Reference Document & Code Library

ACM ICPC Team Notebook of B-)

Rezwan Arefin

December 18, 2018

Contents

1 Data Structure

1.1 Bridge Tree

```
vector<int> adj[N], tree[N]; // tree = Bridge Tree
int vis[N], low[N], comp[N], bicon[N], tym = 1;

void calc(int u, int par, int c) {
  comp[u] = c;
  vis[u] = low[u] = tym++;
  for(int v : adj[u]) {
    if(vis[v]) {
      if(v - par) low[u] = min(low[u], vis[v]);
      else par = -1; // Handles multiple edges.
    } else {
      calc(v, u, c);
      low[u] = min(low[u], low[v]);
    }
  }
}
void shrink(int u, int now) {
  bicon[u] = now;
  for(int v : adj[u]) if(!bicon[v]) {
    if(low[v] > vis[u]) {
      tree[now].push_back(c);
      shrink(v, c++);
    } else shrink(v, now);
  }
}
```

1.2 Centroid Decompostion

```
// Impl. of count number of K-len paths in a tree
const int N = 1e5 + 10;
int n, k, vis[N], sub[N];
vector<int> adj[N];

void calc(int u, int par) { sub[u] = 1;
  for(int v : adj[u]) if(!vis[v] && v - par)
    calc(v, u), sub[u] += sub[v];
}
int centroid(int u, int par, int r) {
  for(int v : adj[u]) if(!vis[v] && v - par)
    if(sub[v] > r) return centroid(v, u, r);
  return u;
}
int dist[N]; ll ans;
int in[N], out[N], vert[N], tym = 0;

void dfs(int u, int par = -1, int d = 0) {
  dist[u] = d;
  in[u] = tym;
  vert[tym++] = u;
  for(int v : adj[u]) if(v - par && !vis[v])
    dfs(v, u, d + 1);
  out[u] = tym - 1;
}
void solve(int u) {
  tym = 0; dfs(u);
  unordered_map<int, int> cnt; cnt[0] = 1;
  for(int v : adj[u]) if(!vis[v]) {
    for(int t = in[v]; t <= out[v]; ++t)
      if(dist[vert[t]] <= k)
```

```
      ans += cnt[k - dist[vert[t]]];
    for(int t = in[v]; t <= out[v]; ++t)
      ++cnt[dist[vert[t]]];
  }
}
void decomp(int u, int par = -1) {
  calc(u, par);
  int c = centroid(u, par, sub[u] / 2);
  solve(c); vis[c] = 1;
  for(int v : adj[c]) if(!vis[v]) decomp(v, c);
}
```

1.3 Centroid Tree

```
// p[u] = parent of u in centroid tree
// u in subterr of centroid c
//   => d[lvl[c]][u] = dist(c, u)
vector<int> adj[N];
int lvl[N], sub[N], p[N], del[N], d[18][N], ans[N];

void calc(int u, int par) { sub[u] = 1;
  for(int v : adj[u]) if(v - par && !del[v])
    calc(v, u), sub[u] += sub[v];
}
int centroid(int u, int par, int r) {
  for(int v : adj[u]) if(v - par && !del[v])
    if(sub[v] > r) return centroid(v, u, r);
  return u;
}
void dfs(int l, int u, int par) {
  if(par + 1) d[l][u] = d[l][par] + 1;
  for(int v : adj[u]) if(v - par && !del[v])
    dfs(l, v, u);
}
void decompose(int u, int par) {
  calc(u, -1);
  int c = centroid(u, -1, sub[u] >> 1);
  del[c] = 1, p[c] = par;
  if(par + 1) lvl[c] = lvl[par] + 1;
  dfs(lvl[c], c, -1);
  for(int v : adj[c]) if(v - par && !del[v])
    decompose(v, c);
}
void update(int u) {
  for(int v = u; v + 1; v = p[v])
    ans[v] = min(ans[v], d[lvl[v]][u]);
}
int query(int u) {
  int ret = 1e9;
  for(int v = u; v + 1; v = p[v])
    ret = min(ret, ans[v] + d[lvl[v]][u]);
  return ret;
}
```

1.4 Convex Hull Trick (Offline)

```
// m[] decreasing:
//   minimum => bad(s-3, s-2, s-1), x[] increasing
//   maximum => bad(s-1, s-2, s-3), x[] decreasing
// If m[] is increasing:
//   maximum => bad(s-3, s-2, s-1), x[] increasing
//   minimum => bad(s-1, s-2, s-3), x[] decreasing
// x[] isn't monotone: Ternary Search

struct CHT {
  vector<ll> m, b; int ptr = 0;
  bool bad(int l1, int l2, int l3) {
    // returns intersect(l1, l3) <= intersect(l1, l2)
    return 1.0 * (b[l3] - b[l1]) * (m[l1] - m[l2]) <=
           1.0 * (b[l2] - b[l1]) * (m[l1] - m[l3]);
  }
  void add(ll _m, ll _b) {
    m.push_back(_m); b.push_back(_b);
    int s = m.size();
    while(s >= 3 && bad(s-3, s-2, s-1)) {
      s--;
      m.erase(m.end()-2);
      b.erase(b.end()-2);
    }
  }
  ll f(int i, ll x) { return m[i]*x + b[i]; }
  ll query(ll x) {
    if(ptr >= m.size()) ptr = m.size()-1;
    while(ptr < m.size()-1 &&
      f(ptr+1, x) < f(ptr, x)) ptr++;
    return f(ptr, x);
  }
};
```

1.5 Convex Hull Trick (Dynamic)

```
// Keeps upper hull for maximums.
// For Minimum: add -m, -b. Return -ans.

const ll is_query = -(1LL<<62);
struct Line {
  ll m, b;
  mutable function<const Line*()> succ;
  bool operator<(const Line& rhs) const {
    if(rhs.b != is_query) return m < rhs.m;
    const Line* s = succ();
    if(!s) return 0;
    ll x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};
struct HullDynamic : public multiset<Line> {
  bool bad(iterator y) {
    auto z = next(y);
    if(y == begin()) {
      if (z == end()) return 0;
      return y->m == z->m && y->b <= z->b;
    }
    auto x = prev(y);
    if(z == end()) return y->m == x->m && y->b <=
        x->b;
    return 1.0 * (x->b - y->b)*(z->m - y->m) >=
           1.0 * (y->b - z->b)*(y->m - x->m);
  }
  void insert_line(ll m, ll b) {
    auto y = insert({ m, b });
    y->succ = [=] { return next(y) == end() ? 0 :
        &*next(y); };
    if(bad(y)) { erase(y); return; }
    while(next(y) != end() && bad(next(y)))
        erase(next(y));
    while(y != begin() && bad(prev(y)))
        erase(prev(y));
  }
  ll eval(ll x) {
    auto l = *lower_bound((Line) { x, is_query });
    return l.m * x + l.b;
  }
};
```

1.6 Dominator Tree

```
// adj[], par[] for main DAG. par[] = reversed edges
// p[][], L[], dom[] for dominator tree

vector<int> adj[N], par[N], topo;
int dom[N], L[N], p[N][22], vis[N];

void addEdge(int u, int v) { // Build DAG like this
  adj[u].push_back(v);
  par[v].push_back(u);
}
void dfs(int u) { // topo sort
  if(vis[u]) return; vis[u] = 1;
  for(int v : adj[u]) if(!vis[v]) dfs(v);
  topo.push_back(u);
}
void addToTree(int par, int u) { // to Dominator Tree
  dom[u] = par; p[u][0] = par;
  L[u] = L[par] + 1;
  for(int i = 1; i <= 20; ++i)
    if(p[u][i - 1] + 1)
      p[u][i] = p[p[u][i-1]][i-1];
}
int lca(int u, int v) { /* LCA routine here */ }
void DominatorTree(int root) {
  dfs(root);
  memset(p, -1, sizeof p);
  memset(dom, -1, sizeof dom);
  for(int i = (int) topo.size() - 2; i >= 0; --i) {
    int u = topo[i], d = -1;
    for(int v : par[u]) d = d == -1 ? v : lca(v, d);
    addToTree(d, u);
  }
}
```

1.7 Heavy Light Decomposition

```
// head[u] = head node of u's chain
// pos[u] = position of u in seg tree

struct SegmentTree { ... } tree;

vector<int> adj[N];
int n, p[N], heavy[N], dep[N], head[N], pos[N];
int dfs(int u, int par) {
  if(par + 1) dep[u] = dep[par] + 1;
  int size = 1, Max = 0; p[u] = par;
  for(int v : adj[u]) if(v - par) {
    int sub = dfs(v, u);
    if(sub > Max) Max = sub, heavy[u] = v;
    size += sub;
  } return size;
}

template <class BinaryOperation>
void processPath(int u, int v, BinaryOperation op) {
  for(; head[u] != head[v]; u = p[head[u]]) {
    if(dep[head[v]] > dep[head[u]]) swap(u, v);
    op(pos[head[u]], pos[u]);
  } if(dep[v] > dep[u]) swap(u, v);
  op(pos[v] + weight_on_edge, pos[u]);
}
int updatePath(int u, int v, int val) {
  processPath(u, v, [&val](int l, int r) {
    tree.update(l, r, v);
  });
}
int queryPath(int u, int v) {
  int ret = 0;
  processPath(u, v, [&ret](int l, int r) {
    ret += tree.query(l, r);
  }); return ret;
}
void init() {
  fill_n(heavy, n, -1); dfs(0, -1);
  for(int i = 0, idx = 0; i < n; i++) {
    if(p[i] == -1 || heavy[p[i]] != i) {
      for(int j = i; j + 1; j = heavy[j])
        head[j] = i, pos[j] = idx++;
    }
  } tree.init(n);
  for(int i = 0; i < n; i++)
    tree.set(pos[i], value[i]);
}
```

