ACM-template 2018 NanJing University - Floydini

Contents

1. Data Structure

1.1. BIT.

```
int a[N];
void add(int x, int v) {
	for(; x <= n; x += x & -x) a[x] += v;
}
int get(int x) {
	int ans = 0;
	for(; x; x -= x & -x) ans += a[x];
	return ans;
}
```

1.2. Segtree.

```
int siz, u[2 * N];
void init(int n) { // [0, n)
	siz = n;
	fill(u, u + 2 * siz, 0);
}
void put(int p, int v) { // [p] = v
	for(p += siz, u[p] += v; p > 1; p >>= 1) u[p >> 1] += v;
}
int get(int l, int r) { // [l, r)
	int res = 0;
	for(l += siz, r += siz; l < r; l >>= 1, r >>= 1) {
		if(l & 1) res += u[l++];
		if(r & 1) res += u[--r];
	}
	return res;
}
```

1.3. Segtree-rec.

```
const int INF = 1 << 30;

int v[N], c[N]; // value, flag
void init(int n) {
	n <<= 2;
	fill(v, v + n, 0);
	fill(c, c + n, 0);
}
void up(int p) {
	v[p] = min(v[p << 1], v[p << 1 | 1]);
}
void down(int p) {
	if(c[p] == 0) return;
	v[p << 1] += c[p]; v[p << 1 | 1] += c[p];
	c[p << 1] += c[p]; c[p << 1 | 1] += c[p];
	c[p] = 0;
}
void put(int p, int l, int r, int x, int d) { // [x] = d
	if(l != r - 1) down(p);
	else {
		v[p] = d;
		return;
	}
	int mid = (l + r) / 2;
	if(x < mid) put(p << 1, l, mid, x, d);
	else put(p << 1 | 1, mid, r, x, d);
	up(p);
}
void segadd(int p, int l, int r, int x, int y, int d) { // [x,y) += d
	if(l != r - 1) down(p);
	if(x <= l && r <= y) {
		v[p] += d;
		c[p] += d;
		return;
	}
	int mid = (l + r) / 2;
	if(x < mid) segadd(p << 1, l, mid, x, y, d);
	if(y > mid) segadd(p << 1 | 1, mid, r, x, y, d);
	up(p);
}
int get(int p, int l, int r, int x, int y) {
	if(l != r - 1) down(p);
	if(x <= l && r <= y) return v[p];
	int a = INF, b = INF;
	int mid = (l + r) / 2;
	if(x < mid) a = get(p << 1, l, mid, x, y);
	if(y > mid) b = get(p << 1 | 1, mid, r, x, y);
	return min(a, b);
}
```

1.4. splay.

```
int p[N], key[N], ch[N][2], root, tot;
int sz[N], f[N]; // subtree size, flip lazy flag.

void up(int x) {
	sz[x] = sz[ch[x][0]] + sz[ch[x][1]];
}

inline void flip(x) { f[x] ^= 1;}

void down(int x) {
	if(f[x] == 0) return;
	if(ch[x][0]) flip(ch[x][0]);
	if(ch[x][1]) flip(ch[x][1]);
	swap(ch[x][0], ch[x][1]);
	f[x] = 0;
}

void newnode(int &x, int fa, int k) {
	x = ++tot;
	p[x] = fa;
```

```
      key[x] = k;
      ch[x][0] = ch[x][1] = 0;
// sz[x]=1;
// f[x]=0;
}

void rise(int x) {
      int y = p[x], c = ch[y][0] == x;
      down(y), down(x);
      ch[y][!c] = ch[x][c];
      p[ch[x][c]] = y;
      if(p[y]) ch[p[y]][ch[p[y]][1] == y] = x;
      p[x] = p[y];
      ch[x][c] = y;
      p[y] = x;
      up(y), up(x);
}

void splay(int x, int goal) { // make p[x]=goal
      for(int y; (y = p[x]) != goal; rise(x))
            if(p[y] != goal) rise(ch[y][ch[p[y]][0] == y] == x ? x : y);
      if(goal == 0) root = x;
}

void insert(int k) { // first node: newnode(root,0,k);
      int x = root;
      while(ch[x][key[x] < k]) x = ch[x][key[x] < k];
      newnode(ch[x][key[x] < k], x, k);
      splay(ch[x][key[x] < k], 0);
}

int kth(int x, int k) {
      while(k) {
            if(sz[ch[x][0]] >= k) x = ch[x][0];
            else if(sz[ch[x][0]] + 1 < k) {
                  x = ch[x][1];
                  k -= sz[ch[x][0]] + 1;
            } else break;
      }
      splay(x, 0);
      return x;
}

int pre() {
      int x = ch[root][0];
      while(ch[x][1]) x = ch[x][1];
      return x;
}

int suc() {
      int x = ch[root][1];
      while(ch[x][0]) x = ch[x][0];
      return x;
}
```

```
}
```

2. Graph

2.1. bipartite match.

```
const int N = 1e3 + 5;
int V;
vector<int> G[N];
int match[N];
bool used[N];

bool dfs(int v) {
      used[v] = true;
      for(auto u : G[v]) {
            int w = match[u];
            if(w < 0 || !used[w] && dfs(w)) {
                  match[v] = u;
                  match[u] = v;
                  return true;
            }
      }
      return false;
}
int bipartite_matching() {
      int res = 0;
      fill(match, match + V, -1);
      for(int v = 0; v < V; v++) {
            if(match[v] < 0) {
                  fill(used, used + V, 0);
                  if(dfs(v)) res++;
            }
      }
      return res;
}
```

