1 Geometry

1.1 Circle intersection

```
inline vector<pt> intersect(const circle& a, const circle& b) {
  if (a.c == b.c) return vector<pt> (abs(a.r - b.r) < EPS ? 3 : 0);
  ld d = dist(a.c, b.c);
  if (d - EPS > a.r + b.r) return vector<pt> ();
  if (d + min(a.r, b.r) + EPS < max(a.r, b.r)) return vector<pt> ();
  ld cosa = (sqr(a.r) + sqr(d) - sqr(b.r)) / (2 * a.r * d);
  cosa = max(ld(-1), min(ld(1), cosa));
  ld sina = sqrt(1 - sqr(cosa));
  pt v = normal(b.c - a.c, a.r);
  vector<pt> ans(2);
  ans[0] = a.c + rotate(v, sina, cosa);
  ans[1] = a.c + rotate(v, -sina, cosa);
  if (abs(sina) < EPS) ans.pop_back();
  return ans;
}
```

1.2 Circle tangents

```
inline vector<line> getTangents(const circle& a, const circle& b) {
  if (a == b) return vector<line> (5);
  vector<line> ans;
  ld x = (b.c - a.c).x, y = (b.c - a.c).y;
  forn(mask, 4) {
    ld r1 = a.r * ((mask & 1) ? 1 : -1);
    ld r2 = b.r * ((mask & 2) ? 1 : -1);
    ld d = sqr(x) + sqr(y) - sqr(r1 - r2);
    if (d < -EPS) continue;
    if (mask > 1 && abs(d) < EPS) continue;
    d = sqrt(max(ld(0), d));
    ans.pb(line());
    ans.back().a = (x * (r2 - r1) - y * d) / (sqr(x) + sqr(y));
    ans.back().b = (y * (r2 - r1) + x * d) / (sqr(x) + sqr(y));
    ans.back().c = r1 - (ans.back().a * a.c.x + ans.back().b * a.c.y);
  }
  return ans;
}
```

1.3 Circle tangents from point

```
inline vector<pt> getTangents(const pt& p, const circle& c) {
  ld d = dist(p, c.c);
  vector<pt> ans;
  if (d + EPS < c.r) return ans;
  ld cosa = max(ld(-1), min(ld(1), c.r / d));
  ld sina = sqrt(1 - sqr(cosa));
  pt v = normal(p - c.c, c.r);
  ans.pb(c.c + rotate(v, sina, cosa));
  ans.pb(c.c + rotate(v, -sina, cosa));
  if (abs(sina) < EPS) ans.pop_back();
  return ans;
}
```

1.4 Sphere intersection

```
inline void intersect(const sphere& a, const sphere& b) {
  ld d = dist(a.c, b.c);
  if (d - EPS > a.r + b.r) return;
  if (min(a.r, b.r) + d + EPS < max(a.r, b.r)) return;
  if (d < EPS) return; // the same spheres

  ld cosa = (sqr(a.r) - sqr(b.r) + sqr(d)) / (2 * a.r * d);
  cosa = max(ld(-1), min(ld(1), cosa));
  ld sina = sqrt(max(ld(0), 1 - sqr(cosa)));

  ld l = a.r * cosa; // (b.c - a.c) - normal of circle
  ld r = a.r * sina; // radius of circle
  pt p = a.c + normal(b.c - a.c, l); // center of circle

  ld sinb = abs(a.c.z - b.c.z) / d;
  sinb = max(ld(-1), min(ld(1), sinb));
  ld cosb = sqrt(max(ld(0), 1 - sqr(sinb)));

  ld dz = r * cosb;

  ld z1 = p.z - dz; // minium and maximum
  ld z2 = p.z + dz; // z-coordinate of intersection
}
```

1.5 Facets

```
pt pole;
inline bool cmp(int i, int j) { return ang(pts[i] - pole) < ang(pts[j] -
    pole); }
inline vector< vector<int> > getFacets() {
  forn(i, sz(pts)) {
    pole = pts[i];
    sort(all(g[i]), cmp);
  }
  vector< vector<int> > facets;
  forn(i, sz(pts))
    forn(j, sz(g[i]))
      if (!us[i][j]) {
        int from = i, to = g[i][j], num = j;
        facets.pb(vector<int> ());
        while (!us[from][num]) {
          us[from][num] = true;
          facets.back().pb(from);
          pole = pts[to];
          num = (lower_bound(all(g[to]), from, cmp) - g[to].begin() + 1) %
              sz(g[to]);
          from = to;
          to = g[from][num];
        }
      }
  return facets;
}
```

1.6 Minimum covering circle

```
inline circle getCircumCircle(const pt& a, const pt& b, const pt& c) {
  if (abs(cross(a - c, b - c)) < EPS) {
    if (dot(a - c, b - c) < 0) return circle(a, b);
    if (dot(a - b, c - b) < 0) return circle(a, c);
    return circle(b, c);
  }
  circle ans;
  pt c1 = (a + b) / 2, c2 = (a + c) / 2, v1 = b - a, v2 = c - a;
  assert(intersect(c1, c1 + pt(-v1.y, v1.x), c2, c2 + pt(-v2.y, v2.x), ans.c));
  ans.r = dist(ans.c, a);
  return ans;
}
circle findMinCircle(vector<pt> a) {
  if (sz(a) == 1) return circle(a[0], 0);
  random_shuffle(all(a));
  circle ans(a[0], a[1]);
  fore(i, 2, sz(a)) {
    if (ans.in(a[i])) continue;
    swap(a[0], a[i]);
    random_shuffle(a.begin() + 1, a.begin() + i + 1);
    ans = circle(a[0], a[1]);
    fore(j, 2, i + 1) {
      if (ans.in(a[j])) continue;
      swap(a[1], a[j]);
      random_shuffle(a.begin() + 2, a.begin() + j + 1);
      ans = circle(a[0], a[1]);
      fore(k, 2, j + 1)
        if (!ans.in(a[k]))
          ans = getCircumCircle(a[0], a[1], a[k]);
    }
  }
  return ans;
}
```

1.7 Polygon tangents

```
inline int calc(const pt& p, const vector<pt>& pol, int lf, int rg, int sign)
{ // [lf, rg) - interval of vertexes that we should process
  int d = rg - lf; // sign - direction in which we processing vertexes
  if (d < 0) d += sz(pol);
  d = max(0, d - 1);
  int l = 0, r = d;
  while (l != r) {
    int mid = (l + r) >> 1;
    int idx = (lf + mid) % sz(pol);
    if (cross(pol[idx] - p, pol[(idx + 1) % sz(pol)] - pol[idx]) * sign > -EPS
        * sign) r = mid;
    else l = mid + 1;
  }
  return (lf + l) % sz(pol);
}
inline pti tangents(const pt& p, const vector<pt>& pol, const vector<pair<ld,
    int> >& angs, const pt& c)
{ // angs - already sorted (by polar angle from center of mass) list of
    polygon vertexes <angle, idx>
  pt pp = p + normal(c - p, 1e9); // p - start of tangents, c - center of mass
  int lf1 = int(lower_bound(all(angs), mp(ang(pp - c), -INF)) - angs.begin())
      % sz(angs);
  lf1 = angs[lf1].sc;
  int lf2 = int(lower_bound(all(angs), mp(ang(p - c), -INF)) - angs.begin()) %
      sz(angs);
  lf2 = angs[lf2].sc;
  return mp(calc(p, pol, lf1, lf2, -1), calc(p, pol, lf2, lf1, +1));
}
```

1.8 Semi-plane intersection

```
struct pt {
  ld x, y;
  explicit pt() { }
  explicit pt(ld x, ld y): x(x), y(y) { }
};
inline ld ang(const pt& v) {
  ld ans = atan2(v.y, v.x);
  (ans < 0) && (ans += 2 * PI);
  return ans;
}
struct line {
  ld a, b, c; // describes semi-plane of kind: ax + by + c < 0
  ld alpha;
  explicit line() { }
  explicit line(ld A, ld B, ld C): a(A), b(B), c(C) {
    ld v = sqrt(sqr(a) + sqr(b));
    a /= v, b /= v, c /= v;
    alpha = ang(pt(-a, -b));
  }
};
inline ld det(const ld& a, const ld& b, const ld& c, const ld& d) { return a *
    d - b * c; }
inline bool intersect(const line& a, const line& b, pt& ans) {
  ld d = det(a.a, b.a, a.b, b.b);
  if (abs(d) < EPS) return false;
  ans = pt(det(-a.c, a.b, -b.c, b.b) / d, det(a.a, -a.c, b.a, -b.c) / d);
  return true;
}
inline bool operator< (const line& a, const line& b) { return a.alpha <
    b.alpha; }
bool remove(list<line>& z) {
  while (sz(z) > 1) {
    line a = z.back(); z.pop_back();
    line last = z.back(); z.pop_back();
    if (abs(det(last.a, last.b, a.a, a.b)) < EPS) {
      if (abs(last.a - a.a) < EPS && abs(last.b - a.b) < EPS) {
        if (a.c < last.c + EPS) {
          z.pb(last);
          break;
        }
        else z.pb(a);
      }
      else {
        if (last.c + EPS > -a.c) return false;
        z.pb(last), z.pb(a);
        break;
      }
```

```
    }
    else if (!z.empty()) {
      line plast = z.back(); z.pop_back();
      pt p;
      if (!intersect(plast, last, p) || a.a * p.x + a.b * p.y + a.c < EPS) {
        z.pb(plast), z.pb(last), z.pb(a);
        break;
      }
      z.pb(plast), z.pb(a);
    }
    else
    {
      z.pb(last), z.pb(a);
      break;
    }
  }
  return true;
}
bool intersect(vector<line> a, vector<pt>& ans) {
  sort(all(a)); // if the intersection is finite returns true
  int cnt = 0;  // and the polygon of intersection in ans,
  forn(i, sz(a)) { // otherwise ans is undefined
    ld a1 = a[i].alpha, a2 = a[(i + 1) % sz(a)].alpha;
    (a2 + EPS < a1) && (a2 += 2 * PI);
    if (a2 - a1 - EPS > PI) cnt++;
  }
  if (cnt == 1) return false;
  list<line> z;
  ans.clear();
  forn(i, sz(a)) {
    z.pb(a[i]);
    if (!remove(z)) return true;
  }
  forn(i, 2) {
    forn(j, sz(z))
      if (sz(z) > 1) {
        z.pb(z.front()), z.pop_front();
        if (!remove(z)) return true;
      }
    z.reverse();
  }
  pt p;
  vector<line> res(all(z));
  forn(i, sz(res)) {
    if (!intersect(res[i], res[(i + 1) % sz(res)], p)) return false;
    ans.pb(p);
  }
  return true;
}
```

1.9 Voronoi diagram with $O(n^3)$

```
inline vector<pt> cut(const vector<pt>& pol, const line& a, const pt& p) {
  vector<pt> ans;
  int sg = sign(a, p);
  forn(i, sz(pol)) {
    pt cur = pol[i], next = pol[(i + 1) % sz(pol)];
    pt mid;
    int sg1 = sign(a, cur) * sg, sg2 = sign(a, next) * sg;
    if (sg1 * sg2 == -1) {
      ld A = next.y - cur.y, B = cur.x - next.x, C = -(A * cur.x + B * cur.y);
      assert(intersect(a, line(A, B, C), mid));
      ans.pb(mid);
    }
    if (sg2 >= 0) ans.pb(next);
  }
  if (sz(ans) < 3) ans.clear();
  return ans;
}
```

1.10 Convex hull in plane

```
vector<pt> convexHull(vector<pt> a) {
  sort(all(a));
  a.erase(unique(all(a)), a.end());
  vector<pt> up, dw;
  forn(i, sz(a)) {
    while (sz(up) > 1 && cross(a[i] - up.back(), up.back() - up[sz(up) - 2])
        <= 0)
      up.pop_back();
    up.pb(a[i]);
  }
  forn(i, sz(a)) {
    while (sz(dw) > 1 && cross(a[i] - dw.back(), dw.back() - dw[sz(dw) - 2])
        >= 0)
      dw.pop_back();
    dw.pb(a[i]);
  }
  reverse(all(up));
  if (sz(up) > 1) dw.insert(dw.end(), up.begin() + 1, up.end() - 1);
  return dw;
}
```

2 Graphs

2.1 Cutpoints, bridges and other

```
int curt, tin[N], z[N];
int u = 0, used[N], cut[N];
// both need to clear
vector<int> bridges, eg[N];
void dfs(int v, int p, int pnum) {
  tin[v] = z[v] = curt++;
  used[v] = u;
  int cnt = 0;
  cut[v] = false;
  forn(i, sz(g[v])) {
    int to = g[v][i], num = id[v][i];
    if (to == p && num == pnum) continue;
    if (used[to] == u) {
      if (p != -1 && tin[to] <= tin[p]) {
        eg[num].pb(pnum);
        eg[pnum].pb(num);
```

```
      }
      z[v] = min(z[v], tin[to]);
      continue;
    }
    dfs(to, v, num);
    cnt++;
    z[v] = min(z[v], z[to]);
    if (p != -1 && z[to] >= tin[v]) cut[v] = true;
    if (z[to] > tin[v]) bridges.pb(num);
    if (p != -1 && z[to] <= tin[p]) {
      eg[num].pb(pnum);
      eg[pnum].pb(num);
    }
  }
  if (p == -1 && cnt > 1) cut[v] = true;
}
```

2.2 Dinic algorithm

```
int lev[V];
int head, tail, q[V];
inline bool bfs(int s, int t) {
  memset(lev, 63, sizeof(lev));
  head = tail = 0;
  lev[s] = 0;
  q[head++] = s;
  while (head != tail) {
    int v = q[tail++];
    forn(i, sz(g[v]))
      if (g[v][i].f < g[v][i].c && lev[g[v][i].to] > lev[v] + 1) {
        lev[g[v][i].to] = lev[v] + 1;
        q[head++] = g[v][i].to;
      }
  }
  return lev[t] < INF / 2;
}
int ptr[V];
int dfs(int v, int t, int f) {
  if (v == t) return f;
  for ( ; ptr[v] < sz(g[v]); ptr[v]++) {
    edge& e = g[v][ptr[v]];
    if (e.f == e.c || lev[e.to] != lev[v] + 1) continue;
    int df = dfs(e.to, t, min(f, e.c - e.f));
    if (df > 0) {
      e.f += df;
      g[e.to][e.rv].f -= df;
      return df;
    }
  }
  return 0;
}
inline int dinic(int s, int t) {
  int ans = 0;
  while (bfs(s, t)) {
    memset(ptr, 0, sizeof(ptr));
    for (int f; (f = dfs(s, t, INF)) != 0; ans += f);
  }
  return ans;
}
```

2.3 Heavy-light decomposition

```
inline bool heavy(int v, int to) { return cnt[to] > cnt[v] / 2; }
int idx[N];
vector< vector<int> > paths;
void dfs2(int v, int p) {
  idx[v] = sz(paths);
  if (p != -1 && heavy(p, v)) idx[v] = idx[p];
  else paths.pb(vector<int> ());
  paths[idx[v]].pb(v);
  forn(i, sz(g[v])) if (g[v][i] != p) dfs2(g[v][i], v);
}
int st[N], pos[N];
vector<int> vs;
inline void prepareHL() {
  dfs(0, 0); // prepare lca, cnt
  paths.clear();
  vs.clear();
  dfs2(0, -1);
  forn(i, sz(paths)) {
    st[i] = sz(vs);
    forn(j, sz(paths[i])) {
      pos[paths[i][j]] = sz(vs);
      vs.pb(paths[i][j]);
    }
  }
  // build tree on array vs
}
inline int calc(int v, int l) {
  for (int ans = 0; ; v = p[0][vs[st[idx[v]]]]) {
    ans = max(ans, max(0, 0, sz(vs) - 1, max(pos[l], st[idx[v]]), pos[v]));
    if (idx[v] == idx[l]) return ans;
  }
}
```

