Standard Code Library

TDL, Beihang University

2017 年 5 月 21 日

1 Dynamic Programming

1.1 Completely Monotonic DP

```
void Mono_DP(int L, int R, int dL, int dR) {
  if(L > R) return;
  int M = (L + R) >> 1, dM;
  for(int i = dL; i <= dR; ++i)
    if(upd(f[M], i)) dM = i; // <= i
  Mono_DP(L, M, dL, dM), Mono_DP(M + 1, R, dM, dR);
} // call : (1, n, 1, n)
```

1.2 Hu-Tucker Tree

```
int n, a[maxn], c, ans;
void dfs(int x) {
  int i, j = a[x - 1] + a[x];
  ans += j, --c;
  for(i = x; i < c; ++i) a[i] = a[i + 1];
  for(i = x - 1; i && a[i - 1] < j; --i)
    a[i] = a[i - 1];
  for(a[i] = j; i >= 2 && a[i] >= a[i - 2];)
    j = c - i, dfs(i - 1), i = c - j;
}
int solve() {
  c = ans = 0;
  for(int i = 0; i < n; ++i)
    for(a[c++] = a[i]; c >= 3
    && a[c - 3] <= a[c - 1]; dfs(c - 2));
  while(c > 1) dfs(c - 1);
  return ans;
}
```

1.3 Steiner Tree

```
/* dp[S][i] = min(dp[T][i] + dp[S - T][i], dp[S][j] + cost[j][i]) */
int n, m, e[N][N], sp[N], f[S][N];
int work() { // after build
  int i, j, k; // init m = 0, f = INF
  for(i = 0; i < n; ++i) if(sp[i])
    f[1 << m][i] = 0, ++m;
  for(i = 1; i < 1 << m; ++i) {
    for(j = (i - 1) & i; j; j = (j - 1) & i)
      for(k = 0; k < n; ++k)
        upd(f[i][k], f[j][k] + f[i ^ j][k]);
    dijkstra(i); // for f[i][j] < INF
  }
  int ans = INF, all = (1 << m) - 1;
  for(i = 0; i < n; ++i) upd(ans,f[all][i]);
  return ans;
}
```

1.4 LCS by bitset

```
int LCS_bitset(char s[maxn], char t[maxn]) {
  int n = strlen(s), m = strlen(t), ans = 0;
  int sz = ((n - 1) >> 6) + 1, cur=0, pre=1;
  static ULL pos[maxd][maxs], f[3][maxs];
  for(int i = 0; i < n; ++i) // REM: clear
    pos[s[i]][i >> 6] |= 1ULL << (i & 63);
  memset(f[cur], 0, sz * sizeof(ULL));
  for(int i = 0; i < m; ++i) {
    pre ^= 1, cur ^= 1;
    for(int j = 0; j < sz; ++j)
      f[2][j] = f[pre][j] | pos[t[i]][j];
    for(int j = 0, x = 1, y; j < sz; ++j, x = y) {
      y = (f[pre][j] >> 63) & 1;
      f[pre][j] = f[pre][j] << 1 | x;
    }
    memcpy(f[cur], f[2], sz * sizeof(ULL));
    for(int j = 0, x = 0, y; j < sz; ++j, x = y) {
      y = f[cur][j] < f[pre][j] + x;
      f[cur][j] -= f[pre][j] + x;
      f[cur][j] = ~f[cur][j] & f[2][j];
    }
  }
  for(int i = 0; i < n; ++i)
    ans += (f[cur][i >> 6] >> (i & 63)) & 1;
  return ans;
}
```

2 Data Structure

2.1 Splay Tree

```
struct Node {
  Node *l, *r, *p;
  int s, k; // size & key
  Node(int k) : l(0),r(0),p(0),s(1),k(k) {}
  void upd(){s=1+(l?l->s:0)+(r?r->s:0);}
};
// Rotates edge x - x's p.
void rotate(Node *x) {
  Node *y = x->p, *z = y->p;
  if (x == y->l) {
    y->l = x->r;
    if (x->r) x->r->p = y;
    x->r = y; y->p = x;
  } else {
    y->r = x->l;
    if (x->l) x->l->p = y;
    x->l = y; y->p = x;
  }
  x->p = z;
  if(z) (z->l == y ? z->l : z->r) = x;
  y->upd(); x->upd();
}
// Splays node x to root, returns new root
Node *splay(Node *x) {
  if (!x) return 0;
  while (x->p) {
    Node *y = x->p, *z = y->p;
    if (!z) // zig
      { rotate(x); break; }
    if((y->l==z) == (z->l==y)) // zig-zig
      rotate(y), rotate(x);
    else // zig-zag
      rotate(x), rotate(x);
  }
  return x;
}
Node *splay(Node *root, int key) {
  Node *x = root;
  while (x && x->k != key) {
    Node *y = (key < x->k) ? x->l : x->r;
    if (!y) break;
    x = y;
  }
  return splay(x);
}
Node *insert(Node *root, int key) {
  Node *x = root, *y = 0, *z=new Node(key);
  while (x) {
    y = x; x->s++;
    if (key < x->k) x = x->l; else x = x->r;
  }
  z->p = y;
  if (y) (key < y->k ? y->l : y->r) = z;
  return splay(z);
}
Node *join(Node *a, Node *b) {
  if (a) a->p = 0;
  if (b) b->p = 0;
  if (!a) return b;
  if (!b) return a;
  while (a->r) a = a->r;
  splay(a);
  a->r = b; b->p = a; a->upd();
  return a;
}
Node *remove(Node *x) {
  splay(x);
  Node *y = join(x->l, x->r);
  delete x;
  return y;
}
```

2.2 Treap

```
const int inf = 0x7fffffff;
struct Node *nil = 0;
struct Node {
  int key, aux, size;
  Node *l, *r;
  Node(int k): key(k), aux(rand()), size(1), l(nil), r(nil) {}
  void upd() {size = l->size + 1 + r->size;}
};
Node *create() {
  if (nil == 0) {
    nil = new Node(0);
    nil->size = 0; nil->aux = inf;
  }
  return nil;
}
void destroy(Node *x) {
  if (x != nil) {
    destroy(x->l); destroy(x->r);
    delete x;
  }
}
Node *rotl(Node *x) {Node *y=x->r; x->r=y->l; y->l=x; x->upd(); y->upd(); return y;}
Node *rotr(Node *x) {Node *y=x->l; x->l=y->r; y->r=x; x->upd(); y->upd(); return y;}
Node *insert(Node *x, Node *z) {
  if (x == nil) return z;
  if (z->key < x->key) {
    x->l = insert(x->l, z); x->upd();
    if (x->l->aux < x->aux) x = rotr(x);
  } else {
    x->r = insert(x->r, z); x->upd();
    if (x->r->aux < x->aux) x = rotl(x);
  } return x;
}
Node *remove(Node *x, int key) {
  if (x == nil) return x;
  if (key < x->key) {
    x->l = remove(x->l, key);
  } else if (x->key < key) {
    x->r = remove(x->r, key);
  } else {
    Node *y;
    if (x->l == nil) { y = x->r; delete x; return y; }
    if (x->r == nil) { y = x->l; delete x; return y; }
    for (y = x->r; y->l != nil; y = y->l);
    x->key = y->key;
    x->r = remove(x->r, x->key);
  }
  x->upd(); return x;
}
Node *find(Node *x, int key) {
  if (x == nil) return nil;
  if (key < x->key) return find(x->l, key);
  if (x->key < key) return find(x->r, key);
  return x;
}
Node *kth(Node *x, int k) {
  if (k < 0 || k >= x->size) return nil;
  for (;;) {
    if (k < x->l->size) x = x->l;
    else if((--k -= x->l->size) >= 0)x=x->r;
    else return x;
  }
}
int rank(Node *x, int key) {
  if (x == nil) return 0;
  if (key < x->key) return rank(x->l, key);
  return x->l->size + (x->key < key? 1 + rank(x->r, key) : 0);
}
```

2.3 k-Dimensional Tree

```
const int K = 2;
const int N = 100000;
typedef array<int, K> Coor;
template<int k> bool cmpT(const Coor &a, const Coor &b) {return a[k] < b[k];}
bool (*cmp[])(const Coor &, const Coor &) = {cmpT<0>, cmpT<1>}; //K==2
struct Filt {
  Coor l, r;
  bool in(const Coor &p) const {
    for (int i = 0; i < K; ++i)
      if(p[i] < l[i] || p[i] > r[i])return 0;
    return 1;
  }
  bool fit(const Filt &f) const {
    for (int i = 0; i < K; ++i)
      if(f.l[i] < l[i] || f.r[i] > r[i]) return 0;
    return 1;
  }
  void mrg(const Filt &f) {
    for (int i = 0; i < K; ++i) {
      l[i] = min(l[i], f.l[i]);
      r[i] = max(r[i], f.r[i]);
    }
  }
};
struct Node {
  Coor p;
  Filt f;
  int s, k;
  Node *l, *r;
  Node() {}
  Node(const Coor &q, int d) {
    f.l = f.r = p = q;
    s = 1; k = d;
    l = r = 0;
  }
  void upd() {
    f.l = f.r = p;
    s = 1;
    if (l) {s += l->s; f.mrg(l->f);}
    if (r) {s += r->s; f.mrg(r->f);}
  }
} *root, *e, E[N + 9];
Node *con(Coor q[], int l, int r, int k) {
  if (l < r) {
    int m = (l + r) >> 1;
    nth_element(q + l, q + m, q + r,cmp[k]);
    Node *p = e++;
    *p = Node(q[m], k);
    p->l = con(q, l, m, (k + 1) % K);
    p->r = con(q, m + 1, r, (k + 1) % K);
    p->upd();
    return p;
  }
  return 0;
}
void ins(Node *&p, const Coor &q, int k) {
  if (p) {
    if (q == p->p) return;
    if (q[k] < p->p[k]) ins(p->l, q, (k + 1) % K);
    else ins(p->r, q, (k + 1) % K);
    p->upd();
  } else {
    *e = Node(q, k); p = e++;
  }
}
const int inf = 0X7FFFFFFF;
int ans;
/* 计算点到区域的最小距离平方 */
int val(const Coor &p, const Filt &f) {
  int d = 0;
  for (int k = 0; k < K; ++k) {
    if (p[k] < f.l[k]) d+=sqr(f.l[k]-p[k]);
    if (p[k] > f.r[k]) d+=sqr(p[k]-f.r[k]);
  }
  return d;
}
/* 找到离 q 最近点距离平方 ans */
void fnd(Node *p, const Coor &q) {
  int d0, dl, dr;
  d0 = sqr(p->p[0]-q[0])+sqr(p->p[1]-q[1]);
  if (ans > d0) ans = d0;
  dl = p->l ? val(q, p->l->f) : inf;
  dr = p->r ? val(q, p->r->f) : inf;
  if (dl < dr) {
    if (dl < ans) fnd(p->l, q);
    if (dr < ans) fnd(p->r, q);
  } else {
    if (dr < ans) fnd(p->r, q);
    if (dl < ans) fnd(p->l, q);
  }
}
/* count the number of point in area f */
int cnt(Node *p, const Filt &f) {
  if (p) {
    if (f.fit(p->f)) return p->s;
    int ret = f.in(p->p);
    int m = p->p[p->k];
    if (f.l[p->k] <= m) ret += cnt(p->l, f);
    if (f.r[p->k] >= m) ret += cnt(p->r, f);
    return ret;
  }
  return 0;
}
```

2.4 Functional Segment Tree

```
struct Node {
  int c;
  Node *l, *r;
} *root[N + 9], *e, E[24 * N + 9];
void inc(Node *&p, int l, int r, int x) {
  if (!p) p = e++;
  ++p->c;
  if (l == r) return;
  int m = (l + r) >> 1;
  if (x <= m) inc(p->l, l, m, x);
  else inc(p->r, m + 1, r, x);
}
Node *merge(Node *p, Node *q) {
  if (!p) return q;
  if (!q) return p;
  p->c += q->c;
  p->l = merge(p->l, q->l);
  p->r = merge(p->r, q->r);
  return p;
}
```

