University of Haifa ICPC Team Notebook 2019 - 2020

Contents

1 Data Structures

1.1 Aho-Corasick

```
/*
Aho-Corasick algorithm/data structure
Complexity: O(M) when M is the sum of length of all words
To search in text (after build) is O(N) when N is the length of the
    text
insert words into trie using "push_word"
create the links for the automaton using "create_links" (use after all
     word had been pushed)
use the automaton using "next_state"
*/
struct AhoCorasick{
    vector<map<char,int>> to;
    vector<bool> endw; //in which state a word is ended (if want which
         word, maintain a vvb
    vector<int> link;
    int sz=1;
    AhoCorasick(): to(vector<map<char,int>>(1)), endw(vector<bool
        >(1,0)){}
    void push_word(string& s){
        int cur = 0;
        for(auto& c: s){
            if (!to[cur][c]) {
                to[cur][c] = sz++;
                to.pb(map<char,int>());
                endw.pb(0);
            }
            cur = to[cur][c];
        }
        endw[cur] = 1;
    }
    void create_links(){
        queue<int> q;
        q.push(0);
        link.resize(to.size(),0);
        link[0] = -1;
        int v,u,j;
        char c;
        while(q.size()){
            v = q.front();
            q.pop();
            for(auto& it:to[v]){
                c = it.x;
                u = it.y;
                j = link[v];
                while(j!=-1 && !to[j][c]) j = link[j];
                if (j!=-1) link[u] = to[j][c];
                else link[u] = 0;
                q.push(u);
                endw[u]=endw[u] || endw[link[u]]; //merge the endw (if
                     endw[i] is vector merge them)
            }
        }
    }
    int next_state(int cur, char c){
        while(cur!=-1 && !to[cur][c]) cur = link[cur];
        if (cur==-1) return 0;
        return to[cur][c];
    }
};
```

1.2 DSU

```
struct DSU{
    vi par;
    DSU(int n){
        par.resize(n, -1);
    }
    int find(int u){
        return par[u] == -1 ? u : par[u] = find(par[u]);
    }
    void uni(int u, int v){
        parent[find(u)] = find(v);
    }
};
```

1.3 Fenwick Tree

```
struct Fenwick{
    int n;
    vi fen;
    Fenwick(int n) : n(n) {fen.resize(n+1,0); }
    void update(int ind, int val){ //*add* val to ind
        for(++ind;ind<=n;ind+=ind&(-ind)) fen[ind] += val;
    }
    int query(int ind){
        int sum = 0;
        for(++ind;ind>0;ind-=ind&(-ind)) sum += fen[ind];
        return sum;
    }
};
```

1.4 Lazy Segment Tree

```
struct SEG{ //add to interval, min on interval
    vi st, lazy;
    int comb = 1,l,r;
    SEG(int n){
        for(int i=n-1;i;i>>=1) comb <<= 1;
        st.resize(comb << 1, 0);
        lazy.resize(comb << 1, 0);
    }
    inline void setRange(int _l, int _r){
        l = _l;
        r = _r;
    }
    inline void push(int cur){
        st[cur] += lazy[cur];
        if(cur < comb){
            lazy[cur << 1] += lazy[cur];
            lazy[cur << 1 | 1] += lazy[cur];
        }
        lazy[cur] = 0;
    }
    inline void update(int l, int r, int val){
        setRange(l,r);
        update(1, 0, comb-1, val);
    }
    inline void update(int cur, int rl, int rr, int val){
        push(cur);
        if(l > rr || r < rl) return;
        if(l <= rl && r >= rr){
            lazy[cur] += val;
            push(cur);
            return;
        }
        int mid = (rl + rr) >> 1;
        update(cur << 1, rl, mid, val);
        update(cur << 1 | 1, mid + 1, rr, val);
        st[cur] = min(st[cur << 1], st[cur << 1 | 1]);
    }
    inline int query(int l, int r){
        setRange(l,r);
        return query(1, 0, comb-1);
    }
    inline int query(int cur, int rl, int rr){
        push(cur);
        if(l > rr || r < rl) return inf;
        if(l <= rl && r >= rr) return st[cur];
        int mid = (rl + rr) >> 1;
        return min(query(cur << 1, rl, mid), query(cur << 1 | 1, mid +
             1, rr));
    }
};
```