2.4 Smith algorithm

```
int u = 0, used[N];
int deg[N];
vector<int> g[N], tg[N], ttg[N];
inline bool update(vector<int>& gr, const vector<pt>& e, int curg) {
  int n = sz(gr);
  forn(i, n) {
    g[i].clear();
    tg[i].clear();
    ttg[i].clear();
  }
  forn(i, sz(e)) {
    g[e[i].x].pb(e[i].y);
    tg[e[i].y].pb(e[i].x);
  }
  memset(deg, 63, sizeof(deg));
  queue<int> q;
  forn(i, n) {
```

```
    if (gr[i] != -1) continue;
    deg[i] = 0;
    u++;
    forn(j, sz(g[i])) {
      int to = g[i][j];
      if (gr[to] != -1) used[gr[to]] = u;
      else {
        ttg[to].pb(i);
        deg[i]++;
      }
    }
    for (int j = 0; ; j++)
      if (used[j] != u) {
        assert(j <= curg);
        if (j != curg) deg[i] = INF;
        if (deg[i] == 0) q.push(i);
        break;
      }
  }
  u++;
  while (!q.empty()) {
    int v = q.front();
    q.pop();
    gr[v] = curg;
    forn(i, sz(tg[v])) {
      int to = tg[v][i];
      if (used[to] == u) continue;
      used[to] = u;
      forn(j, sz(ttg[to])) {
        int vv = ttg[to][j];
        if (--deg[vv] == 0) q.push(vv);
        assert(deg[vv] >= 0);
      }
    }
  }
  return find(all(gr), curg) != gr.end();
}
inline vector<int> smith(int n, vector<pt> e) {
  sort(all(e));
  e.erase(unique(all(e)), e.end());
  vector<int> gr(n, -1);
  for (int g = 0; update(gr, e, g); g++);
  return gr;
}
```

2.5 Stoer-Wagner algorithm

```
inline int findMinCut() {
  vector<int> dsu(n);
  forn(i, n) dsu[i] = i;
  int ans = INF;
  forn(i, n - 1) {
    vector<int> col(n, 1), sum(n, 0);
    int s = -1;
    forn(j, n) if (leader(dsu, j) == j) s = j;
    col[s] = 0;
    forn(j, sz(g[s])) sum[leader(dsu, g[s][j])] += w[s][j];
    int x = -1, y = s, last = INF;
    while (true) {
      int v = -1;
      forn(j, n)
        if (col[j] == 1 && leader(dsu, j) == j && (v == -1 || sum[v] < sum[j]))
          v = j;
      if (v == -1) break;
      x = y, y = v;
      last = sum[v];
      col[v] = 0;
      forn(j, sz(g[v])) sum[leader(dsu, g[v][j])] += w[v][j];
    }
    ans = min(ans, last);
    forn(j, sz(g[y])) g[x].pb(g[y][j]), w[x].pb(w[y][j]);
    dsu[y] = x;
  }
  return ans;
}
```

2.6 Chu-Liu algorithm

```
int relax(int v, const vector<vector<int> >& g, vector<pt>& in)
{ // edge = { x, y, w, id }, if need to hack change id->exist
  int& ans = in[v].ft;
  if (ans != -1) return ans;
  ans = in[v].sc;
  forn(i, sz(g[v])) ans ^= relax(g[v][i], g, in);
  return ans;
}
inline vector<int> arborescence(int root, int n, vector<edge> e)
{ // for forest add edges (x, i) (i=0,n), x - fictitious vertex
  vector<pt> in(sz(e), mp(-1, 0));
  vector<vector<int> > g(sz(e));
  vector<bool> rem(n, true);
  rem[root] = false;
  forn(i, sz(e)) e[i].id = i;
  while (true) {
    vector<int> p(n, -1), pw(n, INT_MAX);
    forn(i, sz(e)) {
      int v = e[i].y;
      if (v != root && pw[v] > e[i].w)
        pw[v] = e[i].w, p[v] = i;
    }
    forn(i, n) if (rem[i] && p[i] == -1) return vector<int> ();
    int c = 0;
    vector<int> cycle, comp(n, -1);
    forn(i, n)
      if (comp[i] == -1) {
        int v = i;
        while (v != -1 && comp[v] == -1) {
          cycle.pb(v);
          comp[v] = c;
          v = p[v] == -1 ? -1 : e[p[v]].x;
        }
        if (v != -1 && comp[v] == c) {
          int idx = int(find(all(cycle), v) - cycle.begin());
          forn(j, idx) comp[cycle[j]] = -1;
          cycle.erase(cycle.begin(), cycle.begin() + idx);
          break;
```

```
        }
        cycle.clear();
        c++;
      }
    if (cycle.empty()) {
      forn(i, n)
        if (p[i] != -1)
          in[e[p[i]].id].sc ^= 1;
      break;
    }
    int mv = INT_MAX, mp = -1;
    forn(i, sz(cycle)) {
      edge& cure = e[p[cycle[i]]];
      in[cure.id].sc ^= 1;
      if (mv > cure.w) mv = cure.w, mp = cure.id;
      rem[cycle[i]] = i == 0;
    }
    in[mp].sc ^= 1;
    forn(i, sz(e))
      if (comp[e[i].x] != c || comp[e[i].y] != c) {
        if (comp[e[i].y] == c) {
          e[i].w -= pw[e[i].y];
          e[i].w += mv;
          g[e[p[e[i].y]].id].pb(e[i].id);
          g[mp].pb(e[i].id);
        }
        if (comp[e[i].x] == c) e[i].x = cycle[0];
        if (comp[e[i].y] == c) e[i].y = cycle[0];
      }
    forn(i, sz(e))
      if (comp[e[i].x] == c && comp[e[i].y] == c) {
        swap(e[i], e.back());
        e.pop_back();
        i--;
      }
  }
  vector<int> ans;
  forn(i, sz(in))
    if (relax(i, g, in))
      ans.pb(i);
  // returns list of indexes of edges in order of e
  return ans;
}
```

2.7 Bipartite graph optimal edge coloring

```
int deg[N], next[N][C];
void repaint(int v, int c1, int c2) {
  int to = next[v][c1];
  if (to == -1) return;
  next[v][c1] = -1, next[to][c1] = -1;
  repaint(to, c2, c1);
  next[v][c2] = to, next[to][c2] = v;
}
inline int paintEdges(int n, int m, const vector<pt>& e)
{ // first part [0, n), second part [n, n + m)
  int maxDeg = 0;
  memset(deg, 0, sizeof(deg));
  memset(next, -1, sizeof(next));
  forn(i, sz(e)) {
    int x = e[i].x, y = e[i].y + n;
    maxDeg = max(maxDeg, ++deg[x]);
    maxDeg = max(maxDeg, ++deg[y]);
    int c1 = -1, c2 = -1;
    forn(c, maxDeg) {
      if (next[x][c] == -1 && next[y][c] == -1) {
        next[x][c] = y, next[y][c] = x;
        goto done;
      }
      if (next[x][c] != -1 && next[y][c] == -1) c1 = c;
      if (next[x][c] == -1 && next[y][c] != -1) c2 = c;
    }
    repaint(x, c1, c2);
    next[x][c1] = y, next[y][c1] = x;
    done:;
  }
  return maxDeg;
}
```

2.8 Hungarian algorithm

```
inline vector<int> minCostMatching(int n, int m, int a[N][N]) { // n <= m
  vector<int> u(m, 0), v(m + 1, 0), p(m + 1, m); // O(n^2m)
  forn(i, n) {
    int jj = m; p[jj] = i;
    vector<int> used(m + 1, false), minv(m, INF), prev(m);
    while (p[jj] != m) {
      int ii = p[jj], minVal = INF, minPos = -1;
      used[jj] = true;
      forn(j, m)
        if (!used[j]) {
          int curVal = a[ii][j] - u[ii] - v[j];
          if (minv[j] > curVal) minv[j] = curVal, prev[j] = jj;
          if (minVal > minv[j]) minVal = minv[j], minPos = j;
        }
      forn(j, m + 1)
        if (used[j]) u[p[j]] += minVal, v[j] -= minVal;
        else minv[j] -= minVal;
      jj = minPos;
    }
    for (int v = jj; v != m; v = prev[v]) p[v] = p[prev[v]];
  }
  vector<int> ans(n); forn(i, m) if (p[i] != m) ans[p[i]] = i;
  return ans;
}
```

2.9 Edmonds algorithm

```
int match[N], p[N], blossom[N], base[N];
inline int lca(int a, int b) {
  static int us[N];
  forn(i, n) us[i] = false;
  while (true) {
    a = base[a];
```

```
    us[a] = true;
    if (match[a] == -1) break;
    a = p[match[a]];
  }
  while (true) {
    b = base[b];
    if (us[b]) return b;
    b = p[match[b]];
  }
}
inline void markPath(int v, int b, int pr) {
  while (base[v] != b) {
    blossom[base[v]] = blossom[base[match[v]]] = true;
    p[v] = pr;
    pr = match[v];
    v = p[match[v]];
  }
}
inline int augment(int root) {
  static int used[N];
  forn(i, n) p[i] = -1, used[i] = false, base[i] = i;
  queue<int> q;
  used[root] = true;
  q.push(root);
  while (!q.empty()) {
    int v = q.front();
    q.pop();
    forn(i, sz(g[v])) {
      int to = g[v][i];
      if (base[to] == base[v] || match[to] == v) continue;
      if (to == root || (match[to] != -1 && p[match[to]] != -1)) {
        int b = lca(v, to);
        assert(base[b] == b);
        forn(j, n) blossom[j] = false;
        markPath(v, b, to);
        markPath(to, b, v);
        forn(j, n)
          if (blossom[base[j]]) {
            base[j] = b;
            if (!used[j]) {
                used[j] = true;
                q.push(j);
            }
          }
      }
      else if (p[to] == -1) {
        p[to] = v;
          if (match[to] == -1) return to;
        used[match[to]] = true;
        q.push(match[to]);
      }
    }
  }
  return -1;
}
inline vector<pt> edmonds() // O(n^3)
{ // add greedy matching to make algo more fast
  forn(i, n) match[i] = -1;
  forn(i, n)
    if (match[i] == -1) {
      int to = augment(i);
      while (to != -1) {
        int next = match[p[to]];
        match[to] = p[to];
        match[p[to]] = to;
        to = next;
      }
    }
  vector<pt> ans;
  forn(i, n) if (match[i] != -1 && i < match[i]) ans.pb(mp(i, match[i]));
  return ans;
}
```

2.10 Gomori-Hu tree

```
void gomoriHu(int ans[N][N]) { // O(n * F) = O(n^2 * m^2)
  static int p[N];
  forn(i, n) p[i] = 0;
  forn(i, n) forn(j, n) ans[i][j] = INF;
  fore(i, 1, n) {
    int f = dinic(i, p[i]);
    fore(j, i + 1, n) if (lev[j] < INF / 2 && p[j] == p[i]) p[j] = i;
    ans[i][p[i]] = ans[p[i]][i] = f;
    forn(j, i) ans[j][i] = ans[i][j] = min(ans[j][p[i]], f);
  }
  forn(i, n) ans[i][i] = 0;
}
```

2.11 Eppstein algorithm

```
template<typename T> vector<T> eppstein(int n, vector<edge> e, vector<pt> q, T
    ans)
{ // call this function with parameter ans equals to 0 with type that you need
  static struct {
    int nt[N];
    int leader(int v) { return nt[v] = (nt[v] == v ? v : leader(nt[v])); }
    void init(int n) { forn(i, n) nt[i] = i; }
    bool merge(int a, int b) {
      a = leader(a), b = leader(b);
      if (a == b) return false;
      if (rand() & 1) nt[a] = b;
      else nt[b] = a;
      return true;
    }
  } dsu;
  static pt perm[M];
  static int u = 0, use[M], epos[M];
  forn(i, sz(e)) perm[i] = mp(e[i].w, i);
  sort(perm, perm + sz(e));
  u++;
  forn(i, sz(q)) use[q[i].x] = u;
  dsu.init(n);
  forn(i, sz(q)) dsu.merge(e[q[i].x].x, e[q[i].x].y);
  vector<int> E1;
  forn(ii, sz(e)) {
```

```
    int i = perm[ii].y;
    if (use[i] != u && dsu.merge(e[i].x, e[i].y)) E1.pb(i), ans += e[i].w;;
  }
  dsu.init(n);
  forn(i, sz(E1)) assert(dsu.merge(e[E1[i]].x, e[E1[i]].y));
  int c = 0; vector<int> num(n);
  forn(i, n) if (dsu.leader(i) == i) num[i] = c++;
  forn(i, n) num[i] = num[dsu.leader(i)];
  dsu.init(c);
  vector<edge> ne;
  forn(ii, sz(e)) {
    int i = perm[ii].y;
    edge nce(num[e[i].x], num[e[i].y], e[i].w);
    if (use[i] == u) epos[i] = sz(ne), ne.pb(nce);
    else if (dsu.merge(num[e[i].x], num[e[i].y])) ne.pb(nce);
  }
  forn(i, sz(q)) q[i].x = epos[q[i].x];
  n = c;
  e = ne;
  if (sz(q) <= 1) {
    forn(i, sz(q)) e[q[i].x].w = q[i].y;
    ans = n == 1 ? ans : (n > 2 || e.empty()) ? INF : (ans +
        min_element(all(e))->w);
    return vector<T> (sz(q), ans);
  }
  int mid = sz(q) >> 1;
  vector<T> a = eppstein(n, e, vector<pt> (q.begin(), q.begin() + mid), ans);
  forn(i, mid) e[q[i].x].w = q[i].y;
  vector<T> b = eppstein(n, e, vector<pt> (q.begin() + mid, q.end()), ans);
  a.insert(a.end(), all(b));
  return a;
}
```

2.12 Goldberg algorithm

The value of mincut is $C(S,T) = |V|W + 2|V_1|\left(mid - \left(\frac{\sum\limits_{i\in V_1} d_i - \sum\limits_{i\in V_1, j\in V_2} w_{ij}}{2|V_1|}\right)\right) =$
$\sum\limits_{j\in V_2} c_{sj} + \sum\limits_{i\in V_1} c_{it} + \sum\limits_{i\in V_1, j\in V_2} c_{ij}$, где $V_1 = S\backslash\{s\}, V_2 = T\backslash\{t\}, d_i = \sum\limits_{j\in V} w_{ij}, W = \sum\limits_{i\in|V_1|, j\in|V_2|} w_{ij}$.

```
vector<int> goldberg(int n, vector<pt> e, vector<int> w)
{ // s/k->max, s - sum of edges among k selected vertexes.
  int W = accumulate(all(w), 0);
  vector<int> d(n, 0);
  forn(i, sz(e)) d[e[i].x] += w[i], d[e[i].y] += w[i];
  // For weighted vertices simply add self-loops.
  vector<int> ans(1, 0); // Replace by forn(t, 100)
  ld lf = 0, rg = W;     // for real weights.
  while ((rg - lf) * n * n - EPS > 1) // n^4 for k * (200 - k)
  { // Recomendation: EPS = 1e-13.
    ld mid = (lf + rg) / 2;
    const int s = n, t = s + 1;
    forn(i, t + 1) fg[i].clear(); // fg - graph for flow.
    forn(i, n)
    { // In place of W one can use any value X so X-d[i]>=0.
      add(s, i, W);           // For k * (200 - k):
      add(i, t, W + 2 * mid - d[i]); // mid -> mid * (200 - n)
    } // Check your function with formulas above the code.
    forn(i, sz(e)) {
      add(e[i].x, e[i].y, w[i]); // For k * (200 - k):
      add(e[i].y, e[i].x, w[i]); // w[i] + 2 * mid
    }

    while (enlarge(s, t)); // Use e.f + EPS < e.c in flow
    vector<int> cur; // cur should contain mincut of s-t
    forn(i, n) if (p[i] != -1) cur.pb(i);
    if (!cur.empty()) ans = cur, lf = mid;
    else rg = mid;
  }
  return ans;
}
```

