

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        // 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        tag[2*v+1]+=tag[v];
26        tag[2*v+2]+=tag[v];
27        tag[v]=0;
28    }
29    void build(int l,int r,int v=0){
30        if(r-l==1){
31            summ[v]=arr[l];
32            // summ[v]=minn[v]=maxx[v]=arr[l];
33            return;
34        }
35        int mid=(l+r)/2;
36        build(l,mid,2*v+1);
37        build(mid,r,2*v+2);
38        pull(l,r,v);
39    }
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 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        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 }

```

```

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 Gaph

3.1 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
46    }
47 }

```

```

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<n;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){
60         cout << t[0]<<" "<<t[1]<<endl;
61     }
62 }
63 }

```

3.2 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 圖最早拜訪的
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 }

```

3.3 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 }

```

3.4 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{ // treeedge
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<n;i++){
29         if(!visit[i])
30             dfs(i,-1);
31     }
32     for(auto x: bridge){
33         cout << x[0]<<" "<< x[1]<<endl;
34     }
35 }

```

3.5 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]){
19             int to = next.second;
20             int weight = next.first;
21             // cout <<from<<' ' <<to<<' ' <<weight;
22             if(dis[from]+weight< dis[to] || dis[to]==-1){
23                 dis[to] = dis[from]+weight;
24                 parent[to] = from;
25                 pq.push({dis[from]+weight,to});
26             }
27         }

```

```

28 }
29 int main(){
30     int startpoint;
31     dijkstra(startpoint);
32     //dis and parent
33 }

```

3.6 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(processingOrder.begin(), processingOrder.end()); // topological sort
18        for (int i = 0; i < 2 * n; ++i) {
19            visited[i] = false;
20        }
21        for (int node : processingOrder) {
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 private:
34     int n;
35     vector<vector<int>>> graph;
36     vector<bool> visited;
37     vector<int> processingOrder;
38     vector<int> scc;
39
40     void dfs1(int node) {
41         visited[node] = true;
42         for (int neighbor : graph[node]) {
43             if (!visited[neighbor]) {
44                 dfs1(neighbor);
45             }
46         }
47         processingOrder.push_back(node);
48     }
49
50     void dfs2(int node) {
51         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.end()) {
64                 return false; // Contradiction found in the same SCC
65             }
66         }
67         return true;
68     }
69 };
70 int main() {
71     int n, m; // Number of variables and clauses
72     TwoSAT twoSat(n);
73     for (int i = 0; i < m; ++i) {
74         int a, b;
75         twoSat.addClause(a, b);
76     }
77     if (twoSat.solve()) {
78         cout << "Satisfiable" << endl;
79     } else {
80         cout << "Unsatisfiable" << endl;
81     }
82 }

```

3.7 Floyd Warshall

```

1 #define maxn 2005
2 vector<vector<int>>> dis(maxn, vector<int>(maxn, 9999999));
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] == -1) {
18                        dis[i][j] = dis[i][k] + dis[k][j];
19                        mid[i][j] = k; // 由 i 點走到 j 點經過了 k 點
20                    }
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; // 沒有中繼點就結束
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         cout << endl;
39     }
40     find_path(0, 2);

```

3.8 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         if (!visited[neighbor]) {
18             children++;
19             parent_array[neighbor] = node;
20             dfs_articulation(neighbor, node);
21             low[node] = min(low[node], low[neighbor]);
22
23             if (low[neighbor] >= disc[node] && parent != -1) {
24                 is_articulation[node] = true;
25             }
26         } else if (neighbor != parent) {
27             low[node] = min(low[node], disc[neighbor]);
28         }
29     }
30     if (parent == -1 && children > 1) {
31         is_articulation[node] = true;
32     }
33 }
34
35 // call for() dfs(i, -1)
36 int main() {
37     for (int i = 0; i < n; ++i) {
38         if (!visited[i]) {
39             dfs_articulation(i, -1);
40         }
41     }
42     cout << "Articulation Points: ";

```

```

48     for (int i = 0; i < n; ++i) {
49         if (is_articulation[i]) {
50             cout << i << " ";
51         }
52     }cout << endl;
53 }

```

3.9 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 }

```

3.10 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>&
13     subgraph) {
14     unordered_set<int> subgraphVertices(subgraph.begin(),
15         subgraph.end());
16     for (int vertex : subgraphVertices) {
17         for (int neighbor : graph.adj[vertex]) {
18             if (subgraphVertices.count(neighbor) == 0) {
19                 bool found = true;
20                 for (int v : subgraph) {
21                     if (v != vertex && v != neighbor) {

```

