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;
}
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);
```

```
    form_chains(g[c][i]);
  }
  if(g[c].empty())
    trees.pb(SegTree<int>([](int a,int b){return max(a,b);},0,(int)chains.back().size(),chains.back()));
}
void update(int v,int val){
  trees[cid[v]].update(id[v],val);
}
int query(int u,int v){
  int r=0;
  while(u!=v){
    if(cid[v]==cid[u]){
      if(depth[v]<depth[u])
        swap(v,u);
      r=max(r,trees[cid[v]].query(id[u]+1,id[v]));
      v=u;
    }
    else{
      if(depth[cpar[cid[v]]]<depth[cpar[cid[u]]])
        swap(v,u);
      r=max(r,trees[cid[v]].query(0,id[v]));
      v=cpar[cid[v]];
    }
  }
  return r;
}
```

2.3 Auxiliary Tree

```

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

sort(ALL(a), [](const int & a, const int & b) -> bool{
  return st[a] < st[b];
});

stack<int> S;
S.push(a[0]);

auto anc = [](int & a, int & b) -> bool{
  return st[b] >= st[a] && en[b] <= en[a];
};

for(int i = 1; i < (int)a.size(); i++){
  while(!anc(S.top(), a[i])) S.pop();
  G[S.top()].pp(a[i]);
  G[a[i]].pp(S.top());
  S.push(a[i]);
}
//G is the Aux tree
```

2.4 Articulation Point and Bridges

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

using namespace std;
const int N = 50;
int dis[N], low[N], par[N], AP[N],vis[N],tits;
void update(int u , int i, int child) {
  //For Cut Vertices
  if(par[u] != -1 && low[i] >= dis[u]) AP[u] = true;
  if(par[u] == -1 && child > 1) AP[u] = true;

  //For Finding Cut Bridge
  if(low[i] > dis[u]){
    //articulation bridge found.
  }
}
void dfs(int u){
  vis[u] = true;
  low[u] = dis[u] = (++tits); int child = 0;
  for(int i : g[u]) {
    if(!vis[i]){
      child++;
      par[i] = u;
      dfs(i);
```

```
      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

```
#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){
  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);
    }
  }
```

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

    for(int i = 0; i < n; i ++){
      if(!vis[i]){
        temp.clear();
        cmpno++;
        dfs1(i);
      }
    }
}
```

2.6 2-SAT

```
/*
 * Make sure to give the size of n atleast a larger than original (n + 100).
 * In the truth_table(), u, v are 1 based indexed.
 * Truth value of the nodes is calculated in the satisfiable() function i.e. val[] vector
 * st[] -> stack
 * comp[] -> component number of every node
 */

class sat_2{
  public:
    int n, m, tag;
    vector<vector<int>> g, grev;
    vector<bool> val;
    vector<int> st;
    vector<int> comp;

    sat_2(){}
    sat_2(int n) : n(n), m(2 * n), tag(0), g(m + 1), grev(m + 1), val(n + 1) { }

    void add_edge(int u, int v) { //u or v
      auto make_edge = [&](int a, int b) {
        if(a < 0) a = n - a;
        if(b < 0) b = n - b;
        g[a].push_back(b);
        grev[b].push_back(a);
      };

      make_edge(-u, v);
      make_edge(-v, u);
    }

    void truth_table(int u, int v, vector<int> t) {
      for(int i = 0; i < 2; i ++) for(int j = 0; j < 2; j ++) {
        if(!t[i * 2 + j])
          add_edge((2 * (i ^ 1) - 1) * u, (2 * (j ^ 1) - 1) * v);
      }
    }

    void dfs(int u, vector<vector<int>> & G, bool first) {
      comp[u] = tag;
      for(int & i : G[u]) if(comp[i] == -1)
        dfs(i, G, first);
      if(first) st.push_back(u);
    }

    bool satisfiable() {
      tag = 0; comp.assign(m + 1, -1);
      for(int i = 1; i <= m; i ++) {
        if(comp[i] == -1)
          dfs(i, g, true);
      }reverse(ALL(st));

      tag = 0; comp.assign(m + 1, -1);
      for(int & i : st) {
        if(comp[i] != -1) continue;
        tag++;
        dfs(i, grev, false);
      }

      for(int i = 1; i <= n; i ++) {
        if(comp[i] == comp[i + n]) return false;
        val[i] = comp[i] > comp[i + n];
      }

      return true;
    }
};
```

2.7 Dinic's Max Flow

```
// from stanford notebook
```

```
struct edge {
  int u, v;
  ll c, f;
  edge() { }
  edge(int _u, int _v, ll _c, ll _f = 0): u(_u), v(_v), c(_c), f(_f) { }
};
int n;
vector<edge> edges;
vector<vector<int> > g;
vector<int> d, pt;

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