2.13 Karp algorithm

```
int removeNegativeCycles(int n) {
  int ans = 0;
  static int d[N][N], p[N][N], nt[N];
  while (true) {
    forn(i, n + 1) forn(j, n) d[i][j] = INF * (i > 0);

    fore(i, 1, n + 1)
      forn(v, n)
        if (d[i - 1][v] != INF)
          forn(j, sz(g[v])) {
            edge& e = g[v][j];
            if (e.f < e.c && d[i][e.to] > d[i - 1][v] + e.ct) {
              d[i][e.to] = d[i - 1][v] + e.ct;
              p[i][e.to] = e.rev;
            }
          }

    int num = 1, den = 0, idx = -1;
    forn(i, n) {
      if (d[n][i] == INF) continue;
      int cnum = -1, cden = 0;
      forn(j, n)
        if (cnum * 1ll * (n - j) < cden * 1ll * (d[n][i] - d[j][i]))
          cnum = d[n][i] - d[j][i], cden = n - j;
      if (cden != 0 && cnum < 0 && num * 1ll * cden > den * 1ll * cnum)
        num = cnum, den = cden, idx = i;
    }
    if (idx == -1) break;

    forn(i, n) nt[i] = -1;
    int sv = -1, sk = -1;
    for (int v = idx, k = n; k >= 0; nt[v] = k, v = g[v][p[k][v]].to, k--)
      if (nt[v] != -1) {
        sv = v, sk = nt[v];
        break;
      }
    assert(sv != -1);

    int minv = INF;
    for (int v = sv, k = sk; v != sv || k == sk; k--) {
      edge& e = g[v][p[k][v]];
      minv = min(minv, g[e.to][e.rev].c - g[e.to][e.rev].f);
```

```
      v = e.to;
    }
    for (int v = sv, k = sk; v != sv || k == sk; k--) {
      edge& e = g[v][p[k][v]];
      g[e.to][e.rev].f += minv; e.f -= minv;
      ans += minv * g[e.to][e.rev].ct;
      v = e.to;
    }
  }
  return ans;
}
```

3 Data structures

3.1 Hash map

```
template<int MAX, typename K, typename V>
class hashMap {
  int u, used[MAX];
  bool rem[MAX];
  K key[MAX];
  V val[MAX];
  int hash(K k) { return int(abs(k) % MAX); }
  //int hash(K k) { return int(abs((k * 26544357691l) >> 32) % MAX); }
  int getPos(K k) {
    int pos = hash(k);
    while (used[pos] == u && key[pos] != k) (++pos >= MAX) && (pos -= MAX);
    return pos;
  }
public:
  hashMap() { // please test this code with local declaration of object
    memset(used, 0, sizeof(used)), u = 1;
    memset(rem, 0, sizeof(rem));
  }
  void clear() { u++; }
  V& operator[] (K k) {
    int pos = getPos(k);
    if (used[pos] != u || rem[pos]) val[pos] = 0;
    if (used[pos] != u) used[pos] = u, key[pos] = k;
    rem[pos] = false;
    return val[pos];
  }
  bool count(K k) {
    int pos = getPos(k);
    return used[pos] == u && !rem[pos];
  }
  void erase(K k) {
    int pos = getPos(k);
    if (used[pos] == u) rem[pos] = true;
  }
};

#include <hash_map> // GNU C++ hash_map usage
struct hash_li {
  size_t operator() (const li& x) const { return size_t(x); }
};
__gnu_cxx::hash_map<li, int, hash_li> z(N);
```

3.2 Persistent md-tree

```
struct node {
  li val;
  int add;
  node *lf, *rg;
  node() { val = add = 0, lf = rg = NULL; }
};
typedef node* tree;
int heappos;
node heap[MAX];
inline tree copy(tree t) {
  assert(heappos < MAX);
  tree res = &heap[heappos++];
  if (t) res->val = t->val, res->add = t->add, res->lf = t->lf, res->rg =
      t->rg;
  else res->val = res->add = 0, res->lf = res->rg = NULL;
  return res;
}
inline void push(tree t, int l, int r) {
  t->lf = copy(t->lf);
  t->rg = copy(t->rg);
  int mid = (l + r) >> 1;
  t->lf->val += (mid - l + 1) * 1LL * t->add;
  t->lf->add += t->add;
  t->rg->val += (r - mid) * 1LL * t->add;
  t->rg->add += t->add;
  t->add = 0;
}
inline li getVal(tree t) { return t ? t->val : 0; }
tree inc(tree t, int l, int r, int lf, int rg, int val)
{ // before call this function need to copy root tree!!!
  if (l == lf && r == rg) {
    t->val += (r - l + 1) * 1LL * val;
    t->add += val;
    return t;
  }
  push(t, l, r);
  int mid = (l + r) >> 1;
  if (lf <= mid) t->lf = inc(t->lf, l, mid, lf, min(mid, rg), val);
  if (mid < rg) t->rg = inc(t->rg, mid + 1, r, max(mid + 1, lf), rg, val);
  t->val = getVal(t->lf) + getVal(t->rg);
  return t;
}
li sum(tree t, int l, int r, int lf, int rg, int add) { // this code works
    without coord compression
  if (!t) return (rg - lf + 1) * 1ll * add;
  if (l == lf && r == rg) return t->val + (rg - lf + 1) * 1ll * add;
  int mid = (l + r) >> 1;
  li res = 0;
  if (lf <= mid) res += sum(t->lf, l, mid, lf, min(mid, rg), add + t->add);
  if (mid < rg) res += sum(t->rg, mid + 1, r, max(mid + 1, lf), rg, add +
      t->add);
  return res;
}
```

3.3 Linkcut-tree

```
struct node {
  node *p, *dp, *l, *r;
  int sz;
  int cost, mcost;
  int rev, add;
  node() {
    p = dp = l = r = NULL;
    sz = 1, rev = add = 0;
    cost = mcost = INF;
  }
  void push() {
    if (l) l->add += add, l->cost += add, l->mcost += add, l->rev ^= rev;
    if (r) r->add += add, r->cost += add, r->mcost += add, r->rev ^= rev;
    if (rev) swap(l, r);
    add = rev = 0;
  }
  void lift() {
    sz = 1, mcost = cost;
    if (l) l->p = this, sz += l->sz, mcost = min(mcost, l->mcost);
    if (r) r->p = this, sz += r->sz, mcost = min(mcost, r->mcost);
  }
};
typedef node* tree;
void rotate(tree v) {
  tree p = v->p, gp = p->p;
  if (p->l == v) p->l = v->r, v->r = p;
  else p->r = v->l, v->l = p;
  if (gp) (gp->l == p ? gp->l : gp->r) = v;
  v->p = gp;
  p->lift(), v->lift();
}
void splay(tree v) {
  if (!v) return;
  while (tree p = v->p) {
    tree gp = p->p;
    if (gp) gp->push();
    p->push(), v->push();
    v->dp = gp ? gp->dp : p->dp;
    if (!gp) rotate(v);
    else if ((gp->l == p) == (p->l == v)) rotate(p), rotate(v);
    else rotate(v), rotate(v);
  }
  v->push();
}
tree expose(tree v) {
  tree ans = NULL;
  splay(v);
  if (v->l) v->l->dp = v, v->l->p = NULL;
  v->l = NULL, v->lift();
  while (true) {
    splay(v);
    tree p = v->dp;
    if (!p) break;
    ans = p;
    splay(p);
    if (p->l) p->l->dp = p, p->l->p = NULL;
    p->l = v, p->lift();
  }
  return ans;
} // no need to call splay(v) or push(v) after expose!!!
void evert(tree v) {
  expose(v);
  v->rev ^= 1;
}
void link(tree v, tree w) {
  evert(v);
  v->dp = w;
}
void cut(tree v) {
  splay(v);
  if (v->r) {
    v->r->dp = v->dp;
    v->dp = v->r->p = NULL;
    v->r = NULL, v->lift();
  } else
    v->dp = NULL;
}
tree parent(tree v) {
  splay(v);
  if (!v->r) return v->dp;
  v = v->r, v->push();
  while (v->l) v = v->l, v->push();
  return splay(v), v;
}
tree root(tree v) {
  expose(v);
  while (v->r) v = v->r, v->push();
  return splay(v), v;
}
tree lca(tree a, tree b) {
  expose(b); //       it's need to call evert(root)
  tree ans = expose(a); // before calling lca(a, b)
  if (ans == NULL) ans = a;
  return ans;
}
```

3.4 Splay-tree

```
struct node {
  int key;
  node *p, *l, *r;
  node(int key = 0): key(key) { p = l = r = NULL; }
  void lift() {
    if (l) l->p = this;
    if (r) r->p = this;
  }
};
typedef node* tree;
void rotate(tree v) {
  tree p = v->p, gp = p->p;
  if (p->l == v) p->l = v->r, v->r = p;
  else p->r = v->l, v->l = p;
  if (gp) (gp->l == p ? gp->l : gp->r) = v;
  v->p = gp;
  v->lift();
  p->lift();
```

```
}
void splay(tree v) {
  if (!v) return;
  while (tree p = v->p) {
    tree gp = p ? p->p : p;
    if (!gp) rotate(v);
    else if ((gp->l == p) == (p->l == v)) rotate(p), rotate(v);
    else rotate(v), rotate(v);
  }
}
tree find(tree& v, int key) {
  if (!v) return v;
  while (v->key != key) {
    tree nv = key < v->key ? v->l : v->r;
    if (!nv) return splay(v), nv;
    v = nv;
  }
  return splay(v), v;
}
void split(tree t, int key, tree& t1, tree& t2) {
  if (!t) return void(t1 = t2 = NULL);
  find(t, key);
  if (t->key <= key) {
    t1 = t, t2 = t->r;
    if (t->r) t->r = t->r->p = NULL;
  } else {
    t1 = t->l, t2 = t;
    if (t->l) t->l = t->l->p = NULL;
  }
}
tree merge(tree t1, tree t2) {
  if (!t1) return t2;
  while (t1->r) t1 = t1->r;
  splay(t1);
  t1->r = t2;
  t1->lift();
  return t1;
}
bool insert(tree& t, int key) {
  tree t1, t2;
  split(t, key, t1, t2);
  if (!t1 || t1->key != key) {
    t = new node(key);
    t->l = t1, t->r = t2;
    t->lift();
    return true;
  }
  t = merge(t1, t2);
  return false;
}
bool remove(tree& t, int key) {
  find(t, key);
  if (t && t->key == key) {
    if (t->l) t->l->p = NULL;
    if (t->r) t->r->p = NULL;
    t = merge(t->l, t->r);
    return true;
  }
  return false;
}
```

```
}
```

3.5 Fenwick tree

```
template<typename T, int N> struct fenwick {
  int n;
  T a[N], b[N];
  void assign(int n) {
    this->n = n;
    forn(i, n) a[i] = b[i] = 0;
  }
  void inc(T t[N], int i, T dv) {
    for ( ; i < n; i |= i + 1)
      t[i] += dv;
  }
  T sum(T t[N], int i) {
    T ans = 0;
    for ( ; i >= 0; i = (i & (i + 1)) - 1)
      ans += t[i];
    return ans;
  }
  void inc(int l, int r, T dv) {
    inc(a, l, -(l - 1) * dv);
    inc(a, r + 1, r * dv);
    inc(b, l, dv);
    inc(b, r + 1, -dv);
  }
  T sum(int l, int r) {
    T ans = 0;
    ans += sum(a, r) + r * sum(b, r);
    ans -= sum(a, l - 1) + (l - 1) * sum(b, l - 1);
    return ans;
  }
};
```

3.6 Maximum density subsegment

```
frac calc(int *a, int n) {
  static li s[N];
  forn(i, n) s[i + 1] = s[i] + a[i];
  auto d = [](int i, int j) { return frac(s[j + 1] - s[i], j + 1 - i); };
  static int phi[N];
  int p = 0, q = 0;
  frac res(a[0], 1);
  forn(i, n) {
    while (p + 1 < q && d(phi[q - 2], phi[q - 1] - 1) >= d(phi[q - 2], i - 1))
      q--;
    phi[q++] = i;
    while (p + 1 < q && d(phi[p], phi[p + 1] - 1) <= d(phi[p], i)) p++;
    res = max(res, d(phi[p], i));
  }
  return res;
} // This code is about not tested, also you can try some dp for this problem
```

4 Strings

4.1 Duval algorithm

```
inline vector<int> duval(char* s, int n)
{ // returns Lindon decomposition
  vector<int> ans; // a1>=a2>=...
  // don't forget to make s=s+s
  for (int i = 0; i < n; )
  { // for minimum shift problem
    int j = i, k = j + 1;
    while (k < n && s[k] >= s[j]) {
      if (s[k] > s[j]) j = i;
      else j++;
      k++;
    }
    while (i <= j) ans.pb(i), i += k - j;
  }
  return ans;
}
```

4.2 Palindromes with O(n)

```
inline void preparePalindromes(char* s, int n, int* z1, int* z2) {
  int l = 0, r = -1;
  forn(i, n) { // s[i - z1[i] + 1, i + z1[i] - 1] - max palindrome
    z1[i] = i < r ? min(r - i, z1[l + (r - 1 - i)]) : 0;
    while (i + z1[i] < n && i - z1[i] >= 0 && s[i + z1[i]] == s[i - z1[i]])
        z1[i]++;
    if (i + z1[i] + 1 > r) l = i - z1[i] + 1, r = i + z1[i];
  }
  l = 0, r = -1;
  forn(i, n) { // s[i - z2[i] + 1, i + z2[i]] - max palindrome
    z2[i] = i + 1 < r ? min(r - i - 1, z2[l + (r - 1 - i) - 1]) : 0;
    while (i + z2[i] + 1 < n && i - z2[i] >= 0 && s[i + z2[i] + 1] == s[i -
        z2[i]]) z2[i]++;
    if (i + z2[i] + 2 > r) l = i - z2[i] + 1, r = i + z2[i] + 1;
  }
}
```

4.3 Prefix function

```
inline void calcp(char* s, int n, int* p) {
  p[0] = 0;
  fore(i, 1, n) {
    p[i] = p[i - 1];
    while (p[i] > 0 && s[p[i]] != s[i]) p[i] = p[p[i] - 1];
    p[i] += s[i] == s[p[i]];
  }
}
```