2.2. centroid decomposition.

```
#define MAXN 100005

struct CentroidDecomposition {
      vector<int> L[MAXN];
      int subsize[MAXN], cpar[MAXN];

      void dfs(int cur, int p) {
            subsize[cur] = 1;

            for(int i = 0; i < L[cur].size(); ++i) {
                  int to = L[cur][i];
                  if(to != p && cpar[to] == -1) {
                        dfs(to, cur);
                        subsize[cur] += subsize[to];
                  }
```

```
            }
      }

      void centroid_decomposition(int cur, int p, int n, int prevc) {
            for(int i = 0; i < L[cur].size(); ++i) {
                  int to = L[cur][i];

                  if(to != p && cpar[to] == -1 && 2 * subsize[to] > n) {
                        centroid_decomposition(to, cur, n, prevc);
                        return;
                  }
            }

            cpar[cur] = prevc;

            for(int i = 0; i < L[cur].size(); ++i) {
                  int to = L[cur][i];

                  if(cpar[to] == -1) {
                        dfs(to, -1);
                        centroid_decomposition(to, cur, subsize[to], cur);
                  }
            }
      }

      void init(int start) {
            memset(cpar, -1, sizeof cpar);
            dfs(start, -1);
            centroid_decomposition(start, -1, subsize[start], -2);
      }
};
```

2.3. dijkstra.

```
#include<bits/stdc++.h>
using namespace std;
using P = pair<int, int>;
const int N = 1e6;
const int INF = 0x3f3f3f3f;
struct edge {int to, w;};
vector<edge> G[N];
int d[N], V;

void dij(int s) {
      priority_queue<P, vector<P>, greater<P> > q;
      fill(d, d + V, INF);
      d[s] = 0;
      q.push(P{s, 0});
      while(!q.empty()) {
            P t = q.top(); q.pop();
            int v = t.first;
            if(d[v] < t.second) continue;
            for(auto e : G[v]) {
                  int to = e.to, w = e.w;
                  if(d[to] > d[v] + w) {
                        d[to] = d[v] + w;
                        q.push(P{to, d[to]});
                  }
            }
      }
}
```

2.4. dsu.

```
//O(nlgn)
//for(int i = 0; i < n; i++) par[i] = i;
int par[N];
void findp(int x) { return par[x] == x ? x : par[x] = find(par[x]); }
void merge(int x, int y) {
      x = findp(x), y = findp(y);
      if(x == y) return;
      par[y] = x;
}


//full version
//for(int i = 0; i < n; i++) par[i] = -1;
int par[N];
void findp(int x) { return par[x] < 0 ? x : par[x] = find(par[x]); }
void merge(int x, int y) {
      x = findp(x), y = findp(y);
      if(x == y) return;
      if(par[x] > par[y]) swap(x, y);
      par[x] += par[y];
      par[y] = x;
}
```

2.5. Hopcroft Karp.

```
const int N = 5e4 + 5;

int n1, n2;
vector<int> G[N];
int mx[N], my[N];
queue<int> q;
int dx[N], dy[N];
bool vis[N];
bool find(int u) {
      for(auto v : G[u]) {
            if(!vis[v] && dy[v] == dx[u] + 1) {
                  vis[v] = true;
                  if(!my[v] || find(my[v])) {
                        mx[u] = v;
                        my[v] = u;
                        return true;
                  }
```

```
            }
      }
      return false;
}
int matching() {
      memset(mx, 0, sizeof(mx));
      memset(my, 0, sizeof(my));
      int ans = 0;
      while(true) {
            bool flag = false;
            while(!q.empty()) q.pop();
            memset(dx, 0, sizeof(dx));
            memset(dy, 0, sizeof(dy));
            for(int i = 1; i <= n1; i++)
                  if(!mx[i]) q.push(i);
            while(!q.empty()) {
                  int u = q.front();
                  q.pop();
                  for(auto v : G[u])
                        if(!dy[v]) {
                              dy[v] = dx[u] + 1;
                              if(my[v]) {
                                    dx[my[v]] = dy[v] + 1;
                                    q.push(my[v]);
                              } else flag = true;
                        }
            }
            if(!flag) break;
            memset(vis, 0, sizeof(vis));
            for(int i = 1; i <= n1; i++)
                  if(!mx[i] && find(i)) ans++;
      }
      return ans;
}
```

2.6. kruskal.

```
#include<bits/stdc++.h>
using namespace std;
using LL = long long;

int pa[N];
void init(int n) { for(int i = 0; i <= n; i++) pa[i] = i; }
void findpa(int x) {return pa[x] == x ? x : pa[x] = findpa(pa[x]);}
int unite(int a, int b) {
      a = findpa(a), b = findpa(b);
      if(a == b) return -1;
      return pa[a] = b;
}

struct edge {int u, v, w;};
bool cmp(edge &a, edge&b) {return a.w < b.w;}
edge e[N];
int V;

int ST() { //min spanning tree
      sort(e, e + V, cmp);
      int ans = 0;
      for(int i = 0; i < V; i++) {
            if(unite(e.u, e.v) == -1) continue;
            ans += e.w;
      }
      return ans;
}
```

