*

0pt0px plus 1px minus 0px-2px plus 0px minus 0px *

0pt0px plus 1px minus 0px0px plus 3px minus 3px
breaklines=true, tabsize=2, breaksymbolleft= perldoc
-

Team Note of Deobureo Minkyu Party

tncks0121, koosaga, alex9801, hyea (alumni)

Compiled on August 20, 2019

Contents

ALL BELOW HERE ARE USELESS IF YOU READ THE STATEMENT WRONG

1.8 Blossom Algorithm for General Matching

cpp const int MAXN = 2020 + 1; // 1-based Vertex index int vis[MAXN], par[MAXN], orig[MAXN], match[MAXN], aux[MAXN], t, N; vector¡int¿ conn[MAXN]; queue¡int¿ Q; void addEdge(int u, int v) conn[u].push_back(v); conn[v].push_back(u); void init(int n) N = n; t = 0; for(int i = 0; i <= n; ++i) conn[i].clear(); match[i] = aux[i] = par[i] = 0; void augment(int u, int v) int pv = v, nv; do pv = par[v]; nv = match[pv]; match[v] = pv; match[pv] = v; v = nv; while(u != pv); int lca(int v, int w) ++t; while(true) if(v != 0) if(aux[v] == t) return v; aux[v] = t; v = orig[par[match[v]]]; swap(v, w); void blossom(int v, int w, int a) while(orig[v] != a) par[v] = w; w = match[v]; if(vis[w] == 1) Q.push_back(w); vis[w] = 0; orig[v] = orig[w] = a; v = par[w]; bool bfs(int u) fill(vis + 1, vis + N + 1, -1); iota(orig + 1, orig + N + 1, 1); Q = queue¡int¿(); Q.push_back(u); vis[u] = 0; while(Q.size()) int v = Q.front(); Q.pop_front(); for(int x : conn[v]) if(vis[x] == -1) par[x] = v; vis[x] = 1; if(!match[x]) for(int lst; x; x = lst) lst = match[v]; match[x] = v; match[v] = x; return true; Q.push_back(x); vis[x] = 0; else if(vis[x] == 0 && orig[v] != orig[x]) int a = lca(orig[v], orig[x]); blossom(x, v, a); blossom(v, x, a); return false; int solve() memset(match + 1, 0, sizeof(int) * N); int matches = 0; for(int i = 1; i <= N; ++i) if(match[i] == 0 && bfs(i)) ++matches; return matches;

2 Graph

2.1 2-SAT

cpp struct TwoSat int N; vector¡bool¿ value; TwoSat(int n = 0) : N(n), value() bool satisfy() vector¡vector¡int¿¿ adj(2 * N); for(int i = 0; i < N; ++i) int a = 2 * i, b = 2 * i + 1; char x = value.size() > i ? value[i] : 2; if(x != 1) adj[a].push_back(b); if(x != 0) adj[b].push_back(a); int n = 2 * N; vector¡int¿ stk; vector¡int¿ pre(n, -1), low(n, -1), comp(n, -1); int Time = 0; function¡void(int)¿ dfs = [&](int u) low[u] = pre[u] = Time++; stk.push_back(u); for(int v : adj[u]) if(pre[v] == -1) dfs(v); low[u] = min(low[u], low[v]); else if(comp[v] == -1) low[u] = min(low[u], pre[v]); if(low[u] == pre[u]) int v; do v = stk.back(); stk.pop_back(); comp[v] = u; while(v != u); ; for(int i = 0; i < n; ++i) dfs(i); value = vector¡bool¿(N); for(int i = 0; i < N; ++i) if(comp[2 * i] == comp[2 * i + 1]) return false; value[i] = comp[2 * i] > comp[2 * i + 1]; return true; ; int main() int n, m; scanf("%d %d", &n, &m); TwoSat ts(n); for(int i = 0; i < m; ++i) int a, b; scanf("%d %d", &a, &b); ts.add_edge(a > 0 ? 2 * a - 2 : -2 * a - 1, b > 0 ? 2 * b - 2 : -2 * b - 1); if(ts.satisfy()) for(int i = 1; i <= n; ++i) printf("%d ", ts.value[i] ? i : -i); puts("");

2.2 BCC

cpp void color(int x, int p) if(p) bcc[p].push_back(x); cmp[x].push_back(p); for(auto i : gph[x]) if(cmp[i].size()) continue; if(low[i] >= dfn[x]) bcc[++c].push_back(x); cmp[x].push_back(c); color(i, c); else color(i, p);

2.3 Splay Tree + Link-Cut Tree

cpp // Checklist 1. Is it link cut, or splay? // Checklist 2. In link cut, is son always root? void rotate(node *x) if(!x->p) return; push(x->p); // if there's lazy stuff push(x); node *p = x->p; bool is_left = (p->l == x); node *b = (is_left ? x->r : x->l); x->p = p->p; if(x->p->l == p) x->p->l = x; if(x->p->r == p) x->p->r = x; if(is_left) if(b) b->p = p; p->l = b; p->p = x; x->r = p; else if(b) b->p = p; p->r = b; p->p = x; void splay(node *x) while(x->p) node *p = x->p; node *g = p->p; if(g) if((p->l == x) ^ (g->l == p)) rotate(x); else rotate(p); rotate(x); void access(node *x) splay(x); push(x); if(x->r) x->r->p = x; x->r = NULL; pull(x); while(x->p) node *pp = x->p->p; splay(x->p); push(x->p); if(pp) x->p->r = x; else x->p = NULL; x->p = pp; x = pp; void link(node *x, node *y) access(x); access(y); x->l = y; y->p = x; pull(x); void cut(node *x) access(x); x->l->p = NULL; x->l = NULL; pull(x); node *root(node *x) access(x); while(x->l) push(x); x = x->l; access(x); return x; node *par(node *x) access(x); if(!x->l) return NULL; push(x); x = x->l; while(x->r) push(x); x = x->r; access(x); return x; node *lca(node *s, node *t) access(s); access(t); splay(s); if(s->p == NULL) return s; return s->p; void link(node *par, node *son) access(par); access(son); son->rev ^= 1; // remove if needed push(son); son->l = par; par->p = son; pull(son); void cut(node *p) access(p); push(p); if(p->l) p->l->p = NULL; p->l = NULL; pull(p);