3 Graph Theory

3.1 Maximal Clique

```
int n, mc[N], lst[N][N], ans;
bool g[N][N], chk;
void dfs(int sz) {
  int i, j, k;
  if(!len[sz - 1]) {
    if(ans < sz) ans = sz, chk = 1;
    return;
  }
  for(k = 1; k <= lst[sz - 1][0] && !chk; ++k) {
    if(sz + lst[sz - 1][0] - k <= ans) break;
    i = lst[sz - 1][k];
    if(sz + mc[i] <= ans) break;
    for(j = k + 1, lst[sz][0] = 0; j < lst[sz - 1][0]; ++j)
      if(g[i][lst[sz - 1][j]]) lst[sz][++lst[sz][0]] = lst[sz - 1][j];
    dfs(sz + 1);
  }
}
void max_cluster() {
  int i, j;
  mc[n] = ans = 1;
  for(i = n - 1; i; --i) {
    found = 0, len[0] = 0;
    for(j = i + 1; j <= n; ++j)
      if(g[i][j]) lst[0][++lst[0][0]] = j;
    dfs(1);
    mc[i] = ans;
  } // return ans;
}
ULL g[N]; // enum cliques O(3^{n/3})
int lowbit(ULL s) { return lbt[s & -s]; }
bool dfs(ULL cur, ULL alw, ULL fbd) {
  if(!alw && !fbd) output();
  if(!alw) return 0;
  ULL z = alw & ~g[lowbit(alw | fbd)];
  for(int u = lowbit(z); u < n; u += lowbit(z >> (u + 1)) + 1) {
    if(dfs(cur | (1ULL << u), alw & g[u], fbd & g[u]))
      return 1;
    alw ^= 1ULL << u, fbd |= 1ULL << u;
  } return 0;
} // dfs(0, (1ULL << n) - 1, 0);
```

3.2 Kosaraju Algo for SCC

```
int n, pre[maxn], rev[maxn], seq[maxn];
int tot, scc[maxn]; bool vis[maxn];
struct Edge { int nxt, v; }e[maxm << 1 | 1];
void addEdge(int u, int v) {
  e[++tot] = (Edge){pre[u], v},pre[u] = tot;
  e[++tot] = (Edge){rev[v], u},rev[v] = tot;
}
void dfs1(int u) {
  vis[u] = 1;
  for(int it = pre[u]; it; it = e[it].nxt)
    if(!vis[e[it].v]) dfs1(e[it].v);
  seq[++tot] = u;
}
void dfs2(int u) {
  scc[u] = tot;
  for(int it = rev[u]; it; it = e[it].nxt)
    if(!scc[e[it].v]) dfs2(e[it].v);
}
void SCC() { // init pre, rev, scc, vis = 0
  tot = 0;
  for(int i = 1; i <= n; ++i) if(!vis[i]) dfs1(i);
  tot = 0;
  for(int i = n; i; --i)
    if(!scc[seq[i]]) ++tot, dfs2(seq[i]);
}
```

3.3 Tarjan for SCC, Cut Vertex, Bridge

```
void dfs(int u) { /* SCC */
  dn[u] = lw[u] = ++tm;
  in[u] = true;
  s[t++] = u;
  int w;
  for (Edge *p = lnk[u]; p; p = p->x) {
    if (!dn[w = p->v]) {
      dfs(w);
      lw[u] = min(lw[u], lw[w]);
    } else if (in[w])
      lw[u] = min(lw[u], dn[w]);
  }
  if (dn[u] == lw[u]) {
    ++cnt;
    do {
      w = s[--t];
      in[w] = false;
    } while (u != w);
  }
}
void dfs(int u) { /* cut */
  lw[u] = dn[u] = ++tm;
  for (nod *p = l[u]; p; p = p->x) {
    if (!dn[k = p->v]) {
      dfs(k);
      if (lw[k] < lw[u]) lw[u] = lw[k];
      if (u != root && lw[k] >= dn[u])
        cut[u] = true;
    } else if (dn[k] < lw[u])
      lw[u] = dn[k];
  }
  if (u == root && child > 1) cut[u] = true;
}
void dfs(int idx, int u) { /* bridge */
  low[u] = dfn[u] = ++cnt;
  for (const auto &e : lnk[u])
    if (e.idx != idx) {
      int v = e.v;
      if (!dfn[v]) {
        dfs(e.idx, v);
        low[u] = min(low[u], low[v]);
        if (dfn[u] < low[v]) brg = e;
      } else {
        low[u] = min(low[u], dfn[v]);
      }
    }
}
```

3.4 Dominator Tree

```
int n, m, S, deg[maxn], que[maxn], L, R;
int dep[maxn], fa[maxd][maxn], sz[maxn];
vector<int> e[maxn]; int ctr[maxn];
int lca(int u, int v) {
  if(dep[u] > dep[v]) swap(u, v);
  for(int i = dep[v] - dep[u], j = 0; i; i >>= 1, ++j)
    if(i & 1) v = fa[j][v];
  if(u == v) return u;
  for(int i = maxd - 1; i >= 0; --i)
    if(fa[i][u] != fa[i][v])
      u = fa[i][u], v = fa[i][v];
  return fa[0][u];
}
int solve() {
  memcpy(ctr + 1, deg + 1, n * sizeof(int));
  L = R = 0, que[R++] = S;
  while(L < R) {
    int u = que[L++];
    for(auto &v : e[u]) if(!(--ctr[v])) que[R++] = v;
  }
  for(int i = 0; i < R; ++i) {
    int u = que[i];
    dep[u] = dep[fa[0][u]] + 1;
    for(int d = 1; d < maxd; ++d) fa[d][u] = fa[d - 1][fa[d - 1][u]];
    for(auto &v : e[u])
      fa[0][v] = fa[0][v] ? u : lca(u, fa[0][v]);
  }
  int ans = 0;
  for(int i = R - 1; i > 0; --i) {
    int u = que[i];
    ++sz[u]; sz[fa[0][u]] += sz[u];
    if(ans < sz[u]) ans = sz[u];
  } return ans;
}
```

3.5 Tree Heavy-Light Decomposition

```
void dfs(int u) {
  s[u] = 1; c[u] = 0;
  for (Edge *p = lnk[u]; p; p = p->x)
    if (p->v != f[u]) {
      f[p->v] = u;
      g[p->v] = p->w;
      d[p->v] = d[u] + 1;
      dfs(p->v);
      s[u] += s[p->v];
      if (s[c[u]] < s[p->v]) c[u] = p->v;
    }
}
void hfs(int u) {
  dfn[u] = ++cnt;
  val[cnt] = g[u];
  if (c[u]) t[c[u]] = t[u], hfs(c[u]);
  for (Edge *p = lnk[u]; p; p = p->x)
    if (p->v != f[u] && p->v != c[u])
      t[p->v] = p->v, hfs(p->v);
}
void change(int u, int v, int w) {
  int t1, t2;
  for (;;) {
    t1 = t[u], t2 = t[v];
    if (t1 == t2) break;
    if (d[t1] > d[t2]) {
      set(0, dfn[t1], dfn[u], w);
      u = f[t1];
    } else {
      set(0, dfn[t2], dfn[v], w);
      v = f[t2];
    }
  }
  if (u != v) {
    if (d[u] > d[v]) u ^= v ^= u ^= v;
    set(0, dfn[c[u]], dfn[v], w);
  }
} // solve :
  f[1] = 1; g[1] = -233; d[1] = 0;
  dfs(1);
  cnt = 0; t[1] = 1;
  hfs(1);
  build(0, 1, n);
```

3.6 Chu-Liu's Algo for MSA

```
int zle(int n, int root) {
  int ans = 0, cnt;
  LOOP: {
    for (int i = 0; i < n; ++i) inw[i] = INF;
    for (int i = 0; i < m; ++i)
      if (u[i] != v[i] && inw[v[i]] > w[i])
        inw[v[i]] = w[i], pre[v[i]] = u[i];
    inw[root] = 0;
    for (int i = 0; i < n; ++i) if (inw[i] == INF) return -1;
    cnt = 0;
    memset(vis, -1, sizeof(vis));
    memset(blg, -1, sizeof(blg));
    for (int i = 0, j; i < n; ++i) {
      ans += inw[i];
      for (j = i; vis[j] != i && blg[j] == -1 && j != root; j = pre[j]) vis[j] = i;
      if (j != root && blg[j] == -1) {
        for (int k = pre[j]; k != j; k = pre[k]) blg[k] = cnt;
        blg[j] = cnt++;
      }
    }
    if (cnt == 0) return ans;
    for (int i = 0; i < n; ++i) if (blg[i] == -1) blg[i] = cnt++;
    for (int i = 0, j; i < m; ++i) {
      j = v[i];
      u[i] = blg[u[i]]; v[i] = blg[v[i]];
      if (u[i] != v[i]) w[i] -= inw[j];
    }
    n = cnt; root = blg[root];
  } goto LOOP;
}
```

3.7 Kuhn-Munkres Algo

```
bool aug(int i) {
  usx[i] = true;
  for (int j = 1; j <= b; ++j)
    if (!usy[j]) {
      int d = f[x[i]][y[j]] - (lbx[i] + lby[j]);
      if (d) {
        slk[j] = min(slk[j], d);
      } else {
        usy[j] = true;
        if (!mat[j] || aug(mat[j])) return (mat[j] = i);
      }
    }
  return false;
}
void fix() {
  int d = inf;
  for (int j = 1; j <= b; ++j) if (!usy[j]) d = min(d, slk[j]);
  for (int i = 1; i <= a; ++i) if (usx[i]) lbx[i] += d;
  for (int j = 1; j <= b; ++j)
    if (usy[j]) lby[j] -= d;
    else slk[j] -= d;
}
int km() {
  memset(lby, 0, sizeof lby);
  for (int i = 1; i <= a; ++i) {
    lbx[i] = inf;
    for (int j = 1; j <= b; ++j)
      lbx[i] = min(lbx[i], f[x[i]][y[j]]);
  }
  memset(mat, 0, sizeof mat);
  for (int i = 1; i <= a; ++i) {
    for (int j = 1; j <= b; ++j) slk[j] = inf;
    for (;;) {
      memset(usx, 0, sizeof usx);
      memset(usy, 0, sizeof usy);
      if (aug(i)) break;
      else fix();
    }
  }
  int ret = 0;
  for (int j = 1; j <= b; ++j) ret += f[x[mat[j]]][y[j]];
  return ret;
}
```

3.8 Dinic's Algo

```
const LL INF = 0x3f3f3f3f3f3f3f3fLL;
int N, M, S, T, lev[maxn], lnk[maxn], cur[maxn];
struct Edge {int nxt,v; LL w;} e[maxm<<1|1];
```

```
void addEdge(int u, int v, LL w) {
  e[M] = (Edge){lnk[u], v, w}, lnk[u] = M++;
  e[M] = (Edge){lnk[v], u, 0}, lnk[v] = M++;
}
bool bfs() {
  int L = 0, R = 0; static int que[maxn];
  memset(lev, -1, N * sizeof(int));
  lev[S] = 0, que[R++] = S;
  while(L < R) {
    int u = que[L++];
    for(int it = lnk[u]; ~it; it = e[it].nxt)
      if(e[it].w > 0 && lev[e[it].v] == -1)
        lev[e[it].v] = lev[u] + 1,
        que[R++] = e[it].v;
  }
  return lev[T] != -1;
}
LL dfs(int u, LL upp) {
  if(u == T) return upp; // if !upp
  LL ret = 0, tmp;
  for(int &it = cur[u]; ~it; it = e[it].nxt)
    if(lev[e[it].v] == lev[u] + 1 && e[it].w > 0
    && (tmp = dfs(e[it].v, std::min(upp - ret, e[it].w))) > 0) {
      e[it].w -= tmp, e[it ^ 1].w += tmp;
      if((ret += tmp) == upp) break;
    }
  if(!ret) lev[u] = -1;
  return ret;
} // N, M = 0, S = N-2, T = N-1, lnk = -1
void dinic() {
  int flow = 0, tmp;
  while(bfs()) {
    memset(cur, lnk, N * sizeof(int));
    while((tmp = dfs(S, INF))) flow += tmp;
  }
}
```