1.8 Interval Container

```
// Intervals are [inclusive, exclusive).
template <class T>
auto addInterval(set<pair<T, T>>& is, T L, T R) {
  if(L == R) return is.end();
  auto it = is.lower_bound({L, R}), before = it;
  while(it != is.end() && it->first <= R) {
    R = max(R, it->second);
    before = it = is.erase(it);
  }
  if(it != is.begin() && (--it)->second >= L) {
    L = min(L, it->first);
    R = max(R, it->second);
    is.erase(it);
  }
  return is.insert(before, {L,R});
};
template <class T>
void removeInterval(set<pair<T, T>>& is, T L, T R) {
  if(L == R) return;
  auto it = addInterval(is, L, R);
  T r2 = it->second;
  if(it->first == L) is.erase(it);
  else(T&)it->second = L;
  if(R != r2) is.emplace(R, r2);
};
```

1.9 KD Tree

```
struct Node {
  Point p, v;
  Node *l, *r;
};
Node* build(int N, Point *sX, Point *sY, int h) {
  Node *root = allocateNode();
  if(N == 1) {
    root->p = sX[0];
  } else {
    Point * sx = (h & 1) ? sX : sY;
    Point * sy = (h & 1) ? sY : sX;
    bool (*cmp)(const Point &, const Point &);
    cmp = (h & 1) ? cmpX : cmpY;
    root->v = sx[N / 2];
    int K = 0;
    REP(i, 0, N) {
      if(!cmp(sy[i], root->v)) {
        tmp[K++] = sy[i];
      } else {
        sy[i - K] = sy[i];
      }
```

```
    }
    REP(i, 0, K) sy[N - K + i] = tmp[i];
    root->l = build(N / 2, sX, sY, h + 1);
    root->r = build(K, sX + N/2, sY + N/2, h + 1);
  }
  return root;
}
void findClosest(Node* root, const Point &p, ll
    &minDist, int h) {
  if(root->l == NULL) {
    if(! (root->p == p) )
      minDist = min(minDist, (p - root->p).len());
    return;
  }
  bool (*cmp)(const Point &, const Point &);
  cmp = (h & 1) ? cmpX : cmpY;
  ll d = (h & 1)?(root->v.x - p.x):(root->v.y - p.y);
  if(cmp(p, root->v)) {
    findClosest(root->l, p, minDist, h + 1);
  } else {
    findClosest(root->r, p, minDist, h + 1);
  }
  if(sqr(d) < minDist) {
    if(cmp(p, root->v)) {
      findClosest(root->r, p, minDist, h + 1);
    } else {
      findClosest(root->l, p, minDist, h + 1);
    }
  }
}
```

1.10 MO's Algo with Updates

```
const int N = 1e5 + 10;
int n, m, a[N], prv[N], ans[N], block;

struct query {
  int l, r, id, t, blcl, blcr;
  query(int _a, int _b, int _c, int _d) {
    l = _a, r = _b; id = _c, t = _d;
    blcl = l / block; blcr = r / block;
  }
  bool operator < (const query &p) const {
    if(blcl != p.blcl) return l < p.l;
    if(blcr != p.blcr) return r < p.r;
    return t < p.t;
  }
}; vector<query> q;

struct update { int pos, pre, now; };
vector<update> u;

int l, r, t; // for mo's left/right/time pointer.
int cnt[N * 2];

void add(int x) { } // Add a[x] to ds
void remove(int x) { } // Remove a[x] from ds
void apply(int i, int x) { // Change a[i] to x
  if(l <= i && i <= r) {
    remove(i); a[i] = x; add(i);
  } else a[i] = x;
}

int main(int argc, char const *argv[]) {
  read(n); read(m);
  block = pow(n, 0.6667);
  for(int i = 0; i < n; ++i)
    read(a[i]), prv[i] = a[i];

  u.push_back({-1, -1, -1});

  for(int i = 0; i < m; ++i) {
    int t, l, r;
    read(t); read(l); read(r);
    if(t == 1) { --r;
      q.push_back({l, r, q.size(), u.size() - 1});
    } else {
      u.push_back({l, prv[l], r});
      prv[l] = r;
    }
  }
  sort(q.begin(), q.end());
  l = 0, r = -1, t = 0;

  for(int i = 0; i < q.size(); i++) {
    while(t < q[i].t) ++t,apply(u[t].pos, u[t].now);
    while(t > q[i].t) apply(u[t].pos, u[t].pre),--t;
    while(r < q[i].r) add(++r);
    while(l > q[i].l) add(--l);
    while(r > q[i].r) remove(r--);
    while(l < q[i].l) remove(l++);
    ans[q[i].id] = ds.get();
  }
  for(int i = 0; i < q.size(); i++)
    printf("%d\n", ans[i]);
}
```

1.11 Persistent Segment Tree

```
// Update version x from vecrsion y:
// root[x] = root[y];
// update(root[x], ...)

struct node { int l, r, sum; } t[N * 20];
int root[N], a[N], n, m, k, idx, M;

void update(int &nd, int l, int r, int &i) {
  t[++idx] = t[nd];
  ++t[nd = idx].sum; // += v;
  if(l == r) return;
  int m = l + r >> 1;
  if(i <= m) update(t[nd].l, l, m, i);
  else update(t[nd].r, m + 1, r, i);
}
// a = root[r], b = root[l - 1]
int lesscnt(int a, int b, int l, int r, int k) {
  if(r <= k) return t[a].sum - t[b].sum;
  int m = l + r >> 1;
  if(k <= m) return lesscnt(t[a].l, t[b].l, l, m, k);
  else return lesscnt(t[a].l, t[b].l, l, m, k) +
       lesscnt(t[a].r, t[b].r, m + 1, r, k);
}
int kthnum(int a, int b, int l, int r, int k) {
  if(l == r) return l;
  int cnt = t[t[a].l].sum - t[t[b].l].sum;
  int m = l + r >> 1;
  if(cnt >= k) return kthnum(t[a].l, t[b].l, l, m, k);
  else return kthnum(t[a].r, t[b].r, m+1, r, k-cnt);
}
void init() {
  t[0].l = t[0].r = t[0].sum = 0;
  for(int i = 1; i <= n; ++i)
    update(root[i] = root[i - 1], 0, M, a[i]);
}
```

1.12 Persistent Trie

```
struct node {
  node *ch[2];
  node() { ch[0] = ch[1] = NULL; }
  node *clone() {
    node *ret = new node();
    if(this) {
      ret -> ch[0] = ch[0];
      ret -> ch[1] = ch[1];
    } return ret;
  }
} *trie[N];
void insert(int v, int p, int val) {
  node *curr = trie[v] = trie[p] -> clone();
  for(int i=31; i>=0; i--) {
    int bit = (val >> i) & 1;
    node* &ch = curr -> ch[bit];
    curr = ch = ch -> clone();
  }
}
```

1.13 Treap (Implicit)

```
struct node {
  int prior, size;
  ll val, sum, lazy;
  node *l, *r;
  node(int v = 0) {
    val = sum = v; lazy = 0;
    prior = rand(); size = 1;
    l = r = NULL;
  }
} *root;
typedef node* pnode;
int sz(pnode t) { return t ? t -> size : 0; }
void upd_sz(pnode t) {
  if(t) t -> size = sz(t -> l) + 1 + sz(t -> r);
}
void push(pnode t) {
  if(!t || !t -> lazy) return;
  t -> val += t -> lazy;
  t -> sum += t -> lazy * sz(t);
  if(t -> l) t -> l -> lazy += t -> lazy;
  if(t -> r) t -> r -> lazy += t -> lazy;
  t -> lazy = 0;
}
void combine(pnode t) { // Reset then update
  if(!t) return;
  push(t -> l); push(t -> r);
  t -> sum = t -> val; // Reset
  if(t -> l) t -> sum += t -> l -> sum;
  if(t -> r) t -> sum += t -> r -> sum;
}
void split(pnode t, pnode &l, pnode &r, int pos, int
    add = 0) {
  if(!t) return void(l = r = NULL);
  push(t);
  int curr = sz(t -> l) + add;
  if(curr <= pos)
    split(t -> r, t -> r, r, pos, curr + 1), l = t;
  else split(t -> l, l, t -> l, pos, add), r = t;
  upd_sz(t); combine(t);
}
void merge(pnode &t, pnode l, pnode r) {
  push(l), push(r);
  if(!l || !r) t = l ? l : r;
  else if(l -> prior > r -> prior)
    merge(l -> r, l -> r, r), t = l;
  else merge(r -> l, l, r -> l), t = r;
  upd_sz(t); combine(t);
}
ll query(pnode t, int l, int r) {
  pnode L, mid, R;
  split(t, L, mid, l - 1); split(mid, t, R, r - l);
  ll ans = t -> sum;
  merge(mid, L, t); merge(t, mid, R);
  return ans;
}
void update(pnode t, int l, int r, ll v) {
  pnode L, mid, R;
  split(t, L, mid, l - 1); split(mid, t, R, r - l);
  t -> lazy += v;
  merge(mid, L, t); merge(t, mid, R);
}
void insert(pnode &t, int pos, int v) {
  pnode L, R, tmp, y = new node(v);
  split(t, L, R, pos - 1);
  merge(tmp, L, y); merge(t, tmp, R);
}
void Del(pnode &t, int pos) {
  pnode L, R, mid;
  split(t, L, mid, pos - 1); split(mid, t, R, 0);
  pnode tmp = t;
  merge(t, L, R); free(tmp);
}
```

