

## ButterRoti ICPC Team Notebook (2018)

## Contents

<b>1 Misc</b>	
1.1 Build	1
1.2 Snippet	1
1.3 Stack Size Increase	1
1.4 Variadic Multiplication and Addition	2
<b>2 Combinatorial optimization</b>	2
2.1 Lowest Common Ancestor	2
2.2 Heavy-Light Decomposition	2
2.3 Auxiliary Tree	3
2.4 Articulation Point and Bridges	3
2.5 Biconnected Components	4
2.6 2-SAT	5
2.7 Dinic's Max Flow	5
2.8 Min Cost Max Flow	6
2.9 Global Min Cut	7
2.10 Bipartite Matching	8
2.11 Hopcraft-Karp	8
2.12 Hungarian	9
2.13 Link	10
<b>3 Data Structures</b>	11
3.1 Implicit Treap	11
3.2 Segment Tree	12
3.3 Lazy Propagation	13
<b>4 Math</b>	14
4.1 Extended Euclid	14
4.2 Fast Fourier Transform	15
4.3 Number Theoretic Transform	16
4.4 Large Factorial	17
4.5 Large Modulo Multiplication	17
4.6 Segmented Sieve	17
4.7 Random Number Generator	18
<b>5 Strings</b>	18
5.1 Aho Corasick	18
5.2 Suffix Array	18
5.3 Suffix Tree	19
5.4 Z Algorithm	20
<b>6 Geometry</b>	20
6.1 Geometry Library	20
6.2 Convex Hull	21
<b>7 Formulas</b>	22
7.1 The Twelffold Way	22
7.2 DP Optimizations	22
7.3 Some primes	23
7.4 Markov Chains	24
7.5 Burnside's Lemma	25
7.6 Bezout's identity	25
7.7 Misc	25
7.7.1 Determinants and PM	25
7.7.2 BEST Theorem	25
7.7.3 Primitive Roots	25
7.7.4 Sum of primes	25
7.7.5 Floor	25

## 1 Misc

## 1.1 Build

```

1 {
2   "cmd": ["g++ -std=c++14 -g -Wall '${file}' &&
           timeout 15s '${file_path}/./a.out' <'${file_path}
           }/input.txt'>'${file_path}/output.txt'"],
3   "shell": true
4 }

```

## 1.2 Snippet

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define SYNC std::ios::sync_with_stdio(false);
6   cin.tie(NULL);
7
8 template<typename T> using V = vector<T>;
9 template<typename T, typename V> using P = pair<T,
10   V>;
11
12 using ll = LL;
13 using ld = long double;
14
15 #define endl '\n'
16 #define FOR0(i,n) for(int i=0, _##i=(n); i<_##i;
17   ++i)
18 #define FOR(i,l,r) for(int i=(l), _##i=(r); i<_##i
19   ; ++i)
20 #define FORD(i,l,r) for(int i=(r), _##i=(l); --i>=
21   _##i; )
22 #define dzx cerr << "here";
23
24 const int MOD = (int)1e9 + 7, inf = 1e9;
25 const ll INF = 1e18;
26
27 int32_t main() {SYNC;
28
29   return 0;
30 }

```

## 1.3 Stack Size Increase

```

1 #include <sys/resource.h>
2
3 int main(){
4     rlimit R;
5     getrlimit(RLIMIT_STACK, &R);
6     R.rlim_cur = R.rlim_max;
7     setrlimit(RLIMIT_STACK, &R);
8 }

```

## 1.4 Variadic Multiplication and Addition

```

1
2 const int MOD = (int)1e9 + 7;
3
4 int add(){ return 0; }
5
6 template<typename... T> int add(int a, T... arg){
7     int b = add(arg...);
8     return (a + b >= MOD ? a + b - MOD : a + b);
9 }
10
11 int multiply(){ return 1; }
12
13 template<typename... Args> int multiply(int a,
14     Args... arg){
15     return (a * 1LL * multiply(arg...)) % MOD;
16 }

```

## 2 Combinatorial optimization

### 2.1 Lowest Common Ancestor

```

1 // 0-based vertex indexing. memset to -1
2 int log(int t){
3     int res = 1;
4     for(; 1 << res <= t; res++);
5     return res;
6 }
7 int lca(int u, int v){
8     if(h[u] < h[v]) swap(u, v);
9     int L = log(h[u]);
10    for(int i = L - 1; i >= 0; i--){
11        if(par[u][i] + 1 && h[u] - (1 << i) >= h[v]){
12            u = par[u][i];
13        }

```

```

14    if(v == u) return u;
15    for(int i = L - 1; i >= 0; i--){
16        if(par[u][i] + 1 && par[u][i] != par[v][i]){
17            u = par[u][i]; v = par[v][i];
18        }
19    }
20    return par[u][0];
21 }

```

### 2.2 Heavy-Light Decomposition

```

1 V<V<int>> > g;
2 int N;
3 V<int> cpar, id, depth, parent;
4 V<int> chain;
5
6 int dfs(int c, int p){
7     parent[c] = p;
8     depth[c] = depth[p] + 1;
9     int sz = 1;
10    auto it = find(g[c].begin(), g[c].end(), p);
11    if(it != g[c].end()){
12        g[c].erase(it);
13        if(g[c].empty())
14            return 1;
15        int mx = 0;
16        for(auto &i : g[c]){
17            int cur = dfs(i, c);
18            sz += cur;
19            if(cur > mx)
20                mx = cur, swap(i, g[c][0]);
21        }
22        return sz;
23    }
24
25    void form_chains(int c, int cp){
26        cpar[c] = cp;
27        id[c] = (int)chain.size();
28        chain.push_back(c);
29        for(int i = 0; i < (int)g[c].size(); i++){
30            if(i)
31                form_chains(g[c][i], g[c][i]);
32            else
33                form_chains(g[c][i], cp);
34        }
35    }

```

```

36
37 void update(int u, int v) {
38     while(u!=v) {
39         if(cpar[v] == cpar[u]) {
40             if(depth[v] < depth[u])
41                 swap(v,u);
42             supdate(0, N - 1, 1, id[u]+1, id[v]);
43             v = u;
44         }
45         else {
46             if(depth[cpar[v]] < depth[cpar[u]])
47                 swap(v,u);
48             supdate(0, N - 1, 1, id[cpar[v]], id[v]);
49             v = parent[cpar[v]];
50         }
51     }
52 }
53
54 void preprocess(int r) {
55     depth.resize(N);
56     depth[r] = 0;
57     cpar.resize(N);
58     parent.resize(N);
59     chain.clear();
60     chain.reserve(N);
61     id.resize(N);
62     dfs(r, r);
63     form_chains(r, r);
64 }

```

## 2.3 Auxiliary Tree

```

1
2 //std::vector<int> a contains vertices to form the
  aux t
3 sort(ALL(a), [](const int & a, const int & b) ->
  bool{
4     return st[a] < st[b];
5 });
6 set<int> s(a);
7 for(int i = 0, k = (int)a.size(); i + 1 < k; i++){
8     int v = lca(a[i], a[i + 1]);
9     if(s.find(v) == s.end())
10         a.push_back(v);
11     s.insert(v);
12 }

```

```

13
14 sort(ALL(a), [](const int & a, const int & b) ->
  bool{
15     return st[a] < st[b];
16 });
17
18 stack<int> S;
19 S.push(a[0]);
20
21 auto anc = [](int & a, int & b) -> bool{
22     return st[b] >= st[a] && en[b] <= en[a];
23 };
24
25 for(int i = 1; i < (int)a.size(); i++){
26     while(!anc(S.top(), a[i])) S.pop();
27     G[S.top()].pp(a[i]);
28     G[a[i]].pp(S.top());
29     S.push(a[i]);
30 }
31 //G is the Aux tree

```

## 2.4 Articulation Point and Bridges

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 const int N = 50;
5 int dis[N], low[N], par[N], AP[N], vis[N], tits;
6 void update(int u, int i, int child) {
7     //For Cut Vertices
8     if(par[u] != -1 && low[i] >= dis[u]) AP[u] =
  true;
9     if(par[u] == -1 && child > 1) AP[u] = true;
10
11     //For Finding Cut Bridge
12     if(low[i] > dis[u]){
13         //articulation bridge found.
14     }
15 }
16 void dfs(int u) {
17     vis[u] = true;
18     low[u] = dis[u] = (++tits); int child = 0;
19     for(int i : g[u]) {
20         if(!vis[i]) {
21             child++;
22             par[i] = u;

```

```

23     dfs(i);
24     low[u] = min(low[u] , low[i]);
25     update(u, i, child);
26 }
27 else if(i != par[u]) {
28     low[u] = min(low[u] , dis[i]);
29 }
30 }
31 }

```

## 2.5 Biconnected Components

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int N = (int)2e5 + 10;
4
5  vector<vector<int>> tree, g;
6  bool isBridge[N << 2], vis[N];
7  int Time, arr[N], U[N], V[N], cmpno, comp[N];
8  vector<int> temp; //temp stores component values
9
10 int adj(int u, int e){
11     return (u == U[e] ? V[e] : U[e]);
12 }
13
14 int find_bridge(int u , int edge){
15     vis[u] = true;
16     arr[u] = Time++;
17     int x = arr[u];
18
19     for(auto & i : g[u]){
20         int v = adj(u, i);
21         if(!vis[v]){
22             x = min(x, find_bridge(v, i));
23         }
24         else if(i != edge){
25             x = min(x, arr[v]);
26         }
27     }
28
29     if(x == arr[u] && edge != -1){
30         isBridge[edge] = true;
31     }
32     return x;
33 }
34

```