3.9 Sap

```
int sap(int u, int flw) {
  if (u == T) return flw;
  int det, sum = 0;
  for (Edge *p = lnk[u]; p; p = p->x)
    if (h[u] == h[p->v] + 1 && p->c) {
      det = sap(p->v, min(p->c, flw - sum));
      p->c -= det;
      p->inv->c += det;
      if ((sum += det) == flw) return sum;
    }
  if (h[S] >= num) return sum;
  if (!--g[h[u]]) h[S] = num;
  ++g[++h[u]];
  return sum;
}
int solve() {
  g[0] = num;
  for (int i = 1; i <= num; ++i) g[i] = h[i] = 0;
  int mxf = 0;
  while (h[S] < num) mxf += sap(S, inf);
  return mxf;
}
```

3.10 Preflow Push

```
int main() {
  scanf("%d%d", &n, &m);
  while (m--) {
    scanf("%d%d%d", &u, &v, &t);
    c[u][v] = t;
  }
  src = 1;
  snk = n;
  h[src] = n;
  for (u = 1; u <= n; ++u) {
    e[u] = f[src][u] = c[src][u];
    if (e[u] && u != snk) ++s[h[q[r++] = u]];
  }
  while (l < r) {
    u = q[l++];
    for (v = 1; v <= n && e[u]; ++v)
      if (h[u] == h[v] + 1) {
        if (f[u][v] < c[u][v]) {
          t = min(e[u], c[u][v] - f[u][v]);
          f[u][v] += t;
          e[u] -= t;
          e[v] += t;
          if (e[v] == t && v != src && v != snk) q[r++] = v;
        }
        if (f[v][u]) {
          t = min(e[u], f[v][u]);
          f[v][u] -= t;
          e[u] -= t;
          e[v] += t;
          if (e[v] == t && v != src && v != snk) q[r++] = v;
        }
      }
    if (e[u] && u != snk && u != src) {
      --s[t = h[u]];
      h[u] = 2 * n;
      for (v = 1; v <= n; ++v)
        if (h[u] > h[v] && (f[u][v] < c[u][v] || f[v][u])) h[u] = h[v];
      ++s[++h[q[r++] = u]];
      if (!s[t])
        for (v = 1; v <= n; ++v)
          if (t < h[v] && h[v] < n) {
            --s[h[v]];
            ++s[h[v] = n];
          }
    }
  }
  printf("%d\n", e[snk]);
}
```

3.11 Stoer-Wagner Algo for Global Cut

```
int StoerWagner(int n) {
  int i, j, k, s, t, p = n, min, cut = inf, dist[maxn];
  bool d[maxn], visit[maxn];
  memset(d, false, sizeof(d)); // merged or not
  while (--p > 0) {
    memset(visit, false, sizeof(visit)); // in S or not
    memset(dist, 0, sizeof(dist)); // sum of weights in/not in S
    for (i = 0; d[i]; ++i);
    visit[i] = true;
    for (j = 0; j < n; j++)
      if (!d[j] && !visit[j])
        dist[j] = g[j][i];
    t = i, s = i;
    for (; i < n; i++) {
      for (min = 0, k = i, j = 0; j < n; j++)
        if (!d[j] && !visit[j] && dist[j] > min)
          min = dist[k = j];
      if (!min) break;
      visit[k] = true;
      for (j = 0; j < n; j++)
        if (!d[j] && !visit[j])
          dist[j] += g[j][k];
      s = t; // last element
      t = k; // current element
    }
    if (cut > dist[t]) cut = dist[t];
    d[t] = true;
    for (i = 0; i < n; i++)
      if (!d[i]) {
        g[s][i] += g[t][i];
        g[i][s] += g[i][t];
      }
  }
  return cut;
}
```

3.12 SPFA for MCMF

```
const static int INF = 0x3f3f3f3f;
int n, m, s, t, flow, cost, lnk[N], d[N], p[N], a[N]; bool inq[N];
struct Edge { int nxt, v, w, c; } e[M << 1];
void AddEdge(int u, int v, int w, int c) {
  e[m] = (Edge){lnk[u],v,w, c},lnk[u] = m++;
  e[m] = (Edge){lnk[v],u,0,-c},lnk[v] = m++;
}
bool BellmanFord() {
  memset(d, 0x3f, n * sizeof(int));
  memset(inq, 0, n * sizeof(bool));
  d[s] = 0; inq[s] = 1; p[s] = 0; a[s] = INF;
  std::queue<int> Q; Q.push(s);
  while(!Q.empty()) {
    int u = Q.front(); Q.pop(); inq[u] = 0;
    for(int it = lnk[u];~it;it = e[u].nxt) {
      int v = e[it].v;
      if(e[it].w&&d[v] > d[u] + e[it].c) {
        d[v] = d[u] + e[it].c; p[v] = it;
        a[v] = std::min(a[u], e[it].w);
        if(!inq[v])Q.push(v),inq[v] = 1;
      }
    }
  }
  if(d[t] == INF) return 0;
  flow += a[t]; cost += d[t] * a[t];
  for(int u = t; u != s; u = e[p[u] ^ 1].v)
    e[p[u]].w += a[t],
    e[p[u] ^ 1].w -= a[t];
  return 1;
}// flow = cost = 0; while(BellmanFord());
```

3.13 Prim-Dual Algo

```
const int V=440, E=V*2, maxint=0x3F3F3F3F;
struct etype {
  int t, c, u;
  etype *next, *pair;
  etype() {}
  etype(int T, int C, int U, etype* N): t(T), c(C), u(U), next(N) {}
  void* operator new(unsigned, void* p){return p;}
} *e[V], Te[E+E], *Pe;
int S, T, n, piS, cost;
bool v[V];
void addedge(int s, int t, int c, int u) {
  e[s] = new(Pe++) etype(t, +c, u, e[s]);
  e[t] = new(Pe++) etype(s, -c, 0, e[t]);
  e[s]->pair = e[t];
  e[t]->pair = e[s];
}
int aug(int no, int m) {
  if (no == T) return cost += piS * m, m;
  v[no] = true;
  int l = m;
  for (etype *i = e[no]; i; i = i->next)
    if (i->u && !i->c && !v[i->t]) {
      int d = aug(i->t, l < i->u ? l : i->u);
      i->u -= d, i->pair->u += d, l -= d;
      if (!l) return m;
    }
  return m - l;
}
bool modlabel() {
  static int d[V]; memset(d, 0x3F, sizeof(d)); d[T] = 0;
  static deque<int> Q; Q.push_back(T);
  while(Q.size()) {
    int dt, no = Q.front(); Q.pop_front();
    for(etype *i = e[no]; i; i = i->next)
      if(i->pair->u && (dt = d[no] - i->c) < d[i->t])
        (d[i->t] = dt) <= d[Q.size() ? Q.front() : 0]
          ? Q.push_front(i->t) : Q.push_back(i->t);
  }
  for(int i = 0; i < n; ++i)
    for(etype *j = e[i]; j; j = j->next)
      j->c += d[j->t] - d[i];
  piS += d[S];
  return d[S] < maxint;
}
int ab[V], *pab[V], w[V];
struct lt {
  bool operator()(int* p1,int* p2) {return *p1 < *p2;}
};
int main() { // POJ 3680
  int t;
  scanf("%d",&t);
  while(t--) {
    memset(e,0,sizeof(e));
    Pe = Te;
    static int m, k;
    scanf("%d %d", &m, &k);
    int abz = 0;
    for(int i = 0; i < m; ++i) {
      scanf("%d", pab[abz] = &ab[abz]), abz++;
      scanf("%d", pab[abz] = &ab[abz]), abz++;
      scanf("%d", &w[i]);
    }
    sort(&pab[0], &pab[abz], lt());
    int c=0xDEADBEEF; n=0;
    for(int i = 0; i < abz; ++i) {
      if(c != *pab[i]) c = *pab[i], ++n;
      *pab[i] = n;
    }
    ++n, S = 0, T = n++;
    for(int i = 0; i < T; ++i) addedge(i, i+1, 0, k);
    for(int i = 0; i < m; ++i) addedge(ab[i+i], ab[i+i+1], -w[i], 1);
    piS = cost = 0;
    while(modlabel())
      do memset(v, 0, sizeof(v));
      while(aug(S, maxint));
    printf("%d\n", -cost);
  }
  return 0;
}
```

3.14 Simplex

```
int n,m,T;
double a[maxn][maxn];
int Ans[maxn],pt[maxn];
void pivot(int l,int i){
  double t;
  swap(Ans[l+n],Ans[i]);
  t=-a[l][i];a[l][i]=-1;for(int j=0;j<=n;j++)a[l][j]/=t;
  for(int j=0;j<=m;j++)if(a[j][i]&&j!=l){t=a[j][i];a[j][i]=0;
    for(int k=0;k<=n;k++)a[j][k]+=t*a[l][k];
  }
}
void solve(){
  scanf("%d%d%d",&n,&m,&T);
  for(int i=1;i<=n;i++)scanf("%lf",&a[0][i]);
  for(int i=1;i<=m;i++){
    for(int j=1;j<=n;j++)scanf("%lf",&a[i][j]),a[i][j]=-a[i][j];scanf("%lf",&a[i][0]);
  }
  for(int i=1;i<=n;i++)Ans[i]=i;
  double t;
  for(;;){int l=0;t=-1e-8;
    for(int j=1;j<=m;j++)if(a[j][0]<t)t=a[l=j][0];if(!l)break;
    int i=0;for(int j=1;j<=n;j++)if(a[l][j]>1e-8){i=j;if(rand()&1)break;}
    if(!i)return puts("Infeasible");
    pivot(l,i);
  }
  for(;;){int i=0;t=1e-8;
    for(int j=1;j<=n;j++)if(a[0][j]>t){t=a[0][i=j];if(rand()&1)break;}if(!i)break;
    int l=0;t=1e30;
    for(int j=1;j<=m;j++)if(a[j][i]<-1e-8){double tmp;
      tmp=-a[j][0]/a[j][i];
      if(t>tmp)t=tmp,l=j;
    }
    if(!l)return puts("Unbounded");
    pivot(l,i);
  }
  printf("%.10lf\n",a[0][0]);
  if(T){
    for(int i=n+1;i<=n+m;i++)pt[Ans[i]]=i-n;
    for(int i=1;i<=n;i++)printf("%.9lf ",pt[i]?a[pt[i]][0]:0);
  }
}
```

4 Mathematics

4.1 Interval Sieve

```
const int maxn = 100001, delta = 200000, maxd = 5001, maxl = delta + 1;
const int BSZ = 6, BASE = 1 << BSZ, STA = 36, BLEN = 6;
const char *cipher = "......"; LL tf[maxd];
int tot, pr[maxn >> 3 | 1], d[maxn];
LL tf[maxd];
LL solve(int L, int R) { // [L, R]
  if(L > R) return 0;
  static int _val[maxl], _rem[maxl];
  int *val = _val - L, *rem = _rem - L, lim = (int)ceil(sqrt(R));
  for(int i = L; i <= R; ++i)
    val[i] = 1, rem[i] = i;
  for(int i = 0, pk; pr[i] <= lim; ++i)
    for(int j = R / pr[i] * pr[i]; j >= L; j -= pr[i]) {
      for(pk = 1; rem[j] % pr[i] == 0; rem[j] /= pr[i], pk *= pr[i]);
      val[j] *= pk - 1;
    }
  LL ret = 0;
  for(int i = L; i <= R; ++i) {
    if(rem[i] > 1) val[i] *= rem[i] - 1;
    ret += val[i];
  }
  return ret;
}
LL solve(int n) { // [1, n]
  int idx = n / delta, low = idx * delta, upp = low + delta;
  return n - low <= upp - n ? tf[idx] + solve(low + 1, n) : tf[idx + 1] - solve(n + 1, upp);
}
void init() {
  for(int i = 2; i < maxn; ++i) {
    if(!d[i]) pr[tot++] = d[i] = i;
    for(int j = 0, k; (k = i * pr[j]) < maxn; ++j) {
      d[k] = pr[j];
      if(d[i] == pr[j]) break;
    }
  }
  const char *ptr = cipher;
  for(int i = 1; i < maxd; ++i) {
    for(int j = 0; j < BLEN; ++j, ++ptr)
      tf[i] = tf[i] << BSZ | (*ptr - STA - (*ptr > '\\'));
    tf[i] += tf[i - 1];
  }
  for(int i = 1; i < maxd; ++i) tf[i] += tf[i - 1];
  for(int n; read(n); writeln(solve(n)));
}
```