4.4 Suffix array

```
vector<int> suffixArray(char* s, int n) {
  static int perm[N], c[N], cnt[N], tmp[N];
  s[n++] = '!'; // s[i] should be larger than '!'=33
  memset(cnt, 0, sizeof(cnt));

  forn(i, n) cnt[(int)s[i]]++;
  fore(i, 1, N) cnt[i] += cnt[i - 1];
  nfor(i, n) perm[--cnt[(int)s[i]]] = i;

  int comp = 1;
  c[perm[0]] = comp - 1;
  fore(i, 1, n) {
    if (s[perm[i]] != s[perm[i - 1]]) comp++;
    c[perm[i]] = comp - 1;
  }
  for (int h = 0; (1 << h) < n && comp < n; h++) {
    forn(i, n) perm[i] = (perm[i] - (1 << h) + n) % n;

    memset(cnt, 0, sizeof(int) * comp);
    forn(i, n) cnt[c[i]]++;
    fore(i, 1, comp) cnt[i] += cnt[i - 1];
    nfor(i, n) tmp[--cnt[c[perm[i]]]] = perm[i];
    memcpy(perm, tmp, sizeof(int) * n);

    comp = 1;
    tmp[perm[0]] = comp - 1;
    fore(i, 1, n) {
      if (c[perm[i]] != c[perm[i - 1]] || c[(perm[i] + (1 << h)) % n]
        != c[(perm[i - 1] + (1 << h)) % n]) comp++;
      tmp[perm[i]] = comp - 1;
    }
    memcpy(c, tmp, sizeof(int) * n);
  }
  s[n - 1] = 0;
  return vector<int> (perm + 1, perm + n);
}
```

4.5 Suffix automaton

```
struct node {
  bool term; // is node terminal
  int link, len; // suffix link and maximum len
  map<char, int> next; // of string ending in node
  node(int link = -1, int len = 0): link(link), len(len) {
    term = false;
    next.clear();
  }
};
int sz, last;
node t[2 * N];
inline void saExtand(char c) {
  int v = sz++;
  t[v] = node(-1, t[last].len + 1);
  for ( ; last != -1 && !t[last].next.count(c); last = t[last].link)
    t[last].next[c] = v;
  if (last == -1) t[v].link = 0;
  else {
```

```
    int next = t[last].next[c];
    if (t[last].len + 1 == t[next].len) t[v].link = next;
    else {
      int clone = sz++;
      t[clone] = t[next];
      t[clone].len = t[last].len + 1;
      t[next].link = clone;
      t[v].link = clone;
      for ( ; last != -1 && t[last].next[c] == next; last = t[last].link)
        t[last].next[c] = clone;
    }
  }
  last = v;
}
inline void buildAutomaton(char* s, int n) {
  sz = 0;
  t[sz] = node();
  last = sz++;
  forn(i, n) saExtand(s[i]);
  for (int v = last; v != -1; v = t[v].link) t[v].term = true;
}
```

4.6 Suffix tree

```
struct node {
  int l, r;
  int parent, link;
  int next[26]; // ALPH
  node(int l = 0, int r = 0, int parent = 0): l(l), r(r), parent(parent) {
    link = -1;
    memset(next, -1, sizeof(next));
  }
};
struct state {
  int v, pos;
  state(int v = 0, int pos = 0): v(v), pos(pos) { }
};
const int N = 1000 * 1000 + 3;
int n;
char s[N];
int tsz = 1;
node t[2 * N];
state ptr;
inline int len(int v) { return t[v].r - t[v].l; }
inline int split(state st) {
  if (st.pos == 0) return t[st.v].parent;
  if (st.pos == len(st.v)) return st.v;
  int cur = tsz++;
  t[cur] = node(t[st.v].l, t[st.v].l + st.pos, t[st.v].parent);
  t[cur].next[s[t[st.v].l + st.pos] - 'a'] = st.v;
  t[t[st.v].parent].next[s[t[st.v].l] - 'a'] = cur;
  t[st.v].parent = cur;
  t[st.v].l += st.pos;
  return cur;
}
state go(state st, int l, int r) {
  while (l < r) {
    if (st.pos == len(st.v)) {
      st = state(t[st.v].next[s[l] - 'a'], 0);
      if (st.v == -1) return st;
    } else {
      if (s[t[st.v].l + st.pos] != s[l]) return state(-1, -1);
      int d = min(len(st.v) - st.pos, r - l);
      l += d;
      st.pos += d;
    }
  }
  return st;
}
int link(int v) {
  int& ans = t[v].link;
  if (ans != -1) return ans;
  if (v == 0) return ans = 0;
  int p = t[v].parent;
  return ans = split(go(state(link(p), len(link(p))), t[v].l + (p == 0),
      t[v].r));
}
inline void treeExtand(int i) {
  while (true) {
    state next = go(ptr, i, i + 1);
    if (next.v != -1) {
      ptr = next;
      break;
    }
    int mid = split(ptr), cur = tsz++;
    t[cur] = node(i, n, mid);
    t[mid].next[s[i] - 'a'] = cur;
    if (mid == 0) break;
    ptr = state(link(mid), len(link(mid)));
  }
}
```

4.7 z-function

```
inline void calcz(char* s, int n, int* z) {
  int l = 0, r = -1;
  z[0] = 0;
  fore(i, 1, n) {
    z[i] = i < r ? min(r - i, z[i - l]) : 0;
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) z[i]++;
    if (i + z[i] > r) l = i, r = i + z[i];
  }
}
```

4.8 Aho-Korasik algorithm

```
struct node {
    char c;
    int parent, link, output;
    int next[ALPH], automata[ALPH];

    node(char c = -1, int parent = -1, int link = -1, int output = -1):
        c(c), parent(parent), link(link), output(output) {
        memset(next, -1, sizeof(next));
```

```
        memset(automata, -1, sizeof(automata));
    }
};
int sz = 1;
node t[SLEN];
inline int add(const string& s) {
    int v = 0;
    forn(i, sz(s)) {
        if (t[v].next[s[i] - 'a'] == -1) {
            t[v].next[s[i] - 'a'] = sz;
            t[sz++] = node(s[i], v, -1, -1);
        }
        v = t[v].next[s[i] - 'a'];
    }
    t[v].output = v;
    return v;
}
int link(int v) {
    int& ans = t[v].link;
    if (ans != -1) return ans;
    if (t[v].parent <= 0) return ans = 0;
    char c = t[v].c;
    int vv = link(t[v].parent);
    while (vv != 0 && t[vv].next[c - 'a'] == -1)
        vv = link(vv);
    return ans = (t[vv].next[c - 'a'] == -1? 0: t[vv].next[c - 'a']);
}
int output(int v) {
    int& ans = t[v].output;
    if (ans != -1) return ans;
    return ans = (v == 0? 0: output(link(v)));
}
int next(int v, char c) {
    int& ans = t[v].automata[c - 'a'];
    if (ans != -1) return ans;
    if (t[v].next[c - 'a'] != -1)
        return ans = t[v].next[c - 'a'];
    return ans = (v == 0? 0: next(link(v), c));
}
```

5 Number theory

5.1 ax + by = c in rectangle

```
inline li myCeil(li, li);
inline li myFloor(li a, li b) { return b < 0 ? myFloor(-a, -b) : a < 0 ?
    -myCeil(-a, b) : a / b; }
inline li myCeil(li a, li b) { return b < 0 ? myCeil(-a, -b) : a < 0 ?
    -myFloor(-a, b) : (a + b - 1) / b; }
inline li getSolutionCount(li a, li b, li c, li x1, li x2, li y1, li y2) {  //
    a * x + b * y = c
  if (a == 0) {
    if (b == 0) return c == 0 ? (x2 - x1 + 1) * (y2 - y1 + 1) : 0;
    if (c % b != 0) return 0;
    return y1 <= c / b && c / b <= y2 ? (x2 - x1 + 1) : 0;
  }
  if (b == 0) {
    if (c % a != 0) return 0;
    return x1 <= c / a && c / a <= x2 ? (y2 - y1 + 1) : 0;
  }
  li x0, y0, g = gcd(a, b, x0, y0);
  if (c % g != 0) return 0;
  li lx, rx;
  if (b == 0) // x = (x0 * c + k * b) / g
    lx = -INF64, rx = INF64;
  else if (b / g > 0)
    lx = myCeil(x1 * g - x0 * c, b), rx = myFloor(x2 * g - x0 * c, b);
  else
    lx = myCeil(x2 * g - x0 * c, b), rx = myFloor(x1 * g - x0 * c, b);
  li ly, ry;
  if (a == 0) // y = (y0 * c - k * a) / g
    ly = -INF64, ry = INF64;
  else if (-a / g > 0)
    ly = myCeil(y0 * c - y1 * g, a), ry = myFloor(y0 * c - y2 * g, a);
  else
    ly = myCeil(y0 * c - y2 * g, a), ry = myFloor(y0 * c - y1 * g, a);
  return max(0ll, min(rx, ry) - max(lx, ly) + 1);
}
```

5.2 Inverse elements with O(n)

```
inline void prepareInverse(int p) {
  inv[1] = 1;
  fore(i, 2, p) inv[i] = (p - p / i * 1ll * inv[p % i] % p) % p;
}
```

5.3 Miller-Rabin test

```
inline li mul(li a, li b, li mod) {
  li q = li(ld(a) * b / mod);
  li ans = (a * b - q * mod) % mod;
  return ans < 0 ? ans + mod : ans;
} // it's need to check small n and small p
inline bool isProbablePrime(li n, int cnt) {
  if (n <= 1) return false;
  forn(t, cnt) {
    li a = myRand() % (n - 1) + 1;
    li b = n - 1;
    int cnt = 0;
    while (!(b & 1)) b >>= 1, cnt++;
    li ans = binPow(a, b, n); // need to use mul(a, b, n)
    forn(i, cnt) {
      li next = mul(ans, ans, n);
      if (next == 1 && ans != 1 && ans != n - 1) return false;
      ans = next;
    }
    if (ans != 1) return false;
  }
  return true;
}
```

5.4 Minimum divisors with O(n)

```
int szp, p[N];
int mind[N];
inline void prepareDivisors() {
  szp = 0;
  mind[1] = -1; // ?????????? -1 <-> 1
  fore(i, 2, N) {
    if (mind[i] == 0) mind[i] = i, p[szp++] = i;
    for (int j = 0; j < szp && p[j] <= mind[i] && i * p[j] < N; j++)
      mind[i * p[j]] = p[j];
  }
}
```

5.5 Pollard algorithm

```
inline vector<li> pollard(li n) { // don't forget about srand :-)
  if (n == 1) return vector<li> (0); // Complexity is ~O(n^{2.57})
  if (isProbablePrime(n, 13)) return vector<li> (1, n);
  for (li d = 2; d < 100; d++) // hacks here are meaningless
    if (n % d == 0) {
      vector<li> ans = pollard(d), rg = pollard(n / d);
      forn(i, sz(rg)) ans.pb(rg[i]);
      return ans;
    } // try to use M=3, but not M=1 or M=2!!!
  const int M = 5; // with M=3 it factorizes 50 numbers ~10^18
  static li q[M], x[M], y[M]; // in 3.4s, for M=5: in 4.2s
  forn(i, M) q[i] = myRand() % n, x[i] = y[i] = 2;
  q[0] = 1;
  for (int i = 1, k = 1; ; i++) {
    forn(t, M) {
          li d = gcd(abs(y[t] - x[t]), n);
        if (d != 1 && d != n) {
          vector<li> ans = pollard(d), rg = pollard(n / d);
          forn(i, sz(rg)) ans.pb(rg[i]);
          return ans;
        }
        x[t] = (mul(x[t], x[t], n) + q[t]) % n;
        if (i == k) y[t] = x[t];
      }
      if (i == k) k *= 2;
    }
}
```

5.6 Rational binary search

```
frac solve(frac lf, frac rg, int start, li maxDenom) {
  li p0 = 0, p1 = 1, q0 = 1, q1 = 0;
  frac last = start ? lf : rg;
  for (int it = start; ; it ^= 1) {
    li a = 1;
    while (true) {
      frac cur = lf + frac(a * p1 + p0, a * q1 + q0);
      if (cur.denom > 2 * maxDenom) return last;
      int iscan = can(cur);
      if (it ^ iscan) break;
      if (iscan) last = cur;
      a <<= 1;
    }
    li b = a;
    a >>= 1;
    while (b - a != 1) {
      li mid = (a + b) >> 1;
      frac cur = lf + frac(mid * p1 + p0, mid * q1 + q0);
      if (it ^ can(cur)) b = mid; else a = mid;
    }
    li v = it ? a : b;
    last = lf + frac(v * p1 + p0, v * q1 + q0);
    li op1 = p1, oq1 = q1;
    p1 = a * p1 + p0, q1 = a * q1 + q0;
    p0 = op1, q0 = oq1;
  }
}
```

5.7 Shanks algorithm for discrete logarithm

```
int solve(int a, int b, int m)
{ // a^x=b (mod m) - returns minimum solution
  int n = int(sqrt(m) + 1), cur = 1, an = 1;
  forn(i, n) an = int((an * 1ll * a) % m);
  map<int, vector<int> > z;
  fore(p, 1, n + 1) {
    cur = int((cur * 1ll * an) % m);
    z[cur].pb(p);
  }
  cur = b;
  int ans = m + 1;
  forn(q, n + 1) {
    if (z.count(cur)) {
      vector<int>& val = z[cur];
      forn(i, sz(val)) {
        int cur = n * val[i] - q;
        if (cur < m && binPow(a, cur, m) == b)
          ans = min(ans, cur);
      }
    }
    cur = int((cur * 1ll * a) % m);
  }
  return ans > m ? -1 : ans;
}
```

5.8 Primitive roots

```
inline li primitiveRoot(li n)
{ // There exists primitivev roots only for:
  if (n <= 4) return n - 1;
  // 2, 4, p^k, 2*p^k, where p is odd
  vector<pair<li, int> > f = factorize(n % 2 ? n : n / 2);
  li p = f[0].ft, a = f[0].sc;
  if (sz(f) != 1 || p % 2 == 0) return -1;
  li pa = 1; forn(i, a - 2) pa *= p;
  vector<pair<li, int> > phif = factorize(p - 1);
```

```
  //li ans = INF64; - if need to find the smallest
  for (li i = 2; i < p; i++) {
    li g[] = { i, (i + p) % n };
    //if (i > ans) break;
    forn(t, 2) {
      forn(j, sz(phif))
        if (binPow(g[t], (p - 1) / phif[j].ft, p) == 1)
          goto done;

      if (a >= 2 && binPow(g[t], pa * (p - 1), n) == 1) continue;
      if (n % 2 == 0 && g[t] % 2 == 0) continue;
      //ans = min(ans, g[t]);
      return g[t];
      done:;
    }
  }
  //return ans;
  throw;
}
```

5.9 Eratosthenes with blockes

```
int u = 0, used[SQRTX];
int szp, p[PCNT];
int szp2, p2[SQRTX];
int MAXP = 100 * 1000 * 1000 + 1;
inline void preparePrimes() { // finds all primes in [2, MAXP)
  szp = szp2 = 0;
  fore(i, 2, SQRTX)
    if (!used[i]) {
      p[szp++] = p2[szp2++] = i;
      for (int j = i * i; j < SQRTX; j += i) used[j] = true;
    }
  u = 0;
  memset(used, 0, sizeof(used));
  for (int i = SQRTX; i < MAXP; i += SQRTX) {
    int lf = i, rg = min(lf + SQRTX, MAXP);
    u++;
    forn(j, szp2) {
      int ft = (lf + p2[j] - 1) / p2[j];
      for (int k = ft * p2[j]; k < rg; k += p2[j]) used[k - lf] = u;
    }
    forn(j, rg - lf) if (used[j] != u) p[szp++] = lf + j;
  }
}
```