```

20         if (graph.adj[v].size() < 3) {
21             found = false;
22             break;
23         }
24     }
25     if (found)
26         return true;
27 }
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,
43         k33a) || containsSubgraph(graph, k33b)) {
44         return false; // The graph is non-planar
45     }
46     return true; // The graph is planar
47 }
48
49 int main() {
50     int vertices, edges;
51     Graph graph(vertices);
52     for (int i = 0; i < edges; ++i) {
53         int u, v; cin >> u >> v;
54         graph.addEdge(u, v);
55     }
56     if (isPlanar(graph)) {
57         cout << "The graph is planar." << endl;
58     } else {
59         cout << "The graph is non-planar." << endl;
60     }
61 }

```

4 Math

4.1 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 Original_Code/Data_Structure

5.1 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) ? 41
67         "Yes" : "No") << endl;
68     cout << "Are 2 and 3 connected? " << (dsu.aresame(2, 3) ? 43
69         "Yes" : "No") << endl;
70     cout << "Are 4 and 5 connected? " << (dsu.aresame(4, 5) ? 45
71         "Yes" : "No") << endl;
72     cout << "Are 6 and 7 connected? " << (dsu.aresame(6, 7) ? 47
73         "Yes" : "No") << endl;
74     cout << "Are 1 and 2 connected? " << (dsu.aresame(1, 2) ? 49
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 }

```

5.2 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 int main(){
48     vector<int> nums = {1,3,-1,-3,5,3,6,7};
49     int k = 3;
50     vector<int> res = maxSlidingWindow(nums,k);
51     for (auto r:res)cout <<r <<" ";
52 }
53 }

```

5.3 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();

```

```

32 BIT bit(n);
33 for(int i = 1;i<n;i++){
34     bit.update(i,freq[i]);
35 }
36 for(int i = 1;i<n;i++){
37     cout << bit.query(i)<<" ";
38 }cout << endl;
39 for(int i = 1;i<n;i++){
40     bit.update(i,-1);
41 }
42 for(int i = 1;i<n;i++){
43     cout << bit.query(i)<<" ";
44 }cout << endl;
45 }

```

5.4 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+
22         segment_tree[x<<1|1].value;
23 }
24 void modify(int position ,int x,int value){
25     if(segment_tree[x].left == position && segment_tree[x].
26         right ==position){ // here is leaf
27         segment_tree[x].value = value;
28         return;
29     }
30     int mid = (segment_tree[x].left+segment_tree[x].right)/2;
31     if(position<=mid){
32         modify(position ,x<<1,value);
33     }else{
34         modify(position ,x<<1|1,value);
35     }
36     segment_tree[x].value = segment_tree[x<<1].value+
37         segment_tree[x<<1|1].value;
38 }
39 int query(int i,int j,int x){
40     // cout <<i <<" "<<j <<" "<<segment_tree[x].left <<" " <<
41     // segment_tree[x].right<<endl;
42     int res = 0;
43     int left = segment_tree[x].left;
44     int right = segment_tree[x].right;
45     int mid = (left+right)/2;

```

```

43 if(segment_tree[x].left==i && segment_tree[x].right ==j){
44
45     return segment_tree[x].value;
46 }
47 if(i>mid)return query(i,j,x*2+1);
48 if(mid>=j)return query(i,j,x*2);
49 return query(i,mid,x*2)+ query(mid+1,j,x*2+1);
50 }
51 int main(){
52     vector<int> nums =
53         {1,10,5,148,78,2,56,231,5,64,65,32,1,8};
54     int n = nums.size();
55     segment_tree.resize(n*4);
56     build(0,n-1,1,nums);
57     modify(5,1,100);
58     // cout << "+++++++\n";
59     for(int i =0;i<n;i++){
60         for(int j = i ;j<n;j++){
61             cout << query(i,j,1)<<" ";
62         }
63     }

```

5.5 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
21             // pop the value smaller and equal nums[i]
22         }
23         if(st.empty())res[i] = -1;
24         else res[i] = st.top();
25         st.push(nums[i]);
26     }
27     return res;
28 }
29
30 int main(){
31     vector<int> res = monotonic_stack({2,1,2,4,3});
32     for(auto r:res){
33         cout << r<<" ";
34     }

