ButterRoti ICPC Team Notebook (2017-18)

Contents

1 Misc

1.1 Build

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

1.2 Snippet

```cpp
#include <bits/stdc++.h>

using namespace std;

#define SYNC std::ios::sync_with_stdio(false);
    cin.tie(NULL);

template<typename T> using V = vector<T>;
template<typename T, typename V> using P = pair<T,
    V>;

using LL = long long;
using ll = LL;
using LD = long double;
using ld = long double;

#define fi first
#define ff first
#define se second
#define ss second
#define endl '\n'
#define FOR0(i,n) for(int i=0, _##i=(n); i<_##i;
    ++i)
#define FOR(i,l,r) for(int i=(l), _##i=(r); i<_##i
    ; ++i)
#define FORD(i,l,r) for(int i=(r), _##i=(l); --i>=
    _##i; )
#define dzx cerr << "here";
#define her cerr << "HERE\n"

const int MOD = (int)1e9 + 7, inf = 1e9;
const ll INF = 1e18;
```

```cpp
int32_t main() {SYNC;

    return 0;
}
```

1.3 Stack Size Increase

```cpp
#include <sys/resource.h>

int main(){
  rlimit R;
  getrlimit(RLIMIT_STACK, &R);
  R.rlim_cur = R.rlim_max;
  setrlimit(RLIMIT_STACK, &R);
}
```

1.4 Variadic Multiplication and Addition

```cpp
const int MOD = (int)1e9 + 7;

int add(){ return 0; }

template<typename... T> int add(int a, T... arg){
  int b = add(arg...);
  return (a + b >= MOD ? a + b - MOD : a + b);
}

int multiply(){return 1;}

template<typename... Args> int multiply(int a,
   Args... arg){
  return (a * 1LL * multiply(arg...)) % MOD;
}
```

2 Combinatorial optimization

2.1 Lowest Common Ancestor

```cpp
// 0-based vertex indexing. memset to -1
int log(int t){
  int res = 1;
  for(; 1 << res <= t; res++);
  return res;
}
```

```cpp
int lca(int u , int v){
  if(h[u] < h[v])swap(u , v);
  int L = log(h[u]);
  for(int i = L - 1; i >= 0; i--){
    if(par[u][i] + 1 && h[u] - (1 << i) >= h[v])
      u = par[u][i];
  }
  if(v == u) return u;
  for(int i = L - 1; i >= 0; i--){
    if(par[u][i] + 1 && par[u][i] != par[v][i]){
      u = par[u][i]; v = par[v][i];
    }
  }
  return par[u][0];
}
```

2.2 Heavy-Light Decomposition

```cpp
V<V<int> > g,chains;
V<int> value,cpar,cid,id,depth;
V<SegTree<int>> trees;
int dfs(int c,int p){
  depth[c]=depth[p]+1;
  int sz=1;
  auto it=find(g[c].begin(),g[c].end(),p);
  if(it!=g[c].end())
    g[c].erase(it);
  if(g[c].empty())
    return 1;
  int mx=0;
  for(auto &i:g[c]){
    int cur=dfs(i,c);
    sz+=cur;
    if(cur>mx)
      mx=cur,swap(i,g[c][0]);
  }
  return sz;
}
void form_chains(int c){
  cid[c]=(int)chains.size()-1;
  id[c]=(int)chains.back().size();
  chains.back().pb(value[c]);
  for(int i=0;i<(int)g[c].size();i++){
    if(i)
      chains.pb({}),cpar.pb(c);
```

```
28        form_chains(g[c][i]);
29     }
30   if(g[c].empty())
31     trees.pb(SegTree<int>([](int a,int b){return
          max(a,b);},0,(int)chains.back().size(),
          chains.back()));
32 }
33 void update(int v,int val){
34   trees[cid[v]].update(id[v],val);
35 }
36 int query(int u,int v){
37   int r=0;
38   while(u!=v){
39     if(cid[v]==cid[u]){
40       if(depth[v]<depth[u])
41         swap(v,u);
42       r=max(r,trees[cid[v]].query(id[u]+1,id[v]));
43       v=u;
44     }
45     else{
46       if(depth[cpar[cid[v]]]<depth[cpar[cid[u]]])
47         swap(v,u);
48       r=max(r,trees[cid[v]].query(0,id[v]));
49       v=cpar[cid[v]];
50     }
51   }
52   return r;
53 }
```