2 Mathematics

2.1 Chinese Reminder Theorem

```
// mods are parewise coprime
ll CRT(vector<ll> &a, vector<ll> &m) {
  ll M = 1, ret = 0;
  for(ll num : m) M *= num;
  for(int i = 0; i < a.size(); i++) {
    ll tmp = (a[i] * (M / m[i])) % M;
    tmp = (tmp * inv(M / m[i], m[i])) % M;
    ret = (ret + tmp) % M;
  } return ret;
}
// (m, n) = 1. finds x: x mod m = a, x mod n = b
ll CRT(ll a, ll m, ll b, ll n) {
  ll x, y; egcd(m, n, x, y);
  ll ret = a * (y + m) % m * n + b * (x + n) % n * m;
  if(ret >= m * n) ret -= n * m;
  return ret;
}
// no restriction on (m, n);
ll CRT_Common(ll a, ll m, ll b, ll n) {
  ll d = __gcd(m, n);
  if(((b -= a) %= n) < 0) b += n;
  if(b % d) return -1; // No soln
  return d * CRT(0, m / d, b / d, n / d) + a;
}
```

2.2 Discrete Logarithm

```
// minimum x: a ** x = b (mod m)
ll shanks(ll a, ll b, ll m) {
  a %= m, b %= m; ll n = sqrt(m) + 1;
  unordered_map<ll, ll> mp;
  ll an = 1, base = 1, ans = -1;
  for(int i = 0; i < n; i++) an = (an * a) % m;
  for(int i = 1; i <= n; i++) {
    base = (base * an) % m;
    if(!mp.count(base))
```

```
      mp[base] = i;
  } base = b;
  for(int j = 0; j <= n; j++) {
    if(mp.count(base)) {
      ll ret = mp[base] * n - j;
      if(ans == -1 || (ret < m && ans > ret)) ans =
          ret;
    } base = (base * a) % m;
  } return ans;
}
```

2.3 Extended Euclid

```
ll egcd(ll a, ll b, ll &x, ll &y) {
  if(!b) { x = 1, y = 0; return a; }
  ll ret = egcd(b, a % b, y,x);
  y -= (a / b) * x;
  return ret;
}
ll inv(ll n, ll mod) {
  ll x, y;
  ll gcd = egcd(n, mod, x,y);
  return (x + mod) % mod;
}
```

2.4 Millar-Rabin

```
ll mul64(ll a, ll b, ll m) { // 64-bit long long
    multiplication
  a %= m, b %= m;
  ll ret = 0;
  for(; b; b >>= 1) {
    if(b & 1) ret = (ret + a) % m;
    a = (a + a) % m;
  } return ret;
}
ll Pow(int a, ll p, ll mod) { /* -_- */ }
bool miller_rabin(ll n, ll b) {
  ll m = n - 1, cnt = 0;
  while (m % 2 == 0) m >>= 1, ++cnt;
  ll ret = Pow(b, m, n);
  if (ret == 1 || ret == n - 1) return true;
  while (cnt > 0) {
    ret = mul64(ret, ret, n);
    if (ret == n - 1) return true;
    --cnt;
  } return false;
}
bool ptest(ll n) { // miller-rabin primality test
    if(n < 2) return false;
    if(n < 4) return true;
    const int BASIC[12] = { 2, 3, 5, 7, 11, 13, 17,
        19, 23, 29, 31, 37 };
    for(int i = 0; i < 12 && BASIC[i] < n; ++i)
      if(!miller_rabin(n, BASIC[i])) return false;
    return true;
}
```

2.5 Partitions

```
int partition(int n) {
  int[] dp = new int[n + 1];
  dp[0] = 1;
  for(int i = 1; i <= n; i++) {
    for(int j = 1, r = 1; i - (3 * j * j - j) / 2 >=
        0; j++, r *= -1) {
      dp[i] += dp[i - (3 * j * j - j) / 2] * r;
      if (i - (3 * j * j + j) / 2 >= 0)
        dp[i] += dp[i - (3 * j * j + j) / 2] * r;
    }
  } return dp[n];
}
```

2.6 Polard-Rho Factorization

```
LL pollard_rho(LL n, LL seed) {
  LL x, y;
  x = y = rand() % (n - 1) + 1;
  int head = 1, tail = 2;
  while (true) {
    x = mul64(x, x, n);
    x = (x + seed) % n;
    if (x == y) return n;
    LL d = gcd(max(x - y, y - x), n);
    if (1 < d && d < n) return d;
    if (++head == tail) y = x, tail <<= 1;
  }
}
void factorize(LL n, vector<LL> &divisor) {
  if (n == 1) return;
  if (ptest(n)) divisor.push_back(n);
  else {
    LL d = n;
    while (d >= n) d = pollard_rho(n, rand()%(n-1)+1);
    factorize(n / d, divisor);
    factorize(d, divisor);
  }
}
vector<LL> divisors(vector<LL> d) {
  vector<LL> ret = {1};
  sort(d.begin(), d.end());
  for (int i = 0, count = 1; i < d.size(); ++i) {
    if (i + 1 == d.size() || d[i] != d[i + 1]) {
      int c = ret.size();
      ret.resize(ret.size() * (count+1));
      LL n = 1;
      for (int j = 1; j <= count + 1; ++j) {
        for (int k = 0; k < c; ++k) {
          ret[(j-1)*c+k] = ret[k]*n;
        }
        n *= d[i];
      }
      count = 1;
    } else count += 1;
  }
  sort(ret.begin(), ret.end());
  return ret;
}
```

2.7 Polynomials

2.7.1 Fast Fourier Transform

```
// Call calcw() in beginning to precalculate all
    w_pre[]
// Call calcrev(sz) before a multiplication.
// Take precautions while calculating sz.

#pragma GCC optimize("Ofast,unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,
    popcnt,abm,mmx,avx,tune=native")

typedef long double ld;
ld PI = acos(-1);

struct base {
```

```
  ld a, b;
  base(ld _a = 0.0, ld _b = 0.0) : a(_a), b(_b) {}
  const base operator + (const base &c) const
    { return base(a + c.a, b + c.b); }
  const base operator - (const base &c) const
    { return base(a - c.a, b - c.b); }
  const base operator * (const base &c) const
    { return base(a * c.a - b * c.b, a * c.b + b *
        c.a); }
};

const int N = 1 << 20;
base w_pre[N|1], w[N|1]; int rev[N];

void calcw() {
  for(int i = 0; i <= N; ++i)
    w_pre[i] = base(cos(2*PI/N*i), sin(2*PI/N*i));
}
void calcrev(int n) {
  int sz = 31 - __builtin_clz(n); sz = abs(sz);
  for(int i = 1; i < n - 1; ++i)
    rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << sz - 1);
}
void fft(base *p, int n, int dir) {
  for(int i = 1; i < n - 1; i++)
    if(i < rev[i]) swap(p[i], p[rev[i]]);
  for(int h = 1; h < n; h <<= 1) { int l = h << 1;
    if(!dir) for(int j = 0; j < h; ++j) w[j] =
        w_pre[N/l*j];
    else for(int j = 0; j < h; ++j) w[j] =
        w_pre[N-N/l*j];
    for(int j = 0 ; j < n; j += l) {
      base t, *wn = w;
      base *u = p + j, *v = u + h, *e = v;
      while(u != e) {
        t = *v * *wn;
        *v = *u - t;
        *u = *u + t;
        ++u, ++v, ++wn;
      }
    }
  } if(dir) for(int i = 0; i < n; ++i) p[i].a /= n,
      p[i].b /= n;
}
```

2.7.2 Fast Walsh Hadamard Transform

```
// Hadamard Matrix -
// 1. For XOR-Convolution -
// H = H_inverse = {{1, 1}, {1, -1}}
// 2.For AND-Convolution -
// H         = {{0, 1}, {1, 1}}
// H_inverse = {{-1, 1}, {1, 0}}
// 3. For OR-Convolution -
// H         = {{1, 1}, {1, 0}}
// H_inverse = {{0, 1}, {1, -1}}

const int mod = 1e9 + 7;
void fwht(int *p, int n) {
  for(int len = 1; 2 * len <= n; len <<= 1) {
    for(int i = 0; i < n; i += 2 * len) {
      for(int j = 0; j < len; j++) {
        int a = p[i + j];
        int b = p[i + j + len];
        p[i + j] = (a + b) % mod;
        p[i + j + len] = (mod + a - b) % mod;
      }
    }
  }
}
```

2.7.3 Number Theoretic Transform

```
const int N = 1 << 18, mod = 7 * 17 * (1 << 23) + 1,
    g = 3;
int rev[N], w[N], inv_n;

void prepare(int &n) {
  int sz = 31 - __builtin_clz(n); sz = abs(sz);
  int r = Pow(g, (mod - 1) / n);
  inv_n = Pow(n, mod - 2);
  w[0] = w[n] = 1;
  for(int i = 1; i < n; ++i)
    w[i] = (ll)w[i - 1] * r % mod;
  for(int i = 1; i < n; ++i)
    rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (sz -
        1));
}
void ntt(int *a, int n, int dir) {
  for(int i = 1; i < n - 1; ++i)
    if(i < rev[i]) swap(a[i], a[rev[i]]);
  for(int m = 2; m <= n; m <<= 1) {
    for(int i = 0; i < n; i += m) {
      for(int j = 0; j < (m >> 1); ++j) {
        int &u = a[i + j], &v = a[i + j + (m >> 1)];
        int t = (ll)v * w[dir ? n - n / m * j : n / m
            * j] % mod;
        v = u - t < 0 ? u - t + mod : u - t;
        u = u + t >= mod ? u + t - mod : u + t;
      }
    }
  } if(dir) for(int i = 0; i < n; ++i) a[i] =
      (ll)a[i] * inv_n % mod;
}

// primitive root, finding the number with order p-1
int primitive_root(int p) {
  vector<int> factor;
  int tmp = p - 1;
  for(int i = 2; i * i <= tmp; ++i) {
    if (tmp % i == 0) {
      factor.push_back(i);
      while (tmp % i == 0) tmp /= i;
    }
  }
  if(tmp != 1) factor.push_back(tmp);
  for(int root = 1; ; ++root) {
    bool flag = true;
    for(int i = 0; i < factor.size(); ++i) {
      if(Pow(root, (p - 1) / factor[i], p) == 1) {
        flag = false;
        break;
      }
    }
    if (flag) return root;
  }
}
```