```

35 void dfs1(int u){
36     int current = cmpno;
37     queue<int> q;
38     q.push(u);
39     vis[u] = 1;
40     temp.push_back(current);
41
42     while(!q.empty()){
43         int v = q.front();
44         q.pop();
45         comp[v] = current;
46
47         for(auto & i : g[v]) {
48             int w = adj(v, i);
49             if(vis[w]) continue;
50             if(isBridge[i]){
51                 cmpno++;
52                 tree[current].push_back(cmpno);
53                 tree[cmpno].push_back(current);
54                 dfs1(w);
55             }
56             else{
57                 q.push(w);
58                 vis[w] = 1;
59             }
60         }
61     }
62 }
63
64 int main(){
65     int n, m;
66     cin >> n >> m;
67     g.resize(n + 2); tree.resize(n + 2);
68
69     for(int i = 0; i < m; i ++){
70         cin >> U[i] >> V[i];
71         g[U[i]].push_back(i);
72         g[V[i]].push_back(i);
73     }
74
75     cmpno = Time = 0;
76     memset(vis, false, sizeof vis);
77
78     for(int i = 0; i < n; i ++){
79         if(!vis[i]){
80             find_bridge(i , -1);
81         }
82     }

```

```

82 }
83
84 memset(vis, false, sizeof vis);
85 cmpno = 0;
86
87 for(int i = 0; i < n; i ++){
88     if(!vis[i]){
89         temp.clear();
90         cmpno++;
91         dfs1(i);
92     }
93 }
94 }

```

## 2.6 2-SAT

```

1 class sat_2{
2     public:
3         int n, m, tag;
4         vector<vector<int>> g, grev;
5         vector<bool> val;
6         vector<int> st;
7         vector<int> comp;
8
9         sat_2(){}
10        sat_2(int n) : n(n), m(2 * n), tag(0), g(m +
11            1), grev(m + 1), val(n + 1) { }
12
13        void add_edge(int u, int v) { //u or v
14            auto make_edge = [&](int a, int b) {
15                if(a < 0) a = n - a;
16                if(b < 0) b = n - b;
17                g[a].push_back(b);
18                grev[b].push_back(a);
19            };
20
21            make_edge(-u, v);
22            make_edge(-v, u);
23        }
24
25        void truth_table(int u, int v, vector<int> t)
26        {
27            for(int i = 0; i < 2; i ++){
28                for(int j = 0; j

```

```

28         ^ 1) - 1) * v);
29     }
30
31     void dfs(int u, vector<vector<int>> & G, bool
32         first) {
33         comp[u] = tag;
34         for(int & i : G[u]) if(comp[i] == -1)
35             dfs(i, G, first);
36         if(first) st.push_back(u);
37     }
38
39     bool satisfiable() {
40         tag = 0; comp.assign(m + 1, -1);
41         for(int i = 1; i <= m; i ++){
42             if(comp[i] == -1)
43                 dfs(i, g, true);
44             reverse(ALL(st));
45
46             tag = 0; comp.assign(m + 1, -1);
47             for(int & i : st){
48                 if(comp[i] != -1) continue;
49                 tag++;
50                 dfs(i, grev, false);
51             }
52
53             for(int i = 1; i <= n; i ++){
54                 if(comp[i] == comp[i + n]) return false;
55                 val[i] = comp[i] > comp[i + n];
56             }
57
58             return true;
59         }
60     };

```

## 2.7 Dinic's Max Flow

```

1 // from stanford notebook
2 struct edge {
3     int u, v;
4     ll c, f;
5     edge() { }
6     edge(int _u, int _v, ll _c, ll _f = 0): u(_u), v
7         (_v), c(_c), f(_f) { }
8 };
9 int n;

```

```

9  vector<edge> edges;
10 vector<vector<int>> > g;
11 vector<int> d, pt;
12
13 void addEdge(int u, int v, ll c, ll f = 0) {
14     g[u].emplace_back(edges.size());
15     edges.emplace_back(edge(u,v,c,f));
16     g[v].emplace_back(edges.size());
17     edges.emplace_back(edge(v,u,0,0));
18 }
19 bool bfs(int s, int t) {
20     queue<int> q({s});
21     d.assign(n+1, n+2);
22     d[s] = 0;
23     while(!q.empty()) {
24         int u = q.front(); q.pop();
25         if (u == t) break;
26         for(int k : g[u]) {
27             edge &e = edges[k];
28             if(e.f < e.c && d[e.v] > d[e.u] + 1){
29                 d[e.v] = d[e.u] + 1;
30                 q.push(e.v);
31             }
32         }
33     }
34     return d[t] < n+2;
35 }
36
37 ll dfs(int u, int t, ll flow = -1) {
38     if(u == t || !flow) return flow;
39     for(int &i = pt[u]; i < (int)(g[u].size()); i++)
40     {
41         edge &e = edges[g[u][i]], &oe=edges[g[u][i
42             ]^1];
43         if(d[e.v] == d[e.u] + 1) {
44             ll amt = e.c - e.f;
45             if (flow != -1 && amt > flow) amt = flow;
46             if(ll pushed = dfs(e.v,t,amt)) {
47                 e.f += pushed;
48                 oe.f -= pushed;
49                 return pushed;
50             }
51         }
52     }
53     return 0;
54 }

```

```

53
54 ll flow(int s, int t) {
55     ll ans = 0;
56     while(bfs(s,t)) {
57         pt.assign(n+1, 0);
58         while(ll val = dfs(s,t)) ans += val;
59     }
60     return ans;
61 }

```

## 2.8 Min Cost Max Flow

```

1  class CostFlowGraph{
2  public:
3      struct Edge{
4          int v,f,c;
5          Edge(){}
6          Edge(int v,int f,int c):v(v),f(f),c(c){}
7      };
8      V<V<int>> > g;
9      V<Edge> e;
10     V<int> pot;
11     int n;
12     int flow;
13     int cost;
14     CostFlowGraph(int sz) {
15         n=sz;
16         g.resize(n);
17         pot.assign(n,0);
18         flow=0;
19         cost=0;
20     }
21     void addEdge(int u,int v,int cap,int c) {
22         g[u].pb((int)e.size());
23         e.pb(Edge(v, cap, c));
24         g[v].pb((int)e.size());
25         e.pb(Edge(u, 0, -c));
26     }
27     void assignPots(int s) {
28         priority_queue<pii,V<pii>,greater<pii>> q;
29         V<int> npot(n,inf);
30         q.push({s,0});
31         while(!q.empty()) {
32             auto cur=q.top();q.pop();
33             if(npot[cur.fi]<=cur.se)
34                 continue;

```

```

35     npot[cur.fi]=cur.se;
36     for(auto i:g[cur.fi]) if(e[i].f>0){
37         int cst=pot[cur.fi]-pot[e[i].v]+e[i].c;
38         q.push({e[i].v,cst+cur.se});
39     }
40 }
41 for(int i=0;i<n;i++) if(npot[i]!=inf){
42     pot[i]+=npot[i];
43 }
44 }
45 void negativeEdges(int s){
46     pot.assign(n,inf);
47     pot[s]=0;
48     for(int j=0;j<n;j++){
49         for(int i=0;i<(int)e.size();i++) if(e[i].f
50             >0 && pot[e[i^1].v]!=inf){
51             pot[e[i].v]=min(pot[e[i].v],pot[e[i^1].v]+
52                 e[i].c);
53 }
54 int augment(int s,int t,int fl,V<bool> &v){
55     if(s==t)
56         return fl;
57     v[s]=1;
58     for(auto i:g[s]) if(!v[e[i].v] && e[i].f>0 &&
59         (pot[s]-pot[e[i].v]+e[i].c)==0){
60         int cf=augment(e[i].v,t,min(fl,e[i].f),v);
61         if(cf!=0){
62             e[i].f-=cf;
63             e[i^1].f+=cf;
64             return cf;
65         }
66     }
67     return 0;
68 }
69 void mcf(int s,int t,bool neg=0){
70     int cur=0;
71     V<bool> vis;
72     if(neg)
73         negativeEdges(s);
74     do{
75         vis.assign(n,0);
76         flow+=cur;
77         cost+=(pot[t]-pot[s]);
78         assignPots(s);
79         cur=augment(s,t,inf,vis);

```

```

78     }while(cur);
79 }
80 };

```

## 2.9 Global Min Cut

```

1 // Adj mat. Stoer-Wagner min cut algorithm.
2 // Running time:O(|V|^3)
3 typedef vector<int> VI;
4 typedef vector<VI> VVI;
5
6 const int INF = 1000000000;
7
8 pair<int, VI> GetMinCut(VVI &weights) {
9     int N = weights.size();
10    VI used(N), cut, best_cut;
11    int best_weight = -1;
12
13    for (int phase = N-1; phase >= 0; phase--) {
14        VI w = weights[0];
15        VI added = used;
16        int prev, last = 0;
17        for (int i = 0; i < phase; i++) {
18            prev = last;
19            last = -1;
20            for (int j = 1; j < N; j++)
21                if (!added[j] && (last == -1 || w[j] > w[last]))
22                    last = j;
23            if (i == phase-1) {
24                for (int j = 0; j < N; j++) weights[prev][j] +=
25                    weights[last][j];
26                for (int j = 0; j < N; j++) weights[j][prev] =
27                    weights[prev][j];
28                used[last] = true;
29                cut.push_back(last);
30                if (best_weight == -1 || w[last] < best_weight)
31                    {
32                        best_cut = cut;
33                        best_weight = w[last];
34                    }
35            } else {
36                for (int j = 0; j < N; j++)
37                    w[j] += weights[last][j];
38                added[last] = true;
39            }

```

```

37     }
38     return make_pair(best_weight, best_cut);
39 }
40
41 int main() {
42     int N;
43     cin >> N;
44     for(int i = 0; i < N; i++) {
45         int n, m;
46         cin >> n >> m;
47         VVI weights(n, VI(n));
48         for (int j = 0; j < m; j++) {
49             int a, b, c;
50             cin >> a >> b >> c;
51             weights[a-1][b-1] = weights[b-1][a-1] = c;
52         }
53         pair<int, VI> res = GetMinCut(weights);
54         cout << "Case #" << i+1 << ": " << res.first
55              << endl;
56     }
57 }

```

## 2.10 Bipartite Matching

```

1  // maximum cardinality bipartite matching using
   // augmenting paths.
2  // assumes that first n elements of graph
   // adjacency list belong to the left vertex set.
3  int n;
4  vector<vector<int>> graph;
5  vector<int> match, vis;
6
7  int augment(int l) {
8      if(vis[l]) return 0;
9      vis[l] = 1;
10     for(auto r: graph[l]) {
11         if(match[r]==-1 || augment(match[r])) {
12             match[r]=l; return 1;
13         }
14     }
15     return 0;
16 }
17
18 int matching() {
19     int ans = 0;

```