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);
```

```cpp
        low[u] = min(low[u] , low[i]);
        update(u, i, child);
      }
    else if(i != par[u]) {
        low[u] = min(low[u] , dis[i]);
      }
    }
  }
}
```

2.5   Biconnected Components

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = (int)2e5 + 10;

vector<vector<int>> tree, g;
bool isBridge[N << 2], vis[N];
int Time, arr[N], U[N], V[N], cmpno, comp[N];
vector<int> temp; //temp stores component values

int adj(int u, int e){
  return (u == U[e] ? V[e] : U[e]);
}

int find_bridge(int u , int edge){
  vis[u] = true;
  arr[u] = Time++;
  int x = arr[u];

  for(auto & i : g[u]){
    int v = adj(u, i);
    if(!vis[v]){
      x = min(x,find_bridge(v, i));
    }
    else if(i != edge){
      x = min(x,arr[v]);
    }
  }

  if(x == arr[u] && edge != -1){
    isBridge[edge] = true;
  }
  return x;
}

void dfs1(int u){
```

```cpp
  int current = cmpno;
  queue<int> q;
  q.push(u);
  vis[u] = 1;
  temp.push_back(current);

  while(!q.empty()){
    int v = q.front();
    q.pop();
    comp[v] = current;

    for(auto & i : g[v]) {
      int w = adj(v, i);
      if(vis[w])continue;
      if(isBridge[i]){
        cmpno++;
        tree[current].push_back(cmpno);
        tree[cmpno].push_back(current);
        dfs1(w);
      }
      else{
        q.push(w);
        vis[w] = 1;
      }
    }
  }
}

int main(){
  int n, m;
  cin >> n >> m;
  g.resize(n + 2); tree.resize(n + 2);

  for(int i = 0; i < m; i ++){
    cin >> U[i] >> V[i];
    g[U[i]].push_back(i);
    g[V[i]].push_back(i);
  }

  cmpno = Time = 0;
  memset(vis, false, sizeof vis);

  for(int i = 0; i < n; i ++){
    if(!vis[i]){
      find_bridge(i , -1);
    }
  }
```

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

2.6   2-SAT

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

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

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
          (_v), c(_c), f(_f) { }
7   };
8   int n;
9   vector<edge> edges;
10  vector<vector<int> > g;
11  vector<int> d, pt;
12
```

```cpp
void addEdge(int u, int v, ll c, ll f = 0) {
  g[u].emplace_back(edges.size());
  edges.emplace_back(edge(u,v,c,f));
  g[v].emplace_back(edges.size());
  edges.emplace_back(edge(v,u,0,0));
}
bool bfs(int s, int t) {
  queue<int> q({s});
  d.assign(n+1, n+2);
  d[s] = 0;
  while(!q.empty()) {
    int u = q.front(); q.pop();
    if (u == t) break;
    for(int k : g[u]) {
      edge &e = edges[k];
      if(e.f < e.c && d[e.v] > d[e.u] + 1){
        d[e.v] = d[e.u] + 1;
        q.push(e.v);
      }
    }
  }
  return d[t] < n+2;
}

ll dfs(int u, int t, ll flow = -1) {
  if(u == t || !flow) return flow;
  for(int &i = pt[u]; i < (int)(g[u].size()); i++)
      {
    edge &e = edges[g[u][i]], &oe=edges[g[u][i
      ]^1];
    if(d[e.v] == d[e.u] + 1) {
      ll amt = e.c - e.f;
      if (flow != -1 && amt > flow) amt = flow;
      if(ll pushed = dfs(e.v,t,amt)) {
        e.f += pushed;
        oe.f -= pushed;
        return pushed;
      }
    }
  }
  return 0;
}

ll flow(int s, int t) {
  ll ans = 0;
  while(bfs(s,t)) {
    pt.assign(n+1, 0);
    while(ll val = dfs(s,t)) ans += val;
  }
  return ans;
}
```

2.8   Min Cost Max Flow