2.7.4 Operations on Formal Power Series

Inverse of a Polynomial: Let, $g_n(z)f(z) \equiv 1 \pmod{z^n}$. Then, $g_{2n}(z) \equiv 2g_n(z) - g_n(z)^2 f(z) \pmod{z^{2n}}$.

```
// b * a = 1 (mod z^n). make sure N >= 2n
int ta[N], tb[N], tc[N];
void polyinv(int *a, int *b, int n) {
  if(n == 1) return void(b[0] = Pow(a[0], mod - 2));
  polyinv(a, b, n >> 1);
```

```
  for(int i = 0; i < n; ++i)
    ta[i] = a[i], tb[i] = b[i];
  for(int i = n; i < (n << 1); ++i)
    ta[i] = tb[i] = 0;
  n <<= 1; prepare(n);
  ntt(ta, n, 0), ntt(tb, n, 0);
  for(int i = 0; i < n; ++i)
    b[i] = (ll) tb[i] * (2 + mod - (ll) ta[i] * tb[i]
         % mod) % mod;
  ntt(b, n, 1);
  fill(b + (n >> 1), b + n, 0);
}
```

Square Root of a Polynomial: Let, $g_n(z)^2 \equiv f(z) \pmod{z^n}$. Then, $g_{2n}(z) \equiv 2^{-1}\{g_n(z) + f(z)g_n(z)^{-1}\} \pmod{z^{2n}}$.

```
// b^2 = a (mod z^n). Make sure N >= 2n
int ta[N], tb[N], inv2 = Pow(2, mod - 2);
void polysqrt(int *a, int *b, int n) {
  if(n == 1) return void(b[0] = 1); // b0 = x : x^2 ≡ a0
  polysqrt(a, b, n >> 1);
  polyinv(b, tb, n);
  for(int i = 0; i < n; ++i)
    ta[i] = a[i];
  for(int i = n; i < (n << 1); ++i)
    ta[i] = tb[i] = 0;
  n <<= 1; prepare(n);
  ntt(ta, n, 0); ntt(tb, n, 0);
  for(int i = 0; i < n; ++i)
    ta[i] = (ll) ta[i] * tb[i] % mod;
  ntt(ta, n, 1);
  for(int i = 0; i < n; ++i)
    b[i] = (ll) inv2 * (ta[i] + b[i]) % mod;
  fill(b + (n >> 1), b + n, 0);
}
```

Rising Factorial: Let $P_N(x) = (x+1)(x+2)\cdots(x+N) = \sum_{i=0}^{N} c_i x^i$. Now, $P_{2N}(x) = P_N(x)P_N(x+N)$. Here, $P_N(x+N) = \sum_{i=0}^{N} c_i(x+N)^i = \sum_{i=0}^{N} h_i x^i$ where,

$$h_i = \frac{1}{i!} \cdot [x^{N-i}]A(x)B(x)$$

$$A(x) = \sum_{i=0}^{N} (c_{N-i} \cdot (N-i)!)\, x^i \text{ and } B(x) = \sum_{i=0}^{N} \left(\frac{N^i}{i!}\right) x^i$$

```
// f = (x+1)(x+2)...(x+n)
int f[M], h[M], a[M], b[M];
int fact[M], inv[M];
void build(int n) {
    if(n == 1) return void(f[0] = f[1] = 1);
  if(n & 1) {
    build(n - 1);
        for(int i = n; i >= 1; i--) {
            f[i] = f[i - 1] + (ll) n * f[i] % mod;
            if(f[i] >= mod) f[i] -= mod;
        } f[0] = (ll) f[0] * n % mod;
        return;
  }
    n >>= 1; build(n);
  int t = n + n + 1, sz = 1;
  while(sz < t) sz <<= 1;
  prepare(sz);

  for(int i = 0; i <= n; i++)
    a[i] = (ll) f[n - i] * fact[n - i] % mod;
  for(int i = 0, p = 1; i <= n; i++) {
    b[i] = (ll) p * inv[i] % mod;
    p = (ll) p * n % mod;
  }
    for(int i = n + 1; i < sz; i++) a[i] = b[i] = 0;

  ntt(a, sz); ntt(b, sz);
  for(int i = 0; i < sz; i++)
    h[i] = (ll) a[i] * b[i] % mod;
  ntt(h, sz, 1);

  reverse(h, h + n + 1);
  for(int i = 0; i <= n; i++)
    h[i] = (ll) h[i] * inv[i] % mod;
  for(int i = n + 1; i < sz; i++) f[i] = h[i] = 0;

  ntt(h, sz); ntt(f, sz);
  for(int i = 0; i < sz; i++)
    f[i] = (ll) f[i] * h[i] % mod;
  ntt(f, sz, 1);
}
```

2.7.5 NTT Friendly Primes

n	2^n	a	$p = a \times 2^n + 1$	g
5	32	3	97	5
6	64	3	193	5
7	128	2	257	3
8	256	1	257	3
9	512	15	7681	17
10	1024	12	12289	11
11	2048	6	12289	11
12	4096	3	12289	11
13	8192	5	40961	3
14	16384	4	65537	3
15	32768	2	65537	3
16	65536	1	65537	3
17	131072	6	786433	10
18	262144	3	786433	10
19	524288	11	5767169	3
20	1048576	7	7340033	3
21	2097152	11	23068673	3
22	4194304	25	104857601	3
23	8388608	20	167772161	3
23	8388608	119	998244353	3
24	16777216	10	167772161	3
25	33554432	5	167772161	3
26	67108864	7	469762049	3
27	134217728	15	2013265921	31

Random Primes: 100003, 200003, 300007, 400009, 500009, 600011, 700001, 800011, 900001,
1000003, 2000003, 3000017, 4100011, 5000011, 8000009, 9000011,
10000019, 20000003, 50000017, 50100007,
100000007, 100200011, 200100007, 250000019

2.8 Segmented Sieve

```
// table[i-L] == true <=> i == prime
const int SQRTN = 1<<16; // upperbound of sqrt(H) + 10
vector<bool> segmentSieve(ll L, ll H) {
  static ll p[SQRTN];
  static int lookup = 0;
  if (!lookup) {
    for (ll i = 2; i < SQRTN; ++i) p[i] = i;
    for (ll i = 2; i*i < SQRTN; ++i)
      if (p[i])
        for (ll j = i*i; j < SQRTN; j += i)
          p[j] = 0;
    remove(p, p+SQRTN, 0);
    lookup = 1;
  }
  vector<bool> table(H - L);
  for (ll i = L; i < H; ++i) table[i - L] = 1;
  for (ll i = 0, j; p[i] * p[i] < H; ++i) { // O(
      \sqrt(H) )
    if (p[i] >= L) j = p[i] * p[i];
    else if (L % p[i] == 0) j = L;
    else j = L - (L % p[i]) + p[i];
    for (; j < H; j += p[i]) table[j-L] = 0;
  }
  return table;
}
```

2.9 Simplex

```
double a[maxN][maxM], b[maxN], c[maxM], d[maxN][maxM];
int ix[maxN + maxM]; // !!! Array all indexed from 0
// Target: max{cx|Ax<=b,x>=0}
// n: Constraints, m: Variables
double simplex(double a[maxN][maxM], double b[maxN],
    double c[maxM], int n, int m) {
  ++m;
  int r = n, s = m - 1;
  memset(d, 0, sizeof(d));
  for(int i = 0; i < n + m; ++i) ix[i] = i;
  for(int i = 0; i < n; ++i) {
    for(int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
    d[i][m - 1] = 1;
    d[i][m] = b[i];
    if(d[r][m] > d[i][m]) r = i;
  }
  for(int j = 0; j < m - 1; ++j) d[n][j] = c[j];
  d[n + 1][m - 1] = -1;
  for(double dd;;) {
    if(r < n) {
      int t = ix[s]; ix[s] = ix[r + m]; ix[r + m] = t;
      d[r][s] = 1.0 / d[r][s];
      for(int j = 0; j <= m; ++j) if (j != s) d[r][j]
          *= -d[r][s];
      for(int i = 0; i <= n + 1; ++i) if (i != r) {
        for(int j = 0; j <= m; ++j) if (j != s)
            d[i][j] += d[r][j] * d[i][s];
        d[i][s] *= d[r][s];
      }
    }
    r = -1; s = -1;
    for(int j = 0; j < m; ++j) if (s < 0 || ix[s] >
        ix[j]) {
      if(d[n + 1][j] > eps || (d[n + 1][j] > -eps &&
          d[n][j] > eps)) s = j;
    }
    if(s < 0) break;
    for(int i = 0; i < n; ++i) if (d[i][s] < -eps) {
      if(r < 0 || (dd = d[r][m] / d[r][s] - d[i][m] /
          d[i][s]) < -eps || (dd < eps && ix[r + m] >
          ix[i + m])) r = i;
    }
    if(r < 0) return -1; // not bounded
  }
  if(d[n + 1][m] < -eps) return -1; // not executable
  double ans = 0;
  for(int i = m; i < n + m; ++i) { // the missing
      enumerated x[i] = 0
    if(ix[i] < m - 1) ans += d[i - m][m] * c[ix[i]];
  } return ans;
}
```