```

20     for(int l = 0; l < n; l++) {
21         vis.assign(n, 0);
22         ans += augment(l);
23     }
24     return ans;
25 }

```

## 2.11 Hopcraft-Karp

```

1  #define MAX 100001
2  #define NIL 0
3  #define INF (1<<28)
4
5  vector< int > G[MAX];
6  int n, m, match[MAX], dist[MAX];
7  // n: number of nodes on left side, nodes are
   // numbered 1 to n
8  // m: number of nodes on right side, nodes are
   // numbered n+1 to n+m
9
10 bool bfs() {
11     int i, u, v, len;
12     queue< int > Q;
13     for(i=1; i<=n; i++) {
14         if(match[i]==NIL) {
15             dist[i] = 0;
16             Q.push(i);
17         }
18         else dist[i] = INF;
19     }
20     dist[NIL] = INF;
21     while(!Q.empty()) {
22         u = Q.front(); Q.pop();
23         if(u!=NIL) {
24             len = G[u].size();
25             for(i=0; i<len; i++) {
26                 v = G[u][i];
27                 if(dist[match[v]]==INF) {
28                     dist[match[v]] = dist[u] + 1;
29                     Q.push(match[v]);
30                 }
31             }
32         }
33     }
34     return (dist[NIL]!=INF);
35 }

```



```

36 bool dfs(int u) {
37     int i, v, len;
38     if(u!=NIL) {
39         len = G[u].size();
40         for(i=0; i<len; i++) {
41             v = G[u][i];
42             if(dist[match[v]]==dist[u]+1) {
43                 if(dfs(match[v])) {
44                     match[v] = u;
45                     match[u] = v;
46                     return true;
47                 }
48             }
49         }
50         dist[u] = INF;
51         return false;
52     }
53     return true;
54 }
55 }
56
57 int hopcroft_karp() {
58     int matching = 0, i;
59     // match[] is assumed NIL for all vertex in G
60     while(bfs())
61         for(i=1; i<=n; i++)
62             if(match[i]==NIL && dfs(i))
63                 matching++;
64     return matching;
65 }

```

## 2.12 Hungarian

```

1 // Min cost BPM via shortest augmenting paths
2 // O(n^3).Solves 1000x1000 in ~1s
3 // cost[i][j] = cost for pairing left node i with
  // right node j
4 // Lmate[i] = index of right node that left node
  // i pairs with
5 // Rmate[j] = index of left node that right node
  // j pairs with
6 // The values in cost[i][j] may be +/- . To
  // perform
7 // maximization, negate cost[[]].
8 typedef vector<double> VD;
9 typedef vector<VD> VVD;

```

```

10 typedef vector<int> VI;
11
12 double MinCostMatching(const VVD &cost, VI &Lmate,
13     VI &Rmate) {
14     int n = int(cost.size());
15     // construct dual feasible solution
16     VD u(n);
17     VD v(n);
18     for (int i = 0; i < n; i++) {
19         u[i] = cost[i][0];
20         for (int j = 1; j < n; j++) u[i] = min(u[i],
21             cost[i][j]);
22     }
23     for (int j = 0; j < n; j++) {
24         v[j] = cost[0][j] - u[0];
25         for (int i = 1; i < n; i++) v[j] = min(v[j],
26             cost[i][j] - u[i]);
27     }
28     // construct primal solution satisfying
29     // complementary slackness
30     Lmate = VI(n, -1);
31     Rmate = VI(n, -1);
32     int mated = 0;
33     for (int i = 0; i < n; i++) {
34         for (int j = 0; j < n; j++) {
35             if (Rmate[j] != -1) continue;
36             if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10)
37                 {
38                     Lmate[i] = j;
39                     Rmate[j] = i;
40                     mated++;
41                     break;
42                 }
43         }
44     }
45     VD dist(n);
46     VI dad(n);
47     VI seen(n);
48     // repeat until primal solution is feasible
49     while (mated < n) {
50         // find an unmatched left node
51         int s = 0;

```

```

52 while (Lmate[s] != -1) s++;
53
54 // initialize Dijkstra
55 fill(dad.begin(), dad.end(), -1);
56 fill(seen.begin(), seen.end(), 0);
57 for (int k = 0; k < n; k++)
58     dist[k] = cost[s][k] - u[s] - v[k];
59
60 int j = 0;
61 while (true) {
62     // find closest
63     j = -1;
64     for (int k = 0; k < n; k++) {
65         if (seen[k]) continue;
66         if (j == -1 || dist[k] < dist[j]) j = k;
67     }
68     seen[j] = 1;
69
70     // termination condition
71     if (Rmate[j] == -1) break;
72
73     // relax neighbors
74     const int i = Rmate[j];
75     for (int k = 0; k < n; k++) {
76         if (seen[k]) continue;
77         const double new_dist = dist[j] + cost[i][k] -
78             u[i] - v[k];
79         if (dist[k] > new_dist) {
80             dist[k] = new_dist;
81             dad[k] = j;
82         }
83     }
84     // update dual variables
85     for (int k = 0; k < n; k++) {
86         if (k == j || !seen[k]) continue;
87         const int i = Rmate[k];
88         v[k] += dist[k] - dist[j];
89         u[i] -= dist[k] - dist[j];
90     }
91     u[s] += dist[j];
92
93     // augment along path
94     while (dad[j] >= 0) {
95         const int d = dad[j];

```

```

97         Rmate[j] = Rmate[d];
98         Lmate[Rmate[j]] = j;
99         j = d;
100     }
101     Rmate[j] = s;
102     Lmate[s] = j;
103     mated++;
104 }
105
106 double value = 0;
107 for (int i = 0; i < n; i++)
108     value += cost[i][Lmate[i]];
109
110 return value;
111 }
112 }

```

## 2.13 Link

```

1 struct Node { // Splay tree. Root's pp contains
2     tree's parent.
3     Node *p = 0, *pp = 0, *c[2];
4     bool flip = 0;
5     Node() { c[0] = c[1] = 0; fix(); }
6     void fix() {
7         if (c[0]) c[0]->p = this;
8         if (c[1]) c[1]->p = this;
9         // (+ update sum of subtree elements etc.
10        if wanted)
11    }
12    void push_flip() {
13        if (!flip) return;
14        flip = 0; swap(c[0], c[1]);
15        if (c[0]) c[0]->flip ^= 1;
16        if (c[1]) c[1]->flip ^= 1;
17    }
18    int up() { return p ? p->c[1] == this : -1; }
19    void rot(int i, int b) {
20        int h = i ^ b;
21        Node *x = c[i], *y = b == 2 ? x : x->c[h],
22            *z = b ? y : x;
23        if ((y->p = p)) p->c[up()] = y;
24        c[i] = z->c[i ^ 1];
25        if (b < 2) {
26            x->c[h] = y->c[h ^ 1];
27            z->c[h ^ 1] = b ? x : this;

```

```

25     }
26     y->c[i ^ 1] = b ? this : x;
27     fix(); x->fix(); y->fix();
28     if (p) p->fix();
29     swap(pp, y->pp);
30 }
31 void splay() { /// Splay this up to the root.
32     Always finishes without flip set.
33     for (push_flip(); p; ) {
34         if (p->p) p->p->push_flip();
35         p->push_flip(); push_flip();
36         int c1 = up(), c2 = p->up();
37         if (c2 == -1) p->rot(c1, 2);
38         else p->p->rot(c2, c1 != c2);
39     }
40     Node* first() { /// Return the min element of
41         the subtree rooted at this, splayed to the
42         top.
43         push_flip();
44         return c[0] ? c[0]->first() : (splay(),
45             this);
46     }
47 };
48
49 struct LinkCut {
50     vector<Node> node;
51     LinkCut(int N) : node(N) {}
52
53     void link(int u, int v) { /// add an edge (u, v)
54         assert(!connected(u, v));
55         make_root(&node[u]);
56         node[u].pp = &node[v];
57     }
58
59     void cut(int u, int v) { /// remove an edge (u,
60         v)
61         Node *x = &node[u], *top = &node[v];
62         make_root(top); x->splay();
63         assert(top == (x->pp ? x->c[0]));
64         if (x->pp) x->pp = 0;
65         else {
66             x->c[0] = top->p = 0;
67             x->fix();
68         }
69     }
70 }

```

```

65     }
66
67     bool connected(int u, int v) { /// are u, v in
68         the same tree?
69         Node* nu = access(&node[u])->first();
70         return nu == access(&node[v])->first();
71     }
72
73     /// Move u to root of represented tree.
74     void make_root(Node* u) {
75         access(u);
76         u->splay();
77         if (u->c[0]) {
78             u->c[0]->p = 0;
79             u->c[0]->flip ^= 1;
80             u->c[0]->pp = u;
81             u->c[0] = 0;
82             u->fix();
83         }
84     }
85
86     /// Move u to root aux tree. Return the root
87     of the root aux tree.
88     Node* access(Node* u) {
89         u->splay();
90         while (Node* pp = u->pp) {
91             pp->splay(); u->pp = 0;
92             if (pp->c[1]) {
93                 pp->c[1]->p = 0; pp->c[1]->pp = pp
94                 ; }
95             pp->c[1] = u; pp->fix(); u = pp;
96         }
97         return u;
98     }
99 };

```

## 3 Data Structures

### 3.1 Implicit Treap

```

1 ///1-based with lazy-updates, range sum query
2 struct node {
3     int val, sum, lazy, prior, size;
4     node *l, *r;
5 };
6 const int N = 2e5;

```