```cpp
class CostFlowGraph{
public:
  struct Edge{
    int v,f,c;
    Edge(){}
    Edge(int v,int f,int c):v(v),f(f),c(c){}
  };
  V<V<int> > g;
  V<Edge> e;
  V<int> pot;
  int n;
  int flow;
  int cost;
  CostFlowGraph(int sz){
    n=sz;
    g.resize(n);
    pot.assign(n,0);
    flow=0;
    cost=0;
  }
  void addEdge(int u,int v,int cap,int c){
    g[u].pb((int)e.size());
    e.pb(Edge(v,cap,c));
    g[v].pb((int)e.size());
    e.pb(Edge(u,0,-c));
  }
  void assignPots(int s){
    priority_queue<pii,V<pii>,greater<pii>> q;
    V<int> npot(n,inf);
    q.push({s,0});
    while(!q.empty()){
      auto cur=q.top();q.pop();
      if(npot[cur.fi]<=cur.se)
        continue;
      npot[cur.fi]=cur.se;
      for(auto i:g[cur.fi]) if(e[i].f>0){
        int cst=pot[cur.fi]-pot[e[i].v]+e[i].c;
        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
            >0 && pot[e[i^1].v]!=inf){
50          pot[e[i].v]=min(pot[e[i].v],pot[e[i^1].v]+
              e[i].c);
51        }
52    }
53    int augment(int s,int t,int fl,V<bool> &v){
54      if(s==t)
55        return fl;
56      v[s]=1;
57      for(auto i:g[s])   if(!v[e[i].v] && e[i].f>0 &&
            (pot[s]-pot[e[i].v]+e[i].c)==0){
58        int cf=augment(e[i].v,t,min(fl,e[i].f),v);
59        if(cf!=0){
60          e[i].f-=cf;
61          e[i^1].f+=cf;
62          return cf;
63        }
64      }
65      return 0;
66    }
67    void mcf(int s,int t,bool neg=0){
68      int cur=0;
69      V<bool> vis;
70      if(neg)
71        negativeEdges(s);
72      do{
73        vis.assign(n,0);
74        flow+=cur;
75        cost+=(pot[t]-pot[s]);
76        assignPots(s);
77        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;
       if (i == phase-1) {
23   for (int j = 0; j < N; j++) weights[prev][j] +=
         weights[last][j];
24   for (int j = 0; j < N; j++) weights[j][prev] =
         weights[prev][j];
25   used[last] = true;
26   cut.push_back(last);
27   if (best_weight == -1 || w[last] < best_weight)
       {
28     best_cut = cut;
29     best_weight = w[last];
30   }
31       } else {
32   for (int j = 0; j < N; j++)
33     w[j] += weights[last][j];
34   added[last] = true;
35       }
36     }
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
             << endl;
55    }
56  }
```

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
37 bool dfs(int u) {
38   int i, v, len;
```

```
39    if(u!=NIL) {
40        len = G[u].size();
41        for(i=0; i<len; i++) {
42            v = G[u][i];
43            if(dist[match[v]]==dist[u]+1) {
44                if(dfs(match[v])) {
45                    match[v] = u;
46                    match[u] = v;
47                    return true;
48                }
49            }
50        }
51        dist[u] = INF;
52        return false;
53    }
54    return true;
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,
      VI &Rmate) {
13   int n = int(cost.size());
14
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],
        cost[i][j]);
21   }
22   for (int j = 0; j < n; j++) {
23     v[j] = cost[0][j] - u[0];
24     for (int i = 1; i < n; i++) v[j] = min(v[j],
        cost[i][j] - u[i]);
25   }
26
27   // construct primal solution satisfying
      complementary slackness
28   Lmate = VI(n, -1);
29   Rmate = VI(n, -1);
30   int mated = 0;
31   for (int i = 0; i < n; i++) {
32     for (int j = 0; j < n; j++) {
33       if (Rmate[j] != -1) continue;
34       if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10)
          {
35         Lmate[i] = j;
36         Rmate[j] = i;
37         mated++;
38         break;
39       }
40     }
41   }
42
43   VD dist(n);
44   VI dad(n);
45   VI seen(n);
46
47   // repeat until primal solution is feasible
48   while (mated < n) {
49
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
63      // find closest
64      j = -1;
65      for (int k = 0; k < n; k++) {
66    if (seen[k]) continue;
67    if (j == -1 || dist[k] < dist[j]) j = k;
68      }
69      seen[j] = 1;
70
71      // termination condition
72      if (Rmate[j] == -1) break;
73
74      // relax neighbors
75      const int i = Rmate[j];
76      for (int k = 0; k < n; k++) {
77    if (seen[k]) continue;
78    const double new_dist = dist[j] + cost[i][k] -
         u[i] - v[k];
79    if (dist[k] > new_dist) {
80      dist[k] = new_dist;
81      dad[k] = j;
82    }
83      }
84    }
85    // update dual variables
86    for (int k = 0; k < n; k++) {
87      if (k == j || !seen[k]) continue;
88      const int i = Rmate[k];
89      v[k] += dist[k] - dist[j];
90      u[i] -= dist[k] - dist[j];
91    }
92    u[s] += dist[j];
93
94    // augment along path
95    while (dad[j] >= 0) {
96      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
104   mated++;
105 }
106
107 double value = 0;
108 for (int i = 0; i < n; i++)
109   value += cost[i][Lmate[i]];
110
111 return value;
112 }
```

