

# 1 Data Structure

## 1.1 DSU

```

1 class DSU{
2 public:
3     DSU(int n){
4         this->n = n;
5         reset();
6     }
7     int n;
8     vector<int> boss;
9     vector<int> rank;
10    vector<int> size;
11    void reset(){
12        this->boss.resize(n);
13        this->rank.resize(n,0);
14        this->size.resize(n,0);
15        for(int i =0;i<n;i++){
16            boss[i] = i;
17        }
18    }
19    int find(int x){
20        if(boss[x]!= x){
21            boss[x] = find(boss[x]);
22        }
23        return boss[x];
24    }
25    int get_size(int x){
26        return size[find(x)];
27    }
28    void merge(int x, int y){
29        int a = find(x);
30        int b = find(y);
31        if(a!=b){
32            if(rank[a]<rank[b]){
33                boss[a] = b;
34                size[b] += size[a];
35            }else if (rank[a]<rank[b]){
36                boss[b] = a;
37                size[a] += size[b];
38            }else{
39                boss[a] = b;
40                size[b] += size[a];
41                rank[b]++;
42            }
43        }
44    }
45    bool aresame(int a,int b){
46        return find(a)==find(b);
47    }
48 };

```

## 1.2 Monotonic Queue

```

1 class Monotonic_queue{
2 private:
3     deque<int> qu;
4 public:
5     void push(int n){

```

```

6         while(!qu.empty() && qu.back()<n){
7             qu.pop_back();
8         }
9         qu.push_back(n);
10    }
11    int max(){
12        return qu.front();
13    }
14    int min(){
15        return qu.back();
16    }
17    int size(){
18        return qu.size();
19    }
20    void pop(){
21        qu.pop_front();
22    }
23 };

```

## 1.3 BIT

```

1 class BIT{
2 public:
3     vector<int> bit;
4     int N;
5     BIT(int n){
6         this->N = n;
7         this->bit.resize(n);
8     }
9     void update(int x,int d){
10        while(x<=N){
11            bit[x] +=d;
12            x +=x&(-x); // lowest bit in x;
13        }
14    }
15    int query(int x){
16        int res = 0;
17        while(x){
18            res+= bit[x];
19            x -= x&(-x);
20        }
21        return res;
22    }
23 };

```

## 1.4 Segment Tree

```

1 class SegmentTree{
2 private:
3     const int n;
4     const vl arr;
5     // vl st;
6     vl summ;
7     vl minn;
8     vl maxx;
9     vl tag;
10    void pull(int l,int r,int v){
11        if(r-l==1){
12            return;
13        }
14        // st[v]=st[2*v+1]+st[2*v+2];

```

```

14        int mid=(l+r)/2;
15        push(1,mid,2*v+1);
16        push(mid,r,2*v+2);
17        summ[v]=summ[2*v+1]+summ[2*v+2];
18        // minn[v]=min(minn[2*v+1],minn[2*v+2]);
19        // maxx[v]=max(maxx[2*v+1],maxx[2*v+2]);
20    }
21    void push(int l,int r,int v){
22        summ[v]+=tag[v]*(r-l);
23        if(r-l==1){
24            return tag[v]=0,void();
25        }
26        tag[2*v+1]+=tag[v];
27        tag[2*v+2]+=tag[v];
28        tag[v]=0;
29    }
30    void build(int l,int r,int v=0){
31        if(r-l==1){
32            summ[v]=arr[l];
33            // summ[v]=minn[v]=maxx[v]=arr[l];
34            return;
35        }
36        int mid=(l+r)/2;
37        build(l,mid,2*v+1);
38        build(mid,r,2*v+2);
39        pull(l,r,v);
40    }
41 public:
42    SegmentTree(vl& arr,int _n):arr(_arr),n(_n){
43        assert(arr.size()==n);
44        summ.assign(4*n,0);
45        // minn.assign(4*n,1e9);
46        // maxx.assign(4*n,-1e9);
47        tag.assign(4*n,0);
48        build(0,arr.size());
49    }
50    void modify(int x,int val,int l,int r,int v=0){
51    }
52    // query sum
53    loli query(int L,int R,int l,int r,int v=0){
54        // dbn(L,R,l,r,v)
55        push(1,r,v);
56        if(l==L && R==r){
57            return summ[v];
58            return minn[v];
59            return maxx[v];
60        }
61        int mid=(l+r)/2;
62        if(R<=mid){
63            return query(L,R,l,mid,2*v+1);
64        }else if(mid<=L){
65            return query(L,R,mid,r,2*v+2);
66        }else{
67            return query(L,mid,l,mid,2*v+1)+query(mid,R,mid,r,2*v+2);
68        }
69    }
70    // plus 'val' to every element in [L,R)
71    void update(int L,int R,loli val,int l,int r,int v=0){
72        // dbn(L,R,l,r,v)
73        push(1,r,v);
74        if(l==L && R==r){
75            tag[v]+=val;
76            push(1,r,v);
77            return;
78        }

```

```

79     int mid=(l+r)/2;
80     if(R<=mid)
81         update(L,R, val, l, mid, 2*v+1);
82     else if(mid<=L)
83         update(L,R, val, mid, r, 2*v+2);
84     else
85         update(L, mid, val, l, mid, 2*v+1), update(mid, R, val,
86             mid, r, 2*v+2);
87     pull(l, r, v);
88 }
89 };
90 void solve(){
91     int n,q;
92     cin>>n>>q;
93     vl arr(n);
94     for(auto&x:arr)
95         cin>>x;
96     SegmentTree st(arr,n);
97     while(q--){
98         int op=0;
99         // str op;
100        cin>>op;
101        if(op&1){
102            loli l,r,val;
103            cin>>l>>r>>val;
104            assert(r>=l);
105            st.update(l-1,r,val,0,n);
106            // loli k,u;
107            // cin>>k>>u;
108            // st.update(k-1,k,u-arr[k-1],0,n);
109            // arr[k-1]=u;
110        }else{
111            int x,y;
112            cin>>x>>y;
113            assert(y>=x);
114            cout<<st.query(x-1,y,0,n)<<endl;
115        }
116    }
117 }

```

## 1.5 Sparse Table

```

1 int a[N], sp[___lg(N) + 1][N]{};
2 void init(int n) { //0-based
3     for (int i = 0; i < n; ++i) {
4         sp[0][i] = a[i];
5     }
6     for (int i = 0; i < ___lg(n); ++i) {
7         for (int j = 0; j+(1<<i) < n; ++j) {
8             sp[i+1][j] = max(sp[i][j], sp[i][j+(1<<i)]);
9         }
10    }
11 }
12 int query(int l, int r) { //[l, r]
13     int p = ___lg(r - l + 1);
14     return max(sp[p][l], sp[p][r-(1<<p)+1]);
15 }

```

## 1.6 Monotonic Stack

```

1 vector<int> monotonic_stack(vector<int> nums){
2     int n = nums.size();
3     vector<int> res(n);
4     stack<int> st;
5     for(int i = n-1; i>=0; i--){
6         while(!st.empty() && st.top()<=nums[i]){
7             st.pop();
8         }
9         if(st.empty())res[i] = -1;
10        else res[i] = st.top();
11        st.push(nums[i]);
12    }
13    return res;
14 }

```

## 2 Flow

### 2.1 Dinic

```

1 #define maxn 2005
2 #define INF 0x3f3f3f3f
3 struct MaxFlow{
4     struct edge{
5         int to, cap, flow, rev;
6         edge(int v, int c, int f, int r) : to(v), cap(c),
7             flow(f), rev(r) {}
8     };
9     vector<edge> G[maxn];
10    int s, t, dis[maxn], cur[maxn], vis[maxn];
11    void add_edge(int from, int to, int cap){
12        G[from].push_back(edge(to, cap, 0, G[to].size()));
13        G[to].push_back(edge(from, 0, 0, G[from].size()-1));
14    }
15    bool bfs(){
16        memset(dis, -1, sizeof(dis));
17        queue<int> qu;
18        qu.push(s);
19        dis[s] = 0;
20        while (!qu.empty()) {
21            int from = qu.front();
22            qu.pop();
23            for (auto &e: G[from]) {
24                if (dis[e.to]==-1 && e.cap != e.flow) {
25                    dis[e.to] = dis[from] + 1;
26                    qu.push(e.to);
27                }
28            }
29        }
30        return dis[t]!=-1;
31    }
32    int dfs(int from, int cap){
33        if(from==t || cap==0)return cap;
34        for(int &i = cur[from]; i<G[from].size(); i++){
35            edge &e = G[from][i];
36            if(dis[e.to]==dis[from]+1 && e.flow!=e.cap){
37                int df = dfs(e.to, min(e.cap-e.flow, cap));
38                if(df){
39                    e.flow+=df;
40                    G[e.to][e.rev].flow-=df;
41                    return df;
42                }
43            }
44        }
45    }

```

```

42    }
43    }
44    dis[from] = -1;
45    return 0;
46 }
47 int Maxflow(int s, int t){
48     this->s = s, this->t = t;
49     int flow = 0;
50     int df;
51     while(bfs()){
52         memset(cur, 0, sizeof(cur));
53         while(df = dfs(s, INF)){
54             flow +=df;
55         }
56     }
57     return flow;
58 }
59 };
60 int main(){
61     int n = 4, m = 6;
62     MaxFlow maxflow;
63     for(int i = 0; i<m; i++){
64         int a, b, cap;
65         cin >>a>>b>>cap;
66         maxflow.add_edge(a, b, cap);
67     }
68     cout << maxflow.Maxflow(1,3)<<endl;;
69 }

```

## 3 Geometry

### 3.1 Sort by Angle

```

1 bool cmp(pii a, pii b) {
2     #define is_neg(k) (k.y < 0 || (k.y == 0 && k.x < 0));
3     int A = is_neg(a), B = is_neg(b);
4     if (A != B)
5         return A < B;
6     if (cross(a, b) == 0)
7         return (a.x*a.x + a.y*a.y) < (b.x*b.x + b.y*b.y);
8     return cross(a, b) > 0;
9 }

