Stratosphere's XCPC Templates

南京大学

平流层 Stratosphere

October 17, 2024

Contents

0 Header

1 图论

1.1 欧拉回路

```
namespace Euler {
    bool directed;
    vector<pii> G[maxn];
    vector<int> ans;
    int vis[maxm];
    int dfs(int x) {
        vector<int> t;
        while (G[x].size()) {
            auto [to, id] = G[x].back();
            G[x].pop_back();
            if (!vis[abs(id)]) {
                vis[abs(id)] = 1, t.push_back(dfs(to)), ans.push_back(id);
            }
        }
        for (int i = 1; i < t.size(); i++) {
            if (t[i] != x) ans.clear();
        }
        return t.size() ? t[0] : x;
    }
    int n, m;
    pii e[maxm];
    int deg[maxn], vv[maxn];
    void clr() {
        for (int i = 1; i <= n; i++) G[i].clear(), deg[i] = vv[i] = 0;
        for (int i = 1; i <= m; i++) vis[i] = 0;
        ans.clear();
        n = m = 0;
    }
    void addedge(int x, int y) {
        ckmax(n, x), ckmax(n, y);
        e[++m] = {x, y};
        if (directed) {
            G[x].push_back({y, m});
            ++deg[x], --deg[y], vv[x] = vv[y] = 1;
        } else {
            G[x].push_back({y, m});
            G[y].push_back({x, -m});
            ++deg[x], ++deg[y], vv[x] = vv[y] = 1;
        }
    }
    using vi = vector<int>;
    pair<vi, vi> work() {
        if (!m) return clr(), pair<vi, vi>{{1}, {}};
        int S = 1;
        for (int i = 1; i <= n; i++)
            if (vv[i]) S = i;
        for (int i = 1; i <= n; i++)
            if (deg[i] > 0 && deg[i] % 2 == 1) S = i;
        dfs(S);
        if ((int)ans.size() != m) return clr(), pair<vi, vi>();
        reverse(ans.begin(), ans.end());
        vi ver, edge = ans;
        if (directed) {
            ver = {e[ans[0]].fi};
            for (auto t : ans) ver.push_back(e[t].se);
        } else {
            ver = {ans[0] > 0 ? e[ans[0]].fi : e[-ans[0]].se};
            for (auto t : ans) ver.push_back(t > 0 ? e[t].se : e[-t].fi);
        }
        clr();
        return {ver, edge};
    }
}  // namespace Euler
```

1.2 Tarjan-SCC

```
void tarjan(int u) {
    dfn[u] = low[u] = ++tim;
    in[u] = 1;
    st[++top] = u;
    for (int v : G[u]) {
        if (!dfn[v]) tarjan(v), ckmin(low[u], low[v]);
        else if (in[v]) ckmin(low[u], dfn[v]);
    }
    if (dfn[u] == low[u]) {
        ++totc;
        int x;
        do { x = st[top--], in[x] = 0, bel[x] = totc; } while (x != u);
    }
}
```

1.3 点双

```
int T;  // assign = n
void tarjan(int u, int fa) {
    dfn[u] = low[u] = ++tim;
    stk[++top] = u;
    for (int v : G[u]) {
        if (v == fa) continue;
        if (!dfn[v]) dfs(v, u), ckmin(low[u], low[v]);
        else ckmin(low[u], dfn[v]);
    }
    if (fa && low[u] >= dfn[fa]) {
        int y;
        ++T;
        do {
            y = stk[top--];
            G2[T].push_back(y), G2[y].push_back(T);
        } while (y != u);
        G2[T].push_back(fa), G2[fa].push_back(T);
    }
}
```

1.4 边双

```
void tarjan(int u, int f) {
    dfn[u] = low[u] = ++tim;
    st[++top] = u;
    for (int v : G[u]) {
        if (v == f) continue;
        if (!dfn[v]) tarjan(v, u), ckmin(low[u], low[v]);
        else ckmin(low[u], dfn[v]);
    }
    if (dfn[u] == low[u]) {
        ++totc;
        int x;
        do { x = st[top--], in[x] = 0, bel[x] = totc; } while (x != u);
    }
}
```

1.5 2-SAT

构造方案时可以通过变量在图中的拓扑序确定该变量的取值。
如果变量 x 的拓扑序在 $\neg x$ 之后, 那么取 x 值为真。
因为 Tarjan 算法求强连通分量时使用了栈, 所以 Tarjan 求得的 SCC 编号相当于反拓扑序。

```
for (int i = 1; i <= n; i++)
    if (bel[i << 1] == bel[i << 1 | 1]) return puts("IMPOSSIBLE"), 0;
puts("POSSIBLE");
for (int i = 1; i <= n; i++) printf("%d ", bel[i << 1] > bel[i << 1 | 1]);
```

1.6 最大流

Dinic 算法

```
namespace Dinic {
    int N, S, T;
    struct edge {
        int to, nxt, cap;
    } e[maxm << 1];
    int head[maxn], cur[maxn], tot = 1;
    int d[maxn];
    void addedge(int u, int v, int c) {
        e[++tot] = (edge){v, head[u], c}, head[u] = tot;
        e[++tot] = (edge){u, head[v], 0}, head[v] = tot;
    }
    bool bfs(int S, int T) {
        queue<int> q;
        for (int i = 1; i <= N; i++) d[i] = 0;
        d[S] = 1;
        q.push(S);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i = head[u]; i; i = e[i].nxt) {
                int v = e[i].to;
                if (e[i].cap && !d[v]) {
                    d[v] = d[u] + 1, q.push(v);
                    if (v == T) return true;
                }
            }
        }
        return false;
    }
    int dfs(int u, int f) {
        if (u == T) return f;
        int r = f;
        for (int& i = cur[u]; i && r; i = e[i].nxt) {
            int v = e[i].to;
            if (e[i].cap && d[v] == d[u] + 1) {
                int x = dfs(v, min(e[i].cap, r));
                if (!x) d[v] = 0;
                e[i].cap -= x, e[i ^ 1].cap += x;
                r -= x;
            }
        }
        return f - r;
    }
    ll work(int _N, int _S, int _T) {
        N = _N, S = _S, T = _T;
        ll ans = 0;
        while (bfs(S, T)) {
            for (int i = 1; i <= N; i++) cur[i] = head[i];
            ans += 1ll * dfs(S, INF);
        }
        return ans;
    }
}  // namespace Dinic
```

ISAP 算法

```
namespace ISAP {
    int N, S, T;
```

```cpp
  struct edge {
    int to, nxt, cap;
  } e[maxm << 1];
  int head[maxn], cur[maxn], gap[maxn], dis[maxn], tot = 1;
  void addedge(int u, int v, int w) {
    e[++tot] = {v, head[u], w}, head[u] = tot;
    e[++tot] = {u, head[v], 0}, head[v] = tot;
  }
  int ISAP(int u, int lim) {
    if (u == T) return lim;
    int res = 0;
    for (int& i = cur[u]; i; i = e[i].nxt) {
      int v = e[i].to;
      if (e[i].cap && dis[u] == dis[v] + 1) {
        ll det = ISAP(v, min(lim, e[i].cap));
        e[i].cap -= det, e[i ^ 1].cap += det;
        lim -= det, res += det;
        if (!lim) return res;
      }
    }
    cur[u] = head[u];
    if (!--gap[dis[u]]) dis[S] = N + 1;
    gap[++dis[u]]++;
    return res;
  }
  ll work(int _N, int _S, int _T) {
    S = _S, T = _T, N = _N;
    ll res = 0;
    while (dis[S] <= N) res += 1ll * ISAP(S, INF);
    return res;
  }
}  // namespace ISAP
```

HLPP 算法

```cpp
namespace HLPP {  // by ProjectEMmm
  int N, S, T;
  struct edge {
    int to, nxt, cap;
  } e[maxm << 1];
  int head[maxn], tot = 1;

  int d[maxn], num[maxn];
  stack<int> lib[maxn];
  ll ex[maxn];
  int level = 0;
  void addedge(int u, int v, int c) {
    e[++tot] = {v, head[u], c}, head[u] = tot;
    e[++tot] = {u, head[v], 0}, head[v] = tot;
  }
  int Push(int u) {
    bool init = (u == S);
    for (int i = head[u]; i; i = e[i].nxt) {
      const int &v = e[i].to, &c = e[i].cap;
      if (!c || init == false && d[u] != d[v] + 1) continue;
      ll k = init ? c : min((ll)c, ex[u]);
      if (v != S && v != T && !ex[v] && d[v] < INF)
        lib[d[v]].push(v), level = max(level, d[v]);
      ex[u] -= k, ex[v] += k, e[i].cap -= k, e[i ^ 1].cap += k;
      if (!ex[u]) return 0;
    }
    return 1;
  }
  void Relabel(int x) {
    d[x] = INF;
    for (int i = head[x]; i; i = e[i].nxt)
      if (e[i].cap) d[x] = min(d[x], d[e[i].to]);
    if (++d[x] < N) {
      lib[d[x]].push(x);
```

```
            level = max(level, d[x]);
            ++num[d[x]];
        }
    }
    bool BFS() {
        for (int i = 1; i <= N; ++i) {
            d[i] = INF;
            num[i] = 0;
        }
        queue<int> q;
        q.push(T), d[T] = 0;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            num[d[u]]++;
            for (int i = head[u]; i; i = e[i].nxt) {
                const int& v = e[i].to;
                if (e[i ^ 1].cap && d[v] > d[u] + 1) d[v] = d[u] + 1, q.push(v);
            }
        }
        return d[S] != INF;
    }
    int Select() {
        while (lib[level].size() == 0 && level > -1) level--;
        return level == -1 ? 0 : lib[level].top();
    }
    ll work(int _N, int _S, int _T) {
        N = _N, S = _S, T = _T;
        if (!BFS()) return 0;
        d[S] = N;
        Push(S);
        int x;
        while (x = Select()) {
            lib[level].pop();
            if (!Push(x)) continue;
            if (!--num[d[x]])
                for (int i = 1; i <= N; ++i)
                    if (i != S && i != T && d[i] > d[x] && d[i] < N + 1)
                        d[i] = N + 1;
            Relabel(x);
        }
        return ex[T];
    }
}  // namespace HLPP
```