2.4 Offline Dynamic MST

cpp int n, m, q; int st[MAXN], ed[MAXN], cost[MAXN], chk[MAXN]; pi qr[MAXN];
bool cmp(int a, int b) return pi(cost[a], a) ¡ pi(cost[b], b);
void contract(int s, int e, vector¡int¿ v, vector¡int¿ must_mst, vector < int > maybe_mst) sort(v.begin(), v.end(), cmp); vector < pi > snapshot; for(int i = s; i <= e; i++) disj.uni(st[qr[i].first], ed[qr[i].first]); for(int i = 0; i < v.size(); i++) if(i) snapshot.push_back(pi(cost[v[i]], v[i])); else must_mst.push_back(v[i]); disj.revert(snapshot);
void solve(int s, int e, vector¡int¿ v, lint cv) if(s == e) cost[qr[s].first] = qr[s].second; if(st[qr[s].first] == ed[qr[s].first]) printf("return; int minv = qr[s].second; for(auto i : v) minv = min(minv, cost[i]); printf("return; int m = (s+e)/2; vector¡int¿ lv = v, rv = v; vector¡int¿ must_mst, maybe_mst; for(int i = m + 1; i <= e; i++) chk[qr[i].first] --; if(chk[qr[i].first] == 0) lv.push_back(qr[i].first); vector < pi > snapshot; contract(s, m, lv, must_mst, maybe_mst); lint lcv = cv; for(auto i : must_mst) lcv += cost[i], disj.uni(st[i], ed[i], snapshot); solve(s, m, maybe_mst, lcv); disj.revert(snapshot); must_mst.clear(); maybe_mst.clear(); for(int i = m + 1; i <= e; i++) chk[qr[i].first] ++; for(int i = s; i <= m; i++) chk[qr[i].first] --; if(chk[qr[i].first] == 0) rv.push_back(qr[i].first); lint rcv = cv; contract(m + 1, e, rv, must_mst, maybe_mst); for(auto i : must_mst) rcv += cost[i], disj.uni(st[i], ed[i], snapshot); solve(m + 1, e, maybe_mst, rcv); disj.revert(snapshot); for(int i = s; i <= m; i++) chk[qr[i].first] ++;
int main() scanf("%d", &n); for(int i=0; i<n; i++) scanf("%d %d %d", &st[i], &ed[i], &cost[i]); scanf("%d", &q); for(int i=0; i<q; i++) scanf("%d %d", &qr[i].first, &qr[i].second); chk[qr[i].first]++; disj.init(n); for(int i=0; i<n; i++) if(!chk[i]) ve.push_back(i); solve(0, q - 1, ve, 0);

2.5 Dominator Tree

cpp vector¡int¿ E[MAXN], RE[MAXN], rdom[MAXN];
int S[MAXN], RS[MAXN], cs; int par[MAXN], val[MAXN], sdom[MAXN], rp[MAXN], dom[MAXN];
void clear(int n) cs = 0; for(int i=0; i<=n; i++) par[i] = val[i] = sdom[i] = rp[i] = dom[i] = S[i] = RS[i] = 0; E[i].clear(); RE[i].clear(); rdom[i].clear(); void add_edge(int x, int y) E[x].push_back(y); void Union(int x, int y) par[x] = y; int Find(int x, int c = 0) if(par[x] == x) return c ? -1 : x; int p = Find(par[x], 1); if(p == -1) return c ? par[x] : val[x]; if(sdom[val[x]] > sdom[val[par[x]]]) val[x] = val[par[x]]; par[x] = p; return c ? p : val[x]; void dfs(int x) S[x] = ++cs; RS[cs] = x; sdom[cs] = dom[cs] = val[cs] = cs; rdom[cs].clear(); for(int i = 0; i < E[x].size(); i++) int e = E[x][i]; if(S[e] == 0) dfs(e); RE[S[e]].push_back(S[x]); rp[S[e]] = S[x]; void solve(int s, int *idom) // Calculate idoms dfs(s); for(int i = cs; i; i--) for(int e : RE[i]) sdom[i] = min(sdom[i], sdom[Find(e)]); if(i > 1) rdom[sdom[i]].push_back(i); for(int x : rdom[i]) int p = Find(x); dom[x] = (sdom[p] == i ? i : p); for(int i = 2; i <= cs; i++) if(dom[i] != sdom[i]) dom[i] = dom[dom[i]]; for(int i = 1; i <= cs; i++) idom[RS[i]] = RS[dom[i]];

2.6 Global Min-Cut

cpp int mincut(int n, vector¡vector¡int¿¿ adj, vector < int > vis) vector < int > dist(n); int mincut = 1e9; while(true) int pos = -1, cur = -1e9; for(int i = 0; i < n; i++) if(!vis[i] && cur < dist[i]) pos = i; cur = dist[i]; if(pos == -1) return mincut; vis[pos] = 1; for(int i = 0; i < n; i++) if(!vis[i]) dist[i] += adj[pos][i]; int solve(int n, vector < int > vis, vector < int > adj) if(n <= 1) return 0; vector < int > vis(n); int mincut = 1e9; for(int i = 0; i < n - 1; i++) int ans = mincut(i, vis, adj); mincut = min(mincut, ans); return mincut;

2.7 Edmond's Directed MST

Should be **revised**.cpp // starts from node 0. assumes there exists at least one dmst. // edge is reversed : if there is edge s -¿ e, INSERT IN gph[e] struct edge int to, cost, id; ; using elist = vector¡edge¿; void dmst(vector¡elist¿ g, vector¡int¿ res) const int n = g.size(); vector¡edge*¿ to(n); vector¡int¿ u(n, 0); for (int i = 1; i ¡ n; ++i) int mn = g[i][0].cost; for (int j = 0; j ¡ g[i].size(); ++j) mn = min(mn, g[i][j].cost); for (int j = 0; j ¡ g[i].size(); ++j) if (g[i][j].cost == mn) to[i] = g[i][j]; g[i][j].cost -= mn; for (int i = 1; i ¡ n; ++i) if (u[i]) continue; int x = i; vector¡int¿ order(1, x); u[x] = 1; while (to[x] -¿ to ¡ 0 ? u[to[x]] : to) == 0) x = to[x]; u[x] = 1; order.push_back(x); int y = to[x] - > to; vector < int > cycle(find(order.begin(), order.end(), y), order.end()); if(cycle.size() == 0) continue; vector < int > in_cycle(n); for(int j = 0; j < cycle.size(); ++j) in_cycle[cycle[j]] = 1; vector < int > nw, d[n); for(int i = 1; i < n; ++i) nw.d[j] = nw.d[j] - > pp = node * nxt = x - > pp; splay(nxt); push(nxt); if(nxt - > r) nxt = nxt - > r - > pp = nxt; nxt - > r - > p = NULL; root(node * x) access(x); while(x - > l) push(x); x = x - > l; access(x); return x; node * par(node * x) access(x); if(!x - > l) return NULL; push(x); x = x - > l; while(x - > r) push(x); x = x - > r; access(x); return x; node* lca(node*s, node*t) access(s); access(t); splay(s); if(s - > pp == NULL) return s; return s - > pp; void link(node* par, node*son) access(par); access(son); son - > rev=1; //remove if needed push(son); son - > l = par; par - > p = son; pull(son); void cut(node* p) access(p); push(p); if(p - > l) p - > l - > p = NULL; p - > l = NULL; pull(p);

