Team Notebook

UPGraded

January 9, 2025

Contents

1 Aho-Corasick 2
2 BinaryLifting 2
3 CHT 2
4 Centroid 3
5 Congruencias 4
6 ConvexHull 4
7 CribaLineal 4
8 Dinics 5
9 GCD 5
10 GaussianElimination 5
11 Grundy 6
12 Hungarian 6
13 KMP 7
14 KnapsackBalance 7
15 MinCostMaxFlow 7
16 OrderStatisticsTree 8
17 SCC 8
18 SegmentTree 9
19 SegmentTreeLazy 9
20 SparseTableLCA 9
21 SuffixTree 10
22 Z-Function 10
23 bashrc 11
24 randomForumlas 11
25 vimrc 11

1 Aho-Corasick

```
namespace aho{
    vector<string> vs;
    ll val[maxn+1];
    ll smatch[maxn+1];

    ll p[maxn+1];
    ll pchar[maxn+1];
    map<ll,ll> vadj[maxn+1];

    ll nCount=1;
    const long long root=0;

    ll flink[maxn+1];
    ll olink[maxn+1];

    void addString(string& s,ll id){
        ll i=root;
        for(char c: s){
            if(vadj[i].find(c) != vadj[i].end()){
                i = vadj[i][c];
                continue;
            }
            ll node = nCount++;
            p[node]=i;
            pchar[node]=c;
            vadj[i][c]=node;
            i=node;
        }
        smatch[i]=id;
        val[i]++;
    }


    void build(){
        for(int i=0; i<maxn; i++) smatch[i]=-1;
        pchar[root]=-1;
        ll id=0;

        for(string& s: vs) addString(s,id++);
        queue<ll> q; q.push(root);

        while(!q.empty()){
            ll x=q.front(); q.pop();
            for(pll yy: vadj[x]) q.push(yy.second);

            ll nx = flink[p[x]];
            ll c = pchar[x];

            while(nx != root && vadj[nx].find(c) == vadj[nx].
                end()) nx = flink[nx];

            if(vadj[nx].find(c) == vadj[nx].end() || vadj[nx
                ][c] == x) flink[x]=root;
            else flink[x]=vadj[nx][c];

            if(smatch[flink[x]] != -1) olink[x] = flink[x];
            else olink[x] = olink[flink[x]];


        }
    }

    void match(ll node, ll pos){
        // match, at T[pos]. If you need the matching P, use
            vs[smatch[node]]
    }

    void aho(string& T){
        ll x=root;
        for(int c=0; c<(ll)T.size(); c++){
            while(x != root && vadj[x].find(T[c]) == vadj[x].
                end()) x = flink[x];
            if(vadj[x].find(T[c]) == vadj[x].end()) continue;
            x = vadj[x][T[c]];
            if(smatch[x] != -1) match(x,c);
            ll mx = olink[x];
            while(mx != root){
                match(mx,c);
                mx = olink[mx];
            }
        }
    }
}
```

2 BinaryLifting

```
namespace LCA{
    const int maxPow = 30;
    int lifting[maxn][maxPow];
    int in[maxn];
    int out[maxn];
    int timeTransversal=0;

    void clear(){ // CALL THIS
        timeTransversal=0;
    }

    void build(int x, int p, vector<vll>& vadj){
        lifting[x][0] = p;
        in[x]=timeTransversal++;
        for(int i=0; i<maxPow-1; i++){
            lifting[x][i+1] = lifting[lifting[x][i]][i];
        }
        for(int y: vadj[x]){
            if(y == p) continue;
            build(y,x,vadj);
        }
        out[x]=timeTransversal++;
    }

    int getKthParent(int x, int k){ // Only if you want to
         know this
        if(k == 0) return x;
        int nextBit = (k&(-k));
        return getKthParent(lifting[x][__builtin_ctz(k)],k-
            nextBit);
    }

    bool isAncestor(int x, int y){ // ancestor x of y
        return (in[x] <= in[y] && out[x] >= out[y]);
    }

    int lca(int x, int y){
        if(isAncestor(x,y)) return x;
        if(isAncestor(y,x)) return y;
        //Moving x;
        for(int pow2 = maxPow-1; pow2>=0; pow2--){
            if(!isAncestor(lifting[x][pow2],y)){
                x = lifting[x][pow2];
            }
        }
        return lifting[x][0];
    }
}
```