2.7. lca.cpp.

```
#define MAX_N 100000
#define LOG2_MAXN 16

// NOTA : memset(parent,-1,sizeof(parent));
int N, parent[MAX_N], L[MAX_N];
int P[MAX_N][LOG2_MAXN + 1];

int get_level(int u) {
      if(L[u] != -1) return L[u];
      else if(parent[u] == -1) return 0;
      return 1 + get_level(parent[u]);
}

void init() {
      memset(L, -1, sizeof(L));
      for(int i = 0; i < N; ++i) L[i] = get_level(i);

      memset(P, -1, sizeof(P));

      for(int i = 0; i < N; ++i) P[i][0] = parent[i];

      for(int j = 1; (1 << j) < N; ++j)
            for(int i = 0; i < N; ++i)
                  if(P[i][j - 1] != -1)
                        P[i][j] = P[P[i][j - 1]][j - 1];
}

int LCA(int p, int q) {
      if(L[p] < L[q]) swap(p, q);

      int log = 1;
      while((1 << log) <= L[p]) ++log;
      --log;

      for(int i = log; i >= 0; --i)
            if(L[p] - (1 << i) >= L[q])
                  p = P[p][i];

      if(p == q) return p;
```

```
	for(int i = log; i >= 0; --i) {
		if(P[p][i] != -1 && P[p][i] != P[q][i]) {
			p = P[p][i];
			q = P[q][i];
		}
	}

	return parent[p];
}
```

2.8. tarjan.

```
void Tarjan(int x) {
   dfn[x] = ++dfs_num ;
   low[x] = dfs_num ;
   vis[x] = true ; //in stack
   stack[++top] = x ;
   for(int i = head[x]; i != 0; i = e[i].next) {
      int temp = e[i].to ;
      if(!dfn[temp]) {
         Tarjan(temp);
         low[x] = gmin(low[x], low[temp]);
      } else if(vis[temp]) low[x] = gmin(low[x], dfn[temp]);
   }
   if(dfn[x] == low[x]) { //strong component
      vis[x] = false;
      color[x] = ++col_num; //color
      while(stack[top] != x) { //clear
         color[stack[top]] = col_num;
         vis[stack[top--]] = false;
      }
      top--;
   }
}
```

3. Flow

3.1. Dinic.

```
const int INF = 0x3f3f3f3f;

struct Edge {
	int from, to, cap, flow, index;
	Edge(int from, int to, int cap, int flow, int index):
		from(from), to(to), cap(cap), flow(flow), index(index) {}
};

struct Dinic {
	int N;
	vector<vector<Edge> > G;
	vector<Edge *> dad;
	vector<int> Q;

	Dinic(int N): N(N), G(N), dad(N), Q(N) {}

	void AddEdge(int from, int to, int cap) {
		G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
		G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
	}
	int BlockingFlow(int s, int t) {
		fill(dad.begin(), dad.end(), (Edge*)NULL);
		dad[s] = &G[0][0] - 1;

		int head = 0, tail = 0;
		Q[tail++] = s;
		while(head < tail) {
			int x = Q[head++];
			for(int i = 0; i < G[x].size(); i++) {
				Edge &e = G[x][i];
				if(!dad[e.to] && e.cap - e.flow > 0) {
					dad[e.to] = &G[x][i];
					Q[tail++] = e.to;
				}
			}
		}
		if(!dad[t]) return 0;

		int totflow = 0;
		for(int i = 0; i < G[t].size(); i++) {
			Edge *start = &G[G[t][i].to][G[t][i].index];
			int amt = INF;
			for(Edge *e = start; amt && e != dad[s]; e = dad[e->from]) {
				if(!e) {amt = 0; break;}
				amt = min(amt, e->cap - e->flow);
			}
			if(amt == 0) continue;
			for(Edge *e = start; amt && e != dad[s]; e = dad[e->from]) {
				e->flow += amt;
				G[e->to][e->index].flow -= amt;
			}
			totflow += amt;
		}
		return totflow;
	}

	int GetFlow(int s, int t) {
		int totflow = 0;
		while(int flow = BlockingFlow(s, t))
			totflow += flow;
		return totflow;
	}
};
```