2.8 Vizing's Theorem

cpp namespace Vizing // returns edge coloring in adjacent matrix G. 1 - based int C[MAXN][MAXN], G[MAXN][MAXN]; void clear(int N) for(int i=0; i¡=N; i++) for(int j=0; j¡=N; j++) C[i][j] = G[i][j] = 0; void solve(vector¡pi¿ E, int N, int M) int X[MAXN] = , a; auto update = [](int u) for(X[u] = 1; C[u][X[u]]; X[u]++); ; auto color = [](int u, int v, int c) int p = C[u][c]; swap(C[u][c], C[v][c]); swap(C[u][c], C[v][c]); if(p) G[u][p] = G[p][u] = c; C[u][c] = v; C[v][c] = u; C[u][p] = C[v][p] = 0; if(p) X[u] = X[v] = p; else update(u), update(v); return p; ; auto flip = [](int u, int c1, int c2) int p = C[u][c1]; swap(C[u][c1], C[u][c2]); if(p) G[u][p] = G[p][u] = c2; if(!C[u][c1]) X[u] = c1; if(!C[u][c2]) X[u] = c2; return p; ; for(int i = 1; i ¡= N; i++) X[i] = 1; for(int t = 0; t ¡ E.size(); t++) int u = E[t].first, v0 = E[t].second, v = v0, c0 = X[u], c = c0, d; vector¡pi¿ L; int vst[MAXN] = ; while(!G[u][v0]) L.emplace_back(v, d = X[v]), vst[v] = 1, c = X[u], v = C[u][c]; for(int a = (int)L.size()-1; a >= 0; a--) c = color(u, L[a].first, c); else if(!C[u][d]) for(int a = (int)L.size() - 1; a >= 0; a --) color(u, L[a].first, L[a].second); else if(vst[d]) break; else vst[d] = 1, v = C[u][d]; if(!G[u][v0]) for(; v; v = flip(v, c, d), swap(c, d)); if(C[u][c0]) for(int a = (int)L.size() - 2; a >= 0 L[a].second

3 Strings

3.1 Aho-Corasick Algorithm

cpp const int MAXN = 100005, MAXC = 26; int trie[MAXN][MAXC], fail[MAXN], term[MAXN], piv; void init(vector¡string¿ v) memset(trie, 0, sizeof(trie)); memset(fail, 0, sizeof(fail)); memset(term, 0, sizeof(term)); piv = 0; for(auto i : v) int p = 0; for(auto j : i) if(!trie[p][j]) trie[p][j] = ++piv; p = trie[p][j]; term[p] = 1; queue¡int¿ que; for(int i=0; i¡MAXC; i++) if(trie[0][i]) que.push(trie[0][i]); while(!que.empty()) int x = que.front(); que.pop(); for(int i=0; i¡MAXC; i++) if(trie[x][i]) int p = fail[x]; while(p !trie[p][i]) p = fail[p]; p = trie[p][i]; fail[trie[x][i]] = p; if(term[p]) term[trie[x][i]] = 1; que.push(trie[x][i]); bool query(string s) int p = 0; for(auto i : s) while(p !trie[p][i]) p = fail[p]; p = trie[p][i]; if(term[p]) return 1; return 0;

3.2 Suffix Array

cpp const int MAXN = 500005; int ord[MAXN], nord[MAXN], cnt[MAXN], aux[MAXN]; void solve(int n, char *str, int *sfx, int *rev, int *lcp) int p = 1; memset(ord, 0, sizeof(ord)); for(int i=0; i¡n; i++) sfx[i] = i; ord[i] = str[i]; int pnt = 1; while(1) memset(cnt, 0, sizeof(cnt)); for(int i=0; i¡n; i++) cnt[ord[min(i+p, n)]]++; for(int i=1; i¡=n —— i¡=255; i++) cnt[i] += cnt[i-1]; for(int i=n-1; i¿=0; i–) aux[–cnt[ord[min(i+p, n)]]] = i; memset(cnt, 0, sizeof(cnt)); for(int i=0; i¡n; i++) cnt[ord[i]]++; for(int i=1; i¡=n —— i¡=255; i++) cnt[i] += cnt[i-1]; for(int i=n-1; i¿=0; i–) sfx[–cnt[ord[aux[i]]]] = aux[i]; if(pnt == n) break; pnt = 1; nord[sfx[0]] = 1; for(int i=1; i¡n; i++) if(ord[sfx[i-1]] != ord[sfx[i]] —— ord[sfx[i-1] + p] != ord[sfx[i] + p]) pnt++; nord[sfx[i]] = pnt; memcpy(ord, nord, sizeof(int) * n); p *= 2; for(int i=0; i¡n; i++) rev[sfx[i]] = i; int h = 0; for(int i=0; i¡n; i++) if(rev[i]) int prv = sfx[rev[i] - 1]; while(str[prv + h] == str[i + h]) h++; lcp[rev[i]] = h; h = max(h-1, 0);

3.3 Manacher's Algorithm

cpp const int MAXN = 1000005; int aux[2 * MAXN - 1]; void solve(int n, int *str, int *ret) // *ret : number of nonobvious palindromic character pair for(int i=0; i¡n; i++) aux[2*i] = str[i]; if(i != n-1) aux[2*i+1] = -1; int p = 0, c = 0; for(int i=0; i¡2*n-1; i++) int cur = 0; if(i ¡= p) cur = min(ret[2 * c - i], p - i); while(i - cur - 1 ¿= 0 i + cur + 1 ¡ 2*n-1 aux[i-cur-1] == aux[i+cur+1]) cur++; ret[i] = cur; if(i + ret[i] ¿ p) p = i + ret[i]; c = i;

3.4 Suffix Array (Linear time)

Should be **revised**.cpp class SuffixArray public: int A[7 * N / 10], B[7 * N / 10], cnt[N + 2], SAV[N]; int mem[5 * N]; int* mem_pt = mem; $void clear(int n) int * ptr = mem; while(ptr != mem_pt) *ptr = 0; ptr++; mem_pt = mem; for(int i = 0; i < n; i++) cnt[i] = 0; for(int i = 0; i < n; i++) cnt[dat[i]]++; for(int i = 1; i < k; i++) cnt[i] += cnt[i-1]; for(int i = n-1; i >= 0; i--) SAV[--cnt[dat[i]]] = i; bool bad(int a, int b) return dat[a] != dat[b] || a+1 >= n || b+1 >= n || dat[a+1] != dat[b+1]; void solve(int *dat, int n, int k) int *sa = mloc(n), *ret = mloc(size_t sz) int * ret = mem_pt; mem_pt = mem_pt + sz; return ret; void rsort(int * a, int * b, int * dat, int n, int k) for(int i = 0; i <= k; i++) cnt[i] = 0; for(int i = 0; i < n; i++) cnt[dat[a[i]]]++; for(int i = 1; i <= k; i++) cnt[i] += cnt[i-1]; for(int i = n-1; i >= 0; i--) b[--cnt[dat[a[i]]]] = a[i];$