```

### 3.2 Convex Hull

```

1 using pdd = pair<double, double>;
2 #define F first
3 #define S second
4 pdd operator-(pdd a, pdd b) {
5     return {a.F - b.F, a.S - b.S};
6 }
7 double cross(pdd a, pdd b) {
8     return a.F * b.S - a.S * b.F;
9 }
10 void solve() {
11     int n;
12     cin >> n;
13     vector<pdd> pnts;

```

```

14 for (int i = 0; i < n; ++i) {
15     double x, y;
16     cin >> x >> y;
17     pnts.push_back(x, y);
18 }
19 sort(iter(pnts));
20 vector<pdd> hull;
21 for (int i = 0; i < 2; ++i) {
22     int t = hull.size();
23     for (pdd j: pnts) {
24         while(hull.size() - t >= 2 && cross(j - hull[hull.size()
25             () - 2], hull.back() - hull[hull.size() - 2]) >=
26             0)
27             hull.pop_back();
28         hull.push_back(j);
29     }
30     hull.pop_back();
31     reverse(iter(pnts));
32 }
33 double area = 0;
34 for (int i=0; i < hull.size(); ++i){
35     area += cross(hull[i], hull[(i + 1) % hull.size()]);
36 }
37 area /= 2.0;

```

### 3.3 Point in Polygon

```

1 const ll inf = 2000000000;
2 struct Point {
3     ll x, y;
4     Point(ll x = 0, ll y = 0):x(x), y(y){}
5     Point operator+(const Point p) const {
6         return Point(x + p.x, y + p.y); }
7     Point operator-(const Point p) const {
8         return Point(x - p.x, y - p.y); }
9     ll operator*(const Point p) const { //dot
10         return x * p.x + y * p.y; }
11     ll operator^(const Point p) const { //cross
12         return x * p.y - y * p.x; }
13 };
14 bool onseg(Point a, Point b, Point o) {
15     return ((a - o) ^ (b - o)) == 0 && ((a - o) * (b - o)) <=
16         0;
17 }
18 int ori(Point a, Point b, Point o) {
19     ll w = (a - o) ^ (b - o);
20     return (w ? (w > 0 ? 1 : -1) : 0);
21 }
22 bool inters(Point a, Point b, Point c, Point d) {
23     if (onseg(a, b, c) || onseg(a, b, d)) return 1;
24     if (onseg(c, d, a) || onseg(c, d, b)) return 1;
25     if (ori(a, b, c) * ori(a, b, d) < 0 && ori(c, d, a) * ori(c, d, b) < 0) return 1;
26     return 0;
27 }
28 Point poly[maxn];
29 void solve(int n, Point p) {
30     poly[n] = poly[0];
31     int cnt = 0;
32     for (int i = 0; i < n; ++i) {
33         if (onseg(poly[i], poly[i + 1], p)) {
34             cnt = -1;

```

```

34         break;
35     }
36     if (inters(poly[i], poly[i + 1], p, Point(inf, p.y))) {
37         ++cnt;
38     }
39     Point hi = (poly[i].y > poly[i + 1].y ? poly[i] : poly[i
40         + 1]);
41     if (hi.y == p.y && hi.x > p.x) {
42         --cnt;
43     }
44 }
45 if (cnt < 0)
46     cout << "BOUNDARY\n";
47 else if (cnt % 2)
48     cout << "INSIDE\n";
49 else
50     cout << "OUTSIDE\n";

```

### 3.4 MinCoveringCircle

```

1 double dis(pdd a, pdd b) {
2     double dx = a.x - b.x, dy = a.y - b.y;
3     return sqrt(dx*dx + dy*dy);
4 }
5 double sq(double x) {
6     return x * x;
7 }
8 pdd excenter(pdd p1, pdd p2, pdd p3) {
9     double a1 = p1.x - p2.x, a2 = p1.x - p3.x;
10    double b1 = p1.y - p2.y, b2 = p1.y - p3.y;
11    double c1 = (sq(p1.x) - sq(p2.x) + sq(p1.y) - sq(p2.y)) /
12        2;
13    double c2 = (sq(p1.x) - sq(p3.x) + sq(p1.y) - sq(p3.y)) /
14        2;
15    double dd = a1*b2 - a2*b1;
16    return {(c1*b2 - c2*b1) / dd, (a1*c2 - a2*c1) / dd};
17 }
18 void solve(pdd a[], int n) {
19     shuffle(a, a + n, rng);
20     pdd center = a[0];
21     double r = 0;
22     for (int i = 1; i < n; ++i) {
23         if (dis(center, a[i]) <= r) continue;
24         center = a[i], r = 0;
25         for (int j = 0; j < i; ++j) {
26             if (dis(center, a[j]) <= r) continue;
27             center.x = (a[i].x + a[j].x) / 2;
28             center.y = (a[i].y + a[j].y) / 2;
29             r = dis(center, a[i]);
30             for (int k = 0; k < j; ++k) {
31                 if (dis(center, a[k]) <= r) continue;
32                 center = excenter(a[i], a[j], a[k]);
33                 r = dis(center, a[i]);
34             }
35         }
36     }
37     cout << fixed << setprecision(10) << r << '\n';
38     cout << center.x << ' ' << center.y << '\n';

```

## 4 Graph

### 4.1 Bipartite Matching

```

1 const int MAXN = 100;
2
3 struct Bipartite_matching{
4     int mx[MAXN], my[MAXN], vy[MAXN]; //matchX, matchY,
5     visitY
6     vector<int> edge[MAXN]; //adjacent list;
7     int x_cnt;
8     bool dfs(int x){
9         for(auto y: edge[x]){ //對 x 可以碰到的邊進行檢查
10             if(vy[y] == 1) continue; //避免遞迴 error
11
12             vy[y] = 1;
13             if(my[y] == -1 || dfs(my[y])){ //分析 3
14                 mx[x] = y;
15                 my[y] = x;
16                 return true;
17             }
18         }
19         return false; //分析 4
20     }
21
22     int bipartite_matching(){
23         memset(mx, -1, sizeof(mx)); //分析 1,2
24         memset(my, -1, sizeof(my));
25         int ans = 0;
26         for(int i = 0; i < x_cnt; i++){ //對每一個 x 節點進
27             行 DFS(最大匹配)
28             memset(vy, 0, sizeof(vy));
29             if(dfs(i)) ans++;
30         }
31         return ans;
32     }
33     vector<vector<int>>> get_match(){
34         vector<vector<int>>> res;
35         for(int i = 0; i < x_cnt; i++){
36             if(mx[i] != -1){
37                 res.push_back({i, mx[i]});
38             }
39         }
40         return res;
41     }
42     void add_edge(int i, int j){
43         edge[i].push_back(j);
44     }
45     void init(int x){
46         x_cnt = x;
47     }
48 };
49 int main(){
50     int n, m;
51     Bipartite_matching bm;
52     for(int i = 0; i < m; i++){
53         int a, b; cin >> a >> b;
54         bm.add_edge(a, b);
55     }
56     bm.init(n);
57     cout << bm.bipartite_matching() << endl;
58     auto match = bm.get_match();
59     for(auto t: match){

```

```

58     cout << t[0]<<" "<<t[1]<<endl;
59 }
60 }
61 }

```

## 4.2 Tarjan SCC

```

1 const int n = 16;
2 vector<vector<int>> graph;
3 int visit[n], low[n], t = 0;
4 int st[n], top = 0;
5 bool instack[n];
6 int contract[n]; // 每個點收縮到的點
7 vector<vector<int>> block;
8 void dfs(int x, int parent){
9     // cout << x << endl;
10    visit[x] = low[x] = ++t;
11    st[top++] = x;
12    instack[x] = true;
13    for(auto to: graph[x]){
14        if(!visit[to])
15            dfs(to, x);
16
17        if(instack[to])
18            low[x] = min(low[x], low[to]);
19    }
20    if(visit[x] == low[x]){ // scc ㊦ 最早拜訪的
21        int j;
22        block.push_back({});
23        do{
24            j = st[--top];
25            instack[j] = false;
26            block[block.size() - 1].push_back(j);
27            contract[j] = x;
28        } while(j != x);
29    }
30 }
31 int main(){
32     for(int i = 0; i < n; i++){
33         if(!visit[i])
34             dfs(i, i);
35     }
36     for(auto t: block){
37         for(auto x: t){
38             cout << x << " ";
39         } cout << endl;
40     }
41 }

```

## 4.3 Bridge

```

1 const int n = 9;
2 vector<vector<int>> graph;
3 vector<int> visit(n, 0);
4 vector<int> trace(n, 0);
5 vector<vector<int>> bridge;
6 int t = 0;
7 void dfs(int x, int parent){
8     visit[x] = ++t;

```

```

9     trace[x] = x; // 最高祖先預設 ㊦ 自己
10    for(auto to: graph[x]){
11        if(visit[to]){ // back edge
12            if(to != parent){
13                trace[x] = to;
14            }
15        }
16        else{ // tree edge
17            dfs(to, x);
18            if(visit[trace[to]] < visit[trace[x]])
19                trace[x] = trace[to];
20
21            // 子樹回不到祖先暨自身。
22            if(visit[trace[to]] > visit[x])
23                bridge.push_back({x, to});
24        }
25    }
26 } // call for() dfs(i, -1)
27 int main(){
28     for(int i = 0; i < 9; i++){
29         if(!visit[i])
30             dfs(i, -1);
31     }
32     for(auto x: bridge){
33         cout << x[0] << " " << x[1] << endl;
34     }
35 }