```
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});
    }
  }
  for(int i=0;i<n;i++)  if(npot[i]!=inf){
    pot[i]+=npot[i];
  }
}
void negativeEdges(int s){
  pot.assign(n,inf);
  pot[s]=0;
  for(int j=0;j<n;j++)
    for(int i=0;i<(int)e.size();i++)  if(e[i].f
      >0 && pot[e[i^1].v]!=inf){
      pot[e[i].v]=min(pot[e[i].v],pot[e[i^1].v]+
        e[i].c);
  }
}
int augment(int s,int t,int fl,V<bool> &v){
  if(s==t)
    return fl;
  v[s]=1;
  for(auto i:g[s])  if(!v[e[i].v] && e[i].f>0 &&
      (pot[s]-pot[e[i].v]+e[i].c)==0){
    int cf=augment(e[i].v,t,min(fl,e[i].f),v);
    if(cf!=0){
      e[i].f-=cf;
      e[i^1].f+=cf;
      return cf;
    }
  }
  return 0;
}
void mcf(int s,int t,bool neg=0){
  int cur=0;
  V<bool> vis;
    if(neg)
      negativeEdges(s);
    do{
      vis.assign(n,0);
      flow+=cur;
      cost+=(pot[t]-pot[s]);
      assignPots(s);
      cur=augment(s,t,inf,vis);
    }while(cur);
  }
};
```

2.9 Global Min Cut

```
// Adj mat. Stoer-Wagner min cut algorithm.
// Running time:O(|V|^3)
typedef vector<int> VI;
typedef vector<VI> VVI;

const int INF = 1000000000;

pair<int, VI> GetMinCut(VVI &weights) {
  int N = weights.size();
  VI used(N), cut, best_cut;
  int best_weight = -1;

  for (int phase = N-1; phase >= 0; phase--) {
    VI w = weights[0];
    VI added = used;
    int prev, last = 0;
    for (int i = 0; i < phase; i++) {
      prev = last;
      last = -1;
      for (int j = 1; j < N; j++)
  if (!added[j] && (last == -1 || w[j] > w[last]))
      last = j;
      if (i == phase-1) {
  for (int j = 0; j < N; j++) weights[prev][j] +=
     weights[last][j];
  for (int j = 0; j < N; j++) weights[j][prev] =
     weights[prev][j];
  used[last] = true;
  cut.push_back(last);
  if (best_weight == -1 || w[last] < best_weight)
     {
```

```
      best_cut = cut;
      best_weight = w[last];
    }
        } else {
  for (int j = 0; j < N; j++)
    w[j] += weights[last][j];
  added[last] = true;
        }
      }
    }
  return make_pair(best_weight, best_cut);
}

int main() {
  int N;
  cin >> N;
  for(int i = 0; i < N; i++) {
    int n, m;
    cin >> n >> m;
    VVI weights(n, VI(n));
    for (int j = 0; j < m; j++) {
      int a, b, c;
      cin >> a >> b >> c;
      weights[a-1][b-1] = weights[b-1][a-1] = c;
    }
    pair<int, VI> res = GetMinCut(weights);
    cout << "Case #" << i+1 << ": " << res.first
        << endl;
  }
}
```

2.10 Bipartite Matching

```
// maximum cardinality bipartite matching using
   augmenting paths.
// assumes that first n elements of graph
   adjacency list belong to the left vertex set.
int n;
vector<vector<int>> graph;
vector<int> match, vis;

int augment(int l) {
  if(vis[l]) return 0;
  vis[l] = 1;
  for(auto r: graph[l]) {
    if(match[r]==-1 || augment(match[r])) {
      match[r]=l; return 1;
    }
  }
  return 0;
}

int matching() {
  int ans = 0;
  for(int l = 0; l < n; l++) {
    vis.assign(n, 0);
    ans += augment(l);
  }
  return ans;
}
```

2.11 Hopcraft-Karp

```
#define MAX 100001
#define NIL 0
#define INF (1<<28)

vector< int > G[MAX];
int n, m, match[MAX], dist[MAX];
// n: number of nodes on left side, nodes are
   numbered 1 to n
// m: number of nodes on right side, nodes are
   numbered n+1 to n+m

bool bfs() {
    int i, u, v, len;
    queue< int > Q;
    for(i=1; i<=n; i++) {
        if(match[i]==NIL) {
            dist[i] = 0;
            Q.push(i);
        }
        else dist[i] = INF;
    }
    dist[NIL] = INF;
    while(!Q.empty()) {
        u = Q.front(); Q.pop();
        if(u!=NIL) {
            len = G[u].size();
            for(i=0; i<len; i++) {
                v = G[u][i];
```

```
                if(dist[match[v]]==INF) {
                    dist[match[v]] = dist[u] + 1;
                    Q.push(match[v]);
                }
            }
        }
    }
    return (dist[NIL]!=INF);
}

bool dfs(int u) {
    int i, v, len;
    if(u!=NIL) {
        len = G[u].size();
        for(i=0; i<len; i++) {
            v = G[u][i];
            if(dist[match[v]]==dist[u]+1) {
                if(dfs(match[v])) {
                    match[v] = u;
                    match[u] = v;
                    return true;
                }
            }
        }
        dist[u] = INF;
        return false;
    }
    return true;
}