define I(x) ((x)define I2(x) (x¡num1)?(3*x+1):(3*(x-num1)+2) static int cmp(int x, int y, int str[], int A[], int num1) if (x if (y else return str[x] ¡ str[y] —— str[x] == str[y] A[I(x + 1)] ¡ A[I(y + 1)]; else return str[x] ¡ str[y] —— str[x] == str[y] cmp(x + 1, y + 1, str, A, num1); void make(int* str, int* sa, int n, int k) if (n == 0) return; int num1 = (n + 2) / 3, num2 = n / 3; int num = num1 + num2; str[n] = str[n + 1] = str[n + 2] = 0; int *nsa = mloc(num), *nstr = mloc(num + 3);

for (int i = 0, j = 0; i ¡ n; i++) if (i if (n rsort(A, B, str + 2, num, k); rsort(B, A, str + 1, num, k); rsort(A, B, str, num, k);

int cnt = 1; nstr[I(B[0])] = 1; for (int i = 1; i ¡ num; i++) int c = B[i], p = B[i - 1]; if (str[p] != str[c] —— str[p + 1] != str[c + 1] —— str[p + 2] != str[c + 2]) cnt++; nstr[I(c)] = cnt; if (cnt == num) for (int i = 0; i ¡ num; i++) nsa[nstr[i] - 1] = i; else make(nstr, nsa, num, cnt);

for (int i = 0, j = 0; i ¡ num; i++) if (nsa[i] ¡ num1) A[j++] = 3 * nsa[i]; rsort(A, B, str, num1, k); for (int i = 0; i ¡ num; i++) A[nsa[i]] = i, nsa[i] = I2(nsa[i]); A[num] = -1; merge(B, B + num1, nsa + (n return cmp(x, y, str, A, num1);); return; sa;

3.5 eertree

cpp int nxt[MAXN][26]; int par[MAXN], len[MAXN], slink[MAXN], ptr[MAXN], diff[MAXN], series[MAXN], piv; void clear(int n = MAXN) memset(par, 0, sizeof(int) * n); memset(len, 0, sizeof(int) * n); memset(slink, 0, sizeof(int) * n); memset(nxt, 0, sizeof(int) * 26 * n); piv = 0; void init(int n, char *a) par[0] = 0; par[1] = 1; a[0] = -1; len[0] = -1; piv = 1; int cur = 1; for(int i=1; i¡=n; i++) while(a[i] != a[i - len[cur] - 1]) cur = slink[cur]; if(!nxt[cur][a[i]]) nxt[cur][a[i]] = ++piv; par[piv] = cur; len[piv] = len[cur] + 2; int lnk = slink[cur]; while(a[i] != a[i - len[lnk] - 1]) lnk = slink[lnk]; if(nxt[lnk][a[i]]) lnk = nxt[lnk][a[i]]; if(len[piv] == 1 —— lnk == 0) lnk = 1; slink[piv] = lnk; diff[piv] = len[piv] - len[lnk]; if(diff[piv] == diff[lnk]) series[piv] = series[lnk]; else series[piv] = piv; cur = nxt[cur][a[i]]; ptr[i] = cur; int query(int s, int e) int pos = ptr[e]; while(len[pos] ¿= e - s + 1) if(len[pos] len[series[pos]] ¡= e - s + 1) return true; pos = series[pos]; pos = slink[pos]; return false; vector¡pi¿ minimum$_partition(int n)//(odd min, even min) vector < pi > dp(n + 1); vector < pi > series_ans(n + 10); dp[0] =$

3.6 Circular LCS

cpp string s1, s2; int dp[4005][2005]; int nxt[4005][2005]; int n, m; void reroot(int px) int py = 1; while(py ¡= m nxt[px][py] != 2) py++; nxt[px][py] = 1; while(px ¡ 2 * n py ¡ m) if(nxt[px+1][py] == 3) px++; nxt[px][py] = 1; else if(nxt[px+1][py+1] == 2) px++; py++; nxt[px][py] = 1; else py++; while(px ¡ 2 * n nxt[px+1][py] == 3) px++; nxt[px][py] = 1;

int track(int x, int y, int e) // use this routine to find LCS as string int ret = 0; while(y != 0 x != e) if(nxt[x][y] == 1) y–; else if(nxt[x][y] == 2) ret += (s1[x] == s2[y]), x–, y–; else if(nxt[x][y] == 3) x–; return ret;

int solve(string a, string b) n = a.size(), m = b.size(); s1 = "" + a + a; s1 = '' + b; for(int i=0; i¡=2*n; i++) for(int j=0; j¡=m; j++) if(j == 0) nxt[i][j] = 3; continue; if(i == 0) nxt[i][j] = 1; continue; dp[i][j] = -1; if(dp[i][j ¡ dp[i][j-1]) dp[i][j] = dp[i][j-1]; nxt[i][j] = 1; if(dp[i][j ¡ dp[i-1][j-1] + (s1[i] == s2[j])) dp[i][j] = dp[i-1][j-1] + (s1[i] == s2[j]); nxt[i][j] = 2; if(dp[i][j ¡ dp[i-1][j]) dp[i][j] = dp[i-1][j]; nxt[i][j] = 3; int ret = dp[n][m]; for(int i=1; i¡n; i++) reroot(i), ret = max(ret, track(n+i, m, i)); return ret;

4 Geometry

4.1 Smallest Enclosing Circle / Sphere

cpp namespace cover$_2$ddoubleeps = 1e − 9; $using Point = complex < double >; struct CirclePointp; double r; do$ namespace cover$_3$ddoubleenclosing$_s$phere$(vector < double > x, vector < double > y, vector < double > z) intn$

4.2 3D Convex Hull

cpp struct vec3 ll x, y, z; vec3(): x(0), y(0), z(0) vec3(ll a, ll b, ll c): x(a), y(b), z(c) vec3 operator*(const vec3 v) const return vec3(y*v.z-z*v.y, z*v.x-x*v.z, x*v.y-y*v.x); vec3 operator-(const vec3 v) const return vec3(x-v.x, y-v.y, z-v.z); ll dot(const vec3 v) const return x*v.x+y*v.y+z*v.z; ;

struct face vec3 norm; ll disc; int I[3]; ;

face make$_face(int i, int j, int k, int i i, vector < vec3 > A)//p^T * norm < disc E[i][j].insert(k); E[i][k].insert(j); E$

vector¡face¿ get$_hull(vector < vec3 > A) int N = A.size(); vector < face > faces; memset(E, -1, sizeof(E)); fa$