```

## 4.4 2 SAT

```

1 class TwoSAT{
2 public:
3     TwoSAT(int n): n(n), graph(2 * n), visited(2 * n, false) {}
4     void addClause(int a, int b) { // 0-base;
5         a *= 2;
6         b *= 2;
7         // Add implications (~a => b) and (~b => a)
8         graph[a ^ 1].push_back(b);
9         graph[b ^ 1].push_back(a);
10    }
11    bool solve() { // Find SCCs and check for contradictions
12        for(int i = 0; i < 2 * n; ++i) {
13            if(!visited[i]) {
14                dfs1(i);
15            }
16        }
17        reverse(processOrder.begin(), processOrder.end()); // topological sort
18        for(int i = 0; i < 2 * n; ++i) {
19            visited[i] = false;
20        }
21        for(int node: processOrder) {
22            if(!visited[node]) {
23                scc.clear();
24                dfs2(node);
25                if(!checkSCCConsistency()) {
26                    return false;
27                }
28            }
29        }
30        return true;
31    }

```

```

32    }
33
34 private:
35     int n;
36     vector<vector<int>> graph;
37     vector<bool> visited;
38     vector<int> processOrder;
39     vector<int> scc;
40
41     void dfs1(int node) {
42         visited[node] = true;
43         for(int neighbor: graph[node]) {
44             if(!visited[neighbor]) {
45                 dfs1(neighbor);
46             }
47         }
48         processOrder.push_back(node);
49     }
50
51     void dfs2(int node) {
52         visited[node] = true;
53         scc.push_back(node);
54         for(int neighbor: graph[node]) {
55             if(!visited[neighbor]) {
56                 dfs2(neighbor);
57             }
58         }
59     }
60
61     bool checkSCCConsistency() {
62         for(int node: scc) {
63             if(find(scc.begin(), scc.end(), node ^ 1) != scc
64                .end()) {
65                 return false; // Contradiction found in the
66                               // same SCC
67             }
68         }
69         return true;
70     }
71
72 int main() {
73     int n, m; // Number of variables and clauses
74     TwoSAT twoSat(n);
75     for(int i = 0; i < m; ++i) {
76         int a, b;
77         twoSat.addClause(a, b);
78     }
79     if(twoSat.solve()) {
80         cout << "Satisfiable" << endl;
81     } else {
82         cout << "Unsatisfiable" << endl;
83     }
84 }

```

## 4.5 Kosaraju 2DFS

```

1 const int n = 16;
2 vector<vector<int>> graph;
3 vector<vector<int>> reverse_graph;
4 int visit[n];
5 int contract[n]; // 每個點收縮到的點
6 vector<vector<int>> block;
7 vector<int> finish; // fake topological sort

```

```

8 // need graph and reverse graph
9 void dfs1(int x){
10     visit[x] = true;
11     for(auto to:graph[x]){
12         if(!visit[to]){
13             dfs1(to);
14         }
15     }
16     finish.push_back(x);
17 }
18 void dfs2(int x,int c){
19     contract[x] = c;
20     block[c].push_back(x);
21     visit[x] = true;
22     for(auto to:reverse_graph[x]){
23         if(!visit[to]){
24             dfs2(to,c);
25         }
26     }
27 }
28 int main(){
29     graph = {};
30     reverse_graph = {};
31
32     for(int i =0;i<n;i++){
33         if (!visit[i])
34             dfs1(i);
35     }
36     int c =0;
37     memset(visit,0,sizeof(visit));
38     for(int i = n-1;i>=0;i--){
39         if(!visit[finish[i]]){
40             block.push_back({});
41             dfs2(finish[i],c++);
42         }
43     }
44     for(auto t: block){
45         for(auto x:t){
46             cout << x <<" ";
47         }cout <<endl;
48     }
49 }

```

## 4.6 Dijkstra

```

1 #define maxn 200005
2 vector<int> dis(maxn,-1);
3 vector<int> parent(maxn,-1);
4 vector<bool> vis(maxn,false);
5 vector<vector<pair<int,int>>> graph;
6 void dijkstra(int source){
7     dis[source] =0 ;
8
9     priority_queue<pair<int,int>,vector<pair<int,int>>>,
10         greater<pair<int,int>>> pq;
11     pq.push({0,source});
12     while(!pq.empty()){
13         int from = pq.top().second;
14         pq.pop();
15         // cout <<vis[from]<<endl;
16         if(vis[from]) continue;
17         vis[from] = true;
18         for(auto next : graph[from]){

```

```

18         int to = next.second;
19         int weight = next.first;
20         // cout <<from<<' ' <<to<<' ' <<weight;
21         if(dis[from]+weight< dis[to] || dis[to]==-1){
22             dis[to] = dis[from]+weight;
23             parent[to] = from;
24             pq.push({dis[from]+weight,to});
25         }
26     }
27 }
28 }
29 int main(){
30     int startpoint;
31     dijkstra(startpoint);
32     //dis and parent
33 }

```

## 4.7 Floyd Warshall

```

1 #define maxn 2005
2 vector<vector<int>>> dis(maxn,vector<int>(maxn,999999));
3 vector<vector<int>>> mid(maxn,vector<int>(maxn,-1));
4 vector<vector<pair<int,int>>> graph;
5
6 void floyd_warshall(int n){ // n is n nodes
7     for(int i =0;i<n;i++){
8         for(auto path:graph[i]){
9             dis[i][path.second] = path.first;
10         }
11     }
12     for (int i=0; i<n; i++)
13         dis[i][i] = 0;
14     for (int k=0; k<n; k++){
15         for (int i=0; i<n; i++){
16             for (int j=0; j<n; j++){
17                 if (dis[i][k] + dis[k][j] < dis[i][j] || dis[i][j]
18                     ]==-1){
19                     dis[i][j] = dis[i][k] + dis[k][j];
20                     mid[i][j] = k; // 由 i 點走到 j 點經過了 k 點
21                 }
22             }
23         }
24     }
25 void find_path(int s, int t){ // 印出最短路徑
26     if (mid[s][t] == -1) return; // 圖中有中繼點就結束
27     find_path(s, mid[s][t]); // 前半段最短路徑
28     cout << mid[s][t]; // 中繼點
29     find_path(mid[s][t], t); // 後半段最短路徑
30 }
31 int main(){
32     int n;
33     floyd_warshall(n);
34     for(int i =0;i<4;i++){
35         for(int j = 0 ; j <4;j++){
36             cout << dis[i][j]<<" ";
37         }
38     }
39     find_path(0,2);
40 }

```

## 4.8 Articulation Vertex

```

1 const int n = 9;
2 int t = 0;
3 vector<int> disc(n, -1); // Discovery time
4 vector<int> low(n, -1); // Low time
5 vector<int> parent_array(n, -1); // Parent in DFS tree
6 vector<bool> visited(n, false);
7 vector<bool> is_articulation(n, false);
8 vector<vector<int>>> graph;
9 void dfs_articulation(int node, int parent){
10     visited[node] = true;
11     disc[node] = t;
12     low[node] = t;
13     t++;
14     int children = 0;
15
16     for (int neighbor : graph[node])
17     {
18         if (!visited[neighbor])
19         {
20             children++;
21             parent_array[neighbor] = node;
22             dfs_articulation(neighbor, node);
23             low[node] = min(low[node], low[neighbor]);
24
25             if (low[neighbor] >= disc[node] && parent != -1)
26             {
27                 is_articulation[node] = true;
28             }
29         }
30         else if (neighbor != parent)
31         {
32             low[node] = min(low[node], disc[neighbor]);
33         }
34     }
35
36     if (parent == -1 && children > 1)
37     {
38         is_articulation[node] = true;
39     }
40 } //call for() dfs(i,-1)
41 int main(){
42     for (int i = 0; i < n; ++i) {
43         if (!visited[i]) {
44             dfs_articulation(i, -1);
45         }
46     }
47     cout << "Articulation Points: ";
48     for (int i = 0; i < n; ++i) {
49         if (is_articulation[i]) {
50             cout << i << " ";
51         }
52     }cout << endl;
53 }

```

## 4.9 Topological Sort

```

1 vector<vector<int>>> graph;
2 vector<int> visit(10,0);
3 vector<int> order;
4 int n;

```

```

5 bool cycle; // 記圖DFS的過程中是否偵測到環
6 void DFS(int i){ //reverse(order) is topo
7     if (visit[i] == 1) {cycle = true; return;}
8     if (visit[i] == 2) return;
9     visit[i] = 1;
10    for(auto to : graph[i])
11        DFS(to);
12    visit[i] = 2;
13    order.push_back(i);
14 } //for() if(!vis[i])DFS(i)
15 int main() {
16     for (int i=0; i<n; ++i){
17         if (!visit[i])
18             DFS(i);
19     }
20     if (cycle)
21         cout << "圖上有環";
22     else
23         for (int i=n-1; i>=0; --i)
24             cout << order[i];
25 }

```

## 4.10 Planar

```

1 class Graph {
2 public:
3     int V;
4     vector<vector<int>>> adj;
5     Graph(int vertices) : V(vertices), adj(vertices) {}
6     void addEdge(int u, int v) {
7         adj[u].push_back(v);
8         adj[v].push_back(u);
9     }
10 };
11
12 bool containsSubgraph(const Graph& graph, const vector<int>&
    subgraph) {
13     unordered_set<int> subgraphVertices(subgraph.begin(),
        subgraph.end());
14     for (int vertex : subgraphVertices) {
15         for (int neighbor : graph.adj[vertex]) {
16             if (subgraphVertices.count(neighbor) == 0) {
17                 bool found = true;
18                 for (int v : subgraph) {
19                     if (v != vertex && v != neighbor) {
20                         if (graph.adj[v].size() < 3) {
21                             found = false;
22                             break;
23                         }
24                     }
25                 }
26                 if (found)
27                     return true;
28             }
29         }
30     }
31     return false;
32 }
33
34 bool isPlanar(const Graph& graph) {
35     // Subgraphs isomorphic to K and K ,
36     vector<int> k5 = {0, 1, 2, 3, 4}; // Vertices of K