2.13 Link

```
1  struct Node { // Splay tree. Root's pp contains
      tree's parent.
2    Node *p = 0, *pp = 0, *c[2];
3    bool flip = 0;
4    Node() { c[0] = c[1] = 0; fix(); }
5    void fix() {
6      if (c[0]) c[0]->p = this;
7      if (c[1]) c[1]->p = this;
8      // (+ update sum of subtree elements etc.
           if wanted)
9    }
10   void push_flip() {
11     if (!flip) return;
12     flip = 0; swap(c[0], c[1]);
13     if (c[0]) c[0]->flip ^= 1;
14     if (c[1]) c[1]->flip ^= 1;
15   }
16   int up() { return p ? p->c[1] == this : -1; }
17   void rot(int i, int b) {
18     int h = i ^ b;
19     Node *x = c[i], *y = b == 2 ? x : x->c[h],
           *z = b ? y : x;
20     if ((y->p = p)) p->c[up()] = y;
21     c[i] = z->c[i ^ 1];
22     if (b < 2) {
23       x->c[h] = y->c[h ^ 1];
24       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.
        Always finishes without flip set.
32        for (push_flip(); p; ) {
33          if (p->p) p->p->push_flip();
34          p->push_flip(); push_flip();
35          int c1 = up(), c2 = p->up();
36          if (c2 == -1) p->rot(c1, 2);
37          else p->p->rot(c2, c1 != c2);
38        }
39      }
40    Node* first() { /// Return the min element of
        the subtree rooted at this, splayed to the
        top.
41        push_flip();
42        return c[0] ? c[0]->first() : (splay(),
          this);
43      }
44  };
45
46  struct LinkCut {
47    vector<Node> node;
48    LinkCut(int N) : node(N) {}
49
50    void link(int u, int v) { // add an edge (u, v
        )
51      assert(!connected(u, v));
52      make_root(&node[u]);
53      node[u].pp = &node[v];
54    }
55
56    void cut(int u, int v) { // remove an edge (u,
        v)
57      Node *x = &node[u], *top = &node[v];
58      make_root(top); x->splay();
59      assert(top == (x->pp ?: x->c[0]));
60      if (x->pp) x->pp = 0;
61      else {
62        x->c[0] = top->p = 0;
63        x->fix();
64      }
65    }
66
67    bool connected(int u, int v) { // are u, v in
```

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

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) +
        1 + sz(t->r); }
11  void lazy(pnode t) {
12      if(!t || !t->lazy) return;
13      t->val+=t->lazy;
14      t->sum+=t->lazy*sz(t);
15      if(t->l)t->l->lazy+=t->lazy;
16      if(t->r)t->r->lazy+=t->lazy;
17      t->lazy = 0;
18  }
19  void reset(pnode t) {
20      if(t) t->sum=t->val;
21  }
22  void combine(pnode& t, pnode l, pnode r) {
23      if(!l || !r) return void(t=l?l:r);
24      t->sum = l->sum + r->sum;
25  }
26  void operation(pnode t) {
27      if(!t) return;
28      reset(t);
29      lazy(t->l); lazy(t->r);
30      combine(t,t->l,t); combine(t,t,t->r);
31  }
32  void split(pnode t, pnode& l, pnode& r, int pos,
        int add = 0) {
33      if(!t) return void(l=r=NULL);
34      lazy(t); int curr_pos = add + sz(t->l);
35      if(curr_pos<pos) split(t->r,t->r,r,pos,
          curr_pos+1),l=t;
36      else split(t->l,l,t->r,pos,add),r=t;
37      upd_sz(t); operation(t);
38  }
39  void merge(pnode& t, pnode l, pnode r) {
40      lazy(l); lazy(r);
41      if(!l || !r) t = l?l:r;
42      else if(l->prior > r->prior) merge(l->r,l->r,r
          ),t=l;
43      else merge(r->l, l, r-> l), t=r;
44      upd_sz(t); operation(t);
45  }
46  pnode init(int val) {
47      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));
            merge(t,l,t);
68  }
```

3.2 Segment Tree

