Contents

1 图论

1.1 二分图匹配

1.1.1 二分图最大匹配 | 最小边覆盖

左到右连单向边，时间复杂度 $O(M|match|)$。

```
std::vector<int> edge[N];
std::vector<int> match(int nl, int nr) {
  std::vector<int> vis(nr + 1), match(nr + 1), ret(nl + 1);
  auto dfs = [&](auto dfs, int x) -> int {
    for(int y : edge[x]) if(!vis[y])
      if(vis[y] = 1, !match[y] || dfs(dfs, match[y]))
        return match[y] = x, 1;
    return 0;
  };
  for(int i = 1;i <= nl;++i) if(dfs(dfs, i))
    memset(vis.data(), 0, vis.size() << 2);
  for(int i = 1;i <= nr;++i) if(int t = match[i]) ret[t] = i;
  return ret;
}
std::array<std::vector<int>, 3> minedgeconver(int nl, int nr) {
  std::vector<int> vis(nr + 1), match(nr + 1), ret(nl + 1);
  auto dfs = [&](auto dfs, int x) -> int {
    for(int y : edge[x]) if(!vis[y])
      if(vis[y] = 1, !match[y] || dfs(dfs, match[y]))
        return match[y] = x, 1;
    return 0;
  };
  for(int i = 1;i <= nl;++i) if(dfs(dfs, i))
    memset(vis.data(), 0, vis.size() << 2);
  for(int i = 1;i <= nr;++i) if(int t = match[i]) ret[t] = i;
  for(int i = 1;i <= nl;++i) if(!ret[i]) dfs(dfs, i);
  std::vector<int> le, ri;
  for(int i = 1;i <= nl;++i) if(ret[i] && !vis[ret[i]]) le.push_back(i);
  for(int i = 1;i <= nr;++i) if(vis[i]) ri.push_back(i);
  return {le, ri, ret};
}
```

1.1.2 二分图最大匹配 (bitset 优化)

左到右连单向边，时间复杂度 $O(n^2/w\ |match|)$。

```
const int N = 505;
using set = std::bitset<N>;
set edge[N];
std::vector<int> match(int nl, int nr) {
  set unvis; unvis.set();
  std::vector<int> match(nr + 1), ret(nl + 1);
  auto dfs = [&](auto dfs, int x) {
    for(set z = edge[x];;) {
      z &= unvis;
      int y = z._Find_first();
      if(y == N) return 0;
      if(unvis.reset(y), !match[y] || dfs(dfs, match[y])) {
        return match[y] = x, 1;
      }
    }
  };
  for(int i = 1;i <= nl;++i)
    if(dfs(dfs, i)) unvis.set();
  for(int i = 1;i <= nr;++i) ret[match[i]] = i;
  return ret[0] = 0, ret;
}
```

1.1.3 二分图最大匹配 (Hopcroft-Karp) & 最小边覆盖

左到右单向边，时间复杂度 $O(M\sqrt{|match|})$

```
std::vector<int> e[N];
std::vector<int> matchl, matchr, a, p;
std::vector<int> match(int nl, int nr) {
  matchl.assign(nl + 1, 0), matchr.assign(nr + 1, 0);
  for(;;) {
    a.assign(nl + 1, 0), p.assign(nl + 1, 0);
    static std::queue<int> Q;
    for(int i = 1;i <= nl;++i)
      if(!matchl[i]) a[i] = p[i] = i, Q.push(i);
    int succ = 0;
    for(;Q.size();) {
      int x = Q.front(); Q.pop();
      if(matchl[a[x]]) continue;
      for(int y : e[x]) {
        if(!matchr[y]) {
          for(succ = 1;y;x = p[x])
            matchr[y] = x, std::swap(matchl[x], y);
          break;
        }
        if(!p[matchr[y]])
          Q.push(y = matchr[y]), p[y] = x, a[y] = a[x];
      }
    }
    if(!succ) break;
  }
  return matchl;
}
std::array<std::vector<int>, 3> minedgecover(int nl, int nr) {
  match(nl, nr);
  std::vector<int> l, r;
  for(int i = 1;i <= nl;++i) if(!a[i]) l.push_back(i);
  for(int i = 1;i <= nr;++i) if(a[matchr[i]]) r.push_back(i);
  return {l, r, matchl};
}
```

1.1.4 二分图最小权匹配

返回左边每个点匹配的右边点的编号，左边的点不多于右边的点（必要时补点）。

```cpp
ll e[N][N];
std::vector<int> KM(int n, int m) {
  std::vector<ll> l(n + 1), r(m + 1);
  std::vector<int> p(m + 1), ans(n + 1);
  for(int i = 1;i <= n;++i) {
    std::vector<ll> d(m + 1, 1e18);
    std::vector<int> pre(m + 1), done(m + 1);
    int x, v, u = 0;
    for(p[0] = i;x = p[u];u = v) {
      done[u] = 1;
      ll min = 1e18;
      for(int j = 1;j <= m;++j) if(!done[j]) {
        auto w = e[x][j] - l[x] - r[j];
        if(w < d[j]) d[j] = w, pre[j] = u;
        if(d[j] < min) min = d[j], v = j;
      }
      for(int j = 0;j <= m;++j) {
        if(done[j]) l[p[j]] += min, r[j] -= min;
        else d[j] -= min;
      }
    }
    for(int v;u;u = v) {
      v = pre[u], p[u] = p[v];
    }
  }
  for(int j = 1;j <= m;++j) ans[p[j]] = j;
  return ans; // 权值和 : -r[0]
}
```

最小权匹配。返回代价和、右边每个点匹配的左边点的编号、对应的边权。过程中能计算出匹配为 $0\cdots|match|$ 的最优解。时间复杂度 $O(|match|(L^2+|E|))$。

```cpp
std::vector<std::pair<int, ll>> e[N];
bool down(ll & x, ll y) { return x > y ? x = y, 1 : 0; }
auto KM(int n, int m) {
  std::vector<ll> h(n + 1), rv(m + 1);
  std::vector<int> R(m + 1);
  ll cost = 0;
  for(;;) {
    std::vector<ll> d(n + 1);
    std::vector<int> vis(n + 1), p0(n + 1), p1(m + 1);
    for(int i = 0;i <= m;++i) d[R[i]] = inf;
    for(int x = 0;vis[x] ^= 1;) {
      ll min = 1e18;
      for(auto [j, w] : e[x]) if(down(d[R[j]], d[x] + w - rv[j])) p0[R[j]]
        = j, p1[j] = x;
      for(int j = 1;j <= n;++j) if(!vis[j] && down(min, d[j] - h[j])) x =
        j;
    }
    if(d[0] >= inf / 2) break; // or inf/2⇒0
    for(int r = p0[0];r;r = p0[R[r]]) {
      rv[r] += d[R[r]] - d[p1[r]];
      R[r] = p1[r];
    }
    cost += d[0], h = d;
  }
  return std::tuple(cost, R, rv);
}
```

1.2 网络最大流 (dinic)

```cpp
// S 编号最小, T 最大, 或者改一下清空
struct Dinic {
  struct T {
    int to, nxt, v;
  } e[N << 3];
  int h[N], head[N], num = 1;
  void link(int x, int y, int v) {
    e[++num] = {y, h[x], v}, h[x] = num;
    e[++num] = {x, h[y], 0}, h[y] = num; // !!!
  }
  int dis[N];
  bool bfs(int s, int t) {
    std::queue<int> Q;
    for(int i = s;i <= t;++i) dis[i] = -1, head[i] = h[i]; //如果编号不是
      [S,T], 只要改这里
    for(Q.push(t), dis[t] = 0;!Q.empty();) {
      int x = Q.front(); Q.pop();
      for(int i = h[x];i;i = e[i].nxt) if(e[i ^ 1].v && dis[e[i].to] < 0)
        {
        dis[e[i].to] = dis[x] + 1, Q.push(e[i].to);
      }
    }
    return dis[s] >= 0;
  }
  int dfs(int s, int t, int lim) {
    if(s == t || !lim) return lim;
    int ans = 0, mn;
    for(int & i = head[s];i;i = e[i].nxt) {
      if(dis[e[i].to] + 1 == dis[s] && (mn = dfs(e[i].to, t, std::min(lim,
        e[i].v)))) {
        e[i].v -= mn, e[i ^ 1].v += mn;
        ans += mn, lim -= mn;
        if(!lim) break;
      }
    }
    return ans;
  }
  int flow(int s, int t) {
    int ans = 0;
    for(;bfs(s, t);) ans += dfs(s, t, 1e9);
    return ans;
  }
} G;
```

1.3 最小费用流

```
// S 编号最小，T 最大，或者改一下清空
namespace mcmf {
  using pr = std::pair<ll, int>;
  const int N = 10005, M = 1e6 + 10;
  struct edge {
    int to, nxt, v, f;
  } e[M << 1];
  int h[N], num = 1;
  void link(int x, int y, int v, int f) {
    e[++num] = {y, h[x], v, f}, h[x] = num;
    e[++num] = {x, h[y], 0, -f}, h[y] = num;
  }
  ll d[N], dis[N];
  int vis[N], fr[N];
  bool spfa(int s, int t) {
    std::queue<int> Q;
    std::fill(d + s, d + t + 1, 1e18); // CHECK
    for(d[s] = 0, Q.push(s);!Q.empty();) {
      int x = Q.front(); Q.pop(); vis[x] = 0;
      for(int i = h[x];i;i = e[i].nxt)
        if(e[i].v && d[e[i].to] > d[x] + e[i].f) {
          d[e[i].to] = d[x] + e[i].f;
          fr[e[i].to] = i;
          if(!vis[e[i].to]) vis[e[i].to] = 1, Q.push(e[i].to);
        }
    }
    return d[t] < 1e17;
  }
  bool dijkstra(int s, int t) { // 正常题目不需要 dijk
    std::priority_queue<pr, std::vector<pr>, std::greater<pr>> Q;
    for(int i = s;i <= t;++i) dis[i] = d[i], d[i] = 1e18, vis[i] = fr[i] = 0; // CHECK
    for(Q.emplace(d[s] = 0, s);!Q.empty();) {
      int x = Q.top().second; Q.pop();
      if(vis[x]) continue;
      vis[x] = 1;
      for(int i = h[x];i;i = e[i].nxt) {
        const ll v = e[i].f + dis[x] - dis[e[i].to];
        if(e[i].v && 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 = s;i <= t;++i) d[i] += dis[i]; // CHECK
    return d[t] < 1e17;
  }
  std::pair<ll, ll> EK(int s, int t) {
    spfa(s, t); // 如果初始有负权且要 dijk
    ll f = 0, c = 0;
    for(;dijkstra(s, t);) { // 正常可以用 spfa
      ll fl = 1e18;
      for(int i = fr[t];i;i = fr[e[i ^ 1].to]) fl = std::min<ll>(e[i].v, fl);
      for(int i = fr[t];i;i = fr[e[i ^ 1].to]) e[i].v -= fl, e[i ^ 1].v += fl;
      f += fl, c += fl * d[t];
    }
    return std::make_pair(f, c);
  }
}
// in flow problems with lower bounds (or with negative cycles), flow the negative edges first
// after the first round, revert the auxiliary edges
```

1.4 一般图最大匹配 (带花树)

edge 里存双向边。运行结束后 match 存放每个点匹配的点的编号，如果没有匹配则为 0。

```
int n;
std::vector<int> edge[N];
int pre[N], match[N], fa[N], vis[N], t[N];
int find(int x) {
  return fa[x] == x ? x : fa[x] = find(fa[x]);
}
void aug(int s) {
  std::queue<int> Q;
  for(int i = 1;i <= n;++i) {
    pre[i] = vis[i] = t[i] = 0, fa[i] = i;
  }
  auto lca = [&](int x, int y) {
    static int tick = 0; ++tick;
    for(;;std::swap(x = pre[match[x]], y)) {
      if(vis[x = find(x)] == tick) return x;
      vis[x] = x ? tick : 0;
    }
  };
  auto blossom = [&](int x, int y, int z) {
    for(;find(x) != z;x = pre[y]) {
      pre[x] = y, y = match[x], fa[x] = fa[y] = z;
      if(t[y] == 2) Q.push(y), t[y] = 1;
    }
  };
  Q.push(s), t[s] = 1;
  for(;Q.size();) {
    int x = Q.front(); Q.pop();
    for(int y : edge[x]) {
      if(t[y] == 2) continue;
      if(t[y] == 1) {
        int z = lca(x, y);
        blossom(x, y, z);
        blossom(y, x, z);
      } else if (int p = match[y]) {
        pre[y] = x, t[y] = 2, t[p] = 1;
        Q.push(p);
      } else {
```

```
        int z;
        for(pre[y] = x;y;y = z) {
          x = pre[y], z = match[x];
          match[x] = y, match[y] = x;
        }
        return ;
      }
    }
  }
}
void run() {
  for(int i = 1;i <= n;++i) if(!match[i]) {
    aug(i);
  }
}
```

