
function<返回值(类型1, 类型2)> dfs = [&](类型1 a, 类型2 b) {
    ...
}
// 好用的库函数
to_string()
stoi(), stol, stoll(string, [idx], [base]) // string -> int, long, long long
iota(a, a+n, 0) // 0, 1 ... n-1
__builtin_popcountll // 二进制中1的个数，末尾是ll就是统计ULL类型
gcd, __gcd // gcd返回的是a，b绝对值的最大公因数，__gcd是不管正负的，有时需要判一下。
// gcd在c++17后可用，但是好像linux下的编译器可以。（win下好像还是要看编译器版本）
__lg() // log_2, [u]int,[u]LL
log() // template
sqrtf(), sqrt(), sqrtl() // float, double, long double
accumulate(a.begin(), a.end(), 0); // 中间运算过程的数据类型由第三个参数的数据类型决定!
// bitset
[bitset].to_string(), to_ulong(), to_ullong()
// 记得使用 0LL / LL(0) / double(0) ..., 内部实现过程是 init = init + *first
// 结构体初始化
struct Info {
    LL key, value;
    Info(LL key, LL value) : key(key), value(value) {}
};
// 编译参数
-fsanitize=undefined 检查未定义行为（如整数溢出）
-fsanitize=address 检查数组越界
-fno-sanitize-recover=undefined 一有整数溢出直接把程序停下（如果没有巨大多输出这个不加也
行）
-D_GLIBCXX_DEBUG 检查误用STL
-Wall -Wextra 不解释
```

```
-Wconversion 检查把long long赋值给int这种东西
```

## 快读快写

```
__int128 read() {
    __int128 x = 0, f = 1;
    char ch = cin.get(); // getchar();
    while(ch < '0' || ch > '9') {
        if(ch == '-') f = -1;
        ch = cin.get(); // getchar();
    }
    while('0' <= ch && ch <= '9')
        x = (x<<3)+(x<<1)+ch-'0', ch = cin.get();  // getchar();
    return x * f;
}
void write(__int128 x) {
    static int _stk_[40];
    int top = 0;
    do {
        _stk_[++ top] = x % 10, x /= 10;
    } while(x);
    while(top)
        cout << (_stk_[top --] + '0');
}
```

## 模数类

```
template<class T>
T qpm(T a, LL b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
LL mul(LL a, LL b, LL p) {
    LL res = a * b - LL(1.L * a * b / p) * p;
    res %= p;
    if (res < 0) {
        res += p;
    }
    return res;
}
template<LL P>
struct MLong {
    LL x;
    MLong() : x{} {}
    MLong(LL x) : x{norm(x % getMod())} {}

    static LL Mod;
    static LL getMod() {
        if (P > 0) {
            return P;
        } else {
```

```
            return Mod;
        }
    }
    static void setMod(LL Mod_) {
        Mod = Mod_;
    }
    LL norm(LL x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
        return x;
    }
    LL val() const {
        return x;
    }
    explicit operator LL() const {
        return x;
    }
    MLong operator - () const {
        MLong res;
        res.x = norm(getMod() - x);
        return res;
    }
    MLong inv() const {
        assert(x != 0);
        return qpm(*this, getMod() - 2);
    }
    MLong &operator *= (MLong rhs) & {
        x = mul(x, rhs.x, getMod());
        return *this;
    }
    MLong &operator += (MLong rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    MLong &operator -= (MLong rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    MLong &operator /= (MLong rhs) & {
        return *this *= rhs.inv();
    }
    friend MLong operator * (MLong lhs, MLong rhs) {
        MLong res = lhs;
        res *= rhs;
        return res;
    }
    friend MLong operator + (MLong lhs, MLong rhs) {
        MLong res = lhs;
        res += rhs;
        return res;
    }
    friend MLong operator - (MLong lhs, MLong rhs) {
        MLong res = lhs;
        res -= rhs;
```

```
        return res;
    }
    friend MLong operator / (MLong lhs, MLong rhs) {
        MLong res = lhs;
        res /= rhs;
        return res;
    }
    friend std::istream &operator >> (std::istream &is, MLong &a) {
        LL v;
        is >> v;
        a = MLong(v);
        return is;
    }
    friend std::ostream &operator << (std::ostream &os, const MLong &a) {
        return os << a.val();
    }
    friend bool operator == (MLong lhs, MLong rhs) {
        return lhs.val() == rhs.val();
    }
    friend bool operator != (MLong lhs, MLong rhs) {
        return lhs.val() != rhs.val();
    }
};

template<>
LL MLong<0LL>::Mod = 1;

template<int P>
struct MInt {
    int x;
    MInt() : x{} {}
    MInt(LL x) : x{norm(x % getMod())} {}

    static int Mod;
    static int getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    static void setMod(int Mod_) {
        Mod = Mod_;
    }
    int norm(int x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
        return x;
    }
    int val() const {
        return x;
    }
    explicit operator int() const {
        return x;
```