5.10 2 equations, 3 variables (Diofant system)

```
bool solve(li a[2][4], li x[3]) {
  forn(i, 3) x[i] = 0;
  li gg[2] = { 0 };
  forn(i, 2) forn(j, 3) gg[i] = gcd(gg[i], a[i][j]);
  if (gg[0] == 0) {
    if (a[0][3] != 0) return false;
    forn(i, 4) swap(a[0][i], a[1][i]);
    swap(gg[0], gg[1]);
  }
  if (gg[0] == 0) return a[0][3] == 0; // Case 1, 2
  int i0 = 0, i1 = 1, i2 = 2;
  if (a[0][i0] == 0) swap(i0, i1);
  if (a[0][i0] == 0) swap(i0, i2);
  assert(a[0][i0]);
  li x0, y0, xx0, yy0;
  li g = gcd(a[0][i0], a[0][i1], x0, y0);
  li h = gcd(g, a[0][i2], xx0, yy0);
  assert(g != 0);
  assert(h != 0);
  if (a[0][3] % h) return false; // Case 3
  if (gg[1] == 0) {
    x[i0] = (a[0][3] / h) * x0 * xx0; // + k * a[0][1] * a[0][3] / (g * h) + l
        * x0 * a[0][2] / h
    x[i1] = (a[0][3] / h) * y0 * xx0; // - k * a[0][0] * a[0][3] / (g * h) + l
        * y0 * a[0][2] / h
    x[i2] = (a[0][3] / h) * yy0;      // - l * g / h
    forn(i, 2) assert(a[i][0] * x[0] + a[i][1] * x[1] + a[i][2] * x[2] ==
        a[i][3]);
    return true; // Case 4
  }
  li A = (a[0][3] / h) * ((a[0][i1] * a[1][i0] - a[0][i0] * a[1][i1]) / g) *
      xx0;
  li B = (a[0][i2] * (x0 * a[1][i0] + y0 * a[1][i1]) - g * a[1][i2]) / h;
  li C = a[1][3] - (a[0][3] / h) * (xx0 * (x0 * a[1][i0] + y0 * a[1][i1]) +
      yy0 * a[1][i2]);
  li X, Y, G = gcd(A, B, X, Y);
  if (G == 0 && C != 0) return false; // Case 5
  if (G != 0 && C % G) return false; // Case 6
  li k, l;
  if (G == 0)
    k = l = 0;
  else {
    k = X * C / G; // + p * B / G
    l = Y * C / G; // - p * A / G
  }
  x[i0] = (a[0][i1] / g) * (a[0][3] / h) * xx0 * k + (a[0][i2] / h) * x0 * l
      + (a[0][3] / h) * x0 * xx0;
  x[i1] = -(a[0][i0] / g) * (a[0][3] / h) * xx0 * k + (a[0][i2] / h) * y0 * l
      + (a[0][3] / h) * y0 * xx0;
  x[i2] = -(g / h) * l + (a[0][3] / h) * yy0;
  forn(i, 2) assert(a[i][0] * x[0] + a[i][1] * x[1] + a[i][2] * x[2] ==
      a[i][3]);
  return true; // Case 7
} /////// Please test this code on some problems!!! ///////
```

6 Other

6.1 Alpha-beta pruning

```
int solve1(int, int, int);
int solve2(int a, int b, int cnt) {
  if (game is ended ...) return cnt;
  bool exist = false;
  forn(i, k)
    if (can make move ...) {
```

```
      exist = true;
      generate new state ...
      b = min(b, solve1(a, b, cnt + 1));
      return to old state ...
      if (b <= a) return b;
    }
  return exist ? b : cnt;
}
int solve1(int a, int b, int cnt) {
  if (game is ended ...) return cnt;
  bool exist = false;
  forn(i, k)
    if (can make move ...) {
      exist = true;
      generate new state ...
      a = max(a, solve2(a, b, cnt));
      return to old state ...
      if (a >= b) return a;
    }
  return exist ? a : cnt;
}
```

6.2 Fast Fourier transform

```
int reverse(int x, int logn) { // use own complex!!!
  int ans = 0; // DON'T USE LONG DOUBLE, IT'S TOO SLOW
  forn(i, logn) if (x & (1 << i)) ans |= 1 << (logn - 1 - i);
  return ans;
}

const int LOGN = 16, N = (1 << LOGN) + 3;
void fft(base a[N], int n, bool inv) {
  static base vv[LOGN][N];
  static bool prepared = false;
  if (!prepared) {
    prepared = true;
    forn(i, LOGN) {
      ld ang = 2 * PI / (1 << (i + 1));
      forn(j, 1 << i) vv[i][j] = base(cos(ang * j), sin(ang * j));
    }
  }
  int logn = 0; while ((1 << logn) < n) logn++;
  forn(i, n) {
    int ni = reverse(i, logn);
    if (i < ni) swap(a[i], a[ni]);
  }
  for (int i = 0; (1 << i) < n; i++)
    for (int j = 0; j < n; j += (1 << (i + 1)))
      for (int k = j; k < j + (1 << i); k++) {
        base cur = inv ? conj(vv[i][k - j]) : vv[i][k - j];
        base l = a[k], r = cur * a[k + (1 << i)];
        a[k] = l + r;
        a[k + (1 << i)] = l - r;
      }
  if (inv) forn(i, n) a[i] /= ld(n);
}

void mul(li a[N], int n, li b[N], int m, li ans[N]) {
  static base fp[N], p1[N], p2[N];
  while (n && !a[n - 1]) n--;
  while (m && !b[m - 1]) m--;
  int cnt = n + m;
  while (cnt & (cnt - 1)) cnt++;

  forn(i, cnt) fp[i] = base(i < n ? a[i] : 0, i < m ? b[i] : 0);
  fft(fp, cnt, false); // one can simply rework it
  forn(i, cnt) {       // with two calls of fft for p1, p2
    p1[i] = (fp[i] + conj(fp[(cnt - i) % cnt])) / base(2, 0);
    p2[i] = (fp[i] - conj(fp[(cnt - i) % cnt])) / base(0, 2);
  }
  forn(i, cnt) fp[i] = p1[i] * p2[i];
  fft(fp, cnt, true);

  forn(i, cnt) ans[i] = li(fp[i].real() + 0.5);
}

void mul(int* a, int* b, int n, const int mod) {
  int smod = int(sqrt((ld) mod));
  static li a1[N], a2[N], b1[N], b2[N], za[N], zb[N];
  forn(i, n) {
    a1[i] = a[i] % smod, a2[i] = a[i] / smod;
    b1[i] = b[i] % smod, b2[i] = b[i] / smod;
    za[i] = a1[i] + a2[i];
    zb[i] = b1[i] + b2[i];
  }
  mul(a1, n, b1, n, a1);
  mul(a2, n, b2, n, a2);
  mul(za, n, zb, n, za);
  forn(i, 2 * n) {
    li cur = a1[i] % mod;
    cur += a2[i] % mod * smod * smod % mod;
    cur += (za[i] - a1[i] - a2[i]) % mod * smod % mod;
    cur %= mod;
    a[i] = int(cur < 0 ? (cur + mod) : cur);
  }
}
```

6.3 Fast Fourier transform integral

```
namespace fftMod {
  typedef int T;
  // Here should be mul, add, sub, inv and binPow implementations
  int logn, n;
  T p, g;
  int get(int i) {
    int ans = 0;
    forn(j, logn) if (i & (1 << j)) ans |= 1 << (logn - 1 - j);
    return ans;
  }
  void fft(T *a, bool inverse) {
    forn(i, n) {
      int ni = get(i);
      if (i < ni) swap(a[i], a[ni]);
    }
    for (int i = 0; i < logn; i++) {
      T cg = binPow(g, 1 << (logn - (i + 1)), p);
```

```
      assert(binPow(cg, 1 << (i + 1), p) == 1);
      assert(binPow(cg, 1 << i, p) != 1);
      if (inverse) cg = inv(cg, p);
      for (int j = 0; j < n; j += (1 << (i + 1))) {
        T cur = 1;
        for (int k = 0; k < (1 << i); k++) {
          T u = a[j + k], v = mul(a[j + (1 << i) + k], cur, p);
          a[j + k] = add(u, v, p);
          a[j + (1 << i) + k] = sub(u, v, p);
          cur = mul(cur, cg, p);
        }
      }
    }
    if (inverse) forn(i, n) a[i] = mul(a[i], inv(n, p), p);
  }
  // p                    | deg | g
  // 469762049              26    3
  // 998244353              23    3
  // 1107296257             24    10
  // 10000093151233         26    5
  // 1000000523862017       26    3
  // 1000000000846594049    26    3
  // 1000000000949747713    26    5
  void prepare() {
    /*n = 1 << 26;
    T c = li(1e17) / n + 1;
    while (!isPrime(c * n + 1)) c++;
    p = c * n + 1;
    g = primitiveRoot(p);*/
    p = 469762049; // You can use your own p = c * 2^logn + 1
    g = 3;         // but you should find it's generator
    assert(!((p - 1) & (n - 1)));
    T cs = (p - 1) >> logn;
    g = binPow(g, cs, p);
    assert(binPow(g, n, p) == 1);
  }
  void mul(T *a, int na, T *b, int nb, T *ans) {
    while (na && !a[na - 1]) na--;
    while (nb && !b[nb - 1]) nb--;
    logn = 0; while ((1 << logn) < na + nb) logn++;
    n = 1 << logn;
    static T _b[N];
    forn(i, n) ans[i] = a[i], _b[i] = b[i];
    prepare();
    fft(ans, false);
    fft(_b, false);
    forn(i, n) ans[i] = mul(ans[i], _b[i], p);
    fft(ans, true);
  }
}
```

6.4 Subset with maximum xor

```
bitset<N> z[LOGN];
int num[LOGN], is[N];
inline pair<vector<int>, int> calcMaxSubset(const vector<li>& a) {
  int n = sz(a), cnt = 0;
  nfor(i, LOGN) {
    forn(j, n) z[cnt][j] = (a[j] >> i) & 1;
    z[cnt][n] = 1;
    forn(j, cnt) if (z[cnt][num[j]]) z[cnt] ^= z[j];
    for (num[cnt] = 0; num[cnt] < n; num[cnt]++)
      if (z[cnt][num[cnt]]) {
        cnt++;
        break;
      }
  }
  forn(i, n) is[i] = 0;
  nfor(i, cnt) {
    is[num[i]] = z[i][n];
    fore(j, i + 1, cnt) is[num[i]] ^= is[num[j]] * z[i][num[j]];
  }
  vector<int> ans;
  forn(i, n) if (is[i]) ans.pb(i);
  // second parameter is the number of free variables
  return mp(ans, n - cnt); // so answer count is 2^sc
}
```

6.5 Sum of floor (n divide i)

```
inline li sum(li n) {
  li ans = 0;
  li minVal = n;
  for (li i = 1; i * i <= n; i++) {
    li lf = n / (i + 1), rg = n / i;
    minVal = lf; // interval (lf, rg]
    ans += (rg - lf) * i;
  }
  fore(i, 1, minVal + 1) ans += n / i;
  return ans;
}
```

6.6 Gray code

```
int g(int n) { return n ^ (n >> 1); }
int invG(int g) {
  int n = 0;
  while (g) n ^= g, g >>= 1;
  return n;
}
```

6.7 Features

```
int days(int y, int m, int d) { // number of days from 1 january 0 (1-indexed)
  if (m < 2) y--, m += 12;
  return 365 * y + y / 4 - y / 100 + y / 400 + (153 * m + 1) / 5 + d - (y ==
      -1);
}
for (int mask = (1 << k) - 1; mask < (1 << n); )
{ // all masks with n bits and k ones, add yout code before if
  if (mask == 0) break;
  int x = mask & -mask, y = mask + x;
  mask = (((mask & ~y) / x) >> 1) | y;
```

```
}

int __builtin_ffs (int x) - returns one plus the index of the least
    significant 1-bit of x, or if x is zero, returns zero.
int __builtin_clz (unsigned int x) - returns the number of leading 0-bits in
    x, starting at the most significant bit position. If x is 0, the result is
    undefined.
int __builtin_ctz (unsigned int x) - returns the number of trailing 0-bits in
    x, starting at the least significant bit position. If x is 0, the result is
    undefined.
int __builtin_popcount (unsigned int x) - returns the number of 1-bits in x.
int __builtin_parity (unsigned int x) - returns the parity of x, i.e. the
    number of 1-bits in x modulo 2.

Bitset functions: size_t _Find_first(), size_t _Find_next(size_t i)

BigDecimal v = ONE.divide(BigDecimal.valueOf(3), 10, ROUND_HALF_UP); // 10
    signs after the point
v = v.setScale(3, ROUND_HALF_UP);     // 3 signs after the point
v = v.multiply(TWO, new MathContext(4)); // The total number of signs is 4
v = v.multiply(TWO);                  // All the signs will remain

#include <ext/rope>
using namespace __gnu_cxx;
rope <char> v;
v.push_back('0');
v.append("2");
v.append(10, '3');
v.append(v);
v.append(v.mutable_begin() + 1, v.mutable_begin() + 5);
/* insert before element with index (first arg) */
v.insert(0, 10, '4');
v.insert(0, v);
/* assignment */
v.mutable_reference_at(5) = 'a';
v.replace(5, 'b');
/* but constant access */
cerr << v[4] << endl;
cerr << v.length() << endl;
v = v + "abacaba";
/* like substr in std::string */
v.substr(3, 10);
v.replace(3, 10, v);
v.erase(3, 10);

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
  tree_order_statistics_node_update> ordered_set;
/* iterator like in random access */
X.find_by_order(2);
/* index of lower_bound element */
X.order_of_key(5);
```

6.8 Simplex method

```
namespace Simplex
{       // A * x <= B, C * x -> max, x >= 0
  int n, m; // dual: A^t * y >= C, B * y -> min, y >= 0
  const int NM = N + M + 1;
  int pos[NM]; // N - number of variables, M - inequalities
  ld a[M][NM], b[NM], c[NM], oc[NM], x[NM];
  void push(int e, int n) {
    forn(i, n + m) c[i] -= a[pos[e]][i] * c[e];
    c[e] = 0;
  }
  void pivot(int e, int l, int n) {
    a[pos[l]][l] = 1;
    forn(i, n + m) if (i != e) a[pos[l]][i] /= a[pos[l]][e];
    b[pos[l]] /= a[pos[l]][e];
    a[pos[l]][e] = 0;
    swap(pos[l], pos[e]);
    forn(i, m) {
      forn(j, n + m) a[i][j] -= a[i][e] * a[pos[e]][j];
      b[i] -= a[i][e] * b[pos[e]];
      a[i][e] = 0;
    }
    push(e, n);
  }
  int iterate(int n) {
    while (true) {
      int e = int(max_element(c, c + n + m) - c);
      if (c[e] < EPS) break;
      assert(pos[e] == -1);
      ld lval = 1e100; int l = -1;
      forn(i, n + m)
        if (pos[i] != -1) {
          ld val = a[pos[i]][e] > EPS ? b[pos[i]] / a[pos[i]][e] : 1e100;
          if (lval > val) lval = val, l = i;
        }
      if (lval > 5e99) return 1;
      pivot(e, l, n);
    }
    forn(i, n + m) x[i] = pos[i] != -1 ? b[pos[i]] : 0;
    return 0;
  }
  void init(int _n, int _m, ld _a[M][N], ld _b[M], ld _c[N]) {
    n = _n, m = _m;
    forn(i, m) forn(j, n + m + 1) a[i][j] = j < n ? _a[i][j] : 0;
    forn(i, n + m + 1) oc[i] = 0, pos[i] = -1;
    forn(i, m) b[i] = _b[i], pos[n + i] = i;
    forn(i, n) oc[i] = _c[i];
  }
  int solve(ld ansx[N]) { // -1 - no solution, 1 - inf solution, 0 - usual case
    int k = int(min_element(b, b + m) - b);
    if (b[k] < -EPS) {
      forn(i, n + m + 1) c[i] = i == n + m ? -1 : 0;
      forn(i, m) a[i][n + m] = -1;
      pivot(n + m, n + k, n + 1);
      assert(iterate(n + 1) == 0);
      if (x[n + m] > EPS) return -1;
      int& r = pos[n + m];
      if (r != -1) {
        int p = 0; forn(i, n + m) if (abs(a[r][p]) < abs(a[r][i])) p = i;
        assert(abs(a[r][p]) > EPS);
        pivot(p, n + m, n + 1);
```

```
        assert(r == -1);
      }
    }
    forn(i, n + m) c[i] = oc[i];
    forn(i, n + m) if (pos[i] != -1) push(i, n);
    int ans = iterate(n); forn(i, n) ansx[i] = x[i];
    return ans;
  }
}
```