int hopcroft_karp() {
    int matching = 0, i;
    // match[] is assumed NIL for all vertex in G
    while(bfs())
        for(i=1; i<=n; i++)
            if(match[i]==NIL && dfs(i))
                matching++;
    return matching;
}
```

2.12 Hungarian

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

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {
  int n = int(cost.size());

  // construct dual feasible solution
  VD u(n);
  VD v(n);
  for (int i = 0; i < n; i++) {
    u[i] = cost[i][0];
    for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
  }
  for (int j = 0; j < n; j++) {
    v[j] = cost[0][j] - u[0];
    for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
  }

  // construct primal solution satisfying complementary slackness
  Lmate = VI(n, -1);
  Rmate = VI(n, -1);
  int mated = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (Rmate[j] != -1) continue;
      if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
    Lmate[i] = j;
    Rmate[j] = i;
    mated++;
    break;
      }
    }
  }
```

```

  VD dist(n);
  VI dad(n);
  VI seen(n);

  // repeat until primal solution is feasible
  while (mated < n) {

    // find an unmatched left node
    int s = 0;
    while (Lmate[s] != -1) s++;

    // initialize Dijkstra
    fill(dad.begin(), dad.end(), -1);
    fill(seen.begin(), seen.end(), 0);
    for (int k = 0; k < n; k++)
      dist[k] = cost[s][k] - u[s] - v[k];

    int j = 0;
    while (true) {

      // find closest
      j = -1;
      for (int k = 0; k < n; k++) {
    if (seen[k]) continue;
    if (j == -1 || dist[k] < dist[j]) j = k;
      }
      seen[j] = 1;

      // termination condition
      if (Rmate[j] == -1) break;

      // relax neighbors
      const int i = Rmate[j];
      for (int k = 0; k < n; k++) {
    if (seen[k]) continue;
    const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
    if (dist[k] > new_dist) {
      dist[k] = new_dist;
      dad[k] = j;
    }
      }
    }
    // update dual variables
    for (int k = 0; k < n; k++) {
      if (k == j || !seen[k]) continue;
      const int i = Rmate[k];
      v[k] += dist[k] - dist[j];
      u[i] -= dist[k] - dist[j];
    }
    u[s] += dist[j];

    // augment along path
    while (dad[j] >= 0) {
      const int d = dad[j];
      Rmate[j] = Rmate[d];
      Lmate[Rmate[j]] = j;
      j = d;
    }
    Rmate[j] = s;
    Lmate[s] = j;

    mated++;
  }

  double value = 0;
  for (int i = 0; i < n; i++)
    value += cost[i][Lmate[i]];

  return value;
}
```

2.13 Link

```
struct Node { // Splay tree. Root's pp contains tree's parent.
    Node *p = 0, *pp = 0, *c[2];
    bool flip = 0;
    Node() { c[0] = c[1] = 0; fix(); }
    void fix() {
        if (c[0]) c[0]->p = this;
        if (c[1]) c[1]->p = this;
        // (+ update sum of subtree elements etc. if wanted)
    }
    void push_flip() {
        if (!flip) return;
        flip = 0; swap(c[0], c[1]);
        if (c[0]) c[0]->flip ^= 1;
        if (c[1]) c[1]->flip ^= 1;
    }
```

```
    int up() { return p ? p->c[1] == this : -1; }
    void rot(int i, int b) {
        int h = i ^ b;
        Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
        if ((y->p = p)) p->c[up()] = y;
        c[i] = z->c[i ^ 1];
        if (b < 2) {
            x->c[h] = y->c[h ^ 1];
            z->c[h ^ 1] = b ? x : this;
        }
        y->c[i ^ 1] = b ? this : x;
        fix(); x->fix(); y->fix();
        if (p) p->fix();
        swap(pp, y->pp);
    }
    void splay() { /// Splay this up to the root. Always finishes without flip set.
        for (push_flip(); p; ) {
            if (p->p) p->p->push_flip();
            p->push_flip(); push_flip();
            int c1 = up(), c2 = p->up();
            if (c2 == -1) p->rot(c1, 2);
            else p->p->rot(c2, c1 != c2);
        }
    }
    Node* first() { /// Return the min element of the subtree rooted at this, splayed to the top.
        push_flip();
        return c[0] ? c[0]->first() : (splay(), this);
    }
};

struct LinkCut {
    vector<Node> node;
    LinkCut(int N) : node(N) {}

    void link(int u, int v) { // add an edge (u, v)
        assert(!connected(u, v));
        make_root(&node[u]);
        node[u].pp = &node[v];
    }

    void cut(int u, int v) { // remove an edge (u, v)
        Node *x = &node[u], *top = &node[v];
        make_root(top); x->splay();
        assert(top == (x->pp ?: x->c[0]));
        if (x->pp) x->pp = 0;
        else {
            x->c[0] = top->p = 0;
            x->fix();
        }
    }

    bool connected(int u, int v) { // are u, v in the same tree?
        Node* nu = access(&node[u])->first();
        return nu == access(&node[v])->first();
    }

    /// Move u to root of represented tree.
    void make_root(Node* u) {
        access(u);
        u->splay();
        if(u->c[0]) {
            u->c[0]->p = 0;
            u->c[0]->flip ^= 1;
            u->c[0]->pp = u;
            u->c[0] = 0;
            u->fix();
        }
    }

    /// Move u to root aux tree. Return the root of the root aux tree.
    Node* access(Node* u) {
        u->splay();
        while (Node* pp = u->pp) {
            pp->splay(); u->pp = 0;
            if (pp->c[1]) {
                pp->c[1]->p = 0; pp->c[1]->pp = pp; }
            pp->c[1] = u; pp->fix(); u = pp;
        }
        return u;
    }
};
```