3 CHT

```
struct CHT {
  struct Frac {
    ll x, y;
    double eps = 1e-9;

    bool operator<(const Frac& other) const {
      return ((double(x) / double(y) + eps) <
```

```
          double(other.x) / double(other.y)) &&
         ((double(x) / double(y) - eps) <
          double(other.x) / double(other.y));
  }

  bool operator==(Frac other) const { return (other.y * x)
       == (other.x * y); }

  Frac(ll a, ll b) : x(a), y(b) {}
  Frac(ll a) : x(a), y(1) {}
};

map<ll, ll> mp;
map<Frac, ll> qrange;
Frac INF = {(1ll << 56), 1};

Frac Intersect(map<ll, ll>::iterator a, map<ll, ll>::
     iterator b) {
  return {a->second - b->second, b->first - a->first};
}

ll prevInf = -1;

bool Check(ll m, ll x) {
  if (mp.find(m) != mp.end()) {
    if (mp[m] >= x) return false;
    auto it = mp.find(m);
    auto n = ++it;
    --it;
    if (n == mp.end()) {
      qrange.erase(INF);
      qrange[INF] = m;
      prevInf = it->first * -1;
    } else {
      qrange.erase(Intersect(it, n));
    }

    if (it == mp.begin()) return true;
    auto xx = --it;
    ++it;
    qrange.erase(Intersect(xx, it));
    return true;
  }
  auto y = mp.upper_bound(m);
  if (y == mp.begin()) return true;

  if (y == mp.end()) {
    qrange.erase(INF);
    prevInf = m * -100;
    return true;
  }
  auto u = (--y);
  ++y;

  Frac sec = Intersect(u, y);
  Frac cross = {u->second - x, m - u->first};
  if (sec < cross) return false;
  qrange.erase(sec);
  return true;
}

void UpdateQRange(map<ll, ll>::iterator it) {
  auto x = ++it;
  --it;
  if (x == mp.end())
    qrange[INF] = it->first;
  else
    qrange[Intersect(it, x)] = it->first;
}

void UpdateRight(ll m) {
  auto it = mp.find(m);
  auto x = ++it;
  --it;
  if (x == mp.end()) return;
  while (1) {
    auto y = ++x;
    --x;
    if (y == mp.end()) return;
    Frac sec = Intersect(y, x);
    Frac cross = Intersect(y, it);
    if (cross < sec) return;

    qrange.erase(sec);
    mp.erase(x);
    x = y;
  }
}

void UpdateLeft(ll m) {
  auto it = mp.find(m);
  if (it == mp.begin()) return;
  auto x = --it;
  ++it;

  while (x != mp.begin()) {
    auto y = --x;
    ++x;
    Frac sec = Intersect(y, x);
    Frac cross = Intersect(y, it);
    if (sec < cross) return;
    qrange.erase(sec);
    mp.erase(x);
    x = y;
  }
}

void Add(ll m, ll x) {
  if (!Check(m, x)) return;

  mp[m] = x;
  UpdateRight(m);
  UpdateLeft(m);
  auto it = mp.find(m);
  UpdateQRange(it);
  if (it == mp.begin()) return;
  --it;
  UpdateQRange(it);
}

ll Eval(map<ll, ll>::iterator it, ll x) { return it->first
     * x + it->second; }

ll GetMax(ll x) {
  auto it = qrange.lower_bound(x);
  return Eval(mp.find(it->second), x);
}
};
```

4 Centroid

```
namespace centroid {
   ll sz[maxn];
   ll vis[maxn];
   ll p[maxn];
   ll root;

   void calc(ll x, ll u) {
       sz[x]=1;
       for (ll y: vadj[x]) {
           if (y == u || vis[y]) continue;
           calc(y,x);
           sz[x] += sz[y];
       }
   }

   ll find(ll x, ll u, ll n) {
       for (ll y: vadj[x]) {
```

```
            if (y == u || vis[y]) continue;
            if (sz[y] > n) return find(y,x,n);
        }
        return x;
    }

    void build(int c) {
        vis[c]=1;
        for (ll y: vadj[c]) {
            if (vis[y]) continue;
            calc(y,y);
            ll c2 = find(y,y,sz[y]/2);
            p[c2]=c;
            build(c2);
        }
    }

    //call this;
    void init() {
        calc(0,0);
        root = find(0,0,sz[0]/2);
        p[root] = root;
        build(root);
    }
}
```

5 Congruencias

```
using rp = pair<ll,pll>;

rp mygcd(ll a, ll b){
    if(a==0) return {b,{0,1}};
    if(b==0) return {a,{1,0}};
    rp prev=mygcd(b%a,a);

    ll ax=prev.second.second-prev.second.first*(b/a);
    ll bx=prev.second.first;

    rp ans={prev.first, {ax,
        bx}};
    return ans;
}

rp solve(ll c1, ll c2, ll b){
    rp ans = mygcd(c1,c2);
    if(b%ans.first != 0) return {-1,{-1,-1}};
    ll bd = b/ans.first;
    ll c2p = c2/ans.first;
    ans.second.first%=c2p;
    bd%=c2p;

    ans.second.first = (__int128(bd)*__int128(ans.second.
        first))%c2p;
    ans.second.first = (ans.second.first+c2p)%c2p;

    ans.second.second = (b-ans.second.first*c1)/c2;
    return ans;
}

// Falta checar que el lcm no cause overflow
pll sistema(vll& c, vll& b){
    pll ans(b[0], c[0]);
    for(size_t i=1; i<c.size(); i++){
        ll pc = ans.second;
        ll pb = ans.first;
        rp sol = solve(pc, c[i],b[i]-pb);
        if(sol.first == -1) return {-1,-1};

        ans.second = pc*(c[i]/sol.first);
        ans.first += pc*sol.second.first;
        ans.first %= ans.second;
    }
    return ans;
}
```