1.5 Persistent Segment Tree

```
struct Node{ //if lazy needed, build new nodes top to buttom, push
    down lazy to new nodes
    int val;
    Node *l;
    Node *r;
    Node(Node* _l = nullptr, Node* _r = nullptr, int _val = 0) : l(_l)
        , r(_r), val(_val) {}
    Node* update(int ind, int rl, int rr, int v){
        if(rl == rr) return new Node(nullptr,nullptr, val + v);
        int mid = (rl+rr)/2;
        if(ind <= mid) return new Node(l->update(ind,rl,mid,v), r, val
             + v);
        else return new Node(l, r->update(ind,mid+1,rr,v), val + v);
    }
    int query(int ql, int qr, int rl ,int rr){
        if(qr < rl || ql > rr) return 0;
        if(ql <= rl && qr >= rr) return val;
        int mid = (rl + rr)/2;
        return l->query(ql,qr,rl,mid) + r->query(ql,qr,mid+1,rr);
```

```
    }
    void init(int rl, int rr){
        if(rl == rr) return;
        int mid = (rl + rr)/2;
        l = new Node(); l->init(rl,mid);
        r = new Node(); r->init(mid + 1,rr);
    }
};

struct PERSEG{
    vector<Node*> ver;
    int comb = 1;
    PERSEG(int n){
        for(int i= n-1;i;i/=2) comb *= 2;
        ver.pb(new Node());
        ver[0]->init(0,comb - 1);
    }
    void update(int ind, int val, int v){
        ver.pb(ver[v]->update(ind, 0, comb - 1, val));
    }
    int query(int l, int r, int v){
        return ver[v]->query(l,r,0,comb - 1);
    }
};
```

2 Dynamic Programming

2.1 CHT Dynamic

```
// Keeps upper hull for maximums.
// add lines with -m and -b and return -ans to
// make this code working for minimums.
// source: http://codeforces.com/blog/entry/11155?#comment-162462
int inf = 2e18;
struct Line {
    int m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != inf) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        int x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct CHT : public multiset<Line> {
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return 1.0 * (x->b - y->b)*(z->m - y->m) >= 1.0 * (y->b - z->b
            )*(y->m - x->m);
    }
    void insertLine(int m, int b) {
        auto y = insert({ m, b});
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        while (y != begin() && bad(prev(y))) erase(prev(y));
        if(y != begin()) prev(y)->succ = [=] { return &*y; };
    }
    int eval(int x) { //query
        auto l = *lower_bound((Line) { x, inf});
        return l.m * x + l.b;
    }
};
```

2.2 CHT Linear

```
struct Frac{
    int p, q;
    Frac(int p, int q = 1) : p(p), q(q){}
};
bool operator<(const Frac &f1, const Frac &f2){
    return f1.p * f2.q < f2.p * f1.q;
}

struct Line{
    int a, b;
    Line(int a, int b) : a(a), b(b){}
    int eval(int x){
        return a * x + b;
    }
};

inline Frac intersection(const Line& l1, const Line& l2){//if overflow
     use double
    return Frac(l1.b - l2.b, l2.a - l1.a);
}

struct CHT{
    int pos = 0;
    vector<Line> hull;
    void insert(const Line& l){
        if (hull.size() != 0 && ii(l.a, l.b) <= ii(hull.back().a, hull
            .back().b)) return;
        if (hull.size() != 0 && l.a == hull.back().a) hull.pop_back();
        while (hull.size() >= 2 && !(intersection(hull[hull.size() -
            2], hull.back()) < intersection(hull.back(), l))) hull.
            pop_back();
        hull.push_back(l);
    }
    int query(int x){
```

```
        pos = min(pos, int(hull.size()) - 1);
        while (pos && hull[pos - 1].eval(x) > hull[pos].eval(x)) --pos
            ;
        while (pos < hull.size() - 1 && hull[pos].eval(x) <= hull[pos
            + 1].eval(x)) ++pos;
        return hull[pos].eval(x);
    }
};
```

2.3 Divide and Conquer Optimization

```
int n, curIter;
vector<vi> dp(2,vi(n+1));

void divAndCon(int l = 1, int r = n, int optl = 1, int optr = n){
    if(l > r) return;
    int mid = (l+r)>>1;
    ii res(inf, -1);
    for(int i=optl;i<=min(optr, mid);i++)
        res = min(res, {dp[1-(curIter%2)][i-1] + FUNCTION(i, mid),i});
    dp[curIter%2][mid] = res.first;
    divAndCon(l,mid - 1, optl, res.second);
    divAndCon(mid + 1, r, res.second, optr);
}
```