```

```

37     vector<int> k33a = {0, 1, 2}; // Vertices of K
38     , (part A)
39     vector<int> k33b = {3, 4, 5}; // Vertices of K
40     , (part B)
41
42     if (containsSubgraph(graph, k5) || containsSubgraph(graph,
        k33a) || containsSubgraph(graph, k33b)) {
43         return false; // The graph is non-planar
44     }
45     return true; // The graph is planar
46 }
47
48 int main() {
49     int vertices, edges;
50     Graph graph(vertices);
51     for (int i = 0; i < edges; ++i) {
52         int u, v; cin >> u >> v;
53         graph.addEdge(u, v);
54     }
55     if (isPlanar(graph)) {
56         cout << "The graph is planar." << endl;
57     } else {
58         cout << "The graph is non-planar." << endl;
59     }
60 }

```

## 4.11 Heavy Light Decomposition

```

1 int dep[N], pa[N], sz[N], nxt[N];
2 int id[N], rt[N];
3 int dfs(int u, int lst, int d = 0) {
4     dep[u] = d;
5     pa[u] = lst;
6     sz[u] = 1;
7     nxt[u] = -1;
8     for (int v: g[u]) {
9         if (v == lst) continue;
10        sz[u] += dfs(v, u, d + 1);
11        if (nxt[u] == -1 || sz[v] > sz[nxt[u]]) {
12            nxt[u] = v;
13        }
14    }
15    return sz[u];
16 }
17 int tn = 0;
18 void mapId(int u, int lst, int root) {
19     id[u] = ++tn;
20     rt[u] = root;
21     if (~nxt[u]) mapId(nxt[u], u, root);
22     for (int v: g[u]) {
23         if (v == lst || v == nxt[u]) continue;
24         mapId(v, u, v);
25     }
26 }
27 void solve() {
28     while (rt[a] != rt[b]) {
29         if (dep[rt[a]] > dep[rt[b]]) swap(a, b);
30         //...
31         b = pa[rt[b]];
32     }
33     if (a != b) {
34         if (id[a] > id[b]) swap(a, b);
35         //...
36     } else {

```

## 4.12 Centroid Decomposition

```

1 int sz[maxn]{};
2 bool ok[maxn]{};
3 int get_subtree_size(int u, int lst) {
4     sz[u] = 1;
5     for (int v: g[u]) {
6         if (v == lst || ok[v]) continue;
7         sz[u] += get_subtree_size(v, u);
8     }
9     return sz[u];
10 }
11 int get_centroid(int u, int lst, int tree_size) {
12     for (int v: g[u]) {
13         if (v == lst || ok[v]) continue;
14         if (2 * sz[v] >= tree_size) {
15             return get_centroid(v, u, tree_size);
16         }
17     }
18     return u;
19 }
20 void centroid_decomp(int u = 1) { //1-based
21     int centroid = get_centroid(u, u, get_subtree_size(u, u));
22     //...
23     ok[centroid] = 1;
24     for (int v: g[centroid]) if (!ok[v]) {
25         centroid_decomp(v);
26     }
27 }

```

## 5 Math

### 5.1 fpow

```

1 ll fpow(ll b, ll p, ll mod) {
2     ll res = 1;
3     while (p) {
4         if (p & 1) res = res * b % mod;
5         b = b * b % mod, p >>= 1;
6     }
7     return res;
8 }

```

### 5.2 extgcd

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int extgcd(int a,int b,int &x,int &y){ //a*x +b*y = 1
5     if(b==0){

```

```

6     x = 1;
7     y = 0;
8     return a; //到達遞歸邊界開始向上一層返回
9 }
10 int r = extgcd(b, a % b, x, y);
11 int temp = y; //把x y變成上一層的
12 y = x - (a / b) * y;
13 x = temp;
14 return r; //得到a b的最大公因數
15 }
16 int main() {
17     int a = 55, b = 80;
18     int x, y; //a*x+b*y = 1;
19     int GCD = extgcd(a, b, x, y);
20 }

```

### 5.3 EulerTotientFunction

```

1 ll phi[maxn];
2 for (int i = 0; i < maxn; ++i) {
3     phi[i] = i;
4 }
5 for (int i = 2; i < maxn; ++i) if (phi[i] == i) {
6     phi[i] = i - 1; //prime
7     for (int j = 2; i * j < maxn; ++j) { //overflow
8         phi[i * j] = (phi[i * j] / i) * (i - 1);
9     }
10 }

```

### 5.4 FFT

```

1 //OI Wiki
2 #include <complex>
3 using cd = complex<double>;
4 const double PI = acos(-1);
5 void change(vector<cd> &y) {
6     vector<int> rev(y.size());
7     for (int i = 0; i < y.size(); ++i) {
8         rev[i] = rev[i >> 1] >> 1;
9         if (i & 1) {
10             rev[i] |= y.size() >> 1;
11         }
12     }
13     for (int i = 0; i < y.size(); ++i) {
14         if (i < rev[i]) {
15             swap(y[i], y[rev[i]]);
16         }
17     }
18 }
19 void fft(vector<cd> &y, bool inv) {
20     change(y);
21     for (int h = 2; h <= y.size(); h <= 1) {
22         cd wn(cos(2 * PI / h), sin(2 * PI / h));
23         for (int j = 0; j < y.size(); j += h) {
24             cd w(1, 0);
25             for (int k = j; k < j + h / 2; ++k) {
26                 cd u = y[k];
27                 cd t = w * y[k + h / 2];
28                 y[k] = u + t;

```

```

29         y[k + h / 2] = u - t;
30         w = w * wn;
31     }
32 }
33 }
34 if (inv) {
35     reverse(begin(y) + 1, end(y));
36     for (int i = 0; i < y.size(); ++i) {
37         y[i] /= y.size();
38     }
39 }
40 }
41 void solve() {
42     int n;
43     int m = 1 << (lg(n) + 1); //power of 2
44     vector<cd> a(m), b(m);
45     //...
46     fft(a, 0);
47     fft(b, 0);
48     vector<cd> c(m);
49     for (int i = 0; i < m; ++i) {
50         c[i] = a[i] * b[i];
51     }
52     fft(c, 1);
53     for (auto p: c) {
54         int ans = int(p.real() + 0.25);
55     }
56 }

```

## 6 Misc

### 6.1 pbds

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4
5 template<typename T>
6 using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
7     tree_order_statistics_node_update>;
8
9 int32_t main() {
10     ordered_set<int64_t> rbt;
11     // .insert(x); .erase(x)
12     // .lower_bound(x); .upper_bound(x): iter
13     // .find_by_order(k): find k-th small value(iter)
14     // .order_of_key(x): return x is k-th big
15     // .join(rbt2): merge with no multiple same element
16     // .split(key, rbt2): rbt keeps value <= key, others to rbt2
17 }

```

### 6.2 Misc

```

1 mt19937 rng(chrono::steady_clock::now().time_since_epoch().
2     count());
3 int randint(int lb, int ub) {
4     return uniform_int_distribution<int>(lb, ub)(rng);
5 }

```

```

4 } //static unsigned x = 19; ++(x *= 0xdefaced);
5
6 #define SECS ((double)clock() / CLOCKS_PER_SEC)
7
8 struct KeyHasher {
9     size_t operator()(const Key& k) const {
10         return k.first + k.second * 100000;
11     };
12 typedef unordered_map<Key, int, KeyHasher> map_t;
13
14 __lg
15 __gcd
16
17 __builtin_popcount // 二進位有幾個1
18 __builtin_clz // 左起第一個1之前0的個數
19 __builtin_parity // 1的個數的奇偶性

```

### 6.3 Mo's Algorithm

```

1 struct Query {
2     int L, R;
3     //...
4 };
5 vector<Query> query;
6 void solve() { //K = n / sqrt(q)
7     sort(iter(query), [&](Query &a, Query &b) {
8         if (a.L / K != b.L / K) return a.L < b.L;
9         return a.L / K % 2 ? a.R < b.R : a.R > b.R;
10    });
11     int L = 0, R = 0;
12     for (auto x: query) {
13         while (R < x.R) add(arr[++R]);
14         while (L > x.L) add(arr[--L]);
15         while (R > x.R) sub(arr[R--]);
16         while (L < x.L) sub(arr[L--]);
17         //...
18     }
19 }

```

## 7 String

### 7.1 Hashing

```

1 const ll P = 401, M = 998244353;
2
3 ll hashes[10005], modp[10005];
4 ll hashp(string s, bool saveval) {
5     ll val = 0;
6     int index = 0;
7     for (char c: s) {
8         val = ((val * P) % M + c) % M;
9         if (saveval) hashes[index++] = val;
10    }
11    return val;
12 }
13 void init(int base, int exp) {
14     ll b = 1;

```



```

15 modp[0] = 1;
16 for (int i = 0; i < exp; i++) {
17     b = (b * base) % M;
18     modp[i + 1] = b;
19 }
20 }
21 ll subseq(int l, int r) { //[1, r]
22     if (l == 0) return hashes[r];
23     return ((hashes[r] - hashes[l-1] * modp[r-l+1]) % M + M) %
24     M;

```

## 7.2 Trie

```

1 struct node {
2     int ch[26]{};
3     int cnt = 0;
4 };
5 struct Trie {
6     vector<node> t;
7     void init() {
8         t.clear();
9         t.emplace_back(node());
10    }
11    void insert(string s) {
12        int ptr = 0;
13        for (char i: s) {
14            if (!t[ptr].ch[i - 'a']) {
15                t[ptr].ch[i - 'a'] = (int)t.size();
16                t.emplace_back(node());
17            }
18            ptr = t[ptr].ch[i - 'a'];
19        }
20        t[ptr].cnt++;
21    }
22 } trie;

```

## 7.3 Zvalue

```

1 vector<int> Zvalue(string &s) { //t + # + s
2     vector<int> Z(s.size());
3     int x = 0, y = 0;
4     for (int i=0; i<s.size(); ++i) {
5         Z[i] = max(0, min(y - i + 1, Z[i - x]));
6         while (i + Z[i] < s.size() && s[Z[i]] == s[i + Z[i]])
7             x = i, y = i + Z[i], ++Z[i];
8     }
9     return Z;
10 }

```

## 7.4 KMP

```

1 int F[maxn]{};
2 vector<int> match(string& s, string& t) {
3     int p = F[0] = -1;
4     for (int i = 1; i < t.size(); ++i) {