6 ConvexHull

```
namespace CH {
    struct Point {
        ll x;
        ll y;
        Point() : x(0), y(0) {}
        Point(ll _x, ll _y) : x(_x), y(_y) {}
        ll operator*(const Point& b) const {
            return x * b.y - y*b.x;
        }
        Point operator-(const Point& b) const {
            return {x-b.x, y-b.y};
        }
        bool operator==(const Point& b) const{
            return ((x == b.x) && (y == b.y));
        }

        bool operator<(Point& b) {
            if (x != b.x) return x < b.x;
            return y < b.y;
        }
    };

    // skips colineal
    vector<Point> partialConvex(const vector<Point>& vl) {
        if (vl.size() == 0) return {};
        vector<Point> c(vl.size());
        c[0] = vl[0];
        int j = 1;
        for (size_t i = 1; i < vl.size(); i++) {
            if (vl[i] == c[j-1]) continue;
            if (j == 1) {
                c[j++] = vl[i];
                continue;
            }
            if (vl[i] == c[j-2]) continue;
            if ((c[j-1]-c[j-2]) * (vl[i]-c[j-2]) < 0) {
                c[j++] = vl[i];
                continue;
            }
            j--; i--;
        }
        c.resize(j);
        return c;
    }

    vector<Point> getConvexHull(vector<Point>& vl) {
        if (vl.size() == 0) return {};
        sort(vl.begin(), vl.end());
        auto ch1 = partialConvex(vl);
        reverse(vl.begin(), vl.end());
        auto ch2 = partialConvex(vl);
        vector<Point> ch;
        for (auto p : ch1) ch.push_back(p);
        for (size_t i = 1; i+1 < ch2.size(); i++) ch.
            push_back(ch2[i]);
        return ch;
    }
}
```

7 CribaLineal

```
const int maxn = 1e7;
std::vector <int> prime;
bool is_composite[maxn];

void sieve (int n) {
```

```
    std::fill (is_composite, is_composite + n, false);
    for (int i = 2; i < n; ++i) {
        if (!is_composite[i]) prime.push_back (i);
        for (int j = 0; j < prime.size () && i * prime[j] < n
             ; ++j) {
            is_composite[i * prime[j]] = true;
            if (i % prime[j] == 0) break;
        }
    }
}
```

8 Dinics

```
const int maxn = 400;

namespace Dinics{
    map<ll,ll> vadj[maxn];
    vpll layered[maxn];
    vpll layeredSum[maxn];
    int level[maxn];
    int index[maxn];
    const long long maxFlow = 1e16;

    void clear(int N){
        for(int i=0; i<N; i++) vadj[i].clear();
    }

    void buildLayered(int N, int S){
        for(int i=0; i<N; i++) level[i] = -1;
        queue<int> q;
        queue<int> q2; //0-1 BFS
        int step = 1;
        level[S] = 0;
        q.push(S);
        while(1){
            while(!q.empty()){
                int x = q.front();
                q.pop();
                for(pll yy: vadj[x]){
                    int y = yy.first;
                    if(level[y] != -1) continue;
                    if(yy.second <= 0) continue;
                    level[y] = step;
                    q2.push(y);
                }
            }
            if(q2.empty()) break;
            step++;
            while(!q2.empty()){
                q.push(q2.front());
                q2.pop();
            }
        }

        for(int i=0; i<N; i++){
            layered[i].clear();
            layeredSum[i].clear();
            for(pll yy:vadj[i]){
                if(level[i]+1 != level[yy.first]) continue;
                layered[i].push_back(yy);
                layeredSum[i].push_back(yy);
            }
        }
    }

    ll blockFlow(int x, ll flow, int T){
        if(flow == 0) return flow;
        if(x == T) return flow;
        if(index[x] >= (ll)layered[x].size()) return 0;
        for(size_t i=index[x]; i<layered[x].size(); index[x
             ]++,i=index[x]){
            ll nextFlow = min(flow, layered[x][i].second);
            ll attempt = blockFlow(layered[x][i].first,
                 nextFlow, T);
            if(attempt!=0){
                layered[x][i].second-=attempt;
                if(layered[x][i].second == 0) index[x]++;
                return attempt;
            }
        }
        return 0;
    }

    ll blockPaths(ll N,ll S,ll T){
        for(int i=0; i<N; i++) index[i]=0;
        ll ans = 0;
        while(1){
            ll flow = blockFlow(S,maxFlow, T);
            ans+=flow;
            if(flow == 0) return ans;
        }
    }

    ll dinics(ll N, ll S, ll T){
        ll ans = 0;
        while(1){
            buildLayered(N,S);
            ll push = blockPaths(N,S,T);
            ans+=push;
            if(push == 0) return ans;
            // actualizar cambios en residual
            for(int i=0; i<N; i++){
                for(size_t j=0; j<layered[i].size(); j++){
                    vadj[i][layered[i][j].first] = layered[i][
                         j].second;
                    vadj[layered[i][j].first][i] += layeredSum
                         [i][j].second-layered[i][j].second;
                }
            }
        }
    }
}
```