2.10 Special Numbers

2.10.1 Binomial Coefficients

■ Calculating Binomial Coefficient with large mod and small n, r

```
// Requires inv, egcd, Pow, prime power factorize,
    CRT :P
ll Leg(ll n, ll p) { // Largest x : p^x | n!
  ll ans = 0;
  while(n) ans += n /= p;
  return ans;
}
ll s_fact[maxn];
ll spf(ll x, ll p, ll mod){
    ll ret = Pow(s_fact[mod - 1], x / mod, mod);
    if (x >= p) ret = ret * spf(x / p, p, mod) % mod;
    return ret * s_fact[x % mod] % mod;
```

```
}
ll C_mod_p_q(ll n, ll r, ll p, ll q) {
  if(r > n) return 0;
  if(n == r || r == 0) return 1;
  ll M = Pow(p, q, 1e18);
  ll t = Leg(n, p) - Leg(r, p) - Leg(n - r, p);
  if(t >= q) return 0;
  s_fact[0] = 1;
  for(ll i = 1; i < M; i++)
    s_fact[i] = s_fact[i-1] * ((i%p)?i:1) % M;
  ll res = spf(n, p, M);
  res *= inv(spf(r, p, M) * spf(n - r, p, M) % M, M);
  res %= M;
  res *= Pow(p, t, M);
  return res % M;
}
ll C(ll n, ll r, int mod) {
  if(r > n || mod == 1) return 0;
  if(n == r || r == 0) return 1;
  vector<ii> ppf = factorize(mod);
  // factorize should return prime power
     factorization.
  vector<ll> a, m;
  for(ii p : ppf) {
    ll pp = Pow(p.first, p.second, 1e7);
    ll aa = C_mod_p_q(n, r, p.first, p.second);
    a.push_back(aa);
    m.push_back(pp);
  } return CRT(a, m);
}
```

■ **Calculating Binomial Coefficient with large n, r and small prime modulo (Lucas)**

```
// Binomial Coefficient mod small prime p and large
     n, r.
int c[p][p]; // Precompute
int C(int n, int k) {
  return n == 0 ? 1 : c[n % p][k % p] * C(n / p, k /
     p) % p;
}
```

■ **Parity of Binomial Coefficient**

- $\binom{n}{k}$ is odd, if and only if $k \subseteq n$.
- For fixed n, number of odd $\binom{n}{k} = 2^{popcount(n)}$.
- Total number of odd entries in first n rows of pascal's triangle - $f(1) = 1, f(2k) = 3f(k), f(2k+1) = 2f(k) + f(k+1)$

2.10.2 Catalan Numbers

Formulas:

- n^{th} Catalan Number is given by $C_n = \frac{1}{n+1}\binom{2n}{n} = \prod_{k=2}^{n} \frac{n+k}{k}$. Trademark: $[1, 1, 2, 5, 14, 42, 132, 429, 1430]$.
- Catalan numbers satisfy recurrences -

$$C_0 = 1;\; C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

$$C_0 = 1;\; C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

Applications:

- C_n counts number of Dyck words of length $2n$. A dyck word consists of n 'X' and n 'Y', with no prefix having more 'Y' than 'X'.
- C_n counts number of valid expression of n pairs of parenthesis.
- C_n is the number of different ways $n+1$ factors can be completely parenthesized. Ex. $((ab)c)d, (a(bc))d, (ab)(cd), a((bc)d), a(b(cd))$.
- C_n is number of full binary trees (a node hase either 2 childs or none) with $n+1$ leafs (and n internal nodes).
- C_n is number of monotonic lattice paths from $(0, 0)$ to (n, n) wihout crossing $x = y$ line. Rotate grid $45°$ to get mountain range.
- C_n is number of triangulation with non-crossing line segments, of convex polygon with $n+2$ edges.
- C_n is number of stack sortable permutations of $[1, 2, \cdots, n]$. A permutations $w = u + n + v$ is stack sortable if $S(w) = [1, 2, \cdots, n]$, and $S(w) = S(u) + S(v) + n$.
- C_n is number of permutations of $[1, 2, \cdots, n]$, which avoids a pattern of length 3.
- C_n is the number of ways to tile a stairstep shape of height n with n rectangles.

2.10.3 Eulerian Numbers

■ **Definition:** $\left\langle {n \atop k} \right\rangle$ counts the number of permutations of the numbers from 1 to n in which exactly k numbers are greater than the previous element. Trademark: $\left\langle {n \atop 4} \right\rangle \Rightarrow [1, 11, 11, 1, 0]$.
The permutations of the multiset $[1, 1, 2, 2, \cdots, n, n]$ which have the property that for each k, all the numbers appearing between the two occurrences of k in the permutation are greater than k are counted by the double factorial number $(2n-1)!!$. The Eulerian number of the second kind, denoted $\left\langle\!\left\langle {n \atop k} \right\rangle\!\right\rangle$, counts the number of all such permutations that have exactly m ascents.

■ **Formulas:**

- Recurrence:

$$\left\langle {n \atop k} \right\rangle = (k+1)\left\langle {n-1 \atop k} \right\rangle + (n-k)\left\langle {n-1 \atop k-1} \right\rangle,\; \left\langle {0 \atop k} \right\rangle = [k = 0].$$

$$\left\langle\!\left\langle {n \atop k} \right\rangle\!\right\rangle = (k+1)\left\langle\!\left\langle {n-1 \atop k} \right\rangle\!\right\rangle + (2n-k-1)\left\langle\!\left\langle {n-1 \atop k-1} \right\rangle\!\right\rangle,$$

- Reflection: $\left\langle {n \atop k} \right\rangle = \left\langle {n \atop n-k-1} \right\rangle$.
- Coefficients that arise when ordinary poweres are written in terms of consecutive binomial coefficients -

$$x^n = \sum_k \left\langle {n \atop k} \right\rangle \binom{x+k}{n}.$$

- Explicit Formula: $\left\langle {n \atop m} \right\rangle = \sum_{k=0}^{m} \binom{n+1}{k}(m+1-k)^n(-1)^k$.
- Special Values: $\left\langle {n \atop 0} \right\rangle = 1$; $\left\langle {n \atop 1} \right\rangle = 2^n - n - 1$; $\left\langle {n \atop 2} \right\rangle = 3^n - (n+1)2^n + \binom{n+1}{2}$.
- Row Sum: $\sum_{k=0}^{n} \left\langle {n \atop k} \right\rangle = n!$; $\sum_{k=0}^{n} \left\langle\!\left\langle {n \atop k} \right\rangle\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$

2.10.4 Fibonacci Numbers

Definition: $F_n = F_{n-1} + F_{n-2}$; $F_0 = 0, F_1 = 1$.

Identities:

- Closed Form: $\frac{\phi^n - (1-\phi)^n}{\sqrt{5}}$; $\phi = \frac{1+\sqrt{5}}{2}$
- Partial Sum: $\sum_{i=1}^{n} F_i = F_{n+2} - 1$.
- Sum of squared terms: $\sum_{i=1}^{n} F_i^2 = F_n F_{n+1}$.
- Sum of odd terms: $\sum_{i=0}^{n-1} F_{2i+1} = F_{2n}$.
- Sum of even terms: $\sum_{i=1}^{n} F_{2i} = F_{2n+1} - 1$.
- Cassini's identity: $F_n^2 - F_{n+1}F_{n-1} = (-1)^{n-1}$.
- Catalan's identity: $F_n^2 - F_{n+r}F_{n-r} = (-1)^{n-r}F_r^2$.
- GCD: $gcd(F_n, F_m) = F_{gcd(m,n)}$. Pairwise gcd of three consecutive Fibonacci number is 1.
- Divisibility: $\begin{cases} p = 5 & \Rightarrow p \mid F_p, \\ p \equiv \pm 1 \pmod 5 & \Rightarrow p \mid F_{p-1}, \\ p \equiv \pm 2 \pmod 5 & \Rightarrow p \mid F_{p+1}. \end{cases}$
- n steps forward:

$$F_{m+n} = F_{m-1}F_n + F_m F_{n+1}$$

$$F_{2n-1} = F_n^2 + F_{n-1}^2$$

$$F_{2n} = (F_{n-1} + F_{n+1})F_n = (2F_{n-1} + F_n)F_n$$

- Right triangles:

$$F_{2n-1}^2 = (2F_nF_{n-1})^2 + (F_n^2 - F_{n-1}^2)^2$$

$$(F_nF_{n+3})^2 = (2F_{n+1}F_{n+2})^2 + (F_{n+1}^2 + F_{n+2}^2)^2$$

- Periodicity: If the members of the Fibonacci sequence are taken mod n, the resulting sequence is periodic with period at most $6n$.

2.10.5 Stirling Numbers of the First Kind

■ **Definition:** $\left[{n \atop k}\right]$ counts the number of ways to arrange n objects into k cycles. Trademark: $\left[{n \atop 4}\right] \Rightarrow [6, 11, 6, 1]$.

■ **Formulas:**

- Recurrence: $\left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$, $\left[{n \atop 0}\right] = [n = 0]$.
- Row Generator:

$$G_n(z) = \sum_{k=0} \left[{n \atop k}\right] x^k = x^{\overline{n}} = x(x+1)(x+2)\cdots(x+n-1)$$

In other works, Stirling Cycle numbers are the coefficients of ordinary powers that yeild rasing factorial powers. Using falling factorial generates signed stirling cycles numbers instead.