```
1  // This code solves problem Help Ashu on
       hackerearth
2  // Iterative segment tree supporting non
       commutative combiner function
3  // The combiner function and identity of the
       combiner function are taken as contructor
       arguments
4  // Assign the initial input into t[size] to t[2*
       size-1] then call build
5  // Memory 2*size*sizeof(T)
6  // Time complexity O(log(size))
7  #include <bits/stdc++.h>
8  using namespace std;
9  /* Equinox */
10  template<typename T>
11  class SegTree{
12  public:
13      vector<T> t;
14      T identity;
15      T (*combine)(T,T);
16      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]=
        combine(t[i<<1],t[i<<1|1]);}
24    T query(int l,int r){
25      T lt=identity;
26      T rt=identity;
27      for(l+=size,r+=size;l<=r;r>>=1,l>>=1){
28        if(l&1)    lt=combine(lt,t[l++]);
29        if(!(r&1))  rt=combine(t[r--],rt);
30      }
31      return combine(lt,rt);
32    }
33    void update(int p,T v){for(t[p+=size]=v;p>>=1;)t
        [p]=combine(t[p<<1],t[p<<1|1]);}
34  };
35  int32_t main(){
36    int n;
37    cin>>n;
38    SegTree<int> tree([](int a,int b){return a+b
        ;},0,n);
39    for(int i=0;i<n;i++){
40      int a;
41      cin>>a;
42      tree.t[i+n]=a&1;
43    }
44    tree.build();
45    int q;
46    cin>>q;
47    while(q--){
48      int c,x,y;
49      cin>>c>>x>>y;
50      switch(c){
51        case 0:
52        tree.update(x-1,y&1);
53        break;
54        case 1:
55        cout<<(y-x+1)-tree.query(x-1,y-1)<<"\n";
56        break;
57        case 2:
58        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
     supporting non commutative combiner functions
3  // The combiner function and identity of the
     combiner function are taken as contructor
     arguments
4  // Also the function for application of lazy nodes
      onto tree nodes is taken as parameter along
     with Zero of lazy node
5  // Assign the initial input into t[size] to t[2*
     size-1] then call build
6  // Memory 2*size*sizeof(T)+2*size*sizeof(L)
7  // Time complexity O(log(size))
8  #include <bits/stdc++.h>
9  using namespace std;
10 /* Equinox */
11 template<typename T,typename L>
12 class SegTree{
13 public:
14   vector<T> t;
15   vector<T> lz;
16   T identity;
17   L zero;
18   T (*combine)(T,T);
19   void (*apply)(T&,L&,L&,int k);
20   int size;
21   int height;
22   SegTree(T (*op)(T,T),T e,void (*pro)(T&,L&,L&,
        int k),L z,int n){
23     combine=op;
24     apply=pro;
25     identity=e;
26     zero=z;
27     t.assign(2*n,e);
28     lz.assign(2*n,z);
29     size=n;
30     height = sizeof(int)*8-__builtin_clz(n);
31   }
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 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 }
```

4.4 Large Modulo Multiplication

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

4.5 Segmented Sieve

```
1  // Segmented Seive
2  // N=sqrt(b)
3  // Time complexity: O(N.log(B-A))
4  #define A 1000000000000LL
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 }
```

4.6 Miller Rabin

```
1  V<int> A{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
     37, 41};
2
3  bool Miller(long long p){
4    if(p < 2){
5      return false;
6    }
7    if(p != 2 && p % 2 == 0){
8      return false;
9    }
10   long long s = p - 1;
11   while(s % 2 == 0){
12     s /= 2;
```