6.9 Joseph problem

```
inline int joseph(int n, int k)
{// returns 0-indexed number of survivor
 // count k, beginning from 0, remove, repeat from the next to the removed
  if (n == 1) return 0;
  if (k == 1) return n - 1;
  if (k > n) return (joseph(n - 1, k) + k) % n;
  int cnt = n / k;
  int res = joseph(n - cnt, k);
  res -= n % k;
  if (res < 0) res += n;
  else res += res / (k - 1);
  return res;
}
```

6.10 Knight distance

```
inline li dist(li x1, li y1, li x2, li y2) {
  li dx = abs(x2 - x1);
  li dy = abs(y2 - y1);
  li ans = (dx + 1) >> 1;
  ans = max(ans, (dy + 1) >> 1);
  ans = max(ans, (dx + dy + 2) / 3);
  while ((ans & 1) != ((dx + dy) & 1)) ans++;
  if (abs(dx) == 1 && dy == 0) return 3;
  if (abs(dy) == 1 && dx == 0) return 3;
  if (abs(dx) == 2 && abs(dy) == 2) return 4;
  return ans;
}
```

6.11 Java template

```
import java.io.*;
import java.util.*;
import static java.math.BigInteger.*;

public class Template implements Runnable {
  public void solve() throws Exception {
  }

  public static void main(String[] argv) throws Exception {
        new Thread(null, new Template(), "", 256000000).start();
  }

  BufferedReader in;
  BufferedWriter out;
  StringTokenizer st;

  public String nextToken() throws Exception {
    while (true) {
      if (st == null || !st.hasMoreTokens()) {
        st = new StringTokenizer(in.readLine());
      } else {
        return st.nextToken();
      }
    }
  }

  public int nextInt() throws Exception {
    return Integer.parseInt(nextToken());
  }

  public long nextLong() throws Exception {
    return Long.parseLong(nextToken());
  }

  public double nextDouble() throws Exception {
    return Double.parseDouble(nextToken());
  }

  public void run() {
    try {
      //in = new BufferedReader(new FileReader(new File("input.txt")));
      in = new BufferedReader(new InputStreamReader(System.in));
      //out = new BufferedWriter(new FileWriter(new File("output.txt")));
      out = new BufferedWriter(new OutputStreamWriter(System.out));
      solve();
      out.close();
      in.close();
    } catch (Exception e) {
      throw new RuntimeException(e);
    }
  }

  public void print(Object... a) throws Exception {
    for (int i = 0; i < a.length; i++) {
      out.write(String.valueOf(a[i]));
    }
  }

  public void println(Object... a) throws Exception {
    for (int i = 0; i < a.length; i++) {
      out.write(String.valueOf(a[i]));
    }
    out.newLine();
  }

  public void println() throws Exception {
    out.newLine();
  }
}
```

6.12 .vimrc

```
set noswf
set ai
set nomagic
filetype plugin indent on
set sw=4
set ts=4
syntax on
set ar
set nu
set backspace=indent,eol,start
set so=10
map <F3> gT
map <F4> gt
nmap <F2> :w<CR>
nmap <F7> :w<CR>:!g++ -pg -Wno-deprecated -Wno-deprecated-declarations -Wall
    -Wextra -Wconversion -lm -x c++ -DSU1 -O2 -pthread -std=c++1y -o %:r %<CR>
nmap <F9> :!./%:r<CR>
:command! -nargs=1 R .,$s<args>/gc

" console
gsettings set org.gnome.settings-daemon.peripherals.keyboard delay 150
" .bashrc
trap 'pwd > ~/.lastdir' EXIT
[ -s ~/.lastdir ] && cd "`cat ~/.lastdir`"
```

6.13 Stuff

```
#undef NDEBUG
#ifdef SU2_PROJ
#define _GLIBCXX_DEBUG
#endif

g++ -Wall -Wextra -Wconversion -static -fno-optimize-sibling-calls
    -fno-strict-aliasing -lm -s -x c++ -Wl,--stack=8388608 -DSU2_PROJ -O2 -o !
    !.!
javac -cp ".;*" !.!
java.exe -Xmx256M -Duser.language=en -Duser.region=US -Duser.variant=US !

ls -la /*info about file*/, top /*process list*/, diff /*fc*/
```

7 Комбинаторика

Разбиение числа n на k слагаемых:
$C_{n+k-1}^{k-1} = C_{n+k-1}^{n}$ — для $a_i \geq 0$, C_{n-1}^{k-1} — для $a_i \geq 1$, C_{n-k-1}^{k-1} — для $a_i \geq 2$.
Биномиальные коэффициенты:
Способы подсчёта $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$:

1. $\binom{n}{k}$ можно получить из $\binom{n}{0}$ за $O(k)$, с помощью формулы $\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$.
2. По простому (маленькому) модулю можно считать с помощью теоремы Люка:
   $\binom{m}{n} \equiv \prod_{i=0}^{k-1} \binom{m_i}{n_i} \ (mod\ p),\ p$ — простое. $m = (m_{k-1}, \ldots, m_0)_p, n = (n_{k-1}, \ldots, n_0)_p$ — представление чисел m и n в p-ичной системе счисления.
3. Предподсчитать факториалы за $O(n+m)$. При $n, k \sim 10^9$ можно предподсчитать в коде каждый 10^6-й факториал.
4. Можно для каждого простого найти степень, с которой оно входит в $\binom{n}{k}$ и далее получить само число.
5. Если $n \sim 10^{18}$, а $k \sim 10^6$ — маленькое, то $\binom{n}{k}$ по простому модулю (в случае составного надо использовать КТО) можно искать с помощью факториального представления: рекурсивная функция считает произведение по всем некратным p, по кратным вычисляет степень p и далее рекурсивно вызывает себя (поскольку кратные без коэффициента p образуют новые факториалы).

$\overline{C}_n^k = C_{n+k-1}^k$ — число сочетаний с повторениями из n по k,
$A_n^k = \frac{n!}{(n-k)!} = C_n^k\, k!$ — количество размещений из n по k,
$\sum_{k=0}^{n} kC_n^k = n2^{n-1},\ \sum_{k=0}^{n} k^2C_n^k = (n+n^2)2^{n-2},\ \sum_{k=0}^{n} (C_n^k)^2 = C_{2n}^n$,
$\sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} C_{n-k}^k = F(n+1)$, где $F(n)$ — n-ое число Фибоначчи,
$\sum_{k \leq n} \binom{r+k}{k} = \binom{r+n+1}{n}, n \in \mathbb{Z}$ — параллельное суммирование,
$\sum_{0 \leq k \leq n} \binom{k}{m} = \binom{n+1}{m+1}, m, n \in \mathbb{Z}, m, n \geq 0$ — верхнее суммирование,
$\sum_{k} \binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}, n \in \mathbb{Z}$ — свёртка Вандермонда.

Числа Каталана: $C_n = \dfrac{1}{n+1}\binom{2n}{n}$.
Числа Стирлинга:
1-го рода: $Z_n^k = (n-1)Z_{n-1}^k + Z_{n-1}^{k-1}$ — число способов разбиения множества из n элементов на k циклов (циклы нельзя переворачивать, только прокручивать).

2-го рода: $S_n^k = kS_{n-1}^k + S_{n-1}^{k-1} = \frac{1}{k!}\sum_{j=0}^{k} (-1)^{k+j}\binom{k}{j} j^n$ — число способов разбиения множества из n элементов на k непустых подмножеств.
Полагаем, что $Z_n^0 = S_n^0 = [n = 0], Z_0^k = S_0^k = [k = 0]$. Дуальность: $S_n^k = Z_{-k}^{-n}$.
$x^n = \sum_k S_n^k x^{\underline{k}},\ x^{\overline{n}} = \sum_k Z_n^k x^k,\ x^n = \sum_k S_n^k (-1)^{n-k} x^{\overline{k}},\ x^{\underline{n}} = \sum_k Z_n^k (-1)^{n-k} x^k, n \in \mathbb{Z}, n \geq 0$.
Количество неубывающих последовательностей из n элементов от 0 до a равно $\binom{n+a}{n}$.
Перестановки: $(a \cdot b)[i] = b[a[i]]; a^{-1}[a[i]] = i$.

Перестановки без неподвижных точек: $cnt = n! \sum_{i=0}^{n} \frac{(-1)^n}{i!} \approx \frac{n!}{e}$

Число Непера: $e = \sum_{n=0}^{\infty} \frac{1}{n!}, \frac{1}{e} = \sum_{n=2}^{\infty} \frac{(-1)^n}{n!}$
Количество диаграмм Юнга
Крюк клетки — она сама, а также клетки, расположенные справа от нее, и клетки, расположенные снизу.
Количество заполнений таблицы равно факториалу количества ее клеток, деленному на произведение длин всех крюков.
Теорема Каммера:
$n, m \in Z$, $n \geq m \geq 0$ и p — простое число, тогда максимальная степень k, что $p^k | C_n^m$ равна количеству переносов при сложении чисел m и $n-m$ в системе счисления p.

8 Теория чисел

Обращение Мебиуса

$g(m) = \sum_{d|m} f(d) \leftrightarrow f(m) = \sum_{d|m} \mu(d) \cdot g(\frac{m}{d})$, где $\mu(d) = \begin{cases} 0, \text{если d не свободно от квадратов;} \\ (-1)^k, \text{k — количество простых в разложении d.} \end{cases}$

Сумма меньших взаимнопростых: $\varphi_1(n) = \frac{n\varphi(n)}{2} = \frac{\varphi(n^2)}{2}$
Китайская теорема об остатках (КТО):
$x \equiv rm_i\ (mod\ m_i)\ (i = \overline{0, n-1})$
Ответ ищем в виде: $x = x_0 + x_1m_0 + \ldots + x_{n-1}m_0m_1 \cdot \ldots \cdot m_{n-2}$
$x_i \leftarrow rm_i$, for $j < i$ do $x_i \leftarrow (x_i - x_j)m_j^{-1}\ (mod\ m_i)$

Вычисление за линейное время: $x \equiv \sum_{i=0}^{n-1} rm_i \frac{M}{m_i} y_i (mod\ M)$, где $M = \prod_{i=0}^{n-1} m_i$ и $y_i \equiv \left(\frac{M}{m_i}\right)^{-1} (mod\ m_i)$.

Расширенный КТО: Заданы числа n, m, a, b, нужно найти число $x \equiv a\ (mod\ n), x \equiv b\ (mod\ m)$. Пусть $g = gcd(n, m)$. Если $a \not\equiv b\ (mod\ g)$, то решения не существует. Пусть $nx_0 + my_0 = g$, тогда $x = \left(\frac{n}{g}x_0a + \frac{m}{g}y_0b\right)\ mod\ lcm(n, m)$ является наименьшим решением.
Решение $x \equiv a^N (mod\ m)$, если $\gcd(a, m) > 1$: Пусть $g = gcd(a^N, m)$, найдем наименьшее k, что $g | a^k$, так что $a^k = a_1 g$, $m = m_1 g$. Тогда $x \equiv a^{N-k} a_1 g (mod\ m_1 g)$, и, следовательно, $x = x_1 g$, где $x_1 \equiv a^{N-k} a_1 (mod\ m_1)$.
Сравнение $ax \equiv b\ (mod\ m)$ имеет либо 0, либо $g = gcd(a, m)$ решений
Функция Кармайкла: $\lambda(n)$ — равна наименьшему показателю m, что $a^m \equiv 1\ (\mathrm{mod}\ n), \forall (a, n) = 1$.
$\lambda(p^\alpha) = \varphi(p^\alpha), \forall p > 2, p \in \mathbb{P}$ или $p^\alpha \in \{2, 4\}$,
$\lambda(2^\alpha) = \frac{\varphi(2^\alpha)}{2}, \forall \alpha > 2$,
$\lambda(p_1^{\alpha_1} p_2^{\alpha_2} \ldots p^{\alpha_k}) = lcm(\lambda(p_1^{\alpha_1}), \lambda(p_2^{\alpha_2}), \ldots, \lambda(p_k^{\alpha_k}))$.
Теорема Вильсона: Натуральное число $p > 1$ является простым тогда и только тогда, когда $(p-1)! + 1 \equiv 0\ (mod\ p)$.
Критерий Эйлера: Пусть $p > 2$ - простое число. Число a, взаимно простое с p, является квадратичным вычетом по модулю p тогда и только тогда, когда $a^{\frac{p-1}{2}} \equiv 1 (mod\ p)$ и является квадратичным невычетом по модулю p тогда и только тогда, когда $a^{\frac{p-1}{2}} \equiv -1 (mod\ p)$.
Свойства чисел Фибоначчи: $F_{n+1}F_{n-1} - F_n^2 = (-1)^n$, $F_{n+k} = F_kF_{n+1} + F_{k-1}F_n$, $F_{2n} = F_n\,(F_{n+1} + F_{n-1})$.
Цепные дроби
$p_n = a_n \cdot p_{n-1} + p_{n-2}$, $q_n = a_n \cdot q_{n-1} + q_{n-2}$.
Цифровой корень:
Последовательность наименьших целых положительных чисел, которые требуют ровно n итераций извлечения цифрового корня в системе счисления по основанию b: $a_0 = 0$, $a_1 = b$, $a_n = 2b^{\frac{a_{n-1}}{b-1}} - 1$ при $n > 1$.
Числа Гаусса: Это комплексные числа, у которых и действительная, и мнимая части целые. Норма числа $a + ib$ определяется как $a^2 + b^2$. Если $a + ib = (c + id)(e + if)$, то $a^2 + b^2 = (c^2 + d^2)(e^2 + f^2)$. Число $a + ib$ делится на число $c + id$ тогда и только тогда, когда $c^2 + d^2$ делит $ac + bd$ и $bc - ad$.
Гауссово число $a + ib$ является простым тогда и только тогда, когда: 1) либо одно из чисел a, b нулевое, а другое - целое простое число вида $\pm(4k + 3)$, 2) либо a, b оба отличны от нуля и норма $a^2 + b^2$ - простое натуральное число.