1.5 最小树形图

用 DMST::link 添加边，然后调用 DMST::solve(root) 得到最小树形图边的集合，如果没有合法解会返回空集 (注意 $n = 1$)。

```
namespace DMST {
  struct edge {
    int u, v, id; ll w;
    bool operator < (const edge & y) const {
      return w < y.w;
    }
  } ent[N], val[M];
  int ls[M], rs[M], size[M], cc; ll tag[M];
  int fs[N], fw[N], rt[N];
  void put(int x, ll v) {
    if(x) val[x].w += v, tag[x] += v;
  }
  void pushdown(int x) {
    put(ls[x], tag[x]);
    put(rs[x], tag[x]);
    tag[x] = 0;
  }
  int merge(int x, int y) {
    if(!x || !y) return x | y;
    if(val[y] < val[x]) std::swap(x, y);
    pushdown(x), rs[x] = merge(rs[x], y);
    if(size[rs[x]] > size[ls[x]]) {
      std::swap(ls[x], rs[x]);
    }
    size[x] += size[y];
    return x;
  }
  void ins(int & x, const edge & z) {
    val[++cc] = z, size[cc] = 1;
    x = merge(x, cc);
  }
  void pop(int & x) { x = merge(ls[x], rs[x]); }
  edge top(int x) { return val[x]; }
  int find(int x, int * anc) {
    return anc[x] == x ? x : anc[x] = find(anc[x], anc);
  }
  void link(int u, int v, int w, int id) {
    ins(rt[v], {u, v, id, w});
  }
  int pa[N * 2], tval[N * 2], up[N * 2], end_edge[M], cmt, baned[M];
  std::vector<int> solve(int r) {
    std::queue<int> roots;
    for(int i = 1;i <= n;++i) {
      fs[i] = fw[i] = i, tval[i] = ++ cmt;
      if(i != r) roots.push(i);
    }
    std::vector<edge> H;
    std::vector<int> ret;
    for(;!roots.empty();) {
      int k = roots.front(); roots.pop();
      if(!rt[k]) return ret;
      edge e = top(rt[k]); pop(rt[k]);
      int i = e.u, j = e.v;
      if(find(i, fs) == k) roots.push(k);
      else {
        H.push_back(e); end_edge[e.id] = tval[k];
        if(find(i, fw) != find(j, fw)) {
          fw[find(j, fw)] = i;
          ent[k] = e;
        } else {
          pa[tval[k]] = ++ cmt, up[tval[k]] = e.id;
          put(rt[k], -e.w);
          for(;(e = ent[find(e.u, fs)]).u;) {
            int p = find(e.v, fs);
            pa[tval[p]] = cmt;
            up[tval[p]] = e.id;
            put(rt[p], -e.w);
            rt[k] = merge(rt[k], rt[p]);
            fs[p] = k;
          }
          tval[k] = cmt;
          roots.push(k);
        }
      }
    }
    reverse(H.begin(), H.end());
    for(edge i : H) if(!baned[i.id]) {
      ret.push_back(i.id);
      for(int j = i.v;j != end_edge[i.id];j = pa[j]) ++ baned[up[j]];
    }
    sort(ret.begin(), ret.end());
    return ret;
  }
}
```

1.6 强连通分量 (kosaraju)

e 存有向边，re 里存反向边。用 solve 得到所有强连通分量。时间复杂度 $O(\frac{n^2}{w})$。

```cpp
using set = std::bitset<N>;
// re 是反向边，需要连好
set e[N], re[N], vis;
std::vector<int> sta;
void dfs0(int x, set * e) {
  vis.reset(x);
  for(;;) {
    int go = (e[x] & vis)._Find_first();
    if(go == N) break;
    dfs0(go, e);
  }
  sta.push_back(x);
}
std::vector<std::vector<int>> solve() {
  vis.set();
  for(int i = 1;i <= n;++i) if(vis.test(i)) dfs0(i, e);
  vis.set();
  auto s = sta;
  std::vector<std::vector<int>> ret;
  for(int i = n - 1;i >= 0;--i) if(vis.test(s[i])) {
    sta.clear(), dfs0(s[i], re), ret.push_back(sta);
  }
  return ret;
}
```

1.7 欧拉回路/欧拉路径

如果是无向图，两个方向都连边，id 相同。不同的边 id 不同且在 $[1, m]$ 里。

src 是起点，如果是欧拉路径，找一个度数为奇数 (无向图)/出度大于入度 (有向图) 的点作为 src，如果是回路，选一个度数非零的点作为 src。

如果有解，求出路径是 s，那么返回 $[(s_1 = src, -1), (s_2, id(s_1 \to s_2)), \ldots, (s_n, id(s_{n-1} \to s_n))]$，如果是回路，那么 $s_n = s_1 = src$。无解则返回空集。

```cpp
std::vector<pr> e[N]; // (dest, id)
std::vector<pr> eulerwalk(int n, int m, int src) {
  std::vector<int> d(n + 1), it(n + 1), vis(m + 1);
  std::vector<pr> ret, s = {pr(src, -1)};
// ++ d[src]; // 如果需要欧拉路径，加上这行
  for(;s.size();) {
    auto [x, id] = s.back();
    int & i = it[x];
    if(i == (int) e[x].size()) {
      ret.push_back(s.back()), s.pop_back();
      continue;
    }
    auto [y, o] = e[x][i++];
    if (!vis[o]) {
      -- d[x], ++ d[y];
      vis[o] = 1, s.emplace_back(y, o);
    }
  }
  for(int x : d) if(x < 0 || size(ret) != m + 1) return {};
  return std::vector(ret.rbegin(), ret.rend());
}
```

1.8 支配树

```cpp
std::vector<int> e[N];
struct domtree {
  std::vector<int> dfn, p, sdom, dom, anc, min, pa;
  std::vector<std::vector<int>> b, re;
  int cc;
  void dfs0(int x) {
    dfn[x] = ++cc, p[cc] = x;
    for(int y : e[x]) {
      if(!dfn[y]) dfs0(y), pa[dfn[y]] = dfn[x];
      re[dfn[y]].push_back(dfn[x]);
    }
  }
  int find(int x) {
    int t = anc[x];
    if(anc[t] == t) return t;
    anc[x] = find(t);
    if(sdom[min[t]] < sdom[min[x]]) min[x] = min[t];
    return anc[x];
  }
  int get(int x) { return find(x), min[x]; }
  domtree() : b(n + 1), re(n + 1), cc(0) { }
  auto getdom(int s) {
    pa = dfn = p = dom = anc = std::vector<int>(n + 1);
    dfs0(s);
    for(int i = 1;i <= cc;++i) anc[i] = i;
    dom = sdom = min = anc;
    for(int i = cc;i;--i) {
      for(int w : re[i]) sdom[i] = std::min(sdom[i], sdom[get(w)]);
      if(i > 1) b[sdom[i]].push_back(i);
      for(int w : b[i]) {
        int v = get(w);
        dom[w] = sdom[v] == sdom[w] ? sdom[w] : v;
      }
      anc[i] = pa[i];
    }
    std::vector<int> ans(n + 1);
    for(int i = 1;i <= cc;++i) {
      if(dom[i] != sdom[i]) dom[i] = dom[dom[i]];
      ans[p[i]] = p[dom[i]];
    }
    return ans;
  }
};
```

1.9 Tree And Graph

1.9.1 树的计数 Prufer序列

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 d_i-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.

相关结论: n 个点完全图, 每个点度数依次为 d_1,d_2,...,d_n, 这样生成树的棵树为: $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\ldots(d_n-1)!}$.

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: $(\sum c_i)^{m-2}\prod c_i$

1.9.2 有根树的计数

首先, 令 $S_{n,j}=\sum_{1\le j\le n/j}$; 于是 $n+1$ 个结点的有根树的总数为 $a_{n+1}=\frac{\sum_{j=1}^{n} j a_j S_{n-j}}{n}$. 注: $a_1=1, a_2=1, a_3=2, a_4=4, a_5=9, a_6=20, a_9=286, a_{11}=1842$.

1.9.3 无根树的计数

n 是奇数时, 有 $a_n-\sum_i^{n/2} a_i a_{n-i}$ 种不同的无根树.

n 时偶数时, 有 $a_n-\sum_i^{n/2} a_i a_{n-i}+\frac{1}{2}a_{n/2}(a_{n/2}+1)$ 种不同的无根树.

1.9.4 生成树计数 Kirchhoff's Matrix-Tree Thoerem

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 $n1$ 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 $|$

1.9.5 有向图欧拉回路计数 BEST Thoerem

$$\mathrm{ec}(G)=t_w(G)\prod_{v\in V}(\deg(v)-1)!$$

其中 deg 为入度 (欧拉图中等于出度), $t_w(G)$ 为以 w 为根的外向树的个数. 相关计算参考生成树计数.

欧拉连通图中任意两点外向树个数相同: $\mathrm{t_v}(G)=\mathrm{t_w}(G)$.

以 1 结尾的欧拉路径计数就是把 deg 视为出度，把 $\deg(1)$ 的贡献改为 $\deg(1)!$.

1.9.6 Tutte Matrix

Tutte matrix A of a graph $G=(V,E)$:

$$A_{ij}=\begin{cases} x_{ij} & \text{if } (i,j)\in E \text{ and } i<j \\ -x_{ij} & \text{if } (i,j)\in E \text{ and } i>j \\ 0 & \text{otherwise} \end{cases}$$

where x_{ij} are indeterminates. The determinant of this skew-symmetric matrix is then a polynomial (in the variables x_{ij}, $i<j$): this coincides with the square of the pfaffian of the matrix A and is non-zero (as a polynomial) if and only if a perfect matching exists.

1.9.7 Edmonds Matrix

Edmonds matrix A of a balanced ($|U|=|V|$) bipartite graph $G=(U,V,E)$:

$$A_{ij}=\begin{cases} x_{ij} & (u_i,v_j)\in E \\ 0 & (u_i,v_j)\notin E \end{cases}$$

where the x_{ij} are indeterminates. G 有完美匹配当且仅当关于 x_{ij} 的多项式 $det(A_{ij})$ 不恒为 0. 完美匹配的个数等于多项式中单项式的个数.

1.9.8 Erdős-Gallai theorem

度数序列 $d_1\ge\cdots\ge d_n$ 存在对应的简单图当且仅当：

$$\begin{cases} \sum_{i=1}^{n} d_i \text{ 为偶数} \\ \forall k\in[1,n]:\sum_{i=1}^{k} d_i\le k(k-1)+\sum_{i=k+1}^{n}\min(d_i,k) \end{cases}$$

2 数论

2.1 取模还原分数

```
std::pair<int, int> approx(int p, int q, int A) {
  int x = q, y = p, a = 1, b = 0;
  for(;x > A;) {
    std::swap(x, y), std::swap(a, b);
    a -= x / y * b, x %= y;
  }
  return {x, a};
} // q ≡ x/a (mod p), x ≤ A, |a| 取到最小值
```

2.2 扩展欧几里得

结果的 x,y 满足：$ax+by=\gcd(a,b)$; 若 a,b 非 0，则：$-b\le x\le b, -a\le y\le a$。

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

2.3 万能欧几里得

```
// 万欧
// 前提 : r < q, r >= q 先提几个 U 出来再用
// 使用: Y * q <= X * p + r, 斜率 p/q, U表示向上, R表示到达一个顶点, 先一些 U 再一个 R
template<class T>
T power(T a, ll k) {
  // 有效率需求可以改为半群乘法
  if(!k) return T();
  T res = a;
  for(--k;k;) {
    if(k & 1) res = res + a;
    if(k >>= 1) a = a + a;
  }
```

```
  | return res;
}
template<class T>
T solve(ll p, ll q, ll r, ll l, T U, T R) {
  | if (p >= q)
  |   | return solve(p % q, q, r, l, U, power(U, p / q) + R);
  | ll m = ((__int128)p * l + r) / q;
  | if (!m) return power(R, l);
  | ll cnt = l - ((__int128)q * m - r - 1) / p;
  | return power(R, (q - r - 1) / p) + U + solve(q, p, (q - r - 1) % p, m - 1,
    R, U) + power(R, cnt);
}
```

2.4 floor-sum

$n < 2^{32}, 1 \le m < 2^{32}$

$$result = \sum_{i=0}^{n-1} \lfloor \frac{ai+b}{m} \rfloor \pmod{2^{63}}$$

```
u64 floor_sum(u64 n, u64 m, u64 a, u64 b) {
  | u64 ans = 0;
  | for(;;) {
  |   | if(a >= m) ans += n * (n - 1) / 2 * (a / m), a %= m;
  |   | if(b >= m) ans += n * (b / m), b %= m;
  |   | u64 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;
}
```

2.5 Min of Mod of Linear

$$result = \min_{0 \le i < n} \{(ai+b)\%m\}$$

```
int minmod(int n, int m, int a, ll b) {
  | int ans = m;
  | for(;n;std::swap(a, m)) {
  |   | a %= m, b %= m;
  |   | if(b < 0) b += m;
  |   | if(b < ans) ans = b;
  |   | n = (ll(n - 1) * a + b) / m;
  |   | b -= (ll) m * n;
  | }
  | return ans;
}
```

2.6 Stern-Brocot Tree 二分

```
using cp = std::complex<ll>;
cp fracBS(ll n, ll m, auto f) {
  | bool dir = 1, A = 1, B = 1;
  | cp lo(0, 1), hi(1, 1); // hi can be (1, 0), f(hi) must be true
  | if (f(lo)) return lo;
  | while(A || B) {
  |   | ll adv = 0, s = 1;
  |   | for (int x = 0;s;(s *= 2) >>= x) {
  |   |   | adv += s;
  |   |   | cp mid = lo * adv + hi;
  |   |   | if (mid.real() > n || mid.imag() > m || dir == !f(mid)) {
  |   |   |   | adv -= s, x = 2;
  |   |   | }
  |   | }
  |   | hi += lo * adv, dir = !dir;
  |   | swap(lo, hi);
  |   | A = B, B = adv;
  | }
  | return dir ? hi : lo;
} // 返回值是最小的使得 f 为真的
  // 另外一个是最大的使得 f 为假的
```

2.7 中国剩余定理

返回满足 $x \equiv a_i \pmod{m_i} (i = 1, 2)$ 的最小非负整数 x，若无解则返回 -1。

```
ll CRT(ll a1, ll p1, ll a2, ll p2) {
  | ll a, b, gcd = std::gcd(p1, p2);
  | if((a1 - a2) % gcd)
  |   | return -1;
  | exgcd(p1, p2, a, b);
  | ll k = i128((a2 - a1) % p2 + p2) * (a + p2) % p2;
  | return p1 / gcd * k + a1;
}
```