9 GCD

```
using rp = pair<ll,pll>;

rp mygcd(ll a, ll b){
    if(a==0) return {b,{0,1}};
    if(b==0) return {a,{1,0}};
    rp prev=mygcd(b%a,a);

    ll bd = b/prev.first;
    ll ax=prev.second.second-prev.second.first*(b/a);
    ax = (ax%bd+ bd)%bd;
    ll bx=(prev.first-a*ax)/b;

    rp ans={prev.first, {ax,
        bx}};
    return ans;
}
// sea c = max(a,b), la solucion |x| <= c

// Soluciones de ax + by = 0.
// d = gcd(a,b); -> x=(b/d)*t, y=-1*(a/d)*t;
```

10 GaussianElimination

```
namespace Gauss {
    const int maxn = 1<<6;
    double mat[maxn][maxn+1];
    double ans[maxn];
```

```
    void Gauss(int N) {
        for (int i = 0; i < N-1; i++) {
            int l = i;
            for (int j = i+1; j < N; j++) {
                if (fabs(mat[j][i]) > fabs(mat[l][i])) l = j;
            }

            for (int k = i; k <= N; k++) swap(mat[i][k], mat[
                 l][k]);
            for (int j = i+1; j < N; j++) {
                for (int k = N; k >= i; k--) {
                    mat[j][k] -= mat[i][k] * mat[j][i] / mat[i
                         ][i];
                }
            }
        }

        for (int i = N-1; i >= 0; i--) {
            double t = 0;
            for (int k = i+1; k < N; k++) {
                t += mat[i][k] * ans[k];
            }
            ans[i] = (mat[i][N] - t) / mat[i][i];
        }
    }
}
```

11 Grundy

```
definiciones:
mex de un set. es el menor numero no incluido en el set.
El numero grundy de un juego/estado G(x) = mex(G(a1), G(a2),
      G(a3), ...).
G(a1), G(a2), ... etc. son las transiciones posibles de x.
G(a1, a2, a3, ...) = XOR G(a1), G(a2), G(a3), ...
```