9 Геометрия

Теорема синусов: $\frac{a}{\sin\alpha} = \frac{b}{\sin\beta} = \frac{c}{\sin\gamma} = 2R$, R — радиус описанной окружности.
Теорема косинусов: $a^2 = b^2 + c^2 - 2bc\sin\widehat{bc}$.
Теорема тангенсов: $\dfrac{a-b}{a+b} = \dfrac{\tan\frac{1}{2}(\alpha-\beta)}{\tan\frac{1}{2}(\alpha+\beta)}$.

Геометрия на сфере:
1. Большой круг задаётся нормалью.
2. Два неравных круга пересекаются в двух противоположных точках: $\pm normal(cross(n_1, n_2), r)$.
3. Площадь многоугольника равна: $S = (\sum_i \alpha_i - \pi * (n-2)) * r^2$, объем конуса (сектора) равен: $V = S * r/3$ (α_i — внутренний двугранный угол = угол между касательными векторами).
4. $\alpha_i = \pi - ang(n_i, n_{i+1})$, где n_j — нормаль к j-й дуге многоугольника направленная внутрь многоугольника.
5. Для сортировки точек на круге C удобно перейти к $2D$ (введя базис в плоскости круга: $e_1 = p_0, e_2 = cross(e_1, n_c)$), p_0 — любая из точек на круге, n_c — нормаль к кругу.
6. Для сортировки по кругу отрезков исходящих из точки C, надо свести задачу к 5.

Треугольники и их свойства: a, b, c — стороны, треугольника; α, β, γ — соответствующие сторонам a, b, c углы, r, R — радиусы вписанной и описанной окружностей, d — расстояние между центрами вписанной и описанной окружностей.

1. $S = \frac{1}{2}ab\sin\gamma = \frac{a^2\sin\beta\sin\gamma}{2\sin\alpha} = 2R^2\sin\alpha\sin\beta\sin\gamma$,
2. $S = \frac{ab}{2} = r^2 + 2rR, r = \frac{ab}{a+b+c} = \frac{a+b-c}{2}$ — для прямоугольного,
3. $S = \frac{a^2\sqrt{3}}{4}$ — для равностороннего,
4. $R = \frac{abc}{4S}$ — центр в точке пересечения серединных перпендикуляров, $r = \frac{S}{p}$ — центр в точке пересечения биссектрис, $d^2 = R^2 - 2Rr$.

Геометрическая инверсия
Инверсия точки P относительно окружности с центром в точке O и радиусом R — это точка P', которая лежит на луче OP, и $OP \cdot OP' = R^2$.
Прямая, проходящая через O, не меняется. Прямая, не проходящая через O, перейдет в окружность, проходящую через O, и наоборот. Окружность, не проходящая через O, перейдет в окружность, попрежнему не проходящую через O.
Если после инверсии точки P и Q переходят в P' и Q', то $\angle PQO = \angle Q'P'O$, $\angle QPO = \angle P'Q'O$ и треугольники $\triangle PQO$ и $\triangle Q'P'O$ подобны.
Преобразование инверсии сохраняет углы в точках пересечения кривых (ориентация меняется на противоположную).
Обобщённая окружность при преобразовании инверсии сохраняется тогда и только тогда, когда она ортогональна окружности, относительно которой производится инверсия.
Чтобы найти окружность, получившуюся в результате инверсии прямой, нужно найти ближайшую к центру инверсии точку Q прямой, применить к ней инверсию, и тогда искомая окружность будет иметь диаметр OQ'.
Чтобы найти окружность, получившуюся в результате инверсии другой окружности, нужно провести через центр инверсии и центр старой окружности прямую, и посмотреть ее точки пересечения S и T со старой окружностью. Отрезок ST после инверсии будет образовывать диаметр, следовательно, центр новой окружности это среднее арифметическое точек S' и T'.
Окружность с центром в точке (x, y) и радиусом r после инверсии относительно окружности с центром в точке (x_0, y_0) и радиусом r_0 перейдет в окружность с центром в точке (x', y') и радиусом r', где
$x' = x_0 + s \cdot (x - x_0)$, $y' = y_0 + s \cdot (y - y_0)$, $r' = |s| \cdot r$, $s = \frac{r_0^2}{(x-x_0)^2 + (y-y_0)^2 - r^2}$.

Шар:
$S = 4\pi R^2$, $V = \frac{4}{3}\pi R^3$ $V = V_n R^n$, $V_n = \frac{\pi^{\lfloor n/2 \rfloor}}{\Gamma(\frac{n}{2}+1)}$, $V_{2k} = \frac{\pi^k}{k!}$, $V_{2k+1} = \frac{2^{k+1}\pi^k}{(2k+1)!!}$
$S = S_n R^n$, $S_0 = 2$, $S_n = 2\pi V_{n-1}$
$V = \frac{1}{3}\pi h^2(3R - h)$, $S = 2\pi Rh$ для шарового сегмента, h — высота сегмента
$V = \frac{2}{3}\pi R^2 h$ для шарового сектора, h — высота соответствующего шарового сегмента

Тор:
$S = 4\pi^2 Rr$, $V = 2\pi^2 Rr^2$, R — расстояние от центра образующей окружности до оси вращения, r — радиус образующей окружности

Тетраэдр:
$V = \frac{\sqrt{2}a^3}{12}$, $h = \frac{\sqrt{6}a}{3}$, $r = \frac{\sqrt{6}a}{12}$, $R = \frac{\sqrt{6}a}{4}$,
$288V^2 = \begin{vmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & d_{12}^2 & d_{13}^2 & d_{14}^2 \\ 1 & d_{12}^2 & 0 & d_{23}^2 & d_{24}^2 \\ 1 & d_{13}^2 & d_{23}^2 & 0 & d_{34}^2 \\ 1 & d_{14}^2 & d_{24}^2 & d_{34}^2 & 0 \end{vmatrix}$. При фиксированных попарных расстояниях тетраэдр построить нельзя, если определитель < 0 или хотя бы на одной грани не выполняется неравенство треугольника.

Конус:
$V = \frac{1}{3}\pi R^2 h$, $S = \pi Rl$ — площадь боковой поверхности, где l — образующая.

Круг:
$S = \frac{R^2}{2}\theta$ для сектора круга, $S = \frac{R^2}{2}(\theta - \sin\theta)$ для сегмента круга.

Формула Эйлера для числа граней в планарном графе:
$V - E + F = 1 + C$, где V - число вершин, E - ребер, F - граней, C - компонент связности графа.

Теорема о секущих:
Если из точки, лежащей вне окружности, провести две секущие, то произведение одной секущей на её внешнюю часть равно произведению другой секущей на её внешнюю часть: $AB \cdot AC = AD \cdot AE$

Пересечение плоскостей:
Необходимо найти параметрическое представление прямой пересечения двух плоскостей: $A_1x + B_1y + C_1z + D_1 = 0$ и $A_2x + B_2y + C_2z + D_2 = 0$. Рассмотрим 3 случая:

$$\Delta_1 = det(A_1, A_2, B_1, B_2) \neq 0,$$
$$x = \frac{det(B_1, B_2, D_1, D_2)}{\Delta_1} + t\frac{det(B_1, B_2, C_1, C_2)}{\Delta_1},$$
$$y = \frac{det(D_1, D_2, A_1, A_2)}{\Delta_1} + t\frac{det(C_1, C_2, A_1, A_2)}{\Delta_1},$$
$$z = 0 + t \cdot 1$$

$$\Delta_2 = det(A_1, A_2, C_1, C_2) \neq 0$$
$$x = \frac{det(C_1, C_2, D_1, D_2)}{\Delta_2} + t\frac{det(C_1, C_2, B_1, B_2)}{\Delta_2}$$
$$y = 0 + t \cdot 1,$$
$$z = \frac{det(D_1, D_2, A_1, A_2)}{\Delta_2} + t\frac{det(B_1, B_2, A_1, A_2)}{\Delta_2}$$

$$\Delta_3 = det(B_1, B_2, C_1, C_2) \neq 0,$$
$$x = 0 + t \cdot 1,$$
$$y = \frac{det(C_1, C_2, D_1, D_2)}{\Delta_3} + t\frac{det(C_1, C_2, A_1, A_2)}{\Delta_3},$$
$$z = \frac{det(D_1, D_2, B_1, B_2)}{\Delta_3} + t\frac{det(A_1, A_2, B_1, B_2)}{\Delta_3}$$

$$\Delta_1 = \Delta_2 = \Delta_3 = 0$$
$$det(A_1, A_2, D_1, D_2) = 0 \Leftrightarrow planes\ are\ equal$$
$$det(A_1, A_2, D_1, D_2) \neq 0 \Leftrightarrow planes\ are\ parralel$$

Пересечение плоскости и прямой:
Дана плоскость $Ax+By+Cz+D = 0$ и прямая в виде $a+vt$, тогда точке пересечения будет соответствовать параметр
$t = -\frac{A \cdot a.x + B \cdot a.y + C \cdot a.z + D}{A \cdot v.x + B \cdot v.y + C \cdot v.z}$

Сферические координаты:
Если $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ — широта, а $\varphi \in [0, 2\pi)$ — долгота, то
$x = r\cos\theta\cos\varphi$, $y = r\cos\theta\sin\varphi$, $z = r\sin\theta$.

Пересечение окружности и прямой: Сдвинем систему координат в центр окружности. Для сдвига прямой на вектор (dx, dy) нужно сделать $C- = A \cdot dx + B \cdot dy$, в данном случае $dx = -x_0, dy = -y_0$. Теперь если $d > r$ ($d = \frac{|C|}{\sqrt{A^2+B^2}}$), то точек пересечения нет. Обозначим $l = \sqrt{r^2 - d^2}, c = (\frac{-AC}{A^2+B^2}, \frac{-BC}{A^2+B^2})$. Тогда точки пересечения имеют вид: $c \pm normal(pt(-B, A), l)$.

Центры масс: Везде далее будем выражать центры масс с помощью радиус-векторов.
Центр масс системы материальных точек: $\vec{r_c} = \frac{\sum_i \vec{r_i}\ m_i}{\sum_i m_i}$,
Центр масс однородного каркаса, как многоугольника, так и многогранника (заменяем каждое ребро точкой в его середине с массой, равной длине ребра): $\vec{r_c} = \frac{\sum_i \vec{r_i^{mid}}\ l_i}{P}$,
Центр масс сплошных фигур: центр масс произвольного сплошного треугольника или тетраэдра - среднее арифметическое его координат (обобщается и на симплексы больших размерностей). Центр масс произвольного сплошного многоугольника/многогранника считается следующим образом: выбирается произвольная точка p из нее проводятся треугольники/тетраэдры к последовательным вершинам фигуры (или к треугольникам из триангуляции грани) и вычисляется взвешенное среднее центров масс треугольников/тетраэдров с весами, равными знаковым площадям/объемам.

Центр масс поверхности многогранника - это взвешенное среднее центров масс граней с весами, равными площадям граней.
Окружности Мальфатти:
$r_1 = \frac{r}{2(p-a)}(p+d-r-e-f)$, $r_2 = \frac{r}{2(p-b)}(p+e-r-d-f)$, $r_3 = \frac{r}{2(p-c)}(p+f-r-d-e)$, где d, e, f — расстояния от инцентра до углов A, B, C соответственно

10 Графы

Теорема Холла: Если в двудольном графе произвольно выбранное множество вершин первой доли покрывает не меньшее по размеру множество вершин второй доли, то в графе существует совершенный парсоч.
Теорема Пика: Для многоугольника без самопересечений с целочисленными вершинами имеем: $S = I + \frac{B}{2} - 1$, где S — площадь, I — количество целочисленных точек внутри, B — количество целочисленных точек на границе.
Аксиомы частично упорядоченных множеств (ЧУМ):

$$1.\ a < b \wedge b < c \Rightarrow a < c;\ 2.\ a < b \Rightarrow a \neq b;\ 3.\ a < b \Rightarrow \overline{b < a}.$$

Цепь — множество попарносравнимых элементов.
Антицепь — множество попарнонесравнимых элементов.
Теорема Дилуорса: Размер максимальной антицепи равен размеру минимального покрытия ЧУМа цепями.
Следствие: Наибольшее количество вершин в орграфе таких, что никакая вершина недостижима из другой равно минимальному покрытию замыкания этого графа путями (циклы не мешают).
DAG minimum covering: Необходимо покрыть DAG наименьшим количеством вершинно-непересекающихся путей. Для этого построим двудольный граф вершины есть которого раздвоенные вершины исходного графа. Дугу (x, y) исходного графа заменяем на дугу (x_1, y_2) нового. Находим макс. парсоч. Теперь для восстановления ответа достаточно сказать, что вершины соединенные ребром парсоча являются соседними в одном из путей ответа.
Дейкстра с потенциалами: Введем потенциалы φ_i, новые веса ребер будут иметь вид: $\overline{c_{ij}} = c_{ij} + (\varphi_i - \varphi_j) \geq 0$. В качестве потенциалов можно выбрать $\varphi_i = d_i$, где d_i — кратчайшие расстояния из истока (специально добавленной в граф вершины из которой есть дуги веса 0 во все остальные вершины) вычисленные с помощью алгоритма Форда-Беллмана. В частности в задаче $minCostFlow$ можно положить исходно $\varphi_i = 0$ и после каждого шага увеличивать все φ_i на величину d_i. φ_i могут переполниться у недостижимых вершин.
Паросочетания:
В произвольном графе: $|MIVS| + |MVC| = |V|, |MM| + |MEC| = |V|$.
В двудольном графе: $|MVC| = |MM|$.
Для нахождения MVC в двудольном графе нужно запустить Куна из ненасыщенных вершин первой доли, тогда вершины по которым мы не прошли в первой доле и прошли во второй образуют MVC ($MIVS = V \backslash MVC$).
Для нахождения MEC построим MM. Теперь из каждой непокрытой (неизолированной) вершины все ребра ведут в насыщенные вершины. Выберем для каждой ненасыщенной вершины любое ребро и добавим эти ребра в MM получим MEC.
MEC представляет собой лес (с деревьями диаметра не более 2), поэтому для получения MM нужно из каждого дерева взять по одному ребру.
Наибольшее доминирование: X — множество вершин первой доли, Y — множество вершин второй доли которые покрыты вершинами из X. Необходимо найти X, чтобы величина $|X| - |Y|$ была максимальна. Для этого построим MM и запустим Куна из ненасыщенных вершин первой доли, тогда вершины по которым мы прошли в первой доле образуют множество X, а вершины по которым мы прошли во второй доле образуют множество Y и при этом значение $|X| - |Y|$ будет максимально.
2-SAT:
1. Задачу всегда НЕОБХОДИМО сводить к каноническому виду $2 - CNF$: $(a_i \vee b_i) \wedge (a_{i+1} \vee b_{i+1})$. Все бинарные операции легко выражаются в этой форме. В частности импликация $x \to y$ выражается, как $(!x \vee y)$.
2. Если заранее известно чему равно значение x_i (т.е. x_i константа, а не переменная), то достаточно ввести одну фиктивную вершину и сказать, что из ее отрицания следует (импликация) известное значение x_iго, а также из ее неотрицания тоже следует значение x_i.
3. Ответ восстанавливается очень просто среди значений x_0, x_1 выбирается то, компонента сильной связности которой стоит позже в порядке топологической сортировки (именно компонента, а не вершина x_i). Т.е. необходимо сравнивать $comp[x_0]$ и $comp[x_1]$, что больше то и нужно выбрать.
4. От произвольной таблицы истинности легко перейти к форме $2 - CNF$: для этого нужно выбрать все строчки в которых значение функции равно 0 и добавить дизъюнкцию в которой переменные равные 1 взяты с отрицанием, а равные 0 — без отрицания.
5. Добавляя произвольную импликацию ОБЯЗАТЕЛЬНО нужно убедиться, что добавлена встречная импликация: $x \to y \Rightarrow !y \to !x$.
Покраска подпути в дереве на min (offline $O(nlog^2n)$):
Для покраски подпути из вершины x в вершину y нужно предварительно подвесить дерево и заменить покраску (x, y) на 2 покраски (x, l), (y, l) ($l = lca(x, y)$). Для покраски (v, p) нужно добавить событие в вершину v, что началась покраска, а в вершину p — покраска закончилась. Далее обходом в глубину, который возвращает указатель на множество покрасок в поддереве, необходимо посчитать искомые значения минимума для каждого ребра. Объединение множеств детей нужно осуществлять стандартным образом, объединяя меньшее множество к большему.
MVC в произвольном графе за $O(\varphi^n)$
MVC ищется перебором. Предварительно для каждой вершины степени 1 нужно взять её соседа (если он тоже степени 1, то нужно взять только одного из них). Далее в переборе каждый раз ищем вершину с наибольшим количеством непокрытых рёбер. Теперь нам нужно взять либо её, либо всех её соседей ребро к которым ещё не покрыто.
Эйлеров цикл/путь: Задача об эйлеровом пути легко сводится к задаче об эйлеровом цикле добавлением одного ребра/дуги. Изолированные вершины стоит игнорировать. Неор. граф: эйлеров цикл существует тогда и только тогда, когда граф связный и в нём отсутствуют вершины нечётной степени. Ор. граф: эйлеров цикл существует тогда и только тогда, когда граф сильно-связный и для каждой вершины её полустепень захода равна её полустепени исхода. В случае отсутсвия Эйлерова цикла алгоритм Флёри всё равно найдёт некоторую последовательность вершин, но она будет некорректной.
Матричная теорема Кирхгофа: Пусть задан неориентированный связный граф с кратными ребрами. Если в матрице смежности графа заменить каждый элемент на противоположный по знаку, а элемент a_{ii} заменить на степень вершины i (с учетом кратности ребер), то все алгебраические дополнения этой матрицы равны между собой и равны количеству остовных деревьев этого графа.
Матрица Татта: Пусть в графе существует совершенное паросочетание, тогда его матрица Татта невырождена. Если алгебраическое дополнение элемента, соответствующего ребру (i, j), т.е. элемент $A^{-1}_{j,i}$, отличен от нуля, то это ребро может входить в совершенное паросочетание.