1.7 最小费用最大流

```
namespace MCMF {
    using pr = pair<ll, int>;
    int N, S, T;
    struct edge {
        int to, nxt, cap, w;
    } e[maxm << 1];
    int head[maxn], tot = 1;
    void addedge(int x, int y, int cap, int w) {
        e[++tot] = {y, head[x], cap, w}, head[x] = tot;
        e[++tot] = {x, head[y], 0, -w}, head[y] = tot;
    }
    ll d[maxn], dis[maxn];
    int vis[maxn], fr[maxn];
    bool spfa() {
        queue<int> Q;
        fill(d + 1, d + N + 1, 1e18);  // CHECK
        for (d[S] = 0, Q.push(S); !Q.empty();) {
            int x = Q.front();
            Q.pop();
            vis[x] = 0;
```

```
            for (int i = head[x]; i; i = e[i].nxt)
                if (e[i].cap && d[e[i].to] > d[x] + e[i].w) {
                    d[e[i].to] = d[x] + e[i].w;
                    fr[e[i].to] = i;
                    if (!vis[e[i].to]) vis[e[i].to] = 1, Q.push(e[i].to);
                }
        }
        return d[T] < 1e17;  // 如果只是最小费用流，当d < 0继续增广
    }
    bool dijkstra() {  // 正常题目不需要 dijk
        priority_queue<pr, vector<pr>, greater<pr>> Q;
        for (int i = 1; i <= N; ++i)
            dis[i] = d[i], d[i] = 1e18, vis[i] = fr[i] = 0;  // CHECK
        Q.emplace(d[S] = 0, S);
        while (!Q.empty()) {
            int x = Q.top().second;
            Q.pop();
            if (vis[x]) continue;
            vis[x] = 1;
            for (int i = head[x]; i; i = e[i].nxt) {
                const ll v = e[i].w + dis[x] - dis[e[i].to];
                if (e[i].cap && d[e[i].to] > d[x] + v) {
                    fr[e[i].to] = i;
                    Q.emplace(d[e[i].to] = d[x] + v, e[i].to);
                }
            }
        }
        for (int i = 1; i <= N; ++i) d[i] += dis[i];  // CHECK
        return d[T] < 1e17;
    }
    std::pair<ll, ll> work(int _N, int _S, int _T) {
        N = _N, S = _S, T = _T;
        spfa();  // 如果初始有负权且要 dijk
        ll f = 0, c = 0;
        for (; dijkstra();) {  // 正常可以用 spfa
            ll fl = 1e18;
            for (int i = fr[T]; i; i = fr[e[i ^ 1].to])
                fl = min((ll)e[i].cap, fl);
            for (int i = fr[T]; i; i = fr[e[i ^ 1].to])
                e[i].cap -= fl, e[i ^ 1].cap += fl;
            f += fl, c += fl * d[T];
        }
        return make_pair(f, c);
    }
}  // namespace MCMF
```

1.8 匹配

1.8.1 二分图最大匹配-Hungary

```
// 匈牙利,左到右单向边, O (M |match|)
int vis[maxn], match[maxn];
bool dfs(int u) {
    for (int v : G[u]) {
        if (vis[v]) continue;
        vis[v] = 1;
        if (!match[v] || dfs(match[v])) return match[v] = u, 1;
    }
    return 0;
}
int work() {
    for (int i = 1; i <= nl; i++)
        if (dfs(i)) fill(vis + 1, vis + nr + 1, 0);
}
// 匈牙利,左到右单向边,bitset, O (n^2|match|/w)
bitset<N> G[N], unvis;
int match[N];
```

```
bool dfs(int u) {
	for (auto s = G[u];;) {
		s &= unvis;
		int v = s._Find_first();
		if (v == N) return 0;
		unvis.reset(v);
		if (!match[v] || dfs(match[v])) return match[v] = u, 1;
	}
	return 0;
}
int work() {
	unvis.set();
	for (int i = 1; i <= nl; i++)
		if (dfs(i)) unvis.set();
}
```

1.8.2 二分图最大匹配-HK

```
// HK, 左到右单向边, O(M \sqrt{|match|})
int nl, nr, m;
vi G[maxn];
int L[maxn], R[maxn], vis[maxn], matchl[maxn], matchr[maxn];
queue<int> Q;
bool bfs() {
	for (int i = 1; i <= nl; i++) L[i] = 0;
	for (int i = 1; i <= nr; i++) R[i] = 0;

	for (int i = 1; i <= nl; i++)
		if (!matchl[i]) L[i] = 1, Q.push(i);
	int succ = 0;
	while (!Q.empty()) {
		int u = Q.front();
		Q.pop();
		for (int v : G[u]) {
			if (R[v]) continue;
			R[v] = L[u] + 1;
			if (matchr[v]) {
				L[matchr[v]] = R[v] + 1, Q.push(matchr[v]);
			} else succ = 1;
		}
	}
	return succ;
}
bool dfs(int u) {
	for (int v : G[u])
		if (R[v] == L[u] + 1 && !vis[v]) {
			vis[v] = 1;
			if (!matchr[v] || dfs(matchr[v]))
				return matchl[u] = v, matchr[v] = u, 1;
		}
	return 0;
}
void HK() {
	while (bfs()) {
		for (int i = 1; i <= nr; i++) vis[i] = 0;
		for (int i = 1; i <= nl; ++i)
			if (!matchl[i]) dfs(i);
	}
	return;
}
```

1.8.3 二分图最大权匹配-KM

```
// KM 二分图最大权匹配 复杂度O(n^3)
namespace KM {
	int nl, nr;
```

```cpp
    ll e[maxn][maxn], lw[maxn], rw[maxn], mnw[maxn];
    int lpr[maxn], rpr[maxn], vis[maxn], fa[maxn];
    void addedge(int x, int y, ll w) { ckmax(e[x][y], w), ckmax(lw[x], w); }
    void work(int x) {
        int xx = x;
        for (int i = 1; i <= nr; i++) vis[i] = 0, mnw[i] = 1e18;
        while (true) {
            for (int i = 1; i <= nr; i++)
                if (!vis[i] && mnw[i] >= lw[x] + rw[i] - e[x][i])
                    ckmin(mnw[i], lw[x] + rw[i] - e[x][i]), fa[i] = x;
            ll mn = 1e18;
            int y = -1;
            for (int i = 1; i <= nr; i++)
                if (!vis[i] && mn >= mnw[i]) ckmin(mn, mnw[i]), y = i;
            lw[xx] -= mn;
            for (int i = 1; i <= nr; i++)
                if (vis[i]) rw[i] += mn, lw[rpr[i]] -= mn;
                else mnw[i] -= mn;
            if (rpr[y]) x = rpr[y], vis[y] = 1;
            else {
                while (y) rpr[y] = fa[y], swap(y, lpr[fa[y]]);
                return;
            }
        }
    }
    void init(int _nl, int _nr) {
        nl = _nl, nr = _nr;
        if (nl > nr) nr = nl;
        for (int i = 1; i <= nl; i++) lw[i] = -1e18;
        for (int i = 1; i <= nl; i++)
            for (int j = 1; j <= nr; j++) e[i][j] = 0;  // or -1e18
    }
    ll work() {
        for (int i = 1; i <= nl; i++) work(i);
        ll tot = 0;
        for (int i = 1; i <= nl; i++) tot += e[i][lpr[i]];
        return tot;
    }
}  // namespace KM
```

1.8.4 一般图最大匹配-带花树

```cpp
namespace blossom {
    vector<int> G[maxn];
    int f[maxn];
    int n, match[maxn];
    int getfa(int x) { return f[x] == x ? x : f[x] = getfa(f[x]); }
    void addedge(int x, int y) { G[x].push_back(y), G[y].push_back(x); }
    int pre[maxn], mk[maxn];
    int vis[maxn], T;
    queue<int> q;
    int LCA(int x, int y) {
        T++;
        for (;; x = pre[match[x]], swap(x, y))
            if (vis[x = getfa(x)] == T) return x;
            else vis[x] = x ? T : 0;
    }
    void flower(int x, int y, int z) {
        while (getfa(x) != z) {
            pre[x] = y;
            y = match[x];
            f[x] = f[y] = z;
            x = pre[y];
            if (mk[y] == 2) q.push(y), mk[y] = 1;
        }
    }
    void aug(int s) {
```

```
    for (int i = 1; i <= n; i++) pre[i] = mk[i] = vis[i] = 0, f[i] = i;
    q = {};
    mk[s] = 1;
    q.push(s);
    while (q.size()) {
      int x = q.front();
      q.pop();
      for (int v : G[x]) {
        int y = v, z;
        if (mk[y] == 2) continue;
        if (mk[y] == 1) z = LCA(x, y), flower(x, y, z), flower(y, x, z);
        else if (!match[y]) {
          for (pre[y] = x; y;)
            x = pre[y], match[y] = x, swap(y, match[x]);
          return;
        } else
          pre[y] = x, mk[y] = 2, q.push(match[y]), mk[match[y]] = 1;
      }
    }
  }
  int work() {
    for (int i = 1; i <= n; i++)
      if (!match[i]) aug(i);
    int res = 0;
    for (int i = 1; i <= n; i++) res += match[i] > i;
    return res;
  }
} // namespace blossom
```

1.8.5 一般图最大权匹配

待补充

1.9 流和匹配的建模技巧

1.9.1 二分图相关

- 二分图最小点覆盖：等于最大匹配 $|match|$。从每一个非匹配点出发，沿着非匹配边正向进行遍历，沿着匹配边反向进行遍历到的点进行标记。选取左部点中没有被标记过的点，右部点中被标记过的点，则这些点可以形成该二分图的最小点覆盖。
- 二分图最大独立集：等于 $n-|match|$，考虑最小点覆盖给所有边都至少有一边有点，取反后必然为最大独立集。
- 二分图最小边覆盖：等于 $n-|match|$，考虑最坏情况每个顶点都要一条边，一个匹配能减小 1 的贡献。
- 最大团：等于补图的最大独立集。
- 最小路径覆盖：对于每条有向边 (u,v)，拆成 $u \to v+n$，u 为进入 u，$v+n$ 为从 v 离开，则答案为 $n-|match|$。
- Hall Theorem: 对于左部顶点集 X，$\forall S \subseteq X, |N(S)| \ge |S| \iff$ 存在完美匹配。