```

7 node pool[N]; int poolptr=0;
8 typedef node* pnode;
9 int sz(pnode t) { return t?t->size:0; }
10 void upd_sz(pnode t) { if(t) t->size = sz(t->l) +
11     1 + sz(t->r); }
12 void lazy(pnode t) {
13     if(!t || !t->lazy) return;
14     t->val+=t->lazy;
15     t->sum+=t->lazy*sz(t);
16     if(t->l)t->l->lazy+=t->lazy;
17     if(t->r)t->r->lazy+=t->lazy;
18     t->lazy = 0;
19 }
20 void reset(pnode t) {
21     if(t) t->sum=t->val;
22 }
23 void combine(pnode& t, pnode l, pnode r) {
24     if(!l || !r) return void(t=l?l:r);
25     t->sum = l->sum + r->sum;
26 }
27 void operation(pnode t) {
28     if(!t) return;
29     reset(t);
30     lazy(t->l); lazy(t->r);
31     combine(t,t->l,t); combine(t,t,t->r);
32 }
33 void split(pnode t, pnode& l, pnode& r, int pos,
34     int add = 0) {
35     if(!t) return void(l=r=NULL);
36     lazy(t); int curr_pos = add + sz(t->l);
37     if(curr_pos<pos) split(t->r,t->r,r,pos,
38         curr_pos+1),l=t;
39     else split(t->l,l,t->r,pos,add),r=t;
40     upd_sz(t); operation(t);
41 }
42 void merge(pnode& t, pnode l, pnode r) {
43     lazy(l); lazy(r);
44     if(!l || !r) t = l?l:r;
45     else if(l->prior > r->prior) merge(l->r,l->r,r),t=l;
46     else merge(r->l, l, r->l), t=r;
47     upd_sz(t); operation(t);
48 }
49 pnode init(int val) {
50     pnode ret = &(pool[poolptr++]);

```

```

48     ret->prior = rand(); ret->size = 1;
49     ret->val = val; ret->sum = val; ret->lazy = 0;
50     return ret;
51 }
52 int query(pnode t, int l, int r) {
53     pnode L,mid,R;
54     split(t, L, mid, l-1); split(mid, t, R, r-1);
55     int ans = t->sum;
56     merge(mid, L, t); merge(t, mid, R);
57     return ans;
58 }
59 void upd(pnode t, int l, int r, int val) {
60     pnode L, mid, R;
61     split(t, L, mid, l-1); split(mid, t, R, r-1);
62     t->lazy += val;
63     merge(mid, L, t); merge(t, mid, R);
64 }
65 void insert(pnode& t, ll val, int pos) {
66     pnode l;
67     split(t,l,t,pos-1); merge(l,l,init(val));
68     merge(t,l,t);
69 }

```

### 3.2 Segment Tree

```

1 // This code solves problem Help Ashu on
2 // hackerearth
3 // Iterative segment tree supporting non
4 // commutative combiner function
5 // The combiner function and identity of the
6 // combiner function are taken as constructor
7 // arguments
8 // Assign the initial input into t[size] to t[2*
9 // size-1] then call build
10 // Memory 2*size*sizeof(T)
11 // Time complexity O(log(size))
12 #include <bits/stdc++.h>
13 using namespace std;
14 /* Equinox */
15 template<typename T>
16 class SegTree{
17 public:
18     vector<T> t;
19     T identity;
20     T (*combine)(T,T);
21     int size;

```

```

17 SegTree(T (*op)(T,T), T e, int n) {
18     combine=op;
19     identity=e;
20     t.assign(2*n, e);
21     size=n;
22 }
23 void build() {for(int i=size-1; i>0; i--) t[i]=
24     combine(t[i<<1], t[i<<1|1]);}
25 T query(int l, int r) {
26     T lt=identity;
27     T rt=identity;
28     for(l+=size, r+=size; l<=r; r>>=1, l>>=1) {
29         if(l&1) lt=combine(lt, t[l++]);
30         if(!(r&1)) rt=combine(t[r--], rt);
31     }
32     return combine(lt, rt);
33 }
34 void update(int p, T v) {for(t[p+=size]=v; p>>=1;) t
35     [p]=combine(t[p<<1], t[p<<1|1]);}
36 };
37 int32_t main() {
38     int n;
39     cin>>n;
40     SegTree<int> tree([](int a, int b) {return a+b
41         ;}, 0, n);
42     for(int i=0; i<n; i++) {
43         int a;
44         cin>>a;
45         tree.t[i+n]=a&1;
46     }
47     tree.build();
48     int q;
49     cin>>q;
50     while(q-->0) {
51         int c, x, y;
52         cin>>c>>x>>y;
53         switch(c) {
54             case 0:
55                 tree.update(x-1, y&1);
56                 break;
57             case 1:
58                 cout<<(y-x+1)-tree.query(x-1, y-1)<<"\n";
59                 break;
60             case 2:
61                 cout<<tree.query(x-1, y-1)<<"\n";

```

```

59     }
60 }
61 return 0;
62 }

```

### 3.3 Lazy Propagation

```

1 // This code solves problem LITE on spoj
2 // Iterative segment tree with lazy propagation
3 // supporting non commutative combiner functions
4 // The combiner function and identity of the
5 // combiner function are taken as contructor
6 // arguments
7 // Also the function for application of lazy nodes
8 // onto tree nodes is taken as parameter along
9 // with Zero of lazy node
10 // Assign the initial input into t[size] to t[2*
11 // size-1] then call build
12 // Memory 2*size*sizeof(T)+2*size*sizeof(L)
13 // Time complexity O(log(size))
14 #include <bits/stdc++.h>
15 using namespace std;
16 /* Equinox */
17 template<typename T, typename L>
18 class SegTree{
19 public:
20     vector<T> t;
21     vector<T> lz;
22     T identity;
23     L zero;
24     T (*combine)(T, T);
25     void (*apply)(T&, L&, L&, int k);
26     int size;
27     int height;
28     SegTree(T (*op)(T, T), T e, void (*pro)(T&, L&, L&,
29         int k), L z, int n) {
30         combine=op;
31         apply=pro;
32         identity=e;
33         zero=z;
34         t.assign(2*n, e);
35         lz.assign(2*n, z);
36         size=n;
37         height = sizeof(int)*8-__builtin_clz(n);
38     }

```

```

32 void build() {for(int i=size-1;i>0;i--)t[i]=
    combine(t[i<<1],t[i<<1|1]);}
33 void push(int p){
34     for(int s=height;s>0;s--){
35         int i=p>>s;
36         apply(t[i<<1],lz[i<<1],lz[i],1<<(s-1));
37         apply(t[i<<1|1],lz[i<<1|1],lz[i],1<<(s-1));
38         lz[i]=zero;
39     }
40 }
41 void reassign(int p){
42     for(p>>=1;p>0;p>>=1)
43         if(lz[p]==zero)
44             t[p]=combine(t[p<<1],t[p<<1|1]);
45 }
46 T query(int l,int r){
47     push(l+=size);
48     push(r+=size);
49     T lt=identity;
50     T rt=identity;
51     for(;l<=r;r>>=1,l>>=1){
52         if(l&1) lt=combine(lt,t[l++]);
53         if(!(r&1)) rt=combine(t[r--],rt);
54     }
55     return combine(lt,rt);
56 }
57 void update(int p,T v){push(p+=size);for(t[p]=v;
    p>>=1;)t[p]=combine(t[p<<1],t[p<<1|1]);}
58 void update(int l,int r,L v){
59     push(l+=size);
60     push(r+=size);
61     int k=1;
62     int l0=l,r0=r;
63     for(;l<=r;r>>=1,l>>=1,k<=&1){
64         if(l&1) apply(t[l],lz[l],v,k),l++;
65         if(!(r&1)) apply(t[r],lz[r],v,k),r--;
66     }
67     reassign(l0);
68     reassign(r0);
69 }
70 };
71 int32_t main(){
72     int n,m;
73     cin>>n>>m;
74     SegTree<int,int> s([](int a, int b){return a + b

```

```

    ;},0,[])(int &v,int &l,int &u,int k){if(u)v=k-
    v;l^=u;},0,n);
75 while(m--){
76     int c;
77     cin>>c;
78     if(!c){
79         int l,r;
80         cin>>l>>r;
81         s.update(l-1,r-1,1);
82     }
83     else{
84         int l,r;
85         cin>>l>>r;
86         cout<<s.query(l-1,r-1)<<"\n";
87     }
88 }
89 return 0;
90 }

```

## 4 Math

### 4.1 Extended Euclid

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 using LL = long long;
5
6 template<typename T> T gcd(T a , T b){return (a ?
    gcd(b % a , a) : b);} //supposing a is small and
    b is large.
7 template<typename T> pair<T,T> extend_euclid(T a,
    T b){ //supposing a is small and b is large.
8     pair<T,T> a_one = {1, 0} , b_one = {0 , 1};
9     // b_one is just the second last step's
    coefficient, a_one is the last step's
    coefficient
10    if(!b) return a_one;
11    while(a){
12        /* We first start from writing
13        b = 0(a) + 1(b), for which it's b_one
14        a = 1(a) + 0(b), for which it's a_one
15        b = b % a + (b / a)*a, then
16        */
17        T q = b / a; T r = b % a;

```

```

18     T dx = b_one.first - q*a_one.first;
19     T dy = b_one.second - q*a_one.second;
20     b = a; a = r;
21     b_one = a_one;
22     a_one = {dx , dy};
23 }
24 return b_one;
25 }
26
27 int main() {
28     LL a, m; cin >> a >> m;
29     auto ans = extend_euclid(a, m);
30     LL x = (ans.first + m)%m; //Inverse Modulo (m) $
        ax=1 mod(m) and gcd(a,m) == 1
31     cout << (ans.first + m) % m << endl;
32     return 0;
33 }

```

## 4.2 Fast Fourier Transform

```

1  const long double PI=acos(-1.0);
2  typedef long long ll;
3  typedef long double ld;
4  typedef vector<ll> VL;
5  int bits(int x) {
6      int r=0;
7      while(x) {
8          r++;
9          x>>=1;
10     }
11     return r;
12 }
13 int reverseBits(int x,int b) {
14     int r=0;
15     for(int i=0;i<b;i++) {
16         r<<=1;
17         r|=(x&1);
18         x>>=1;
19     }
20     return r;
21 }
22 class Complex{
23 public:
24     ld r,i;
25     Complex() {r=0.0;i=0.0;}
26     Complex(ld a,ld b) {r=a;i=b;}