3.2. Mincostflow.

```
const int N = 1000 + 5;
const int INF = 0x3f3f3f3f;
using P = pair<int, int>;
struct edge {int to, cap, cost, rev;};
int d[N], h[N], pv[N], pe[N], V; // V should be set
vector<edge> G[N];

void add_edge(int from, int to, int cap, int cost) {
      G[from].push_back((edge) {to, cap, cost, (int)G[to].size()});
      G[to].push_back((edge) {from, 0, -cost, (int)G[from].size() - 1});
}

int min_cost_flow(int s, int t, int flow, int &cost) { // return used flow
      int f = 0;
      cost = 0;
      fill(h, h + V, 0);
      while(f < flow) {
            priority_queue<P, vector<P>, greater<P> >q;
            fill(d, d + V, INF);
            d[s] = 0;
            q.push(P(0, s));
            while(!q.empty()) {
                  P p = q.top(); q.pop();
                  int v = p.second;
                  if(d[v] < p.first) continue;
                  for(int i = 0; i < (int)G[v].size(); i++) {
                        edge &e = G[v][i];
                        if(e.cap > 0 && d[e.to] > d[v] + e.cost + h[v] - h[e.to]) {
                              d[e.to] = d[v] + e.cost + h[v] - h[e.to];
                              pv[e.to] = v;
                              pe[e.to] = i;
                              q.push(P(d[e.to], e.to));
                        }
                  }
            }
            if(d[t] == INF) return f;
            for(int v = 0; v < V; v++) h[v] += d[v];
            int inc = flow - f;
            for(int v = t; v != s; v = pv[v]) {
                  inc = min(inc, G[pv[v]][pe[v]].cap);
            }
            f += inc;
            cost += inc * h[t];
            for(int v = t; v != s; v = pv[v]) {
                  edge &e = G[pv[v]][pe[v]];
                  e.cap -= inc;
                  G[v][e.rev].cap += inc;
            }
      }
      return f;
}
```

4. Math

4.1. euler sieve.

```
bool flag[MAXN];
int p[MAXN];
int sieve_euler(int n) {
      int t = 0;
      for(int i = 2; i <= n; i++) {
            if(!used[i]) p[t++] = i;
            for(int j = 0; p[j]*i <= n; j++) {
                  used[p[j]*i] = true;
                  if(i % p[j] == 0) break;
            }
      }
      return t;
}
```

4.2. fft.

```
const double PI = acos(-1.0);
struct Cp {
      double r, i;
      Cp() {}
      Cp(double x, double y): r(x), i(y) {}
};
Cp operator + (Cp &x, Cp &y) { return Cp(x.r + y.r, x.i + y.i); }
Cp operator - (Cp &x, Cp &y) { return Cp(x.r - y.r, x.i - y.i); }
Cp operator * (Cp &x, Cp &y) { return Cp(x.r * y.r - x.i * y.i, x.r * y.i + x.i * y.r); }

void change(Cp y[], int len) {
      int t = len >> 1;
      for(int i = 1, j = t, k; i < len - 1; i++) {
            if(i < j) swap(y[i], y[j]);
            for(k = t; j >= k; k >>= 1) j -= k;
            j += k;
      }
}

void fft(Cp y[], int len, int on) {
      change(y, len);
      for(int i = 2; i <= len; i <<= 1) {
            double ang = 2.0 * on * PI / i;
            Cp wn = Cp(cos(ang), sin(ang));
            for(int j = 0; j < len; j += i) {
                  Cp w = Cp(1.0, 0.0);
                  for(int k = j; k < j + i / 2; k++) {
                        Cp a = y[k];
                        Cp b = y[k + i / 2] * w;
                        y[k] = a + b;
                        y[k + i / 2] = a - b;
                        w = w * wn;
                  }
            }
```

```
      }
      if(on == -1) for(int i = 0; i < len; i++) y[i].r = y[i].r / len;
}

// len = 2*\lceil n \rceil
// the result is in y1
void sol(Cp y1[], Cp y2[], int len) {
      fft(y1, len, 1);
      if(y1 != y2) fft(y2, len, 1);
      for(int i = 0; i < len; i++) y1[i] = y1[i] * y2[i];
      fft(y1, len, -1);
}
```

4.3. gauss.

```
const double eps = 1e-8;
double a[MAXN][MAXN];
void Gauss(int n, int m) {
      int r, c, k;
      for(r = c = 0; r < n && c < m; r++, c++) {
            for(k = r; k < n; k++) if(fabs(a[k][c]) > eps) break;
            if(r == n) continue;
            if(k != r) for(int j = 0; j <= m; j++) swap(a[k][j], a[r][j]);
            for(int j = c + 1; j <= m; j++) a[r][j] /= a[r][c];
            a[r][c] = 1.0;
            for(int i = 0; i < n; i++) {
                  if(i == r || fabs(a[i][c]) < eps) continue;
                  for(int j = c + 1; j <= m; j++) a[i][j] -= a[i][c] * a[r][j];
                  a[i][c] = 0.0;
            }
      }
}
```

4.4. prime test.

```
bool ptest(LL x, LL n) {
   LL i = n - 1, ans = 1;
   while(i) {
      if(i & 1) ans = (ans * x) % n;
      if((x * x) % n == 1 && x != 1 && x != n - 1) return false;
      x = (x * x) % n;
      i = i >> 1;
   }
   if(ans == 1) return true;
   return false;
}
```

5. String

5.1. kmp.

```
// S,T start from 0
// border[0,n] = [0,f[n])
int f[N];
void kmp(char *S, char *T) {
      int a = strlen(S), b = strlen(T);
      f[0] = 0;
      for(int i = 1, j = 0; i < b; i++) {
            while(j > 0 && T[i] != T[j]) j = f[j - 1];
            if(T[i] == T[j]) ++j;
            f[i] = j;
      }

      for(int i = 0, j = 0; i < a; i++) {
            while(j > 0 && S[i] != T[j]) j = f[j - 1];
            if(S[i] == T[j]) ++j;
            if(j == b) j = f[j - 1]; // S match T at i-b+1
      }
}
```