3 Data Structures

3.1 Implicit Treap

```
//1-based with lazy-updates, range sum query
struct node {
    int val, sum, lazy, prior, size;
    node *l, *r;
};
const int N = 2e5;
node pool[N]; int poolptr=0;
typedef node* pnode;
int sz(pnode t) { return t?t->size:0; }
void upd_sz(pnode t) { if(t) t->size = sz(t->l) +
    1 + sz(t->r); }
void lazy(pnode t) {
    if(!t || !t->lazy) return;
    t->val+=t->lazy;
    t->sum+=t->lazy*sz(t);
    if(t->l)t->l->lazy+=t->lazy;
    if(t->r)t->r->lazy+=t->lazy;
    t->lazy = 0;
}
void reset(pnode t) {
    if(t) t->sum=t->val;
}
void combine(pnode& t, pnode l, pnode r) {
    if(!l || !r) return void(t=l?l:r);
    t->sum = l->sum + r->sum;
}
void operation(pnode t) {
    if(!t) return;
    reset(t);
    lazy(t->l); lazy(t->r);
    combine(t,t->l,t); combine(t,t,t->r);
}
void split(pnode t, pnode& l, pnode& r, int pos,
    int add = 0) {
    if(!t) return void(l=r=NULL);
    lazy(t); int curr_pos = add + sz(t->l);
    if(curr_pos<pos) split(t->r,t->r,r,pos,
        curr_pos+1),l=t;
    else split(t->l,l,t->r,pos,add),r=t;
    upd_sz(t); operation(t);
}
void merge(pnode& t, pnode l, pnode r) {
    lazy(l); lazy(r);
    if(!l || !r) t = l?l:r;
    else if(l->prior > r->prior) merge(l->r,l->r,r
        ),t=l;
    else merge(r->l, l, r-> l), t=r;
    upd_sz(t); operation(t);
}
pnode init(int val) {
    pnode ret = &(pool[poolptr++]);
    ret->prior = rand(); ret->size = 1;
    ret->val = val; ret->sum = val; ret->lazy = 0;
    return ret;
}
int query(pnode t, int l, int r) {
    pnode L,mid,R;
    split(t, L, mid, l-1); split(mid, t, R, r-l);
    int ans = t->sum;
    merge(mid, L, t); merge(t, mid, R);
    return ans;
}
void upd(pnode t, int l, int r, int val) {
    pnode L, mid, R;
    split(t, L, mid, l-1); split(mid, t, R, r-l);
    t->lazy += val;
    merge(mid, L, t); merge(t, mid, R);
}
void insert(pnode& t, ll val, int pos) {
    pnode l;
    split(t,l,t,pos-1); merge(l,l,init(val));
        merge(t,l,t);
}
```

3.2 Segment Tree

```
// This code solves problem Help Ashu on
    hackerearth
// Iterative segment tree supporting non
    commutative combiner function
// The combiner function and identity of the
    combiner function are taken as contructor
    arguments
// Assign the initial input into t[size] to t[2*
    size-1] then call build
```

```
// Memory 2*size*sizeof(T)
// Time complexity O(log(size))
#include <bits/stdc++.h>
using namespace std;
/* Equinox */
template<typename T>
class SegTree{
public:
  vector<T> t;
  T identity;
  T (*combine)(T,T);
  int size;
  SegTree(T (*op)(T,T),T e,int n){
    combine=op;
    identity=e;
    t.assign(2*n,e);
    size=n;
  }
  void build(){for(int i=size-1;i>0;i--)t[i]=
     combine(t[i<<1],t[i<<1|1]);}
  T query(int l,int r){
    T lt=identity;
    T rt=identity;
    for(l+=size,r+=size;l<=r;r>>=1,l>>=1){
      if(l&1)   lt=combine(lt,t[l++]);
      if(!(r&1))  rt=combine(t[r--],rt);
    }
    return combine(lt,rt);
  }
  void update(int p,T v){for(t[p+=size]=v;p>>=1;)t
     [p]=combine(t[p<<1],t[p<<1|1]);}
};
int32_t main(){
  int n;
  cin>>n;
  SegTree<int> tree([](int a,int b){return a+b
     ;},0,n);
  for(int i=0;i<n;i++){
    int a;
    cin>>a;
    tree.t[i+n]=a&1;
  }
  tree.build();
  int q;
  cin>>q;
  while(q--){
    int c,x,y;
    cin>>c>>x>>y;
    switch(c){
      case 0:
      tree.update(x-1,y&1);
      break;
      case 1:
      cout<<(y-x+1)-tree.query(x-1,y-1)<<"\n";
      break;
      case 2:
      cout<<tree.query(x-1,y-1)<<"\n";
    }
  }
  return 0;
}
```