```

```

27 };
28 Complex operator*(Complex a,Complex b) {
29     return Complex(a.r*b.r-a.i*b.i,a.r*b.i+a.i*b.r);
30 }
31 Complex operator-(Complex a,Complex b) {
32     return Complex(a.r-b.r,a.i-b.i);
33 }
34 Complex operator+(Complex a,Complex b) {
35     return Complex(a.r+b.r,a.i+b.i);
36 }
37 Complex operator/(Complex a,ld b) {
38     return Complex(a.r/b,a.i/b);
39 }
40 Complex EXP(ld theta) {
41     return Complex(cos(theta),sin(theta));
42 }
43
44 typedef vector<Complex> VC;
45
46 void FFT(VC& A,int inv) {
47     int l=A.size();
48     int b=bits(l)-1;
49     VC a(A);
50     for(int i=0;i<l;i++) {
51         A[reverseBits(i,b)]=a[i];
52     }
53     for(int i=1;i<=b;i++) {
54         int m=(1<<i);
55         int n=m>>1;
56         Complex wn=EXP((ld)inv*(ld)2.0*PI/(ld)m);
57         for(int j=0;j<l;j+=m) {
58             Complex w(1.0,0.0);
59             for(int k=j;k<j+n;k++) {
60                 Complex t1=A[k]+w*A[k+n];
61                 Complex t2=A[k]-w*A[k+n];
62                 A[k]=t1;
63                 A[k+n]=t2;
64                 w=w*wn;
65             }
66         }
67     }
68     if(inv==-1) {
69         for(auto &i:A) {
70             i=i/(ld)l;
71         }
72     }

```

```

73 }
74
75 VL Convolution(VL & a, VL & b) {
76     int tot_size = (int)a.size() + (int)b.size();
77     int bit = bits(tot_size);
78     int l = 1 << bit;
79     VC A, B, C;
80     A.reserve(l); B.reserve(l); C.reserve(l);
81     for(int i = 0; i < l; i++) {
82         if(i < (int)a.size()) A.pb({(ld)a[i], 0.0});
83         else A.pb({0.0, 0.0});
84         if(i < (int)b.size()) B.pb({(ld)b[i], 0.0});
85         else B.pb({0.0, 0.0});
86     }
87     FFT(A, 1);
88     FFT(B, 1);
89     for(int i = 0; i < l; i++) {
90         C.pb(A[i] * B[i]);
91     }
92     FFT(C, -1);
93     VL c;
94     for(auto & i : C) {
95         c.pb(round(i.r));
96     }
97     return c;
98 }

```

### 4.3 Number Theoretic Transform

```

1  const ll mod = 163577857, root = 8*23; // =
   998244353
2
3
4  ll modpow(ll x, ll y)
5  {
6      ll ret=1;
7      for (; y >= 1, x = x*x%mod)
8          if (y&1) ret = x*ret%mod;
9      return ret;
10 }
11
12 ll modinv(ll a)
13 {
14     return modpow(a, mod-2);
15 }
16
17 // https://arxiv.org/pdf/1708.01873.pdf

```

```

18 void BitReverse(vl& vec, int n) {
19     int L = 31 - __builtin_clz(n);
20     int rev = 0;
21     rep(i, 0, n)
22     {
23         if (i < rev) swap(vec[i], vec[rev]);
24         int tail = i ^ (i + 1), shift = 32 -
           __builtin_clz(tail);
25         rev ^= tail << (L - shift);
26     }
27 }
28
29
30 // For p < 2^30 there is also e . g . (5 << 25, 3)
   , (7 << 26, 3) ,
31 // (479 << 21, 3) and (483 << 21, 5) . The last
   two are > 10^9.
32 void ntt(vl& x, vl roots) {
33     int n = x.size();
34     if (n == 1) return;
35     BitReverse(x, n);
36
37     int L = 31 - __builtin_clz(n);
38     int mask = (1<<L)-1;
39
40     for (int s=1; s<=L; ++s) {
41         int m = 1<<s;
42         int dx = n/m;
43         for (int k = 0; k<n; k+=m) {
44             int index = 0;
45             for (int j = 0; j<m/2; ++j)
46             {
47                 ll t = roots[index] * (x[k+j+m/2] %
                     mod);
48                 ll u = x[k+j];
49                 x[k+j] = (u+t);
50                 x[k+j+m/2] = (u-t);
51                 index = (index+dx)&mask;
52             }
53         }
54     }
55
56     for (int i=0; i<n; ++i)
57         x[i] %= mod;
58 }
59

```



```

60 // Remember roots to speed up fft
61 vl precalced_roots[30][2];
62
63 void ntt(vl& x, bool inv = false) {
64     int L = sz(x);
65     int s = 31 - __builtin_clz(L);
66
67     if (precalced_roots[s][inv].size() == 0)
68     {
69         ll e = modpow(root, (mod-1) / L);
70         if (inv)
71             e = modpow(e, mod-2);
72         vl roots(L, 1);
73         rep(i, 1, L)
74         {
75             roots[i] = roots[i-1] * e % mod;
76         }
77         precalced_roots[s][inv] = roots;
78     }
79     ntt(x, precalced_roots[s][inv]);
80 }
81
82 vl conv(vl a, vl b) {
83     int s = sz(a) + sz(b) - 1; if (s <= 0) return {};
84     int L = s > 1 ? 32 - __builtin_clz(s - 1) : 0,
85           n = 1 << L;
86     if (s <= 100 || sz(a) < 100 || sz(b) < 100) { // (
87         factor 10 optimization for jaj, j b j =
88         10)
89         vl c(s);
90         rep(i, 0, sz(a)) rep(j, 0, sz(b))
91         {
92             c[i+j] = (c[i+j] + a[i] * b[j]) %
93             mod;
94         }
95         return c;
96     }
97     a.resize(n); ntt(a);
98     b.resize(n); ntt(b);
99
100     ll d = modpow(n, mod-2);
101     rep(i, 0, n) a[i] = a[i] * b[i] % mod * d % mod; // c[i] = a
102     [i] * b[i] % mod * d % mod;
103     ntt(a, true);
104     a.resize(s);
105     return a;

```

101 }

#### 4.4 Large Factorial

```

1 ll fmod(ll x, ll md, ll p) {
2     V<ll> pre(md);
3     pre[0] = 1;
4     for (ll i = 1; i < md; i++) {
5         if (i % p != 0)
6             pre[i] = (pre[i-1] * i) % md;
7         else
8             pre[i] = pre[i-1];
9     }
10    ll r = 1;
11    while (x) {
12        ll cy = x / md;
13        r = (r * modex(pre[md-1], cy, md)) % md;
14        r = (r * pre[x % md]) % md;
15        x /= p;
16    }
17    return r;
18 }

```

#### 4.5 Large Modulo Multiplication

```

1 // Finds (a*b)%m when either can be as big as
2 // 10^18
3 #define ll long long
4 #define ld long double
5 ll mulmod(ll a, ll b, ll m) {
6     a %= m; b %= m;
7     ll q = (ll) (((ld)a * (ld)b) / (ld)m);
8     ll r = a * b - q * m;
9     if (r > m) r %= m;
10    if (r < 0) r += m;
11    return r;

```

#### 4.6 Segmented Sieve

```

1 // Segmented Sieve
2 // N = sqrt(b)
3 // Time complexity: O(N.log(B-A))
4 #define A 10000000000000LL

```

```

5 #define B 1000000100000LL
6 bitset<B-A> p;
7 void seive() {
8     p.set();
9     for (ll i=2; i*i<=B; i++) {
10         for (ll j=((A+i-1)/i)*i; j<=B; j+=i) {
11             p.reset(j-A);
12         }
13     }
14 }

```

## 4.7 Random Number Generator

```

1 random_device rd;
2 mt19937 gen(rd()); // mersenne twister (only for
   // 32 bit unsigned numbers)
3 uniform_int_distribution<int> dis(1, 10000); //
   // dis(L, R) uniformly generates [L, R] (inclusive)
4
5 /* For 64 bit numbers */
6 mt19937_64 gen2(rd());
7 uniform_int_distribution<LL> dis2((int)1e9 + 7, (
   int)1e10);
8 cout << dis2(gen) << endl << dis2(gen) << endl;

```

## 5 Strings

### 5.1 Aho Corasick

```

1 const int N = 500*5005;
2 map<char, int> nxt[N], go[N];
3 int par[N], occ[N], sz = 1, link[N];
4 char parc[N];
5 void add(string& s, int i) {
6     int cur = 1;
7     for(char c : s) {
8         if(!nxt[cur][c]) {
9             sz++;
10            parc[sz]=c, par[sz]=cur, nxt[cur][c]=sz,
               cur=sz;
11        }
12        else cur=nxt[cur][c];
13    }
14    occ[cur]++;

```

```

15 }
16 int GO(int p, char c);
17 int getlink(int p) {
18     if(!link[p]) {
19         if(p==1 || par[p]==1) link[p]=1;
20         else {
21             link[p]=GO(getlink(par[p]), parc[p]);
22             occ[p] += occ[link[p]];
23         }
24     }
25     return link[p];
26 }
27 int GO(int p, char c) {
28     auto it = nxt[p].find(c);
29     if(it == nxt[p].end()) {
30         auto it = go[p].find(c);
31         if (it==go[p].end())
32             return (go[p][c]= p==1 ? 1: GO(getlink
               (p), c));
33         else return it->ss;
34     } else return it->ss;
35 }

```

### 5.2 Suffix Array

```

1 vector<int> suffix_array(string s) {
2     s += "$";
3     int n = s.size();
4     const int alphabet = 256;
5     vector<int> p(n), c(n), cnt(max(alphabet, n),
6         0);
7     for (int i = 0; i < n; i++)
8         cnt[s[i]]++;
9     for (int i = 1; i < alphabet; i++)
10        cnt[i] += cnt[i-1];
11    for (int i = 0; i < n; i++)
12        p[--cnt[s[i]]] = i;
13    c[p[0]] = 0;
14    int classes = 1;
15    for (int i = 1; i < n; i++) {
16        if (s[p[i]] != s[p[i-1]])
17            classes++;
18        c[p[i]] = classes - 1;
19    }
20    vector<int> pn(n), cn(n);