- Special Values: $\left[{n \atop 1}\right] = (n-1)!$, $\left[{n \atop 2}\right] = (n-1)!H_{n-1}$, $\left[{n \atop n-1}\right] = \binom{n}{2}$, $\left[{n \atop n-2}\right] = \frac{1}{4}(3n-1)\binom{n}{3}$, $\left[{n \atop n-3}\right] = \binom{n}{2}\binom{n}{4}$.
- Parity: $\left[{n \atop k}\right] \equiv \binom{n//2}{m - n//2} \pmod 2$
- Every cycle represents a permutation. Hence, $\sum_{k=0}^{n} \left[{n \atop k}\right] = n! = \left[{n+1 \atop 1}\right]$.

■ **Applications:**

- Number of permutation of first n natural numbers with k elements maximum than all elements to its left is $\left[{n \atop k}\right]$.

2.10.6 Stirling Numbers of the Second Kind

■ **Definition:** $\left\{{n \atop k}\right\}$ stands for the number of ways to partition a set of n elements into k non-empty subsets. Trademark: $\left\{{n \atop 4}\right\} \Rightarrow [1, 7, 6, 1]$.

■ **Formulas:**

- Recurrence: $\left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$, $\left\{{n \atop 0}\right\} = [n = 0]$
- Column Generator:

$$G_k(x) = \sum_n \left\{{n \atop k}\right\} x^n = \frac{x^n}{(1-x)(1-2x)(1-3x)\cdots(1-k}$$

- Explicit Formula: $\left\{{n \atop k}\right\} = \sum_{r=1}^{k} (-1)^{k-r} \frac{r^n}{r!(k-r)!}$
- Coefficients that arise when x^n is written in terms of falling factorials $x^{\underline{k}}$:

$$x^{\underline{k}} = x(x-1)(x-2)\cdots(x-k+1)$$

$$x^n = \sum_{k=0}^{n} \left\{{n \atop k}\right\} x^{\underline{k}}$$

.

- Special Values: $\left\{{n \atop 1}\right\} = \left\{{n \atop n}\right\} = 1$, $\left\{{n \atop 2}\right\} = 2^{n-1} - 1$, $\left\{{n \atop n-1}\right\} = \binom{n}{2}$

- Parity: $\left\{ {n \atop k} \right\} \equiv [((n-k)\&((k-1)//2) = 0]$
- Reduced Stirling numbers of the second kind: Lets use $S^d(n,k)$ to denote number of ways to partition first n natural numbers into k non-empty subsets such that all elements in each subset have pairwise distance at least d. Then $S^d(n,k) = \left\{ {n-d+1 \atop k-d+1} \right\}, n \geq k \geq d$.
- Associated Stirling numbers of the second kind: Number of ways to partition a set of n objects into k subsets, with each subset containing at least r elements is $S_r(n,k) = kS_r(n-1,k) + \binom{n-1}{r-1} S_r(n-r, k-1)$.

■ **Applications:**

- Draw n cards from a deck of k cards, with replacement. Probability that each card was drawn atleast once is given by $\left\{ {n \atop k} \right\} k!/k^n$.
- $\left\{ {n \atop k} \right\} k!$ counts the number of ways n labled objects can be distributed into k nonempty parcels.
- Number of partitions of first n natural numbers into k nonempty subsets of nonconsecutive integers is $\left\{ {n-1 \atop k-1} \right\}$.
- Number of k size collections of pairwise disjoint, nonempty subsets of $[n]$ is $\left\{ {n+1 \atop k+1} \right\}$. Example: $\left\{ {3+1 \atop 2+1} \right\} = 6 \Rightarrow \{(1)(23), (12)(3), (13)(2), (1)(2), (1)(3), (2)(3)\}$
- $\left\{ {n \atop k} \right\}$ = Number of patterns (ex. $AADCBB = XXEGTT$) of length n using k distinct symbols.
- Number of ways to nest n matryoshkas so that exactly k matryoshkas are not contained in any other matryoshka is $\left\{ {n \atop k} \right\}$.

3 Graph Theory

3.1 Articulation Point and Bridge

```
int vis[N], low[N], cut[N], now = 0;

void dfs(int u, int par) {
  low[u] = vis[u] = ++now; int ch = 0;
  for(int v : adj[u]) if(v - par) {
    if(vis[v]) low[u] = min(low[u], vis[v]);
    else { ch++;
      dfs(v, u);
      low[u] = min(low[u], low[v]);
      if(par + 1 && low[v] >= vis[u])
        cut[u] = 1;
      if(low[v] > vis[u]) {
        printf("Bridge %d -- %d\n", u, v);
      }
    }
  } if(par == -1 && ch > 1) cut[u] = 1;
}
```

3.2 Directed Minimum Spanning Tree

```
// Directed MST starting from node r.
int dmst(vector<edge> &e, int r, int n) {
  int ans = 0;
  int m = n;
  while(true) {
    vector<int> lo(m, inf), pi(m, inf);
    for(int i = 0; i < e.size(); ++i) {
      int u = e[i].u, v = e[i].v, w = e[i].w;
      if(w < lo[v] && u != v)
        lo[v] = w, pi[v] = u;
    }
    lo[r] = 0;
    for(int i = 0; i < lo.size(); ++i) if(i != r)
      if(lo[i] == inf) return -1;
    int cur_id = 0;
    vector<int> id(m, -1), mark(m, -1);
    for(int i = 0; i < m; ++i) {
      ans += lo[i];
      int u = i;
      while(u != r && id[u] < 0 && mark[u] != i) {
        mark[u] = i;
        u = pi[u];
      }
      if(u != r && id[u] < 0) { // Cycle
        for(int v = pi[u]; v != u; v = pi[v])
          id[v] = cur_id;
        id[u] = cur_id++;
      }
    }
    if(cur_id == 0) break;
    for(int i = 0; i < m; ++i)
      if(id[i] < 0) id[i] = cur_id++;
    for(int i = 0; i < e.size(); ++i) {
      int u = e[i].u, v = e[i].v, w = e[i].w;
      e[i].u = id[u];
      e[i].v = id[v];
      if (id[u] != id[v])
        e[i].w -= lo[v];
    }
    m = cur_id;
    r = id[r];
  } return ans;
}
```

3.3 Eulerian Path and Circuit

Unirected Graph: Number of nodes with odd degree should be 0 or 2. Start with a odd degree node if there are any. This code works for Eulerian Circuit in undirected graph too -

```
struct Edge;
typedef list<Edge>::iterator iter;
struct Edge { int v; iter u; Edge(int _v) : v(_v){} };
list<Edge> adj[N]; vector<int> path;
void dfs(int u) {
  while(adj[u].size()) {
    int v = adj[u].front().v;
    adj[v].erase(adj[u].front().u);
    adj[u].pop_front();
    dfs(v);
  } path.push_back(u);
}
void addEdge(int u, int v) {
  adj[u].push_front(Edge(v));
  adj[v].push_front(Edge(u));
  iter a = adj[u].begin(), b = adj[v].begin();
  a -> u = b, b -> u = a;
}
```

Directed Graph: Number of nodes where indegree and outdegree differ can be 0 or 2. Start with a vertex having out degree greater than in degree.

```
void dfs(int u) {
    while(done[u] < adj[u].size())
        dfs(adj[u][done[u]++]);
    path.push_back(u);
} // Eulerian path in 'path', reversed order.
```

3.4 Hopcroft Karp

```
// Left nodes are numbered from 1 to n
// Right nodes are numbered from n + 1 to n + m
// Complexity O(E sqrt V)

const int N = 2*5e4 + 10;
const int inf = 1e9;

vector<int> adj[N];
int match[N], dist[N], n, m, p;

bool bfs() {
  queue<int> Q;
  for(int i = 1; i <= n; i++) {
    if(!match[i]) dist[i] = 0, Q.push(i);
    else dist[i] = inf;
  } dist[0] = inf;
  while(!Q.empty()) {
    int u = Q.front(); Q.pop();
    if(u == 0) continue;
    for(int v : adj[u]) {
      if(dist[match[v]] == inf) {
        dist[match[v]] = dist[u] + 1;
        Q.push(match[v]);
      }
    }
  } return dist[0] != inf;
}
bool dfs(int u) {
  if(!u) return 1;
  for(int v : adj[u]) {
    if(dist[match[v]] == dist[u] + 1) {
      if(dfs(match[v])) {
        match[u] = v, match[v] = u;
        return 1;
      }
    }
  } dist[u] = inf; return 0;
}
int hopcroft_karp() {
  int ret = 0;
  while(bfs())
    for(int i = 1; i <= n; i++)
      if(!match[i] && dfs(i))
        ret++;
  return ret;
}
```

3.5 Hungarian

```
struct Hungarian {
  static const int N = 650;
  static const int INF = 2147483647;
  int n, match[N], vx[N], vy[N];
  int e[N][N], lx[N], ly[N], slack[N];

  void init(int _n) { n = _n;
    for(int i = 0; i < n; i++)
      for(int j = 0; j < n; j++)
        e[i][j] = 0;
  }
  void addEdge(int x, int y, int w) { e[x][y] = w; }
  bool DFS(int x){
    vx[x] = 1;
    for(int y = 0; y < n; y++) {
      if(vy[y]) continue;
      if(lx[x] + ly[y] > e[x][y]) {
        slack[y] = min(slack[y], lx[x] + ly[y] -
            e[x][y]);
      } else {
        vy[y] = 1;
        if(match[y] == -1 || DFS(match[y])) {
          match[y] = x;
          return true;
        }
      }
    } return false;
  }
  int solve() {
    fill(match, match + n, -1);
    fill(lx, lx + n, -INF);
    fill(ly, ly + n, 0);
    for(int i = 0; i < n; i++)
      for(int j = 0; j < n; j++)
        lx[i] = max(lx[i], e[i][j]);
    for(int i = 0; i < n; i++) {
      fill(slack, slack + n,INF);
      while(true) {
        fill(vx, vx + n,0);
        fill(vy, vy + n,0);
        if(DFS(i)) break;
        int d = INF;
        for(int j = 0; j < n; j++)
          if(!vy[j]) d = min(d, slack[j]);
        for(int j = 0; j < n; j++){
          if(vx[j]) lx[j] -= d;
          if(vy[j]) ly[j] += d;
          else slack[j] -= d;
        }
      }
    }
    int res = 0;
    for(int i = 0; i < n; i++)
      res += e[match[i]][i];
    return res;
  }
} graph;
```