3.3 Lazy Propagation

```
// This code solves problem LITE on spoj
// Iterative segment tree with lazy propagation
   supporting non commutative combiner functions
// The combiner function and identity of the
   combiner function are taken as contructor
   arguments
// Also the function for application of lazy nodes
    onto tree nodes is taken as parameter along
   with Zero of lazy node
// Assign the initial input into t[size] to t[2*
   size-1] then call build
// Memory 2*size*sizeof(T)+2*size*sizeof(L)
// Time complexity O(log(size))
#include <bits/stdc++.h>
using namespace std;
/* Equinox */
template<typename T,typename L>
class SegTree{
public:
  vector<T> t;
  vector<T> lz;
  T identity;
  L zero;
  T (*combine)(T,T);
  void (*apply)(T&,L&,L&,int k);
  int size;
  int height;
```

```cpp
  SegTree(T (*op)(T,T),T e,void (*pro)(T&,L&,L&,int k),L z,int n){
    combine=op;
    apply=pro;
    identity=e;
    zero=z;
    t.assign(2*n,e);
    lz.assign(2*n,z);
    size=n;
    height = sizeof(int)*8-__builtin_clz(n);
  }
  void build(){for(int i=size-1;i>0;i--)t[i]=combine(t[i<<1],t[i<<1|1]);}
  void push(int p){
    for(int s=height;s>0;s--){
      int i=p>>s;
      apply(t[i<<1],lz[i<<1],lz[i],1<<(s-1));
      apply(t[i<<1|1],lz[i<<1|1],lz[i],1<<(s-1));
      lz[i]=zero;
    }
  }
  void reassign(int p){
    for(p>>=1;p>0;p>>=1)
      if(lz[p]==zero)
        t[p]=combine(t[p<<1],t[p<<1|1]);
  }
  T query(int l,int r){
    push(l+=size);
    push(r+=size);
    T lt=identity;
    T rt=identity;
    for(;l<=r;r>>=1,l>>=1){
      if(l&1)   lt=combine(lt,t[l++]);
      if(!(r&1))  rt=combine(t[r--],rt);
    }
    return combine(lt,rt);
  }
  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]);}
  void update(int l,int r,L v){
    push(l+=size);
    push(r+=size);
    int k=1;
    int l0=l,r0=r;
    for(;l<=r;r>>=1,l>>=1,k<<=1){
      if(l&1)   apply(t[l],lz[l],v,k),l++;
      if(!(r&1))  apply(t[r],lz[r],v,k),r--;
    }
    reassign(l0);
    reassign(r0);
  }
};
int32_t main(){
  int n,m;
  cin>>n>>m;
  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);
  while(m--){
    int c;
    cin>>c;
    if(!c){
      int l,r;
      cin>>l>>r;
      s.update(l-1,r-1,1);
    }
    else{
      int l,r;
      cin>>l>>r;
      cout<<s.query(l-1,r-1)<<"\n";
    }
  }
  return 0;
}
```

4 Math

4.1 Extended Euclid

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

using namespace std;
using LL = long long;

template<typename T> T gcd(T a , T b){return (a ? gcd(b % a , a): b);} //supposing a is small and b is large.
template<typename T> pair<T,T> extend_euclid(T a, T b){ //supposing a is small and b is large.
  pair<T,T> a_one = {1, 0} , b_one = {0 , 1};
```

```
  // b_one is just the second last step's coefficient, a_one is the last step's coefficient
  if(!b)return a_one;
  while(a){
    /* We first start from writing
    b = 0(a) + 1(b), for which it's b_one
    a = 1(a) + 0(b), for which it's a_one
    b = b % a + (b / a)*a, then
    */
    T q = b / a; T r = b % a;
    T dx = b_one.first - q*a_one.first;
    T dy = b_one.second - q*a_one.second;
    b = a; a = r;
    b_one = a_one;
    a_one = {dx , dy};
  }
  return b_one;
}

int main(){
  LL a, m; cin >> a >> m;
  auto ans = extend_euclid(a, m);
  LL x = (ans.first + m)%m; //Inverse Modulo (m) $ ax=1 mod(m) and gcd(a,m) == 1
  cout << (ans.first + m) % m << endl;
  return 0;
}
```

4.2 Fast Fourier Transform

```
const long double PI=acos(-1.0);
typedef long long ll;
typedef long double ld;
typedef vector<ll> VL;
int bits(int x){
  int r=0;
  while(x){
    r++;
    x>>=1;
  }
  return r;
}
int reverseBits(int x,int b){
  int r=0;
  for(int i=0;i<b;i++){
    r<<=1;
    r|=(x&1);
    x>>=1;
  }
  return r;
}
class Complex{
  public:
  ld r,i;
  Complex(){r=0.0;i=0.0;}
  Complex(ld a,ld b){r=a;i=b;}
};
Complex operator*(Complex a,Complex b){
  return Complex(a.r*b.r-a.i*b.i,a.r*b.i+a.i*b.r);
}
Complex operator-(Complex a,Complex b){
  return Complex(a.r-b.r,a.i-b.i);
}
Complex operator+(Complex a,Complex b){
  return Complex(a.r+b.r,a.i+b.i);
}
Complex operator/(Complex a,ld b){
  return Complex(a.r/b,a.i/b);
}
Complex EXP(ld theta){
  return Complex(cos(theta),sin(theta));
}

typedef vector<Complex> VC;

void FFT(VC& A,int inv){
  int l=A.size();
  int b=bits(l)-1;
  VC a(A);
  for(int i=0;i<l;i++){
    A[reverseBits(i,b)]=a[i];
  }
  for(int i=1;i<=b;i++){
    int m=(1<<i);
    int n=m>>1;
    Complex wn=EXP((ld)inv*(ld)2.0*PI/(ld)m);
    for(int j=0;j<l;j+=m){
      Complex w(1.0,0.0);
      for(int k=j;k<j+n;k++){
        Complex t1=A[k]+w*A[k+n];
        Complex t2=A[k]-w*A[k+n];
```

```
        A[k]=t1;
        A[k+n]=t2;
        w=w*wn;
      }
    }
  }
  if(inv==-1){
    for(auto &i:A){
      i=i/(ld)l;
    }
  }
}