```

```

20 for (int h = 0; (1 << h) < n; ++h) {
21     for (int i = 0; i < n; i++) {
22         pn[i] = p[i] - (1 << h);
23         if (pn[i] < 0)
24             pn[i] += n;
25     }
26     fill(cnt.begin(), cnt.begin() + classes,
27          0);
28     for (int i = 0; i < n; i++)
29         cnt[c[pn[i]]]++;
30     for (int i = 1; i < classes; i++)
31         cnt[i] += cnt[i-1];
32     for (int i = n-1; i >= 0; i--)
33         p[--cnt[c[pn[i]]]] = pn[i];
34     cn[p[0]] = 0;
35     classes = 1;
36     for (int i = 1; i < n; i++) {
37         pair<int, int> cur = {c[p[i]], c[(p[i]
38             + (1 << h)) % n]};
39         pair<int, int> prev = {c[p[i-1]], c[(p
40             [i-1] + (1 << h)) % n]};
41         if (cur != prev)
42             ++classes;
43         cn[p[i]] = classes - 1;
44     }
45     c.swap(cn);
46 }
47 p.erase(p.begin()); // remove "$" suffix
48 return p;
49 }
50
51 vector<int> lcp_array(string s, vector<int> sa) {
52     int n=s.size(), k=0;
53     vector<int> lcp(n,0), r(n,0);
54     for(int i=0; i<n; i++) r[sa[i]]=i;
55     for(int i=0; i<n; i++, k?k--:0) {
56         if(r[i]==n-1) {k=0; continue;}
57         int j=sa[r[i]+1];
58         while(i+k<n && j+k<n && s[i+k]==s[j+k]) k
59             ++;
60         lcp[r[i]]=k;
61     }
62     return lcp;
63 }

```

### 5.3 Suffix Tree

```

1  const int N=1000000, // maximum possible number
    of nodes in suffix tree
2  INF=1000000000; // infinity constant
3  string a; // input string for which the
    suffix tree is being built
4  int t[N][26], // array of transitions (state,
    letter)
5  l[N], // left...
6  r[N], // ...and right boundaries of the
    substring of a which correspond to incoming
    edge
7  p[N], // parent of the node
8  s[N], // suffix link
9  tv, // the node of the current suffix (if
    we're mid-edge, the lower node of the edge)
10 tp, // position in the string which
    corresponds to the position on the edge (
    between l[tv] and r[tv], inclusive)
11 ts, // the number of nodes
12 la; // the current character in the string
13
14 void ukkadd(int c) { // add character s to the
    tree
15     suff++; // we'll return here after each
    transition to the suffix (and will add
    character again)
16     if (r[tv]<tp) { // check whether we're still
    within the boundaries of the current edge
    // if we're not, find the next edge. If it
    doesn't exist, create a leaf and add
    it to the tree
17         if (t[tv][c]==-1) {t[tv][c]=ts;l[ts]=la;p[
    ts++]=tv;tv=s[tv];tp=r[tv]+1;goto suff
    ;}
18         tv=t[tv][c];tp=l[tv];
19     } // otherwise just proceed to the next edge
20     if (tp==-1 || c==a[tp]-'a')
21         tp++; // if the letter on the edge equal c
    , go down that edge
22     else {
23         // otherwise split the edge in two with
    middle in node ts
24         l[ts]=l[tv];r[ts]=tp-1;p[ts]=p[tv];t[ts][a
25         l[ts]=l[tv];r[ts]=tp-1;p[ts]=p[tv];t[ts][a

```

```

26     [tp] - 'a'] = tv;
    // add leaf ts+1. It corresponds to
    // transition through c.
27 t[ts][c] = ts+1; l[ts+1] = la; p[ts+1] = ts;
28 // update info for the current node -
    // remember to mark ts as parent of tv
29 l[tv] = tp; p[tv] = ts; t[p[ts]][a[l[ts]] - 'a'] =
    ts; ts += 2;
30 // prepare for descent
31 // tp will mark where are we in the
    // current suffix
32 tv = s[p[ts-2]]; tp = l[ts-2];
33 // while the current suffix is not over,
    // descend
34 while (tp <= r[ts-2]) { tv = t[tv][a[tp] - 'a'];
    tp += r[tv] - l[tv] + 1; }
35 // if we're in a node, add a suffix link
    // to it, otherwise add the link to ts
36 // (we'll create ts on next iteration).
37 if (tp == r[ts-2] + 1) s[ts-2] = tv; else s[ts
    - 2] = ts;
38 // add tp to the new edge and return to
    // add letter to suffix
39 tp = r[tv] - (tp - r[ts-2]) + 2; goto suff;
40 }
41 }
42
43 void Build() {
44     ts = 2;
45     tv = 0;
46     tp = 0;
47     fill(r, r+N, (int)a.size() - 1);
48     // initialize data for the root of the tree
49     s[0] = 1;
50     l[0] = -1;
51     r[0] = -1;
52     l[1] = -1;
53     r[1] = -1;
54     memset(t, -1, sizeof t);
55     fill(t[1], t[1] + 26, 0);
56     // add the text to the tree, letter by letter
57     for (la = 0; la < (int)a.size(); ++la)
58         ukkadd(a[la] - 'a');

```

## 5.4 Z Algorithm

```

1 vector<int> z_compute(const string & S_) {
2     int zn = S_.size();
3     vector<int> z((int)S_.size() + 10);
4     int l = 0, r = 0;
5     z[0] = 0;
6     for (int i = 1; i < zn; i++) {
7         if (i <= r)
8             z[i] = min(r - i + 1, z[i - l]);
9         while (i + z[i] < zn && S_[z[i]] == S_[i + z[i]
10             ])
11             ++z[i];
12         if (i + z[i] - 1 > r)
13             l = i, r = i + z[i] - 1;
14     }
15     return z;

```

## 6 Geometry

### 6.1 Geometry Library

```

1
2 /*Returns the orientation of Point C wrt line from
    B to A
3 * It returns :-
4 * -1 if C lies to left
5 * +1 if C lies to the right of the line
6 * 0 if C lies on the line
7 */
8
9 int ccw(Point a, Point b, Point c) {
10     int ans = (a - c) ^ (b - c);
11     return ans < 0 ? -1 : ans > 0;
12 }
13
14 /* 0 means outside, 1 means looselyinside the
    polygon(include on the edges of the polygon)*/
15
16 /* To change it strictly inside
17 * change the type of this function to int
18 * 0 means on the edge / point
19 * +1 means strictly inside
20 * -1 means strictly outside
21 * winding number = 0 means outside
22 * winding number != 0 means inside
23 */

```

```

24
25 bool is_inside(auto & p, auto & pt){
26     int n = (int)p.size();
27     int cnt = 0;
28
29     for(int i = 0; i < n; i++){
30         if(p[i] == pt) return true;
31         int j = (i + 1) % n;
32         if(p[i].y == pt.y && p[j].y == pt.y){
33             if(pt.x >= min(p[i].x, p[j].x) && pt.x <=
34                 max(p[i].x, p[j].x))
35                 return true;
36         }else{
37             bool below = p[i].y < pt.y;
38             if(below != (p[j].y < pt.y)){
39                 auto orientation = ccw(p[i], p[j], pt);
40                 if(!orientation) return true;
41                 if(below == (orientation > 0)) cnt +=
42                     below ? 1 : -1;
43             }
44         }
45     }
46     return (cnt != 0);

```

## 6.2 Convex Hull

```

1 vector<Point> half_hull(vector<Point> &pts, int t){
2     vector<Point> hull;
3     hull.pb(pts[0]);

```

```

4     hull.pb(pts[1]);
5     for(int i=2; i<n; i++){
6         while((int)hull.size()>1){
7             Point p1=hull[(int)hull.size()-2];
8             Point p2=hull.back();
9             if(((p1-pts[i])*(p2-pts[i]))*t)>=0){
10                 hull.pop_back();
11             }
12             else break;
13         }
14         hull.pb(pts[i]);
15     }
16     return move(hull);
17 }
18
19 vector<Point> convex_hull(vector<Point> &pts){
20     sort(pts.begin(), pts.end(), [](Point &a, Point
21         &b){
22         if(a.x==b.x)
23             return a.y<b.y;
24             return a.x<b.x;
25         });
26     vector<Point> uh, lh;
27     uh=half_hull(pts, 1);
28     lh=half_hull(pts, -1);
29     lh.pop_back();
30     reverse(lh.begin(), lh.end());
31     uh.insert(uh.end(), lh.begin(), lh.end());
32     return move(uh);

```

## 7 Formulas

Catalan	$C_0 = 1, C_n = \frac{1}{n+1} \binom{2n}{n} = \sum_{i=0}^{n-1} C_i C_{n-i-1} = \frac{4n-2}{n+1} C_{n-1}$	
Stirling 1st kind	$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1, \begin{bmatrix} n \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ n \end{bmatrix} = 0, \begin{bmatrix} n \\ k \end{bmatrix} = (n-1) \begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$	#perms of $n$ objs with exactly $k$ cycles
Stirling 2nd kind	$\left\{ \begin{matrix} n \\ 1 \end{matrix} \right\} = \left\{ \begin{matrix} n \\ n \end{matrix} \right\} = 1, \left\{ \begin{matrix} n \\ k \end{matrix} \right\} = k \left\{ \begin{matrix} n-1 \\ k \end{matrix} \right\} + \left\{ \begin{matrix} n-1 \\ k-1 \end{matrix} \right\}$	#ways to partition $n$ objs into $k$ nonempty sets
Euler	$\langle \begin{matrix} n \\ 0 \end{matrix} \rangle = \langle \begin{matrix} n \\ n-1 \end{matrix} \rangle = 1, \langle \begin{matrix} n \\ k \end{matrix} \rangle = (k+1) \langle \begin{matrix} n-1 \\ k \end{matrix} \rangle + (n-k) \langle \begin{matrix} n-1 \\ k-1 \end{matrix} \rangle$	#perms of $n$ objs with exactly $k$ ascents
Euler 2nd Order	$\llbracket \begin{matrix} n \\ k \end{matrix} \rrbracket = (k+1) \llbracket \begin{matrix} n-1 \\ k \end{matrix} \rrbracket + (2n-k-1) \llbracket \begin{matrix} n-1 \\ k-1 \end{matrix} \rrbracket$	#perms of $1, 1, 2, 2, \dots, n, n$ with exactly $k$ ascents
Bell	$B_1 = 1, B_n = \sum_{k=0}^{n-1} B_k \binom{n-1}{k} = \sum_{k=0}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\}$	#partitions of $1..n$ (Stirling 2nd, no limit on $k$ )