2.8 Miller-Rabin

调用 checkprime 判断是否为质数。

```
using f64 = long double;
ll p;
f64 invp;
void setmod(ll x) {
  | p = x, invp = (f64) 1 / x;
}
ll mul(ll a, ll b) {
  | ll z = a * invp * b + 0.5;
  | ll res = a * b - z * p;
  | return res + (res >> 63 & p);
}
ll pow(ll a, ll x, ll res = 1) {
```

```
  | for(;x;x >>= 1, a = mul(a, a))
  |   | if(x & 1) res = mul(res, a);
  | return res;
}
bool checkprime(ll p) {
  | if(p == 1) return 0;
  | setmod(p);
  | ll d = __builtin_ctzll(p - 1), s = (p - 1) >> d;
  | for(ll a : {2, 3, 5, 7, 11, 13, 82, 373}) {
  |   | if(a % p == 0)
  |   |   | continue;
  |   | ll x = pow(a, s), y;
  |   | for(int i = 0;i < d;++i, x = y) {
  |   |   | y = mul(x, x);
  |   |   | if(y == 1 && x != 1 && x != p - 1)
  |   |   |   | return 0;
  |   | }
  |   | if(x != 1) return 0;
  | }
  | return 1;
}
```

2.9 Pollard-rho

调用 factor 获得 x 质因子的可重集合（已从小到大排序）。

```
ll rho(ll n) {
  | if(!(n & 1)) return 2;
  | static std::mt19937_64 gen((size_t)"hehezhou");
  | ll x = 0, y = 0, prod = 1;
  | auto f = [&](ll o) { return mul(o, o) + 1; };
  | setmod(n);
  | for(int t = 30, z = 0;t % 64 || std::gcd(prod, n) == 1;++t) {
  |   | if (x == y) x = ++ z, y = f(x);
  |   | if(ll q = mul(prod, x + n - y)) prod = q;
  |   | x = f(x), y = f(f(y));
  | }
  | return std::gcd(prod, n);
}
std::vector<ll> factor(ll x) {
  | std::vector<ll> res;
  | auto f = [&](auto f, ll x) {
  |   | if(x == 1) return ;
  |   | if(checkprime(x)) return res.push_back(x);
  |   | ll y = rho(x);
  |   | f(f, y), f(f, x / y);
  | };
  | f(f, x), sort(res.begin(), res.end());
  | return res;
}
```

2.10 二次剩余

quadres 用于判断是否存在二次剩余，sqrtp 用于计算结果。p 或 b 需要是质数（或者 1）。

高斯引理：

$$\left(\frac{a}{p}\right) = (-1)^{\sum_{i=1}^{\frac{p-1}{2}} 1[(ai \bmod p) > p/2]}$$

推论：

$$\begin{cases} \left(\frac{-1}{p}\right) = 1 & \Leftrightarrow p \equiv 1 \pmod 4, \\ \left(\frac{2}{p}\right) = 1 & \Leftrightarrow p \equiv \pm 1 \pmod 8, \\ \left(\frac{3}{p}\right) = 1 & \Leftrightarrow p \equiv \pm 1 \pmod{12}, \\ \left(\frac{5}{p}\right) = 1 & \Leftrightarrow p \equiv \pm 1 \pmod 5. \end{cases}$$

```
bool quadres(u64 a, u64 b) {
  | if(a <= 1) return 1;
  | for(;a % 4 == 0;) a /= 4;
  | if(a % 2) return (a % 4 == 1 || b % 4 == 1) == quadres(b % a, a);
  | return (b % 8 == 1 || b % 8 == 7) == quadres(a / 2, b);
}
int sqrtp(int x, int p) {
  | if(x <= 1) return x;
  | static std::mt19937_64 gen;
  | int w, v, k = (p + 1) / 2;
  | do w = gen() % p; while(quadres(v = ((u64) w * w + p - x) % p, p));
  | using ai = std::array<int, 2>;
  | auto prod = [&](ai x, ai y) {
  |   | return ai{
  |   |   | ((u64) x[0] * y[0] + (u64) x[1] * y[1] % p * v) % p,
  |   |   | ((u64) x[0] * y[1] + (u64) x[1] * y[0]) % p
  |   | };
  | };
  | ai ans{1, 0}, a{w, 1};
  | for(;k;k >>= 1, a = prod(a, a))
  |   | if(k & 1) ans = prod(ans, a);
  | return ans[0];
}
```

3 Math

3.1 拉格朗日反演

$$G(F(x)) = H(x) \Rightarrow [x^n]G(x) = \frac{1}{n}[u^{n-1}]H'(u)\left(\frac{u}{F(u)}\right)^n$$

$$G(F(x)) = x \Rightarrow [x^n]H(G(x)) = \frac{1}{n}[u^{n-1}]H'(u)\left(\frac{u}{F(u)}\right)^n$$

$$G(F(x)) = x \Rightarrow [x^n]G^k(x) = \frac{k}{n}[u^{n-k}]\left(\frac{u}{F(u)}\right)^n$$

3.2 分拆数|五边形数

$$\prod_{i\geq 1}(1-x^i) = \sum_{k=-\infty}^{\infty}(-1)^k x^{\frac{k(3k-1)}{2}}$$

3.3 Fast Fourier Transform

```
using db = double;
using cp = std::complex<db>;
// cp::real, cp::imag, std::conj, std::arg
const db pi = std::acos(-1);
int rev[N], lim;
cp wn[N];
void init(int len) {
  for(lim = 1;lim < len;lim <<= 1);
  for(static int i = 1;i < lim;i += i) {
    for(int j = 0;j < i;++j) {
      wn[i + j] = std::polar(1., db(j) / i * pi);
    }
  }
  for(int i = 1;i < lim;++i) {
    rev[i] = rev[i >> 1] >> 1 | (i % 2u * lim / 2);
  }
}
void DFT(cp * a) {
  for(int i = 0;i < lim;++i) {
    if(rev[i] < i) std::swap(a[rev[i]], a[i]);
  }
  for(int i = 1;i < lim;i += i) {
    for(int j = 0;j < lim;j += i + i) {
      for(int k = 0;k < i;++k) {
        cp x = a[i + j + k] * wn[i + k];
        a[i + j + k] = a[k + j] - x;
        a[k + j] += x;
      }
    }
  }
}
void IDFT(cp * a) {
  DFT(a), std::reverse(a + 1, a + lim);
  for(int i = 0;i < lim;++i)
    a[i] /= lim;
}
```

3.4 Number Theoretic Transform

```
int rev[N], wn[N], lim, invlim;
int pow(int a, int b, int ans = 1) {
  for(;b;b >>= 1, a = (u64) a * a % mod) if(b & 1)
    ans = (u64) ans * a % mod;
  return ans;
}
void init(int len) {
  for(lim = 1;lim < len;lim <<= 1);
  invlim = mod - (mod - 1) / lim;
  for(static int i = 1;i < lim;i += i) {
    wn[i] = 1;
    const int w = pow(3, mod / i / 2);
    for(int j = 1;j < i;++j) {
      wn[i + j] = (u64) wn[i + j - 1] * w % mod;
    }
  }
  for(int i = 1;i < lim;++i) {
    rev[i] = rev[i >> 1] >> 1 | (i % 2u * lim / 2);
  }
}
void DFT(int * a) {
  static u64 t[N];
  for(int i = 0;i < lim;++i) t[i] = a[rev[i]];
  for(int i = 1;i < lim;i += i) {
    if(i >> 19 & 1)
      for(int k = 0;k < lim;++k) t[k] %= mod;
    for(int j = 0;j < lim;j += i + i) {
      for(int k = 0;k < i;++k) {
        const u64 x = t[i + j + k] * wn[i + k] % mod;
        t[i + j + k] = t[k + j] + (mod - x), t[k + j] += x;
      }
    }
  }
  for(int i = 0;i < lim;++i) a[i] = t[i] % mod;
}
void IDFT(int * a) {
  DFT(a), std::reverse(a + 1, a + lim);
  for(int i = 0;i < lim;++i)
    a[i] = (u64) a[i] * invlim % mod;
}
```

3.5 Generating function

```
void cpy(int * a, int * b, int n) {
  if(a != b) memcpy(a, b, n << 2);
  memset(a + n, 0, (lim - n) << 2);
}
void inv(int * a, int * b, int n) { // b = inv(a) mod x^n
  if(n == 1) return void(*b = pow(*a, mod - 2));
  static int c[N], d[N];
  int m = (n + 1) / 2;
  inv(a, b, m);
  init(n + m), cpy(c, b, m), cpy(d, a, n);
  DFT(c), DFT(d);
  for(int i = 0;i < lim;++i) c[i] = (u64) c[i] * c[i] % mod * d[i] % mod;
  IDFT(c);
  for(int i = m;i < n;++i) b[i] = norm(mod - c[i]);
}
void log(int * a, int * b, int n) { // b = log(a) (mod x^n)
  static int c[N], d[N];
  inv(a, c, n), init(n + n);
  for(int i = 1;i < n;++i) d[i - 1] = (u64) a[i] * i % mod;
  cpy(d, d, n - 1), cpy(c, c, n);
  DFT(c), DFT(d);
  for(int i = 0;i < lim;++i) c[i] = (u64) c[i] * d[i] % mod;
  IDFT(c), *b = 0;
  for(int i = 1;i < n;++i) b[i] = pow(i, mod - 2, c[i - 1]);
}
```

3.6 全在线卷积

```
struct oc {
  | std::vector<int> f, g, res;
  | std::vector<std::vector<int>> fa, fb;
  | int n, p;
  | oc(int n) : f(n), g(n), res(n), n(n), p(0) { }
  | void push(int v0, int v1) {
  |   | f[p] = v0;
  |   | res[p] = (res[p] + (u64) f[0] * v1 + (u64) g[0] * v0) % mod;
  |   | g[p++] = v1;
  |   | static int A[N], B[N];
  |   | int lb = p & -p;
  |   | init(lb * 2);
  |   | memset(A, 0, lim << 2);
  |   | memset(B, 0, lim << 2);
  |   | for(int i = 0;i < lb;++i) A[i] = g[p - lb + i], B[i] = f[p - lb + i];
  |   | DFT(A), DFT(B);
  |   | if(lb == p) {
  |   |   | fa.emplace_back(A, A + lim);
  |   |   | fb.emplace_back(B, B + lim);
  |   |   | for(int i = 0;i < lim;++i) A[i] = (u64) A[i] * B[i] % mod;
  |   | } else {
  |   |   | auto & C = fb[std::__lg(lim)], & D = fa[std::__lg(lim)];
  |   |   | for(int i = 0;i < lim;++i) A[i] = ((u64) A[i] * C[i * 2] + (u64)
  |   |     B[i] * D[i * 2]) % mod;
  |   | }
  |   | IDFT(A);
  |   | for(int j = p;j < p + lb && j < n;++j) res[j] = (res[j] + A[j - p +
  |     lb]) % mod;
  | }
};
struct Exp : oc {
  | std::vector<int> res;
  | Exp(int n) : oc(n), res(n) { }
  | void push(int v) {
  |   | if(!res[0]) return void(res[0] = 1);
  |   | oc::push(res[p], v * u64(p + 1) % mod);
  |   | res[p] = (u64) oc::res[p - 1] * inv[p] % mod;
  | }
};
struct Ln : oc {
  | std::vector<int> res; int fi;
  | Ln(int n) : oc(n), res(n), fi(0) {}
  | void push(int v) {
  |   | if(!fi) return void(fi = 1);
  |   | oc::push(res[p] * (u64) p % mod, v);
  |   | res[p] = ((u64) v * p + mod - oc::res[p - 1]) % mod * inv[p] % mod;
  | }
};
struct Inv : oc {
  | std::vector<int> res; int fi;
  | Inv(int n) : oc(n), res(n), fi(0) {}
  | void push(int v) {
  |   | res[p] = fi ? (oc::res[p] + (u64) v * res[0]) % mod * (mod - res[0]) %
  |     mod : pow(fi = v, mod - 2);
  |   | oc::push(res[p], v);
  | }
};
```

3.7 常系数线性递推

3.7.1 Berlekamp Massey

返回最短的 c 使得：$\forall n \geq size(c) : a_n = \sum_{i=0}^{size(c)-1} c_i a_{n-1-i}$

```
std::vector<int> BM(std::vector<int> a) {
  | int n = size(a), len = 0, s = 1, m = 0;
  | std::vector<int> res(n), lst(n), tmp;
  | res[0] = 1;
  | for(int i = 0;i < n;++i) {
  |   | int d = 0; ++m;
  |   | for(int j = 0;j <= len;++j)
  |   |   | d = (d + (u64) res[j] * a[i - j]) % mod;
  |   | if(!d) continue;
  |   | tmp = res;
  |   | u64 k = pow(s, mod - 2, mod - d);
  |   | for(int j = m;j < n;++j) res[j] = (res[j] + k * lst[j - m]) % mod;
  |   | if(len * 2 > i) continue;
  |   | len = i - len + 1, lst = tmp, s = d, m = 0;
  | }
  | std::vector<int> c(len);
  | for(int i = 0;i < len;++i) c[i] = (mod - res[i + 1]) % mod;
  | return c;
}
```

3.7.2 递推式转分式

$$a_i = \sum_{j=1}^{d} c_j a_{i-j}, i \geq d \Leftrightarrow a_i = [x^i] \frac{p(x)}{q(x)}$$

```
std::pair<fps, fps> convert(fps a, fps c) {
  | int d = c.size();
  | fps q(d + 1); q[0] = 1;
  | for(int i = 0;i < d;++i) q[i + 1] = norm(mod - c[i]);
  | for(;q.back() == 0;) q.pop_back();
  | fps p = conv(a, q);
  | p.resize(a.size());
  | return {p, q};
}
```

3.7.3 Bostan-Mori

$$[x^k] \frac{p(x)}{q(x)}$$

```cpp
int bostan_mori(fps p, fps q, ll k) {
  int n = q.size();
  for(;k;k >>= 1) {
    auto nq = q;
    for(int i = 1;i < n;i += 2) nq[i] = mod - nq[i];
    auto u = conv(p, nq), v = conv(q, nq);
    int t = k & 1, m = (size(u) + 1 - t) / 2;
    p.resize(m);
    for(int i = 0;i < m;++i) p[i] = u[i * 2 + t];
    for(int i = 0;i < n;++i) q[i] = v[i * 2];
  }
  return pow(q[0], mod - 2, p[0]);
}
```