```
    }
    MInt operator - () const {
        MInt res;
        res.x = norm(getMod() - x);
        return res;
    }
    MInt inv() const {
        assert(x != 0);
        return qpm(*this, getMod() - 2);
    }
    MInt &operator *= (MInt rhs) & {
        x = 1LL * x * rhs.x % getMod();
        return *this;
    }
    MInt &operator += (MInt rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    MInt &operator -= (MInt rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    MInt &operator /= (MInt rhs) & {
        return *this *= rhs.inv();
    }
    friend MInt operator * (MInt lhs, MInt rhs) {
        MInt res = lhs;
        res *= rhs;
        return res;
    }
    friend MInt operator + (MInt lhs, MInt rhs) {
        MInt res = lhs;
        res += rhs;
        return res;
    }
    friend MInt operator - (MInt lhs, MInt rhs) {
        MInt res = lhs;
        res -= rhs;
        return res;
    }
    friend MInt operator / (MInt lhs, MInt rhs) {
        MInt res = lhs;
        res /= rhs;
        return res;
    }
    friend std::istream &operator >> (std::istream &is, MInt &a) {
        LL v;
        is >> v;
        a = MInt(v);
        return is;
    }
    friend std::ostream &operator << (std::ostream &os, const MInt &a) {
        return os << a.val();
    }
    friend bool operator == (MInt lhs, MInt rhs) {
        return lhs.val() == rhs.val();
    }
    friend bool operator != (MInt lhs, MInt rhs) {
```

```
        return lhs.val() != rhs.val();
    }
};

template<>
int MInt<0>::Mod = 1;

template<int V, int P>
MInt<P> CInv = MInt<P>(V).inv();

const int MO = 998244353;
using Z = MInt<MO>;
```

## 分式类

如果是丢map，可以直接丢，因为是判等有没有出现，所以只要值一样，就不会重复插入。

因为过程中没有化简，很可能溢出，不太懂估计，但是常数小。

```
template<class T>
struct Frac {
    T num;
    T den;
    Frac(T num_, T den_) : num(num_), den(den_) {
        if (den < 0) {
            den = -den;
            num = -num;
        }
    }
    Frac() : Frac(0, 1) {}
    Frac(T num_) : Frac(num_, 1) {}
    void reduce() {
        T g = __gcd(num, den);
        num /= g;
        den /= g;
        if(den < 0) {
            num = -num;
            den = -den;
        }
    }
    explicit operator double() const {
        return 1. * num / den;
    }
    Frac &operator += (const Frac &rhs) {
        num = num * rhs.den + rhs.num * den;
        den *= rhs.den;
        return *this;
    }
    Frac &operator -= (const Frac &rhs) {
        num = num * rhs.den - rhs.num * den;
        den *= rhs.den;
        return *this;
    }
    Frac &operator *= (const Frac &rhs) {
        num *= rhs.num;
        den *= rhs.den;
        return *this;
```

```
    }
    Frac &operator /= (const Frac &rhs) {
        num *= rhs.den;
        den *= rhs.num;
        if(den < 0) {
            num = -num;
            den = -den;
        }
        return *this;
    }
    friend Frac operator + (Frac lhs, const Frac &rhs) {
        return lhs += rhs;
    }
    friend Frac operator - (Frac lhs, const Frac &rhs) {
        return lhs -= rhs;
    }
    friend Frac operator * (Frac lhs, const Frac &rhs) {
        return lhs *= rhs;
    }
    friend Frac operator / (Frac lhs, const Frac &rhs) {
        return lhs /= rhs;
    }
    friend Frac operator - (const Frac &a) {
        return Frac(-a.num, a.den);
    }
    friend bool operator == (const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den == rhs.num * lhs.den;
    }
    friend bool operator != (const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den != rhs.num * lhs.den;
    }
    friend bool operator < (const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den < rhs.num * lhs.den;
    }
    friend bool operator > (const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den > rhs.num * lhs.den;
    }
    friend bool operator <= (const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den <= rhs.num * lhs.den;
    }
    friend bool operator >= (const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den >= rhs.num * lhs.den;
    }
    friend std::ostream &operator << (std::ostream &os, Frac x) {
        T g = __gcd(x.num, x.den);
        if(x.den == g) {
            return os << x.num / g;
        }
        else {
            return os << x.num / g << "/" << x.den / g;
        }
    }
};