#labeled rooted trees

$n^{n-1}$

#labeled unrooted trees

$n^{n-2}$

#forests of  $k$  rooted trees

$\frac{k}{n} \binom{n}{k} n^{n-k}$

$$\sum_{i=1}^n i^2 = n(n+1)(2n+1)/6$$

$$\sum_{i=1}^n i^3 = n^2(n+1)^2/4$$

$$!n = n \times !(n-1) + (-1)^n$$

$$!n = (n-1)(!(n-1) + !(n-2))$$

$$\sum_{i=1}^n \binom{n}{i} F_i = F_{2n}$$

$$\sum_i \binom{n-i}{i} = F_{n+1}$$

$$a \equiv b \pmod{x, y} \Rightarrow a \equiv b \pmod{\text{lcm}(x, y)}$$

$$\sum_{d|n} \phi(d) = n$$

$$ac \equiv bc \pmod{m} \Rightarrow a \equiv b \pmod{\frac{m}{\gcd(c, m)}}$$

$$(\sum_{d|n} \sigma_0(d))^2 = \sum_{d|n} \sigma_0(d)^3$$

$$p \text{ prime} \Leftrightarrow (p-1)! \equiv -1 \pmod{p}$$

$$\gcd(n^a - 1, n^b - 1) = n^{\gcd(a, b)} - 1$$

$$\sigma_x(n) = \prod_{i=0}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$$

$$\sigma_0(n) = \prod_{i=0}^r (a_i + 1)$$

$$\sum_{k=0}^m (-1)^k \binom{n}{k} = (-1)^m \binom{n-1}{m}$$

$$2^{\omega(n)} = O(\sqrt{n})$$

$$\sum_{i=1}^n 2^{\omega(i)} = O(n \log n)$$

$$d = v_i t + \frac{1}{2} a t^2$$

$$v_f^2 = v_i^2 + 2ad$$

$$v_f = v_i + at$$

$$d = \frac{v_i + v_f}{2} t$$

### 7.1 The Twelfold Way

Putting  $n$  balls into  $k$  boxes.

Balls	same	distinct	same	distinct	Remarks
Boxes	same	same	distinct	distinct	
-	$p_k(n)$	$\sum_{i=0}^k \left\{ \begin{matrix} n \\ i \end{matrix} \right\}$	$\binom{n+k-1}{k-1}$	$k^n$	$p_k(n)$ : #partitions of $n$ into $\leq k$ positive parts
size $\geq 1$	$p(n, k)$	$\left\{ \begin{matrix} n \\ k \end{matrix} \right\}$	$\binom{n-1}{k-1}$	$k! \left\{ \begin{matrix} n \\ k \end{matrix} \right\}$	$p(n, k)$ : #partitions of $n$ into $k$ positive parts
size $\leq 1$	$[n \leq k]$	$[n \leq k]$	$\binom{k}{n}$	$n! \binom{k}{n}$	$[cond]$ : 1 if $cond = true$ , else 0

### 7.2 DP Optimizations

- **1D1D**  $dp(x) = \min_{i=1}^{x-1} \{dp(i) + w[i, x]\}$  and follows quadrangle inequality  $w[i, j] + w[i+1, j+1] \leq w[i, j+1] + w[i+1, j]$ . Then arg array is non-decreasing. Construct arg array using binary search.

### 7.3 Some primes

- 7 digits - 2171159, 9368299, 1874351, 9873623, 3934741, 3932941, 4753739, 1251703, 8324893, 5610793
- 8 digits - 59707699, 84765091, 64216913, 36853373, 91814719, 29647939, 99082553, 68007601, 35386633, 91221883
- 9 digits - 267222157, 248334941, 853519241, 879700489, 529560481, 160736231, 308615471, 722344243, 546428819, 528094447
- 12 digits - 744903658181, 805685255317, 901677551977, 645778995493, 951016942451, 743768119319, 463374658853, 390290791217, 730300933471
- 16 digits - 6934008823912991, 6133523110774669, 4707120596051539, 5856250400014373, 5824952666729017, 5619411481414127, 6239941242022171, 3765554534448349, 3773976086888701, 6077904809921143

• **Legendre symbol:**  $\left(\frac{a}{b}\right) = a^{(b-1)/2} \pmod{b}$ ,  $b$  odd prime.

• **Heron's formula:** A triangle with side lengths  $a, b, c$  has area  $\sqrt{s(s-a)(s-b)(s-c)}$  where  $s = \frac{a+b+c}{2}$ .

• **Pick's theorem:** A polygon on an integer grid strictly containing  $i$  lattice points and having  $b$  lattice points on the boundary has area  $i + \frac{b}{2} - 1$ . (Nothing similar in higher dimensions)

• **Euler characteristic:** A finite, connected, planar graph is drawn in the plane without any edge intersections where  $v$  denotes  $|V|$ ,  $e$  denotes  $|E|$  and  $f$  denotes the number of faces, then  $v - e + f = 2$

• **Baby Step Giant Step:** Given a cyclic group  $\mathcal{G}$  of order  $n$ , a generator  $\alpha$  of the group and a group element  $\beta$ , find  $x$  such that  $\alpha^x = \beta$

**Algorithm:**

- Write  $x$  as  $x = im + j$ , where  $m = \lceil \sqrt{n} \rceil$  and  $0 \leq i < m$  and  $0 \leq j < m$ .
- Hence, we have  $\beta(\alpha^{-m})^i = \alpha^j$ .
- $\forall j$  where  $0 \leq j < m$  : calculate  $\alpha^j$  and add them to  $\text{std::unordered\_map<int, int>}$
- $\forall i$  where  $0 \leq i < m$  : check if  $\beta(\alpha^{-m})^i$  exists in the

$\text{std::unordered\_map<int, int>}$  or not

• **Euler's totient:** The number of integers less than  $n$  that are coprime to  $n$  are  $n \prod_{p|n} \left(1 - \frac{1}{p}\right)$  where each  $p$  is a distinct prime factor of  $n$ .

**Calculation of  $\phi(n)$   $\forall n$  where  $2 \leq n < 10^6$**

- In the regular sieve initialize  $\phi(i) = i \forall i$ .
- As soon as a prime  $i$  is found, update  $\phi(j) = \phi(j) - \phi(j)/i$

• **Gauss Generalization and Wilson's theorem:** Let  $p$  be an odd prime and  $\alpha$  be a positive integer, then in  $\mathbb{Z}/(n)$

$$\prod_{k=1}^{\phi(n)} = \begin{cases} 0 & n = 1, \\ -1 & n = 4, p^\alpha, 2p^\alpha, \\ 1 & \text{otherwise} \end{cases}$$

• **Chinese Remainder Theorem:** Given pairwise coprime positive integers  $n_1, n_2, \dots, n_k$  and arbitrary integers  $a_1, a_2, \dots, a_k$ , the system of simultaneous congruences such that

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

has a solution, and the solution is unique modulo  $N = n_1 n_2 \cdots n_k$ . To construct the solution, do the following

1. Compute  $N = n_1 \times n_2 \cdots \times n_k$ .
2. For each  $i = 1, 2, \dots, k$ , compute

$$y_i = \frac{N}{n_i} = n_1 n_2 \cdots n_i n_{i+1} \cdots n_k.$$

3. For each  $i = 1, 2, \dots, k$ , compute  $z_i \equiv y_i^{-1} \pmod{n_i}$  using Euclid's extended algorithm
4. The integer  $x = \sum_{i=1}^k a_i y_i z_i$  is a solution to the system of the congruences and  $x \pmod{N}$  is the unique solution modulo  $N$ .

• **Shoelace Formula for Area of Simple Polygon:** Polygon represented by  $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ , then it's area  $\mathcal{A}$  is

$$\mathcal{A} = \frac{1}{2} \left| \sum_{i=0}^{n-1} x_i (y_{i+1} - y_{i-1}) \right|$$

where  $(i+1) \equiv (i+1) \pmod{n}$   
 where  $(i-1) \equiv (i-1+n) \pmod{n}$

• **Line Intersection Formula:** Given 2 lines

$$\begin{cases} A_1 x + B_1 y + C_1 = 0, \\ A_2 x + B_2 y + C_2 = 0 \end{cases}$$

We find their intersection using Cramer's rule where **Note the minus signs in front of them**

$$x = -\frac{\begin{vmatrix} C_1 & B_1 \\ C_2 & B_2 \end{vmatrix}}{\begin{vmatrix} A_1 & B_1 \\ A_2 & B_2 \end{vmatrix}}, y = -\frac{\begin{vmatrix} A_1 & C_1 \\ A_2 & C_2 \end{vmatrix}}{\begin{vmatrix} A_1 & B_1 \\ A_2 & B_2 \end{vmatrix}},$$

- **Circle-Line Intersection:** Intersection of a circle and a line given by

$$\begin{cases} x^2 + y^2 = r^2 \\ Ax + By + C = 0 \end{cases}$$

If the circle is centered at point  $(x_c, y_c)$ , transform the coordinate system using

$$\begin{aligned} x &= X + x_c \\ y &= Y + y_c \end{aligned}$$

Calculate the point closest to origin  $(x_0, y_0)$ . It's distance from origin is  $d_0 = \frac{|C|}{\sqrt{A^2 + B^2}}$ , therefore Point  $(x_0, y_0)$ ,

$$\begin{aligned} x_0 &= \frac{-AC}{A^2 + B^2} \\ y_0 &= \frac{-BC}{A^2 + B^2} \end{aligned}$$

If  $d_0 < r$ , then there are 2 intersections. If  $d_0 = r$ , then there is only one intersection. If  $d_0 > r$ , no intersection. Calculate  $d = \sqrt{r^2 - \frac{C^2}{A^2 + B^2}}$  and  $m = \sqrt{\frac{d^2}{A^2 + B^2}}$ . The two points of intersections  $(a_x, a_y)$  and  $(b_x, b_y)$  are (if  $d_0 < r$ )

$$\begin{aligned} a_x &= x_0 + B \cdot m, a_y = y_0 - A \cdot m \\ b_x &= x_0 - B \cdot m, b_y = y_0 + A \cdot m \end{aligned}$$

If  $d_0 = r$ , then  $(x_0, y_0)$  is the intersection point which is tangent to the surface.