4.2 Miller-Rabin & Pollard's rho Algo

```
int cnt; LL n, p[maxc];
bool miller_rabin(LL n) {
  if(n == 2) return 1;
  if(n < 2 || !(n & 1)) return 0;
  LL s = 0, r = n - 1;
  for( ; !(r & 1); r >>= 1) ++s;
  for(int i = 0; i < maxt; ++i) {
    LL cur = mod_pow(rand() % (n - 2) + 2, r, n), nxt;
    for(int j = 0; j < s; ++j) {
      nxt = mod_mul(cur, cur, n);
      if(nxt == 1 && cur != 1 && cur != n - 1) return 0;
      cur = nxt;
    }
    if(cur != 1) return 0;
  }
  return 1;
}
LL pollard_rho(LL n) {
  LL x = rand() % n, y = x, c = rand() % n, r;
  for(LL i = 1, k = 2; ; ++i) {
    if(i == k) y = x, k <<= 1;
    x = mod_add(mod_mul(x, x, n), c, n);
    r = std::__gcd(std::abs(y - x), n);
    if(r != 1) return r;
  }
}
void decompose(LL n) {
  for(int i = 0; i < cnt; ++i)
    if(n % p[i] == 0) n /= p[i], p[cnt++] = p[i];
  if(n < maxp) { // 1e6
    while(n > 1) p[cnt++] = d[n], n /= d[n];
  } else if(miller_rabin(n)) {
    p[cnt++] = n;
  } else {
    LL fact; while((fact = pollard_rho(n)) == n);
    decompose(fact), decompose(n / fact);
  }
}
```

4.3 Gauss for Matrix

```
int gauss(int n, int m) {
  int rank = 0;
  for (int i = 0; i < m; ++i) {
    int k = rank;
    for (int j = rank + 1; j < n; ++j)
      if (fabs(a[k][i]) < fabs(a[j][i]))
        k = j;
    if (fabs(a[k][i]) < eps) continue;
    swap(a[rank], a[k]);
    double t = a[rank][i];
    for (int j = 0; j < m; ++j) a[rank][j] /= t;
    for (int k = 0; k < n; ++k) {
      if (k != rank) {
        t = a[k][i];
        for (int j = 0; j < m; ++j)
          a[k][j] -= a[rank][j] * t;
      }
    }
    if (++rank == n) break;
  }
  return rank;
}
```

4.4 FFT modulo integer

```
const int maxLen = 18, maxm = 1 << maxLen | 1, mod;
const LL maxv = (LL)1e13; // 1e14, 1e15
const DB pi = acos(-1.0); // double
struct complex {
  DB r, i;
  complex() {}
  complex(DB r, DB i) : r(r), i(i) {}
  complex operator + (complex const &t) const {
    return complex(r + t.r, i + t.i);
  } complex operator - (complex const &t) const {
    return complex(r - t.r, i - t.i);
  } complex operator * (complex const &t) const {
    return complex(r * t.r - i * t.i, r * t.i + i * t.r);
  } complex conj() const {
    return complex(r, -i);
  }
} w[maxm]; int nlim, sp, msk;
void FFT_init() {
  for(int i = 0, ilim = 1 << maxLen; i < ilim; ++i) {
    int j = i, k = ilim >> 1; // 2 pi / ilim
    for( ; !(j & 1) && !(k & 1); j >>= 1, k >>= 1);
    w[i] = complex(cos(pi / k * j), sin(pi / k * j));
  }
  nlim = maxv / (mod - 1) / (mod - 1);
  for(sp = 1; 1 << (sp << 1) < mod; ++sp);
  msk = (1 << sp) - 1; // init
}
void FFT(int n, complex a[], int flag) {
  static int bitLen = 0, bitRev[maxm] = {};
  if(n != (1 << bitLen)) {
    for(bitLen = 0; 1 << bitLen < n; ++bitLen);
    for(int i = 1; i < n; ++i)
      bitRev[i] = (bitRev[i >> 1] >> 1) | ((i & 1) << (bitLen - 1));
  }
  for(int i = 0; i < n; ++i)
    if(i < bitRev[i]) std::swap(a[i], a[bitRev[i]]);
  for(int i = 1, d = 1; d < n; ++i, d <<= 1)
    for(int j = 0; j < n; j += d << 1)
      for(int k = 0; k < d; ++k) {
        complex &AL = a[j + k], &AH = a[j + k + d];
        complex TP = w[k << (maxLen - i)] * AH;
        AH = AL - TP, AL = AL + TP;
      }
  if(flag == -1)
    return;
  std::reverse(a + 1, a + n);
  for(int i = 0; i < n; ++i) a[i].r /= n, a[i].i /= n;
}
void polyMul(int aLen, int a[], int bLen, int b[], int c[]) { // c not in {a, b}
  static complex A[maxm], B[maxm], C[maxm], D[maxm];
  int cLen = aLen + bLen - 1, len;
  for(len = 1; len < cLen; len <<= 1);
  if(std::min(aLen, bLen) <= nlim) {
    for(int i = 0; i < len; ++i)
      A[i] = complex(i < aLen ? a[i] : 0, i < bLen ? b[i] : 0);
    FFT(len, A, 1);
    complex tr(0, -0.25);
    for(int i = 0, &&j = (len - i) & (len - 1); i < len; ++i)
      B[i] = (A[i] * A[i] - (A[j] * A[j]).conj()) * tr;
    FFT(len, B, -1);
    for(int i = 0; i < cLen; ++i) c[i] = (LL)(B[i].r + 0.5) % mod;
    return;
  } // if min(aLen, bLen) * mod <= maxv
  for(int i = 0; i < len; ++i) {
    A[i] = i < aLen ? complex(a[i] & msk, a[i] >> sp) : complex(0, 0);
    B[i] = i < bLen ? complex(b[i] & msk, b[i] >> sp) : complex(0, 0);
  }
  FFT(len, A, 1), FFT(len, B, 1);
  complex trL(0.5, 0), trH(0, -0.5), tr(0, 1);
  for(int i = 0, &&j = (len - i) & (len - 1); i < len; ++i) {
    complex AL = (A[i] + A[j].conj()) * trL;
    complex AH = (A[i] - A[j].conj()) * trH;
    complex BL = (B[i] + B[j].conj()) * trL;
    complex BH = (B[i] - B[j].conj()) * trH;
    C[i] = AL * (BL + BH * tr);
    D[i] = AH * (BL + BH * tr);
  }
  FFT(len, C, -1), FFT(len, D, -1);
  for(int i = 0; i < cLen; ++i) {
    int v11 = (LL)(C[i].r + 0.5) % mod, v12 = (LL)(C[i].i + 0.5) % mod;
    int v21 = (LL)(D[i].r + 0.5) % mod, v22 = (LL)(D[i].i + 0.5) % mod;
    c[i] = ((((LL)v22 << sp) + v12 + v21) << sp) + v11) % mod;
  }
}
```

4.5 NTT & primes

469762049 26 3; 167772161 25 3; 754974721 24 11; 998244353 23 3; 985661441 22 3; 1004535809 21 3;

```
const int maxLen = 18, maxm = 1 << maxLen | 1, mod = 998244353, gen = 3;
int w[maxm], inv2[maxLen + 1];
int get_ori(int mod) {
  int cnt = 0, pr[10], tmp = mod - 1;
  for(int i = 2; i * i <= tmp; ++i)
    if(tmp % i == 0)
      for(pr[cnt++] = i; tmp % i == 0; tmp /= i);
  if(tmp > 1)
    pr[cnt++] = tmp;
  for(int ori = 2; ; ++ori) {
    bool flag = mod_pow(ori, mod - 1) == 1;
    for(int i = 0; i < cnt && flag; ++i)
      flag &= mod_pow(ori, (mod - 1) / pr[i]) != 1;
    if(flag) return ori;
  }
}
void NTT_init() {
  gen = get_ori(mod);
  w[0] = 1; // make sure that mod = 2 ^ maxLen * k + 1
  w[1] = mod_pow(gen, (mod - 1) >> maxLen); // avoid maxm
  for(int i = 2; i < maxm; ++i) // w[1 << maxLen] = 1
    w[i] = (LL)w[i - 1] * w[1] % mod;
  inv2[0] = 1;
  inv2[1] = (mod + 1) >> 1;
  for(int i = 2; i <= maxLen; ++i)
    inv2[i] = (LL)inv2[i - 1] * inv2[1] % mod;
}
void NTT(int len, int x[], int flag) {
  static int bitLen = 0, ivs = 1, bitRev[maxm] = {};
  if(len != (1 << bitLen)) {
    for(bitLen = 0; 1 << bitLen < len; ++bitLen);
    ivs = inv2[bitLen];
    for(int i = 1; i < len; ++i)
      bitRev[i] = (bitRev[i >> 1] >> 1) | ((i & 1) << (bitLen - 1));
  }
  for(int i = 1; i < len; ++i)
    if(i < bitRev[i]) std::swap(x[i], x[bitRev[i]]);
  for(int i = 1, d = 1; d < len; ++i, d <<= 1)
    for(int j = 0; j < len; j += d << 1)
      for(int k = 0, *X = x + j; k < d; ++k) {
        int t = (LL)w[k << (maxLen - i)] * X[k + d] % mod; // avoid maxm
        X[d + k] = mod_sub(X[k], t);
        X[k] = mod_add(X[k], t);
      }
  if(flag != -1) return;
  std::reverse(x + 1, x + len);
  for(int i = 0; i < len; ++i) x[i] = (LL)x[i] * ivs % mod;
}
```

4.6 Polynomial Division

```
void polyInv(int n, int cur[], int nxt[]) {
  if(n == 1) { nxt[0] = mod_inv(cur[0]); return; }
  int plen = n + 1 >> 1;
  polyInv(plen, cur, nxt);
  static int h[maxm];
  polyMul(n, f, plen, g, n, h);
  polyMul(plen, g, n - plen, h + plen, n - plen, g + plen);
  for(int i = plen; i < n; ++i)
    g[i] = g[i] ? mod - g[i] : 0;
}
void polyDiv(int n, int a[], int m, int b[], int d[], int r[]) {
  if(n < m) {
    d[0] = 0;
    memcpy(r, a, n * sizeof(int));
    memset(r + a, 0, (m - n) * sizeof(int));
    return;
  }
  int plen = n - m + 1;
  static int x[maxn], y[maxn];
  memcpy(x, b, m * sizeof(int));
  if(m < plen) memset(x + m, 0, (plen - m) * sizeof(int));
  std::reverse(x, x + m);
  polyInv(plen, x, y);
  memset(y + plen, 0, (len - plen) * sizeof(int));
  std::reverse(x, x + n);
  polyMul(plen, x, plen, y, plen, x);
  std::reverse(x, x + plen);
  polyMul(plen, x, m, b, m, y);
  memcpy(d, x, plen * sizeof(int));
  for(int i = 0; i < m; ++i)
    r[i] = mod_sub(a[i], y[i]);
}
```

4.7 FWT for exclusive-or

```
void FWT_iterate(int x[], int len, int dt) {
  for(int i = 0; i < len; i += dt << 1)
    for(int j = 0; j < dt; ++j) {
      int t = x[i + dt + j];
      x[i + dt + j] = mod_sub(x[i + j], t);
      x[i + j] = mod_add(x[i + j], t);
    }
}
void FWT(int x[], int len, int flag) {
  if(flag > 0) {
    for(int i = 1; i < len; i <<= 1)
      FWT_iterate(x, len, i);
  } else {
    for(int i = len >> 1; i > 0; i >>= 1)
      FWT_iterate(x, len, i);
    for(int i = 0; i < len; ++i)
      x[i] = (LL)x[i] * inv_len % mod;
  }
}
```