```

```

5         while (p != -1 && t[p + 1] != t[i]) p = F[p];
6         if (t[p + 1] == t[i]) ++p;
7         F[i] = p;
8     }
9     p = -1;
10    vector<int> v;
11    for (int i = 0; i < s.size(); ++i) {
12        while (p != -1 && t[p + 1] != s[i]) p = F[p];
13        if (t[p + 1] == s[i]) ++p;
14        if (p == t.size() - 1) v.push_back(i - p), p = F[p];
15    }
16    return v; //0-based
17 }

```

## 7.5 Manacher

```

1 int z[maxn * 2]{};
2 int manacher(string& s) {
3     string t = "#";
4     for (char c: s) t += c, t += '#';
5     int l = 0, r = 0, ans = 0; //l: mid, r: right
6     for (int i = 1; i < t.size(); ++i) {
7         z[i] = (r > i ? min(z[2 * l - i], r - i) : 1);
8         while (i - z[i] >= 0 && i + z[i] < t.size()) {
9             if (t[i - z[i]] == t[i + z[i]])
10                ++z[i];
11             else
12                 break;
13         }
14         if (i + z[i] > r) r = i + z[i], l = i;
15     }
16     for (int i = 1; i < t.size(); ++i) ans = max(ans, z[i] - 1);
17     ;
18     string res;
19     for (int i = 1; i < t.size(); ++i) if (ans == z[i] - 1) {
20         for (int j = i - ans + 1; j < i + ans; ++j) if (t[j] != '#') {
21             res += t[j];
22         }
23         break;
24     }
25     return ans;

```

## 8 Tree

### 8.1 LCA

```

1 int n, logn, t=0;
2 vector<vector<int>>> graph;
3 vector<vector<int>>> ancestor;
4 vector<int> tin, tout;
5 void dfs(int x){
6     tin[x] = t++;
7     for(auto y: graph[x]){
8         if(y!= ancestor[x][0]){
9             ancestor[y][0] = x;
10            dfs(y);

```

```

11        }
12    }
13    tout[x] = t++;
14 }
15 bool is_ancestor(int x, int y){
16     return tin[x] <= tin[y] && tout[x] >= tout[y];
17 }
18 void table(){
19     for (int i=1; i<logn; i++)// 上兩輩祖先、上四輩祖先、上八輩
20         祖先、...
21         for (int x=0; x<n; ++x)
22             ancestor[x][i] = ancestor[ancestor[x][i-1]][i-1];
23 }
24 int kth_ancestor(int x, int k){
25     for (int i=0; i<logn; i++)// k 拆解成二進位位數，找到第k祖
26         先。不斷上升逼近之。
27         if (k & (1<<i))
28             x = ancestor[x][i];
29     return x;
30 }
31 void rooted_tree(int root){// build the tree with root at "
32     root"
33     ancestor[root][0] = root;
34     dfs(root);
35     table();
36 }
37 int LCA(int x,int y){
38     if (is_ancestor(x, y)) return x;
39     if (is_ancestor(y, x)) return y;
40     for (int i=logn-1; i>=0; i--)
41         if (!is_ancestor(ancestor[x][i], y))
42             x = ancestor[x][i];
43     return ancestor[x][0];
44 }
45 int main(){
46     graph = {
47         {1,2},
48         {3},
49         {5,6},
50         {7},
51         {},
52         {},
53         {8},
54         {4},
55     };
56     n = 9;
57     logn = ceil(log2(n));
58     ancestor.resize(n, vector<int>(logn));
59     tin.resize(n);
60     tout.resize(n);
61
62     rooted_tree(0);
63     while(true){
64         int a,b;
65         cin >>a>>b;
66         cout <<LCA(a,b)<<endl;
67     }
68 }
69 int main(){
70     n = 9;
71     logn = ceil(log2(n));
72     ancestor.resize(n, vector<int>(logn));

```



```

73     tin.resize(n);
74     tout.resize(n);
75     rooted_tree(0);
76     while(true){
77         int a,b;
78         cin >>a>>b;
79         cout <<LCA(a,b)<<endl;;
80     }
81 }

```

## 8.2 Diameter

```

1 vector<vector<int>> graph;
2 int diameter = 0;
3 int dfs(int start, int parent){
4     int h1 = 0, h2 = 0;
5     for (auto child : graph[start]){
6         if (child != parent){
7             int h = dfs(child, start) + 1;
8             if (h > h1){
9                 h2 = h1;
10                h1 = h;
11            }
12            else if (h > h2){
13                h2 = h;
14            }
15        }
16    }
17    diameter = max(diameter, h1 + h2);
18    return h1;
19 }
20 // call diameter
21 int main(){
22     dfs(0,-1);
23     cout << diameter<<endl;
24 }

```

## 8.3 Radius

```

1 // Perform DFS to find the farthest node and its distance
  from the given node
2 pair<int, int> dfs(int node, int distance, vector<bool> &
  visited, const vector<vector<int>> &adj_list){
3     visited[node] = true;
4     int max_distance = distance;
5     int farthest_node = node;
6
7     for (int neighbor : adj_list[node]){
8         if (!visited[neighbor]){
9             auto result = dfs(neighbor, distance + 1, visited
10                , adj_list);
11             if (result.first > max_distance){
12                 max_distance = result.first;
13                 farthest_node = result.second;
14             }
15         }
16     }
17     return make_pair(max_distance, farthest_node);
18 }

```

```

19 // Calculate the radius of the tree using DFS
20 int tree_radius(const vector<vector<int>> &adj_list){
21     int num_nodes = adj_list.size();
22     vector<bool> visited(num_nodes, false);
23
24     // Find the farthest node from the root (node 0)
25     auto farthest_result = dfs(0, 0, visited, adj_list);
26
27     // Reset visited array
28     fill(visited.begin(), visited.end(), false);
29
30     // Calculate the distance from the farthest node
31     int radius = dfs(farthest_result.second, 0, visited,
32         adj_list).first;
33
34     return radius;
35 }
36 int main() {
37     vector<vector<int>> adj_list;
38     int radius = tree_radius(adj_list);
39     cout << "Tree radius: " << radius << endl;
40     return 0;
41 }

```

# 9 Z\_Original\_Code/Data\_Structure

## 9.1 dsu-class

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 class DSU{
6 public:
7     DSU(int n){
8         this->n = n;
9         reset();
10    }
11    int n;
12    vector<int> boss;
13    vector<int> rank;
14    vector<int> size;
15
16    void reset(){
17        this->boss.resize(n);
18        this->rank.resize(n,0);
19        this->size.resize(n,0);
20        for(int i =0;i<n;i++){
21            boss[i] = i;
22        }
23    }
24    int find(int x){
25        if(boss[x]!= x){
26            boss[x] = find(boss[x]);
27        }
28        return boss[x];
29    }
30    int get_size(int x){
31        return size[find(x)];
32    }

```

```

33 void merge(int x, int y){
34     int a = find(x);
35     int b = find(y);
36     // if(a!=b){
37     //     boss[a] = b;
38     //     size[b] += size[a];
39     // }
40     if(a!=b){
41         if(rank[a]<rank[b]){
42             boss[a] = b;
43             size[b] += size[a];
44         }else if (rank[a]<rank[b]){
45             boss[b] = a;
46             size[a] += size[b];
47         }else{
48             boss[a] = b;
49             size[b] += size[a];
50             rank[b]++;
51         }
52     }
53 }
54 bool aresame(int a,int b){
55     return find(a)==find(b);
56 }
57 };
58 int main(){
59     DSU dsu(10);
60
61     dsu.merge(0, 1);
62     dsu.merge(2, 3);
63     dsu.merge(4, 5);
64     dsu.merge(6, 7);
65
66     cout << "Are 0 and 1 connected? " << (dsu.aresame(0, 1) ?
67         "Yes" : "No") << endl;
68     cout << "Are 2 and 3 connected? " << (dsu.aresame(2, 3) ?
69         "Yes" : "No") << endl;
70     cout << "Are 4 and 5 connected? " << (dsu.aresame(4, 5) ?
71         "Yes" : "No") << endl;
72     cout << "Are 6 and 7 connected? " << (dsu.aresame(6, 7) ?
73         "Yes" : "No") << endl;
74     cout << "Are 1 and 2 connected? " << (dsu.aresame(1, 2) ?
75         "Yes" : "No") << endl;
76
77     dsu.merge(1, 2);
78
79     cout << "Are 0 and 2 connected? " << (dsu.aresame(0, 2) ?
80         "Yes" : "No") << endl;
81     cout << "Are 1 and 3 connected? " << (dsu.aresame(1, 3) ?
82         "Yes" : "No") << endl;
83
84     return 0;
85 }

```

## 9.2 monotonic-queue

```

1 //ref:leetcode
2 #include<bits/stdc++.h>
3
4 using namespace std;
5
6 class Monotonic_queue{
7 private:

```

```

8 deque<int> qu;
9 public:
10 void push(int n){
11     while(!qu.empty() && qu.back() < n){
12         qu.pop_back();
13     }
14     qu.push_back(n);
15 }
16 int max(){
17     return qu.front();
18 }
19 int min(){
20     return qu.back();
21 }
22 int size(){
23     return qu.size();
24 }
25 void pop(){
26     qu.pop_front();
27 }
28 };
29
30 vector<int> maxSlidingWindow(vector<int> nums, int k) {
31     Monotonic_queue window;
32     vector<int> res;
33     for (int i = 0; i < nums.size(); i++) {
34         if (i < k - 1) {
35             window.push(nums[i]);
36         } else {
37             window.push(nums[i]);
38             res.push_back(window.max());
39             if (window.max() == nums[i - k + 1]) {
40                 window.pop();
41             }
42         }
43     }
44     return res;
45 }
46
47 }
48 int main(){
49     vector<int> nums = {1,3,-1,-3,5,3,6,7};
50     int k = 3;
51     vector<int> res = maxSlidingWindow(nums,k);
52     for (auto r:res) cout << r << " ";
53 }

```

### 9.3 BIT

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 class BIT{
5 public:
6     vector<int> bit;
7     int N;
8     BIT(int n){
9         this->N = n;
10        this->bit.resize(n);
11    }
12    void update(int x,int d){
13        while(x<=N){
14            bit[x] +=d;