12 Hungarian

```
namespace hungarian {

bool vis[2][maxn];
ll edge[maxn][maxn];
ll vl[2][maxn];
ll up[2][maxn];
ll match[2][maxn];

pll minEdge[maxn];
ll N;

void dfs(ll a, ll x, ll p) {
  if (vis[a][x]) return;
  vis[a][x] = true;
  up[a][x] = p;

  if (a == 1) {
    if (match[a][x] == -1) return;
    dfs(0, match[a][x], x);
    return;
  }

  for (int i = 0; i < N; i++) {
    ll c = edge[x][i];
    if (vl[1][i] + vl[0][x] != c) continue;
    dfs(1, i, x);
  }
  return;
}

void setup() {
  for (int i = 0; i < N; i++) match[0][i] = -1;
  for (int i = 0; i < N; i++) match[1][i] = -1;
  for (int i = 0; i < N; i++) vl[0][i] = 0;
  for (int i = 0; i < N; i++) vl[1][i] = 0;
}

void updatemin(ll x) {
  minEdge[x] = {1e9, -1};
  for (int j = 0; j < N; j++) {
    if (vis[1][j]) continue;
    minEdge[x] = min(minEdge[x], {edge[x][j] - vl[1][j], j});
  }
}

pll findmin() {
  pll ans = {1e9, -1};
  ll x = -1;
  for (int i = 0; i < N; i++) {
    if (!vis[0][i]) continue;
    if (ans.first <= minEdge[i].first - vl[0][i]) continue;
    ans = minEdge[i];
    ans.first -= vl[0][i];
    x = i;
  }
  return {x, ans.second};
}

void rundfs(ll x) {
  for (int i = 0; i < N; i++) vis[0][i] = 0;
  for (int i = 0; i < N; i++) vis[1][i] = 0;

  for (int i = 0; i < N; i++) {
    if (match[0][i] != -1) continue;
    dfs(0, i, -1);
    break;
  }

  for (int i = 0; i < N; i++) {
    if (!vis[0][i])
      minEdge[i] = {1e9, -1};
    else
      updatemin(i);
  }
}

void invert(ll x, ll u) {
  ll p = up[1][u];
  while (p != x) {
    match[1][u] = p;
    match[0][p] = u;
    u = up[0][p];
    p = up[1][u];
  }
  match[1][u] = p;
  match[0][p] = u;
  return;
}

void hall(ll x) {
  rundfs(x);
  for (int i = 0; i < N; i++) {
    if (!vis[1][i]) continue;
    if (match[1][i] != -1) continue;
    invert(x, i);
    return;
  }
  while (1) {
    pll e = findmin();
    if (vis[1][e.second]) {
      updatemin(e.first);
      continue;
    }
    ll c = edge[e.first][e.second] - vl[0][e.first] - vl[1][e
         .second];
    for (int i = 0; i < N; i++) {
      if (vis[1][i]) vl[1][i] -= c;
```

```
      if (vis[0][i]) vl[0][i] += c;
    }

    assert(c >= 0);

    up[1][e.second] = e.first;
    vis[1][e.second] = true;
    if (match[1][e.second] != -1) {
      up[0][match[1][e.second]] = e.second;
      vis[0][match[1][e.second]] = true;
      updatemin(e.first);
      updatemin(match[1][e.second]);
      continue;
    }
    invert(x, e.second);
    return;
  }
}

// use this
ll mincost(ll n) {
  N = n;
  setup();
  while (1) {
    bool perfect = true;
    for (int i = 0; i < N; i++) {
      if (match[0][i] == -1) {
        perfect = false;
        hall(i);
        break;
      }
    }
    if (perfect) break;
  }
  ll ans = 0;
  for (int i = 0; i < N; i++) ans += vl[0][i] + vl[1][i];
  return ans;
}
} // namespace hungarian
```

13 KMP

```
vector<int> kmp(string& s){
    int n = s.size();
    vector<int> vs(n);
    //vs[i] = kmp que acaba en la posicion i
    for(int i=1; i<n; i++){
        int j = vs[i-1]; // j = aproximacion anterior
        while(j!=0 && s[i] != s[j]){
            j = vs[j-1];
        }
        if(s[i] == s[j]) j++;
        vs[i] = j;
    }
    return vs;
}
```

14 KnapsackBalance

```
ll knapsack(vll& ww, ll C) {
    vll w;
    for (ll x : ww) {
        if (x > C) continue;
        if (x == 0) continue;
        w.push_back(x);
    }
    if (w.empty()) return 0;

    ll maxw = w[0];
    for (size_t i = 0; i < w.size(); i++) maxw = max(maxw, w[
        i]);
    vll dp(maxw*2 + 1, -1);
    vll prev(maxw*2 + 1, -1);
    auto xy = [=](ll x) {
        return (x-C+maxw);
    };

    size_t b = 0; ll sum = 0;
    for (size_t i = 0; i < w.size(); i++) {
        if (sum + w[i] > C) break;
        sum+=w[i]; b = i;
    }

    prev[xy(sum)] = b+1;
    for (size_t i = b+1; i < w.size(); i++) {
        for (ll j = C-maxw; j <= C + maxw; j++) dp[xy(j)] =
            prev[xy(j)];
        for (ll j = C-maxw; j <= C; j++) dp[xy(j+w[i])] = max
            (prev[xy(j)], dp[xy(j+w[i])]);
        for (ll j = C + maxw; j > C; j--) {
            for (ll k = dp[xy(j)]-1; k > prev[xy(j)]; k--) dp
                [xy(j-w[k])] = max(dp[xy(j-w[k])], k);
        }
        swap(prev, dp);
    }
    for (int i = C; i >= 0; i--) if (prev[xy(i)] != -1)
        return i;
    return -1;
}
```