3.7.4 远处区间值

$$\left[x^{[k,k+m)}\right]\frac{p(x)}{q(x)}$$

如果 $deg(p) \geq deg(q)$, 请使用 solve_x，不然可以直接使用 solve。

```cpp
fps inv(fps a, int n) {
  static int res[N];
  if(a.size() < n) a.resize(n);
  inv(a.data(), res, n);
  return fps(res, res + n);
}
fps cut(const fps & f, int d) {
  return fps(f.begin(), f.begin() + std::min((int)f.size(), d));
}
fps revp(const fps & f) { return fps(f.rbegin(), f.rend()); }
std::pair<fps, fps> polydiv(fps f, fps g) {
  int n = size(f), gd = size(g) - 1, qd = n - gd;
  if (n <= gd) return {fps(), f};
  auto F = cut(revp(f), qd);
  auto G = inv(revp(g), qd);
  fps q = revp(cut(conv(F, G), qd));
  g = conv(g, q);
  for(int i = 0;i < gd;++i) f[i] = (f[i] + mod - g[i]) % mod;
  return {q, cut(f, gd)};
}
fps solve(fps p, fps q, u64 k, int m) {
  const int d = size(q) - 1;
  auto rec = [&](auto rec, ll k) {
    if(k == 0) return d > 0 ? fps{1} : fps();
    if(k == 1) {
      int p = pow(q[0], mod - 2); fps res(d);
      for(int i = 0;i < d;++i) res[i] = u64(mod - p) * q[i + 1] % mod;
      return res;
    }
    fps f = rec(rec, (k + 1) / 2);
    f = conv(f, f);
    if(k & 1) f.insert(f.begin(), 0);
    return polydiv(f, q).second;
  };
  fps res = rec(rec, k);
  res = polydiv(conv(res, p), q).second;
  res = conv(res, inv(q, m));
  return fps(res.begin(), res.begin() + m);
}
fps solve_x(fps p, fps q, u64 k, int m) {
  auto [a, p0] = polydiv(p, q);
  auto ans = p0.size() ? solve(p0, q, k, m) : fps(m);
  for(int i = 0;i < m;++i) {
    if(k + i < (int) a.size()) ans[i] = (ans[i] + a[i + k]) % mod;
  }
  return ans;
}
```

3.8 线性规划 (单纯形法)

$x[1\cdots n]$ 是变量，$a[1\cdots m][\cdots]$ 是约束。

所有约束为：

$$\forall i \in [1,m] : a[i][0] + \sum_{j=1}^{n} a[i][j]x[j] \geq 0$$

$$\forall j \in [1,n] : x[j] \geq 0$$

最大化

$$\sum_{j=1}^{n} a[0][j]x[j]$$

如果无解返回 nan，如果无界返回 inf。最后的 x 里存放的是最优解。

```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;
  db a[M + N][N], x[N + M];
  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;
    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];
 }
}
```

3.9 黄金三分

返回 f 在 $[min(a,c), max(a,c)]$ 上的极大值点 (f 需要是单峰函数)。

```
db findmax(db a, db c, auto f) {
 auto g = [&](db l, db r) {
  return l + (r - l) * (std::numbers::phi_v<db> - 1);
 };
 db b = g(a, c), bv = f(b);
 for(int i = 0;i < 45;++i) {
  db x = g(a, b), xv = f(x);
  if(xv > bv) { // change here if findmin
   c = b, b = x, bv = xv;
  } else {
   a = c, c = x;
  }
 }
 return bv;
} // log_{1.618}(2) ≈ 1.44
```

4 字符串

4.1 后缀自动机 SAM

需要两倍点数量。

```
int c[N][26], mx[N], fail[N], tot = 1;
int append(int id, int w) {
 int p = id, now = ++tot;
 //right[now] = id;
 for(mx[now] = mx[p] + 1;p && !c[p][w];p = fail[p])
  c[p][w] = now;
 if(!p) fail[now] = 1;
 else {
  int q = c[p][w];
  if(mx[q] == mx[p] + 1) fail[now] = q;
  else {
   int x = ++tot; mx[x] = mx[p] + 1;
   memcpy(c[x], c[q], sizeof(c[0])), fail[x] = fail[q]; //right[x] = right[q];
   for(fail[q] = fail[now] = x;p && c[p][w] == q;p = fail[p])
    c[p][w] = x;
  }
 }
 return now;
}
void buildtree() { // 倒着建
 for(int i = 2;i <= tot;++i)
  son[fa[i]][s[right[i] + mx[fa[i]]] - 'a'] = i;
}
```

4.2 回文自动机 PAM

```
int c[N][26], fail[N], len[N], tot;
void init() {
 fail[0] = 1, len[++tot] = -1;
 // root is 1
}
int get_fail(int o, char * x) {
 for(;*x != x[-len[o] - 1];)
  o = fail[o];
 return o;
}
int append(int o, char * x) {
 o = get_fail(o, x);
 int & p = c[o][*x - 'a'];
 if(!p) {
  fail[++tot] = c[get_fail(fail[o], x)][*x - 'a'];
  len[p = tot] = len[o] + 2;
 }
 return p;
}
```

4.3 AC 自动机

```
const int sig = 26;
int son[N][sig], fail[N], cnt;
int ins(const char * c) {
 int x = 0;
```

```
  | for(;*c;++c) {
  |   | int & p = son[x][*c - 'a'];
  |   | if(!p) p = ++ cnt;
  |   | x = p;
  | }
  | return x;
}
void build_ac() {
  | std::queue<int> Q;
  | for(int i = 0;i < sig;++i) if(son[0][i]) Q.push(son[0][i]);
  | for(;Q.size();) {
  |   | int x = Q.front(); Q.pop();
  |   | for(int i = 0;i < sig;++i)
  |   |   | if(son[x][i]) fail[son[x][i]] = son[fail[x]][i], Q.push(son[x][i]);
  |   |   | else son[x][i] = son[fail[x]][i];
  | }
}
```

4.4 SA

后缀数组，sa 是个 $0 \sim n-1$ 的排列，$h[i] = lcp(s[sa[i]:], s[sa[i+1]:])$。

```
auto SA(const std::vector<int> & s) {
  | int n = s.size();
  | std::vector<int> sa(n), gap(n), rk(n), h(n - 1);
  | std::vector<std::array<int, 2>> o(n);
  | gap[0] = 1;
  | for(int i = 0;i < n;++i) o[i] = {s[i], i};
  | for(int L = 1;;L *= 2) {
  |   | for(int p = n, i = n - 1;i >= 0;--i) if(gap[i]) {
  |   |   | sort(o.begin() + i, o.begin() + p, [](auto u, auto v) { return u[0]
  |   |   |   < v[0]; });
  |   |   | for(;--p > i;) gap[p] = o[p][0] != o[p - 1][0];
  |   | }
  |   | int s = 0;
  |   | for(int i = 0;i < n;++i) rk[o[i][1]] = s += gap[i];
  |   | if(s == n) break;
  |   | for(auto & [v, p] : o) v = p + L < n ? rk[p + L] : 0;
  | }
  | for(int i = 0;i < n;++i) sa[i] = o[i][1];
  | for(int i = 0, k = 0;i < n;++i) if(rk[i] != n) {
  |   | int j = sa[rk[i]];
  |   | for(k -= !!k;i + k < n && j + k < n && s[i + k] == s[j + k];++k);
  |   | h[rk[i] - 1] = k;
  | }
  | return std::pair(sa, h);
}
```

4.5 Z algorithm

返回的 $lcp[i] = lcp(s, s[i:])$。

```
std::vector<int> Zalgo(const std::string & s) {
  | std::vector<int> lcp(s.size());
  | lcp[0] = s.size();
  | for(int i = 1, l = 0, r = 0;i < (int) s.size();++i) {
  |   | int & x = lcp[i];
  |   | if(i < r) x = std::min(lcp[i - l], r - i);
  |   | for(;i + x < (int) s.size() && s[x] == s[i + x];)
  |   |   | ++ x;
  |   | if(i + x > r) l = i, r = i + x;
  | }
  | return lcp;
}
```

4.6 Manacher

返回每个回文中心的回文半径大小，奇偶分开考虑。

```
std::vector<int> manacher(const std::string & s, bool is_even) {
  | const int N = s.size() - is_even;
  | std::vector<int> o(N); // 半径
  | for(int i = 0, mid = 0, r = 0;i < N;++i) {
  |   | int & x = o[i] = 0;
  |   | if(i < r) x = std::min(r - i, o[mid - i]);
  |   | for(;i - x >= 0 && i + x < N && s[i - x] == s[i + x + is_even];)
  |   |   | ++ x;
  |   | if(i + x > r) {
  |   |   | mid = i * 2;
  |   |   | r = i + x;
  |   | }
  | }
  | return o;
}
```

4.7 最小表示法

返回 s 的最小表示的起始位置。可以用 rotate 算出最小表示的字符串。

```
int minrep(const std::vector<int> & s) {
  | int k = 0, i = 0, j = 1, n = size(s);
  | for(;k < n && i < n && j < n;) {
  |   | if(int d = s[(i + k) % n] - s[(j + k) % n]) {
  |   |   | (d > 0 ? i : j) += k + 1;
  |   |   | i += i == j, k = 0;
  |   | } else {
  |   |   | ++ k;
  |   | }
  | }
  | return std::min(i, j);
} // rotate(s.begin(), s.begin() + minrep(s), s.end());
```

4.8 Lyndon 分解

```
auto duval(const std::vector<int> & s) {
```

```
  | int n = s.size();
  | std::vector<std::vector<int>> res;
  | for(int i = 0;i < n;) {
  |   | int j = i + 1, k = i;
  |   | for(;j < n && s[k] <= s[j];++j) {
  |   |   | if(s[k] < s[j]) {
  |   |   |   | k = i;
  |   |   | } else {
  |   |   |   | ++ k;
  |   |   | }
  |   | }
  |   | for(;i <= k;i += j - k) {
  |   |   | res.emplace_back(s.begin() + i, s.begin() + i + (j - k));
  |   | }
  | }
  | return res;
}
```

5 数据结构

5.1 区间加区间求和树状数组

add 区间加，query 区间求和。区间都是闭区间。

```
// 后缀加，前缀求和
struct BIT {
  | ll a[N], b[N];
  | void add(ll p, int v) {
  |   | for(int i = p;i < N;i += i & -i)
  |   |   | a[i] += v, b[i] += p * v;
  | }
  | ll qry(ll p) {
  |   | ll res = 0;
  |   | for(int i = p;i;i &= i - 1) res += (p + 1) * a[i] - b[i];
  |   | return res;
  | }
  | void add(int l, int r, int v) { add(l, v), add(r + 1, -v); }
  | ll qry(int l, int r) { return qry(r) - qry(l - 1); }
} bit;
```

5.2 zkw 线段树

单点修改区间半群。如果半群没有单位元可以用 qry2。

```
struct seg {
  | ll o[1 << 20]; int L;
  | void upt(int x) {
  |   | o[x] = o[x << 1] + o[x << 1 | 1];
  | }
  | void init(int n, int * w) {
  |   | L = 2 << std::__lg(n + 1);
  |   | for(int i = 1;i <= n;++i) o[i + L] = w[i];
  |   | for(int i = L;i >= 1;--i) upt(i);
  | }
  | void upt(int p, int v) {
  |   | for(o[p += L] += v;p >>= 1;upt(p));
  | }
  | ll qry(int l, int r) {
  |   | l += L - 1, r += L + 1;
  |   | ll ans = 0;
  |   | for(;l ^ r ^ 1;l >>= 1, r >>= 1) {
  |   |   | if((l & 1) == 0) ans += o[l ^ 1];
  |   |   | if((r & 1) == 1) ans += o[r ^ 1];
  |   | }
  |   | return ans;
  | }
  | ll qry2(int l, int r) {
  |   | if(l == r) return o[l + L];
  |   | ll le = o[l + L], ri = o[r + L];
  |   | l += L, r += L;
  |   | for(;l ^ r ^ 1;l >>= 1, r >>= 1) {
  |   |   | if((l & 1) == 0) le = le + o[l ^ 1];
  |   |   | if((r & 1) == 1) ri = o[r ^ 1] + ri;
  |   | }
  |   | return le + ri;
  | }
} sgt;
```

5.3 静态区间半群

$O(n \log n)$ 预处理，$O(1)$ 查询。

```
template<class T> struct rq {
  | std::vector<std::vector<info>> v; // change type here
  | T op;
  | rq(auto & a, T op) : v(1, a), op(op) {
  |   | int n = a.size();
  |   | for(int d = 2;d < n;d += d) {
  |   |   | auto & o = v.emplace_back(a);
  |   |   | for(int j = d;j < n;j += d + d) {
  |   |   |   | for(int k = 1;k < d;++k) {
  |   |   |   |   | if(j + k < n) o[j + k] = op(o[j + k - 1], o[j + k]);
  |   |   |   |   | o[j - k - 1] = op(o[j - k - 1], o[j - k]);
  |   |   |   | }
  |   |   | }
  |   | }
  | }
  | auto query(int l, int r) {
  |   | if(l == r) return v[0][l];
  |   | const int lg = std::__lg(l ^ r);
  |   | return op(v[lg][l], v[lg][r]);
  | }
};
```

5.4 Link Cut Tree

```
int son[N][2], fa[N], rev[N];
```

```
int get(int x, int p = 1) { return son[fa[x]][p] == x; }
void update(int x) { }
int is_root(int x) { return !(get(x) || get(x, 0)); }
void rotate(int x) {
  int y = fa[x], z = fa[y], b = get(x);
  if(!is_root(y)) son[z][get(y)] = x;
  son[y][b] = son[x][!b], son[x][!b] = y;
  fa[son[y][b]] = y, fa[y] = x, fa[x] = z;
  update(y);
}
void put(int x) {
  if(x) rev[x] ^= 1, std::swap(son[x][0], son[x][1]);
}
void down(int x) {
  if(rev[x]) {
    put(son[x][0]);
    put(son[x][1]);
    rev[x] = 0;
  }
}
void pushdown(int x) {
  if(!is_root(x)) pushdown(fa[x]);
  down(x);
}
void splay(int x) {
  for(pushdown(x);!is_root(x);rotate(x)) if(!is_root(fa[x]))
    rotate(get(x) ^ get(fa[x]) ? x : fa[x]);
  update(x);
}
void access(int x) {
  for(int t = 0;x;son[x][1] = t, t = x, x= fa[x])
    splay(x);
}
void makeroot(int x) {
  access(x), splay(x), put(x);
}
```