```

```

15        x +=x&(-x); // lowest bit in x;
16    }
17 }
18 int query(int x){
19     int res = 0;
20     while(x){
21         res += bit[x];
22         x -= x&(-x);
23     }
24     return res;
25 }
26 };
27 // Driver program to test above functions
28 int main()
29 {
30     vector<int> freq = {0, 2, 1, 1, 3, 2, 3, 4, 5, 6, 7, 8,
31         9};
32     int n = freq.size();
33     BIT bit(n);
34     for (int i = 1; i < n; i++) {
35         bit.update(i, freq[i]);
36     }
37     for (int i = 1; i < n; i++) {
38         cout << bit.query(i) << " ";
39     }
40     cout << endl;
41     for (int i = 1; i < n; i++) {
42         bit.update(i, -1);
43     }
44     for (int i = 1; i < n; i++) {
45         cout << bit.query(i) << " ";
46     }
47     cout << endl;
48 }

```

### 9.4 segment-tree-simple-add

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 struct node{
5     int left;
6     int right;
7     int value;
8 };
9 vector<node> segment_tree;
10 void build(int left,int right,int x,vector<int> & nums){
11     segment_tree[x].left = left;
12     segment_tree[x].right = right;
13     // cout << left << " " << right << " " << x << endl;
14     if(left == right){ // here is leaf
15         segment_tree[x].value = nums[left];
16         return;
17     }
18     int mid = (left+right)/2;
19     build(left, mid, x<<1, nums);
20     build(mid+1, right, x<<1|1, nums);
21     segment_tree[x].value = segment_tree[x<<1].value +
        segment_tree[x<<1|1].value;
22 }
23 void modify(int position,int x,int value){
24     if(segment_tree[x].left == position && segment_tree[x].
        right == position){ // here is leaf
25         segment_tree[x].value = value;
26         return;

```

```

27     }
28     int mid = (segment_tree[x].left+segment_tree[x].right)/2;
29
30     if(position <= mid){
31         modify(position, x<<1, value);
32     } else {
33         modify(position, x<<1|1, value);
34     }
35     segment_tree[x].value = segment_tree[x<<1].value +
        segment_tree[x<<1|1].value;
36 }
37 int query(int i,int j,int x){
38     // cout << i << " " << j << " " << segment_tree[x].left << " " <<
        segment_tree[x].right << endl;
39     int res = 0;
40     int left = segment_tree[x].left;
41     int right = segment_tree[x].right;
42     int mid = (left+right)/2;
43     if(segment_tree[x].left == i && segment_tree[x].right == j){
44         return segment_tree[x].value;
45     }
46     if(i > mid) return query(i, j, x*2+1);
47     if(mid >= j) return query(i, j, x*2);
48     return query(i, mid, x*2) + query(mid+1, j, x*2+1);
49 }
50
51 int main(){
52     vector<int> nums =
        {1,10,5,148,78,2,56,231,5,64,65,32,1,8};
53     int n = nums.size();
54     segment_tree.resize(n*4);
55     build(0,n-1,1,nums);
56     modify(5,1,100);
57     // cout << "++++++\n";
58     for (int i = 0; i < n; i++) {
59         for (int j = i; j < n; j++) {
60             cout << query(i, j, 1) << " ";
61         }
62     }
63 }

```

### 9.5 monotonic-stack

```

1 /*
2 input: array A
3 output: array B
4 bi is the value aj such that j>i and aj>bi (j)
5 ex:
6 A = [2,1,2,4,3]
7 B = [4,3,4,-1,-1]
8 */
9 #include<bits/stdc++.h>
10
11 using namespace std;
12
13 vector<int> monotonic_stack(vector<int> nums){
14     int n = nums.size();
15     vector<int> res(n);
16     stack<int> st;
17     for (int i = n-1; i >= 0; i--) {
18         while(!st.empty() && st.top() <= nums[i]) {
19             st.pop();

```

```

20 // we want the value greater than nums[i], so we
    pop the value smaller and equal nums[i]
21 }
22 if(st.empty())res[i] = -1;
23 else res[i] = st.top();
24 st.push(nums[i]);
25 }
26 return res;
27 }
28
29 int main(){
30     vector<int> res = monotonic_stack({2,1,2,4,3});
31     for(auto r:res){
32         cout << r<<" ";
33     }
34 }

```

## 10 Z\_Original\_Code/Flow

### 10.1 dicnic

```

1 #include <bits/stdc++.h>
2 #define maxn 2005
3 #define INF 0x3f3f3f3f
4 using namespace std;
5 struct MaxFlow{
6     struct edge{
7         int to, cap, flow, rev;
8         edge( int v, int c, int f, int r) : to(v), cap(c),
          flow(f), rev(r) {}
9     };
10    vector<edge> G[maxn];
11    int s, t, dis[maxn], cur[maxn], vis[maxn];
12    void add_edge(int from, int to, int cap){
13        G[from].push_back(edge(to, cap, 0, G[to].size()));
14        G[to].push_back(edge(from, 0, 0, G[from].size()-1));
15    }
16    bool bfs(){
17        memset(dis, -1, sizeof(dis));
18        queue<int> qu;
19        qu.push(s);
20        dis[s] = 0;
21        while (!qu.empty()) {
22            int from = qu.front();
23            qu.pop();
24            for (auto &e: G[from]) {
25                if (dis[e.to]==-1 && e.cap != e.flow) {
26                    dis[e.to] = dis[from] + 1;
27                    qu.push(e.to);
28                }
29            }
30        }
31        return dis[t]!=-1;
32    }
33    int dfs(int from, int cap){
34        if(from==t || cap==0)return cap;
35        for(int &i = cur[from]; i<G[from].size(); i++){
36            edge &e = G[from][i];
37            if(dis[e.to]==dis[from]+1 && e.flow!=e.cap){
38                int df = dfs(e.to, min(e.cap-e.flow, cap));
39                if(df){

```

```

40                e.flow+=df;
41                G[e.to][e.rev].flow-=df;
42                return df;
43            }
44        }
45        dis[from] = -1;
46        return 0;
47    }
48    int Maxflow(int s, int t){
49        this->s = s, this->t = t;
50        int flow = 0;
51        int df;
52        while(bfs()){
53            memset(cur, 0, sizeof(cur));
54            while(df = dfs(s, INF)){
55                flow +=df;
56            }
57        }
58        return flow;
59    }
60 }
61 };

```

## 11 Z\_Original\_Code/Graph

### 11.1 planar

```

1 #include <iostream>
2 #include <vector>
3 #include <unordered_set>
4
5 using namespace std;
6
7 class Graph {
8 public:
9     int V;
10    vector<vector<int>>> adj;
11    Graph(int vertices) : V(vertices), adj(vertices) {}
12    void addEdge(int u, int v) {
13        adj[u].push_back(v);
14        adj[v].push_back(u);
15    }
16 };
17
18 bool containsSubgraph(const Graph& graph, const vector<int>&
    subgraph) {
19    unordered_set<int> subgraphVertices(subgraph.begin(),
        subgraph.end());
20    for (int vertex : subgraphVertices) {
21        for (int neighbor : graph.adj[vertex]) {
22            if (subgraphVertices.count(neighbor) == 0) {
23                bool found = true;
24                for (int v : subgraph) {
25                    if (v != vertex && v != neighbor) {
26                        if (graph.adj[v].size() < 3) {
27                            found = false;
28                            break;
29                        }
30                    }
31                }
32                if (found)

```

```

33                return true;
34            }
35        }
36    }
37    return false;
38 }
39
40 bool isPlanar(const Graph& graph) {
41    // Subgraphs isomorphic to K and K ,
42    vector<int> k5 = {0, 1, 2, 3, 4}; // Vertices of K
43    vector<int> k33a = {0, 1, 2}; // Vertices of K
44    vector<int> k33b = {3, 4, 5}; // Vertices of K
45    // (part A)
46    // (part B)
47
48    if (containsSubgraph(graph, k5) || containsSubgraph(graph,
        k33a) || containsSubgraph(graph, k33b)) {
49        return false; // The graph is non-planar
50    }
51    return true; // The graph is planar
52 }
53
54 int main() {
55     int vertices, edges;
56     cin >> vertices;
57     cin >> edges;
58
59     Graph graph(vertices);
60     for (int i = 0; i < edges; ++i) {
61         int u, v;
62         cin >> u >> v;
63         graph.addEdge(u, v);
64     }
65     if (isPlanar(graph)) {
66         cout << "The graph is planar." << endl;
67     } else {
68         cout << "The graph is non-planar." << endl;
69     }
70     return 0;
71 }

```

### 11.2 Dijkstra

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define maxn 200005
5 vector<int> dis(maxn, -1);
6 vector<int> parent(maxn, -1);
7 vector<bool> vis(maxn, false);
8 vector<vector<pair<int, int>>> graph;
9 void dijkstra(int source){
10     dis[source] = 0;
11
12     priority_queue<pair<int, int>, vector<pair<int, int>>,
        greater<pair<int, int>>> pq;
13     pq.push({0, source});
14     while (!pq.empty()) {
15         int from = pq.top().second;
16         pq.pop();
17         // cout << vis[from] << endl;
18         if (vis[from]) continue;

```

```

19     vis[from] = true;
20     for(auto next : graph[from]){
21         int to = next.second;
22         int weight = next.first;
23         // cout << from << ' ' << to << ' ' << weight;
24         if(dis[from]+weight < dis[to] || dis[to]==-1){
25             dis[to] = dis[from]+weight;
26             parent[to] = from;
27             pq.push({dis[from]+weight, to});
28         }
29     }
30 }
31 }
32 }
33 int main(){
34     graph = {
35         {{4,1},{5,3}},
36         {{3,3}},
37         {{}},
38         {{4,0},{2,1},{7,2}}
39     };
40     dijkstra(0);
41     for(int i =0;i<4;i++){
42         cout << dis[i]<<" ";
43     }
44     for(int i =0;i<4;i++){
45         cout << parent[i]<<" ";
46     }
47 }