4.8 factorial modulo prime powers

```
pair<int, LL> mod_fact(LL n, int p) {
  if(!n) return make_pair(1, 0);
  pair<int, LL> tmp = mod_fact(n / p, p);
  tmp.second += n / p;
  if(~(n / p) & 1) ret.first = (LL)ret.first * fact[n % p] % p;
  else ret.first = (LL)ret.first * (p - fact[n % p]) % p;
  return ret;
}
int p, sz; LL pp, md[maxn], c[maxn][maxn], s[maxp][maxn], f[maxn], g[3][maxn];
void poly_mul(LL f[], LL g[], LL h[]) {
  memcpy(h, f, sz * sizeof(LL)), memset(f, 0, sz * sizeof(LL));
  for(int i = 0; i < sz; ++i) if(g[i])
    for(int j = 0, k = i; k < sz; ++j, ++k) if(h[j])
      f[k] = mod_add(f[k], mod_mul(g[i], h[j], md[k]), md[k]);
}
void poly_val(LL f[], LL x, LL g[], LL h[]) {
  memset(g, 0, sz * sizeof(LL)), h[0] = 1;
  for(int i = 1; i < sz; ++i) h[i] = mod_mul(h[i - 1], x, pp);
  for(int i = 0; i < sz; ++i) if(f[i])
    for(int j = 0; j <= i; ++j) if(c[i][j]) {
      LL v = mod_mul(f[i], h[i - j], md[j]);
      if(!v) continue;
      g[j] = mod_add(g[j], mod_mul(v, c[i][j], md[j]), md[j]);
    }
}
pair<LL, LL> mod_fact2(LL n) {
  int len = 0; static int dig[maxl];
  for(LL tmp = n; tmp > 0; dig[len++] = tmp % p, tmp /= p);
  pair<LL, LL> ret = make_pair(1, 0);
  memset(f, 0, sz * sizeof(LL)), f[0] = 1;
  LL val = 0, lst;
  for(int i = len - 1; i >= 0; --i) {
    ret.first = mod_mul(ret.first, f[0], pp), ret.second += val;
    val *= p, lst = val;
    for(int j = 1; j <= dig[i]; ++j)
      ret.first = mod_mul(ret.first, ++val, pp);
    if(i == 0) continue;
    if(lst)
      if(p == 2) {
        poly_val(f, lst, g[0], g[1]);
        poly_mul(f, g[0], g[1]);
      } else { // p == 5
        poly_val(f, lst, g[0], g[2]);
        poly_mul(g[0], f, g[2]);
        poly_val(g[0], lst * 2, g[1], g[2]);
        poly_mul(g[1], g[0], g[2]);
        poly_val(g[1], lst, g[0], g[2]);
        poly_mul(f, g[0], g[2]);
      }
    if(dig[i]) {
      poly_val(s[dig[i]], lst * p, g[0], g[1]);
      poly_mul(f, g[0], g[1]);
    }
  }
  return ret;
}
void mod_fact2_init(int _p, int _sz) {
  p = _p, sz = _sz, md[sz] = pp = 1;
  for(int i = sz; i; --i) md[i] = (pp *= p);
  memset(s[0], 0, sz * sizeof(LL)), s[0][0] = 1;
  memcpy(s[1], s[0], sz * sizeof(LL));
  for(int i = 1; i < p; ++i)
    for(int j = sz - 1; j >= 0; --j) {
      s[1][j] = mod_mul(s[1][j], i, md[j]);
      if(j == 0) continue;
      s[1][j] = mod_add(s[1][j - 1] % md[j], s[1][j], md[j]);
    }
  for(int i = 2; i < p; ++i) {
    poly_val(s[1], i * p, s[i], g[0]);
    poly_mul(s[i], s[i - 1], g[0]);
  }
  for(int i = 0; i < sz; ++i) {
    c[i][0] = c[i][i] = 1;
    for(int j = 1; j < i; ++j)
      c[i][j] = mod_add(c[i - 1][j - 1] % md[j], c[i - 1][j], md[j]);
  }
}
```

5 String

5.1 KMP Algo

```
void KMP(char pat[], char tex[], int f[], int g[]) {
  int n = strlen(pat), m = strlen(tex);
  f[0] = f[1] = 0;
  for(int i = 1, j = 0; i < n; ++i) {
    while(j && pat[i] != pat[j]) j = f[j];
    f[i + 1] = pat[i] == pat[j] ? ++j : 0;
  }
  for(int i = 0, j = 0; i < m; ++i) {
    while(j && tex[i] != pat[j]) j = f[j];
    g[i] = tex[i] == pat[j] ? ++j : 0;
    if(j == n) j = f[j];
  }
}
```

5.2 Z Algo

```
void ZAlgo(char pat[], char tex[], int z[], int f[]) {
  int n = strlen(pat), m = strlen(tex);
  for(int i = 1, L, R = i, x; i < n; ++i) {
    if(i < R && i + z[i - L] < R) {
      z[i] = z[i - L];
      continue;
    }
    for(x = max(i, R); pat[x] == pat[x - i]; ++x);
    if((z[i] = x - i) > 0) L = i, R = x;
  } // update last cover [L, R)
  for(int i = 0, L, R = i, x; i < m; ++i) {
    if(i < R && i + z[i - L] < R) {
      f[i] = z[i - L];
      continue;
    }
    for(x = max(i, R); tex[x] && tex[x] == pat[x - i]; ++x);
    if((f[i] = x - i) > 0) L = i, R = x;
  }
}
```

5.3 Manacher Algo

```
void manacher(char str[], int h[]) {
  int n = strlen(str), m = 0;
  static char buf[M];
  for(int i = 0; i < n; ++i)
    buf[m++] = str[i], buf[m++] = '#';
  buf[m] = '\0';
  for(int i = 0, mx = 0, id = 0; i < m; ++i) {
    h[i] = i < mx ? std::min(mx - i, h[(id << 1) - i]) : 0;
    while(h[i] <= i && buf[i - h[i]] == buf[i + h[i]]) ++h[i];
    if(mx < i + h[i]) mx = i + h[i], id = i;
  }
}
```

5.4 Minimal Representing String

```
int MinRep(int *s, int l) {
  int i = 0, j = 1, k;
  while(i < l && j < l) {
    for(k = 0; k < l && s[i + k] == s[j + k]; ++k);
    if(k == l) return i;
    if(s[i + k] > s[j + k])
      if(i + k + 1 > j) i = i + k + 1;
      else i = j + 1;
    else if(j + k + 1 > i) j = j + k + 1;
    else j = i + 1;
  }
  return i < l ? i : j;
}
```

5.5 Aho-Corasick Automation

```
int tot;
struct Trie {
  int val, ch[maxc], fail, last;
} p[maxn];
void insert(char s[maxl]) {
  int rt = 0;
  for(int i = 0; s[i]; ++i) {
    int o = s[i] - 'A';
    if(!p[rt].ch[o])
```

```
      p[++tot] = p[0], p[rt].ch[o] = tot;
    rt = p[rt].ch[o];
  } ++p[rt].val;
}
void bfs() {
  int L = 0, R = 0; static int que[N];
  for(int i = 0; i < maxc; ++i)
    if(p[0].ch[i]) que[R++] = p[0].ch[i];
  while(L < R) {
    int u = que[L++];
    for(int i = 0; i < maxc; ++i) {
      if(!p[u].ch[i]) continue;
      int v = p[u].ch[i], w = p[u].fail;
      while(w && !p[w].ch[i]) w = p[w].fail;
      p[v].fail = w = p[w].ch[i];
      p[v].last = p[w].val ? w : p[w].last;
      que[R++] = v;
    }
  }
}
```

5.6 Suffix Array

```
int n, len, sa[N], rk[N], Log[N], st[D][N], *ht = st[0];
int m, f[N], g[N], c[N], slen[N], pos[N], val[N];
char str[N];
inline int lcp(int L, int R) {
  if(L == R) return N;
  int k = Log[R - (L++)];
  return std::min(st[L][k], st[R - (1 << k) + 1][k]);
}
void suffix_array() {
  int i, j, k, *x = f, *y = g; m = 256;
  scanf("%d", &n);
  for(i = 1; i <= n; ++i) {
    scanf("%s", str + len);
    slen[i] = strlen(str + len);
    pos[i] = len;
    for(int k = 0; k < slen[i]; ++k) val[len + k] = i;
    len += slen[i];
    str[len++] = '$';
  }
  memset(c, 0, m * sizeof(int));
  for(i = 0; i < len; ++i) ++c[x[i] = str[i]];
  for(i = 1; i < m; ++i) c[i] += c[i - 1];
  for(i = len - 1; i >= 0; --i) sa[--c[x[i]]] = i;
  for(k = 1; k <= len; k <<= 1, m = j) {
    j = 0;
    for(i = len - k; i < len; ++i) y[j++] = i;
    for(i = 0; i < len; ++i)
      if(sa[i] >= k) y[j++] = sa[i] - k;
    memset(c, 0, m * sizeof(int));
    for(i = 0; i < len; ++i) ++c[x[y[i]]];
    for(i = 0; i < m; ++i) c[i] += c[i - 1];
    for(i = len - 1; i >= 0; --i)
      sa[--c[x[y[i]]]] = y[i];
    std::swap(x, y);
    j = 1, x[sa[0]] = 0;
    for(i = 1; i < len; ++i)
      x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k] ? j - 1 : j++;
    if(j >= len) break;
  }
  for(i = 0; i < len; ++i) rk[sa[i]] = i;
  for(i = k = 0; i < len; ++i) {
    if(k) --k;
    if(!rk[i]) continue;
    for(j = sa[rk[i] - 1]; str[i + k] == str[j + k]; ++k);
    ht[rk[i]] = k;
  }
  for(i = 2; i <= len; ++i) Log[i] = Log[i >> 1] + 1;
  for(i = 1; 1 << i <= len; ++i) // ht = st[0]
    for(j = 0; j + (1 << i - 1) < len; ++j)
      st[i][j] = std::min(st[i - 1][j], st[i - 1][j + (1 << i - 1)]);
}
```

```
void build(int m) {
  int a1, a2, *x = t1, *y = t2;
  for (int i = 0; i < m; ++i) c[i] = 0;
  for (int i = 0; i < n; ++i) ++c[x[i] = s[i]];
  for (int i = 1; i < m; ++i) c[i] += c[i - 1];
  for (int i = n - 1; ~i; --i) sa[--c[x[i]]] = i;
  for (int k = 1; k <= n; k <<= 1) {
    int p = 0;
    for (int i = n - k; i < n; ++i) y[p++] = i;
    for (int i = 0; i < n; ++i) if (sa[i] >= k) y[p++] = sa[i] - k;
    for (int i = 0; i < m; ++i) c[i] = 0;
    for (int i = 0; i < n; ++i) ++c[x[y[i]]];
    for (int i = 1; i < m; ++i) c[i] += c[i - 1];
    for (int i = n - 1; ~i; --i) sa[--c[x[y[i]]]] = y[i];
    swap(x, y);
    p = 1;
    x[sa[0]] = 0;
    for (int i = 1; i < n; ++i) {
      a1 = sa[i - 1] + k < n ? y[sa[i - 1] + k] : -1;
            a2 = sa[i] + k < n ? y[sa[i] + k] : -1;
      x[sa[i]] = (y[sa[i - 1]] == y[sa[i]] && a1 == a2)? p - 1 : p++;
    }
    if (p >= n) break;
    m = p;
  }
  int k = 0;
  for (int i = 0; i < n; ++i) rank[sa[i]] = i;
  for (int i = 0; i < n; ++i) {
    k && --k;
    if (!rank[i]) continue;
    int j = sa[rank[i] - 1];
    while (s[i + k] == s[j + k]) ++k;
    h[rank[i]] = k;
  }
  for (int i = 1; i < n; ++i) f[0][i] = h[i];
  for (int j = 1; 1 << j <= n; ++j)
    for (int i = 1; i + (1 << j) <= n; ++i)
      f[j][i] = min(f[j - 1][i], f[j - 1][i + (1 << j - 1)]);
}
int lcp(int x, int y) {
  x = rank[x], y = rank[y];
  if (x > y) swap(x, y);
  int j = lg2[y - x++];
  return min(f[j][x], f[j][y - (1 << j) + 1]);
}
```