2.4 Knuth Optimization

```
int knuth(vi& arr){
    int n = arr.size();
    vector<vi> dp(n + 1,vi(n+1,inf)), piv(n + 1, vi(n+1));
    vi ps(n+1);
    ps[0] = 0;
    for(int i=1;i<=n;i++){
        dp[i][i] = 0;
        piv[i][i] = i;
        ps[i] = ps[i-1] + arr[i-1];
    }
    for(int len=2;len<=n;len++){
        for(int i=1;i+len-1<=n;i++){
            int j = i+len-1;
            for(int p=piv[i][j-1];p<=piv[i+1][j];p++){
                int cur = dp[i][p] + dp[p][j] + ps[j] - ps[i-1];
                if(dp[i][j] > cur){
                    dp[i][j] = cur;
                    piv[i][j] = p;
                }
            }
        }
    }
    return dp[1][n];
}
```

3 Geometry

3.1 Basics

```
inline int cross(ii p1, ii p2){
    return p1.first * p2.second - p1.second * p2.first;
}

inline int dot(ii p1, ii p2){
    return p1.x * p2.x + p1.y * p2.y;
}

inline int lowest_point(vector<ii>& P){
        return min_element(P.begin(), P.end(), [](ii a, ii b){return
            ii(a.second, a.first) < ii(b.second, b.first);}) - P.begin
            ();
}

inline ii operator-(ii p1, ii p2){
    return {p1.x - p2.x, p1.y - p2.y};
}


inline int sign(double x){
    return x > 0 ? 1 : x == 0 ? 0 : -1;
}

inline bool onSegment(ii s, ii e, ii p) {
        return cross(p - s, p - e) == 0 && dot(s - p, e - p) <= 0;
}

inline int norm(ii x){
    return x * x;
}

inline double distance(ii a, ii b){
    return sqrt(norm(b - a));
}

int area(vector<ii> shape){ // counter-clockwise
    int sum = 0;
    while (shape.size() > 2){
        sum += cross(shape[shape.size() - 2] - shape[0], shape[shape.
            size() - 1] - shape[0]);
        shape.pop_back();
    }
    return sum;
}
```

3.2 Convex Hull

```
vector<ii> convex_hull(vector<ii> S){
```

```
    int first = lowest_point(S);
    ii origin = S[first];
    S.erase(S.begin() + first);
    //sort by angle
    sort(S.begin(), S.end(), [&origin](ii p1, ii p2){return ii(-cross(
        p1 - origin, p2 - origin), norm(p1 - origin)) < ii(0, norm(p2
        - origin));});

    vector<ii> hull = {origin};
    for (auto& p : S){
        while(hull.size() >= 2 && cross(hull[hull.size() - 1] - hull[
            hull.size() - 2], p - hull[hull.size() - 1]) <= 0) hull.
            pop_back();
        hull.push_back(p);
    }
    return hull;
}
```

3.3 Convex Hull Point Location

```
bool inTriangle(ii a, ii b, ii c, ii p){ //not colinear points
    int sign1 = sign(cross(a-p,b-a));
    int sign2 = sign(cross(b-p,c-b));
    int sign3 = sign(cross(c-p,a-c));
    if(max(sign1,max(sign2,sign3)) == 1 && min(sign1,min(sign2,sign3))
         == -1)
        return false;
    return true;
}


bool inHull(vector<ii> & hull,ii &p){//hull standart format, no
     colinear!
    if(hull.size() < 3) return onSegment(hull[0], hull[1], p);
    if(ii(p.y, p.x) < ii(hull[0].y, hull[0].x)) return false;
    int l = 1, r = hull.size() - 1, ans = 1;
    for(int mid;l <= r;){
        mid = (l+r) / 2;
        if(cross(p - hull[0], hull[mid] - hull[0]) <= 0) ans = mid, l
            = mid+1;
        else r = mid - 1;
    }
    if(ans == hull.size() - 1) --ans;
    return inTriangle(hull[0], hull[ans], hull[ans+1], p);
}
```

3.4 Garham Scan

```
int comp(ii &p1, ii &p2){
    if((p1.y >= 0) ^ (p2.y >= 0)) return p1.y >= 0;
    if(p1.y == 0 && p2.y == 0){
        if( (p1.x >= 0) ^ (p2.x >= 0) ) return p1.x > p2.x;
        return norm(p1) < norm(p2);
    }
    return ii(-cross(p1,p2), norm(p1)) < ii(0,norm(p2));
}

void garhamScan(ii p, vector<ii> &pnt){
    for(auto &po : pnt) po.x -= p.x, po.y -= p.y;
    sort(all(pnt), comp);
    for(auto &po : pnt) po.x += p.x, po.y += p.y;
}
```