```

### 11.3 Floyd\_Warshall

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define maxn 2005
5 vector<vector<int>>> dis(maxn, vector<int>(maxn, 9999999));
6 vector<vector<int>>> mid(maxn, vector<int>(maxn, -1));
7 vector<vector<pair<int, int>>> graph;
8
9 void floyd_warshall(int n){ // n is n nodes
10     for(int i =0;i<n;i++){
11         for(auto path:graph[i]){
12             dis[i][path.second] = path.first;
13         }
14     }
15     for(int i=0; i<n; i++){
16         dis[i][i] = 0;
17         for(int k=0; k<n; k++){
18             for(int i=0; i<n; i++){
19                 for(int j=0; j<n; j++){
20                     if(dis[i][k] + dis[k][j] < dis[i][j] || dis[i][j]
21                        ==-1){
22                         dis[i][j] = dis[i][k] + dis[k][j];
23                         mid[i][j] = k; // 由 i 點走到 j 點經過了 k 點
24                     }
25                 }
26             }
27         }
28     }
29     void find_path(int s, int t){ // 印出最短路徑
30         if (mid[s][t] == -1) return; // 沒有中繼點就結束
31         find_path(s, mid[s][t]); // 前半段最短路徑

```

```

31     cout << mid[s][t]; // 中繼點
32     find_path(mid[s][t], t); // 後半段最短路徑
33 }
34 int main(){
35     graph = {
36         {{4,1},{5,3}},
37         {{3,3}},
38         {{}},
39         {{4,0},{2,1},{7,2}}
40     };
41     floyd_warshall(4);
42     for(int i =0;i<4;i++){
43         for(int j = 0 ; j <4;j++){
44             cout << dis[i][j]<<" ";
45         }
46         cout << endl;
47     }
48     find_path(0,2);
49 }

```

### 11.4 2\_sat

```

1 #include <iostream>
2 #include <vector>
3 #include <stack>
4 #include <algorithm>
5
6 using namespace std;
7
8 class TwoSAT {
9 public:
10     TwoSAT(int n) : n(n), graph(2 * n), visited(2 * n, false) {}
11
12     void addClause(int a, int b) { // 0-base;
13         a *=2;
14         b *=2;
15         // Add implications (~a => b) and (~b => a)
16         graph[a ^ 1].push_back(b);
17         graph[b ^ 1].push_back(a);
18     }
19
20     bool solve() {
21         // Find SCCs and check for contradictions
22         for (int i = 0; i < 2 * n; ++i) {
23             if (!visited[i]) {
24                 dfs1(i);
25             }
26         }
27         reverse(processOrder.begin(), processOrder.end());
28         // topological sort
29         for (int i = 0; i < 2 * n; ++i) {
30             visited[i] = false;
31         }
32         for (int node : processOrder) {
33             if (!visited[node]) {
34                 scc.clear();
35                 dfs2(node);
36                 if (!checkSCCConsistency()) {
37                     return false;
38                 }
39             }
40         }

```

```

41         return true;
42     }
43
44 private:
45     int n;
46     vector<vector<int>>> graph;
47     vector<bool> visited;
48     vector<int> processOrder;
49     vector<int> scc;
50
51     void dfs1(int node) {
52         visited[node] = true;
53         for (int neighbor : graph[node]) {
54             if (!visited[neighbor]) {
55                 dfs1(neighbor);
56             }
57         }
58         processOrder.push_back(node);
59     }
60
61     void dfs2(int node) {
62         visited[node] = true;
63         scc.push_back(node);
64         for (int neighbor : graph[node]) {
65             if (!visited[neighbor]) {
66                 dfs2(neighbor);
67             }
68         }
69     }
70
71     bool checkSCCConsistency() {
72         for (int node : scc) {
73             if (find(scc.begin(), scc.end(), node ^ 1) != scc
74                 .end()) {
75                 return false; // Contradiction found in the
76                             // same SCC
77             }
78         }
79         return true;
80     }
81
82 int main() {
83     int n, m;
84     cin >> n >> m; // Number of variables and clauses
85
86     TwoSAT twoSat(n);
87
88     for (int i = 0; i < m; ++i) {
89         int a, b;
90         cin >> a >> b;
91         twoSat.addClause(a, b);
92     }
93
94     if (twoSat.solve()) {
95         cout << "Satisfiable" << endl;
96     } else {
97         cout << "Unsatisfiable" << endl;
98     }
99
100     return 0;

```

## 11.5 bipartite\_matching

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int MAXN = 100;
4
5 struct Bipartite_matching{
6     int mx[MAXN], my[MAXN], vy[MAXN]; //matchX, matchY,
7     visitY
8     vector<int> edge[MAXN]; //adjacent list;
9     int x_cnt;
10    bool dfs(int x){
11        for(auto y: edge[x]){ //對 x 可以碰到的邊進行檢查
12            if(vy[y] == 1) continue; //避免遞迴 error
13
14            vy[y] = 1;
15            if(my[y] == -1 || dfs(my[y])){ //分析 3
16                mx[x] = y;
17                my[y] = x;
18                return true;
19            }
20        }
21        return false; //分析 4
22    }
23
24    int bipartite_matching(){
25        memset(mx, -1, sizeof(mx)); //分析 1,2
26        memset(my, -1, sizeof(my));
27        int ans = 0;
28        for(int i = 0; i < x_cnt; i++){ //對每一個 x 節點進
29            行 DFS(最大匹配)
30            memset(vy, 0, sizeof(vy));
31            if(dfs(i)) ans++;
32        }
33        return ans;
34    }
35    vector<vector<int>> get_match(){
36        vector<vector<int>> res;
37        for(int i = 0; i < x_cnt; i++){
38            if(mx[i] != -1){
39                res.push_back({i, mx[i]});
40            }
41        }
42        return res;
43    }
44    void add_edge(int i, int j){
45        edge[i].push_back(j);
46    }
47    void init(int x){
48        x_cnt = x;
49    }
50};
51int main(){
52    /*
53    0 3
54    0 4
55    1 3
56    1 5
57    2 3
58    2 4
59    2 5
60    */
61    Bipartite_matching bm;
62    for(int i = 0; i < 7; i++){

```

```

62        int a, b;
63        cin >> a >> b;
64        bm.add_edge(a, b);
65    }
66    bm.init(3);
67    cout << bm.bipartite_matching() << endl;
68    auto match = bm.get_match();
69    for(auto t: match){
70        cout << t[0] << " " << t[1] << endl;
71    }
72 }
73 }

```

## 11.6 tarjan-SCC

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int n = 16;
4 vector<vector<int>> graph;
5 int visit[n], low[n], t = 0;
6 int st[n], top = 0;
7 bool instack[n];
8 int contract[n]; // 每個點收縮到的點
9 vector<vector<int>> block;
10 void dfs(int x, int parent){
11     // cout << x << endl;
12     visit[x] = low[x] = ++t;
13     st[top++] = x;
14     instack[x] = true;
15     for(auto to: graph[x]){
16         if(!visit[to]){
17             dfs(to, x);
18
19             if(instack[to])
20                 low[x] = min(low[x], low[to]);
21         }
22         if(visit[x] == low[x]){ //scc ㊦ 最早拜訪的
23             int j;
24             block.push_back({});
25             do{
26                 j = st[--top];
27                 instack[j] = false;
28                 block[block.size() - 1].push_back(j);
29                 contract[j] = x;
30             } while(j != x);
31         }
32     }
33 }
34 int main(){
35     graph = {
36         {1},
37         {3, 4, 5},
38         {6},
39         {2},
40         {7},
41         {11, 15},
42         {2, 3},
43         {4, 6, 9},
44         {},
45         {},
46         {},
47         {15},
48         {14},
49         {13, 5},

```

```

49         {15},
50         {10, 12, 13}
51     };
52     for(int i = 0; i < n; i++){
53         if(!visit[i])
54             dfs(i, i);
55     }
56     for(auto t: block){
57         for(auto x: t){
58             cout << x << " ";
59         }
60     }
61 }

```

## 11.7 topological\_sort

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 vector<vector<int>> graph;
4 vector<int> visit(10, 0);
5 vector<int> order;
6 int n;
7 bool cycle; // 記 ㊦DFS 的過程中是否偵測到環
8 void DFS(int i)
9 {
10     if (visit[i] == 1) {cycle = true; return;}
11     if (visit[i] == 2) return;
12     visit[i] = 1;
13     for(auto to: graph[i])
14         DFS(to);
15     visit[i] = 2;
16     order.push_back(i);
17 }
18
19 int main() {
20     graph = {
21         {1, 2},
22         {3},
23         {3, 4},
24         {4},
25         {}
26     };
27     n = 5;
28     cycle = false;
29     for (int i = 0; i < n; ++i){
30         if (!visit[i])
31             DFS(i);
32     }
33     if (cycle)
34         cout << "圖上有環";
35     else
36         for (int i = n - 1; i >= 0; --i)
37             cout << order[i];
38 }

```

## 12 Z\_Original\_Code/Math

### 12.1 extgcd

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int extgcd(int a,int b,int &x,int &y)//擴展歐幾里得算法
5 {
6     if(b==0)
7     {
8         x = 1;
9         y = 0;
10        return a;    //到達遞歸邊界開始向上一層返回
11    }
12    int r = extgcd(b,a%b,x,y);
13    int temp=y;    //把x y變成上一層的
14    y = x - (a / b) * y;
15    x = temp;
16    return r;    //得到a b的最大公因數
17 }
18
19 int main(){
20     int a = 55,b = 80;
21     int x,y;
22     int GCD = extgcd(a,b,x,y);
23     cout << "GCD: " << GCD << endl;;
24     cout << x << " " << y << endl;
25     cout << a*x+b*y << endl;
26 }

```

## 13 Z\_Original\_Code/Tree

### 13.1 LCA

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 int n;
4 int logn;
5 vector<vector<int>> graph;
6 vector<vector<int>> ancestor;
7 vector<int> tin,tout;
8 int t = 0;
9 void dfs(int x){
10     tin[x] = t++;
11     for(auto y:graph[x]){
12         if(y!= ancestor[x][0]){
13             ancestor[y][0] = x;
14             dfs(y);
15         }
16     }
17     tout[x] = t++;
18 }
19 bool is_ancestor(int x, int y){
20     return tin[x] <= tin[y] && tout[x] >= tout[y];
21 }