- **Intersection of Circle and Circle:** Intersection of two circles whose equations are given as follows

$$\begin{cases} x^2 + y^2 = r_1^2 \\ (x - x_2)^2 + (y - y_2)^2 = r_2^2 \end{cases}$$

Subtract these two equations to get the equation of line given as

$$\begin{aligned} Ax + By + C &= 0 \\ A &= -2x_2 \\ B &= -2y_2 \\ C &= x_2^2 + y_2^2 + r_1^2 - r_2^2 \end{aligned}$$

Now, solve this problem for intersection of a line and circle. Now, handle the degenerate case when  $x_2 = y_2 = 0$  and equation of line is  $C = r_1^2 - r_2^2 = 0$ . If the radii of the circles are same, then there are infinitely many intersections, if they differ, then there are no intersections.

- **Konig's theorem:** In any bipartite graph  $G = (L \cup R, E)$ , the number of edges in a maximum matching is equal to the number of vertices in a minimum vertex cover. Let  $U$  be the set of unmatched vertices in  $L$ , and  $Z$  be the set of vertices that are either in  $U$  or are connected to  $U$  by an alternating path. Then  $K = (L \setminus Z) \cup (R \cap Z)$  is the minimum vertex cover.
- **Dilworth's Theorem:** There exists an antichain  $A$ , and a partition of the order into a family  $P$  of chains, such that the number of chains in the partition equals the cardinality of  $A$ .
- **Mirsky's Theorem:** A poset of height  $h$  can be partitioned into  $h$  antichains.
- The number of vertices of a graph is equal to its minimum vertex cover number plus the size of a maximum independent set.

- A minimum Steiner tree for  $n$  vertices requires at most  $n - 2$  additional Steiner vertices.

• **Lagrange polynomial** through points  $(x_0, y_0), \dots, (x_k, y_k)$  is  $L(x) = \sum_{j=0}^k y_j \prod_{\substack{0 \leq m \leq k \\ m \neq j}} \frac{x - x_m}{x_j - x_m}$

- **Hook length formula:** If  $\lambda$  is a Young diagram and  $h_\lambda(i, j)$  is the hook-length of cell  $(i, j)$ , then then the number of Young tableaux  $d_\lambda = n! / \prod h_\lambda(i, j)$ .

• **Moebius inversion formula:** If  $f(n) = \sum_{d|n} g(d)$ , then  $g(n) = \sum_{d|n} \mu(d) f(n/d)$ . If  $f(n) = \sum_{m=1}^n g(\lfloor n/m \rfloor)$ , then  $g(n) = \sum_{m=1}^n \mu(m) f(\lfloor \frac{n}{m} \rfloor)$ .  $\sum_{d|n} \mu(d) = [n = 1]$   $\sum_{i=1}^n \sum_{j=1}^n [gcd(i, j) = 1] = \sum_{d=1}^n \mu(d) \lfloor \frac{n}{d} \rfloor^2$

- **Farey Sequence:**  $F_n$  Sequence of reduced fractions with denominators  $\leq n$ . For neighbors  $\frac{a}{b}$  and  $\frac{c}{d}$ ,  $bc - ad = 1$ .

- #primitive pythagorean triples with hypotenuse  $< n$  approx  $n/(2\pi)$ .

- **Frobenius Number:** largest number which can't be expressed as a linear combination of numbers  $a_1, \dots, a_n$  with non-negative coefficients.  $g(a_1, a_2) = a_1 a_2 - a_1 - a_2$ ,  $N(a_1, a_2) = (a_1 - 1)(a_2 - 1)/2$ .  $g(d \cdot a_1, d \cdot a_2, a_3) = d \cdot g(a_1, a_2, a_3) + a_3(d - 1)$ . An integer  $x > (\max_i a_i)^2$  can be expressed in such a way iff.  $x \mid \gcd(a_1, \dots, a_n)$ .

## 7.4 Markov Chains

A Markov Chain can be represented as a weighted directed graph of states, where the weight of an edge represents the probability of transitioning over that edge in one timestep. Let  $P^{(m)} = (p_{ij}^{(m)})$  be the probability matrix of transitioning from state  $i$  to state  $j$  in  $m$  timesteps,



and note that  $P^{(1)}$  is the adjacency matrix of the graph. **Chapman-Kolmogorov:**  $p_{ij}^{(m+n)} = \sum_k p_{ik}^{(m)} p_{kj}^{(n)}$ . It follows that  $P^{(m+n)} = P^{(m)} P^{(n)}$  and  $P^{(m)} = P^m$ . If  $p^{(0)}$  is the initial probability distribution (a vector), then  $p^{(0)} P^{(m)}$  is the probability distribution after  $m$  timesteps.

The return times of a state  $i$  is  $R_i = \{m \mid p_{ii}^{(m)} > 0\}$ , and  $i$  is *aperiodic* if  $\gcd(R_i) = 1$ . A MC is aperiodic if any of its vertices is aperiodic. A MC is *irreducible* if the corresponding graph is strongly connected.

A distribution  $\pi$  is stationary if  $\pi P = \pi$ . If MC is irreducible then  $\pi_i = 1/\mathbb{E}[T_i]$ , where  $T_i$  is the expected time between two visits at  $i$ .  $\pi_j/\pi_i$  is the expected number of visits at  $j$  in between two consecutive visits at  $i$ . A MC is *ergodic* if  $\lim_{m \rightarrow \infty} p^{(0)} P^m = \pi$ . A MC is ergodic iff. it is irreducible and aperiodic.

A MC for a random walk in an undirected weighted graph (unweighted graph can be made weighted by adding 1-weights) has  $p_{uv} = w_{uv}/\sum_x w_{ux}$ . If the graph is connected, then  $\pi_u = \sum_x w_{ux}/\sum_v \sum_x w_{vx}$ . Such a random walk is aperiodic iff. the graph is not bipartite.

An *absorbing* MC is of the form  $P = \begin{pmatrix} Q & R \\ 0 & I_r \end{pmatrix}$ . Let  $N = \sum_{m=0}^{\infty} Q^m = (I_t - Q)^{-1}$ . Then, if starting in state  $i$ , the expected number of steps till absorption is the  $i$ -th entry in  $N1$ . If starting in state  $i$ , the probability of being absorbed in state  $j$  is the  $(i, j)$ -th entry of  $NR$ .

Many problems on MC can be formulated in terms of a system of recurrence relations, and then solved using Gaussian elimination.

## 7.5 Burnside's Lemma

Let  $G$  be a finite group that acts on a set  $X$ . For each  $g$  in  $G$  let  $X^g$  denote the set of elements in

$X$  that are fixed by  $g$ . Then the number of orbits

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

$$Z(S_n) = \frac{1}{n} \sum_{l=1}^n a_l Z(S_{n-l})$$

## 7.6 Bezout's identity

If  $(x, y)$  is any solution to  $ax + by = d$  (e.g. found by the Extended Euclidean Algorithm), then all solutions are given by

$$\left( x + k \frac{b}{\gcd(a, b)}, y - k \frac{a}{\gcd(a, b)} \right)$$

## 7.7 Misc

### 7.7.1 Determinants and PM

$$\det(A) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^n a_{i, \sigma(i)}$$

$$\text{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^n a_{i, \sigma(i)}$$

$$\begin{aligned} pf(A) &= \frac{1}{2^n n!} \sum_{\sigma \in S_{2n}} \text{sgn}(\sigma) \prod_{i=1}^n a_{\sigma(2i-1), \sigma(2i)} \\ &= \sum_{M \in \text{PM}(n)} \text{sgn}(M) \prod_{(i,j) \in M} a_{i,j} \end{aligned}$$

### 7.7.2 BEST Theorem

Count directed Eulerian cycles. Number of OST given by Kirchoff's Theorem (remove r/c with root)  $\# \text{OST}(G, r) \cdot \prod_v (d_v - 1)!$

### 7.7.3 Primitive Roots

Only exists when  $n$  is  $2, 4, p^k, 2p^k$ , where  $p$  odd prime. Assume  $n$  prime. Number of primitive roots  $\phi(\phi(n))$ . Let  $g$  be primitive root. All primitive roots are of the form  $g^k$  where  $k, \phi(p)$  are coprime.

$k$ -roots:  $g^{i \cdot \phi(n)/k}$  for  $0 \leq i < k$

**How to find a primitive root?** To test that  $a$  is a primitive root of  $p$  you need to do the following. First, let  $s = \phi(p)$  where  $\phi()$  is [the Euler's totient function][1]. If  $p$  is prime, then  $s = p - 1$ . Then you need to determine all the prime factors of  $s$ :  $p_1, \dots, p_k$ . Finally, calculate  $a^{s/p_i} \bmod p$  for all  $i = 1 \dots k$ , and if you find 1 among residuals then it is NOT a primitive root, otherwise it is.

So, basically you need to calculate and check  $k$  numbers where  $k$  is the number of different prime factors in  $\phi(p)$ .

### 7.7.4 Sum of primes

For any multiplicative  $f$ :

$$S(n, p) = S(n, p-1) - f(p) \cdot (S(n/p, p-1) - S(p-1, p-1))$$

### 7.7.5 Floor

$$\begin{aligned} \lfloor \lfloor x/y \rfloor / z \rfloor &= \lfloor x/(yz) \rfloor \\ x \% y &= x - y \lfloor x/y \rfloor \end{aligned}$$