5.2. extkmp.

```
// S,T start from 0
// f[i]: T[i,f[i]) match T[0,f[i]-i)
// g[i]: S[i,g[i]) match T[0,g[i]-i)
int f[N], g[N];
void extkmp(char *S, char *T) {
      int a = strlen(S), b = strlen(T);
      f[0] = 0;
      for(int i = 1, l = 1, r = 1; i < b; i++) {
            if(f[i - l] >= r - i) {
                  l = i, r = max(l, r);
                  while(r < b && T[r] == T[r - i]) r++;
                  f[i] = r - l;
            } else f[i] = f[i - l];
      }

      for(int &i = g[0] = 0; i < a && i < b && S[i] == T[i]; i++);
      for(int i = 1, l = 0, r = g[0]; i < a; i++) {
            if(f[i - l] >= r - i) {
                  l = i, r = max(l, r);
                  while(r < b && S[r] == T[r - i]) r++;
                  g[i] = r - l;
            } else g[i] = f[i - l];
      }
}
```

5.3. manacher.

```
void manacher(int n, const char s[], int p[]) {
      for(int i = 0, j = 0, k = 0; i <= 2 * (n - 1); ++i) {
            int l = i < k ? min(p[j + j - i], (k - i) / 2) : 0;
            int a = i / 2 - l, b = (i + 1) / 2 + l;
```

```
            while(0 <= a && b < n && s[a] == s[b]) --a, ++b, ++l;
            p[i] = l;

            if(k < 2 * b) {
                  j = i;
                  k = 2 * b;
            }
      }
}
```

5.4. mininum rotation.

```
int min_rotation(char *s) {
      int N = strlen(s), ans = 0, p = 1, len = 0;

      while(p < N && ans + len + 1 < N) {
            if(s[ans + len] == s[(p + len) % N]) ++len;
            else if(s[ans + len] < s[(p + len) % N]) {
                  p = p + len + 1;
                  len = 0;
            } else {
                  ans = max(ans + len + 1, p);
                  p = ans + 1;
                  len = 0;
            }
      }
      return ans;
}
```

5.5. sam.

```
const int N = 250000 + 5;
const int NN = N << 1;
int go[NN][26], len[NN], fa[NN], tot = 1, root = 1, last = 1;
void app(int x) {
      int p = last, s = last = ++tot; len[s] = len[p] + 1;
      for(; p && !go[p][x]; p = fa[p]) go[p][x] = s;
      if(p) {
            int q = go[p][x];
            if(len[q] != len[p] + 1) {
                  int nq = ++tot; len[nq] = len[p] + 1; fa[nq] = fa[q];
                  memcpy(go[nq], go[q], sizeof(go[q]));
                  fa[q] = fa[s] = nq;
                  for(; p && go[p][x] == q; p = fa[p]) go[p][x] = nq;
            } else fa[s] = q;
      } else fa[s] = root;
}
```

5.6. suffix array.

```
// O(nlogn)
// note: string a[] start from 1

int sa[N], rk[N], r[N], h[N], c[N], a[N], n, m;

int cmp(int *f, int x, int y, int w) { return f[x] == f[y] && f[x + w] == f[y + w]; }

void rsort() {
      for(int i = 0; i <= m; i++) c[i] = 0;
      for(int i = 1; i <= n; i++) c[rk[r[i]]]++;
      for(int i = 1; i <= m; i++) c[i] += c[i - 1];
      for(int i = n; i >= 1; i--) sa[c[rk[r[i]]]--] = r[i];
}
void suffix() {
      //SA
      for(int i = 1; i <= n; i++) rk[i] = a[i], r[i] = i;
      rsort();
      for(int w = 1, p = 1, i; p < n; w <<= 1, m = p) {
            for(p = 0, i = n - w + 1; i <= n; i++) r[++p] = i;
            for(i = 1; i <= n; i++) if(sa[i] > w) r[++p] = sa[i] - w;

            rsort(), swap(rk, r), rk[sa[1]] = p = 1;
            for(i = 2; i <= n; i++) rk[sa[i]] = cmp(r, sa[i], sa[i - 1], w) ? p : ++p;
      }
      //height
      int j, k = 0;
      for(int i = 1; i <= n; h[rk[i++]] = k)
            for(k = k ? k - 1 : k, j = sa[rk[i] - 1]; a[i + k] == a[j + k]; ++k);
}
```