```

```

22 void table(){
23     // 上兩輩祖先、上四輩祖先、上八輩祖先、...
24     for (int i=1; i<logn; i++)
25         for (int x=0; x<n; ++x)
26             ancestor[x][i] = ancestor[ancestor[x][i-1]][i-1];
27 }
28
29 int kth_ancestor(int x, int k){
30     // k 拆解成二進位位數，找到第k祖先。不斷上升逼近之。
31     for (int i=0; i<logn; i++)
32         if (k & (1<<i))
33             x = ancestor[x][i];
34     return x;
35 }
36
37 void rooted_tree(int root){
38     ancestor[root][0] = root;
39     dfs(root);
40     table();
41 }
42
43 int LCA(int x,int y){
44     if (is_ancestor(x, y)) return x;
45     if (is_ancestor(y, x)) return y;
46     for (int i=logn-1; i>=0; i--)
47         if (!is_ancestor(ancestor[x][i], y))
48             x = ancestor[x][i];
49     return ancestor[x][0];
50 }
51
52 int main(){
53     graph = {
54         {1,2},
55         {3},
56         {5,6},
57         {7},
58         {},
59         {},
60         {8},
61         {4},
62     };
63     n = 9;
64     logn = ceil(log2(n));
65     ancestor.resize(n,vector<int>(logn));
66     tin.resize(n);
67     tout.resize(n);
68
69     rooted_tree(0);
70     while(true){
71         int a,b;
72         cin >> a >> b;
73         cout << LCA(a,b) << endl;;
74     }

```

### 13.2 diameter

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 vector<vector<int>> graph;
6 int diameter = 0;
7 int dfs(int start, int parent){

```

```

8     int h1 = 0,h2 = 0;
9     for(auto child: graph[start]){
10         if(child!= parent){
11             int h = dfs(child,start)+1;
12             if(h>h1){
13                 h2= h1;
14                 h1 = h;
15             }
16             else if(h>h2){
17                 h2 = h;
18             }
19         }
20     }
21     diameter = max(diameter,h1+h2);
22     return h1;
23 }
24
25 int main(){
26     graph = {
27         {1,3},
28         {0},
29         {3},
30         {0,2,4},
31         {3}
32     };
33     dfs(0,-1);
34     cout << diameter<<endl;
35 }

```

### 13.3 radius

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 // Perform DFS to find the farthest node and its distance
4 // from the given node
5 pair<int, int> dfs(int node, int distance, vector<bool>&
6     visited, const vector<vector<int>>& adj_list) {
7     visited[node] = true;
8     int max_distance = distance;
9     int farthest_node = node;
10
11     for (int neighbor : adj_list[node]) {
12         if (!visited[neighbor]) {
13             auto result = dfs(neighbor, distance + 1, visited,
14                 adj_list);
15             if (result.first > max_distance) {
16                 max_distance = result.first;
17                 farthest_node = result.second;
18             }
19         }
20     }
21
22     return make_pair(max_distance, farthest_node);
23 }
24
25 // Calculate the radius of the tree using DFS
26 int tree_radius(const vector<vector<int>>& adj_list) {
27     int num_nodes = adj_list.size();
28     vector<bool> visited(num_nodes, false);
29
30     // Find the farthest node from the root (node 0)
31     auto farthest_result = dfs(0, 0, visited, adj_list);

```

```

30 // Reset visited array
31 fill(visited.begin(), visited.end(), false);
32
33 // Calculate the distance from the farthest node
34 int radius = dfs(farthest_result.second, 0, visited,
35                 adj_list).first;
36
37 return radius;
38 }
39
40 int main() {
41     vector<vector<int>> adj_list = {
42         {1, 2},
43         {0, 3, 4},
44         {0, 5},
45         {1},
46         {1},
47         {2}
48     };
49
50 int radius = tree_radius(adj_list);
51 cout << "Tree radius: " << radius << endl;
52
53 return 0;
54 }

```

### 13.4 bridge

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int n = 9;
4 vector<vector<int>> graph;
5 vector<int> visit(n,0);
6 vector<int> trace(n,0);
7 vector<vector<int>> bridge;
8 int t = 0;
9 void dfs(int x,int parent){
10     visit[x] = ++t;
11     trace[x] = x; // 最高祖先預設為自己
12     for(auto to:graph[x]){
13         if(visit[to]){ //back edge
14             if(to!=parent){
15                 trace[x] = to;
16             }
17         }else{ //tree edge
18             dfs(to,x);
19             if (visit[trace[to]] < visit[trace[x]])
20                 trace[x] = trace[to];
21
22             // 子樹回不到祖先暨自身。
23             if (visit[trace[to]] > visit[x])
24                 bridge.push_back({x,to});
25         }
26     }
27 }
28 int main(){
29     graph = {
30         {1,2},
31         {3},
32         {5,6},
33         {7},
34         {},
35         {},

```

```

36     {},
37     {8},
38     {4},
39 };
40 for(int i =0;i<9;i++){
41     if(!visit[i])
42         dfs(i,-1);
43 }
44 for(auto x: bridge){
45     cout << x[0]<<" "<< x[1]<<endl;
46 }
47 }

```

### 13.5 Articulation\_vertex

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int n = 9;
4 int t =0;
5 vector<int> disc(n,-1); // Discovery time
6 vector<int> low(n,-1); // Low time
7 vector<int> parent_array(n,-1); // Parent in DFS tree
8 vector<bool> visited(n,false);
9 vector<bool> is_articulation(n,false);
10 vector<vector<int>> graph;
11 void dfs_articulation(int node, int parent) {
12     visited[node] = true;
13     disc[node] = t;
14     low[node] = t;
15     t++;
16     int children = 0;
17
18     for (int neighbor : graph[node]) {
19         if (!visited[neighbor]) {
20             children++;
21             parent_array[neighbor] = node;
22             dfs_articulation(neighbor, node);
23             low[node] = min(low[node], low[neighbor]);
24
25             if (low[neighbor] >= disc[node] && parent != -1)
26                 is_articulation[node] = true;
27         }
28         else if (neighbor != parent) {
29             low[node] = min(low[node], disc[neighbor]);
30         }
31     }
32
33     if (parent == -1 && children > 1) {
34         is_articulation[node] = true;
35     }
36 }
37 int main(){
38     graph = {
39         {1,2},
40         {3},
41         {5,6},
42         {7},
43         {},
44         {},
45         {},
46         {8},
47         {4},

```

```

48     };
49     for (int i = 0; i < n; ++i) {
50         if (!visited[i]) {
51             dfs_articulation(i, -1);
52         }
53     }
54     cout << "Articulation Points: ";
55     for (int i = 0; i < n; ++i) {
56         if (is_articulation[i]) {
57             cout << i << " ";
58         }
59     }
60     cout << endl;
61 }

```



# NYCU\_SEGMENTTREE CODEBOOK

## Contents

<b>1 Data Structure</b>	<b>1</b>	<b>4.4 2 SAT</b>	<b>4</b>	<b>8.3 Radius</b>	<b>9</b>
1.1 DSU	1	4.5 Kosaraju 2DFS	4	<b>9 Z_Original_Code/Data_Structure</b>	<b>9</b>
1.2 Monotonic Queue	1	4.6 Dijkstra	5	9.1 dsu-class	9
1.3 BIT	1	4.7 Floyd Warshall	5	9.2 monotonic-queue	9
1.4 Segment Tree	1	4.8 Articulation Vertex	5	9.3 BIT	10
1.5 Sparse Table	2	4.9 Topological Sort	5	9.4 segment-tree-simple-add	10
1.6 Monotonic Stack	2	4.10 Planar	6	9.5 monotonic-stack	10
<b>2 Flow</b>	<b>2</b>	4.11 Heavy Light Decomposition	6	<b>10 Z_Original_Code/Flow</b>	<b>11</b>
2.1 Dinic	2	4.12 Centroid Decomposition	6	10.1 dicnic	11
<b>3 Geometry</b>	<b>2</b>	<b>5 Math</b>	<b>6</b>	<b>11 Z_Original_Code/Graph</b>	<b>11</b>
3.1 Sort by Angle	2	5.1 fpow	6	11.1 planar	11
3.2 Convex Hull	2	5.2 extgcd	6	11.2 Dijkstra	11
3.3 Point in Polygon	3	5.3 EulerTotientFunction	7	11.3 Floyd_Warshall	12
3.4 MinCoveringCircle	3	5.4 FFT	7	11.4 2_sat	12
<b>4 Graph</b>	<b>3</b>	<b>6 Misc</b>	<b>7</b>	11.5 bipartite_matching	13
4.1 Bipartite Matching	3	6.1 pbds	7	11.6 tarjan-SCC	13
4.2 Tarjan SCC	4	6.2 Misc	7	11.7 topological_sort	13
4.3 Bridge	4	6.3 Mo'sAlgorithm	7	<b>12 Z_Original_Code/Math</b>	<b>14</b>
		<b>7 String</b>	<b>7</b>	12.1 extgcd	14
		7.1 Hashing	7	<b>13 Z_Original_Code/Tree</b>	<b>14</b>
		7.2 Trie	8	13.1 LCA	14
		7.3 Zvalue	8	13.2 diameter	14
		7.4 KMP	8	13.3 radius	14
		7.5 Manacher	8	13.4 bridge	15
		<b>8 Tree</b>	<b>8</b>	13.5 Articulation_vertex	15
		8.1 LCA	8		
		8.2 Diameter	9		