15 MinCostMaxFlow

```
const int maxn = 50 * 101 + 2;
namespace MCMF {
    typedef struct edge {
        ll f;
        ll cost;
        ll cap;
        ll v;
        struct edge* back;
        edge(ll _cost, ll _cap, ll _v) :
            f(0), cost(_cost), cap(_cap), v(_v) {}
    } edge;
    const long long INF = 1e16;

    vector<edge*> vadj[maxn];

    void addEdge(ll u, ll v, ll cost, ll cap) {
        edge* a = new edge(cost, cap, v);
        edge* b = new edge(cost*-1, 0, u);
        a->back = b;
        b->back = a;
        vadj[u].push_back(a);
        vadj[v].push_back(b);
    }

    pll sendFlow(ll S, ll T, ll N) {
        vll vl(N, INF);
        vector<edge*> prev(N, NULL);
        queue<ll> q; q.push(S);
        vl[S] = 0;

        vll inQueue(N);
        inQueue[S] = 1;
        while (!q.empty()) {
            int k = q.front(); q.pop();
            inQueue[k] = 0;
            for (edge* e : vadj[k]) {
                if (e->f == e->cap) continue;
                if (e->cost + vl[k] >= vl[e->v]) continue;
                vl[e->v] = e->cost + vl[k];
                prev[e->v] = e;
                if (inQueue[e->v]) continue;
```

```
                q.push(e->v);
                inQueue[e->v]=1;
            }
        }

        if (vl[T] == INF) return {0,0};
        ll f = INF;
        ll x = T;
        while (x != S) {
            f = min(prev[x]->cap - prev[x]->f, f);
            x = prev[x]->back->v;
        }
        x = T;
        while (x != S) {
            prev[x]->f += f;
            prev[x]->back->f -= f;
            x = prev[x]->back->v;
        }
        return {f, vl[T]};
    }

    pll maxFlow(ll S, ll T, ll N) {
        pll ans {0,0};
        while (1) {
            pll x = sendFlow(S, T, N);
            if (x.first == 0) return ans;
            ans.first += x.first;
            ans.second += x.second * x.first;
        }
    }
}
```

16 OrderStatisticsTree

```
//Order Statistics Tree
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

using map_t = tree<ll,null_type,
      less<ll>, rb_tree_tag,
      tree_order_statistics_node_update>;

// No inserta duplicados.
// tiene mismas funciones que set o map (dependiendo de que
     uses);

int main(){
    map_t mp;
    mp.insert(3);
    mp.insert(5);
    mp.order_of_key(3); // devuelve 0;
    mp.order_of_key(4); // devolveria 1;
    mp.order_of_key(0); // devolveria 0;
    mp.find_by_order(2); // devolveria el iterador al 5 (es
         un puntero);
    mp.find_by_order(1); // devuelve el iterador al 5. (osea
         es 0 ordering);
}
```

17 SCC

```
const int maxn = // add maxn

namespace Tarjan{
    int scc=0; // Number of resulting scc;
    bool inStack[maxn];
    int lo[maxn];
    int hi[maxn];
    int vis[maxn];
    stack<int> st;
    int step=0;
    int sz[maxn]; //Size of each component. Components are 0
         indexed
    vector<vll> comps; // Each component
    int myComp[maxn]; // Maps each node to each component
    unordered_set<ll> compVadj[maxn]; // New graph.
         Components are 0 indexed

    void clear(int N){
        scc=0;
        step = 0;
        for(int i=0; i<N; i++){
            vis[i] =0;
            inStack[i] =0;
            sz[i] = 0;
            myComp[i] =0;
            compVadj[i].clear();
        }
        comps.clear();
    }

    int tarjan(int x, vector<vll>& vadj){
        if(vis[x]) return maxn+2;
        vis[x] = 1;
        inStack[x] = 1;
        st.push(x);
        lo[x] = hi[x] = step++;
        for(int y : vadj[x]){
            if(inStack[y]){
                lo[x] = min(lo[x], hi[y]);
                continue;
            }
            lo[x] = min(lo[x], tarjan(y, vadj));
        }

        if(lo[x] == hi[x]){
            int currSz = 0;
            vll thisComp;
            while(st.top() != x){
                thisComp.push_back(st.top());
                inStack[st.top()] = 0;
                myComp[st.top()] = scc;
                int y = st.top();
                for(int z : vadj[y]){
                    if(inStack[z]) continue;
                    z = myComp[z];
                    if(z == scc) continue;
                    compVadj[scc].insert(z);
                }
                currSz++;
                st.pop();
            }
            thisComp.push_back(x);
            inStack[st.top()] = 0;
            myComp[st.top()] = scc;
            int y = st.top();
            for(int z : vadj[y]){
                if(inStack[z]) continue;
                z = myComp[z];
                if(z == scc) continue;
                compVadj[scc].insert(z);
            }
            st.pop();
            currSz++;
            comps.push_back(thisComp);
            sz[scc] = currSz;
            scc++;
        }
        return lo[x];
    }

    void tarjanMain(int N, vector<vll>& vadj){ // Use this.
        clear(N); // Modify this if you use a range [1,N]
```

```
        for(int i=0; i<N; i++){ //Modify this if your nodes
             start from 1
            if(vis[i] == 0) tarjan(i,vadj);
        }
    }
}
```