5.7 Suffix Automaton

```
int root , last , nodecnt;
int u[N << 1][26] , val[N << 1] , f[N << 1];
inline int newnode(int _val) {
    ++ nodecnt;
    memset(u[nodecnt] , 0 , sizeof(u[nodecnt]));
    val[nodecnt] = _val , f[nodecnt] = 0;
    return nodecnt;
}
void extend(int c) {
    int p = last , np = newnode(val[p] + 1);
    while (p && u[p][c] == 0)
        u[p][c] = np , p = f[p];
    if (p == 0)
        f[np] = root;
    else {
        int q = u[p][c];
        if (val[p] + 1 == val[q]) {
            f[np] = q;
        } else {
            int nq = newnode(val[p] + 1);
            memcpy(u[nq] , u[q] , sizeof(u[q]));
            f[nq] = f[q];
            f[q] = f[np] = nq;
            while (p && u[p][c] == q)
                u[p][c] = nq , p = f[p];
        }
    }
    last = np;
}
void work() {
    nodecnt = 0;
    root = last = newnode(0);
}
```

5.8 Palindrome Tree

```
struct PalinTree {
  char str[N];
  int n;
  int u[N][26];
  int len[N] , f[N] , cnt[N];
  int nodecnt , root;
  void init() {
    scanf("%s" , str);
    n = strlen(str);
    nodecnt = 2;
    len[1] = -1 , len[2] = 0;
    f[1] = 0 , f[2] = 1;
    memset(u[1] , 0 , sizeof(u[1]));
    memset(u[2] , 0 , sizeof(u[2]));
    root = 1;
    for (int i = 0 ; i < n ; ++ i)
      extend(i , str[i] - 'a');
  }
  void extend(int i , int c) {
    int p = root;
    while (str[i - 1 - len[p]] != str[i])
      p = f[p];
    int& pp = u[p][c];
    if (!pp) {
      pp = ++ nodecnt;
      len[pp] = len[p] + 2;
      cnt[pp] = 0;
      memset(u[pp] , 0 , sizeof(u[pp]));
      int q = f[p];
      while (q && str[i - 1 - len[q]] != str[i])
        q = f[q];
      f[pp] = q ? u[q][c] : 2;
    }
    ++ cnt[pp];
    root = pp;
  }
}
```

6 Geometry

6.1 Adaptive Simpson's Method

```
DB simpson(DB L, DB R) { DB M = L + (R - L) / 2; return (F(L) + 4 * F(M) + F(R)) * (R - L) / 6; }
DB asr(DB L, DB R, DB eps, DB area) {
  DB M = L + (R - L) / 2, La = simpson(L, M), Ra = simpson(M, R);
  if(fabs(L + R - area) <= 15 * eps) return L + R + (L + R - area) / 15;
  return asr(L, M, eps / 2, La) + asr(M, R, eps / 2, Ra);
}
DB asr(DB L, DB R, DB eps) { return asr(L, R, eps, simpson(L, R)); }
```

6.2 2D Convex Hull

```
int ConvexHull(Point *p, int n, Point *ch) {
  int m = 0;
  std::sort(p, p + n);
  for(int i = 0; i < n; ++i) {
    while(m > 1 && dcmp(area(ch[m - 2], ch[m - 1], p[i])) <= 0) --m;
      ch[m++] = p[i];
    }
    int k = m;
    for(int i = n - 2; i >= 0; --i) {
    while(m > k && dcmp(area(ch[m - 2], ch[m - 1], p[i])) <= 0) --m;
      ch[m++] = p[i];
    }
    return n > 1 ? m - 1 : m;
}
```

6.3 Halfplane Intersection

```
int HalfplaneInt(Line *a, int n, Point *poly) {
  int L = 0, R = 0; // [L, R]
  static Point p[N]; static Line q[N];
  q[0] = L[0];
  std::sort(a, a + n, ang_ord);
  for(int i = 1; i < n; ++i) {
    while(L < R && !OnLeft(a[i], p[R - 1])) --R;
    while(L < R && !OnLeft(a[i], p[L])) ++L;
    if(dcmp(det(q[R].vec, a[i].vec)))
      q[++R] = a[i];
    else if(OnLeft(q[R], a[i].ori))
      q[R] = a[i];
    if(L < R)
      p[R - 1] = LineInt(q[R - 1], q[R]);
  }
  while(L < R && !OnLeft(q[L], p[R - 1])) --R;
  if(R - L + 1 <= 2) return 0;
  p[R] = LineInt(q[R], q[L]);
  for(int i = L; i <= R; ++i) poly[i - L] = p[i];
  return R - L;
}
```

6.4 2D Geometry Fundamental

```
/* #include <complex>
using namespace std;
typedef complex<double> Point;
typedef Point Vector;
double Dot(Vector A, Vector B) { return real(conj(A) * B); }
double Cross(Vector A, Vector B) { return imag(conj(A) * B); }
Vector Rotate(Vector A, double rad) { return A * exp(Point(0, rad)); } */
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <vector>
#include <algorithm>
using namespace std;
const double PI = acos(-1), eps = 1e-10;
int dcmp(double x) {
  if(fabs(x) < eps) return 0;
  return x < 0 ? -1 : 1;
}
struct Point {
  double x, y;
  Point(double x = 0, double y = 0) : x(x), y(y) {}
};
typedef Point Vector;
Vector operator + (Vector A, Vector B) { return Vector(A.x + B.x, A.y + B.y); }
Vector operator - (Vector A, Vector B) { return Vector(A.x - B.x, A.y - B.y); }
Vector operator * (Vector A, double p) { return Vector(A.x * p, A.y * p); }
Vector operator / (Vector A, double p) { return Vector(A.x / p, A.y / p); }
bool operator < (const Point &a, const Point &b) {
  return a.x < b.x || (a.x == b.x && a.y < b.y);
}
bool operator == (const Point &a, const Point &b) {
  return dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) == 0;
}
```

```
double Dot(Vector A, Vector B) { return A.x * B.x + A.y * B.y; }
double Length(Vector A) { return sqrt(Dot(A, A)); }
double Angle(Vector A, Vector B) { return acos(Dot(A, B) / Length(A) / Length(B)); }
double Angle(Vector v) { return atan2(v.y, v.x); }
double Cross(Vector A, Vector B) { return A.x * B.y - A.y * B.x; }
double Area2(Point A, Point B, Point C) { return Cross(B - A, C - A); }
Vector Rotate(Vector A, double rad) {
  return Vector(A.x * cos(rad) - A.y * sin(rad), A.x * sin(rad) + A.y * cos(rad));
}
Vector Normal(Vector A) {// A must have length
  double L = Length(A);
  return Vector(-A.y / L, A.x / L);
}
struct Line {// direct line (half plane in the left)
  Point P;
  Vector v;
  double ang;
  Line(void) {}
  Line(Point P, Vector v) : P(P), v(v) { ang = atan2(v.y, v.x); }
  bool operator < (const Line &L) const {
    return ang < L.ang;
  }
  Point point(double t1) { return P + v * t1; }
};
bool OnLeft(Line L, Point p) { // strict
  return dcmp(Cross(L.v, p - L.P)) > 0;
}
Point GetLineIntersection(Line a, Line b) { // assume int exists
  Vector u = a.P - b.P;
  double t = Cross(b.v, u) / Cross(a.v, b.v);
  return a.P + a.v * t;
}
Point GetLineIntersection(Point P, Vector v, Point Q, Vector w) {
  Vector u = P - Q;
  double t = Cross(w, u) / Cross(v, w);
  return P + v * t;
}
double DistanceToLine(Point P, Point A, Point B) {
  Vector v1 = B - A, v2 = P - A;
  return fabs(Cross(v1, v2)) / Length(v1); // direct dist without fabs
}
double DistanceToSegment(Point P, Point A, Point B) {
  if(A == B) return Length(P - A);
  Vector v1 = B - A, v2 = P - A, v3 = P - B;
  if(dcmp(Dot(v1, v2)) < 0) return Length(v2);
  else if(dcmp(Dot(v1, v3)) > 0) return Length(v3);
  else return fabs(Cross(v1, v2)) / Length(v1);
}
Point GetLineProjection(Point P, Point A, Point B) {
  Vector v = B - A;
  return A + v * (Dot(v, P - A) / Dot(v, v));
}
bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2) {
  double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2 - a1, b2 - a1), c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2 - b1, a2 - b1);
  return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
}
bool OnSegment(Point p, Point a1, Point a2) {
  return dcmp(Cross(a1 - p, a2 - p)) == 0 && dcmp(Dot(a1 - p, a2 - p)) < 0;
}
double PolygonArea(Point *p, int n) { // direct area
  double area = 0;
  for(int i = 1; i < n - 1; ++i) area += Cross(p[i] - p[0], p[i + 1] - p[0]);
  return area / 2;
}
typedef vector<Point> Polygon;
int isPointInPolygon(Point p, Polygon poly) {
  int wn = 0, n = poly.size();
  for(int i = 0; i < n; ++i) {
    if(OnSegment(p, poly[i], poly[(i + 1) % n])) return -1;
    int k = dcmp(Cross(poly[(i + 1) % n] - poly[i], p - poly[i])), d1 = dcmp(poly[i].y - p.y), d2 = dcmp(poly[(i + 1) % n].y - p.y);
    if(k > 0 && d1 <= 0 && d2 > 0) ++wn;
    if(k < 0 && d2 <= 0 && d1 > 0) --wn;
  }
  if(wn != 0) return 1; // inside
  return 0; // outside
}
Polygon CutPolygon(Polygon poly, Point A, Point B) {
  Polygon newpoly;
  int n = poly.size();
  for(int i = 0; i < n; ++i) {
    Point C = poly[i], D = poly[(i + 1) % n];
    if(dcmp(Cross(B - A, C - A)) >= 0) newpoly.push_back(C);
    if(dcmp(Cross(B - A, C - D)) != 0) {
      Point ip = GetLineIntersection(A, B - A, C, D - C);
      if(OnSegment(ip, C, D)) newpoly.push_back(ip);
    }
  }
  return newpoly;
}
struct Circle {
  Point c;
  double r;
  Circle(Point c, double r) : c(c), r(r) {}
  Point point(double a) {
    return Point(c.x + cos(a) * r, c.y + sin(a) * r);
  }
};
int getLineCircleIntersection(Line L, Circle C, double &t1, double &t2, vector<Point> &sol) {
  double a = L.v.x, b = L.P.x - C.c.x, c = L.v.y, d = L.P.y - C.c.y;
  double e = a * a + c * c, f = 2 * (a * b + c * d), g = b * b + d * d - C.r * C.r;
  double delta = f * f - 4 * e * g;// delta
  if(dcmp(delta) < 0) return 0;  // outside
  if(dcmp(delta) == 0) {         // tangent
    t1 = t2 = -f / (2 * e); sol.push_back(L.point(t1));
    return 1;
  }
  // intersect
  t1 = (-f - sqrt(delta)) / (2 * e); sol.push_back(L.point(t1));
  t2 = (-f + sqrt(delta)) / (2 * e); sol.push_back(L.point(t2));
  return 2;
}
int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point> &sol) {
  double d = Length(C1.c - C2.c);
  if(dcmp(d) == 0) {
    if(dcmp(C1.r - C2.r) == 0) return 1;// cover each other
    return 0;
  }
  if(dcmp(C1.r + C2.r - d) < 0) return 0;
  if(dcmp(fabs(C1.r - C2.r) - d) > 0) return 0;

  double a = Angle(C2.c - C1.c);
  double da = acos((C1.r * C1.r + d * d - C2.r * C2.r) / (2 * C1.r * d));// C1C2 to C1P1
  Point p1 = C1.point(a - da), p2 = C1.point(a + da);

  sol.push_back(p1);
  if(p1 == p2) return 1;
  sol.push_back(p2);
  return 2;
}
int getTangents(Point p, Circle C, Vector *v) {
  Vector u = C.c - p;
  double dist = Length(u);
  if(dist < C.r) return 0;
  else if(dcmp(dist - C.r) == 0) {
    v[0] = Rotate(u, PI / 2);
    return 1;
  }
  double ang = asin(C.r / dist);
  v[0] = Rotate(u, -ang);
  v[1] = Rotate(u, ang);
  return 2;
}
int getTangents(Circle A, Circle B, Point *a, Point *b) { // points at A, B
  if(A.r < B.r) { swap(A, B); swap(a, b); }
  double d2 = (A.c.x - B.c.x) * (A.c.x - B.c.x) + (A.c.y - B.c.y) * (A.c.y - B.c.y), rdiff = A.r - B.r, rsum = A.r + B.r;
  if(d2 < rdiff * rsum) return 0;// contain
  double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
  if(d2 == 0 && A.r == B.r) return -1;// infinate
  if(d2 == rdiff * rsum) { // inner tangent
     a[0] = A.point(base), b[0] = B.point(base);
     return 1;
  } // outer tangent
  int cnt = 0;
  double ang = acos((A.r - B.r) / sqrt(d2));
  a[cnt] = A.point(base + ang), b[cnt] = B.point(base + ang); ++cnt;
  a[cnt] = A.point(base - ang), b[cnt] = B.point(base - ang); ++cnt;
  if(d2 == rsum * rsum) a[cnt] = A.point(base), b[cnt] = B.point(PI + base), ++cnt;
  else if(d2 > rsum * rsum) {
    ang = acos((A.r + B.r) / sqrt(d2));
    a[cnt] = A.point(base + ang), b[cnt] = B.point(base + ang); ++cnt;
    a[cnt] = A.point(base - ang), b[cnt] = B.point(base - ang); ++cnt;
  }
  return cnt;
}
Circle CircumscribedCircle(Point p1, Point p2, Point p3) {
  double Bx = p2.x - p1.x, By = p2.y - p1.y, Cx = p3.x - p2.x, Cy = p3.y - p2.y;
  double D = 2 * (Bx * Cy - By * Cx);
  Point p = Point((Cy * (Bx * Bx + By * By) - By * (Cx * Cx + Cy * Cy)) / D + p1.x, (Bx * (Cx * Cx + Cy * Cy) - Cx * (Bx * Bx + By * By)) / D + p1.y);
  return Circle(p, Length(p1 - p));
}
Circle InscribedCircle(Point p1, Point p2, Point p3) {
  double a = Length(p2 - p3), b = Length(p3 - p1), c = Length(p1 - p2);
  Point p = (p1 * a + p2 * b + p3 * c) / (a + b + c);
  return Circle(p, DistanceToLine(p, p1, p2));
}
Circle min_cover_circle(int n, Point *p) {
  random_shuffle(p,p+n);
  Circle ret = Circle(p[0], 0);
  for(int i = 1; i < n; ++i) { // first
    if(dist(ret.c, p[i]) > r + eps) {
      ret = Circle(p[i], 0);
      for(int j = 0; j < i; ++j) // second
        if(dist(ret.c, p[j]) > r + eps) {
          ret.c = (ret.c + p[j]) / 2;
          ret.r = dis(ret.c, p[j]);
          for(int k = 0; k < j; ++k) // third
            if(dist(ret.c, p[k]) > r + eps)
              ret = circumcenter(p[i], p[j], p[k]);
        }
    }
  }
}
```