1.9.2 网络流相关

- 二分图最大权独立集：考虑连边 (S,x,w_x)，原图边 (x,y,∞)，(y,T,w_y)，变为最小割。
- 最大权闭合子图：正权 w_u 连 (S,u,w_u)，负权 w_v 连 $(v,T,-w_v)$，原图边连 ∞。此时最小割之后源点 S 能到达的点即为最大权闭合子图，答案即为正权和 $-\text{mincut}$。
- 无源汇上下界可行流：建源汇 S,T，$l(u,v),r(u,v)$ 分别为流量上下界。记 $d(i)=\sum l(u,i)-\sum l(i,v)$。
  - 原边 (u,v) 连 $(u,v,r(u,v)-l(u,v))$。
  - 对于每个点 u，若 $d_u>0$，连 (S,u,d_u)。
  - 若 $d_u<0$，连 $(u,T,-d_u)$。

  若 S 的出边全部流满则存在解。
- 有源汇上下界可行流：原图源汇连边 $(T \to S,(0,\infty))$，则转化为无源汇。

- 有源汇上下界最大流：从 T 到 S 连一条下界为 0，上界为 $+\infty$ 的边，转化为无源汇网络。按照无源汇上下界可行流的做法求一次无源汇上下界超级源 SS 到超级汇 TT 的最大流。删去所有附加边，在上一步的**残量网络**基础上，求一次 S 到 T 的最大流。两者之和即为答案。
- 有源汇上下界最小流：从 T 到 S 连一条下界为 0，上界为 $+\infty$ 的边，转化为无源汇网络。按照无源汇上下界可行流的做法求一次无源汇上下界超级源 SS 到超级汇 TT 的最大流。删去所有附加边，在上一步的**残量网络**基础上，求一次 T 到 S 的最大流。两者之差即为答案。
- 最小费用可行流：同有源汇上下界可行流，在超级源汇跑最小费用最大流，答案为费用 + 下界流量的费用。
- 平面图最小割 = 对偶图最短路

1.10 最短路相关

1.10.1 差分约束

x_i 向 x_j 连一条权值为 c 的有向边表示 $x_j - x_i \le c$。
用 BF 判断是否存在负环, 存在即无解。

1.10.2 最小环

记原图中 u, v 之间边的边权为 $val\,(u, v)$。

我们注意到 Floyd 算法有一个性质：在最外层循环到点 k 时（尚未开始第 k 次循环），最短路数组 dis 中，$dis_{u,v}$ 表示的是从 u 到 v 且仅经过编号在 $[1, k)$ 区间中的点的最短路。

由最小环的定义可知其至少有三个顶点，设其中编号最大的顶点为 w，环上与 w 相邻两侧的两个点为 u, v，则在最外层循环枚举到 $k = w$ 时，该环的长度即为 $dis_{u,v} + val\,(v, w) + val\,(w, u)$。

故在循环时对于每个 k 枚举满足 $i < k, j < k$ 的 (i, j)，更新答案即可。

1.10.3 Steiner 树

状态设计：$dp(i, S)$ 以 i 为根，树中关键点集合为 S 的最小值。

1. 树根度数不为 1 ，考虑拆分成两个子集 $T, S - T$:
$$dp(i, S) \leftarrow dp(i, S - T) + dp(i, T)$$
2. 树根度数为 1 :
$$dp(i, S) \leftarrow dp(j, S) + w(i, j)$$

相当于超级源到每个顶点距离为 $dp(i, S)$，求到每个顶点的最短路，dij 即可。

1.11 三四元环计数

```cpp
static int id[maxn], rnk[maxn];
for (int i = 1; i <= n; i++) id[i] = i;
sort(id + 1, id + n + 1, [](int x, int y) {
    return pii{deg[x], x} < pii{deg[y], y};
});
for (int i = 1; i <= n; i++) rnk[id[i]] = i;
for (int i = 1; i <= n; i++)
    for (int v : G[i])
        if (rnk[v] > rnk[i]) G2[i].push_back(v);
int ans3 = 0;  // 3-cycle
for (int i = 1; i <= n; i++) {
    static int vis[maxn];
    for (int v : G2[i]) vis[v] = 1;
    for (int v1 : G2[i])
        for (int v2 : G2[v1])
            if (vis[v2]) ++ans3;  // (i,v1,v2)
    for (int v : G2[i]) vis[v] = 0;
}
ll ans4 = 0;  // 4-cycle
for (int i = 1; i <= n; i++) {
    static int vis[maxn];
    for (int v1 : G[i])
        for (int v2 : G2[v1])
```

```cpp
            if (rnk[v2] > rnk[i]) ans4 += vis[v2], vis[v2]++;
    for (int v1 : G[i])
        for (int v2 : G2[v1]) vis[v2] = 0;
}
```

1.12 支配树

```cpp
namespace Dom_DAG {
    int idom[maxn];
    vector<int> G[maxn], ANS[maxn];  // ANS: final tree
    int deg[maxn];
    int fa[maxn][25], dep[maxn];
    int lca(int x, int y) {
        if (dep[x] < dep[y]) swap(x, y);
        for (int i = 20; i >= 0; i--)
            if (fa[x][i] && dep[fa[x][i]] >= dep[y]) x = fa[x][i];
        if (x == y) return x;
        for (int i = 20; i >= 0; i--)
            if (fa[x][i] != fa[y][i]) x = fa[x][i], y = fa[y][i];
        return fa[x][0];
    }
    void work() {
        queue<int> q;
        q.push(1);
        while (!q.empty()) {
            int x = q.front();
            q.pop();
            ANS[idom[x]].push_back(x);
            fa[x][0] = idom[x];
            dep[x] = dep[idom[x]] + 1;
            for (int i = 1; i <= 20; i++) fa[x][i] = fa[fa[x][i - 1]][i - 1];
            for (int v : G[x]) {
                --deg[v];
                if (!deg[v]) q.push(v);
                if (!idom[v]) idom[v] = x;
                else idom[v] = lca(idom[v], x);
            }
        }
    }
}  // namespace Dom_DAG
namespace Dom {
    vector<int> G[maxn], rG[maxn];
    int dfn[maxn], id[maxn], anc[maxn], cnt;
    void dfs(int x) {
        id[dfn[x] = ++cnt] = x;
        for (int v : G[x])
            if (!dfn[v]) {
                Dom_DAG::G[x].push_back(v);
                Dom_DAG::deg[v]++;
                anc[v] = x;
                dfs(v);
            }
    }
    int fa[maxn], mn[maxn];
    int find(int x) {
        if (x == fa[x]) return x;
        int tmp = fa[x];
        fa[x] = find(fa[x]);
        ckmin(mn[x], mn[tmp]);
        return fa[x];
    }
    int semi[maxn];
    void work() {
        dfs(1);
        for (int i = 1; i <= n; i++) fa[i] = i, mn[i] = 1e9, semi[i] = i;
        for (int w = n; w >= 2; w--) {
            int x = id[w];
```

```
|   |   |   int cur = 1e9;
|   |   |   if (w > cnt) continue;
|   |   |   for (int v : rG[x]) {
|   |   |   |   if (!dfn[v]) continue;
|   |   |   |   if (dfn[v] < dfn[x]) ckmin(cur, dfn[v]);
|   |   |   |   else find(v), ckmin(cur, mn[v]);
|   |   |   }
|   |   |   semi[x] = id[cur];
|   |   |   mn[x] = cur;
|   |   |   fa[x] = anc[x];
|   |   |   Dom_DAG::G[semi[x]].push_back(x);
|   |   |   Dom_DAG::deg[x]++;
|   |   }
|   }
|   void addedge(int x, int y) { G[x].push_back(y), rG[y].push_back(x); }
}   // namespace Dom
```

1.13 图论计数

1.13.1 Prufer 序列

有标号无根树和其 prufer 编码一一对应, 一颗 n 个点的树, 其 prufer 编码长度为 $n-2$, 且度数为 d_i 的点在 prufer 编码中出现 $di-1$ 次.

由树得到序列: 总共需要 $n-2$ 步, 第 i 步在当前的树中寻找具有最小标号的叶子节点, 将与其相连的点的标号设为 Prufer 序列的第 i 个元素 p_i , 并将此叶子节点从树中删除, 直到最后得到一个长度为 $n-2$ 的 Prufer 序列和一个只有两个节点的树.

由序列得到树: 先将所有点的度赋初值为 1, 然后加上它的编号在 Prufer 序列中出现的次数, 得到每个点的度; 执行 $n-2$ 步, 第 i 步选取具有最小标号的度为 1 的点 u 与 $v=p_i$ 相连, 得到树中的一条边, 并将 u 和 v 的度减一. 最后再把剩下的两个度为 1 的点连边, 加入到树中.

推论:

- n 个点完全图, 要求每个点度数依次为 $d_1, d_2, \cdots, d_n$, 这样生成树的棵树为: $\dfrac{(n-2)!}{\prod(d_i-1)!}$
- 左边有 n_1 个点, 右边有 n_2 个点的完全二分图的生成树棵树为 $n_1^{n_2-1}\times n_2^{n_1-1}$
- m 个连通块, 每个连通块有 c_i 个点, 把他们全部连通的生成树方案数: $(\sum c_i)^{m-2}\prod c_i$

1.13.2 无标号树计数

(1) 有根树计数:

$$f_n=\frac{\sum_{i=1}^{n-1} f_{n-i}\sum_{d|i} f_d\cdot d}{n-1}$$

记 $g_i=\sum_{d|i} f_d\cdot d$ 即可做到 $\Theta(n^2)$。

(2) 无根树计数:

当 n 是奇数时

如果根不是重心，必然存在恰好一个子树，它的大小超过 $\left\lfloor\frac{n}{2}\right\rfloor$（设它的大小为 k）减去这种情况即可。

因此答案为

$$f_n-\sum_{k=\lfloor\frac{n}{2}\rfloor+1}^{n-1} f_k\cdot f_{n-k}$$

当 n 是偶数时

有可能存在两个重心，且其中一个是根（即存在一棵子树大小恰为 $\frac{n}{2}$），额外减去 $\binom{f_{\frac{n}{2}}}{2}$ 即可

1.13.3 有标号 DAG 计数

$$F_i=\sum_{j=1}^{i}\binom{i}{j}(-1)^{j+1}2^{j(i-j)}F_{i-j}$$

想法是按照拓扑序分层，每次剥开所有入度为零的点。

1.13.4 有标号连通简单图计数

记 $g(n) = 2^{\binom{n}{2}}$ 为有标号简单图数量，$c(n)$ 为有标号简单连通图数量，那么枚举 1 所在连通块大小，有

$$g(n) = \sum_{i=1}^{n} \binom{n-1}{i-1} c(i) g(n-i)$$

易递推求 $c(n)$。多项式做法考虑 exp 组合意义即可。

1.13.5 生成树计数

Kirchhoff Matrix $T = Deg - A$, Deg 是度数对角阵, A 是邻接矩阵.
无向图度数矩阵是每个点度数; 有向图度数矩阵是每个点入度.
邻接矩阵 $A[u][v]$ 表示 $u \to v$ 边个数, 重边按照边数计算, 自环不计入度数.
无向图生成树计数: $c = |K$ 的任意 1 个 $n-1$ 阶主子式 $|$
有向图外向树计数: $c = |$ 去掉根所在的那阶得到的主子式 $|$
若求边权和则邻接矩阵可以设为 $(1 + wx)$，相当于一次项的系数。