```
13      }
14      for(auto & a : A){
15          long long temp = s;
16          long long mod = power(a, temp, p);
17          while(temp != p - 1 && mod != 1 && mod != p-1)
                    {
18              mod = mulmod(mod, mod, p);
19              temp *= 2;
20          }
21          if(mod != p - 1 && temp % 2 == 0){
22              return false;
23          }
24      }
25      return true;
26  }
```

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 #include bits/stdc++.h
2 using namespace std;
3
4 // suffixRank is table hold the rank of each
      string on each iteration
5 // suffixRank[i][j] denotes rank of jth suffix at
      ith iteration
6
7 int suffixRank[20][int(1E6)];
8
9 // Example "abaab"
10 // Suffix Array for this (2, 3, 0, 4, 1)
11 // Create a tuple to store rank for each suffix
12
13 struct myTuple {
```

```
14      int originalIndex;      // stores original index
                                //    of suffix
15      int firstHalf;          // store rank for first
                                //    half of suffix
16      int secondHalf;         // store rank for second
                                //    half of suffix
17  };
18
19
20  // function to compare two suffix in O(1)
21  // first it checks whether first half chars of 'a'
        are equal to first half chars of b
22  // if they compare second half
23  // else compare decide on rank of first half
24
25  int cmp(myTuple a, myTuple b) {
26      if(a.firstHalf == b.firstHalf) return a.
            secondHalf < b.secondHalf;
27      else return a.firstHalf < b.firstHalf;
28  }
29
30  int main() {
31
32      // Take input string
33      // initialize size of string as N
34
35      string s; cin >> s;
36      int N = s.size();
37
38      // Initialize suffix ranking on the basis of
            only single character
39      // for single character ranks will be 'a' = 0,
            'b' = 1, 'c' = 2 ... 'z' = 25
40
41      for(int i = 0; i < N; ++i)
42          suffixRank[0][i] = s[i] - 'a';
43
44      // Create a tuple array for each suffix
45
46      myTuple L[N];
47
48      // Iterate log(n) times i.e. till when all the
            suffixes are sorted
49      // 'stp' keeps the track of number of
            iteration
50      // 'cnt' store length of suffix which is going
            to be compared
51
52      // On each iteration we initialize tuple for
            each suffix array
53      // with values computed from previous
            iteration
54
55      for(int cnt = 1, stp = 1; cnt < N; cnt *= 2,
            ++stp) {
56
57          for(int i = 0; i < N; ++i) {
58              L[i].firstHalf = suffixRank[stp - 1][i
                    ];
59              L[i].secondHalf = i + cnt < N ?
                    suffixRank[stp - 1][i + cnt] : -1;
60              L[i].originalIndex = i;
61          }
62
63          // On the basis of tuples obtained sort
                the tuple array
64
65          sort(L, L + N, cmp);
66
67          // Initialize rank for rank 0 suffix after
                sorting to its original index
68          // in suffixRank array
69
70          suffixRank[stp][L[0].originalIndex] = 0;
71
72          for(int i = 1, currRank = 0; i < N; ++i) {
73
74              // compare ith ranked suffix ( after
                    sorting ) to (i - 1)th ranked
                    suffix
75              // if they are equal till now assign
                    same rank to ith as that of (i - 1)
                    th
76              // else rank for ith will be currRank
                    ( i.e. rank of (i - 1)th ) plus 1,
                    i.e ( currRank + 1 )
77
78              if(L[i - 1].firstHalf != L[i].
                    firstHalf || L[i - 1].secondHalf !=
                    L[i].secondHalf)
79                  ++currRank;
80
81              suffixRank[stp][L[i].originalIndex] =
                    currRank;
```

```cpp
82            }
83
84        }
85
86        // Print suffix array
87
88        for(int i = 0; i < N; ++i) cout << L[i].
              originalIndex << endl;
89
90        return 0;
91 }
```