18 SegmentTree

```
struct segmentTree{
    struct str{
        ll first;
        ll second;
        str() : first(0), second(0) {}
        str(ll a, ll b) : first(a), second(b) {}
    };

    str func(str a, str b){
        //merge function
    }

    str st[2*maxn];
    ll n;

    void build(vll& vl, int sizn){
        n = sizn;
        for(int i=0;i<n;i++) st[i+n] = {vl[i],i};
        for(int i=n-1;i>0;i--) st[i] = func(st[i*2],st[i
             *2+1]);
    }

    // 0-indexed
    // Value is replaced
    void update(int pos, str x){
        for(pos+=n,st[pos]=x,pos/=2; pos; pos/=2){
            st[pos] = func(st[pos*2],st[pos*2+1]);
        }
    }

    //[L,R) 0-indexed
    str query(int l, int r){
        str ansl={0,0};
        str ansr={0,0};
        for(l+=n,r+=n;l<r;l/=2,r/=2){
            if(l%2) ansl=func(ansl,st[l++]);
            if(r%2) ansr=func(st[--r],ansr);
        }
        return func(ansl,ansr);
    }
};
```

19 SegmentTreeLazy

```
struct segmentTree{
    struct str{
        ll hash;
        ll sz;
        str() : hash(0), sz(0) {}
        str(ll h, ll s) : hash(h), sz(s) {}
    };

    str st[2*maxn];
    ll n;
    ll h;
    ll d[2*maxn];
    str merge(str& a, str& b){
        // combine a,b
    }

    str ifPropagated(ll idx){
        if(d[idx] == 0) return st[idx];
        // value of st[idx] if you had propagated d[idx]
        // If propagated is alwas asked after st[idx] = merge
             (st[idx*2], st[idx*2 + 1]);
    }

    void apply(ll i, ll x){
        // apply lazy x. Remember to update d[i], flag not
             propagated to children
        // apply directly to st[i], that is the value used in
              queries
    }

    void build(vll& vl, ll sz){
        n=sz;
        h=64-__builtin_clzll(n);
        for(int i=0; i<n; i++) st[i+n] = {};
        for(int i=n-1; i>0; i--) st[i]=merge(st[i*2],st[i
             *2+1]);
    }

    void bi(ll x){
        for(x/=2; x; x/=2){
            st[x] = merge(st[x*2],st[x*2+1]);
            st[x]=ifPropagated(x);
        }
    }

    void push(ll x){
        for(int s=h; s>0; s--){
            int i=x>>s;
            if(d[i] != 0){
                apply(i*2, d[i]);
                apply(i*2+1,d[i]);
            }
            d[i]=0;
        }
    }

    //[l,r) 0-indexed
    void update(ll l, ll r, ll x){
        ll l0 = l+n;
        ll r0 = r+n-1;
        push(l0);
        push(r0);

        for(l+=n,r+=n; l<r; l/=2,r/=2){
            if(l%2) apply(l++,x);
            if(r%2) apply(--r,x);
        }
        bi(l0);
        bi(r0);
    }

    //[l,r) 0-indexed
    str query(ll l, ll r){
        push(l+n);
        push(r+n-1);
        str ansl(0,0);
        str ansr(0,0);
        for(l+=n,r+=n; l<r; l/=2,r/=2){
            if(l%2) ansl = merge(ansl,st[l++]);
            if(r%2) ansr = merge(st[--r],ansr);
        }
        return merge(ansl,ansr);
    }
};
```

20 SparseTableLCA

```
namespace lca{
    const int maxPow=20;
    const int maxn=1e5;
    ll vl[maxn];
```

```
    vll vadj[maxn];
    pll depth[2*maxn];
    pll dp[2*maxn][maxPow];
    ll t,d;

    void build(int x, int p) {
        vl[x]=t;
        depth[t++]={d,x};
        for(ll y: vadj[x]) {
            if (y==p) continue;
            ++d;
            build(y,x);
            depth[t++]={d,x};
        }
        --d;
    }

    void buildPow() {
        for(int i=0; i<(2*N-1); i++) dp[i][0]=depth[i];
        for(int j=1; j<maxPow; j++) {
            for (int i=0; i<(2*N-1); i++) {
                dp[i][j] = dp[i][j-1];
                int k = i+(1<<(j-1));
                if(k >= 2*N-1) continue;
                dp[i][j]=min(dp[i][j], dp[k][j-1]);
            }
        }
    }

    ll query(ll _a, ll _b) {
        ll a = min(vl[_a], vl[_b]);
        ll b = max(vl[_a], vl[_b]);
        ll dis = b-a+1;
        ll lz = 63-__builtin_clzll(dis);
        ll k = b-(1<<lz)+1;
        pll ans = min(dp[a][lz], dp[k][lz]);
        return ans.second;
    }

    // call this;
    void preprocess(ll x) {
        t=d=0;
        build(x,x);
        buildPow();
    }
}
```