3.6 Kth Best Sortest Path

```
const int N = 105, K = 100, inf = 1e9;
vector<int> adj[N], cost[N];
vector<vector<int> > ans[N]; // ans[u][v][k] = ...
int n, m;
struct info {
  int v, w, k;
  bool operator < (const info &p) const {
    return w > p.w;
  }
};
// call with dist[N+1][K] = ans[i]
void kthbest(int s, vector<vector<int> > &dist) {
  info u, v;
  for(int i = 1; i <= n; i++)
    for(int j = 0; j < K; j++)
      dist[i][j] = inf;
  u.v = s, u.k = 0, u.w = 0;
```

```
  priority_queue<info> Q;
  Q.push(u);
  while(!Q.empty()) {
    u = Q.top(); Q.pop();
    for(int i = 0; i < adj[u.v].size(); i++) {
      v.v = adj[u.v][i];
      int d = u.w + cost[u.v][i];
      for(v.k = u.k; v.k < K && d != inf; v.k++) {
        if(dist[v.v][v.k] > d) {
          swap(d, dist[v.v][v.k]);
          v.w = dist[v.v][v.k];
          Q.push(v);
          break;
        }
      }
      for(v.k++; v.k < K && d != inf; v.k++)
        if(dist[v.v][v.k] > d)
          swap(d, dist[v.v][v.k]);
    }
  }
}
```

3.7 Kuhn

```
// Complexity O(V * min(BPM^2, E)).
// Left nodes - 0, 1, ..., n - 1. Right rest.
vector<int> adj[N];
int vis[N], match[N], iter;
int dfs(int u) {
  if(vis[u] == iter) return 0;
  vis[u] = iter;
  for(int v : adj[u]) {
    if(match[v] < 0 || dfs(match[v])) {
      match[u] = v, match[v] = u;
      return 1;
    }
  } return 0;
}
int kuhn() {
  memset(match, -1, sizeof match);
  int ans = 0;
  for(int i = 0; i < n; i++) {
    ++iter; ans += dfs(i);
  } return ans;
}
```

3.8 Manhattan Minimum Spanning Tree

```
struct point {
  int x, y, idx;
  bool operator<(const point &p) const {
    return x == p.x ? y < p.y : x < p.x;
  }
} p[N];

struct node { int val, p; } T[N];

int query(int x) {
  int r = inf, p = -1;
  for(; x <= n; x += x & -x)
    if(r > T[x].val) r = T[x].val, p = T[x].p;
  return p;
}
void update(int x, int w, int p) {
  for(; x > 0; x -= x & -x)
    if(T[x].val > w) T[x].val = w, T[x].p = p;
}
void addEdge(int u, int v, int c) { /* -_- */ }
int kruskal() { /* -_- */ }

// Adds edge to nearest neighbour in each octant.
// In a fixed dir, for a point (x, y), finds point
// (x', y') such that x <= x', (y' - x') <= (y - x)
// and (x' + y') is minimum possible.

int manhattan() {
  for(int i = 1; i <= n; ++i) p[i].idx = i;
  for(int dir = 1; dir <= 4; ++dir) {
    if(dir == 2 || dir == 4) {
      for(int i = 1; i <= n; ++i)
        swap(p[i].x, p[i].y);
    } else if(dir == 3) {
      for(int i = 1; i <= n; ++i)
        p[i].x = -p[i].x;
    }
    sort(p + 1, p + 1 + n);
    vector<int> v;
    static int a[N];
    for(int i = 1; i <= n; ++i) {
      a[i] = p[i].y - p[i].x;
      v.push_back(a[i]);
    }
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
    for(int i = 1; i <= n; ++i)
      a[i] = lower_bound(v.begin(), v.end(), a[i]) -
          v.begin() + 1;
    for(int i = 1; i <= n; ++i)
      T[i].val = inf, T[i].p = -1;
    for(int i = n; i >= 1; --i) {
      int pos = query(a[i]);
      if(pos != -1) addEdge(p[i].idx, p[pos].idx,
          dist(p[i], p[pos]));
      update(a[i], p[i].x + p[i].y, i);
    }
  } return kruskal();
}
```

3.9 Maximum Closure

A closure of a directed graph is a set of vertices with no outgoing edges.
Formally, given directed graph $G = (V, E)$ and cost function $F : V \to R$. Find maximum (or minimum) weight closure.
Min Closure = Compliment of Max Closure
Max Closure = Min cut in graph H:

- Add source s and sink t.
- For all vertices $v \in V(G)$:
  - if $f(v) > 0$: add edge $(s, v, f(v))$.
  - if $f(v) < 0$: add edge $(v, t, -f(v))$
- For all edge $(u, v) \in E(G)$: add edge $(u, v, +\infty)$.
- Vertices in same side of s forms a closure.
- Weight(cut) $= \sum f(v)[f(v) > 0] -$ Weight(closure)
- Cut is minimum when closure is maximum.

3.10 Max Flow - Dinic

```
struct Dinic { // Complexity O(V^2 E)
  static const int MXN = 10000;
  struct Edge { int v,f,re; };
```

```
int n, s, t, level[MXN];
vector<Edge> E[MXN];
void init(int _n, int _s, int _t){
  n = _n; s = _s; t = _t;
  for(int i = 0; i < n; i++) E[i].clear();
}
void add_edge(int u, int v, int f){
  E[u].PB({v, f, SZ(E[v])});
  E[v].PB({u, 0, SZ(E[u])-1});
}
bool BFS(){
  for(int i = 0; i < n; i++) level[i] = -1;
  queue<int> que({s});
  level[s] = 0;
  while(!que.empty()){
    int u = que.front(); que.pop();
    for(auto it : E[u]){
      if(it.f > 0 && level[it.v] == -1) {
        level[it.v] = level[u]+1;
        que.push(it.v);
      }
    }
  } return level[t] != -1;
}
int DFS(int u, int nf){
  if(u == t) return nf;
  int res = 0;
  for(auto &it : E[u]){
    if(it.f > 0 && level[it.v] == level[u]+1) {
      int tf = DFS(it.v, min(nf,it.f));
      res += tf; nf -= tf; it.f -= tf;
      E[it.v][it.re].f += tf;
      if(nf == 0) return res;
    }
  }
  if(!res) level[u] = -1;
  return res;
}
int flow(int res = 0){
  while(BFS())
    res += DFS(s,2147483647);
  return res;
}
} flow;
```

3.11 Max Flow with Edge Bound

Without Source or Sink, Check

Set capacity = upper bound − lower bound
Add source s, sink t.
Let $M_v = \sum$ ingoing edge lower bounds $- \sum$ outgoing edge lower bounds.
For all v, if $M_v > 0$, add edge (s, v, M), else add edge $(v, t, -M)$.
If all outgoing edges from S are full $\implies$ feasible flow exists, it is flow + lower bounds.

With Source or Sink, Find Max Flow

Binary search on Max Flow. Add edge (sink, source, Max Flow). Check whether feasible flow exists without source / sink. (Notice that we need to add another 2 source and sink for checking that).

3.12 Minimum Cost Maximum Flow

```
struct CostFlow {
  static const int MXN = 205;
  static const ll INF = 102938475610293847LL;
  struct Edge {
    int v, r;
    ll f, c;
  };
  int n, s, t, prv[MXN], prvL[MXN], inq[MXN];
  ll dis[MXN], fl, cost;
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t) {
    n = _n; s = _s; t = _t;
    for(int i = 0; i < n; i++) E[i].clear();
    fl = cost = 0;
  }
  void add_edge(int u, int v, ll f, ll c) {
    E[u].PB({v, E[v].size() , f, c});
    E[v].PB({u, E[u].size()-1, 0, -c});
  }
  pair<ll, ll> flow() {
    while(true) {
      for(int i = 0; i < n; i++)
        dis[i] = INF, inq[i] = 0;
      dis[s] = 0;
      queue<int> q({s});
      while(!q.empty()) {
        int u = q.front(); q.pop();
        inq[u] = 0;
        for(int i = 0; i < E[u].size(); i++) {
          int v = E[u][i].v;
          ll w = E[u][i].c;
          if(E[u][i].f > 0 && dis[v] > dis[u] + w) {
            prv[v] = u; prvL[v] = i;
            dis[v] = dis[u] + w;
            if(!inq[v]) {
              inq[v] = 1;
              q.push(v);
            }
          }
        }
      }
      if(dis[t] == INF) break;
      ll tf = INF;
      for(int v = t, u, l; v != s; v = u) {
        u = prv[v]; l = prvL[v];
        tf = min(tf, E[u][l].f);
      }
      for(int v = t, u, l; v != s; v =u) {
        u = prv[v]; l = prvL[v];
        E[u][l].f -= tf;
        E[v][E[u][l].r].f += tf;
      }
      cost += tf * dis[t];
      fl += tf;
    }
    return {fl, cost};
  }
} flow;
```