5.3 Suffix Tree

```cpp
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
17        // if we're not, find the next edge. If it
            doesn't exist, create a leaf and add
             it to the tree
18        if (t[tv][c]==-1) {t[tv][c]=ts;l[ts]=la;p[
            ts++]=tv;tv=s[tv];tp=r[tv]+1;goto suff
            ;}
19      tv=t[tv][c];tp=l[tv];
20    } // otherwise just proceed to the next edge
21    if (tp==-1 || c==a[tp]-'a')
22        tp++; // if the letter on the edge equal c
            , go down that edge
23    else {
24        // otherwise split the edge in two with
            middle in node ts
25        l[ts]=l[tv];r[ts]=tp-1;p[ts]=p[tv];t[ts][a
            [tp]-'a']=tv;
26        // 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');
59 }
```

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 <=
                max(p[i].x, p[j].x))
34              return true;
35      }else{
36          bool below = p[i].y < pt.y;
37          if(below != (p[j].y < pt.y)){
38              auto orientation = ccw(p[i], p[j], pt);
39              if(!orientation) return true;
40              if(below == (orientation > 0)) cnt +=
                    below ? 1 : -1;
41          }
42      }
43     }
44
45     return (cnt != 0);
46 }
```

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
13                 break;
14         }
15         hull.pb(pts[i]);
16     }
17     return move(hull);
18 }
19 vector<Point> convex_hull(vector<Point> &pts){
20     sort(pts.begin(), pts.end(),[](Point &a,Point
          &b){
21         if(a.x==b.x)
22             return a.y<b.y;
23         return a.x<b.x;
24     });
```

```
25        vector<Point> uh,lh;
26        uh=half_hull(pts,1);
27        lh=half_hull(pts,-1);
28        lh.pop_back();
29        reverse(lh.begin(), lh.end());
```

```
30        uh.insert(uh.end(),lh.begin(), lh.end());
31        return move(uh);
32 }
```

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	$\begin{Bmatrix} n \\ 1 \end{Bmatrix} = \begin{Bmatrix} n \\ n \end{Bmatrix} = 1, \begin{Bmatrix} n \\ k \end{Bmatrix} = k\begin{Bmatrix} n-1 \\ k \end{Bmatrix} + \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix}$	#ways to partition n objs into k nonempty sets
Euler	$\left\langle \begin{matrix} n \\ 0 \end{matrix} \right\rangle = \left\langle \begin{matrix} n \\ n-1 \end{matrix} \right\rangle = 1, \left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle = (k+1)\left\langle \begin{matrix} n-1 \\ k \end{matrix} \right\rangle + (n-k)\left\langle \begin{matrix} n-1 \\ k-1 \end{matrix} \right\rangle$	#perms of n objs with exactly k ascents
Euler 2nd Order	$\left\langle\!\!\left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle\!\!\right\rangle = (k+1)\left\langle\!\!\left\langle \begin{matrix} n-1 \\ k \end{matrix} \right\rangle\!\!\right\rangle + (2n-k-1)\left\langle\!\!\left\langle \begin{matrix} n-1 \\ k-1 \end{matrix} \right\rangle\!\!\right\rangle$	#perms of $1,1,2,2,...,n,n$ with exactly k ascents
Bell	$B_1 = 1, B_n = \sum_{k=0}^{n-1} B_k \binom{n-1}{k} = \sum_{k=0}^{n} \begin{Bmatrix} n \\ k \end{Bmatrix}$	#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	$\frac{k}{n}\binom{n}{k}n^{n-k}$
$\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$	$\sum_{i=1}^{n} i^3 = n^2(n+1)^2/4$
$!n = n \times !(n-1) + (-1)^n$	$!n = (n-1)(!(n-1) + !(n-2))$
$\sum_{i=1}^{n} \binom{n}{i} F_i = F_{2n}$	$\sum_i \binom{n-i}{i} = F_{n+1}$
$a \equiv b \pmod{x, y} \Rightarrow a \equiv b \pmod{\text{lcm}(x, y)}$	$\sum_{d|n} \phi(d) = n$
$ac \equiv bc \pmod{m} \Rightarrow a \equiv b \pmod{\frac{m}{\gcd(c,m)}}$	$(\sum_{d|n} \sigma_0(d))^2 = \sum_{d|n} \sigma_0(d)^3$
$p \text{ prime} \Leftrightarrow (p-1)! \equiv -1 \pmod{p}$	$\gcd(n^a - 1, n^b - 1) = n^{\gcd(a,b)} - 1$
$\sigma_x(n) = \prod_{i=0}^{r} \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$	$\sigma_0(n) = \prod_{i=0}^{r}(a_i + 1)$
$\sum_{k=0}^{m}(-1)^k \binom{n}{k} = (-1)^m \binom{n-1}{m}$	
$2^{\omega(n)} = O(\sqrt{n})$	$\sum_{i=1}^{n} 2^{\omega(i)} = O(n \log n)$
$d = v_i t + \frac{1}{2}at^2$	$v_f^2 = v_i^2 + 2ad$
$v_f = v_i + at$	$d = \frac{v_i + v_f}{2}t$

7.1 The Twelvefold Way

Putting n balls into k boxes.