5.5 压位 Trie

```
int ctz(u64 x) { return __builtin_ctzll(x); }
int clz(u64 x) { return __builtin_clzll(x); }
struct ds {
  std::vector<std::vector<u64>> o;
  int d;
  ds(int s) { // [0, s]
    do o.emplace_back((s >>= 6) + 1); while(s);
    d = o.size();
  }
  void ins(int x) {
    for(int i = 0;i < d;++i) {
      u64 w = 1ull << (x & 63), & v = o[i][x >>= 6];
      if(v & w) return ;
      v |= w;
    }
  }
  void del(int x) {
    for(int i = 0;i < d;++i) {
      u64 w = 1ull << (x & 63), & v = o[i][x >>= 6];
      if(v &= ~w) return ;
    }
  }
  int next(int x) const {
    for(int i = 0;i < d;++i) {
      int b = x & 63;
      if(u64 v = o[i][x >>= 6] >> b >> 1) {
        int ans = x * 64 + ctz(v) + b + 1;
        for(int j = i - 1;j >= 0;--j)
          ans = ans * 64 + ctz(o[j][ans]);
        return ans;
      }
    }
    return -1;
  }
  int prev(int x) const {
    for(int i = 0;i < d;++i) {
      int b = x & 63;
      if(u64 v = o[i][x >>= 6] & ((1ull << b) - 1)) {
        int ans = x * 64 + (63 - clz(v));
        for(int j = i - 1;j >= 0;--j) {
          ans = ans * 64 + (63 - clz(o[j][ans]));
        }
        return ans;
      }
    }
    return -1;
  }
  int min() const {
    int ans = 0;
    for(int j = d - 1;j >= 0;--j) ans = ans * 64 + ctz(o[j][ans]);
    return ans;
  }
  int max() const {
    int ans = 0;
    for(int j = d - 1;j >= 0;--j) ans = ans * 64 + (63 - clz(o[j][ans]));
    return ans;
  }
};
```

5.6 pbds tree

```
#include <bits/extc++.h>
using namespace __gnu_pbds;
template<class T> // insert, erase, join, order_of_key, find_by_order(return iterator), order is 0-index
using Tree = tree<T, null_type, std::less<T>, rb_tree_tag, tree_order_statistics_node_update>;
```

6 geometry

6.1 向量

```cpp
using db = long double;
const db eps = 1e-10;
db sgn(db x) { return x < -eps ? -1 : x > eps; }
db eq(db x, db y) { return !sgn(x - y); }
struct p2 {
  db x, y;
  db norm() const { return x * x + y * y; }
  db abs() const { return std::sqrt(x * x + y * y); }
  db arg() const { return atan2(y, x); }
};
db arg(p2 x, p2 y) {
  db a = y.arg() - x.arg();
  if(a > pi) a -= pi * 2;
  if(a < -pi) a += pi * 2;
  return a;
}
p2 r90(p2 x) { return {-x.y, x.x}; }
p2 operator + (p2 x, p2 y) { return {x.x + y.x, x.y + y.y}; }
p2 operator - (p2 x, p2 y) { return {x.x - y.x, x.y - y.y}; }
p2 operator / (p2 x, db y) { return {x.x / y, x.y / y}; }
p2 operator * (p2 x, db y) { return {x.x * y, x.y * y}; }
p2 operator * (db y, p2 x) { return {x.x * y, x.y * y}; }
db operator * (p2 x, p2 y) { return x.x * y.y - x.y * y.x; }
db operator % (p2 x, p2 y) { return x.x * y.x + x.y * y.y; }
int half(p2 x){return x.y < 0 || (x.y == 0 && x.x <= 0); }
int half(p2 x){return x.y < -eps || (std::fabs(x.y) < eps && x.x < eps);}
bool cmp(p2 a, p2 b) { return half(a) == half(b) ? a * b > 0 : half(b); }
bool cmp_eq(p2 A, p2 B) { return half(A) == half(B) && eq(A * B, 0) ; }
// 判断 A, B, C 三个向量是否是逆时针顺序
// 如果是, 返回 1
// 如果 (A, B), (C, B) 同方向共线, 返回 -1
// 如果是顺时针, 返回 0
bool cmp_ct(p2 A, p2 B, p2 C) {
  if(cmp_eq(A, B)) return -1;
  if(cmp_eq(C, B)) return -1;
  if(cmp(A, B)) {
    return cmp(B, C) || cmp(C, A);
  } else {
    return cmp(B, C) && cmp(C, A);
  }
}
// 凸包 DP
struct pr { int i, j; p2 get() const { return a[j] - a[i]; } };
bool cmpseg(pr x, pr y) {
  p2 A = x.get(), B = y.get();
  if(!cmp(A, B) && !cmp(B, A)) return a[x.i] % A < a[y.i] % A;
  return cmp(A, B);
}
```

6.2 三角形各心

```cpp
p2 bary(p2 a, p2 b, p2 c, db A, db B, db C) {
  return (a * A + b * B + c * C) / (A + B + C);
}
p2 centroid(p2 A, p2 B, p2 C) {
  return bary(A, B, C, 1, 1, 1);
} // 重心
p2 circumcenter(p2 A, p2 B, p2 C) {
  db a = (B - C).norm(), b = (C - A).norm(), c = (A - B).norm();
  return bary(A, B, C, a*(b+c-a), b*(c+a-b), c*(a+b-c));
} // 外心，外接圆圆心，三边中垂线的交点
p2 incenter(p2 A, p2 B, p2 C) {
  return bary(A, B, C, (B-C).abs(), (A-C).abs(), (A-B).abs());
} // 内心，内接圆圆心，三角角平分线的交点
p2 orthocenter(p2 A, p2 B, p2 C) {
  db a = (B - C).norm(), b = (C - A).norm(), c = (A - B).norm();
  return bary(A, B, C, (a+b-c)*(c+a-b), (b+c-a)*(a+b-c), (c+a-b)*(b+c-a));
} // 垂心，三条高线所在的交点
p2 excenter(p2 A, p2 B, p2 C) {
  db a = (B - C).abs(), b = (A - C).abs(), c = (A - B).abs();
  return bary(A, B, C, -a, b, c);
  // return bary(A, B, C, a, -b, c);
  // return bary(A, B, C, a, b, -c);
} // 旁心 一个内角的平分线和其他两个内角的外角平分线的交点
```

6.3 直线半平面

```cpp
struct line : p2 {
  db z;
  // a * x + b * y + c (= or >) 0
  line() = default;
  line(db a, db b, db c) : p2{a, b}, z(c) {}
  line(p2 a, p2 b) : p2(r90(b - a)), z(a * b) { } //左侧 > 0
  db operator ()(p2 a) const { return a % p2(*this) + z; }
  line perp() const { return {y, -x, 0}; } // 垂直
  line para(p2 o) { return {x, y, z - (*this)(o)}; } // 平行
};
p2 operator & (line x, line y) {
  return p2{p2{x.z, x.y} * p2{y.z, y.y}, p2{x.x, x.z} * p2{y.x, y.z}} /
    -(p2(x) * p2(y));
  // 注意此处精度误差较大，以及 res.y 需要较高精度
}
p2 proj(p2 x, line l){return x - p2(l) * (l(x) / l.norm());}//投影
p2 refl(p2 x, line l){return x - p2(l) * (l(x) / l.norm()) * 2;}//对称
db dist(line l, p2 x={0, 0}){return l(x) / l.abs();}//有向点到线距离
bool is_para(line x, line y){return eq(p2(x) * p2(y), 0);}//判断线平行
bool is_perp(line x, line y){return eq(p2(x) % p2(y), 0);}//判断线垂直
bool online(p2 x, line l) { return eq(l(x), 0); } // 判断点在线上
int ccw(p2 a, p2 b, p2 c) {
  int sign = sgn((b - a) * (c - a));
  if(sign == 0) {
    if(sgn((b - a) % (c - a)) == -1) return 2;
    if((c - a).norm() > (b - a).norm() + eps) return -2;
  }
```

```
  | return sign;
}
db det(line a, line b, line c) {
  | p2 A = a, B = b, C = c;
  | return c.z * (A * B) + a.z * (B * C) + b.z * (C * A);
}
db check(line a, line b, line c) { // sgn same as c(a & b), 0 if error
  | return sgn(det(a, b, c)) * sgn(p2(a) * p2(b));
}
bool paraS(line a, line b) { // 射线同向
  | return is_para(a, b) && p2(a) % p2(b) > 0;
}
```

6.4 半平面交

```
std::vector<p2> HPI(std::vector<line> vs) {
  | auto cmp = [](line a, line b) {
  |   | if(paraS(a, b)) return dist(a) < dist(b);
  |   | return ::cmp(p2(a), p2(b));
  | };
  | sort(vs.begin(), vs.end(), cmp);
  | int ah = 0, at = 0, n = size(vs);
  | std::vector<line> deq(n + 1);
  | std::vector<p2> ans(n);
  | deq[0] = vs[0];
  | for(int i = 1;i <= n;++i) {
  |   | line o = i < n ? vs[i] : deq[ah];
  |   | if(paraS(vs[i - 1], o)) continue;
  |   | for(;ah < at && check(deq[at - 1], deq[at], o) < 0;) -- at;//maybe <=
  |   | if(i != n) for(;ah < at && check(deq[ah], deq[ah + 1], o) < 0;) ++ ah;
  |   | if(!is_para(o, deq[at])) {
  |   |   | ans[at] = o & deq[at];
  |   |   | deq[++at] = o;
  |   | }
  | }
  | if(at - ah <= 2) return {};
  | return {ans.begin() + ah, ans.begin() + at};
}
```

6.5 线段

```
struct seg {
  | p2 x, y;
  | seg() {}
  | seg(const p2 & A, const p2 & B) : x(A), y(B) {}
  | bool onseg(const p2 & o) const {
  |   | return (o - x) % (o - y) < eps && std::fabs((o - x) * (o - y)) < eps;
  | }
};
db dist(const seg & o, const p2 & x) {
  | if((o.x - o.y) % (x - o.y) <= eps) return (x - o.y).abs();
  | if((o.y - o.x) % (x - o.x) <= eps) return (x - o.x).abs();
  | return fabs((o.x - x) * (o.y - x) / (o.x - o.y).abs());
}
```

```
bool is_isc(const seg & x, const seg & y) {
  | return
  |   | ccw(x.x, x.y, y.x) * ccw(x.x, x.y, y.y) <= 0 &&
  |   | ccw(y.x, y.y, x.x) * ccw(y.x, y.y, x.y) <= 0;
}
db dist(const seg & x, const seg & y) {
  | if(is_isc(x, y)) return 0;
  | return std::min({dist(y, x.x), dist(y, x.y), dist(x, y.x), dist(x, y.y)});
}
```

6.6 简单多边形

```
using polygon = std::vector<p2>;
// counter-clockwise
db area(const polygon & x) {
  | db res = 0;
  | for(int i = 2;i < (int) x.size();++i) {
  |   | res += (x[i - 1] - x[0]) * (x[i] - x[0]);
  | }
  | return res / 2;
}
bool is_convex(const polygon & x, bool strict = 1) {
  | // warning, maybe wrong
  | const db z = strict ? eps : -eps;
  | for(int i = 2;i < (int) x.size() + 2;++i) {
  |   | if((x[(i - 1) % x.size()] - x[i - 2]) * (x[i % x.size()] - x[i - 2]) <
  |     z) return 0;
  | }
  | return 1;
}
int contain(const std::vector<p2> & a, p2 o) { // 简单多边形包含判定
  | bool in = 0;
  | for(int i = 0;i < (int) a.size();++i) {
  |   | p2 x = a[i] - o, y = a[(i + 1) % a.size()] - o;
  |   | if(x.y > y.y) std::swap(x, y);
  |   | if(x.y <= eps && y.y > eps && x * y < -eps) in ^= 1;
  |   | if(std::fabs(x * y) < eps && x % y < eps) return 2; // 在线段上，看情况改
  | }
  | return in;
}
```

6.7 简单多边形三角剖分

```
std::vector<std::array<int, 3>> triangulate(std::vector<p2> a) {
  | int n = a.size();
  | std::vector<int> prev(n), next(n), ear(n);
  | auto in = [](p2 p, p2 a, p2 b, p2 c) {
  |   | return area(p, a, b) >= -eps && area(p, b, c) >= -eps && area(p, c, a)
  |     >= -eps;
  | };
  | auto is_ear = [&](int x) {
  |   | if(area(a[prev[x]], a[x], a[next[x]]) <= eps) return 0;
  |   | for(int i = next[next[x]];i != prev[x];i = next[i])
  |   |   | if(in(a[i], a[prev[x]], a[x], a[next[x]]))
```

```
    | | | | return 0;
    | | return 1;
    | };
    | for(int i = 0;i < n;++i) {
    | | next[i] = (i + 1) % n, prev[(i + 1) % n] = i;
    | }
    | for(int i = 0;i < n;++i) {
    | | ear[i] = is_ear(i);
    | }
    | std::vector<std::array<int, 3>> ans(n - 2);
    | for(int ec = 0, cur = 0;ec < n - 2;++ec) {
    | | for(;!ear[cur];) cur = next[cur];
    | | int a = prev[cur], b = cur, c = next[cur];
    | | ans[ec] = {a, b, c};
    | | next[a] = c, prev[c] = a;
    | | ear[a] = is_ear(a), ear[c] = is_ear(c);
    | | cur = next[cur];
    | }
    | return ans;
} // 输入逆时针
```