4.3 Dynamic Convex Hull Trick

cpp using line$_t$ = $double; const line_t is_query = -1e18;$

struct Line $line_t m, b; mutable function < const Line * () > succ; bool operator < (const Line rhs) const if(rhs.b != is_query) return m < rhs.m; const Line * s = succ(); if(!s) return 0; line_t x = rhs.m$

struct HullDynamic : public multiset¡Line¿ // will maintain upper hull for maximum bool bad(iterator y) auto z = next(y); if (y == begin()) if (z == end()) return 0; re-

turn 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 (x-¿b - y-¿b)*(z-¿m - y-¿m) ¿= (y-¿b - z-¿b)*(y-¿m - x-¿m); void insert$_l ine(line_t m, line_t b) auto y = insert(m, b); y- > succ = [=]return next(y) == end()?0 : *next(y);; if(bad(y)) erase(y); while(next(y) != end() && bad(y)) erase(next(y)); while(y != begin() && prev(y) != end() && bad(y)) erase(p

4.4 Half-plane Intersection

cpp const double eps = 1e-8; typedef pair¡long double, long double¿ pi; bool z(long double x) return fabs(x) ¡ eps; struct line long double a, b, c; bool operator¡(const line l)const bool flag1 = pi(a, b) ¿ pi(0, 0); bool flag2 = pi(l.a, l.b) ¿ pi(0, 0); if(flag1 != flag2) return flag1 ¿ flag2; long double t = ccw(pi(0, 0), pi(a, b), pi(l.a, l.b)); return z(t) ? c * hypot(l.a, l.b) ¡ l.c * hypot(a, b) : t ¿ 0; pi slope() return pi(a, b); ; pi cross(line a, line b) long double det = a.a * b.b - b.a * a.b; return pi((a.c * b.b - a.b * b.c) / det, (a.a * b.c - a.c * b.a) / det); bool bad(line a, line b, line c) if(ccw(pi(0, 0), a.slope(), b.slope()) ¡= 0) return false; pi crs = cross(a, b); return crs.first * c.a + crs.second * c.b ¿= c.c; bool solve(vector¡line¿ v, vector¡pi¿ solution) // ax + by ¡= c; sort(v.begin(), v.end()); deque¡line¿ dq; for(auto i : v) if(!dq.empty() z(ccw(pi(0, 0), dq.back().slope(), i.slope()))) continue; while(dq.size() ¿= 2 bad(dq[dq.size()-2], dq.back(), i)) dq.pop$_b ack(); while(dq.size() >= 2 bad(i, dq[0], dq[1])) dq.pop$_f ront(); dq.push$_b ack(i); while(dq.size() > 2 bad(dq[dq.size() - 2], dq.back(), dq[0])) dq.pop$_b ack(); while(dq.size() > 2 bad(dq.back(), dq[0], dq[1])) dq.pop$_f ront(); vector < pi > tmp; for(int i = 0; i < dq.size(); i++) line cur = dq[i], nxt = dq[(i + 1) if(ccw(pi(0, 0), cur.slope(), nxt.slope()) == 0 && cur.b > nxt.b) return false; tmp.push$_b ack(cross(cur, nxt)); solution = tmp; return true;

4.5 Point-in-polygon test / Point-to-polygon tangent

cpp // C : counter$_c lockwise(C[0] == C[N]), N >= 3 // return highest point in C < -P(clockwise) or -1 if strictly in P // polygon is strongly convex, C[i]! = P int convex$_t angent(vector < pi > C, pi P, int up = 1) auto sign = [](int c) return c > 0?up : c == 0?0 : -up;; auto local = [](pi P, pi a, pi b, pi c) return sign(ccw(P, a, b)) <= 0 && sign(ccw(P, b, c)) >= 0;; int N = C.size() - 1, s = 0, e = N, m; if(local(P, C[1], C[0], C[N - 1])) return 0; i

4.6 kd-tree

cpp typedef pair¡int, int¿ pi; struct node pi pnt; int spl, sx, ex, sy, ey; tree[270000];
pi a[100005]; int n, ok[270000];
lint sqr(int x) return 1ll * x * x; bool cmp1(pi a, pi b) return a ¡ b; bool cmp2(pi a, pi b) return pi(a.second, a.first) ¡ pi(b.second, b.first);
// init(0, n-1, 1) : Initialize kd-tree // set dap = INF, and call solve(1, P). dap = (closest point from P) void init(int s, int e, int p) // Initialize kd-tree int minx = 1e9, maxx = -1e9, miny = 1e9, maxy = -1e9; int m = (s+e)/2; for(int i=s; i¡=e; i++) minx = min(minx, a[i].first); miny = min(miny, a[i].second); maxx = max(maxx, a[i].first); maxy = max(maxy, a[i].second); tree[p].spl = (maxx - minx ¡ maxy - miny); sort(a+s, a+e+1, [](const pi a, const pi b) return tree[p].spl ? cmp2(a, b) : cmp1(a, b);); ok[p] = 1; tree[p] = a[m], tree[p].spl, minx, maxx, miny, maxy; if(s ¡= m-1) init(s, m-1, 2*p); if(m+1 ¡= e) init(m+1, e, 2*p+1);
lint dap = 3e18;
void solve(int p, pi x) // find closest point from point x (L^2) if(x! = tree[p].pnt) dap = min(dap, sqr(x.first - tree[p].pnt.first) + sqr(x.second - tree[p].pnt.second)); if(tree[p].spl) if(!cmp2(tree[p].pnt, x)) if(ok[2 * p]) solve(2 * p, x); if(dap[2 * p + 1] sqr(tree[2 * p + 1].sy - x.second) < dap) solve(2 * p + 1, x); else if(ok[2 * p + 1]) solve(2 * p + 1, x); if(ok[2 * p] sqr(tree[2 *

5 Math

5.1 FFT / NTT

cpp typedef complex¡double¿ base; void fft(vector¡base¿ a, bool inv) int n = a.size(), j = 0; vector¡base¿ roots(n/2); for(int i=1; i¡n; i++) int bit = (n ¿¿ 1); while(j ¿= bit) j -= bit; bit ¿¿= 1; j += bit; if(i ¡ j) swap(a[i], a[j]); double ang = 2 * acos(-1) / n * (inv ? -1 : 1); for(int i=0; i¡n/2; i++) roots[i] = base(cos(ang * i), sin(ang * i)); /* In NTT, let prr = primitive root. Then, int ang = ipow(prr, (mod - 1) / n); if(inv) ang = ipow(ang, mod - 2); for(int i=0; i¡n/2; i++) roots[i] = (i ? (1ll * roots[i-1] * ang XOR Convolution : set roots[*] = 1. OR Convolution : set roots[*] = 1, and do following: if (!inv) a[j + k] = u + v; a[j + k + i/2] = u; else a[j + k] = v; a[j + k + i/2] = u - v; */ for(int i=2; i¡=n; i¡¡=1) int step = n / i; for(int j=0; j¡n; j+=i) for(int k=0; k¡i/2; k++) base u = a[j+k], v = a[j+k+i/2] * roots[step * k]; a[j+k] = u+v; a[j+k+i/2] = u-v; if(inv) for(int i=0; i¡n; i++) a[i] /= n; // skip for OR convolution.
cpp include ¡smmintrin.h¿ include ¡immintrin.h¿ pragma GCC target("avx2") pragma GCC target("fma") cpp