1.13.6 BEST 定理

设 G 是有向欧拉图，k 为任意顶点，那么 G 的不同欧拉回路总数 $\mathrm{ec}(G)$ 是

$$\mathrm{ec}(G) = t^{\mathrm{root}}(k) \prod_{v \in V} (\deg(v) - 1)!.$$

$t^{\mathrm{root}}(k)$ 为以 k 为根的外向树个数。

2 树论

2.1 快速 LCA

查询 $[dfn_u+1, dfn_v]$ 深度最小节点的父亲
可以简化为在 ST 表的最底层记录父亲，比较时取时间戳较小的结点。
取决于 st 表实现可以做到 $O(n)$ or $O(n\log n)$ 预处理 $O(1)$ 查询

```
int getmin(int x, int y) { return dfn[x] < dfn[y] ? x : y; }
void dfs(int u, int f) {
    dfn[u] = ++tim;
    a[dfn[u]] = f;  // TODO: build ST for a[i]
    for (int v : G[u])
        if (v != f) dfs(v, u);
}
int lca(int u, int v) {
    if (u == v) return u;
    if ((u = dfn[u]) > (v = dfn[v])) swap(u, v);
    return RMQ(dfn[u] + 1, dfn[v]);
}
```

2.2 虚树

```
vector<int> Gn[maxn];
int st[maxn], top;
void build(vector<int> v) {
    sort(v.beign(), v.end(),
          [&](const int& a, const int& b) { return dfn[a] < dfn[b]; });
    top = 0;
    if (v[0] != 1) st[++top] = 1;  // Assume 1 is the root
    for (int u : v) {
        if (!top) {
            st[++top] = u;
            continue;
        }
        int anc = lca(st[top], u);
        if (anc == st[top]) {
            st[++top] = u;
            continue;
        }
        while (top > 1 && dfn[lca] <= dfn[st[top - 1]]) {
            Gn[st[top - 1]].pb(st[top]), top--;
        }
        if (anc != st[top]) Gn[anc].pb(st[top]), st[top] = anc;
        st[++top] = u;
    }
    while (top) Gn[st[top - 1]].pb(st[top]), top--;
}
// use DFS to clear Gn
```

2.3 长链剖分

2.3.1 优化 dp

优化以深度为下标的树形 DP
例如 $dp(u,i)$ 表示 u 子树到达 u 距离为 i 的顶点信息，则考虑对于树进行长链剖分，dfn_u 表示 u 在长链剖分的 dfn 序。
则可以将 $dp(u,i)$ 记为 $dp(dfn_u+i)$，就可以做到长链直接继承。

2.3.2 k 级祖先

待补充

2.4 静态点分治

```
void get_root(int u, int f) {
    sz[u] = 1, wt[u] = 0;
    for (int v : G[u]) {
        if (v == f || vis[v]) continue;
        get_root(v, u), sz[u] += sz[v], ckmax(wt[u], sz[v]);
    }
    ckmax(wt[u], Tsize - sz[u]);
    if (wt[Rt] > wt[u]) Rt = u;
}
void solve(int u) {
    vis[u] = 1;
    for (int v : G[u]) {
        if (vis[v]) continue;
        Rt = 0, Tsize = sz[v], get_root(v, 0);
        solve(Rt);
    }
}
wt[Rt = 0] = INF, Tsize = n;
get_root(1, 0);
solve(Rt);
```

2.5 点分树

待验证，以下为邻域点权和模版（震波）

```
void build(int u) {
    vis[u] = 1;
    t2[u].add(0, a[u]);
    for (int v : G[u]) {
        if (vis[v]) continue;
        Rt = 0, mxdep = 0, Tsize = sz[v];
        get_root(v, 0, 1);
        fa[Rt] = u;
        t1[Rt].init(mxdep + 5);
        t2[Rt].init(mxdep + 5);
        get_dis(v, u, 1);
        build(Rt);
    }
}
void modify(int u, int val) {
    for (int i = u; i; i = fa[i]) {
        t2[i].add(dis(u, i), val - a[u]);
        if (fa[i]) t1[i].add(dis(u, fa[i]), val - a[u]);
    }
    a[u] = val;
}
int query(int u, int k) {
    int rt = 0;
    for (int i = u; i; i = fa[i]) {
        rt += t2[i].query(k - dis(u, i));
        if (fa[i]) rt -= t1[i].query(k - dis(u, fa[i]));
    }
    return rt;
}
```

2.6 动态 dp

```
void dfs1(int u) {
    siz[u] = 1;
    dep[u] = dep[fa[u]] + 1;
    for (int v : G[u]) {
        dfs1(v);
        siz[u] += siz[v];
```

```
|   |   if (siz[v] > siz[son[u]]) son[u] = v;
|   }
}
int endc[maxn];
Vector dp[maxn];  // F[u] 为 u 的 dp 值
Matrix trans[maxn];
// 考虑u点所有轻儿子以及u点点权的贡献转移矩阵，则某点u的dp值为 trans[u]*dp[son[u]]
void dfs2(int u, int t) {
|   dfn[u] = ++tim, id[tim] = u;
|   top[u] = t, endc[t] = max(endc[t], tim);
|   // TODO: 初始化 F[u] 和 trans[u]
|   if (son[u]) dfs2(son[u], t);
|   for (int v : G[u]) {
|   |   if (v == son[u]) continue;
|   |   dfs2(v, v);
|   |   // TODO: 用 dp[v] 更新 trans[u]
|   }
|   dp[u] = trans[u] * dp[son[u]];
}

struct Segtree {
|   Matrix t[maxn << 2];
|   void build(int u, int l, int r);  // t[u] = trans[id[x]];
|   void pushup(int u);
|   void update(int u, int l, int r, int x);  // t[u] = trans[id[x]]
|   Matrix query(int u, int l, int r, int L, int R);
} T;

void update(int u) {
|   // TODO: 更新 trans[u] 和 dp[u]
|   Matrix aft;
|   while (u != 0) {
|   |   T.update(1, 1, n, dfn[u]);
|   |   aft = T.query(1, 1, n, dfn[top[u]], endc[top[u]]);
|   |   int v = top[u];
|   |   u = fa[v];
|   |   if (u) {}  // TODO: 用 aft 更新 trans[u] 和 dp[u]
|   }
}
Vector query() { return T.query(1, 1, n, id[1], endc[1]) * dp[id[endc[1]]]; }
```

2.7 树上背包

```
// 背包大小上界为 m，复杂度为 O(nm)
void solve(int u) {
|   sz[u] = 1;
|   for (int v : G[u]) {
|   |   solve(v);
|   |   for (int i = 0; i <= m; i++) tmp[i] = 0;
|   |   for (int i = 0; i <= min(m, sz[u]); i++)
|   |   |   for (int j = 0; j <= min(m - i, sz[v]); j++)
|   |   |   |   update(tmp[i + j], dp[u][i], dp[v][j]);
|   |   sz[u] += sz[v];  // DON'T MOVE THIS!!!
|   |   for (int i = 0; i <= m; i++) dp[u][i] = tmp[i];
|   }
}
```

2.8 树哈希

```
mt19937_64 rnd(time(nullptr));
const ull mask = rnd();
const ull base = rnd();
ull xorshift(ull x) {
|   x ^= mask;
|   x ^= x << 13;
```

```
    x ^= x >> 7;
    x ^= x << 17;
    x ^= mask;
    return x;
}
ull hsh[maxn], sz[maxn];
void dfs(int u, int f) {
    hsh[u] = base, sz[u] = 1;
    for (int v : G[u]) {
        if (v == f) continue;
        dfs(v, u);
        hsh[u] += xorshift(hsh[v]);
        sz[u] += sz[v];
    }
    hsh[u] += xorshift(sz[u]);
}
```

3 数论

3.1 数论分块

每一次 $[l,r]$ 都是 $n/l=n/r, m/l=m/r$ 的极大区间。
多个 n,m 只要对多个 $n/(n/l)$ 取 min 即可，复杂度为 $O(|cnt|\sqrt{V})$

```
for (ll l = 1, r = 1; l <= min(n, m); l = r + 1) {
    r = min(n / (n / l), m / (m / l));
    // Do something here
}
```

3.2 积性函数线性筛

欧拉函数和莫比乌斯函数可以更简单的线性筛，见注释

```
bool vis[maxn];
int prime[maxn], totp, mnpe[maxn], f[maxn];
void init() {
    vis[1] = 1;
    mnpe[1] = 1;  // mu[1] = ph[1] = 1
    for (int i = 2; i <= N; i++) {
        if (!vis[i])
            prime[++totp] = i, mnpe[i] = i;  // mu[i] = -1, phi[i] = i - 1;
        for (int j = 1; j <= totp && i * prime[j] <= N; j++) {
            vis[i * prime[j]] = 1;
            if (i % prime[j] == 0) {
                mnpe[i * prime[j]] = mnpe[i] * prime[j];
                // mu[i * prime[j]] = 0;
                // phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            mnpe[i * prime[j]] = prime[j];
            // mu[i * prime[j]] = -mu[i];
            // phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
    }
    for (int i = 1; i <= totp; i++)
        for (int e = 1, p = prime[i]; p <= N; e++, p *= prime[i]) {
            // TODO: 在这里计算素数幂处的值 f[p]
        }
    for (int i = 1; i <= N; i++)
        if (i != mnpe[i]) f[i] = f[mnpe[i]] * f[i / mnpe[i]];
}
```

3.3 筛子

3.3.1 杜教筛

若想要求出 f 在 n 处的前缀和 $s(n)=\sum_{i=1}^{n} f(i)$，构造积性函数 g，设 $h=f*g$，则

$$\begin{aligned}&\sum_{i=1}^{n} h(i)\\=&\sum_{ij\le n} f(i)g(j)\\=&\sum_{d=1}^{n} g(d)\sum_{i=1}^{\frac{n}{d}} f(i)\\=&\sum_{d=1}^{n} g(d)s\left(\left\lfloor\frac{n}{d}\right\rfloor\right)\end{aligned}$$