3.5 Minimal Circle

```
struct SmallestEnclosingCircle {
    Circle getCircle(vector<Point> points) {
        assert(!points.empty());

        random_shuffle(points.begin(), points.end());
        Circle c(points[0], 0);
        int n = points.size();

        for (int i = 1; i < n; i++)
            if ((points[i] - c).len() > c.r + EPS)
            {
                c = Circle(points[i], 0);
                for (int j = 0; j < i; j++)
                    if ((points[j] - c).len() > c.r + EPS)
                    {
                        c = Circle((points[i] + points[j]) / 2, (
                            points[i] - points[j]).len() / 2);
                        for (int k = 0; k < j; k++)
                            if ((points[k] - c).len() > c.r + EPS)
                                c = getCircumcircle(points[i], points[
                                    j], points[k]);
                    }
            }

        return c;
    }

    // NOTE: This code work only when a, b, c are not collinear and no
         2 points are same --> DO NOT
    // copy and use in other cases.
    Circle getCircumcircle(Point a, Point b, Point c) {
        assert(a != b && b != c && a != c);
        assert(ccw(a, b, c));

        double d = 2.0 * (a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x
            * (a.y - b.y));
        assert(fabs(d) > EPS);
        double x = (a.norm() * (b.y - c.y) + b.norm() * (c.y - a.y) +
            c.norm() * (a.y - b.y)) / d;
        double y = (a.norm() * (c.x - b.x) + b.norm() * (a.x - c.x) +
            c.norm() * (b.x - a.x)) / d;
        Point p(x, y);
```

```
        return Circle(p, (p - a).len());
    }
};
```

3.6 Sweep Non Intersecting Segments

```
struct Seg{
    ii l, r;
    int id;
    Seg() {}
    Seg(ii &l, ii &r) : l(l), r(r) {}
    double getY(int x) const{
        return l.y + (x - l.x) * (r.y - l.y) / (double) (r.x - l.x);
    }
    bool operator<(const Seg &rhs) const{
        int x = max(l.x, rhs.l.x);
        return getY(x) < rhs.getY(x);
    }
};

struct Eve{
    ii p;
    int id;
    bool fin;
    Eve(ii &p, int &id, bool fin = false) : p(p), id(id), fin(fin) {}
    bool operator<( const Eve &rhs) const{
        return p < rhs.p; //define event order
    }
};

struct Sweep{
    vector<Seg> seg;
    vector<Eve> ev;
    Sweep(vector<Seg> &seg) : seg(seg){ //get array of non-
        intersecting segments
        for(auto &s : seg) ev.pb(Eve(s.l, s.id)), ev.pb(Eve(s.r, s.id,
             true));
        sort(all(ev));
        set<Seg> st;
        vector<set<Seg> :: iterator> where;
        for(auto &e : ev){
            Seg cur = seg[e.id];
            if(e.fin) st.erase(where[cur.id]);
            else where[cur.id] = st.insert(cur).first; //insert
                returns pair
            //do something with status tree
        }
    }
};
```

4 Graphs

4.1 Bipartite Matching

```
struct BPMatching{ //first l vertices are left, next r vertices are
    right.
    vi ml, mr; // ml[i] = j vertex i is matched with j (i < l, j >= l)
    int res = 0;
    vector<bool> seen;
    vector<vi> &g;
    BPMatching(vector<vi> & g, int l, int r) : g(g){
        ml.resize(l,-1);
        mr.resize(l+r,-1); // mr is -1 for first L cells
        bipartite_matching();
    }

    bool find_match(int i) {
        for (auto &v : g[i]) {
            if (seen[v]) continue;
            seen[v] = true;
            if (mr[v] < 0 || find_match(mr[v])){
                ml[i] = v;
                mr[v] = i;
                return true;
            }
        }
        return false;
    }

    void bipartite_matching() {
        for (int i = 0; i < ml.size(); i++) {
            seen.clear(); seen.resize(g.size(),false);
            if (find_match(i)) res++;
        }
    }
};
```

4.2 Bridge

```
struct BRIDGE
{
    // function bFind computes array bridge - edges have nei, id
    vector<vector<Edge>> &g;
    int n,m;
    vi dep;
    vector<bool> bridge, check;

    BRIDGE(vector<vector<Edge>> &_g, int _m) : g(_g),m(_m){
        n = g.size();
        dep.resize(n); bridge.resize(m); check.resize(n,false);
        dep[0] = 0; bFind(0);
    }
```

```
    int bFind(int cur,int p = -1){
        check[cur] = true;
        int res = dep[cur];
        for(auto &e:g[cur]){
            if(e.nei == p) continue;
            if(check[e.nei]) res = min(res,dep[e.nei]);
            else{
                dep[e.nei] = dep[cur] + 1;
                int child = bFind(e.nei, cur);
                bridge[e.id] = child > dep[cur];
                res = min(res, child);
            }
        }
        return res;
    }
};
```