5.2 Hell-Joseon style FFT

cpp include ¡smmintrin.h¿ include ¡immintrin.h¿ pragma GCC target("avx2") pragma GCC target("fma") cpp

5.3 NTT Polynomial Division

cpp vector¡lint¿ get$_n v(int n, const vector < lint > p) vector < lint > q = ipow(p[0], mod - 2); for(int i = 2; i <= lint > poly$_d ivide(const vector < lint > a, const vector < lint > b) assert(b.back()! = 0); // please trim leading zero int n = a.size(), m = b.size(); int k = 2; while(k < n - m + 1) k <

5.4 Black Box Linear Algebra + Kitamasa

cpp vector¡int¿ berlekamp$_m assey(vector < int > x) vector < int > ls, cur; int lf, ld; for(int i = 0; i < x.size(); i int > rec, vector < int > dp, int n) int m = rec.size(); vector < int > s(m), t(m); s[0] = 1; if(m! = 1) t[1] = 1; else int > x, int n) if(n < x.size()) return x[n]; vector < int > v = berlekamp$_m assey(x); if(v.empty()) return 0; retur −v, 0 − based. no duplicate please..vector < int > get$_m in$_p oly(int n, vector < elem > M) // smallest poly P such that A^i = sum$_{j<i}$ A^j × P$_j$ vector < int > rnd1, rnd2; mt19937 rng(0x14004); autorandin elem > M) vector < int > rnd; mt19937 rng(0x14004); autorandint = [rng](int lb, int ub) return uniform$_i nt_d istri

5.5 Gaussian Elimination

cpp int n, inv; vector¡int¿ basis[505]; lint gyesu = 1;
void insert(vector¡int¿ v) for(int i=0; i¡n; i++) if(basis[i].size()) inv =1; // inversion numincreasesif(v[i] basis[i].empty()) basis[i] = v; return; if(v[i]) lint minv = ipow(basis[i][i], mod
// Sample: Calculates Determinant in Z$_p$ Field int main() scanf("%d", &n); for(int i = 0; i < n; i++) vector < int > v(n);

5.6 Simplex Algorithm

cpp using T = long double; const int N = 410, M = 30010; const T eps = 1e-7; int n, m; int Left[M], Down[N]; // time complexity: exponential. fast $O(MN^2)$ in experiment. dependent on the modeling. // Ax ¡= b, max c^{T}x. : v, : sol[i].1 based Ta[M][N], b[M], c[N], v, sol[N]; bool eq(Ta, Tb) return fabs(a − b) < eps; bool ls(Ta, Tb) return a < b && !eq(
for(int i = 1; i ¡= m; i++) if(i == x —— eq(a[i][y], 0)) continue; k = a[i][y]; a[i][y] = 0; b[i] -= k*b[x]; for(int j : nz) a[i][j] -= k*a[x][j]; if(eq(c[y], 0)) return; k = c[y]; c[y] = 0; v += k*b[x]; for(int i : nz) c[i] -= k*a[x][i]; // 0: found solution, 1: no feasible solution, 2: unbounded int solve() for(int i = 1; i ¡= n; i++) Down[i] = i; for(int i = 1; i ¡= m; i++) Left[i] = n+i; while(1) // Eliminating negative b[i] int x = 0, y = 0; for(int i = 1; i ¡= m; i++) if (ls(b[i], 0) (x == 0 —— b[i] ¡ b[x])) x = i; if(x == 0) break; for(int i = 1; i ¡= n; i++) if (ls(a[x][i], 0) (y == 0 —— a[x][i] ¡ a[x][y])) y = i; if(y == 0) return 1; pivot(x, y);

while(1) int x = 0, y = 0; for(int i = 1; i ¡= n; i++) if (ls(0, c[i]) (!y —— c[i] ¿ c[y])) y = i; if(y == 0) break; for(int i = 1; i ¡= m; i++) if (ls(0, a[i][y]) (!x —— b[i]/a[i][y] ¡ b[x]/a[x][y])) x = i; if(x == 0) return 2; pivot(x, y); for(int i = 1; i ¡= m; i++) if(Left[i] ¡= n) sol[Left[i]] = b[i]; return 0;

5.7 Pentagonal Number Theorem for Partition Number Counting

cpp vector¡pair¡int, int¿¿ gp; lint P[MAXN+1] = ; gp.emplace$_back(0,0); for(inti = 1; gp.back().second <= MAXN; i++) gp.emplace_back(igp.emplace_back(iP[1] = 1; for(intn = 2; n <= MAXN; n++) for(autoit : gp) if(n >= it.second) P[n] += P[n - it.second] * it.first + MOD; P[n]$

5.8 De Bruijn Sequence

cpp // Create cyclic string of length k$^n thatcontainseverylengthnstringassubstring.alphabet = [0, k - 1] intres[10000000]; // >= k^n intaux[10000000]; // >= k * nintde_bruijn(intk, intn) // Returnssize(k^n) if(k == 1) res[0] = 0; return1; for(inti = 0; i < k*n; i++) aux[i] = 0; intsz = 0; function < void(int, int) > db = [](intt, intp) if(t > n) if(n for(inti = 1; i <= p; i++) res[sz + -$

5.9 Discrete Kth root

cpp /* * Solve x for x$^P = AmodQ * (P, Q - 1) = 1 -> P^-1 mod(Q - 1) exists * xhassolutioniffA^((Q - 1)/P) = 1modQ * PP|(Q - 1) -> P < sqrt(Q), solvelgQroundsofdiscretelog * else -> findas.t.s|(Pa - 1) -> ans = A^a * /usingLL = longlong; LLmul(LLx, LLy, LLmod) return(_int128) x*y LLadd(LLx,LLy,LLmod) return(x+y) LLpw(LLx,LLy,LLmod) LLret=1, piv=x; while(y) if(y1) ret=mul(ret,piv,mod); piv=mul(piv,piv,mod); y>>=1; returnret voidgcd(LLa,LLb,LLx,LLy,$