3.13 Stoer Wagner

```
const int inf = 1e9; // larger than Min Cut
const int N = 160;
int cost[N][N], w[N];
bool vis[N], merged[N];
vector<int> bestCut;

int MinCut(int n) {
  int best = inf; merged[0] = 1;
  for(int i = 1; i < n; i++) merged[i] = 0;
  // vector<int> cut;
```

```
  for(int phase = 1; phase < n; ++phase) {
    vis[0] = 1;
    for(int i = 1; i < n; ++i) if(!merged[i])
      vis[i] = 0, w[i] = cost[0][i];

    int prv = 0, nxt;
    for(int i = n - 1 - phase; i >= 0; --i) {
      nxt = -1;
      for(int j = 1; j < n; ++j)
        if(!vis[j] && (nxt == -1 || w[j] > w[nxt]))
          nxt = j;
      vis[nxt] = 1;
      if(i) { prv = nxt;
        for(int j = 1; j < n; ++j)
          if(!vis[j]) w[j] += cost[nxt][j];
      }
    }
    for(int i = 0; i < n; i++)
      cost[i][prv] = (cost[prv][i] += cost[nxt][i]);
    merged[nxt] = 1;
    // cut.push_back(nxt);
    if(best > w[nxt]) {
      best = w[nxt];
      // bestCut = cut;
    }
  } return best;
}
```

3.14 Strongly Connected Components

```
vector<int> adj[maxn], trans[maxn];
int col[maxn], vis[maxn], idx = 0, n, m;
stack<int> st;
void dfs(int u) {
  vis[u] = 1;
  for(int v : adj[u]) if(!vis[v]) dfs(v);
  st.push(u);
}
void dfs2(int u) {
  col[u] = idx;
  for(int v : trans[u]) if(!col[v]) dfs2(v);
}
void scc() {
  for(int i = 1; i <= n; i++) if(!vis[i]) dfs(i);
  for(int u = 1; u <= n; u++)
    for(int v : adj[u])
      trans[v].push_back(u);
  while(!st.empty()) {
    int u = st.top(); st.pop();
    if(col[u]) continue;
    idx++; dfs2(u);
  }
}
```

4 String Processing

4.1 Aho-Corasick

```
const int N = 1e6 + 10;
int trie[N][26], link[N], idx, tot[N];
char p[N], s[N];

void insert() {
  int u = 0, len = strlen(p);
  for(int i = 0; i < len; i++) {
    int &v = trie[u][p[i] - 'a'];
    u = v = v ? v : ++idx;
  } tot[u]++;
}
void bfs() {
  queue<int> q;
  for(q.push(0); !q.empty(); ) {
    int u = q.front(); q.pop();
    for(int c = 0; c < 26; c++) {
      int &v = trie[u][c];
      if(!v) v = trie[link[u]][c];
      else {
        link[v] = u ? trie[link[u]][c] : 0;
        tot[v] += tot[link[v]];
        q.push(v);
      }
    }
  }
}
int match() {
  int u = 0, len = strlen(s);
  int ret = 0;
  for(int i = 0; i < len; i++) {
    u = trie[u][s[i] - 'a'];
    ret += tot[u];
  } return ret;
}
```

4.2 Knuth Morris Pratt

```
const int maxn = 1e5 + 10;
string s, t;
int pi[maxn];

void prefixFn() {
  int now = pi[0] = -1;
  for(int i = 1; i < s.size(); i++) {
    while(now != -1 && s[now + 1] != s[i]) now =
        pi[now];
    if(s[now + 1] == s[i]) pi[i] = ++now;
    else pi[i] = now = -1;
  }
}
int kmp() {
  prefixFn();
  int cnt = 0, now = -1;
  for(int i = 0; i < t.size(); i++) {
    while(now != -1 && s[now + 1] != t[i]) now =
        pi[now];
    if(s[now + 1] == t[i]) ++now;
    else now = -1;
    if(now == s.size()-1) { now = pi[now]; cnt++; }
  } return cnt;
}
```

4.3 Lyndon Factorization

```
// Factorize s into w1 w2 .. wk : k maximum possible
    and w1 >= w2 >= ...
// each wi is Lyndon word (strictly smaller than all
    its rotation)
void lyndon(string s) {
  int n = s.length(), i = 0;
  while(i < n) {
    int j = i + 1, k = i;
    while(j < n && s[k] <= s[j]) {
      if(s[k] < s[j]) k = i;
      else ++k;
      ++j;
    } while(i <= k) {
      cout << s.substr(i, j - k) << ' ';
```

```
      i += j - k;
    }
  } cout << endl;
}
```

4.4 Manacher

```
// l[2 * i] = len of palindrome centered at s[i]
// l[2*i+1] = len of palindrome centered at s[i],
    s[i+1]
vector<int> manacher(string &s) {
  int n = s.size(); vector<int> l(2 * n);
  for(int i = 0, j = 0, k; i < n * 2; i += k, j =
      max(j - k, 0)) {
    while(i >= j && i + j + 1 < n * 2 && s[(i - j)/2]
        == s[(i + j + 1)/2]) ++j;
    l[i] = j;
    for(k = 1; i >= k && j >= k && l[i - k] != j - k;
        ++k)
      l[i + k] = min(l[i - k], j - k);
  } return l;
}
```

4.5 Suffix Array

```
char s[N];
int SA[N], iSA[N], cnt[N], next[N], lcp[N];
bool bh[N], b2h[N];

void buildSA(int n) {
  for(int i = 0; i < n; i++) SA[i] = i;
  sort(SA, SA + n, [](int i, int j) { return s[i] <
      s[j]; });
  for(int i = 0; i < n; i++) {
    bh[i] = i == 0 || s[SA[i]] != s[SA[i - 1]];
    b2h[i] = 0;
  }
  for(int h = 1; h < n; h <<= 1) {
    int tot = 0;
    for(int i = 0, j; i < n; i = j) {
      j = i + 1;
      while(j < n && !bh[j]) j++;
      next[i] = j; tot++;
    } if(tot == n) break;
    for(int i = 0; i < n; i = next[i]) {
      for(int j = i; j < next[i]; j++)
        iSA[SA[j]] = i;
      cnt[i] = 0;
    }
    cnt[iSA[n - h]]++;
    b2h[iSA[n - h]] = 1;
    for(int i = 0; i < n; i = next[i]) {
      for(int j = i; j < next[i]; j++) {
        int s = SA[j] - h;
        if(s < 0) continue;
        int head = iSA[s];
        iSA[s] = head + cnt[head]++;
        b2h[iSA[s]] = 1;
      }
      for(int j = i; j < next[i]; j++) {
        int s = SA[j] - h;
        if(s < 0 || !b2h[iSA[s]]) continue;
        for(int k = iSA[s] + 1; !bh[k] && b2h[k]; k++)
          b2h[k] = 0;
      }
    }
    for(int i = 0; i < n; i++) {
      SA[iSA[i]] = i;
      bh[i] |= b2h[i];
    }
  }
  for(int i = 0; i < n; i++) iSA[SA[i]] = i;
}
void buildLCP(int n) {
  for(int i = 0, k = 0; i < n; i++) {
    if(iSA[i] == n - 1) { k = 0; continue; }
    int j = SA[iSA[i] + 1];
    while(i + k < n && j + k < n && s[i + k] == s[j +
        k]) ++k;
    lcp[iSA[i]] = k;
    if(k) k--;
  }
}
```

4.6 Suffix Automata

```
const int A = 26;
int len[N << 1], link[N << 1], sz, last;
int adj[N << 1][A];

void init() {
  sz = last = 0;
  len[0] = 0; link[0] = -1;
  memset(adj, -1, sizeof adj);
  sz++;
}
void extend(int c) {
  int cur = sz++, p;
  len[cur] = len[last] + 1;
  for(p = last; p != -1 && adj[p][c] == -1; p =
      link[p])
    adj[p][c] = cur;
  last = cur;
  if(p == -1) return void(link[cur] = 0);

  int q = adj[p][c];
  if(len[q] == len[p] + 1)
    return void(link[cur] = q);

  int clone = sz++;
  len[clone] = len[p] + 1;
  link[clone] = link[q];

  for(int i = 0; i < A; i++)
    adj[clone][i] = adj[q][i];
  for(; p != -1 && adj[p][c] == q; p = link[p])
    adj[p][c] = clone;
  link[q] = link[cur] = clone;
}
```

4.7 Z-Algorithm

```
void Zfn(const char *s, int n, int *z) {
  z[0] = 0;
  for(int b = 0, i = 1; i < n; i++) {
    z[i] = max(min(z[i - b], z[b] + b - i), 0);
    while(s[i + z[i]] == s[z[i]]) ++z[i];
    if(i + z[i] > b + z[b]) b = i;
  }
}
```

4.8 Zhou Yuan

```
int minimumExpression(string s) { s = s + s;
  int i = 0, j = 1, k = 0, len = s.size();
  while(i + k < len && j + k < len) {
```

```
    if(s[i + k] == s[j + k]) k++;
    else if(s[i + k] < s[j + k]) {
      j = max(j + k + 1, i + 1); k = 0;
    } else { i = max(i + k + 1, j + 1); k = 0; }
  } return min(i, j);
}
```

5 Miscellaneous

5.1 Dotfiles

```
.vimrc
-------
filetype plugin indent on

sy on | colo morning

set cin nu ai so=10 ts=4 sw=4 sts=4 et noswapfile
set bs=indent,eol,start bg=light

nm <C-s> :up<CR> | im <C-s> <esc><C-s>
im jk <esc> | im kj <esc>

.bashrc
--------
stty -ixon
```