4.3 Dijkstra

```
vector<int> dijkstra(const graph_w& G, int s){
    int n = G.size();
    vector<int> dis(n, inf);
    set<ii> S;
    dis[s] = 0;
    S.insert({0, s});
    while(!S.empty()){
        int u = S.begin()->second;
        S.erase(S.begin());

        for(auto& e : G[u]){
            int v = e.first, w = e.second;
            if(dis[v] > dis[u] + w){
                S.erase({dis[v], v});
                dis[v] = dis[u] + w;
                S.insert({dis[v], v});
            }
        }
    }
    return dis;
}
```

4.4 Dinic

```
//
// Dinic algorithm for maximum flow / minimum cut
// time: O(VVE), usually faster, no more than O(maxflow * E)
// space: O(V+E)
//
struct Edge {
  int u, v;
  int cap, flow;
  Edge() {}
  Edge(int u, int v, int cap): u(u), v(v), cap(cap), flow(0) {}
};

struct Dinic {
  int N;
  vector<Edge> E;
  vector<vector<int> > g;
  vector<int> d, pt;

  Dinic(int N): N(N), E(0), g(N), d(N), pt(N) {}

  void addEdge(int u, int v, int cap) {
    if (u != v) {
      E.emplace_back(Edge(u, v, cap));
      g[u].emplace_back(E.size() - 1);
      E.emplace_back(Edge(v, u, 0));
      g[v].emplace_back(E.size() - 1);
    }
  }

  bool BFS(int S, int T) {
    queue<int> q({S});
    fill(d.begin(), d.end(), N + 1);
    d[S] = 0;
    while(!q.empty()) {
      int u = q.front(); q.pop();
      if (u == T) break;
      for (int k: g[u]) {
        Edge &e = E[k];
        if (e.flow < e.cap && d[e.v] > d[e.u] + 1) {
          d[e.v] = d[e.u] + 1;
          q.emplace(e.v);
        }
      }
    }
    return d[T] != N + 1;
  }

  int DFS(int u, int T, int flow = -1) {
    if (u == T || flow == 0) return flow;
    for (int &i = pt[u]; i < g[u].size(); ++i) {
      Edge &e = E[g[u][i]];
      Edge &oe = E[g[u][i]^1];
      if (d[e.v] == d[e.u] + 1) {
        int amt = e.cap - e.flow;
        if (flow != -1 && amt > flow) amt = flow;
        if (int pushed = DFS(e.v, T, amt)) {
          e.flow += pushed;
          oe.flow -= pushed;
          return pushed;
        }
      }
    }
    return 0;
  }
```

```
  int maxFlow(int S, int T) {
    int total = 0;
    while (BFS(S, T)) {
      fill(pt.begin(), pt.end(), 0);
      while (int flow = DFS(S, T))
        total += flow;
    }
    return total;
  }
};
```

4.5 Hungarian Alg

```
// solves 1000 x 1000 in 1 sec. complexity O(mn^2)
vector<int> HungarianMinCost(vector<vi> &a){ // matrix is 1-indexed,
    rows <= cols, ans[i] is the vertex assigned to i
    int n  = a.size() - 1, m = a[0].size() - 1;
    vi u (n+1), v (m+1), p (m+1), way (m+1);
    for (int i=1; i<=n; ++i) {
        p[0] = i;
        int j0 = 0;
        vi minv (m+1, inf);
        vector<char> used (m+1, false);
        do {
            used[j0] = true;
            int i0 = p[j0],  delta = inf,  j1;
            for (int j=1; j<=m; ++j)
                if (!used[j]) {
                    int cur = a[i0][j]-u[i0]-v[j];
                    if (cur < minv[j])
                        minv[j] = cur,  way[j] = j0;
                    if (minv[j] < delta)
                        delta = minv[j],  j1 = j;
                }
            for (int j=0; j<=m; ++j)
                if (used[j])
                    u[p[j]] += delta,  v[j] -= delta;
                else
                    minv[j] -= delta;
            j0 = j1;
        } while (p[j0] != 0);
        do {
            int j1 = way[j0];
            p[j0] = p[j1];
            j0 = j1;
        } while (j0);
    }
    vi ans (n+1);
    for (int j=1; j<=m; ++j)
        ans[p[j]] = j;
        return ans;
}
```