6.5 2D Geometry Inversion

```
double add(double a, double b) {
  return abs(a + b) < eps * (abs(a) + abs(b)) ? 0. : a + b;
}
struct Circle;
struct Point {
  double x, y;
  Point(double x = 0., double y = 0.): x(x), y(y) {}
  Point operator +(const Point &P) const {
    return Point(add(x, P.x), add(y, P.y));
  }
  Point operator -(const Point &P) const {
    return Point(add(x, -P.x), add(y, -P.y));
  }
  double operator *(const Point &P) const {
    return add(x * P.x, y * P.y);
  }
  double operator %(const Point &P) const {
    return add(x * P.y, -y * P.x);
  }
  Point operator *(double d) const {
    return Point(x * d, y * d);
  }
  Point operator /(double d) const {
    return Point(x / d, y / d);
  }
  bool operator <(const Point &P) const {
    return x == P.x? y < P.y : x < P.x;
  }
  Point &operator +=(const Point &P) {
     return *this = *this + P;
  }
  Point &operator -=(const Point &P) {
    return *this = *this - P;
  }
  double angle() const {
    return atan2(y, x);
  }
  double mod2() const {
    return *this * *this;
  }
  double mod() const {
    return sqrt(mod2());
  }
  Point norm() const { // 单位法向量
    return Point(-y, x) / mod();
  }
  Point rott(double a) const  { // 逆时针旋转
    return Point(add(x * cos(a), -y * sin(a)), add(x * sin(a), y * cos(a)));
  }
  Point inv(const Circle &O); // 点反演到点
};
struct Line {
  Point o, v;
  Line() {}
  Line(const Point &o, const Point &v): o(o), v(v) {}
  bool have(const Point &P) {
    return !dcmp((P - o) % v);
  }
  Circle inv2c(const Circle &O); // 直线反演到圆
};
Point intersect(const Line &l, const Line &r) {
    return l.o + l.v * (r.v % (l.o - r.o)) / (l.v % r.v);
}
Point outer_center(const Point &a, const Point &b, const Point &c) {
  Line l = Line((a + b) * .5, (b - a).norm());
  Line r = Line((a + c) * .5, (c - a).norm());
  return intersect(l, r);
}
struct Circle {
  Point o;
  double r;
  Circle() {}
  Circle(const Point o, double r): o(o), r(r) {}
  Point at(double a) {
    return Point(o.x + r * cos(a), o.y + r * sin(a));
  }
  Circle inv2c(const Circle &O) { // 圆反演到圆
    double d2 = (o - O.o).mod2();
    double r2 = sqr(O.r) * r / (d2 - sqr(r));
    double d = sqr(O.r) / (sqrt(d2) + r) + r2;
    return Circle(O.o + (o - O.o) / sqrt(d2) * d, r2);
  }
  Line inv2l(const Circle &O) { // 圆反演到直线
    return Line(o * 2 - O.o, (o - O.o).norm());
  }
};
Point Point::inv(const Circle &O) {
  Point OP = (*this - O.o);
  return O.o + OP * (sqr(O.r) / OP.mod2());
}
Circle Line::inv2c(const Circle &O) {
  Point n = v.norm();
  if (dcmp(n * (O.o - o)) > 0) n = n * -1.;
  double r = sqr(O.r) / abs(v % (O.o - o)) * v.mod() / 2;
  return Circle(O.o + n * r, r);
}
int comm_tan(Circle A, Circle B, Point *a, Point *b) { // 两圆公切线
  int cnt = 0;
  if (A.r < B.r) swap(A, B), swap(a, b);
  double dist = (A.o - B.o).mod(), dr = A.r - B.r, sr = A.r + B.r;
  if (dcmp(dist) == 0 && dcmp(dr) == 0) return -1; // 重合
  if (dcmp(dist - dr) < 0) return 0; // 内含
  double base = (B.o - A.o).angle(); // 内切
  if (dcmp(dist - dr) == 0) {
    a[0] = A.at(base);
    b[0] = B.at(base);
    return 1;
  }
  double ang = acos(dr / dist); // 两条外公切线
  a[cnt] = A.at(base + ang), b[cnt] = B.at(base + ang), ++cnt;
  a[cnt] = A.at(base - ang), b[cnt] = B.at(base - ang), ++cnt;
  if (dcmp(dist - sr) == 0) { // 外切，中间一条内公切线
    a[cnt] = A.at(base), b[cnt] = B.at(pi + base), ++cnt;
  } else {
    ang = acos(sr / dist); // 相离，两条内公切线
    a[cnt] = A.at(base + ang), b[cnt] = B.at(pi + base + ang), ++cnt;
    a[cnt] = A.at(base - ang), b[cnt] = B.at(pi + base - ang), ++cnt;
  }
  return cnt;
}
```

6.6 3D Convex Hull

```
double mix(const Point &a, const Point &b, const Point &c) {
    return a % (b ^ c);
}
const int N = 305;
int mark[N][N];
Point info[N];
int n , cnt;
double area(int a, int b, int c) {
    return ((info[b] - info[a]) ^ (info[c] - info[a])).len();
}
double volume(int a, int b, int c, int d) {
    return mix(info[b] - info[a], info[c] - info[a], info[d] - info[a]);
}
struct Face {
    int v[3];
    Face() {}
    Face(int a, int b, int c) {
        v[0] = a , v[1] = b , v[2] = c;
    }
    int& operator [] (int k) {
        return v[k];
    }
};
vector <Face> face;
inline void insert(int a, int b, int c) {
    face.push_back(Face(a, b, c));
}
void add(int v) {
    vector <Face> tmp;
    int a, b, c;
    cnt ++;
    for (int i = 0; i < face.size() ; ++ i) {
        a = face[i][0] , b = face[i][1] , c = face[i][2];
        if (dcmp(volume(v, a, b, c)) < 0)
            mark[a][b] = mark[b][a] = mark[b][c] = mark[c][b] = mark[c][a] = mark[a][c] = cnt;
        else
            tmp.push_back(face[i]);
    }
    face = tmp;
    for (int i = 0; i < tmp.size() ; ++ i) {
        a = face[i][0] , b = face[i][1] , c = face[i][2];
        if (mark[a][b] == cnt) insert(b, a, v);
        if (mark[b][c] == cnt) insert(c, b, v);
        if (mark[c][a] == cnt) insert(a, c, v);
    }
}
int Find() {
    for (int i = 2; i < n; ++ i) {
        Point ndir = (info[0] - info[i]) ^ (info[1] - info[i]);
        if (ndir == Point())
            continue;
        swap(info[i], info[2]);
        for (int j = i + 1; j < n; j++)
            if (dcmp(volume(0, 1, 2, j)) != 0) {
                swap(info[j], info[3]);
                insert(0, 1, 2);
                insert(0, 2, 1);
                return 1;
            }
    }
    return 0;
}
void work() {
    for (int i = 0; i < n; ++ i)
        info[i].input();
    sort(info, info + n);
    n = unique(info, info + n) - info;
    face.clear();
    random_shuffle(info, info + n);
    if (Find()) {
        memset(mark, 0, sizeof(mark));
        cnt = 0;
        for (int i = 3; i < n; ++ i) add(i);
        vector<Point> Ndir;
        for (int i = 0; i < face.size() ; ++ i) {
            Point p = (info[face[i][0]] - info[face[i][1]]) ^ (info[face[i][2]] - info[face[i][1]]);
            p = p / p.len();
            Ndir.push_back(p);
        }
        sort(Ndir.begin(), Ndir.end());
        int ans = unique(Ndir.begin(), Ndir.end()) - Ndir.begin();
        printf("%d\n", ans);
    } else {
        printf("1\n");
    }
}
```