6. Geometry

6.1. geometry.

```
#include <bits/stdc++.h>
using namespace std;

const double EPS = 1e-10;
const double PI = acos(-1);

struct Point {
      double x, y;
      Point(double x = 0, double y = 0) : x(x), y(y) {}
};

typedef Point Vector;

Vector operator+(Vector A, Vector B) { return Vector(A.x + B.x, A.y + B.y); }
Vector operator-(Vector A, Vector B) { return Vector(A.x - B.x, A.y - B.y); }
Vector operator*(Vector A, double p) { return Vector(A.x * p, A.y * p); }
Vector operator/(Vector A, double p) { return Vector(A.x / p, A.y / p); }
bool operator<(const Point &a, const Point &b) { return a.x < b.x || (a.x == b.x && a.y < b.

int dcmp(double x) {
      if(fabs(x) < EPS) return 0;
      else return x < 0 ? -1 : 1;
}
```

```
bool operator == (const Point &a, const Point &b) { return dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y); }

//a
double Angle(const Vector& v) { return atan2(v.y, v.x); }

//
double Dot(Vector A, Vector B) { return A.x * B.x + A.y * B.y; }

//
double Length(Vector A) { return sqrt(Dot(A, A)); }

//
double Angle(Vector A, Vector B) { return acos(Dot(A, B) / Length(A) / Length(B)); }

//
double Cross(Vector A, Vector B) { return A.x * B.y - A.y * B.x; }

//
double Area2(Point A, Point B, Point C) { return Cross(B - A, C - A); }

//rad()
Vector Rotate(Vector A, double rad) {
      return Vector(A.x * cos(rad) - A.y * sin(rad), A.x * sin(rad) + A.y * cos(rad));
}

//A90A
Vector Normal(Vector A) {
      double L = Length(A);
      return Vector(-A.y / L, A.x / L);
}

/*********************************************************************

#include <complex>
typedef complex<double> Point;
typedef Point Vector;
double Dot(Vector A, Vector B) { return real(conj(A)*B)}
double Cross(Vector A, Vector B) { return imag(conj(A)*B);}
Vector Rotate(Vector A, double rad) { return A*exp(Point(0, rad)); }
*********************************************************************/

/*********************************************************************
* p0vPP = P0+t*v;
* B-A, A+(B-A)*t;
* t
* t > 0
* 0 < t < 1
*********************************************************************/

//,
Point GetLineIntersection(Point P, Vector v, Point Q, Vector w) {
      Vector u = P - Q;
      double t = Cross(w, u) / Cross(v, w);
      return P + v * t;
}

//
double DistanceToLine(Point P, Point A, Point B) {
      Vector v1 = B - A, v2 = P - A;
      return fabs(Cross(v1, v2) / Length(v1)); //
}

//
double DistanceToSegmentS(Point P, Point A, Point B) {
      if(A == B) return Length(P - A);
      Vector v1 = B - A, v2 = P - A, v3 = P - B;
      if(dcmp(Dot(v1, v2)) < 0) return Length(v2);
      else if(dcmp(Dot(v1, v3)) > 0) return Length(v3);
      else return fabs(Cross(v1, v2)) / Length(v1);
}

//
Point GetLineProjection(Point P, Point A, Point B) {
      Vector v = B - A;
      return A + v * (Dot(v, P - A) / Dot(v, v));
}

//
bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2) {
      double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2 - a1, b2 - a1);
      double c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2 - b1, a2 - b1);
      return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
}

//
bool OnSegment(Point P, Point a1, Point a2) {
      return dcmp(Cross(a1 - P, a2 - P) == 0 && dcmp((Dot(a1 - P, a2 - P)) < 0));
}

//
double ConvexPolygonArea(Point *p, int n) {
      double area = 0;
      for(int i = 1; i < n - 1; i++)
            area += Cross(p[i] - p[0], p[i + 1] - p[0]);
      return area / 2;
}

//
double PolygonArea(Point *p, int n) {
      double area = 0;
      for(int i = 1; i < n - 1; i++)
            area += Cross(p[i] - p[0], p[i + 1] - p[0]);
      return area / 2;
}
```

```
/*************************************************************************
* Morley
* V,E,FV+F-E = 2;
*************************************************************************/

struct Circle {
	Point c;
	double r;

	Circle(Point c, double r) : c(c), r(r) {}
	//
	Point point(double a) {
		return Point(c.x + cos(a) * r, c.y + sin(a) * r);
	}
};

struct Line {
	Point p;
	Vector v;
	double ang;
	Line() {}
	Line(Point p, Vector v) : p(p), v(v) {}
	bool operator < (const Line& L) const { return ang < L.ang; }
};

//sol
//sol
int getLineCircleIntersecion(Line L, Circle C, double& t1, double& t2, vector<Point>& sol) {
	double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;
	double e = a * a + c * c, f = 2 * (a * b + c * d), g = b * b + d * d - C.r * C.r;
	double delta = f * f - 4 * e * g;
	if(dcmp(delta) < 0) return 0; //
	if(dcmp(delta) == 0) { //
		t1 = t2 = -f / (2 * e);
		sol.push_back(C.point(t1));
		return 1;
	}
	//
	t1 = (-f - sqrt(delta)) / (2 * e); sol.push_back(C.point(t1));
	t2 = (-f + sqrt(delta)) / (2 * e); sol.push_back(C.point(t2));
	return 2;
}

//
int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point>& sol) {