11 Формулы

Достаточное условие корректности оптимизации Кнута:
Считаем динамику вида: $dp_{ij} = min_{i<k<j}(dp_{ik} + dp_{kj} + C_{ij})$. Достаточное условие: 1. $C_{ac} + C_{bd} \leq C_{ad} + C_{bc}$, 2. $C_{bc} \leq C_{ad}$, при всех $a \leq b \leq c \leq d$.
Элементарная тригонометрия:
$\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$; $\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta$; $\tan(\alpha \pm \beta) = \frac{\tan\alpha \pm \tan\beta}{1 \mp \tan\alpha\tan\beta}$;
$\sin\alpha\cos\beta = \frac{1}{2}(\sin(\alpha+\beta) + sin(\alpha-\beta))$; $\sin\alpha\sin\beta = \frac{1}{2}(\cos(\alpha-\beta) - cos(\alpha+\beta))$;
$\cos\alpha\cos\beta = \frac{1}{2}(\cos(\alpha-\beta) + cos(\alpha+\beta))$; $\sin\alpha \pm \sin\beta = 2\sin\frac{\alpha \pm \beta}{2}\cos\frac{\alpha \mp \beta}{2}$; $\cos\alpha - \cos\beta = -2\sin\frac{\alpha+\beta}{2}\sin\frac{\alpha-\beta}{2}$;
$\cos\alpha + \cos\beta = 2\cos\frac{\alpha+\beta}{2}\cos\frac{\alpha-\beta}{2}$; $\tan\alpha \pm \tan\beta = \frac{\sin(\alpha \pm \beta)}{\cos\alpha\cos\beta}$; $\cot\alpha \pm \cot\beta = \frac{\sin(\beta \pm \alpha)}{\sin\alpha\sin\beta}$
Волшебная сумма: $\sum\limits_{0 \leq k < m} \lfloor \frac{nk+x}{m} \rfloor = \sum\limits_{0 \leq k < n} \lfloor \frac{mk+x}{n} \rfloor = d\lfloor \frac{x}{d} \rfloor + \frac{(m-1)(n-1)}{2} + \frac{d-1}{2}$.
Суммирование по частям:
$\Delta f(x) = f(x+1) - f(x), Ef(x) = f(x+1) \Rightarrow \sum u\Delta v = uv - \sum Ev\Delta u$,
$\Delta x^{\underline{m}} = mx^{\underline{m-1}}, \Delta c^x = (c-1)c^x, \Delta(af + bg) = a\Delta f + b\Delta g, \Delta fg = f\Delta g + Eg\Delta f$.
Формулы округлений:
$\lfloor x \rfloor = n \Leftrightarrow n \leq x < n+1 \Leftrightarrow x - 1 < n \leq x$, $\lceil x \rceil = n \Leftrightarrow n - 1 < x \leq n \Leftrightarrow x \leq n < x + 1$,
$x < n \Leftrightarrow \lfloor x \rfloor < n$, $n < x \Leftrightarrow n < \lceil x \rceil$, $x \leq n \Leftrightarrow \lceil x \rceil \leq n$, $n \leq x \Leftrightarrow n \leq \lfloor x \rfloor$.
Интерполяционный многочлен Лагранжа:
Заданы пары значений (x_i, y_i) $(i = \overline{0, n})$ — узел и значение функции в узле. Тогда существует единственный многочлен степени не более n принимающий значения y_i в узлах x_i: $f(x) = \sum\limits_{i=0}^{n} y_i \prod\limits_{j=0, j \neq i}^{n} \frac{x - x_j}{x_i - x_j}$
Интерполяционный многочлен Ньютона:

Заданы пары значений (x_i, y_i) $(i = \overline{0,n})$ — узел и значение функции в узле.
Определим разделенные разности вперед: $[y_\nu] := y_\nu (\nu = \overline{0,n})$;
$[y_\nu, \dots, y_{\nu+j}] := \frac{[y_{\nu+1}, \dots, y_{\nu+j}] - [y_\nu, \dots, y_{\nu+j-1}]}{x_{\nu+j} - x_\nu} (\nu = \overline{0, \dots, n-j}, j = \overline{1, \dots, n})$.
$f(x) = \sum\limits_{i=0}^{n} [y_0, \dots, y_i] \prod\limits_{j=0}^{i-1} (x - x_j)$

Криволинейный интеграл первого рода:
Пусть l — гладкая, спрямляемая (имеет конечную длину) кривая, заданная параметрически: $x = x(t), y = y(t), z = z(t)$. Пусть $f(x, y, z)$ определена и интегрируема вдоль кривой l. Тогда:

$$\int\limits_l f(x,y,z)dl = \int\limits_a^b f(x(t), y(t), z(t))\sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2}dt$$

Здесь точка — это производная по t. На плоскости удобно вводить параметризацию: $x = t, y = y(t)$. Для вычисления длины кривой нужно положить $f(x, y, z) \equiv 1$.

Поверхностный интеграл первого рода:
Пусть на поверхности Ф можно ввести единую параметризацию посредством функций $x = x(u, v), y = y(u, v), z = z(u, v)$, заданных в ограниченной области Ω плоскости (u, v) и принадлежащих классу C^1 (непрерывнодифференцируемых) в этой области. Если функция $f(M) = f(x, y, z)$ непрерывна на поверхности Ф, то поверхностный интеграл первого рода от этой функции по поверхности Ф существует и может быть вычислен по формуле:

$$\iint\limits_\Phi f(M)d\sigma = \iint\limits_\Omega f(x(u,v), y(u,v), z(u,v))\sqrt{EG - F^2}dudv$$

где $E = (x'_u)^2 + (y'_u)^2 + (z'_u)^2, F = x'_u x'_v + y'_u y'_v + z'_u z'_v, G = (x'_v)^2 + (y'_v)^2 + (z'_v)^2$.

Покраска отрезка с помощью dsu:
Покраску нужно осуществлять начиная с последнего запроса и заканчивая первым.
for (int v = leader(lf); v <= rg; v = dsu[v] = leader(v + 1)) col[v] = newCol;

Теорема Безу: Остаток от деления многочлена $P(x)$ на двучлен $(x - a)$ равен $P(a)$.

Теорема Байеса:
$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$, где $P(A|B)$ - вероятность наступления A, если уже наступило B

Неприводимые многочлены:
Количество неприводимых многочленов (неразложимых на произведение) степени n в поле по простому модулю p равно: $cnt = \frac{1}{n} \sum\limits_{d|n} \mu(d) p^{n/d}$.

Коды Грея:
Код Грея — это такая перестановка битовых строк длины n, что каждая следующая отличается от предыдущей ровно в одном бите. n-й код Грея соответствует гамильтонову циклу вдоль вершин n-го куба.

НОД многочлена и его производной
$deg(p) = deg(GCD(p(x), p'(x))) + k(p)$, где $k(p)$ — количество различных корней.
$p(x) = c \cdot (x - a_0)^{n_0} \cdot (x - a_1)^{n_1} \cdot \dots \cdot (x - a_k)^{n_k}$
$p'(x) = c \cdot [(x - a_0)^{n_0 - 1} \cdot (x - a_1)^{n_1} \cdot \dots \cdot (x - a_k)^{n_k} + (x - a_0)^{n_0} \cdot (x - a_1)^{n_1 - 1} \cdot \dots \cdot (x - a_k)^{n_k} + (x - a_0)^{n_0} \cdot (x - a_1)^{n_1} \cdot \dots \cdot (x - a_k)^{n_k - 1}]$
$GCD(p(x), p'(x)) = c \cdot (x - a_0)^{n_0 - 1} \cdot (x - a_1)^{n_1 - 1} \cdot \dots \cdot (x - a_k)^{n_k - 1}$

Метод касательных Ньютона:
Для решения уравнения $f(x) = 0$ будем проводить итерации: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, где x_n — n-е приближение решения. В качестве x_0 можно выбрать любое значение, но для улучшения скорости сходимости лучше выбрать x_0 близким к искомому решению.

Метод простой итерации (в матричном виде): $x = (A + E)x - b$.

Метод наименьших квадратов: Решает для n точек задачу вида $\sum\limits_{i=1}^{n} (ax_i + b - y_i)^2 \to min$. Решением задачи является $a = \frac{n \sum\limits_{i=1}^{n} x_i y_i - \sum\limits_{i=1}^{n} x_i \sum\limits_{i=1}^{n} y_i}{n \sum\limits_{i=1}^{n} x_i^2 - \left(\sum\limits_{i=1}^{n} x_i\right)^2}$, $b = \frac{\sum\limits_{i=1}^{n} y_i - a \sum\limits_{i=1}^{n} x_i}{n}$.

Обратная польская нотация: Левоассоциативные операции (т.е. те, которые в случае равенства приоритета выполняются слева направо, например, $+, -, *$) выталкивают из стека операции с $\geq$ приоритетом. Правоассоциативные (например, возведение в степень) выталкивают операции с $>$ приоритетом. Унарные операции удобно предварительно заменить специальными символами, они имеют наибольший приоритет и зачастую считаются правоассоциативными. Открывающаяся скобка просто помещается в стек, а закрывающаяся выталкивает все операции, пока не встретит открывающуюся.

Формула Райзера для перманента: $Per(A) = (-1)^n \sum\limits_{S \subseteq \{1,\dots,n\}} (-1)^{|S|} \prod\limits_{i=1}^{n} \sum\limits_{j \in S} a_{ij}$.

Рюкзак на маленьких весах: Задан набор чисел A_i, а также набор чисел x_i. Для каждого x_i необходимо определить можно ли его набрать числами из набора A_i, причем каждое A_i можно брать более одного раза. Для решения этой задачи зафиксируем произвольное число из набора, например можно зафиксировать минимальное среди них $c := A_0$. Теперь посчитаем величины d_i — наименьшая сумма, которую можно набрать числами из набора A_i, такая что остаток этой суммы по модулю c равен i (это можно сделать например алгоритмом Дейкстры, хотя граф весьма специфичен и можно это делать быстрее). Теперь проверка числа x_i сводится к проверке неравенства $d[x_i \mod c] \leq x_i$.

Пятнашки
Пусть a_i — перестановка чисел от 0 до 15 (в порядке сверху-вниз, слева-направо), N — количество инверсий в a, $K = \lfloor \frac{z-1}{4} \rfloor + 1$ — номер строки с нулем, где z — 1-индексированный номер позиции нуля в a. Тогда решение существует тогда и только тогда, когда $N + K$ четно.

12 Полезные числа

n	2^n	2^{n+20}	3^n	$n!$	B_n	C_n
0	1	1'048'576	1	1	1	1
1	2	2'097'152	3	1	1	1
2	4	4'194'304	9	2	2	2
3	8	8'388'608	27	6	5	5
4	16	16'777'216	81	24	15	14
5	32	33'554'432	243	120	52	42
6	64	67'108'864	729	720	203	132
7	128	134'217'728	2'187	5'040	877	429
8	256	268'435'456	6'561	40'320	4'140	1'430
9	512	536'870'912	19'683	362'880	21'147	4'862
10	1'024	1'073'741'824	59'049	3'628'800	115'975	16'796
11	2'048	2'147'483'648	177'147	39'916'800	678'570	58'786
12	4'096	4'294'967'296	531'441	479'001'600	4'213'597	208'012
13	8'192	8'589'934'592	1'594'323	6'227'020'800	27'644'437	742'900
14	16'384	17'179'869'184	4'782'969	-	190'899'322	2'674'440
15	32'768	34'359'738'368	14'348'907	-	1'382'958'545	9'694'845
16	65'536	68'719'476'736	43'046'721	-	-	35'357'670
17	131'072	137'438'953'472	129'140'163	-	-	129'644'790
18	262'144	274'877'906'944	387'420'489	-	-	477'638'700
19	524'288	549'755'813'888	1'162'261'467	-	-	1'767'263'190

$F(0) = 0, F(1) = 1, F(n) = F(n-2) + F(n-1), F(26) = 121393, F(35) \sim 10^7, F(60) \sim 10^{12}, F(91) \sim 4 \cdot 10^{18}$

S_n^k	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0	0	0
3	0	1	3	1	0	0	0	0	0	0	0	0
4	0	1	7	6	1	0	0	0	0	0	0	0
5	0	1	15	25	10	1	0	0	0	0	0	0
6	0	1	31	90	65	15	1	0	0	0	0	0
7	0	1	63	301	350	140	21	1	0	0	0	0
8	0	1	127	966	1'701	1'050	266	28	1	0	0	0
9	0	1	255	3'025	7'770	6'951	2'646	462	36	1	0	0
10	0	1	511	9'330	34'105	42'525	22'827	5'880	750	45	1	0
11	0	1	1'023	28'501	145'750	246'730	179'487	63'987	11'880	1'155	55	1