```

6 Original_Code/Flow

6.1 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),
9              flow(f), rev(r) {}
10     };
11     vector<edge> G[maxn];
12     int s, t, dis[maxn], cur[maxn], vis[maxn];
13     void add_edge(int from, int to, int cap){
14         G[from].push_back(edge(to, cap, 0, G[to].size()));
15         G[to].push_back(edge(from, 0, 0, G[from].size()-1));
16     }
17     bool bfs(){
18         memset(dis, -1, sizeof(dis));
19         queue<int> qu;
20         qu.push(s);
21         dis[s] = 0;
22         while (!qu.empty()) {
23             int from = qu.front();
24             qu.pop();
25             for (auto &e: G[from]) {
26                 if (dis[e.to]==-1 && e.cap != e.flow) {
27                     dis[e.to] = dis[from] + 1;
28                     qu.push(e.to);
29                 }
30             }
31         }
32         return dis[t]!=-1;
33     }
34     int dfs(int from, int cap){
35         if(from==t || cap==0)return cap;
36         for(int &i = cur[from]; i<G[from].size(); i++){
37             edge &e = G[from][i];
38             if(dis[e.to]==dis[from]+1 && e.flow!=e.cap){
39                 int df = dfs(e.to, min(e.cap-e.flow, cap));
40                 if(df){
41                     e.flow+=df;
42                     G[e.to][e.rev].flow-=df;
43                     return df;
44                 }
45             }
46         }
47         dis[from] = -1;
48         return 0;
49     }
50     int Maxflow(int s, int t){
51         this->s = s, this->t = t;
52         int flow = 0;
53         int df;
54         while(bfs()){
55             memset(cur, 0, sizeof(cur));
56             while(df = dfs(s, INF)){
57                 flow +=df;
58             }

```

```

59         return flow;
60     }
61 };

```

7 Original_Code/Graph

7.1 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>&
19     subgraph) {
20     unordered_set<int> subgraphVertices(subgraph.begin(),
21         subgraph.end());
22     for (int vertex : subgraphVertices) {
23         for (int neighbor : graph.adj[vertex]) {
24             if (subgraphVertices.count(neighbor) == 0) {
25                 bool found = true;
26                 for (int v : subgraph) {
27                     if (v != vertex && v != neighbor) {
28                         if (graph.adj[v].size() < 3) {
29                             found = false;
30                             break;
31                         }
32                     }
33                 }
34                 if (found)
35                     return true;
36             }
37         }
38     }
39     return false;
40 }
41
42 bool isPlanar(const Graph& graph) {
43     // Subgraphs isomorphic to K4 and K5,
44     vector<int> k5 = {0, 1, 2, 3, 4}; // Vertices of K5
45     vector<int> k33a = {0, 1, 2}; // Vertices of K3,3 (part A)
46     vector<int> k33b = {3, 4, 5}; // Vertices of K3,3 (part B)
47
48     if (containsSubgraph(graph, k5) || containsSubgraph(graph,
49         k33a) || containsSubgraph(graph, k33b)) {
50         return false; // The graph is non-planar
51     }

```



```

49     return true; // The graph is planar
50 }
51
52 int main() {
53     int vertices, edges;
54     cin >> vertices;
55     cin >> edges;
56
57     Graph graph(vertices);
58     for (int i = 0; i < edges; ++i) {
59         int u, v;
60         cin >> u >> v;
61         graph.addEdge(u, v);
62     }
63     if (isPlanar(graph)) {
64         cout << "The graph is planar." << endl;
65     } else {
66         cout << "The graph is non-planar." << endl;
67     }
68
69     return 0;
70 }

```

7.2 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>>,
13         greater<pair<int, int>>> pq;
14     pq.push({0, source});
15     while(!pq.empty()){
16         int from = pq.top().second;
17         pq.pop();
18         // cout << vis[from] << endl;
19         if (vis[from]) continue;
20         vis[from] = true;
21         for (auto next : graph[from]){
22             int to = next.second;
23             int weight = next.first;
24             // cout << from << " " << to << " " << weight;
25             if (dis[from] + weight < dis[to] || dis[to] == -1){
26                 dis[to] = dis[from] + weight;
27                 parent[to] = from;
28                 pq.push({dis[from] + weight, to});
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 }

```

7.3 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; // 因有中繼點就結束
31     find_path(s, mid[s][t]); // 前半段最短路徑
32     cout << mid[s][t]; // 中繼點
33     find_path(mid[s][t], t); // 後半段最短路徑
34 }
35 int main(){
36     graph = {
37         {{4,1},{5,3}},
38         {{3,3}},
39         {{}},
40         {{4,0},{2,1},{7,2}}
41     };
42     floyd_warshall(4);
43     for (int i = 0; i < 4; ++i){
44         for (int j = 0; j < 4; ++j){
45             cout << dis[i][j] << " ";
46         }
47         cout << endl;
48     }
49     find_path(0, 2);
50 }