若 g,h 的前缀和可以快速求出，则

$$s(n)=\sum_{i=1}^{n}h(i)-\sum_{d=2}^{n}g(d)s\left(\left\lfloor\frac{n}{d}\right\rfloor\right)$$

预处理 f 的前缀和到 $n^{2/3}$ 处即可做到单次查询 $O(n^{2/3})$。

```
ll sum_f(ll x) {
    if (x <= N) return sf[x];
    if (Sf[n / x]) return Sf[n / x];
    ll ans = Sum_h(x);
    for (ll l = 2, r; l <= x; l = r + 1) {
        r = x / (x / l);
        sub(ans, sum_g(l, r) * sum_f(x / l));
    }
    return Sf[n / x] = ans;
}
```

3.3.2 min-25 筛（质数个数）

```
const int N = 1e6;
double inv[maxn];
void sieve() {
    // 见线性筛部分
    for (int i = 1; i <= N; ++i) inv[i] = 1.0 / i;
}
ll val[maxn], pos[maxn], id1[maxn], id2[maxn], cnt;
ll solve(ll n) {
    ll T = sqrt(n) + 3;
    for (ll i = 1, pi; i <= n; i = pi + 1) {
        pos[++cnt] = n / i;
        pi = n / pos[cnt];
        val[cnt] = pos[cnt] - 1;
        (n / i <= T ? id1[n / i] : id2[pi]) = cnt;
    }
    auto getid = [&](ll x) -> int { return x <= T ? id1[x] : id2[n / x]; };
    for (int i = 1; 1ll * prime[i] * prime[i] <= n; i++) {
        int p = id1[prime[i - 1]];
        for (int j = 1; pos[j] >= 1ll * prime[i] * prime[i]; j++) {
            int q = getid(1.0 * pos[j] * inv[prime[i]] + 1e-7);
            val[j] -= val[q] - val[p];
        }
    }
    return val[1];
}
```

3.3.3 min-25 筛

未验证，待补充

```
void init_g() {
    for (ll l = 1, r; l <= n; l = r + 1) {
        r = n / (n / l);
        w[++tot] = n / l;
        g ^ k[tot] = sum(1..w [tot] ^ k);
        if (n / l <= N) ind1[n / l] = tot;
        else ind2[l] = tot;
    }
}
void calc_g() {
    for (int i = 1; i <= totp; i++) {
        for (int j = 1; j <= tot && prime[i] * prime[i] <= w[j]; j++) {
            ll k = w[j] / prime[i] <= N ? ind1[w[j] / prime[i]]
                                        : ind2[n / (w[j] / prime[i])];
            sub(g ^ e[j], prime[i] ^ e * (g ^ e[k] - sp ^ e[i - 1]));
        }
    }
```

```cpp
}
ll S(ll x, int y) {
    cnt++;
    if (prime[y] >= x) return 0;
    ll k = x <= N ? ind1[x] : ind2[n / x];
    ll ans = f(g ^ e(x)) - f(sp ^ e[y]);
    for (int i = y + 1; i <= totp && prime[i] * prime[i] <= x; i++) {
        ll p = prime[i];
        for (int e = 1; p <= x; e++, p *= prime[i]) {
            add(ans, f(p) * (S(x / p, i) + (e != 1)));
        }
    }
    return ans % MOD;
}
```

3.3.4 PowerfulNumber 筛

未验证，待补充。

求 $\sum_{i=1}^{n} f(i)$，找到 $g(i)$ 满足 $\forall p \in \mathbf{P}, f(p) = g(p)$，g 可杜教筛。

不妨假设 $g * h = f$，那么 h 仅在 PN 处有值，于是

$$\begin{aligned}\sum_{i=1}^{n} f(i) &= \sum_{ij} g(i)h(j) \\ &= \sum_{i \le n} h(i) \sum_{j=1}^{n/i} g(j)\end{aligned}$$

计算 h 依靠递推：

$$f(p^k) = \sum_{i+j=k} g(p^i)h(p^j)$$

$$h(p^k) = f(p^k) - \sum_{i=0}^{k-1} h(p^i)g(p^{k-i})$$

```cpp

void power_num(int k, ll m, int h) {
    if (k > totp || m * prime[k] > n) {
        if (n / m <= N) add(ans, 1ll * h * sxphi[n / m] % MOD);
        else add(ans, 1ll * h * Sxphi[n / (n / m)] % MOD);
        return;
    }
    power_num(k + 1, m, h);
    ll p = 1ll * prime[k] * prime[k];
    for (int e = 2; m * p <= n; p *= prime[k], e++) {
        power_num(k + 1, m * p,
                  p % MOD * (prime[k] - 1) % MOD * (e - 1) % MOD * h % MOD);
    }
}
```

3.3.5 洲阁筛

我不会，长大后再学习。

3.4 扩展欧几里得

```cpp
ll exgcd(ll a, ll b, ll& x, ll& y) {
    if (!b) return x = 1, y = 0, a;
    else {
        ll rt = exgcd(b, a % b, y, x);
        y -= (a / b) * x;
        return rt;
    }
}
```

3.5 欧拉定理

当 $(a,m)=1$ 时,

$$a^{\varphi(m)} \equiv 1 (\bmod\ m)$$

。

当 $(a,m)\neq 1$ 时,

$$a^b \equiv a^{\min\{b, b \bmod \ \varphi(m)+\varphi(m)\}} (\bmod\ m)$$

。

3.6 中国剩余定理

解方程:

$$\begin{cases} x \equiv a_1 (\bmod\ m_1) \\ x \equiv a_2 (\bmod\ m_2) \\ \vdots \\ x \equiv a_n (\bmod\ m_n) \end{cases}$$

若 m_i 两两互质,则可以使用以下公式得到:

$$x \equiv \sum_{i=1}^{n} M_i \times N_i \times a_i (\bmod\ M)$$

$$\text{where: } \begin{cases} M = \prod_{i=1}^{n} m_i \\ M_i = \frac{M}{m_i} \\ N_i \times M_i \equiv 1 (\bmod\ m_i) \end{cases}$$

否则参考以下 exCRT。

```cpp
ll exCRT(vector<int> a, vector<int> m) {
    assert(a.size() == m.size());
    ll ans = a[0], M = b[0];
    for (int i = 1; i < a.size(); i++) {
        ll x = 0, y = 0;
        ll A = M, B = m[i], C = (a[i] - ans % B + B) % B;
        ll gcd = exgcd(A, B, x, y), bg = B / gcd;
        x = x * (C / gcd) % B;
        ans += x * M;
        M *= bg;
        ans = (ans % M + M) % M;
    }
    return (ans % M + M) % M;
}
```

3.7 BSGS

```cpp
gp_hash_table<ll, ll> s;
ll exgcd(ll a, ll b) {
    if (a == 1) return 1;
    return (1 - b * exgcd(b % a, a)) / a;  // not ll
}
ll exBSGS(ll a, ll b, ll p) {
    s.clear();
    a %= p, b %= p;
    ll j = 1 % p, cnt = 0;
    for (int i = 0; i <= __lg(p); i++, j = j * a % p)
        if (j == b) return i;
    ll x, y = 1;
    while (true) {
        x = gcd(a, p);
        if (x == 1) break;
        if (b % x) return -1;  // no sol
```

```
        cnt++;
        p /= x, b /= x;
        y = y * (a / x) % p;
    }
    a %= p;
    b = (ll)b * (p + exgcd(y, p)) % p;
    x = ceil(sqrt(p)), j = 1;
    for (int i = 0; i < x; i++, j = j * a % p) {
        if (j == b) return i + cnt;
        s[j * b % p] = i + 1;
    }
    int k = j;
    for (int i = 1; i <= x; i++, j = (ll)j * k % p)
        if (s[j]) return (ll)i * x + cnt - s[j] + 1;
    return -1;
}
```

3.8 Millar-Robin

Pollard-Rho 的 2 ∼ 30 行即为 Millar-Robin

3.9 Pollard-Rho

```
1  namespace factor {
2      using f64 = long double;
3      ll p;
4      f64 invp;
5      inline void setmod(ll x) { p = x, invp = (f64)1 / x; }
6      inline ll mul(ll a, ll b) {
7          ll z = a * invp * b + 0.5;
8          ll res = a * b - z * p;
9          return res + (res >> 63 & p);
10     }
11     inline ll pow(ll a, ll x, ll res = 1) {
12         for (; x; x >>= 1, a = mul(a, a))
13             if (x & 1) res = mul(res, a);
14         return res;
15     }
16     inline bool checkprime(ll p) {
17         if (p == 1) return 0;
18         setmod(p);
19         ll d = __builtin_ctzll(p - 1), s = (p - 1) >> d;
20         for (ll a : {2, 3, 5, 7, 11, 13, 82, 373}) {
21             if (a % p == 0) continue;
22             ll x = pow(a, s), y;
23             for (int i = 0; i < d; ++i, x = y) {
24                 y = mul(x, x);
25                 if (y == 1 && x != 1 && x != p - 1) return 0;
26             }
27             if (x != 1) return 0;
28         }
29         return 1;
30     }
31     inline ll rho(ll n) {
32         if (!(n & 1)) return 2;
33         static std::mt19937_64 gen((size_t) "hehezhou");
34         ll c = gen() % (n - 1) + 1, y = gen() % (n - 1) + 1;
35         auto f = [&](ll o) {
36             o = mul(o, o) + c;
37             return o >= n ? o - n : o;
38         };
39         setmod(p);
40         for (int l = 1;; l <<= 1) {
41             ll x = y, g = 1;
42             for (int i = 0; i < l; ++i) y = f(y);
43             const int d = 512;
```

```
            for (int i = 0; i < l; i += d) {
                ll sy = y;
                for (int j = 0; j < min(d, l - i); ++j) {
                    y = f(y), g = mul(g, (y - x + n));
                }
                g = gcd(n, g);
                if (g == 1) continue;
                if (g == n)
                    for (g = 1, y = sy; g == 1;)
                        y = f(y), g = gcd(n, y - x + n);
                return g;
            }
        }
    }
    inline std::vector<ll> factor(ll x) {
        std::queue<ll> q;
        q.push(x);
        std::vector<ll> res;
        for (; q.size();) {
            ll x = q.front();
            q.pop();
            if (x == 1) continue;
            if (checkprime(x)) {
                res.push_back(x);
                continue;
            }
            ll y = rho(x);
            q.push(y), q.push(x / y);
        }
        sort(res.begin(), res.end());
        return res;
    }
} // namespace factor
```