21 SuffixTree

```
namespace suffix_tree{
    // nodes from [0, sz); //root is 0
    // maxn = maxn+1 if string has special character
    const long long inf = 1e9;
    char s[maxn];
    int to[2*maxn][40];
    int len[2*maxn], fpos[2*maxn], link[2*maxn];
    int node=0, pos=0;
    int sz=1, n=0;

    int lid=0;
    int leaves[2*maxn];

    void match(ll node, string& c, ll mc){
        // match what now
    }

    int make_node(int _pos, int _len){
        fpos[sz] = _pos;
        len[sz] = _len;
        return sz++;
    }

    void go_edge(){
        while(pos>len[to[node][(int)s[n-pos]]]){
            node = to[node][(int)s[n-pos]];
            pos -= len[node];
        }
    }

    void add_letter(int c){
        s[n++] = c;
        pos++;
        int last=0;
        while(pos>0){
            go_edge();
            int edge = s[n-pos];
            int &v = to[node][edge];
            int t = s[fpos[v] + pos - 1];
            if(v == 0){
                v = make_node(n-pos, inf);
                link[last]=node;
                leaves[lid++]=v;
                last=0;
            }
            else if(t == c){
                link[last]=node;
                return;
            }
            else{
                int u = make_node(fpos[v],pos-1);
                to[u][c] = make_node(n-1,inf);
                leaves[lid++]=to[u][c];

                to[u][t] = v;
                fpos[v] += pos-1;
                len[v] -= pos-1;
                v=u;
                link[last] = u;
                last = u;
            }
            if(node == 0) pos--;
            else node=link[node];
        }
    }

    void add_string(string& x){
        ll i = n;
        node=0; pos=0;
        len[0]=inf;
        for(char c: x) add_letter(c);
        for(int j=i; j<n; j++) len[leaves[j]] = n-fpos[leaves
            [j]];
        // from [i, n) are leaves
        len[0]=0;
    }

    void search(string& c){
        ll node=0;
        ll sz=0;
        for(size_t i=0; i<c.size(); i++){
            sz++;
            if(len[node] < sz){
                node = to[node][(int)c[i]];
                sz=1;
            }
            ll t = s[fpos[node]+sz-1];
            if(node == 0 || t != c[i]) return;
        }
        match(node,c,c.size());
    }
}
```

22 Z-Function

```
vector<int> z_func(string a) {
```

```
    // z[i] = max x such that [0, 1, ..., x) = [i, i+1, ..., i+x)
    // z[i] is the length
    vector<int> z(a.size(),0);
    for(int i = 1, l = 0, r = 0; i < (int)a.size(); ++i){
        if(i <= r) z[i] = min(r-i+1, z[i-l]);
        while(z[i] + i < (int)a.size() && a[z[i]] == a[z[i]+i])z[i]++;
        if(i+z[i]-1 > r)l = i, r = z[i]+i-1;
    }
    // optional: z[0] = a.size();
    return z;
}
```

23 bashrc

```
## Compilation and testing ###
function gc {
    if g++ $1.cpp -o $1.exe -Wall -Werror --std=c++20 2> log.txt; then
        echo "Compiled Succesfully!"
    else
        less log.txt;
    fi
}

function tn {
    for (( i=1; i<=$2; i++))
    do
        if ./$1.exe < $i.in > $i.out; then echo "Ran Test $i"
        else
            echo "Exploded Test $i"
        fi
    done
}

function dn {
    for (( i=1; i<=$1; i++))
    do
        if cmp $i.out $i.ans -s; then
            echo "Passed test $i"
        else
            echo "Failed test $i"
        fi
    done
}

export -f gc
export -f tn
export -f dn
```

24 randomForumlas

```
// Primos menores a 1e9
999999191
999999193
999999197
999999223
999999229
999999323
999999337
999999353
999999391
999999433

// Calcular Area de un triangulo dados sus lados
((a+b+c)(a+b-c)(a-b+c)(-a+b+c))^(1/2) / 4

// Calcular Angulos de triangulos Rectangulos

 |\         Angulo AB=90 degrees
 | \ C      Angulo BC=acos(B/C) // in radians in c++
A|  \       Convertir Radianes in Degrees= Ans*(360)/(2*PI)
 |___\      PI en C++ = const double pi = std::acos(-1);
   B
```

25 vimrc

```
# Create .vimrc file in home
set nu
set rnu
set nowrap

set expandtab
set shiftwidth=4
set softtabstop=4
filetype indent on
syntax on

inoremap hnh <esc>
onoremap hnh <esc>
vnoremap hnh <esc>
tnoremap hnh <C-\><C-n>
nnoremap hnh :Ex<cr>
```