6.8 线段 in 多边形

```
bool contains(p2 x, p2 y, const std::vector<p2> & a) {
    | using pr = std::pair<double, int>;
    | std::vector<pr> e = {pr(-inf, 0), pr(inf, 0)};
    | p2 t = y - x;
    | auto f = [&](p2 a, p2 b, p2 c, p2 d) {
    | | return (b - a).abs() * ((c - a) * (d - c)) / ((b - a) * (d - c));
    | };
    | for(int i = 0;i < (int) a.size();++i) {
    | | p2 u = a[i], v = a[(i + 1) % a.size()];
    | | int a = sgn(t * (u - x));
    | | int b = sgn(t * (v - x));
    | | if(a != b) e.emplace_back(f(x, y, u, v), b - a);
    | }
    | sort(e.begin(), e.end());
    | int sum = 0; db R = t.abs();
    | for(int i = 0;i + 1 < (int) e.size();++i) {
    | | sum += e[i].second;
    | | if(sum == 0 && std::max(e[i].first, 0.) + eps < std::min(e[i +
      1].first, R)) {
    | | | return 0;
    | | }
    | }
    | return 1;
}
```

6.9 图形求交

```
struct circle : p2 { db r; };
std::vector<p2> operator & (circle o, line l) {
    | p2 v = l, Rv = r90(v); db L = l.abs();
    | db d = l(p2(o)) / L, x = o.r * o.r - d * d;
    | if(x < -eps) return {};
    | x = std::sqrt(x * sgn(x));
    | p2 z = p2(o) - v * (d / L), p = Rv * (x / L);
    | return {z + p, z - p};
} // l 如果是构造函数给出，那么返回交点按射线顺序
std::vector<p2> operator & (circle o, seg s) {
    | std::vector<p2> b;
    | for(p2 x : (o & s.to_l()))
    | | if(s.onseg(x)) b.push_back(x);
    | return b;
}
std::vector<p2> operator & (circle o0, circle o1) {
    | p2 tmp = (p2(o1) - p2(o0)) * 2.;
    | return o0 & line(tmp.x, tmp.y, o1.r * o1.r - o0.r * o0.r + o0.norm() -
      o1.norm());
}
std::vector<p2> tang(circle o, p2 x) {
    | db d = (x - p2(o)).abs();
    | if(d <= o.r + eps) return {};
    | return o & circle{x, sqrt(d * d - o.r * o.r)};
}
// 三角形 (o, a, b) 和圆 o 的交的有向面积 * 2
db intersect(circle o, p2 a, p2 b) {
    | a = a - p2(o), b = b - p2(o); o.x = o.y = 0;
    | int va = a.abs() <= o.r + eps;
    | int vb = b.abs() <= o.r + eps;
    | if(va && vb) return a * b;
    | auto v = o & seg{a, b}; // 注意这里，有必要改一下 onseg，去掉平行判定
    | if(v.empty()) return arg(a, b) * o.r * o.r;
    | db sum = 0;
    | sum += va ? a * v[0] : arg(a, v[0]) * o.r * o.r;
    | sum += vb ? v.back() * b : arg(v.back(), b) * o.r * o.r;
    | if(v.size() > 1) sum += v[0] * v[1];
    | return sum;
}
// 有向弓形面积 * 2, arg 不能改
db csegS(circle o, p2 a, p2 b) {
    | a = a - p2(o);
    | b = b - p2(o);
    | db d = b.arg() - a.arg();
    | if(d < 0) d += pi * 2;
    | return d * o.r * o.r - a * b;
}
// 两圆交的面积 * 2
db intersect(circle o0, circle o1) {
    | if(o0.r > o1.r) std::swap(o0, o1);
    | db d = (p2(o0) - p2(o1)).abs();
    | if(d <= (o1.r - o0.r) + eps) return 2 * pi * o0.r * o0.r;
    | if(d >= o1.r + o0.r - eps) return 0;
    | auto v = o0 & o1;
    | return csegS(o0, v[1], v[0]) + csegS(o1, v[0], v[1]);
}
```

6.10 凸包

结果为逆时针。

```
db cross(p2 x, p2 y, p2 z) { return (y.x - x.x) * (z.y - x.y) - (y.y - x.y) * (z.x - x.x); }
std::vector<p2> gethull(std::vector<p2> o) {
  rgs::sort(o, [](p2 x, p2 y) { return eq(x.x, y.x) ? x.y < y.y : x.x < y.x; });
  o.erase(unique(o.begin(), o.end(), [](p2 x, p2 y) {
    return eq(x.x, y.x) && eq(x.y, y.y);
  }), o.end());
  std::vector<p2> s;
  for(int i = 0;i < (int) o.size();++i) {
    for(;s.size() >= 2 && cross(s.rbegin()[1], s.back(), o[i]) <= eps;)
      s.pop_back();
    s.push_back(o[i]);
  }
  for(int i = o.size() - 2, t = s.size();i >= 0;--i) {
    for(;s.size() > t && cross(s.rbegin()[1], s.back(), o[i]) <= eps;)
      s.pop_back();
    s.push_back(o[i]);
  }
  if(s.size() > 1) s.pop_back();
  return s;
} // 把两个 eps 改成 -eps 可求出所有在凸包上的点
int findmin(std::vector<p2> & a, auto cmp) {
  int l = 0, r = a.size() - 1, d = 1;
  if(cmp(a.back(), a[0])) std::swap(l, r), d = -1;
  for(;(r - l) * d > 1;) {
    int mid = (l + r) >> 1;
    if(cmp(a[mid], a[mid - d]) && cmp(a[mid], a[l])) {
      l = mid;
    } else {
      r = mid;
    }
  }
  return l;
} // cmp is less, and a.size()>0 plz
int contains(std::vector<p2> & a, p2 x) {
  auto it = lower_bound(a.begin() + 2, a.end(), x, [&](p2 x, p2 y) {
    return cross(a[0], x, y) > 0;
  });
  ll c0 = cross(it[-1], *it, x), c1 = cross(a[0], a[1], x);
  if(it != a.end() && c0 >= 0 && c1 >= 0) {
    return c0 > 0 && c1 > 0 && cross(a.back(), a[0], x) > 0 ? IN : ON;
  } else {
    return 0;
  }
} // a.size()>2 plz
```

6.11 上凸壳

结果显然为顺时针。

```
std::vector<p2> gethull(std::vector<p2> o) {
  sort(o.begin(), o.end(), [](p2 x, p2 y) {
    if(x.x == y.x) {
      return x.y > y.y; // gt ⇒ lt
    } else {
      return x.x < y.x;
    }
  });
  std::vector<p2> stack;
  for(p2 x : o) {
    if(stack.size() && stack.back().x == x.x) {
      continue;
    }
    for(;stack.size() >= 2 && cross(stack.rbegin()[1], stack.back(), x) >= 0;) { // gt ⇒ lt
      stack.pop_back();
    }
    stack.push_back(x);
  }
  return stack;
}
```

6.12 最小圆覆盖

```
struct circle : p2 { db r; };
circle incircle(p2 a, p2 b, p2 c) {
  db A = (b - c).abs(), B = (c - a).abs(), C = (a - b).abs();
  return {(a * A + b * B + c * C) / (A + B + C), fabs((b - a) * (c - a)) / (A + B + C)};
} // 三点确定内心，不是最小圆覆盖内容
circle circumcircle(p2 a, p2 b, p2 c) {
  p2 bc = c - b, ca = a - c, ab = b - a;
  p2 o = (b + c - r90(bc) * (ca % ab) / (ca * ab)) / 2;
  return {o, (a - o).abs()};
} // 三点确定外接圆
circle cir(p2 a, p2 b) { // 根据直径生成圆
  return {(a + b) / 2, (a - b).abs() / 2};
}
bool in(circle x, p2 y) { return (p2(x) - y).abs() <= x.r + eps; }
circle mincircle(std::vector<p2> a) { // 最小圆覆盖，需要 shuffle
  circle o = {a[0], 0};
  int n = a.size();
  for(int i = 1;i < n;++i) {
    if(in(o, a[i])) continue;
    o = cir(a[0], a[i]);
    for(int j = 1;j < i;++j) {
      if(in(o, a[j])) continue;
      o = cir(a[j], a[i]);
      for(int k = 0;k < j;++k) {
        if(in(o, a[k])) continue;
        o = circumcircle(a[i], a[j], a[k]);
      }
    }
  }
```

```
  | }
  | return o;
}
```

6.13 最近点对

```
db mindist(std::vector<p2> a) {
  | db ans = 1e18;
  | sort(a.begin(), a.end(), [](p2 x, p2 y) { return x.x < y.x; });
  | ans = (a[0] - a[1]).abs();
  | auto solve = [&](auto s, int l, int r) {
  |   | if(l + 1 == r) return ;
  |   | int mid = (l + r) >> 1;
  |   | db mx = a[mid].x;
  |   | s(s, l, mid), s(s, mid, r);
  |   | static std::vector<p2> b; b.clear();
  |   | inplace_merge(a.begin() + l, a.begin() + mid, a.begin() + r, [](p2 x,
      p2 y) { return x.y < y.y; });
  |   | for(int i = l;i < r;++i)
  |   |   | if(fabs(a[i].x - mx) <= ans) b.push_back(a[i]);
  |   | for(int i = 1;i < (int) b.size();++i)
  |   |   | for(int j = i - 1;j >= 0 && b[i].y <= b[j].y + ans;--j) ans =
        std::min(ans, (b[i] - b[j]).abs());
  | };
  | solve(solve, 0, a.size());
  | return ans;
}
```

6.14 凸包直径

```
db convex_diameter(std::vector<p2> & o) {
  | int n = size(o);
  | db max = 0;
  | for(int i = 0, j = 1 % n;i < j;++i) {
  |   | for(;;j = (j + 1) % n) {
  |   |   | max = std::max(max, (o[j] - o[i]).abs());
  |   |   | if((o[i + 1] - o[i]) * (o[(j + 1) % n] - o[j]) <= 0) break;
  |   | }
  | }
  | return max;
} // 凸包直径，不能有三点共线
```

6.15 切凸包

```
std::vector<p2> cut(const std::vector<p2> & o, line l) {
  | std::vector<p2> res;
  | int n = size(o);
  | for(int i = 0;i < n;++i) {
  |   | p2 a = o[i], b = o[(i + 1) % n];
  |   | if(sgn(l(a)) >= 0) res.push_back(a); // 注意 sgn 精度
  |   | if(sgn(l(a)) * sgn(l(b)) < 0) res.push_back(line(a, b) & l);
  | }
  | if(res.size() <= 2) return {};
  | return res;
} // 切凸包
```

6.16 V 图

```
std::vector<line> cut(const std::vector<line> & o, line l) {
  | std::vector<line> res;
  | int n = size(o);
  | for(int i = 0;i < n;++i) {
  |   | line a = o[i], b = o[(i + 1) % n], c = o[(i + 2) % n];
  |   | int va = check(a, b, l), vb = check(b, c, l);
  |   | if(va > 0 || vb > 0 || (va == 0 && vb == 0)) {
  |   |   | res.push_back(b);
  |   | }
  |   | if(va >= 0 && vb < 0) {
  |   |   | res.push_back(l);
  |   | }
  | }
  | if(res.size() <= 2) return {};
  | return res;
} // 切凸包
line bisector(p2 a, p2 b) { return line(a.x - b.x, a.y - b.y, (b.norm() -
a.norm()) / 2); }
line bisector(circle a, circle b) {
  | return line(a.x - b.x, a.y - b.y, (b.norm() - a.norm() + a.r * a.r - b.r *
    b.r) / 2);
} // a b 不能是包含关系, for power diagram

std::vector<std::vector<line>> voronoi(std::vector<p2> p) {
  | int n = p.size();
  | auto b = p; shuffle(b.begin(), b.end(), gen);
  | const db V = 1e5; // 边框大小, 重要
  | std::vector<std::vector<line>> a(n, {
  |   | {V, 0, V * V}, {0, V, V * V},
  |   | {-V, 0, V * V}, {0, -V, V * V},
  | });
  | for(int i = 0;i < n;++i) {
  |   | for(p2 x : b) if((x - p[i]).abs() > eps) {
  |   |   | a[i] = cut(a[i], bisector(p[i], x));
  |   | }
  | }
  | return a;
}
```