3.10 原根

你说的对，但是感觉不如原根。

原根，是一个数学符号。设 m 是正整数，a 是整数，若 a 模 m 的阶等于 $\varphi(m)$ （定义 a 模 m 的阶 $\delta_m(a)$ 为最小的 x 满足 $a^x \equiv 1 \pmod m$），则称 a 为模 m 的一个原根。

假设一个数 g 是 $p \in \mathbf{P}$ 的原根，那么 $\forall 0 < i < p, g^i \bmod p$ 的结果两两不同，归根到底就是 $g^a \equiv 1 \pmod p$ 当且仅当指数 a 为 $p-1$ 的倍数时成立。

你的数学很差，我现在每天用原根都能做 10^5 次数据规模 10^6 的 NTT，每个月差不多 3×10^6 次卷积，即 2×10^6 次常系数齐次线性递推，也就是现实生活中 6.4×10^{19} 次乘法运算，换算过来最少也要算 2×10^4 年。虽然我只有 14 岁，但是已经超越了中国绝大多数人（包括你）的水平，这便是原根给我的骄傲的资本。

性质：

- 最小原根大小数量级在 $O(m^{1/4})$ 左右，求最小原根直接枚举 g 并检验对于 $\varphi(m)$ 的每个素因数 p，都有 $g^{\frac{\varphi(m)}{p}} \not\equiv 1 \pmod m$ 即可。
- $$\delta_m(a^k) = \frac{\delta_m(a)}{(\delta_m(a), k)}$$
  通常这里的 a 是 g。
- 将模 m 剩余系看成一个 $\times g$ 的循环群。

3.11 二次剩余魔法

待补充

3.12 类欧几里得和万能欧几里得

未验证，待补充板子为直线下点数：

$$ans = \sum_{i=1}^{n-1} \lfloor \frac{ai+b}{m} \rfloor$$

```cpp
ull floor_sum(ull n, ull m, ull a, ull b) {
    ull ans = 0;
    while (true) {
        if (a >= m) ans += n * (n - 1) / 2 * (a / m), a %= m;
        if (b >= m) ans += n * (b / m), b %= m;
        ull ymax = a * n + b;  // use u128 if it's big
        if (ymax < m) break;
        n = ymax / m;
        b = ymax % m;
        std::swap(m, a);
    }
    return ans;
}
```

3.13 Lucas 和扩展 Lucas

$$\binom{n}{m} \equiv \binom{n_1}{m_1}\binom{n_2}{m_2}\binom{n_3}{m_3}\cdots \pmod p$$

推论：$\binom{n}{m} \not\equiv 0 \pmod p$ 当且仅当 m 和 $n-m$ 在 p 进制相加没有进位。例如 $p=2$ 时只有 $n \wedge m = m$ 时 $\binom{n}{m} \equiv 1(mod\ 2)$。

扩展 Lucas 待补充。

4 数学

4.1 矩阵

4.2 多项式合集

务必记得 Init(n) 在进行多项式乘法前!!!

```
const int MOD = 998244353;
const int N = 20;
const int G = 3;
const int MAXN = 3e5 + 5;
typedef vector<int> poly;
void print(const poly& a) {
    for (int i = 0; i < (int)a.size(); i++)
        cout << (i == 0 ? " " : "+ ") << a[i] << "*x^" << i << ' ';
    cout << endl;
}
namespace Poly {
    const int mod = 998244353;
    int rev[MAXN], w[MAXN], wn[N];
    void addmod(int& x, int y) {
        x += y;
        if (x >= mod) x -= mod;
    }
    void submod(int& x, int y) {
        x -= y;
        if (x < 0) x += mod;
    }
    int add(int x, int y) {
        addmod(x, y);
        return x;
    }
    int sub(int x, int y) {
        submod(x, y);
        return x;
    }
    int power(int x, int y) {
        int res = 1;
        while (y) {
            if (y & 1) res = (ll)res * x % mod;
            x = (ll)x * x % mod;
            y >>= 1;
        }
        return res;
    }
    int Inv(int x) { return power(x, mod - 2); }
    void InitNTT(int n) {
        wn[n] = power(G, (mod - 1) / (1 << n));
        for (int i = n - 1; i >= 0; i--)
            wn[i] = (ll)wn[i + 1] * wn[i + 1] % mod;
    }
    int Init(int n) {
        int len = 1;
        while (len < n) len <<= 1;
        for (int i = 0; i < len; i++)
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) * (len >> 1));
        for (int i = 1, t = 1; i < len; i <<= 1, t += 1) {
            w[i] = 1;
            for (int j = 1; j < i; j++)
                w[i + j] = (ll)w[i + j - 1] * wn[t] % mod;
        }
        return len;
    }
    void NTT(poly& a, int flag) {
        int n = a.size();
        for (int i = 0; i < n; i++)
            if (i < rev[i]) swap(a[i], a[rev[i]]);
        for (int i = 2; i <= n; i <<= 1) {
```

```
            int mid = (i >> 1);
            for (int j = 0; j < n; j += i) {
                for (int k = j; k < j + mid; k++) {
                    int x = a[k], y = (ll)a[k + mid] * w[k - j + mid] % mod;
                    a[k] = add(x, y);
                    a[k + mid] = sub(x, y);
                }
            }
        }
        if (flag == -1) {
            reverse(a.begin() + 1, a.begin() + n);
            int invn = Inv(n);
            for (int i = 0; i < n; i++) a[i] = (ll)a[i] * invn % mod;
        }
    }
    poly PolyAdd(const poly& A, const poly& B) {
        poly res = A;
        for (int i = 0; i < (int)A.size(); i++) addmod(res[i], B[i]);
        return res;
    }
    poly PolyMul(const poly& A, const poly& B, int need = 0) {
        int n = A.size(), m = B.size();
        if (n < 5 || m < 5) {
            poly a;
            a.resize(n + m - 1);
            for (int i = 0; i < n; i++)
                for (int j = 0; j < m; j++)
                    addmod(a[i + j], (ll)A[i] * B[j] % mod);
            if (need) a.resize(need);
            return a;
        }
        int len = Init(n + m);
        poly a = A, b = B;
        a.resize(len), b.resize(len);
        NTT(a, 1);
        NTT(b, 1);
        for (int i = 0; i < len; i++) a[i] = (ll)a[i] * b[i] % mod;
        NTT(a, -1);
        a.resize(need ? need : n + m - 1);
        return a;
    }
    poly PolyInv(const poly& A) {
        int n = A.size();
        if (n == 1) { return {Inv(A[0])}; }
        poly a = A, b = PolyInv(poly(A.begin(), A.begin() + ((n + 1) >> 1)));
        int len = Init(n << 1);
        a.resize(len), b.resize(len);
        NTT(a, 1);
        NTT(b, 1);
        for (int i = 0; i < len; i++)
            b[i] = (ll)sub(2, (ll)a[i] * b[i] % mod) * b[i] % mod;
        NTT(b, -1);
        b.resize(n);
        return b;
    }
    poly PolyDeriv(const poly& A) {
        int n = A.size();
        poly a = A;
        for (int i = 1; i < n; i++) a[i - 1] = (ll)i * A[i] % mod;
        a[n - 1] = 0;
        return a;
    }
    poly PolyInter(const poly& A) {
        int n = A.size();
        poly a = A;
        for (int i = 1; i < n; i++)
            a[i] = (ll)A[i - 1] * power(i, mod - 2) % mod;
        a[0] = 0;
```

```cpp
        return a;
    }
    pair<poly, poly> PolyMod(const poly& A, const poly& B) {
        int n = A.size(), m = B.size();
        if (n < m) return make_pair(poly(1), A);
        poly a = A, b = B;
        reverse(a.begin(), a.end());
        reverse(b.begin(), b.end());
        b.resize(n - m + 1);
        b = PolyInv(b);
        a.resize(n - m + 1);
        a = PolyMul(a, b, n - m + 1);
        reverse(a.begin(), a.end());
        b = PolyMul(a, B, m - 1);
        for (int i = 0; i < m - 1; i++) b[i] = sub(A[i], b[i]);
        return make_pair(a, b);
    }
    poly PolyLn(const poly& A) {
        int n = A.size();
        poly a = A;
        for (int i = 1; i < n; i++) a[i - 1] = (ll)i * A[i] % mod;
        a[n - 1] = 0;
        a = PolyMul(a, PolyInv(A), n);
        for (int i = n - 1; i >= 1; i--)
            a[i] = (ll)a[i - 1] * power(i, mod - 2) % mod;
        a[0] = 0;
        return a;
    }
    poly PolyExp(const poly& A) {
        int n = A.size();
        if (n == 1) return {1};
        poly b = PolyExp(poly(A.begin(), A.begin() + ((n + 1) >> 1)));
        b.resize(n);
        poly c = PolyLn(b);
        for (int i = 0; i < n; i++) c[i] = sub(A[i], c[i]);
        addmod(c[0], 1);
        poly d = PolyMul(b, c, n);
        return d;
    }
    poly PolyPow(const poly& A, int k) {
        int n = A.size();
        poly a;
        a.resize(n);
        if (!k) {
            a[0] = 1;
            return a;
        }
        int p = 0;
        while (p < n && !A[p]) p += 1;
        if ((ll)p * k >= n) return a;
        int m = n - p * k;
        a.resize(m);
        int coef = power(A[p], k), icoef = power(A[p], mod - 2);
        for (int i = 0; i < m; i++) a[i] = (ll)A[i + p] * icoef % mod;
        a = PolyLn(a);
        for (int i = 0; i < m; i++) a[i] = (ll)a[i] * k % mod;
        a = PolyExp(a);
        poly b;
        b.resize(n);
        for (int i = 0; i < m; i++) b[i + p * k] = (ll)a[i] * coef % mod;
        return b;
    }
    poly tmp[MAXN];
#define lson k << 1
#define rson k << 1 | 1
    void pre_eval(const poly& A, int k, int l, int r) {
        if (l == r) {
            tmp[k].resize(2);
```

```
            tmp[k][0] = sub(0, A[l]);
            tmp[k][1] = 1;
            return;
        }
        int mid = (l + r) >> 1;
        pre_eval(A, lson, l, mid);
        pre_eval(A, rson, mid + 1, r);
        tmp[k] = PolyMul(tmp[lson], tmp[rson]);
    }
    void solve_eval(const poly& A, const poly& B, poly& C, int k, int l,
                    int r) {
        if (r - l <= 30) {
            for (int i = l; i <= r; i++)
                for (int j = A.size() - 1; j >= 0; j--)
                    C[i] = (A[j] + (ll)C[i] * B[i] % mod) % mod;
            return;
        }
        int mid = (l + r) >> 1;
        solve_eval(PolyMod(A, tmp[lson]).second, B, C, lson, l, mid);
        solve_eval(PolyMod(A, tmp[rson]).second, B, C, rson, mid + 1, r);
    }
    poly PolyEval(const poly& A, const poly& B) {
        int m = B.size();
        pre_eval(B, 1, 0, m - 1);
        poly c;
        c.resize(m);
        solve_eval(PolyMod(A, tmp[1]).second, B, c, 1, 0, m - 1);
        return c;
    }
    poly solve_itpl(const poly& A, int k, int l, int r) {
        if (l == r) return {A[l]};
        int mid = (l + r) >> 1;
        return PolyAdd(PolyMul(solve_itpl(A, lson, l, mid), tmp[rson]),
                       PolyMul(solve_itpl(A, rs on, mid + 1, r), tmp[lson]));
    }
    poly PolyItpl(const poly& A, const poly& B) {
        int n = A.size();
        pre_eval(A, 1, 0, n - 1);
        poly a;
        a.resize(n);
        solve_eval(PolyDeriv(tmp[1]), A, a, 1, 0, n - 1);
        for (int i = 0; i < n; i++) a[i] = (ll)B[i] * Inv(a[i]) % MOD;
        return solve_itpl(a, 1, 0, n - 1);
    }
#undef lson
#undef rson
    struct Initializer {
        Initializer() { InitNTT(N - 1); }
    } initializer;
}  // namespace Poly
```