VL Convolution(VL & a,VL & b){
  int tot_size = (int)a.size() + (int)b.size();
  int bit = bits(tot_size);
  int l = 1 << bit;
  VC A, B, C;
  A.reserve(l); B.reserve(l); C.reserve(l);
  for(int i = 0; i < l; i ++) {
    if(i < (int)a.size()) A.pb({(ld)a[i], 0.0});
    else A.pb({0.0, 0.0});
    if(i < (int)b.size()) B.pb({(ld)b[i], 0.0});
    else B.pb({0.0, 0.0});
  }
  FFT(A, 1);
  FFT(B, 1);
  for(int i = 0; i < l; i ++) {
    C.pb(A[i] * B[i]);
  }
  FFT(C, -1);
  VL c;
  for(auto & i : C) {
    c.pb(round(i.r));
  }
  return c;
}
```

4.3 Large Factorial

```
ll fmod(ll x,ll md,ll p){
  V<ll> pre(md);
  pre[0]=1;
  for(ll i=1;i<md;i++){
    if(i%p!=0)
      pre[i]=(pre[i-1]*i)%md;
    else
      pre[i]=pre[i-1];
  }
  ll r=1;
  while(x){
    ll cy=x/md;
    r=(r*modex(pre[md-1],cy,md))%md;
    r=(r*pre[x%md])%md;
    x/=p;
  }
  return r;
}
```

4.4 Large Modulo Multiplication

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

4.5 Segmented Sieve

```
// Segmented Seive
// N=sqrt(b)
// Time complexity: O(N.log(B-A))
#define A 1000000000000LL
#define B 1000000100000LL
bitset<B-A> p;
void seive(){
  p.set();
  for(ll i=2;i*i<=B;i++){
    for(ll j=((A+i-1)/i)*i;j<=B;j+=i){
      p.reset(j-A);
    }
  }
}
```

4.6 Miller Rabin

```
V<int> A{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
    37, 41};

bool Miller(long long p){
  if(p < 2){
    return false;
  }
  if(p != 2 && p % 2 == 0){
    return false;
  }
  long long s = p - 1;
  while(s % 2 == 0){
    s /= 2;
  }
  for(auto & a : A){
    long long temp = s;
    long long mod = power(a, temp, p);
    while(temp != p - 1 && mod != 1 && mod != p-1)
      {
      mod = mulmod(mod, mod, p);
      temp *= 2;
    }
    if(mod != p - 1 && temp % 2 == 0){
      return false;
    }
  }
  return true;
}
```

4.7 Random Number Generator

```
random_device rd;
mt19937 gen(rd()); // mersenne twister (only for
    32 bit unsigned numbers)
uniform_int_distribution<int> dis(1, 10000); //
    dis(L, R) uniformly generates [L, R] (inclusive
    )

/* For 64 bit numbers */
mt19937_64 gen2(rd());
uniform_int_distribution<LL> dis2((int)1e9 + 7, (
    int)1e10);
cout << dis2(gen) << endl << dis2(gen) << endl;
```

5 Strings

5.1 Aho Corasick

```
const int N = 500*5005;
map<char, int> nxt[N], go[N];
int par[N], occ[N], sz = 1, link[N];
char parc[N];
void add(string& s, int i) {
    int cur = 1;
    for(char c : s) {
        if(!nxt[cur][c]) {
            sz++;
            parc[sz]=c,par[sz]=cur,nxt[cur][c]=sz,
                cur=sz;
        }
        else cur=nxt[cur][c];
    }
    occ[cur]++;
}
int GO(int p, char c);
int getlink(int p) {
    if(!link[p]) {
        if(p==1 || par[p]==1) link[p]=1;
        else {
            link[p]=GO(getlink(par[p]),parc[p]);
            occ[p] += occ[link[p]];
        }
    }
    return link[p];
}
int GO(int p, char c) {
    auto it = nxt[p].find(c);
    if(it == nxt[p].end()) {
        auto it = go[p].find(c);
        if (it==go[p].end())
            return (go[p][c]= p==1 ? 1: GO(getlink
                (p),c));
        else return it->ss;
    } else return it->ss;
}
```

5.2 Suffix Array

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

// suffixRank is table hold the rank of each string on each iteration
// suffixRank[i][j] denotes rank of jth suffix at ith iteration

int suffixRank[20][int(1E6)];

// Example "abaab"
// Suffix Array for this (2, 3, 0, 4, 1)
// Create a tuple to store rank for each suffix

struct myTuple {
    int originalIndex;   // stores original index of suffix
    int firstHalf;       // store rank for first half of suffix
    int secondHalf;      // store rank for second half of suffix
};


// function to compare two suffix in O(1)
// first it checks whether first half chars of 'a' are equal to first half chars of b
// if they compare second half
// else compare decide on rank of first half

int cmp(myTuple a, myTuple b) {
    if(a.firstHalf == b.firstHalf) return a.secondHalf < b.secondHalf;
    else return a.firstHalf < b.firstHalf;
}

int main() {

    // Take input string
    // initialize size of string as N

    string s; cin >> s;
    int N = s.size();

    // Initialize suffix ranking on the basis of only single character
    // for single character ranks will be 'a' = 0, 'b' = 1, 'c' = 2 ... 'z' = 25

    for(int i = 0; i < N; ++i)
        suffixRank[0][i] = s[i] - 'a';

    // Create a tuple array for each suffix

    myTuple L[N];

    // Iterate log(n) times i.e. till when all the suffixes are sorted
    // 'stp' keeps the track of number of iteration
    // 'cnt' store length of suffix which is going to be compared

    // On each iteration we initialize tuple for each suffix array
    // with values computed from previous iteration

    for(int cnt = 1, stp = 1; cnt < N; cnt *= 2, ++stp) {

        for(int i = 0; i < N; ++i) {
            L[i].firstHalf = suffixRank[stp - 1][i];
            L[i].secondHalf = i + cnt < N ? suffixRank[stp - 1][i + cnt] : -1;
            L[i].originalIndex = i;
        }

        // On the basis of tuples obtained sort the tuple array

        sort(L, L + N, cmp);

        // Initialize rank for rank 0 suffix after sorting to its original index
        // in suffixRank array

        suffixRank[stp][L[0].originalIndex] = 0;

        for(int i = 1, currRank = 0; i < N; ++i) {

            // compare ith ranked suffix ( after sorting ) to (i - 1)th ranked
```

```
                suffix
            // if they are equal till now assign same rank to ith as that of (i - 1)th
            // else rank for ith will be currRank ( i.e. rank of (i - 1)th ) plus 1, i.e ( currRank + 1 )

            if(L[i - 1].firstHalf != L[i].firstHalf || L[i - 1].secondHalf != L[i].secondHalf)
                ++currRank;

            suffixRank[stp][L[i].originalIndex] = currRank;
        }

    }

    // Print suffix array

    for(int i = 0; i < N; ++i) cout << L[i].originalIndex << endl;

    return 0;
}
```