```

7.4 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
13     void addClause(int a, int b) { // 0-base;
14         a *= 2;
15         b *= 2;
16         // Add implications (~a => b) and (~b => a)
17         graph[a ^ 1].push_back(b);
18         graph[b ^ 1].push_back(a);
19     }
20
21     bool solve() {
22         // Find SCCs and check for contradictions
23         for (int i = 0; i < 2 * n; ++i) {
24             if (!visited[i]) {
25                 dfs1(i);
26             }
27         }
28         reverse(processOrder.begin(), processOrder.end());
29         // topological sort
30         for (int i = 0; i < 2 * n; ++i) {
31             visited[i] = false;
32         }
33         for (int node : processOrder) {
34             if (!visited[node]) {
35                 scc.clear();
36                 dfs2(node);
37                 if (!checkSCCConsistency()) {
38                     return false;
39                 }
40             }
41         }
42         return true;
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;
}

```

7.5 bipartite_matching

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int MAXN = 100;
4
5
6 struct Bipartite_matching{
7     int mx[MAXN], my[MAXN], vy[MAXN]; //matchX, matchY,
8     visitY
9     vector<int> edge[MAXN]; //adjacent list;
10    int x_cnt;
11    bool dfs(int x){
12        for(auto y: edge[x]){ //對 x 可以碰到的邊進行檢查
13            if(vy[y] == 1) continue; //避免遞迴 error
14
15            vy[y] = 1;
16            if(my[y] == -1 || dfs(my[y])){ //分析 3
17                mx[x] = y;
18                my[y] = x;
19            }
20        }
21    }
22
23    int ans = 0;
24    for(int i = 0; i < x_cnt; i++){ //對每一個 x 節點進
25        //行 DFS(最大匹配)
26        if(dfs(i)) ans++;
27    }
28    return ans;
29 }
30
31 vector<vector<int>> get_match(){
32     vector<vector<int>> res;
33     for(int i = 0; i < x_cnt; i++){
34         if(mx[i] != -1){
35             res.push_back({i, mx[i]});
36         }
37     }
38     return res;
39 }
40
41 void add_edge(int i, int j){
42     edge[i].push_back(j);
43 }
44
45 void init(int x){
46     x_cnt = x;
47 }
48
49 };
50
51 int 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++){
63         int a, b;
64         cin >> a >> b;
65         bm.add_edge(a, b);
66     }
67     bm.init(3);
68     cout << bm.bipartite_matching() << endl;
69     auto match = bm.get_match();
70     for(auto t: match){
71         cout << t[0] << " " << t[1] << endl;
72     }
73 }

```

7.6 tarjan-SCC

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int n = 16;

```

```

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         if(dfs(i)) ans++;
31     }
32     return ans;
33 }
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
45 void add_edge(int i, int j){
46     edge[i].push_back(j);
47 }
48
49 void init(int x){
50     x_cnt = x;
51 }
52
53 };
54
55 int main(){
56     /*
57     0 3
58     0 4
59     1 3
60     1 5
61     2 3
62     2 4
63     2 5
64     */
65     Bipartite_matching bm;
66     for(int i = 0; i < 7; i++){
67         int a, b;
68         cin >> a >> b;
69         bm.add_edge(a, b);
70     }
71     bm.init(3);
72     cout << bm.bipartite_matching() << endl;
73     auto match = bm.get_match();
74     for(auto t: match){
75         cout << t[0] << " " << t[1] << endl;
76     }
77 }
78
79 vector<vector<int>> graph;
80 int visit[n], low[n], t = 0;
81 int st[n], top = 0;
82 bool instack[n];
83 int contract[n]; // 每個點收縮到的點
84 vector<vector<int>> block;
85 void dfs(int x, int parent){
86     // cout << x << endl;
87     visit[x] = low[x] = ++t;
88     st[top++] = x;
89     instack[x] = true;
90     for(auto to: graph[x]){
91         if(!visit[to]){
92             dfs(to, x);
93
94             if(instack[to])
95                 low[x] = min(low[x], low[to]);
96         }
97         if(visit[x] == low[x]){ //scc ㊦ 最早拜訪的
98             int j;
99             block.push_back({});
100             do{
101                 j = st[--top];
102                 instack[j] = false;
103                 block[block.size() - 1].push_back(j);
104                 contract[j] = x;
105             }while(j != x);
106         }
107     }
108 }
109
110 int main(){
111     graph = {
112         {1},
113         {3,4,5},
114         {6},
115         {2},
116         {7},
117         {11,15},
118         {2,3},
119         {4,6,9},
120         {},
121         {},
122         {},
123         {15},
124         {14},
125         {13,5},
126         {15},
127         {10,12,13}
128     };
129     for(int i = 0; i < n; i++){
130         if (!visit[i])
131             dfs(i, i);
132     }
133     for(auto t: block){
134         for(auto x: t){
135             cout << x << " ";
136         }
137         cout << endl;
138     }
139 }

```

7.7 topological_sort