LL dis$_log(LLx) FOR(i, X) LLiaXi = iaXe[i]; LLrst = mul(x, iaXi, Q); if(ht.count(rst)) returni * X + ht[rst]; LLmain2() cin >> P >> A >> Q; LLt = 0, s = Q - 1; while(s + +t, s/ = P; if(A == 0) return0; if(t == 0) // a^{P^-1 mod phi(Q)} LLx, y,; gcd(P, Q - 1, x, y,); if(x < 0) x = (xLLans = pw(A, x, Q); if(pw(ans, P, Q)! = A) while(1); returnans; // AisnotP - residueif(pw(A, (Q - 1)/P, Q)! = 1) return - 1; for(g = 2; g < Q; ++g) if(pw(g, (Q - 1)/P, Q)! = 1) break; LLalpha = 0; LLy,; gcd(P, s, alpha, y,); if(alpha < 0) alpha = (alphaif(t == 1) LLans = pw(A, alpha, Q); returnans; LLa = pw(g, (Q - 1)/P, Q); build(a); LLb = pw(A, add(mul(PLLc = pw(g, s, Q); LLh = 1; LLe = (Q - 1)/s/P; // r^{t-1} REP(i, 1, t - 1) e/ = P; LLd = pw(b, e, Q); LLj = 0; if(d! = 1) j = -dis_log(d); if(j < 0) j = (jb = mul(b, pw(c, mul(Ph = mul(h, pw(c, j, Q), Q); c = pw(c, P, Q); returnmul(pw(A, alpha, Q), h, Q);$

5.10 Miller-Rabin Test + Pollard Rho Factorization

cpp namespace miller$_rabinlintmul(lintx, linty, lintmod) return(_int128) x*y lintipow(lintx,linty,lintp) lintret=1, piv=x while(y) if(y1) ret=mul(ret,piv,p); piv=mul(piv,piv,p); y>>=1; returnret; boolmiller_rabin(lintx,linta) if(xlintd=x-1; wh$

namespace pollard$_rholintf(lintx, lintn, lintc) return(c + miller_rabin :: mul(x, x, n)) voidrec(lintn, vector < lint > v) if(n == 1) return; if(n v.push_back(2); rec(n/2, v); return; if(miller_rabin :: isprime(n)) v.push_back(n); lint > factorize(lintn) vector < lint > ret; rec(n, ret); sort(ret.begin(), ret.end()); returnret;;$

5.11 Highly Composite Numbers, Large Prime

cpp ¡ 10$^k numberdivisors$ 2357111317192329313 7 -
164112601221138403231114756064331158316012833111672072024042111178648640448631111873513440768531111197351344001344632111110698377680023045321111119777287520040326322111112963761198400672064211 1
¡ 10$^k primeofprime < 10^k prime$ -
174109999999996729725119999999999773997168129999999999989499731229139999999999997159999195921499999999999999736999983784981599999999999999897999999166457916999999999999999937899999989576145517999999999999999

NTT Prime:

$998244353 = 119 \times 2^{23} + 1$. Primitive root: 3.
$985661441 = 235 \times 2^{22} + 1$. Primitive root: 3.
$1012924417 = 483 \times 2^{21} + 1$. Primitive root: 5.

6 Miscellaneous

6.1 Mathematics

- **Tutte Matrix**. For a simple undirected graph G, Let M be a matrix with entries $A_{i,j} = 0$ if $(i,j) \notin E$ and $A_{i,j} = -A_{j,i} = X$ if $(i,j) \in E$. X could be any random value. If the determinants are non-zero, then a perfect matching exists, while other direction might not hold for very small probability.
- **Cayley's Formula**. Given a degree sequence $d_1, d_2 \cdots, d_n$ for each labeled vertices, there exists $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$ spanning trees. Summing this for every possible degree sequence gives n^{n-2}.
- **Kirchhoff's Theorem**. For a multigraph G with no loops, define Laplacian matrix as $L = D - A$. D is a diagonal matrix with $D_{i,i} = deg(i)$, and A is an adjacency matrix. If you remove any row and column of L, the determinant gives a number of spanning trees.
- **Green's Theorem**. Let C is positive, smooth, simple curve. D is region bounded by C.
$\oint_C (Ldx + Mdy) = \iint_D (\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y})$

To calculate area, $\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y} = 1$, common selection is $M = \frac{1}{2}x$, $L = -\frac{1}{2}y$.

Line integral of circle parametrized by $(x,y) = (x_C + r_C \cos\theta, y_C + r_C \sin\theta)$ when $\theta = t\theta_i + (1-t)\theta_f$, is given as follows.: $\frac{1}{2}(r_C(x_C(\sin\theta_f - \sin\theta_i) - y_C(\cos\theta_f - \cos\theta_i)) + (\theta_f - \theta_i)r_C^2)$.

Line integral of line parametrized by $(x,y) = t(x_1,y_1) + (1-t)(x_2,y_2)$ is given as follows.: $\frac{1}{2}(x_1y_2 - x_2y_1)$.

- **Burnside's lemma / Pólya enumeration theorem**. let G and H be groups of permutations of finite sets X and Y. Let $c_m(g)$ denote the number of cycles of length m in $g \in G$ when permuting X. The number of colorings of X into $|Y| = n$ colors with exactly r_i occurrences of the i-th color is the coefficient of $w_1^{r_1} \dots w_n^{r_n}$ in the following polynomial:
$P(w_1, \dots, w_n) = \frac{1}{|H|}\sum_{h \in H} \frac{1}{|G|}\sum_{g \in G} \prod_{m \ge 1} (\sum_{h^m(b)=b} (w_b^m))^{c_m(g)}$

When $H = \{I\}$ (No color permutation):
$P(w_1, \dots, w_n) = \frac{1}{|G|}\sum_{g \in G} \prod_{m \ge 1} (w_1^m + \dots + w_n^m)^{c_m(g)}$

Without the occurrence restriction:
$P(1, \dots, 1) = \frac{1}{|G|}\sum_{g \in G} n^{c(g)}$
where $c(g)$ could also be interpreted as the number of elements in X that are fixed up to g.

- **Pick's Theorem**. $A = i + \frac{b}{2} - 1$, where: P is a simple polygon whose vertices are grid points, A is area of P, i is # of grid points in the interior of P, and b is # of grid points on the boundary of P. If h is # of holes of P ($h+1$ simple closed curves in total), $A = i + \frac{b}{2} + h - 1$. cpp // number of (x, y) : (0 ¡= x ¡ n 0 ¡ y ¡= k/d x + b/d) // argument should be positive ll count$_solve(lln, llk, llb, lld) if(k == 0) return(b/d) * n; if(k >= d || b >= d) return((k/d) * (n - 1) + 2 * (b/d))$
- **Xudyh Sieve**. $F(n) = \sum_{d|n} f(d)$
$S(n) = \sum_{i \le n} f(i) = \sum_{i \le n} F(i) - \sum_{d=2}^{n} S(\lfloor \frac{n}{d} \rfloor)$
Preprocess $S(1)$ to $S(M)$ (Set $M = n^{\frac{2}{3}}$ for complexity)
$S(n) = \sum f(i) = \sum_{i \le n} \left[F(i) - \sum_{j|i, j \ne i} f(j)\right] = \sum F(i) - \sum_{i/j=d=2}^{n} \sum_{dj \le n} f(j)$
$S(n) = \sum if(i) = \sum_{i \le n} i \left[F(i) - \sum_{j|i, j \ne i} f(j)\right] = \sum iF(i) - \sum_{i/j=d=2}^{n} \sum_{dj \le n} djf(j)$
$\sum_{d|n} \varphi(d) = n$ $\sum_{d|n} \mu(d) =$ if $(n > 1)$ then 0 else 1 $\sum_{d|n} (\mu(\frac{n}{d}) \sum_{e|d} f(e)) = f(n)$

6.2 Popular Optimization Technique

- CHT. DnC optimization. Mo's ~~algorithm~~ trick (on tree). IOI 2016 Aliens trick. IOI 2009 Regions trick.