5.3 Suffix Tree

```
const int N=1000000,    // maximum possible number of nodes in suffix tree
    INF=1000000000; // infinity constant
string a;       // input string for which the suffix tree is being built
int t[N][26],   // array of transitions (state, letter)
    l[N],   // left...
    r[N],   // ...and right boundaries of the substring of a which correspond to incoming edge
    p[N],   // parent of the node
    s[N],   // suffix link
    tv,     // the node of the current suffix (if we're mid-edge, the lower node of the edge)
    tp,     // position in the string which corresponds to the position on the edge (between l[tv] and r[tv], inclusive)
    ts,     // the number of nodes
    la;     // the current character in the string

void ukkadd(int c) { // add character s to the tree
    suff:;      // we'll return here after each transition to the suffix (and will add character again)
    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
        if (t[tv][c]==-1) {t[tv][c]=ts;l[ts]=la;p[ts++]=tv;tv=s[tv];tp=r[tv]+1;goto suff;}
        tv=t[tv][c];tp=l[tv];
    } // otherwise just proceed to the next edge
    if (tp==-1 || c==a[tp]-'a')
        tp++; // if the letter on the edge equal c, go down that edge
    else {
        // otherwise split the edge in two with middle in node ts
        l[ts]=l[tv];r[ts]=tp-1;p[ts]=p[tv];t[ts][a[tp]-'a']=tv;
        // add leaf ts+1. It corresponds to transition through c.
        t[ts][c]=ts+1;l[ts+1]=la;p[ts+1]=ts;
        // update info for the current node - remember to mark ts as parent of tv
        l[tv]=tp;p[tv]=ts;t[p[ts]][a[l[ts]]-'a']=ts;ts+=2;
        // prepare for descent
        // tp will mark where are we in the current suffix
        tv=s[p[ts-2]];tp=l[ts-2];
        // while the current suffix is not over, descend
        while (tp<=r[ts-2]) {tv=t[tv][a[tp]-'a'];tp+=r[tv]-l[tv]+1;}
        // if we're in a node, add a suffix link to it, otherwise add the link to ts
        // (we'll create ts on next iteration).
        if (tp==r[ts-2]+1) s[ts-2]=tv; else s[ts
```

```
            -2]=ts;
        // add tp to the new edge and return to add letter to suffix
        tp=r[tv]-(tp-r[ts-2])+2;goto suff;
    }
}

void build() {
    ts=2;
    tv=0;
    tp=0;
    fill(r,r+N,(int)a.size()-1);
    // initialize data for the root of the tree
    s[0]=1;
    l[0]=-1;
    r[0]=-1;
    l[1]=-1;
    r[1]=-1;
    memset (t, -1, sizeof t);
    fill(t[1],t[1]+26,0);
    // add the text to the tree, letter by letter
    for (la=0; la<(int)a.size(); ++la)
        ukkadd (a[la]-'a');
}
```