4.3 BM

$$\forall i, \sum_{j=0}^{m} a_{i-j} v_j = 0$$

```
vector<int> berlekamp_massey(const vector<int>& a) {
    vector<int> v, last;  // v is the answer, 0-based
    int k = -1, delta = 0;
    for (int i = 0; i < (int)a.size(); i++) {
        int tmp = 0;
        for (int j = 0; j < (int)v.size(); j++)
            tmp = (tmp + (long long)a[i - j - 1] * v[j]) % p;
        if (a[i] == tmp) continue;
        if (k < 0) {
            k = i;
```

```cpp
            delta = (a[i] - tmp + p) % p;
            v = vector<int>(i + 1);
            continue;
        }
        vector<int> u = v;
        int val = (long long)(a[i] - tmp + p) * qpow(delta, p - 2) % p;
        if (v.size() < last.size() + i - k) v.resize(last.size() + i - k);
        (v[i - k - 1] += val) %= p;
        for (int j = 0; j < (int)last.size(); j++) {
            v[i - k + j] = (v[i - k + j] - (long long)val * last[j]) % p;
            if (v[i - k + j] < 0) v[i - k + j] += p;
        }
        if ((int)u.size() - i < (int)last.size() - k) {
            last = u;
            k = i;
            delta = a[i] - tmp;
            if (delta < 0) delta += p;
        }
    }
    for (auto& x : v) x = (p - x) % p;
    v.insert(v.begin(), 1);  //一般是需要最小递推式的，处理一下
    return v;
}
```

4.4 线性规划单纯形法

```cpp
using db = long double;
const db eps = 1e-16;
int sgn(db x) { return x < -eps ? -1 : x > eps; }
namespace LP {
    const int N = 21, M = 21;
    int n, m;  // n : 变量个数,m : 约束个数
    db a[M + N][N], x[N + M];
    // 约束:对于 1 <= i <= m : a[i][0] + \sum_j x[j] * a[i][j] >= 0
    // x[j] >= 0
    // 最大化 \sum_j x[j] * a[0][j]
    int id[N + M];
    void pivot(int p, int o) {
        std::swap(id[p], id[o + n]);
        db w = -a[o][p];
        for (int i = 0; i <= n; ++i) a[o][i] /= w;
        a[o][p] = -1 / w;
        for (int i = 0; i <= m; ++i)
            if (sgn(a[i][p]) && i != o) {
                db w = a[i][p];
                a[i][p] = 0;
                for (int j = 0; j <= n; ++j) a[i][j] += w * a[o][j];
            }
    }
    db solve() {  // nan : 无解,inf : 无界,否则返回最大值
        for (int i = 1; i <= n + m; ++i) id[i] = i;
        for (;;) {
            int p = 0, min = 1;
            for (int i = 1; i <= m; ++i) {
                if (a[i][0] < a[min][0]) min = i;
            }
            if (a[min][0] >= -eps) break;
            for (int i = 1; i <= n; ++i)
                if (a[min][i] > eps && id[i] > id[p]) { p = i; }
            if (!p) return nan("");
            pivot(p, min);
        }
        for (;;) {
            int p = 1;
            for (int i = 1; i <= n; ++i)
                if (a[0][i] > a[0][p]) p = i;
            if (a[0][p] < eps) break;
```

```cpp
            db min = INFINITY;
            int o = 0;
            for (int i = 1; i <= m; ++i)
                if (a[i][p] < -eps) {
                    db w = -a[i][0] / a[i][p];
                    int d = sgn(w - min);
                    if (d < 0 || !d && id[i] > id[o]) o = i, min = w;
                }
            if (!o) return INFINITY;
            pivot(p, o);
        }
        for (int i = 1; i <= m; ++i) x[id[i + n]] = a[i][0];
        return a[0][0];
    }
} // namespace LP
```

4.5 FWT

$$C_i = \sum_{j \oplus k = i} A_j \times B_k$$

。A, B FWT 后对应位相乘在 iFWT 回去。

```cpp
// op = 1 / -1
inline void FMT_OR(int* a, int n, int op) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < (1 << n); j++)
            if ((1 << i) & j)
                add(a[j], op == 1 ? a[j ^ (1 << i)] : MOD - a[j ^ (1 << i)]);
}
inline void FMT_AND(int* a, int n, int op) {
    for (int i = 0; i < n; i++)
        for (int j = (1 << n) - 1; j >= 0; j--)
            if (!((1 << i) & j))
                add(a[j], op == 1 ? a[j ^ (1 << i)] : MOD - a[j ^ (1 << i)]);
}
const int inv2 = 499122177;
inline void FWT_XOR(int* a, int n, int op) {
    for (int i = 1; i < (1 << n); i <<= 1)
        for (int j = 0; j < (1 << n); j += (i << 1))
            for (int k = 0; k < i; k++) {
                int t = a[j + k];
                a[j + k] = mod1(t + a[i + j + k]);
                a[i + j + k] = mod1(t + MOD - a[i + j + k]);
                if (~op) {
                    a[j + k] = 1ll * a[j + k] * inv2 % MOD;
                    a[i + j + k] = 1ll * a[i + j + k] * inv2 % MOD;
                }
            }
}
```

4.6 常见数和公式

待补充。

- 一般幂转下降幂：

$$x^n = \sum_{i=0}^{n} \left\{ {n \atop i} \right\} x^{\underline{i}}, n \geq 0$$