4.6 Strong Connected Components

```
struct SCC
{
    // findSCC computes: label - label of SCC for every vertex, gn -
        the connected components graph
    vi check, reach, label;
    stack<int> st;
    int t = 0;
    vector<bool> hist;
    vector<vector<Edge>>& g, gn;
    vector<vi>comp;
    SCC(vector<vector<Edge>> &_g):g(_g){
        findScc();
    }

    void findScc(){
        int n = g.size();
        check.resize(n,-1); reach.resize(n); label.resize(n); hist.
            resize(n);
        for(int i=0;i<n;i++){
            if(check[i] == -1) dfs1(i);
        }
        //////////////////////////build gn
            ///////////////////////////////////
        gn.resize(comp.size());
        hist.resize(comp.size(),0);
        for(int i=0;i<comp.size();i++){
            for(auto &v:comp[i]){
                for(auto &e:g[v]){
                    if(hist[label[e.nei]] || label[e.nei] == i)
                        continue;
                    gn[i].pb(Edge(e));
                    hist[label[e.nei]] = true;
                }
            }
            for(auto &e:gn[i]) hist[e.nei] = false;
        }
        //
            ///////////////////////////////////////////////////////////////////
    }

    int dfs1(int cur){
        reach[cur] = check[cur] = t++;
        st.push(cur);
        hist[cur] = true;
        for(auto &e:g[cur]){
            if(check[e.nei] == -1){
                reach[cur] = min(reach[cur], dfs1(e.nei));
            }
            else if(hist[e.nei]){
                reach[cur] = min(reach[cur], check[e.nei]);
            }
```

```
        }
        if(reach[cur] == check[cur]){
            comp.pb(vi());
            for(;st.top() != cur; st.pop()){
                comp.back().pb(st.top());
                label[st.top()] = comp.size() - 1;
                hist[st.top()] = false;
            }
            comp.back().pb(st.top());
            label[st.top()] = comp.size() - 1;
            hist[st.top()] = false;
            st.pop();
        }
        return reach[cur];
    }
};
```

5 Miscellaneous

5.1 Aho-Corasick

```
/*
Aho-Corasick algorithm/data structure
Complexity: O(M) when M is the sum of length of all words
To search in text (after build) is O(N) when N is the length of the text
insert words into trie using "push_word"
create the links for the automaton using "create_links" (use after all word had been pushed)
use the automaton using "next_state"
*/
struct AhoCorasick{
    vector<map<char,int>> to;
    vector<bool> endw; //in which state a word is ended (if want which word, maintain a vvb
    vector<int> link;
    int sz=1;
    AhoCorasick(): to(vector<map<char,int>>(1)), endw(vector<bool>(1,0)){}
    void push_word(string& s){
        int cur = 0;
        for(auto& c: s){
            if (!to[cur][c]) {
                to[cur][c] = sz++;
                to.pb(map<char,int>());
                endw.pb(0);
            }
            cur = to[cur][c];
        }
        endw[cur] = 1;
    }
    void create_links(){
        queue<int> q;
        q.push(0);
        link.resize(to.size(),0);
        link[0] = -1;
        int v,u,j;
        char c;
        while(q.size()){
            v = q.front();
            q.pop();
            for(auto& it:to[v]){
                c = it.x;
                u = it.y;
                j = link[v];
                while(j!=-1 && !to[j][c]) j = link[j];
                if (j!=-1) link[u] = to[j][c];
                else link[u] = 0;
                q.push(u);
                endw[u]=endw[u] || endw[link[u]]; //merge the endw (if endw[i] is vector merge them)
            }
        }
    }
    int next_state(int cur, char c){
        while(cur!=-1 && !to[cur][c]) cur = link[cur];
        if (cur==-1) return 0;
        return to[cur][c];
    }
};
```

5.2 FFT

```
struct com { // works also with c++ complex class but twice slower
    double a, b;
    com(double a = 0, double b = 0) : a(a), b(b){}
};
com inline operator + (com l, com r) { return com(l.a + r.a, l.b + r.b); }
com inline operator - (com l, com r) { return com(l.a - r.a, l.b - r.b); }
com inline operator * (com l, com r) { return com(l.a * r.a - l.b * r.b, l.b * r.a + l.a * r.b); }
com inline operator / (com c, double b) { return com(c.a / b, c.b / b); }


void inline dft(vector<com> &a, int len, vi &rev, int tp = 1) {
    const double pi = acos(-1);
    for(int i=0;i<= len ;i++)
        if (rev[i] > i) swap(a[i], a[rev[i]]);

    for (int k = 1; k < len; k <<= 1) {
        com w0(cos(2 * pi / (k << 1)), tp * sin(2 * pi / (k << 1)));
        for (int l = 0; l < len; l += (k << 1)) {
            com w(1);
            for (int i = 0; i < k; ++i, w = w * w0) {
```

```
            com p0 = a[l + i], p1 = w * a[l + k + i];
            a[l + i] = p0 + p1, a[l + k + i] = p0 - p1;
        }
      }
    }
}

vector<com> inline multiply(vector<com> & p,vector<com> & q){//modify
    p,q!!!
    int len, k = 0;
    for (len = 1; len <= p.size() + q.size(); len <<= 1, ++k);
    vector<com> res(len*2, 0);
    p.resize(2*len, 0); q.resize(2*len, 0); res.resize(2*len, 0);
    vi rev(len+1, 0);
    for(int i=0;i <= len;i++)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (k - 1));

    dft(p, len, rev), dft(q,len, rev);
    for(int i=0;i <= len;i++)
        res[i] = p[i] * q[i];

    dft(res, len, rev, -1);
    for(int i=0;i <= len;i++)
        res[i] = res[i] / len;
    return res;
}
```