6 Geometry

6.1 Geometry Library

```

/*Returns the orientation of Point C wrt line from B to A
 * It returns :-
 * -1 if C lies to left
 * +1 if C lies to the right of the line
 * 0 if C lies on the line
 */

int ccw(Point a, Point b, Point c){
  int ans = (a - c) ^ (b - c);
  return ans < 0 ? -1 : ans > 0;
}

/* 0 means outside, 1 means looselyinside the polygon(include on the edges of the polygon)*/

/* To change it strictly inside
 * change the type of this function to int
 * 0 means on the edge / point
 * +1 means strictly inside
 * -1 means strictly outside
 * winding number = 0 means outside
 * winding number != 0 means inside
 */

bool is_inside(auto & p, auto & pt){
   int n = (int)p.size();
   int cnt = 0;

   for(int i = 0; i < n; i++){
    if(p[i] == pt) return true;
    int j = (i + 1) % n;
    if(p[i].y == pt.y && p[j].y == pt.y){
      if(pt.x >= min(p[i].x, p[j].x) && pt.x <= max(p[i].x, p[j].x))
        return true;
    }else{
      bool below = p[i].y < pt.y;
      if(below != (p[j].y < pt.y)){
        auto orientation = ccw(p[i], p[j], pt);
        if(!orientation) return true;
        if(below == (orientation > 0)) cnt += below ? 1 : -1;
      }
    }
   }

   return (cnt != 0);
}
```

6.2 Convex Hull

```
vector<Point> half_hull(vector<Point> &pts,int t){
    vector<Point> hull;
    hull.pb(pts[0]);
    hull.pb(pts[1]);
    for(int i=2;i<n;i++){
        while((int)hull.size()>1){
            Point p1=hull[(int)hull.size()-2];
            Point p2=hull.back();
            if((((p1-pts[i])*(p2-pts[i]))*t)>=0){
                hull.pop_back();
```

```
            }
            else
                break;
        }
        hull.pb(pts[i]);
    }
    return move(hull);
}
vector<Point> convex_hull(vector<Point> &pts){
    sort(pts.begin(), pts.end(),[](Point &a,Point &b){
        if(a.x==b.x)
            return a.y<b.y;
        return a.x<b.x;
    });
    vector<Point> uh,lh;
    uh=half_hull(pts,1);
    lh=half_hull(pts,-1);
    lh.pop_back();
    reverse(lh.begin(), lh.end());
    uh.insert(uh.end(),lh.begin(), lh.end());
    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	$\left[{0 \atop 0}\right] = 1,\ \left[{n \atop 0}\right] = \left[{0 \atop n}\right] = 0,\ \left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$	#perms of n objs with exactly k cycles
Stirling 2nd kind	$\left\{{n \atop 1}\right\} = \left\{{n \atop n}\right\} = 1,\ \left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$	#ways to partition n objs into k nonempty sets
Euler	$\left\langle{n \atop 0}\right\rangle = \left\langle{n \atop n-1}\right\rangle = 1,\ \left\langle{n \atop k}\right\rangle = (k+1)\left\langle{n-1 \atop k}\right\rangle + (n-k)\left\langle{n-1 \atop k-1}\right\rangle$	#perms of n objs with exactly k ascents
Euler 2nd Order	$\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle = (k+1)\left\langle\!\!\left\langle{n-1 \atop k}\right\rangle\!\!\right\rangle + (2n-k-1)\left\langle\!\!\left\langle{n-1 \atop k-1}\right\rangle\!\!\right\rangle$	#perms of $1, 1, 2, 2, ..., n, n$ with exactly k ascents
Bell	$B_1 = 1,\ B_n = \sum_{k=0}^{n-1} B_k \binom{n-1}{k} = \sum_{k=0}^{n} \left\{{n \atop k}\right\}$	#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	$\frac{k}{n}\binom{n}{k} n^{n-k}$
$\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$	$\sum_{i=1}^{n} i^3 = n^2(n+1)^2/4$
$!n = n \times !(n-1) + (-1)^n$	$!n = (n-1)(!(n-1) + !(n-2))$
$\sum_{i=1}^{n} \binom{n}{i} F_i = F_{2n}$	$\sum_i \binom{n-i}{i} = F_{n+1}$
$a \equiv b \pmod{x, y} \Rightarrow a \equiv b \pmod{\operatorname{lcm}(x, y)}$	$\sum_{d \mid n} \phi(d) = n$
$ac \equiv bc \pmod m \Rightarrow a \equiv b \pmod{\frac{m}{\gcd(c,m)}}$	$(\sum_{d \mid n} \sigma_0(d))^2 = \sum_{d \mid n} \sigma_0(d)^3$
p 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 Twelvefold Way

Putting n balls into k boxes.

Balls Boxes	same same	distinct same	same distinct	distinct distinct	Remarks
-	$\mathrm{p}_k(n)$	$\sum_{i=0}^{k} \left\{{n \atop i}\right\}$	$\binom{n+k-1}{k-1}$	k^n	$\mathrm{p}_k(n)$: #partitions of n into $\leq k$ positive parts
size ≥ 1	$\mathrm{p}(n, k)$	$\left\{{n \atop k}\right\}$	$\binom{n-1}{k-1}$	$k!\left\{{n \atop k}\right\}$	$\mathrm{p}(n, k)$: #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:** $\left(\frac{a}{b}\right) = a^{(b-1)/2} \pmod b$, 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 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

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

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

$$\begin{cases} A_1x + B_1y + C_1 = 0, \\ A_2x + B_2y + C_2 = 0 \end{cases}$$

  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 $d_0 = \frac{|C|}{\sqrt{A^2+B^2}}$, 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 $d = \sqrt{r^2 - \frac{C^2}{A^2+B^2}}$ 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 $d_0 < r$)

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

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

- **Farey Sequence:** F_n Sequence of reduced fractions with denominators $\le n$. 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 $a_1, \ldots, a_n$ with nonnegative 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

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

7.6.2 BEST Theorem

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

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

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