using F = Frac<__int128>;
```

## 哈希

```
template<int P>
struct HashTable {
    struct Info {
        LL key, value;
        Info(LL key, LL value) : key(key), value(value) {}
    };
    vector<vector<Info>> data;
    void init() {
        data.assign(P+1, vector<Info>());
    }
    int hash(LL key) {
        return (key % P + P) % P; }
    int get(LL key) {
        int keyHash = hash(key);
        for(auto &[k, v] : data[keyHash]) {
            if(k == key) {
                return v;
            }
        }
        return -1;
    }
    LL modify(LL key, LL value) {
        int keyHash = hash(key);
        for(auto &[k, v] : data[keyHash]) {
            if(k == key) {
                return v = value;
            }
        }
        return -1;
    }
    LL add(LL key, LL value) {
        if(get(key) != -1) return -1;
        int keyHash = hash(key);
        data[keyHash].emplace_back(key, value);
        return 1;
    }
};
HashTable<13331> ht;
```

## Meet in the Middle

```
// 区间放
void dfs(int cur, int goal, int sum, vector<int> &ans) { // int ans[], int &id
    if(goal == cur) { // >
        ans.push_back(sum);
        return;
    }
    for() {
        // 枚举
    }
}
// 根据集合大小动态放
void step(vector<LL> &ans, int p) {
    int siz = ans.size();
```

```
    for(int i = 0; i < siz; i ++) {
        LL x = ans[i];
        while(x*p <= n) {
            x *= p;
            ans.push_back(x);
        }
    }
};

// 上面 == 用开区间
dfs(0, n/2, 0, ans1);
dfs(n/2, n, 0, ans2);
// 上面 >  用闭区间
dfs(0, n/2-1, 0, ans1);
dfs(n/2, n-1, 0, ans2);
/**********或者************/
dfs(1, n/2, 0, ans1);
dfs(n/2+1, n, 0, ans2);
/* 不然会造成RE/MLE等情况，因为不是一半分
    就记忆下面的这种就行，上面是都-1。
    ！最终是为了状态数平分，所以划分可以一个一个进来，
    判断两个集合的大小，看放在哪里，或者先跑，
    然后找到一个不错的划分线                        */
```

## 模拟退火

```
const int N = 110;

mt19937 rnd(time(0));
int n;
vector<PDD> p;
PDD ansp;
double ans = 1e8;

double Rand(double l, double r) {
    return (double)rnd() / RAND_MAX * (r-l) + l;
}
double GetDist(PDD a, PDD b) {
    double dx = a.fi-b.fi, dy = a.se-b.se;
    return sqrt(dx*dx + dy*dy);
}
double Calc(PDD a) {
    double res = 0;
    for(auto x : p) {
        res += GetDist(a, x);
    }
    ans = min(ans, res);
    return res;
}
void SimulateAnneal() {
    double t = 1e4;
    PDD curp = ansp;
    while(t > 1e-4) {
        PDD newp(Rand(curp.fi-t, curp.fi+t), Rand(curp.se-t, curp.se+t));
        double delta = Calc(newp) - Calc(curp);
        if(exp(-delta/t) > Rand(0, 1))
            curp = newp;
```

```
        t *= 0.985;
    }
}
void solve() {
    srand(time(0));
    cin >> n;
    p.resize(n);
    for(auto &[x, y] : p) {
        cin >> x >> y;
        ansp.fi += x, ansp.se += y;
    }
    ansp.fi /= n, ansp.se /= n;
    // for(int i = 0; i < 100; i++)
    while ((double)clock()/CLOCKS_PER_SEC < 0.8)
        SimulateAnneal();
    cout << ans << '\n';
}
/*
    T 一般状态数的一半，结束T一般为精度要求-2位
    1. 普通类函数SA，可以当前最优为起点，开始可以取中心。
    2. 分组SA - 先random_shuffle序列，在加贪心/DP
    3. 排序SA - swap两个变化较小，算有连续性，check下满足性质。。
    4. 生成树SA - 先生成，然后改边。
*/
```

## 莫队

```
struct Rec {
    int l, r, qid;
};

int n, m, l, r;
LL tans;
int bNum, bSize;
vector<int> a, belong, cnt;
vector<Rec> query;