6.17 Delaunay 三角剖分

```
using i128 = __int128;
using Q = struct Quad*;
p2 arb(LLONG_MAX, LLONG_MAX);
struct Quad {
  | Q rot, o; p2 p = arb; bool mark;
  | p2& F() { return r() -> p; }
  | Q& r() { return rot->rot; }
  | Q prev() { return rot->o->rot; }
```

```
  | Q next() { return r()->prev(); }
} *H;
ll cross(p2 a, p2 b, p2 c) {
  | return (b - a) * (c - a);
}
bool circ(p2 p, p2 a, p2 b, p2 c) { // p 是否在 a, b, c 外接圆中
  | i128 p2 = p.norm(), A = a.norm() - p2, B = b.norm() - p2, C = c.norm() -
    p2;
  | a = a - p, b = b - p, c = c - p;
  | return (a * b) * C + (b * c) * A + (c * a) * B > 0;
}
Q link(p2 orig, p2 dest) {
  | Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  | H = r -> o; r -> r() -> r() = r;
  | for(int i = 0;i < 4;++i)
  |   | r = r -> rot, r -> p = arb, r -> o = i & 1 ? r : r -> r();
  | r -> p = orig, r -> F() = dest;
  | return r;
}
void splice(Q a, Q b) {
  | std::swap(a -> o -> rot -> o, b -> o -> rot -> o);
  | std::swap(a -> o, b -> o);
}
Q conn(Q a, Q b) {
  | Q q = link(a -> F(), b -> p);
  | splice(q, a -> next());
  | splice(q -> r(), b);
  | return q;
}
std::pair<Q, Q> rec(const std::vector<p2> & s) {
  | int N = size(s);
  | if(N <= 3) {
  |   | Q a = link(s[0], s[1]), b = link(s[1], s.back());
  |   | if(N == 2) return {a, a -> r()};
  |   | splice(a -> r(), b);
  |   | ll side = cross(s[0], s[1], s[2]);
  |   | Q c = side ? conn(b, a) : 0;
  |   | return {side < 0 ? c->r() : a, side < 0 ? c : b -> r() };
  | }
#define H(e) e -> F(), e -> p
#define valid(e) (cross(e->F(), H(base)) > 0)
  | int half = N / 2;
  | auto [ra, A] = rec({s.begin(), s.end() - half});
  | auto [B, rb] = rec({s.end() - half, s.end()});
  | while((cross(B -> p, H(A)) < 0 && (A = A -> next())) ||
  |   |   | (cross(A -> p, H(B)) > 0 && (B = B -> r() -> o)));
  | Q base = conn(B -> r(), A);
  | if(A -> p == ra -> p) ra = base -> r();
  | if(B -> p == rb -> p) rb = base;
#define DEL(e, init, dir) Q e = init -> dir; if(valid(e)) \
  | for(;circ(e -> dir -> F(), H(base), e -> F());) { \
  |   | Q t = e -> dir; \
  |   | splice(e, e -> prev()); \
  |   | splice(e -> r(), e -> r() -> prev()); \
  |   | e -> o = H, H = e, e = t; \
  | }
  | for(;;) {
  |   | DEL(LC, base -> r(), o);
  |   | DEL(RC, base, prev());
  |   | if(!valid(LC) && !valid(RC)) break;
  |   | if(!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
  |   |   | base = conn(RC, base -> r());
  |   | else
  |   |   | base = conn(base -> r(), LC -> r());
  | }
  | return {ra, rb};
}
std::vector<p2> triangulate(std::vector<p2> a) {
  | sort(a.begin(), a.end()); // unique
  | if((int)size(a) < 2) return {};
  | Q e = rec(a).first;
  | std::vector<Q> q = {e};
  | while(cross(e -> o -> F(), e -> F(), e -> p) < 0) e = e -> o;
#define ADD { Q c = e; do { c -> mark = 1; a.push_back(c -> p); \
  | q.push_back(c -> r()), c = c -> next(); } while(c != e); }
  | ADD; a.clear();
  | for(int qi = 0;qi < (int) size(q);) if(!(e = q[qi++]) -> mark) ADD;
  | return a;
} // 返回若干逆时针三角形 \{t[0][0], t[0][1], t[0][2], t[1][0], \dots\}
```

7 geometry3d

7.1 向量

```
struct p3 {
  | db x, y, z;
  | db norm() const { return x * x + y * y + z * z; }
  | db abs() const { return std::sqrt(norm()); }
};
p3 operator + (p3 x, p3 y){ return {x.x + y.x, x.y + y.y, x.z + y.z}; }
p3 operator - (p3 x, p3 y){ return {x.x - y.x, x.y - y.y, x.z - y.z}; }
p3 operator * (p3 x, db y) { return {x.x * y, x.y * y, x.z * y}; }
p3 operator / (p3 x, db y) { return {x.x / y, x.y / y, x.z / y}; }
p3 operator * (p3 x, p3 y) { // 三维叉积需要更高的精度
  | return {
  |   | x.y * y.z - x.z * y.y,
  |   | x.z * y.x - x.x * y.z,
  |   | x.x * y.y - x.y * y.x
  | };
}
db operator % (p3 x, p3 y) { return x.x * y.x + x.y * y.y + x.z * y.z; }
p3 perpvec(p3 x) {
  | return fabs(x.x) > fabs(x.z) ? p3{ x.y, -x.x, 0 } : p3{0, -x.z, x.y};
} // 找到一个与给定向量垂直的向量
db area(p3 a, p3 b, p3 c) { return ((b - a) * (c - a)).abs(); } // 三角形面积两
倍
db volume(p3 d, p3 a, p3 b, p3 c) { // 四面体有向体积六倍
  | return (d - a) % ((b - a) * (c - a));
}
```

7.2 平面

```
struct plane {
  p3 n; db d; // n dot x = d
  plane() {}
  plane(p3 a, p3 b, p3 c) : n((c - a) * (b - a)) { d = n % a; }
  db side(p3 x) const { return n % x - d; }
  db dist(p3 w) const { return side(w) / n.abs(); }
  p3 proj(p3 w) const { return w - n * (side(w) / n.norm()); }
};
```

7.3 直线

```
struct line3 {
  p3 d, o; // kd + o
  line3() {}
  line3(p3 p, p3 q) : d(q - p), o(p) {}
  line3(plane p1, plane p2) : d(p1.n * p2.n) { // 平面交出直线
    o = (p2.n * p1.d - p1.n * p2.d) * d / d.norm();
  }
  db dist(p3 p) const { return (d * (p - o)).abs() / d.abs(); }
  p3 proj(p3 p) const { return o + d * (d % (p - o)) / d.norm(); }//投影
  p3 relf(p3 p) const { return proj(p) * 2 - p; } // 对称
  p3 operator & (const plane & p) const { // 线与平面交
    return o - d * p.side(o) / (p.n % d);
  }
};
db dist(line3 l1, line3 l2) {
  p3 n = l1.d * l2.d;
  if(n.abs() < eps) return l1.dist(l2.o);
  return abs((l2.o - l1.o) % n) / n.abs();
}
p3 closestOnL1(line3 l1, line3 l2) {
  p3 n2 = l2.d * (l1.d * l2.d);
  return l1.o + l1.d * ((l2.o - l1.o) % n2) / (l1.d % n2);
}
bool ispara(plane p1, plane p2){return(p1.n * p2.n).abs() < eps;}//判断是否平行
bool ispara(line3 p1, line3 p2){return(p1.d * p2.d).abs() < eps;}//判断是否平行
bool isperp(plane p1, plane p2){return fabs(p1.n % p2.n) < eps;}//判断是否垂直
bool isperp(line3 p1, line3 p2){return fabs(p1.d % p2.d) < eps;}//判断是否垂直
line3 perpthrough(plane p, p3 o){return line3(o, o + p.n);}//过平面一点做垂线
```

7.4 凸包

```
const int N = 2005;
struct face { int a[3]; plane p; };
int vis[N][N];
std::vector<face> f;
void convex3d(const std::vector<p3> & a) { // need to deal coplane
  if(a.size() < 3) return ;
  auto getface = [&](int i, int j, int k) -> face { return {{i, j, k},
    plane(a[i], a[j], a[k])}; };
  f = {getface(0, 1, 2), getface(0, 2, 1)};
  std::vector<face> tmp[2];
  for(int i = 3;i < (int) a.size();++i) {
    for(auto x : f) {
      if(x.p.dist(a[i]) < -eps) {
        tmp -> push_back(x);
      } else {
        tmp[1].push_back(x);
        for(int t : {0, 1, 2}) vis[x.a[t]][x.a[(t + 1) % 3]] = i;
      }
    }
    for(auto x : tmp[1]) {
      for(int t : {0, 1, 2}) {
        if(vis[x.a[t]][x.a[(t + 1) % 3]] == i && vis[x.a[(t + 1) % 3]]
          [x.a[t]] != i)
          tmp[0].push_back(getface(x.a[t], x.a[(t + 1) % 3], i));
      }
    }
    f = tmp[0]; tmp[0].clear(); tmp[1].clear();
  }
  for(int i = 0;i < (int)a.size();++i) memset(vis[i], 0, a.size() << 2);
}
```

8 Misc

8.1 Pragma

```
#pragma GCC optimize("Ofast")
#pragma GCC optimize("unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2")
#pragma pack(1) // default=8
```

8.2 Barrett

```
struct DIV {
  u64 x;
  void init(u64 v) { x = -1ull / v + 1; }
}; // 带误差版本 x = -1ull/v;
// ret=ans while x*(y-1)<2^64, ans-1<=ret<=ans while x<2^64
u64 operator / (const u64 & x, const DIV & y) {
  return (u128) x * y.x >> 64;
}
```

8.3 快速整除判定

要求 p 是奇数。如果是 (u)int128，求逆循环次数要改成 6。

```
struct Divcheck {
  u64 inv, l;
  Divcheck(u64 p) {
    inv = p, l = -1ull / p;
    for(int j = 0;j < 5;++j) inv *= 2 - inv * p;
  }
  bool check(u64 x) const {
```

```
  |  | return x * inv <= l;
  | }
};
struct Divcheckll {
  | u64 inv; ll l, r;
  | Divcheckll(ll p) {
  |  | inv = p, l = LLONG_MIN / p, r = LLONG_MAX / p;
  |  | for(int j = 0;j < 5;++j) inv *= 2 - inv * p;
  | }
  | bool check(ll x) const {
  |  | return x *= inv, l <= x && x <= r;
  | }
};
```

8.4 LCS

```
int lim;
struct bitset {
  | static const int B = 63;
  | u64 a[N / B + 1];
  | void set(int p) { a[p / B] |= 1ull << (p % B); }
  | bool test(int p) { return a[p / B] >> (p % B) & 1; }
  | void run(const bitset & o) {
  |  | u64 c = 1;
  |  | for(int i = 0;i < lim;++i) {
  |  |  | u64 x = a[i], y = x | o.a[i];
  |  |  | x += x + c + (~y & (1ull << 63) - 1);
  |  |  | a[i] = x & y, c = x >> 63;
  |  | }
  | }
} dp;
```

8.5 日期公式

getday 返回自 1/1/1 起到 $y/m/d$ 的天数，1/1/1 是星期一，所以模 7 可以得算出是周几。

date 是 getday 的逆。

```
// Mon = 0, ... % 7
// days since 1/1/1
int getday(int y, int m, int d) {
  | if(m < 3) -- y, m += 12;
  | return (365 * y + y / 4 - y / 100 + y / 400 + (153 * (m - 3) + 2) / 5 + d
    - 307);
}
void date(int n, int & y, int & m, int & d) {
  | n += 429 + ((4 * n + 1227) / 146097 + 1) * 3 / 4;
  | y = (4 * n - 489) / 1461;
  | n -= y * 1461 / 4;
  | m = (5 * n - 1) / 153;
  | d = n - m * 153 / 5;
  | if (--m > 12) m -= 12, ++y;
}
```

8.6 Xorshift

```
u64 xorshift(u64 x) { x ^= x << 13; x ^= x >> 7; x ^= x << 17; return x; }
u32 xorshift(u32 x) { x ^= x << 13; x ^= x >> 17; x ^= x << 5; return x; }
```

8.7 distributions

8.8 python

```
import sys
sys.set_int_max_str_digits(10**9) # for large integer string conversion
sys.stdout.flush() # IO 交互
import math
print(math.comb(10, 5), math.factorial(10))
print(math.gcd(3, 6, 9), math.gcd(16, 12))
print(math.lcm(4, 6, 9), math.lcm(4, 8))
print(math.isqrt(5))
from fractions import Fraction
a=Fraction(math.e).limit_denominator(10) # 返回分母不超过 10 的最近分数,
  ↪default=1e6
u,v=a.as_integer_ratio()
u,v=a.numerator,a.denominator # 和上面一样
print(u,'/',v)
from decimal import*
getcontext().prec = 100 #default=28
a, b = Decimal(114), Decimal('514')
print(a/b,(a/b)**3)
import sys
input=lambda:sys.stdin.readline().rstrip() # 快读
print('haha', flush=True) # 交互用
sys.stdout.flush() # 或者用这个
```

8.9 浮点数精度

```
float       : 1 bit sign, 8 bit exponent, 23 bit mantissa
double      : 1 bit sign, 11 bit exponent, 52 bit mantissa
long double : 1 bit sign, 15 bit exponent, 64 bit mantissa
```

9 配置

9.1 vimrc

```
set si ci ts=4 sw=4 nu cino=j1 backup undofile
syntax on
map<F9> <ESC>:!g++ % -o %< -std=gnu++20 -g -Wall -Dzqj<CR>
map<F10> <ESC>:!./%<<CR>
map<F4> <ESC>:!gdb %<<CR>
```

9.2 bashrc

```
ulimit -s 1048576
ulimit -v 1048576
```

9.3 对拍

需要 chmod +x

```
while true; do
  | ./gen > 1.in
  | ./naive < 1.in > std.out
  | ./a < 1.in > 1.out
  | if diff 1.out std.out; then
  |   | echo ac
  | else
  |   | echo wa
  |   | break
  | fi
done
```

9.4 编译参数

-D_GLIBCXX_DEBUG : STL debug mode
-fsanitize=address : 内存错误检查
-fsanitize=undefined : UB 检查

9.5 随机素数

947804089 989878776171074651
947167861 995515271526070499
906937111 924834416464491959
919302773 998042656257251069
996011983 971635031393707027

9.6 常数表

n	$\log_{10} n$	$n!$	$C(n, n/2)$	LCM$(1 \dots n)$	P_n
2	0.30102999	2	2	2	2
3	0.47712125	6	3	6	3
4	0.60205999	24	6	12	5
5	0.69897000	120	10	60	7
6	0.77815125	720	20	60	11
7	0.84509804	5040	35	420	15
8	0.90308998	40320	70	840	22
9	0.95424251	362880	126	2520	30
10	1	3628800	252	2520	42
11	1.04139269	39916800	462	27720	56
12	1.07918125	479001600	924	27720	77
15	1.17609126	1.31e12	6435	360360	176
20	1.30103000	2.43e18	184756	232792560	627
25	1.39794001	1.55e25	5200300	26771144400	1958
30	1.47712125	2.65e32	155117520	1.444e14	5604
P_n	37338_{40}	204226_{50}	966467_{60}	190569292_{100}	$1e9_{114}$