	double d = Length(C1.c - C2.c);
	if(dcmp(d) == 0) {
		if(dcmp(C1.r - C2.r == 0)) return -1; //
		return 0; //
	}
	if(dcmp(C1.r + C2.r - d) < 0) return 0;
	if(dcmp(fabs(C1.r - C2.r) == 0)) return -1;
	double a = Angle(C2.c - C1.c); //C1C2
	double da = acos((C1.r * C1.r + d * d - C2.r * C2.r) / (2 * C1.r * d));
	//C1C2C1P1
	Point p1 = C1.point(a - da), p2 = C1.point(a + da);
	sol.push_back(p1);
	if(p1 == p2) return 1;
	sol.push_back(p2);
	return 2;
}

//
//pCv[i]i
int getTangents(Point p, Circle C, Vector* v) {
	Vector u = C.c - p;
	double dist = Length(u);
	if(dist < C.r) return 0;
	else if(dcmp(dist - C.r) == 0) {
		v[0] = Rotate(u, PI / 2);
		return 1;
	} else {
		double ang = asin(C.r / dist);
		v[0] = Rotate(u, -ang);
		v[1] = Rotate(u, +ang);
		return 2;
	}
}

//
//-1
//a[i], b[i] iAB
int getTangents(Circle A, Circle B, Point *a, Point *b) {
	int cnt = 0;
	if(A.r < B.r) {
		swap(A, B); swap(a, b);
	}
	int d2 = (A.c.x - B.c.x) * (A.c.x - B.c.x) + (A.c.y - B.c.y) * (A.c.y - B.c.y);
	int rdiff = A.r - B.r;
	int rsum = A.r + B.r;
	if(d2 < rdiff * rdiff) return 0; //
	double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
	if(d2 == 0 && A.r == B.r) return -1; //
	if(d2 == rdiff * rdiff) { //
		a[cnt] = A.point(base);
		b[cnt] = B.point(base);
		cnt++;
		return 1;
	}
	//
	double ang = acos((A.r - B.r) / sqrt(d2));
	a[cnt] = A.point(base + ang); b[cnt] = B.point(base + ang); cnt++;
	a[cnt] = A.point(base - ang); b[cnt] = B.point(base - ang); cnt++;
	if(d2 == rsum * rsum) { //
		a[cnt] = A.point(base);
```

```
            b[cnt] = B.point(PI + base);
            cnt++;
      } else if(d2 > rsum * rsum) { //
            double ang = acos((A.r + B.r) / sqrt(d2));
            a[cnt] = A.point(base + ang); b[cnt] = B.point(PI + base + ang); cnt++;
            a[cnt] = A.point(base - ang); b[cnt] = B.point(PI + base - ang); cnt++;
      }
      return cnt;
}

typedef vector<Point> Polygon;

//
int isPointInPolygon(Point p, Polygon poly) {
      int wn = 0;
      int n = poly.size();
      for(int i = 0; i < n; i++) {
            if(OnSegment(p, poly[i], poly[(i + 1) % n])) return -1; //
            int k = dcmp(Cross(poly[(i + 1) % n] - poly[i], p - poly[i]));
            int d1 = dcmp(poly[i].y - p.y);
            int d2 = dcmp(poly[(i + 1) % n].y - p.y);
            if(k > 0 && d1 <= 0 && d2 > 0) wn++;
            if(k < 0 && d2 <= 0 && d1 > 0) wn++;
      }
      if(wn != 0) return 1; //
      return 0; //
}

//
/*****************************************************************
* p p ch
* <= <
* dcmp
*
*****************************************************************/
int ConvexHull(Point *p, int n, Point* ch) {
      sort(p, p + n); //xy
      int m = 0;
      for(int i = 0; i < n; i++) {
            while(m > 1 && Cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) <= 0) m--;
            ch[m++] = p[i];
      }
      int k = m;
      for(int i = n - 2; i >= 0; i++) {
            while(m > k && Cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) <= 0) m--;
            ch[m++] = p[i];
      }
      if(n > 1) m--;
      return m;
}

//A->Bpoly
//O(n2);
```

```
Polygon CutPolygon(Polygon poly, Point A, Point B) {
      Polygon newpoly;
      int n = poly.size();
      for(int i = 0; i < n; i++) {
            Point C = poly[i];
            Point D = poly[(i + 1) % n];
            if(dcmp(Cross(B - A, C - A)) >= 0) newpoly.push_back(C);
            if(dcmp(Cross(B - A, C - D)) != 0) {
                  Point ip = GetLineIntersection(A, B - A, C, D - C);
                  if(OnSegment(ip, C, D)) newpoly.push_back(ip);
            }
      }
      return newpoly;
}

//

//pL
bool Onleft(Line L, Point p) {
      return Cross(L.v, p - L.p) > 0;
}

//
Point GetIntersection(Line a, Line b) {
      Vector u = a.p - b.p;
      double t = Cross(b.v, u) / Cross(a.v, b.v);
      return a.p + a.v * t;
}

int HalfplaneIntersection(Line* L, int n, Point* poly) {
      sort(L, L + n); //

      int first, last; //
      Point *p = new Point[n]; //p[i]q[i]q[i+1]
      Line *q = new Line[n]; //
      q[first = last = 0] = L[0]; //L[0]
      for(int i = 0; i < n; i++) {
            while(first < last && !Onleft(L[i], p[last - 1])) last--;
            while(first < last && !Onleft(L[i], p[first])) first++;
            q[++last] = L[i];
            if(fabs(Cross(q[last].v, q[last - 1].v)) < EPS) {
                  last--;
                  if(Onleft(q[last], L[i].p)) q[last] = L[i];
            }
            if(first < last) p[last - 1] = GetIntersection(q[last - 1], q[last]);
      }
      while(first < last && !Onleft(q[first], p[last - 1])) last--;
      //
      if(last - first <= 1) return 0; //
      p[last] = GetIntersection(q[last], q[first]);

      //deque
      int m = 0;
```

```
    for(int i = first; i <= last; i++) poly[m++] = p[i];
    return m;
}
```