LL f(int c) {
    return 1LL*c*(c-1)*(c-2)/6;
}
void add(int x) {
    auto &c = cnt[a[x]];
    tans -= f(c);
    c ++;
    tans += f(c);
}
void del(int x) {
    auto &c = cnt[a[x]];
    tans -= f(c);
    c --;
    tans += f(c);
}
void solve() {
    cin >> n >> m;
    a.resize(n+1), belong.resize(n+1), query.resize(m);
    // bSize在 m < n时，取n/sqrt(m)，但在 m > n时，这样取可能会下取整导致bSize = 0，导致
RE。
```

```
    if(n > m) bSize = n/sqrt(m);
    else bSize = sqrt(n);
    bNum = (n-1)/bSize + 1;
    int mx = 0;
    for(int i = 1; i <= n; i ++) {
        cin >> a[i];
        mx = max(mx, a[i]);
        belong[i] = (i-1)/bSize + 1;
    }
    cnt.resize(mx+1);
    for(int i = 0; i < m; i ++) {
        auto &[l, r, qid] = query[i];
        cin >> l >> r;
        qid = i;
    }
    sort(all(query), [&](const Rec &a, const Rec &b) {
        int abl = belong[a.l], bbl = belong[b.l];
        if(abl != bbl) {
            return abl < bbl;
        }
        else {
            if(abl & 1) return a.r < b.r;
            else return a.r > b.r;
        }
    });

    vector<LL> anss(m);
    l = 1, r = 0;
    for(int i = 0; i < m; i ++) {
        auto [ll, rr, qid] = query[i];
        // 整体加减顺序，现在遇到的都没关系，但建议想一想。
        while(l < ll) del(l ++);
        while(l > ll) add(-- l);
        while(r < rr) add(++ r);
        while(r > rr) del(r --);
        anss[qid] = tans;
    }
    for(int i = 0; i < m; i ++) {
        cout << anss[i] << '\n';
    }
}
```

## 离散化

```
// 对大小没要求的也可以直接map记下标号，用val数组记下原本的值，（少一次sort，可能可以少一点点常数（
for(int i = 1; i <= n; i ++) {
    cin >> a[i];
    val[i] = a[i];
}
sort(1+all(val));
val.resize(unique(1+all(val)) - val.begin()); // 从大到小 greater<int>()
auto getVal = [&](int x) {
    return (lower_bound(1+all(val), x) - val.begin()); // 从大到小 greater<int>()
};
for(int i = 1; i <= n; i ++) {
    a[i] = getVal(a[i]);
}
```

## cdq分治

```
FenwickTree<int> fen;
struct Node {
    int a, b, c, cnt, res;
};

const int N = 1e5+10;

int n, nn, k;
Node p[N], np[N], tmp[N];

bool cmp1(const Node &a, const Node &b) {
    if(a.a == b.a) {
        if(a.b == b.b) {
            return a.c < b.c;
        }
        return a.b < b.b;
    }
    return a.a < b.a;
}
// [l, r]，无重复点。
void cdq(Node np[], int l, int r) {
    if(l == r) return;
    int mid = l+r >> 1, i = l, j = mid+1, k = l;
    cdq(l, mid), cdq(mid+1, r);
    while(i <= mid && j <= r) {
        if(np[i].b <= np[j].b) {
            fen.add(np[i].c, np[i].cnt);
            tmp[k++] = np[i++];
        }
        else {
            np[j].res += fen.sum(np[j].c);
            tmp[k++] = np[j++];
        }
    }
    while(j <= r) {
        np[j].res += fen.sum(np[j].c);
        tmp[k++] = np[j++];
    }
    for(int x = l; x < i; x ++) {
```

```
        fen.add(np[x].c, -np[x].cnt);
    }
    while(i <= mid) {
        tmp[k++] = np[i++];
    }
    for(int i = l; i <= r; i ++) {
        np[i] = tmp[i];
    }
}
void solve() {
    cin >> n >> k;
    fen.init(k);
    for(int i = 0; i < n; i ++) {
        auto &[a, b, c, cnt, res] = p[i];
        cin >> a >> b >> c;
        cnt = 1, res = 0;
    }
    sort(p, p+n, cmp1);
    np[nn++] = p[0];
    for(int i = 1; i < n; i ++) {
        auto &pp = np[nn-1];
        auto &cp = p[i];
        if(pp.a != cp.a || pp.b != cp.b || pp.c != cp.c) {
            np[nn++] = cp;
        }
        else {
            pp.cnt ++;
        }
    }
    cdq(np, 0, nn-1);
    vector<int> ans(n);
    for(int i = 0; i < nn; i ++) {
        int t = np[i].res + np[i].cnt - 1;
        ans[t] += np[i].cnt;
    }
    for(int i = 0; i < n; i ++) {
        cout << ans[i] << '\n';
    }
}
```