$n \le$	10	100	1e3	1e4	1e5	1e6
max $\omega(n)$	2	3	4	5	6	7
max $d(n)$	4	12	32	64	128	240
$\pi(n)$	4	25	168	1229	9592	78498
$n \le$	1e7	1e8	1e9	1e10	1e11	1e12
max $\omega(n)$	8	8	9	10	10	11
max $d(n)$	448	768	1344	2304	4032	6720
$\pi(n)$	664579	5761455	5.08e7	4.55e8	4.12e9	3.7e10
$n \le$	1e13	1e14	1e15	1e16	1e17	1e18
max $\omega(n)$	12	12	13	13	14	15
max $d(n)$	10752	17280	26880	41472	64512	103680
$\pi(n)$	Prime number theorem: $\pi(x) \sim x/\log(x)$					

10 注意事项

10.1 测试项目

pbds tree, float128, int128, long double, submit 命令, printfile, MLE ?= RE, pragma, axv2, python, 测试代码长度限制, 尝试触发 NO-OUTPUT, OUTPUT-LIMIT, RUN-ERROR

10.2 bugs

看数据范围（多测总和）, 变量 shadow, 清空, long long, 数组大小, 模数, MLE?, 对拍记得看输出在不在变, 输出格式，inf 开小，答案初值，STL 重构导致引用失效，极端情况 (n=1)

11 tables

11.1 导数积分

$(\frac{u}{v})' = \frac{u'v-uv'}{v^2}$
$(a^x)' = (\ln a)a^x$
$(\tan x)' = \sec^2 x$
$(\cot x)' = \csc^2 x$
$(\sec x)' = \tan x \sec x$
$(\csc x)' = -\cot x \csc x$
$(\arcsin x)' = \frac{1}{\sqrt{1-x^2}}$
$(\arccos x)' = -\frac{1}{\sqrt{1-x^2}}$
$(\arctan x)' = \frac{1}{1+x^2}$
$(\operatorname{arccot} x)' = -\frac{1}{1+x^2}$
$(\operatorname{arccsc} x)' = -\frac{1}{x\sqrt{1-x^2}}$
$(\operatorname{arcsec} x)' = \frac{1}{x\sqrt{1-x^2}}$
$(\tanh x)' = \operatorname{sech}^2 x$
$(\coth x)' = -\operatorname{csch}^2 x$
$(\operatorname{sech} x)' = -\operatorname{sech} x \tanh x$
$(\operatorname{csch} x)' = -\operatorname{csch} x \coth x$
$(\operatorname{arcsinh} x)' = \frac{1}{\sqrt{1+x^2}}$
$(\operatorname{arccosh} x)' = \frac{1}{\sqrt{x^2-1}}$
$(\operatorname{arctanh} x)' = \frac{1}{1-x^2}$
$(\operatorname{arccoth} x)' = \frac{1}{x^2-1}$
$(\operatorname{arccsch} x)' = -\frac{1}{|x|\sqrt{1+x^2}}$
$(\operatorname{arcsech} x)' = -\frac{1}{x\sqrt{1-x^2}}$

$ax^2+bx+c(a>0)$

1. $\int \frac{dx}{ax^2+bx+c} = \begin{cases} \frac{2}{\sqrt{4ac-b^2}} \arctan \frac{2ax+b}{\sqrt{4ac-b^2}} + C & (b^2<4ac) \\ \frac{1}{\sqrt{b^2-4ac}} \ln\left|\frac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}}\right| + C & (b^2>4ac) \end{cases}$
2. $\int \frac{x}{ax^2+bx+c}dx = \frac{1}{2a}\ln|ax^2+bx+c| - \frac{b}{2a}\int \frac{dx}{ax^2+bx+c}$

$\sqrt{\pm ax^2+bx+c}(a>0)$

1. $\int \frac{dx}{\sqrt{ax^2+bx+c}} = \frac{1}{\sqrt{a}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
2. $\int \sqrt{ax^2+bx+c}dx = \frac{2ax+b}{4a}\sqrt{ax^2+bx+c} + \frac{4ac-b^2}{8\sqrt{a^3}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
3. $\int \frac{x}{\sqrt{ax^2+bx+c}}dx = \frac{1}{a}\sqrt{ax^2+bx+c} - \frac{b}{2\sqrt{a^3}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$
4. $\int \frac{dx}{\sqrt{c+bx-ax^2}} = -\frac{1}{\sqrt{a}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$
5. $\int \sqrt{c+bx-ax^2}dx = \frac{2ax-b}{4a}\sqrt{c+bx-ax^2} + \frac{b^2+4ac}{8\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$
6. $\int \frac{x}{\sqrt{c+bx-ax^2}}dx = -\frac{1}{a}\sqrt{c+bx-ax^2} + \frac{b}{2\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$

$\sqrt{\pm\frac{x-a}{x-b}}$ 或 $\sqrt{(x-a)(x-b)}$

1. $\int \frac{dx}{\sqrt{(x-a)(b-x)}} = 2\arcsin\sqrt{\frac{x-a}{b-x}} + C\,(a<b)$
2. $\int \sqrt{(x-a)(b-x)}dx = \frac{2x-a-b}{4}\sqrt{(x-a)(b-x)} + \frac{(b-a)^2}{4}\arcsin\sqrt{\frac{x-a}{b-x}} + C,\,(a<b)$

三角函数的积分

1. $\int \tan x dx = -\ln|\cos x| + C$
2. $\int \cot x dx = \ln|\sin x| + C$
3. $\int \sec x dx = \ln\left|\tan\left(\frac{\pi}{4}+\frac{x}{2}\right)\right| + C = \ln|\sec x + \tan x| + C$
4. $\int \csc x dx = \ln\left|\tan\frac{x}{2}\right| + C = \ln|\csc x - \cot x| + C$
5. $\int \sec^2 x dx = \tan x + C$
6. $\int \csc^2 x dx = -\cot x + C$
7. $\int \sec x \tan x dx = \sec x + C$
8. $\int \csc x \cot x dx = -\csc x + C$
9. $\int \sin^2 x dx = \frac{x}{2} - \frac{1}{4}\sin 2x + C$
10. $\int \cos^2 x dx = \frac{x}{2} + \frac{1}{4}\sin 2x + C$
11. $\int \sin^n x dx = -\frac{1}{n}\sin^{n-1}x\cos x + \frac{n-1}{n}\int \sin^{n-2}x dx$
12. $\int \cos^n x dx = \frac{1}{n}\cos^{n-1}x\sin x + \frac{n-1}{n}\int \cos^{n-2}x dx$
13. $\int \frac{dx}{\sin^n x} = -\frac{1}{n-1}\frac{\cos x}{\sin^{n-1}x} + \frac{n-2}{n-1}\int \frac{dx}{\sin^{n-2}x}$
14. $\int \frac{dx}{\cos^n x} = \frac{1}{n-1}\frac{\sin x}{\cos^{n-1}x} + \frac{n-2}{n-1}\int \frac{dx}{\cos^{n-2}x}$
15. $$\begin{aligned}&\int \cos^m x \sin^n x dx \\ =&\frac{1}{m+n}\cos^{m-1}x\sin^{n+1}x + \frac{m-1}{m+n}\int \cos^{m-2}x\sin^n x dx \\ =&-\frac{1}{m+n}\cos^{m+1}x\sin^{n-1}x + \frac{n-1}{m+1}\int \cos^m x\sin^{n-2}x dx\end{aligned}$$
16. $\int \frac{dx}{a+b\sin x} = \begin{cases} \frac{2}{\sqrt{a^2-b^2}}\arctan\frac{a\tan\frac{x}{2}+b}{\sqrt{a^2-b^2}} + C & (a^2>b^2) \\ \frac{1}{\sqrt{b^2-a^2}}\ln\left|\frac{a\tan\frac{x}{2}+b-\sqrt{b^2-a^2}}{a\tan\frac{x}{2}+b+\sqrt{b^2-a^2}}\right| + C & (a^2<b^2) \end{cases}$
17. $\int \frac{dx}{a+b\cos x} = \begin{cases} \frac{2}{a+b}\sqrt{\frac{a+b}{a-b}}\arctan\left(\sqrt{\frac{a-b}{a+b}}\tan\frac{x}{2}\right) + C & (a^2>b^2) \\ \frac{1}{a+b}\sqrt{\frac{a+b}{a-b}}\ln\left|\frac{\tan\frac{x}{2}+\sqrt{\frac{a+b}{b-a}}}{\tan\frac{x}{2}-\sqrt{\frac{a+b}{b-a}}}\right| + C & (a^2<b^2) \end{cases}$
18. $\int \frac{dx}{a^2\cos^2 x + b^2\sin^2 x} = \frac{1}{ab}\arctan\left(\frac{b}{a}\tan x\right) + C$
19. $\int \frac{dx}{a^2\cos^2 x - b^2\sin^2 x} = \frac{1}{2ab}\ln\left|\frac{b\tan x+a}{b\tan x-a}\right| + C$
20. $\int x\sin ax dx = \frac{1}{a^2}\sin ax - \frac{1}{a}x\cos ax + C$
21. $\int x^2\sin ax dx = -\frac{1}{a}x^2\cos ax + \frac{2}{a^2}x\sin ax + \frac{2}{a^3}\cos ax + C$
22. $\int x\cos ax dx = \frac{1}{a^2}\cos ax + \frac{1}{a}x\sin ax + C$
23. $\int x^2\cos ax dx = \frac{1}{a}x^2\sin ax + \frac{2}{a^2}x\cos ax - \frac{2}{a^3}\sin ax + C$

反三角函数的积分 (其中 $a>0$)

1. $\int \arcsin\frac{x}{a}dx = x\arcsin\frac{x}{a} + \sqrt{a^2-x^2} + C$
2. $\int x\arcsin\frac{x}{a}dx = (\frac{x^2}{2}-\frac{a^2}{4})\arcsin\frac{x}{a} + \frac{x}{4}\sqrt{x^2-x^2} + C$
3. $\int x^2\arcsin\frac{x}{a}dx = \frac{x^3}{3}\arcsin\frac{x}{a} + \frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2} + C$
4. $\int \arccos\frac{x}{a}dx = x\,arccos\frac{x}{a} - \sqrt{a^2-x^2} + C$
5. $\int x\arccos\frac{x}{a}dx = (\frac{x^2}{2}-\frac{a^2}{4})\arccos\frac{x}{a} - \frac{x}{4}\sqrt{a^2-x^2} + C$
6. $\int x^2\arccos\frac{x}{a}dx = \frac{x^3}{3}\arccos\frac{x}{a} - \frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2} + C$
7. $\int \arctan\frac{x}{a}dx = x\arctan\frac{x}{a} - \frac{a}{2}\ln(a^2+x^2) + C$
8. $\int x\arctan\frac{x}{a}dx = \frac{1}{2}(a^2+x^2)\arctan\frac{x}{a} - \frac{a}{2}x + C$
9. $\int x^2\arctan\frac{x}{a}dx = \frac{x^3}{3}\arctan\frac{x}{a} - \frac{a}{6}x^2 + \frac{a^3}{6}\ln(a^2+x^2) + C$

指数函数的积分

1. $\int a^x dx = \frac{1}{\ln a}a^x + C$
2. $\int e^{ax}dx = \frac{1}{a}a^{ax} + C$
3. $\int xe^{ax}dx = \frac{1}{a^2}(ax-1)a^{ax} + C$
4. $\int x^n e^{ax}dx = \frac{1}{a}x^n e^{ax} - \frac{n}{a}\int x^{n-1}e^{ax}dx$
5. $\int xa^x dx = \frac{x}{\ln a}a^x - \frac{1}{(\ln a)^2}a^x + C$
6. $\int x^n a^x dx = \frac{1}{\ln a}x^n a^x - \frac{n}{\ln a}\int x^{n-1}a^x dx$
7. $\int e^{ax}\sin bx dx = \frac{1}{a^2+b^2}e^{ax}(a\sin bx - b\cos bx) + C$
8. $\int e^{ax}\cos bx dx = \frac{1}{a^2+b^2}e^{ax}(b\sin bx + a\cos bx) + C$
9. $\int e^{ax}\sin^n bx dx = \frac{1}{a^2+b^2n^2}e^{ax}\sin^{n-1}bx(a\sin bx - nb\cos bx) + \frac{n(n-1)b^2}{a^2+b^2n^2}\int e^{ax}\sin^{n-2}bx dx$
10. $\int e^{ax}\cos^n bx dx = \frac{1}{a^2+b^2n^2}e^{ax}\cos^{n-1}bx(a\cos bx + nb\sin bx) + \frac{n(n-1)b^2}{a^2+b^2n^2}\int e^{ax}\cos^{n-2}bx dx$

对数函数的积分

1. $\int \ln x dx = x\ln x - x + C$
2. $\int \frac{dx}{x\ln x} = \ln|\ln x| + C$
3. $\int x^n\ln x dx = \frac{1}{n+1}x^{n+1}(\ln x - \frac{1}{n+1}) + C$
4. $\int (\ln x)^n dx = x(\ln x)^n - n\int (\ln x)^{n-1}dx$
5. $\int x^m(\ln x)^n dx = \frac{1}{m+1}x^{m+1}(\ln x)^n - \frac{n}{m+1}\int x^m(\ln x)^{n-1}dx$

STL 积分/求和 (need std::)

1. $\int_0^1 t^{x-1}(1-t)^{y-1}dt = beta(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$
2. $\int_0^{+\infty} t^{num-1}e^{-t}dt = tgamma(num) = e^{lgamma(num)} = \Gamma(num)$
3. $\int_{num}^{+\infty}\frac{e^{-t}}{t}dt = -expint(-num)$
4. $\sum_{n=1}^{+\infty} n^{-num} = riemann\_zeta(num)$
5. $\frac{2}{\sqrt{\pi}}\int_0^{arg} e^{-t^2}dt = erf(arg)$
6. $\int_0^{phi}\frac{d\theta}{\sqrt{1-k^2\sin^2\theta}} = ellint\_1(k,phi)$
7. $\int_0^{phi}\sqrt{1-k^2\sin^2\theta}d\theta = ellint\_2(k,phi)$

椭圆离心率: $e = \sqrt{1-\frac{b^2}{a^2}}$, 周长: $2a\,ellint\_2(e,\pi)$