7. MATH FACTS

7.1. Catalan number.

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

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}$$

7.2. Stirling number of first class.

$$\begin{bmatrix} n \\ k \end{bmatrix} = (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$$

7.3. Stirling number of second class.

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = k\begin{Bmatrix} n-1 \\ k \end{Bmatrix} + \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix}$$

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

7.4. Bell number.

$$\mathcal{B}_{n+1} = \sum_{k=0}^{n}\binom{n}{k}\mathcal{B}_k$$

x	0	1	2	3	4	5	6	7	8	9	10
$\mathcal{B}_x$	1	1	2	5	15	52	203	877	4'140	21'147	115'975

7.5. Derangement.

$$!n = (n-1)(!(n-1)+!(n-2)); !1 = 0, !2 = 1$$

$$!n = n!\sum_{k=0}^{n}\frac{(-1)^k}{k!}$$

7.6. Harmonic series.

$$H_n = \sum_{k=1}^{n}\frac{1}{k}$$

$$\frac{1}{2n+1} < H_n - \ln n - \gamma < \frac{1}{2n}$$

$$\gamma = 0.57721566490153286060651209008240243104215933 5\ldots$$

7.7. Fibonacci sequence. $f_0 = 0$, $f_1 = 1$:

$$f_n = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^n$$

$$f_{n+1}^2 + f_n^2 = f_{2n+1}, f_{n+2}^2 - f_n^2 = f_{2n+2}$$

$$f_n = \sum_{j=0}^{\lfloor \frac{n}{2} \rfloor}\binom{n-j}{j}$$

$$\gcd(f_n, f_m) = f_{\gcd(n,m)}$$

7.8. combination.

$$\sum_{i=n}^{m}\binom{i}{n} = \binom{m+1}{n+1}$$

$$\sum_{i=0}^{k}\binom{n}{i}\binom{m}{k-i} = \binom{n+m}{k}$$

7.9. Lucas Theorem. p is a prime number.

$n = (n_k n_{k-1} \ldots n_0)_p$, $m = (m_k m_{k-1} \ldots m_0)_p$

$$\binom{m}{n} \equiv \prod_{i=0}^{k}\binom{m_i}{n_i} \mod p$$

7.10. **Generating functions.**

$(1,1,1,1,1,1,\ldots)$	$\frac{1}{1-z}$
$(1,-1,1,-1,1,-1,\ldots)$	$\frac{1}{1+z}$
$(1,0,1,0,1,0,\ldots)$	$\frac{1}{1-z^2}$
$(1,\binom{m+1}{m},\binom{m+2}{m},\binom{m+3}{m},\ldots)$	$\frac{1}{(1-z)^{m+1}}$
$(1,c,\binom{c+1}{2},\binom{c+2}{3},\ldots)$	$\frac{1}{(1-z)^c}$
$(1,c,c^2,c^3,\ldots)$	$\frac{1}{1-cz}$
$(0,1,\frac{1}{2},\frac{1}{3},\frac{1}{4},\ldots)$	$\ln\frac{1}{1-z}$

$$\frac{1}{1-z}G(z)=\sum_n\sum_{k\le n}g_kz^n$$

7.11. **The twelvefold way.** function $f\colon N\to X$

N	X	Any f	Injective	Surjective
dist.	dist.	x^n	$(x)_n$	$x!\left\{{n \atop x}\right\}$
indist.	dist.	$\binom{x+n-1}{n}$	$\binom{x}{n}$	$\binom{n-1}{n-x}$
dist.	indist.	$\left\{{n \atop 1}\right\}+\ldots+\left\{{n \atop x}\right\}$	$[n\le x]$	$\left\{{n \atop k}\right\}$
indist.	indist.	$p_1(n)+\ldots p_x(n)$	$[n\le x]$	$p_x(n)$

Where $\binom{a}{b}=\frac{1}{b!}(a)_b$ and $p_x(n)$ is the number of ways to partition the integer n using x summands.

7.12. **Burnside's Lemma.** G is a permutation group of set X, $S_x=\{g\in G: g*x=x\}$, $Fix(g)=\{x\in X: g*x=x\}$. The number of equivalence classes in X:

$$N=\frac{1}{|G|}\sum_{x\in X}|S_x|=\frac{1}{|G|}\sum_{g\in G}|Fix(g)|$$

7.13. **Power reduction.**

$$a^b\equiv a^{b\%\varphi(p)+\varphi(p)}\mod p\quad(b>\varphi(p))$$

7.14. **Fast Walsh-Hadamard Transform.** $A=(A_0,A_1)$

op.	$F(A)$	$iF(A)$
and	$(F(A_0+A_1),F(A_1))$	$(iF(A_0-A_1),iF(A_1))$
or	$(F(A_0),F(A_0+A_1))$	$(iF(A_0),iF(A_1-A_0))$
xor	$(F(A_0+A_1),F(A_0-A_1))$	$(iF(\frac{A_0+A_1}{2}),iF(\frac{A_0-A_1}{2}))$