7 Miscellaneous

7.1 C++ fast I/O

```
namespace fastIO{
  #define BUF_SIZE 100000
  #define OUT_SIZE 100000
  #define ll long long
  //fread->read
  bool IOerror=0;
  inline char nc(){
    static char buf[BUF_SIZE],*p1=buf+BUF_SIZE,*pend=buf+BUF_SIZE;
    if (p1==pend){
      p1=buf; pend=buf+fread(buf,1,BUF_SIZE,stdin);
      if (pend==p1){IOerror=1;return -1;}
    }
    return *p1++;
  }
  inline bool blank(char ch){return ch==' '||ch=='\n'||ch=='\r'||ch=='\t';}
  inline void read(int &x){
    bool sign=0; char ch=nc(); x=0;
    for (;blank(ch);ch=nc());
    if (IOerror)return;
    if (ch=='-')sign=1,ch=nc();
    for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
    if (sign)x=-x;
  }
  //fwrite->write
  struct Ostream_fwrite{
    char *buf,*p1,*pend;
    Ostream_fwrite(){buf=new char[BUF_SIZE];p1=buf;pend=buf+BUF_SIZE;}
    void out(char ch){
      if (p1==pend){
        fwrite(buf,1,BUF_SIZE,stdout);p1=buf;
      }
      *p1++=ch;
    }
    void print(int x){
      static char s[15],*s1;s1=s;
      if (!x)*s1++='0';if (x<0)out('-'),x=-x;
      while(x)*s1++=x%10+'0',x/=10;
      while(s1--!=s)out(*s1);
    }
    void print(char *s){while (*s)out(*s++);}
    void flush(){if (p1!=buf){fwrite(buf,1,p1-buf,stdout);p1=buf;}}
    ~Ostream_fwrite(){flush();}
  }Ostream;
  inline void print(int x){Ostream.print(x);}
  inline void print(char *s){Ostream.print(s);}
  inline void flush(){Ostream.flush();}
  #undef ll
  #undef OUT_SIZE
  #undef BUF_SIZE
};
using namespace fastIO;
```

7.2 bitset

```
int n, m, bits[65537], brev[16][65537];
ULL f[maxs], g[maxs];
char buf[maxn];
inline ULL mask(int len) {
  return (len < 64 ? 1ULL << len : 0ULL) - 1;
}
inline ULL getbits(ULL f[], int pL, int pR) {
  return (pL >> 6) == (pR >> 6) ? (f[pL >> 6] >> (pL & 63)) & mask(pR - pL + 1) :
  (f[pR >> 6] & mask((pR & 63) + 1)) << (((pR >> 6) << 6) - pL) | (f[pL >> 6] >> (pL & 63));
}
inline void putbits(ULL f[], int pL, int pR, ULL msk) {
  if((pL >> 6) == (pR >> 6))
    f[pL >> 6] ^= (((f[pL >> 6] >> (pL & 63)) & mask(pR - pL + 1)) ^ msk) << (pL & 63);
  else {
    putbits(f, pL, ((pR >> 6) << 6) - 1, msk & mask(((pR >> 6) << 6) - pL));
    putbits(f, (pR >> 6) << 6, pR, msk >> (((pR >> 6) << 6) - pL));
  }
}
inline int countbits(ULL msk) {
  int ret = 0;
  for( ; msk; msk >>= 16) ret += bits[msk & 65535];
  return ret;
}
inline ULL reversebits(ULL msk, int len) {
  ULL ret = 0;
  for( ; len > 0; len -= 16, msk >>= 16) {
    int blen = len < 16 ? len : 16;
    ret = ret << blen | brev[blen - 1][msk & ((1 << blen) - 1)];
  }
  return ret;
}
void solve() {
  for(int i = 1; i < 65536; ++i)
    bits[i] = bits[i >> 1] + (i & 1);
  for(int i = 0; i < 16; ++i)
    for(int j = 0; j < 1 << (i + 1); ++j)
      brev[i][j] = (brev[i][j >> 1] >> 1) | ((j & 1) << i);
  // segment assign [L, R] buf[0]
  for(int i = L >> 6; i <= (R >> 6); ++i) {
    int pL = i == (L >> 6) ? (L & 63) : 0;
    int pR = i == (R >> 6) ? (R & 63) : 63;
    ULL msk = mask(pR - pL + 1) << pL;
    buf[0] == 'b' ? f[i] |= msk : f[i] &= ~msk;
  }
  // segment reverse [L, R]
  for(int i = R; i >= L; i -= 64) {
    int j = i - 63 >= L ? i - 63 : L;
    putbits(g, R - i, R - j, reversebits(getbits(f, j, i), i - j + 1));
  }
  for(int i = L; i <= R; i += 64) {
    int j = i + 63 <= R ? i + 63 : R;
    putbits(f, i, j, getbits(g, i - L, j - L));
  }
}
```

7.3 Notes

Zeller Formula

令 $c=\left\lfloor\frac{year}{100}\right\rfloor, y=year \bmod 100, m=month, d=day$ ，如果 $m\le 2$ ，则视为上一年末尾月份，则有 $w\equiv\left\lfloor\frac{c}{4}\right\rfloor-2c+y+\left\lfloor\frac{y}{4}\right\rfloor+\left\lfloor\frac{13(m+1)}{5}\right\rfloor+d-1 \pmod 7$ ，在1582年10月4日或之前为 $w\equiv y+\left\lfloor\frac{y}{4}\right\rfloor+\left\lfloor\frac{c}{4}\right\rfloor-2c+\left\lfloor\frac{13(m+1)}{5}\right\rfloor+d+2 \pmod 7$ 。

Suspension

悬线法实际上是枚举了下边界具体值，枚举了上边界的取值范围，找到对应的左右边界的取值范围。

Mo's Algo & Sqrt Decomposition

莫队算法和根号分解可以避免集合合并，只进行元素添删，从而降低复杂度。莫队先扩张后收缩防止无效访问。根号分解使得相邻变化点之间变化量可控。树上莫队可参考欧拉序进行分块，树块剖分对大度数点效果不好。

Matrix-Tree & BEST Theorem

Matrix-Tree: 对于无向图，生成树个数等于度数矩阵减去邻接矩阵的任意 $|V|-1$ 阶主子式；对于有向图，以 u 为根的树形图个数等于度数矩阵减去邻接矩阵移除 u 所在行列的主子式。

BEST: 有向图中欧拉图个数为任选一点为根的树形图个数乘以所有点的出度减一的阶乘。

Minimum Product Spanning Tree

对于向量 (x,y) 求解对应投影的最小生成树，如能划分两个平面则继续求解。

Minimum Manhattan Spanning Tree

每个点只会分别向八个区域最近点连边，用扫描线维护。

Hall Theorem & Bipartite

如果 X 中的任意 k 个点至少与 Y 中的 k 个点相邻，则存在完美匹配。二分图最小路径覆盖 = 最大独立集 = 总节点数 - 最大匹配数，最小点覆盖 = 最大匹配数。任意图中，最大独立集 + 最小点覆盖集 $=|V|$ ，最大团 = 补图的最大独立集。

Net Flow Notes

无源汇上下界可行流：建立源汇平衡下界流量；有源汇上下界可行流：添加原汇到原源的边；有源汇上下界最大流：先跑可行流，然后去掉新边做最大流；有源汇上下界最小流：之后做汇到源的最大流。

线性规划转费用流：添加松弛变量，将不等号都变为等号。分别用下一个式子减去上一个式子，如果每个变量只出现了两次且符号一正一负，可转费用流。对于每个式子建立一个点，那么每个变量对应一条边，从一个点流出，向另一个点流入。这样对于等式右边的常数 C ，如果是正的，对应从源点向该点连一条流量 C ，费用 0 的边；如果是负的对应从该点向汇点连一条流量 $-C$ ，费用 0 的边。对于每个变量，从它系数为正的式子向系数为负的式子连一条容量 INF ，费用为它在目标函数里系数的边。

Divisor Function and Similars

定义 $\omega(n)=\sum_{p|n}1, \Omega(n)=\sum_{p^k|n}1, \tau(n)=\sum_{d|n}1$ ，显然有 $2^{\omega(n)}\le 2^{\Omega(n)}\le\tau(n)$ 和 $\max_{x\le n}\Omega(x)=O(\log_2 n)$ ，此外根据有限个点的修正有 $\tau(n)=O(n^{\frac{1.066}{\ln\ln n}})$ 。

Prime Count Function

定义 $S(n,k)$ 表示前 n 个正整数里不被前 k 个质数整除的数字个数， $C(n)$ 表示不超过 n 的素数个数，第 n 个质数是 $p(n)$ ，则 $S(n,k)=S(n,k-1)-S(\left\lfloor\frac{n}{p(k)}\right\rfloor,k-1)$。如果 $C(\lfloor\sqrt{n}\rfloor)\le k$ ，说明 $S(n,k)$ 代表的便是 1 与不超过 n 的素数中不属于前 k 个素数的数字个数，当 $n\le T$ 时，可以用 $C(n)$ 和 k 的关系直接得到答案。还可以剪枝 k 较小的情况，例如 $S(n,0)=n, S(n,1)=n-\left\lfloor\frac{n}{2}\right\rfloor$ 。

Extended Baby Step Giant Step

当底数与模数不互质时，取出底数与模数相消，否则视情况分块查找。

Linear Recurrence Relations

已知 $a(n)=\sum_{i=1}^{k}c(i)\,a(n-i)$ ，令 $a(n)=\sum_{i=0}^{k-1}f_n(i)\,a(i), F_n(z)=\sum_{i=0}^{k-1}f_n(i)\,z^i$ ，则 $F_n(z)$ 定义在在模 $z^k-\sum_{i=0}^{k-1}c(k-i)\,z^i$ 意义下。

Lagrange Interpolation

对于 $f(x)=\sum_{i=0}^{deg}a_i\,x^i$ ，有 $f(n)=\sum_{i=0}^{deg}(-1)^{deg-i}f(i)\frac{\prod_{i\neq j}(n-j)}{(deg-i)!i!}$

Newton Iteration & Lagrange Inversion Th

Newton: 已知 $G(z)$ ，求 $z=F(x)$ 使得 $G(F(x))\equiv 0 \pmod{x^n}$ 。设 $F_k(x)$ 满足 $G(F_k(x))\equiv 0 \pmod{x^{2^k}}$ ，则有 $F_{k+1}(x)\equiv F_k(x)-\frac{G(F_k(x))}{G'(F_k(x))} \pmod{x^{2^{k+1}}}$ 。

Lagrange: 已知 $F(x),G(x)$ 满足 $G(F(x))=x$ ，则有 $[x^n]F(x)=\frac{1}{n}[x^{n-1}](\frac{x}{G(x)})^n$ 。

Subset Convolution

正变换为 $F(n)=\sum_{i\subseteq n}\sum_m f_m(i)\cdot z^m+\sum_{k=|n|+1}^{\infty}\epsilon(k)\cdot z^k$ 。

Stirling & Bernoulli & Powers

$\left[{n \atop m}\right]$ 表示 n 个不同元素构成 m 个圆排列的数目，有 $\left[{n+1 \atop m}\right]=\left[{n \atop m-1}\right]+n\left[{n \atop m}\right]$ ； $\left\{{n \atop m}\right\}$ 表示 n 个不同元素构成 m 个集合的数目，有 $\left\{{n+1 \atop m}\right\}=\left\{{n \atop m-1}\right\}+m\left\{{n \atop m}\right\}$ ；伯努利数满足 $\sum_{k=0}^{n}\binom{n+1}{k}B_k=0$ ，以及 $\sum_{k=1}^{n}k^m=\frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k n^{m+1-k}$ 。

Catalan & Young's Tableau

卡特兰数表示 n 对括号序列数量，有 $C(n)=C(n-1)\frac{4n-2}{n+1}=\frac{1}{n+1}\binom{2n}{n}=\binom{2n}{n}-\binom{2n}{n-1}$ ，从 $(0,0)$ 走到 (n,m) 不超越对角线的方案数为 $\frac{n-m+1}{n+1}\binom{n+m}{m}$ ；一种形态的杨氏矩阵数量等于格数阶乘除以每个格子向右向下能覆盖的格子数。

Game Theorem

威佐夫博弈 $(n,\left\lfloor\frac{1+\sqrt{5}}{2}n\right\rfloor)$ 有解。

Tree Theorem

带标号 n 点 m 棵有根树且树根给定，方案数为 mn^{n-m-1} 。

对于 n 点 m 连通块的图，设每个连通块有 a_i 个点，那么用 $s-1$ 条边把它连通的方案数为 $n^{s-2}a_1a_2\cdots a_m$ 。