5.3 KMP

```
vector<int> kmp(string T,string P){
        int n = T.size();
        int m = P.size();
        vector<int> prefix(m,-1);
        int j = -1;
        for(int i = 1;i<m;i++){
            while(j != -1 && P[i] != P[j+1])
                j = prefix[j];
            if(P[i] == P[j+1])
                j++;
            prefix[i] = j;
        }
    j = -1;
        vector<int> pos;
        for(int i = 0;i<n;i++){
            while(j!=-1 && T[i] != P[j+1])
                j = prefix[j];
            if(T[i] == P[j+1])
                j++;
            if(j == m-1)
                pos.push_back(i),j = prefix[j];
        }
        return pos;
}
```

5.4 Mancher

```
ii mancher(string &s){ //return {length, center} when even center is
    middle right
    ii res(0,-1);
    int n = s.size();
    vector<vi> dp(2,vi(n));
    for(int e=0;e<2;++e){
        for(int l=0,r=-1,i=0;i<n;i++){
            int k = (i > r) ? 1 - e : min(dp[e][r - i + l + e], r - i
                + 1);
            while(i - k - e >= 0 && i + k < n && s[i - k - e] == s[i +
                 k]) k++;
            dp[e][i] = k--;
            if(i + k > r){
                l = i - k - e;
                r = i + k;
            }
            res = max(res, {dp[e][i] * 2 - 1 + e, i});
        }
    }
    return res;
}
```

5.5 Order Statistics Tree

```
//add this code. pay attention to #define int64_t int

#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
    tree_order_statistics_node_update
using namespace __gnu_pbds;
typedef tree<
int,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update>
Tree;
///////////// use of unique //////////
sort(v.begin(), v.end());
v.erase(unique(v.begin(), v.end()), v.end());
```

6 Number Theory

6.1 Baby Step Giant Step

```
int BSGS(int a, int b){
    int sq = (int)sqrt(mod+.0) + 1;
    int an = power(a, sq);
```

```
    map<int,int> mp;
    for(int i = 1, val = an; i <= sq; i++, val = val*an%mod) mp[val]=i
        ;
    for(int i = 0, val = b; i < sq; i++, val = val*a%mod)
        if (mp.count(val))
            return ((mp[val] * sq - i) % mod + mod) % mod;
    return -1;
}
```

6.2 Chinese Reminder Theorem

```
// Chinese remainder theorem (special case): find z such that
// z % m1 = r1, z % m2 = r2.  Here, z is unique modulo M = lcm(m1, m2)
    .
// Return (z, M).  On failure, M = -1.
ii chinese_remainder_theorem(int m1, int r1, int m2, int r2) {
        int s, t;
        int g = extended_euclid(m1, m2, s, t);
        if (r1%g != r2%g) return make_pair(0, -1);
        return make_pair(mod(s*r2*m1 + t*r1*m2, m1*m2) / g, m1*m2 / g)
            ;
}

// Chinese remainder theorem: find z such that
// z % m[i] = r[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (m[i]).  Return (z, M). On
// failure, M = -1. Note that we do not require the a[i]'s
// to be relatively prime.
ii chinese_remainder_theorem(const vi &m, const vi &r) {
        ii ret = make_pair(r[0], m[0]);
        for (int i = 1; i < m.size(); i++) {
                ret = chinese_remainder_theorem(ret.second, ret.first,
                     m[i], r[i]);
                if (ret.second == -1) break;
        }
        return ret;
}
```

6.3 Extended GCD

```
int extended_euclid(int a, int b, int &x, int &y) {
        int xx = y = 0;
        int yy = x = 1;
        while (b) {
                int q = a / b;
                int t = b; b = a%b; a = t;
                t = xx; xx = x - q*xx; x = t;
                t = yy; yy = y - q*yy; y = t;
        }
        return a;
}
```