- Knuth's $O(n^2)$ Optimal BST : minimize $D_{i,j} = Min_{i \le k < j}(D_{i,k} + D_{k+1,j}) + C_{i,j}$. Quadrangle Inequality : $C_{a,c} + C_{b,d} \le C_{a,d} + C_{b,c}$, $C_{b,c} \le C_{a,d}$. Now monotonicity holds.
- Sqrt batch processing - Save queries in buffer, and update in every sqrt steps (cf : IOI 2011 Elephant. hyea calls it "ainta technique")
- Dynamic insertion in static set (Make $O(\log n)$ copy. Merge like binomial heap.)
- Offline insertion / deletion in insert-only set (Pair insertion-deletion operation, and regard it as range query)
- Atcoder Median Pyramid : Reduce the input to binary, and solve the easier problem.
- LP Duality. max $c^T x$ sjt to $Ax \le b$. Dual problem is min $b^T x$ sjt to $A^T x \ge c$. By strong duality, min max value coincides.

6.3 Fast LL Division / Modulo

cpp inline void fasterLLDivMod(unsigned long long x, unsigned y, unsigned out$_d$, $unsigned out_m) unsigned xh = (unsigned)(x >> 32), xl = (unsigned)x, d, m; if def_{GNUC}{}_{asm("divl:"=a"(d),"=d"(m):"d"(xh),"a"(xl),"1"(y));out_m=m;out_d=d;}elsedivmod(x,y,out_d,out_m);$

6.4 Bit Twiddling Hack

cpp int $_{builtin_clz(intx);//numberofleadingzeroint_{builtin_ctz(intx);//numberoftrailingzeroint_{builtin_clzll(longlongx);//numberofleadingzeroint_{builtin_ctzll(longlongx);//numberoftrailingzeroint_{builtin_popcount(intx);//numberof1-bitsinxint_{builtin_parity(intx);//1ifnumberof1-bitsinxiseven}}}}}}$

lsb(n): (n -n); // last bit (smallest) floor(log2(n)): 31 - $_{builtin_clz(n|1);floor(log2(n)):63-_{builtin_clzll(n|1);}}$

// compute next perm. ex) 00111, 01011, 01101, 01110, 10011, 10101.. long long next$_perm(longlongv) longlongt = v|(v-1); return(t+1)|(((t- t)-1) >> ({}_{builtin_ctz(v)+1}));$

6.5 Fast Integer IO

cpp static char buf[1 ¡¡ 19]; // size : any number geq than 1024 static int idx = 0; static int bytes = 0; static inline int $_read() if(!bytes||idx == bytes) bytes = (int)fread(buf, sizeof(buf[0]), sizeof(buf), stdin); idx = 0; return buf[idx++]; static inline int readInt() int x = 0, s = 1; int c =_r ead(); while(c <= 32) c =_r ead(); if(c ==' -') s = -1,$

6.6 OSRank in g++

cpp include ¡ext/pb$_ds$/$assoc_container.hpp$ > $include$ < $ext/pb_ds/tree_policy.hpp$ > $using namespace_{gnu_pbds};$

typedef tree¡int, null$_type, less < int >, rb_tree_tag, tree_order_statistics_node_update > ordered_set$;

ordered$_set X; X.insert(1); X.insert(2); X.insert(4); X.insert(8); X.insert(16);$

cout¡¡*X.find$_by_order(1)$ << $endl; //2 cout << *X.find_by_order(2)$ << $endl; //4 cout << *X.find_by_order(4)$ << $endl; //16 cout << (end(X) == X.find_by_order(6))$ << $endl; //true$

cout¡¡X.order$_of_key(-5)$ << $endl; //0 cout << X.order_of_key(1)$ << $endl; //0 cout << X.order_of_key(3)$ << $endl; //2 cout << X.order_of_key(4)$ << $endl; //2 cout << X.order_of_key(400)$ << $endl; //5$

6.7 Nasty Stack Hacks

cpp // 64bit ver. int main2() return 0; int main() size$_t sz = 1 <<$ 29; //512MB $void * newstack = malloc(sz); void * sp_dest = newstack + sz - sizeof(void*); asm_{volatile}("movq" "movq" "movq main2(); asm_{volatile}("pop return 0;$

6.8 C++ / Environment Overview

cpp // vimrc : set nu sc ci si ai sw=4 ts=4 bs=2 mouse=a syntax on

// compile : g++ -o PROB PROB.cpp -std=c++11 -Wall -O2 // options : -fsanitize=address -Wfatal-errors

struct StupidGCCCantEvenCompileThisSimpleCode pair¡int, int¿ array[1000000]; ; // https://gcc.gnu.org/bugzilla/show$_bug.cgi?id = 68203$

// how to use rand (in 2018) mt19937 rng(0x14004); int randint(int lb, int ub) return uniform$_int_distribution < int > (lb, ub)(rng);$

// comparator overload auto cmp = [](seg a, seg b) return a.func() ¡ b.func(); ; set¡seg, decltype(cmp)¿ s(cmp); map¡seg, int, decltype(cmp)¿ mp(cmp); priority$_queue < seg, vector < seg >, decltype(cmp) > pq(cmp); //max heap$

// hash func overload struct point int x, y; bool operator==(const point p)const return x == p.x y == p.y; ; struct hasher size$_t operator()(const point p)const return p.x * 2 + p.y * 3;; unordered_map < point, int, hasher > hsh;$

6.9 Credits

[noitemsep,nolistsep]cki86201, zigui, PavelKunyavskiy https://gist.github.com/msg555/4963794 https://github.com/niklasb/contest-algos/blob/master/convex_hull/dynamic.cpp https://github.com/jaehyunp/stanfordacm https://github.com/tzupengwang/PECaveros/blob/master/codebook/graph/BorrowedGeneralWeightedMatching.cpp https://github.com/tzupengwang/PECaveros/blob/master/codebook/math/DiscreteKthsqrt.cpp http://www-math.mit.edu/~etingof/groups.pdf