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

7.2 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:** $\left(\frac{a}{b}\right) = a^{(b-1)/2}$ (mod 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 α of the group and a group element β, find x such that $\alpha^x = \beta$
  Algorithm:

  - Write x as $x = im + j$, where $m = \lceil \sqrt{n} \rceil$ and $0 \le i < m$ and $0 \le j < m$.
  - Hence, we have $\beta(\alpha^{-m})^i = \alpha^j$.
  - $\forall j$ where $0 \le j < m$: calculate α^j and add them to `std::unordered_map<int, int>`
  - $\forall i$ where $0 \le i < m$: check if $\beta(\alpha^{-m})^i$ exists in the `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 $2 \le n < 10^6$

  - 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 α 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 $n_1, n_2, \cdots, n_k$ and arbitrary integers $a_1, a_2, \cdots, a_k$, the system of simultaneous congruences such that

$$x \equiv a_1 \pmod{n_1}$$
$$x \equiv a_2 \pmod{n_2}$$
$$\vdots$$
$$x \equiv a_k \pmod{n_k}$$

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 $i = 1, 2, \cdots, k$, compute

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

  3. For each $i = 1, 2, \cdots k$, 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 mod N is the unique solution modulo N.

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

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

- **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

$$x = X + x_c$$
$$y = Y + y_c$$

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

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

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 $d = \sqrt{r^2 - \dfrac{C^2}{A^2 + B^2}}$ and $m = \sqrt{\dfrac{d^2}{A^2 + B^2}}$. The two points of intersections (a_x, a_y) and (b_x, b_y) are (if $d_0 < r$)

$$a_x = x_0 + B \cdot m, a_y = y_0 - A \cdot m$$
$$b_x = x_0 - B \cdot m, b_y = y_0 + A \cdot m$$

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

$$Ax + By + C = 0$$
$$A = -2x_2$$
$$B = -2y_2$$
$$C = x_2^2 + y_2^2 + r_1^2 - r_2^2$$

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

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

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

- **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 $g(n) = \sum_{m=1}^{n} \mu(m) f(\lfloor \frac{n}{m} \rfloor)$.
$\sum_{d|n} \mu(d) = [n = 1]$
$\sum_{i=1}^{n} \sum_{j=1}^{n} [gcd(i, j) = 1] = \sum_{d=1}^{n} \mu(d) \lfloor \frac{n}{d} \rfloor^2$

- #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 $a_1, \ldots, a_n$ 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. $x \mid \gcd(a_1, \ldots, a_n)$.

7.3 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 π 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. π_j/π_i is the expected number of visits at j in between two consecutive visits at i. A MC is *ergodic* if

$\lim_{m \to \infty} p^{(0)} P^m = \pi$. 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.

7.4   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

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

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

7.5   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

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

7.6   Misc

7.6.1   Determinants and PM

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

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

$$pf(A) = \frac{1}{2^n n!} \sum_{\sigma \in S_{2n}} \text{sgn}(\sigma) \prod_{i=1}^{n} a_{\sigma(2i-1),\sigma(2i)}$$

$$= \sum_{M \in \text{PM}(n)} \text{sgn}(M) \prod_{(i,j) \in M} a_{i,j}$$

7.6.2   BEST Theorem

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

7.6.3   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 $0 \le i < k$

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: $p_1, \ldots, p_k$. Finally, calculate $a^{s/p_i} \mod p$ for all $i = 1 \ldots k$, 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 $\phi(p)$.

7.6.4   Sum of primes

For any multiplicative f:

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

7.6.5   Floor

$$\lfloor \lfloor x/y \rfloor / z \rfloor = \lfloor x/(yz) \rfloor$$
$$x \% y = x - y \lfloor x/y \rfloor$$