6.4 Factorize

```
vi factor(int n){
    vi p;
    for(int i = 2; i * i <= n; i++){
        if(n % i == 0) p.pb(i);
        while(n % i == 0) n /= i;
    }
    if(n > 1) p.pb(n);
    return p;
}
```

6.5 Find Primitive Element

```
int findPrimitive(){ //mod is prime, generator of multiplicative F_mod
    vi primes = factor(mod - 1);
    mt19937 gen(chrono::steady_clock::now().time_since_epoch().count()
        );
    std::uniform_int_distribution<> dis(1, mod - 1);
    for(int g;true;){
        g = dis(gen);
        bool flag = true;
        for(auto &p:primes){
            if(power(g, (mod - 1) / p) == 1){
                flag = false;
                break;
            }
        }
        if(flag) return g;
    }
}
```

6.6 First N Inverses

```
void first_n_inverses(int n, int mod, vi &inv) // assign inv[i] for i
    =0,...,n
{
    inv.resize(n + 1);
    inv[0] = inv[1] = 1;
    for(int i=2;i<=n;i++) inv[i] = inv[mod % i] * (mod - mod / i) %
        mod;
}
```

6.7 Modular Linear Equation

```
// finds all solutions to ax = b (mod n)
vi modular_linear_equation_solver(int a, int b, int n) {
        int x, y;
        vi ret;
        int g = extended_euclid(a, n, x, y);
```

```
        if (!(b%g)) {
                x = mod(x*(b / g), n);
                for (int i = 0; i < g; i++)
                        ret.push_back(mod(x + i*(n / g), n));
        }
        return ret;
}
```

7 Trees

7.1 Binary Lift

```
struct LCA
{
    vector<vector<Edge>> &g;
    int n, logn = 1;
    vi dep;
    vector<vi> anc;
    LCA(vector<vector<Edge>>& _g) : g(_g)
    {
        n = g.size();
        for(int i = n-1;i;i/=2) ++logn;
        dep.resize(n); anc.resize(n);
        dep[0] = 0; dfs(0);
    }
    void dfs(int cur, int p = -1)
    {
        anc[cur].resize(logn,-1);
        anc[cur][0] = p;
        for(int i = 1;anc[cur][i-1] != -1;i++) anc[cur][i] = anc[anc[
            cur][i-1]][i-1];
        for(auto &e:g[cur]){
            if(e.nei == p) continue;
            dep[e.nei] = dep[cur] + 1;
            dfs(e.nei, cur);
        }
    }
    int lift(int a, int d)
    {
        for(int i = logn-1; i>=0; i--) if(d >= 1LL<<i) a = anc[a][i],
            d -= 1LL<<i;
        return a;
    }
    int query(int a, int b)
    {
        if(dep[a] < dep[b]) swap(a,b);
        a = lift(a, dep[a] - dep[b]);
        if(a == b) return a;
        for(int i = logn - 1;i>=0;i--){
            if(anc[a][i] != anc[b][i]){
                a = anc[a][i];
                b = anc[b][i];
            }
        }
        return anc[a][0];
    }
};
```

7.2 Tree Eccentricity

```
struct EccentricityTree{ //change max to min, > to <, -inf to inf for
    distances to leaf
    int n;
    vector<vector<Edge>> g;
    vi hight, goUp, ans; // goUp - eccentricy not in subtree, ans =
        max(goUp, hight)
    EccentricityTree(vector<vector<Edge>> &g) : g(g){
        n = g.size();
        hight.resize(n,0); goUp.resize(n);  ans.resize(n);
        if(g[0].size() <= 1) goUp[0] = 0; //leaf
        dfs(0);
        dfs2(0);
    }

    void dfs(int cur, int p = -1){
        ii maxi(-inf,-inf); //two maximal hights of cur
        for(auto &e : g[cur]){
            if(e.nei == p) continue;
            dfs(e.nei, cur);
            maxi.y = max(maxi.y, hight[e.nei] + e.val);
            if(maxi.y > maxi.x) swap(maxi.y, maxi.x);
        }
        if(g[cur].size() == 1 && p != -1) maxi.x = 0; //leaf
        hight[cur] = maxi.x;
        for(auto &e : g[cur]){
            if(e.nei == p) continue;
            if(hight[e.nei] + e.val != maxi.x) goUp[e.nei] = e.val +
                maxi.x;
            else goUp[e.nei] = e.val + maxi.y;
        }
    }

    void dfs2(int cur, int p = -1){
        ans[cur] = max(hight[cur], goUp[cur]);
        for(auto &e : g[cur]){
            if(e.nei == p) continue;
            goUp[e.nei] = max(goUp[e.nei], e.val + goUp[cur]);
            dfs2(e.nei, cur);
        }
    }
};
```