5 字符串

Delete This

6 数据结构

Delete This

7 计算几何

7.1 计算几何

by yhx-12243，暂未验证

```
const double eps = 1e-8;

#define lt(x, y) ((x) < (y)-eps)
#define gt(x, y) ((x) > (y) + eps)
#define le(x, y) ((x) <= (y) + eps)
#define ge(x, y) ((x) >= (y)-eps)
#define eq(x, y) (le(x, y) && ge(x, y))
#define dot(x, y, z) (((y) - (x)) * ((z) - (x)))
#define cross(x, y, z) (((y) - (x)) ^ ((z) - (x)))
struct vec2 {
    double x, y;
    vec2(double x0 = 0.0, double y0 = 0.0) : x(x0), y(y0) {}
    vec2* read() {
        scanf("%lf%lf", &x, &y);
        return this;
    }
    inline vec2 operator-() const { return vec2(-x, -y); }
    inline vec2 operator+(const vec2& B) const {
        return vec2(x + B.x, y + B.y);
    }
    inline vec2 operator-(const vec2& B) const {
        return vec2(x - B.x, y - B.y);
    }
    inline vec2 operator*(double k) const { return vec2(x * k, y * k); }
    inline vec2 operator/(double k) const { return *this * (1.0 / k); }
    inline double operator*(const vec2& B) const { return x * B.x + y * B.y; }
    inline double operator^(const vec2& B) const { return x * B.y - y * B.x; }
    inline double norm2() const { return x * x + y * y; }
    inline double norm() const { return sqrt(x * x + y * y); }
    inline bool operator<(const vec2& B) const {
        return lt(x, B.x) || le(x, B.x) && lt(y, B.y);
    }
    inline bool operator==(const vec2& B) const {
        return eq(x, B.x) && eq(y, B.y);
    }
    inline bool operator<<(const vec2& B) const {
        return lt(y, 0) ^ lt(B.y, 0)
                   ? lt(B.y, 0)
                   : gt(*this ^ B, 0)
                         || ge(*this ^ B, 0) && ge(x, 0) && lt(B.x, 0);
    }
    inline vec2 trans(double a11, double a12, double a21, double a22) const {
        return vec2(x * a11 + y * a12, x * a21 + y * a22);
    }
};
/*
operator * : Dot product
operator ^ : Cross product
norm2() : |v|^2 = v.v
norm() : |v| = sqrt(v.v)
operator < : Two-key compare
operator << : Polar angle compare
trans : Transition with a 2x2 matrix
*/
struct line {
    double A, B, C;  // Ax + By + C = 0, > 0 if it represents halfplane.
    line(double A0 = 0.0, double B0 = 0.0, double C0 = 0.0)
        : A(A0), B(B0), C(C0) {}
    line(const vec2& u, const vec2& v)
        : A(u.y - v.y), B(v.x - u.x), C(u ^ v) {}  // left > 0
    inline vec2 normVec() const { return vec2(A, B); }
    inline double norm2() const { return A * A + B * B; }
    inline double operator()(const vec2& P) const {
        return A * P.x + B * P.y + C;
```

```cpp
    }
};
inline vec2 intersection(const line u, const line v) {
    return vec2(u.B * v.C - u.C * v.B, u.C * v.A - u.A * v.C)
             / (u.A * v.B - u.B * v.A);
}
inline bool parallel(const line u, const line v) {
    double p = u.normVec() ^ v.normVec();
    return eq(p, 0);
}
inline bool perpendicular(const line u, const line v) {
    double p = u.normVec() * v.normVec();
    return eq(p, 0);
}
inline bool sameDir(const line u, const line v) {
    return parallel(u, v) && gt(u.normVec() * v.normVec(), 0);
}
inline line bisector(const vec2 u, const vec2 v) {
    return line(v.x - u.x, v.y - u.y, 0.5 * (u.norm2() - v.norm2()));
}
inline double dis2(const vec2 P, const line l) {
    return l(P) * l(P) / l.norm2();
}
inline vec2 projection(const vec2 P, const line l) {
    return P - l.normVec() * (l(P) / l.norm2());
}
inline vec2 symmetry(const vec2 P, const line l) {
    return P - l.normVec() * (2 * l(P) / l.norm2());
}
// Relation of 3 points. (2 inside, 1 outside, 0 not collinear)
inline int collinear(const vec2 u, const vec2 v, const vec2 P) {
    double p = cross(P, u, v);
    return eq(p, 0) ? 1 + le(dot(P, u, v), 0) : 0;
}
// Perimeter of a polygon
double perimeter(int n, vec2* poly) {
    double ret = (poly[n - 1] - *poly).norm();
    for (int i = 1; i < n; ++i) ret += (poly[i - 1] - poly[i]).norm();
    return ret;
}
// Directed area of a polygon (positive if CCW)
double area(int n, vec2* poly) {
    double ret = poly[n - 1] ^ *poly;
    for (int i = 1; i < n; ++i) ret += poly[i - 1] ^ poly[i];
    return ret * 0.5;
}
// Point in polygon (2 on boundary, 1 inside, 0 outside)
int contain(int n, vec2* poly, const vec2 P) {
    int in = 0;
    vec2 *r = poly + (n - 1), *u, *v;
    for (int i = 0; i < n; r = poly, ++poly, ++i) {
        if (collinear(*r, *poly, P) == 2) return 2;
        gt(r->y, poly->y) ? (u = poly, v = r) : (u = r, v = poly);
        if (ge(P.y, v->y) || lt(P.y, u->y)) continue;
        in ^= gt(cross(P, *u, *v), 0);
    }
    return in;
}
// Convex Hall
int graham(int n, vec2* p, vec2* dest) {
    int i;
    vec2* ret = dest;
    std::iter_swap(p, std::min_element(p, p + n));
    std::sort(p + 1, p + n, [p](const vec2 A, const vec2 B) {
        double r = cross(*p, A, B);
        return gt(r, 0) || (ge(r, 0) && lt((A - *p).norm2(), (B - *p).norm2()));
    });
    for (i = 0; i < 2 && i < n; ++i) *ret++ = p[i];
    for (; i < n; *ret++ = p[i++])
        for (; ret != dest + 1 && ge(cross(ret[-2], p[i], ret[-1]), 0); --ret)
```

```cpp
            ;
    return *ret = *p, ret - dest;
}
double convDiameter(int n, vec2* poly) {
    int l = std::min_element(poly, poly + n) - poly,
        r = std::max_element(poly, poly + n) - poly, i = l, j = r;
    double diam = (poly[l] - poly[r]).norm2();
    do {
        (ge(poly[(i + 1) % n] - poly[i] ^ poly[(j + 1) % n] - poly[j], 0)
             ? ++j
             : ++i) %= n;
        up(diam, (poly[i] - poly[j]).norm2());
    } while (i != l || j != r);
    return diam;
}
//
inline vec2 circumCenter(const vec2 A, const vec2 B, const vec2 C) {
    vec2 a = B - A, b = C - A, AO;
    double det = 0.5 / (a ^ b), na = a.norm2(), nb = b.norm2();
    AO = vec2((na * b.y - nb * a.y) * det, (nb * a.x - na * b.x) * det);
    return A + AO;
}
double minCircleCover(int n, vec2* p, vec2* ret = NULL) {
    int i, j, k;
    double r2 = 0.0;
    std::random_shuffle(p + 1, p + (n + 1));
    vec2 C = p[1];
    for (i = 2; i <= n; ++i)
        if (gt((p[i] - C).norm2(), r2))
            for (C = p[i], r2 = 0, j = 1; j < i; ++j)
                if (gt((p[j] - C).norm2(), r2))
                    for (C = (p[i] + p[j]) * 0.5, r2 = (p[j] - C).norm2(),
                            k = 1;
                         k < j; ++k)
                        if (gt((p[k] - C).norm2(), r2))
                            C = circumCenter(p[i], p[j], p[k]),
                            r2 = (p[k] - C).norm2();
    return ret ? *ret = C : 0, r2;
}
//半平面交
inline bool HPIcmp(const line u, const line v) {
    return sameDir(u, v) ? gt((fabs(u.A) + fabs(u.B)) * v.C,
                              (fabs(v.A) + fabs(v.B)) * u.C)
                         : u.normVec() << v.normVec();
}
inline bool geStraight(const vec2 A, const vec2 B) {
    return lt(A ^ B, 0) || le(A ^ B, 0) && lt(A * B, 0);
}
inline bool para_nega_test(const line u, const line v) {
    return parallel(u, v) && lt(u.normVec() * v.normVec(), 0)
           && (fabs(u.A) + fabs(u.B)) * v.C + (fabs(v.A) + fabs(v.B)) * u.C
                  < -7e-14;
}
int HPI(int n, line* l, vec2* poly, vec2*& ret) {  // -1 : Unbounded, -2 :
    Infeasible int h = 0, t = -1, i, j, que[n + 5];
    std::sort(l, l + n, HPIcmp);
    n = std::unique(l, l + n, sameDir) - l;
    for (j = i = 0; i < n && j < n; ++i) {
        for (up(j, i + 1); j < n && !geStraight(l[i].normVec(), l[j].normVec());
             ++j)
            ;
        if (para_nega_test(l[i], l[j])) return -2;
    }
    if (n <= 2 || geStraight(l[n - 1].normVec(), l->normVec())) return -1;
    for (i = 0; i < n; ++i) {
        if (geStraight(l[que[t]].normVec(), l[i].normVec())) return -1;
        for (; h < t && le(l[i](poly[t - 1]), 0); --t)
            ;
        for (; h < t && le(l[i](poly[h]), 0); ++h)
```

```
            ;
        que[++t] = i;
        h < t ? poly[t - 1] = intersection(l[que[t - 1]], l[que[t]]) : 0;
    }
    for (; h < t && le(l[que[h]](poly[t - 1]), 0); --t)
        ;
    return h == t ? -2
                  : (poly[t] = intersection(l[que[t]], l[que[h]]),
                     ret = poly + h, t - h + 1);
}
// circles
const double pi = M_PI, pi2 = pi * 2., pi_2 = M_PI_2;
inline double angle(const vec2 u, const vec2 v) { return atan2(u ^ v, u * v); }
// intersection of circle and line
int intersection(double r2, const vec2 O, const line l, vec2* ret) {
    double d2 = dis2(O, l);
    vec2 j = l.normVec();
    if (gt(d2, r2)) return ret[0] = ret[1] = vec2(INFINITY, INFINITY), 0;
    if (ge(d2, r2)) return ret[0] = ret[1] = projection(O, l), 1;
    if (le(d2, 0)) {
        j = j * sqrt(r2 / j.norm2());
        ret[0] = O + j.trans(0, -1, 1, 0);
        ret[1] = O + j.trans(0, 1, -1, 0);
    } else {
        double T = copysign(sqrt((r2 - d2) / d2), l(O));
        j = j * (-l(O) / j.norm2());
        ret[0] = O + j.trans(1, T, -T, 1);
        ret[1] = O + j.trans(1, -T, T, 1);
    }
    return 2;
}
// area of intersection(x^2 + y^2 = r^2, triangle OBC)
double interArea(double r2, const vec2 B, const vec2 C) {
    if (eq(B ^ C, 0)) return 0;
    vec2 is[2];
    if (!intersection(r2, vec2(), line(B, C), is))
        return 0.5 * r2 * angle(B, C);
    bool b = gt(B.norm2(), r2), c = gt(C.norm2(), r2);
    if (b && c)
        return 0.5
               * (lt(dot(*is, B, C), 0)
                      ? r2 * (angle(B, *is) + angle(is[1], C)) + (is[0] ^ is[1])
                      : r2 * angle(B, C));
    else if (b) return 0.5 * (r2 * angle(B, *is) + (*is ^ C));
    else if (c) return 0.5 * ((B ^ is[1]) + r2 * angle(is[1], C));
    else return 0.5 * (B ^ C);
}
// tangents to circle((0, 0), r) through P
int tangent(double r, const vec2 P, line* ret) {
    double Q = P.norm2() - r * r;
    if (lt(Q, 0))
        return ret[0] = ret[1] = line(INFINITY, INFINITY, INFINITY), 0;
    if (le(Q, 0)) return ret[0] = ret[1] = line(P.x, P.y, -P.norm2()), 1;
    Q = sqrt(Q) / r;
    ret[0] = line(P.x + Q * P.y, P.y - Q * P.x, -P.norm2());
    ret[1] = line(P.x - Q * P.y, P.y + Q * P.x, -P.norm2());
    return 2;
}
// tangets to circle(O, r) through P
int tangent(double r, const vec2 O, const vec2 P, line* ret) {
    int R = tangent(r, P - O, ret);
    if (R)
        ret[0].C -= ret[0].A * O.x + ret[0].B * O.y,
            ret[1].C -= ret[1].A * O.x + ret[1].B * O.y;
    return R;
}
```

7.2 自适应辛普森

```
double Simpson(double L, double M, double R, double fL, double fM, double fR,
               double eps) {
|  double LM = (L + M) * 0.5, fLM = f(LM), MR = (M + R) * 0.5, fMR = f(MR);
|  double A = (fL + fM * 4.0 + fR) * (R - L) * sixth,
|         AL = (fL + fLM * 4.0 + fM) * (M - L) * sixth,
|         AR = (fM + fMR * 4.0 + fR) * (R - M) * sixth;
|  if (fabs(AL + AR - A) < eps) return (2.0 * (AL + AR) + A) * third;
|  return Simpson(L, LM, M, fL, fLM, fM, eps * 0.6)
|         + Simpson(M, MR, R, fM, fMR, fR, eps * 0.6);
}
```

8 杂项

Delete This

```
```