```

1 #include <bits/stdc++.h>
2

```

```

3 using namespace std;
4 vector<vector<int>> graph;
5 vector<int> visit(10,0);
6 vector<int> order;
7 int n;
8 bool cycle; // 記DFS的過程中是否偵測到環
9 void DFS(int i)
10 {
11     if (visit[i] == 1) {cycle = true; return;}
12     if (visit[i] == 2) return;
13     visit[i] = 1;
14     for(auto to : graph[i])
15         DFS(to);
16     visit[i] = 2;
17     order.push_back(i);
18 }
19
20 int main() {
21     graph = {
22         {1, 2},
23         {3},
24         {3, 4},
25         {4},
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 }

```

8 Original_Code/Math

8.1 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;

```

```

18     return r; //得到a b的最大公因數
19 }
20
21 int main(){
22     int a = 55,b = 80;
23     int x,y;
24     int GCD = extgcd(a,b,x,y);
25     cout << "GCD: "<<GCD<<endl;
26     cout <<x<<" "<<y<<endl;
27     cout <<a*x+b*y<<endl;
28 }

```

9 Original_Code/Tree

9.1 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 int LCA(int x,int y){
43     if (is_ancestor(x, y)) return x;
44     if (is_ancestor(y, x)) return y;
45     for (int i=logn-1; i>=0; i--)

```

```

46     if (!is_ancestor(ancestor[x][i], y))
47         x = ancestor[x][i];
48     return ancestor[x][0];
49 }
50 int main(){
51     graph = {
52         {1,2},
53         {3},
54         {5,6},
55         {7},
56         {},
57         {},
58         {},
59         {8},
60         {4},
61     };
62     n = 9;
63     logn = ceil(log2(n));
64     ancestor.resize(n,vector<int>(logn));
65     tin.resize(n);
66     tout.resize(n);
67
68     rooted_tree(0);
69     while(true){
70         int a,b;
71         cin >>a>>b;
72         cout <<LCA(a,b)<<endl;
73     }
74 }

```

9.2 diameter

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 vector<vector<int>> graph;
5 int diameter = 0;
6 int dfs(int start, int parent){
7     int h1 = 0,h2 = 0;
8     for(auto child: graph[start]){
9         if(child!= parent){
10             int h = dfs(child,start)+1;
11             if(h>h1){
12                 h2= h1;
13                 h1 = h;
14             }
15             else if(h>h2){
16                 h2 = h;
17             }
18         }
19     }
20     diameter = max(diameter,h1+h2);
21     return h1;
22 }
23
24 int main(){
25     graph = {
26         {1,3},
27         {0},
28         {3},
29         {0,2,4},
30         {3}
31     }

```

```

32     };
33     dfs(0,-1);
34     cout << diameter<<endl;
35 }

```

9.3 radius

```

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

```

```

53 }

```

9.4 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{ //treeedge
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
29 int main(){
30     graph = {
31         {1,2},
32         {3},
33         {5,6},
34         {7},
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 }

```

9.5 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         } else if (neighbor != parent) {
28             low[node] = min(low[node], disc[neighbor]);
29         }
30     }
31
32     if (parent == -1 && children > 1) {
33         is_articulation[node] = true;
34     }
35 }
36
37 int main(){
38     graph = {
39         {1,2},
40         {3},
41         {5,6},
42         {7},
43         {},
44         {},
45         {8},
46         {4},
47     };
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 }

```

10 Tree

10.1 LCA

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int n, logn, t=0;
4 vector<vector<int>> graph;
5 vector<vector<int>> ancestor;
6 vector<int> tin, tout;
7 void dfs(int x){
8     tin[x] = t++;
9     for(auto y: graph[x]){
10         if(y != ancestor[x][0]){
11             ancestor[y][0] = x;
12             dfs(y);
13         }
14     }
15     tout[x] = t++;
16 }
17 bool is_ancestor(int x, int y){
18     return tin[x] <= tin[y] && tout[x] >= tout[y];
19 }
20 void table(){
21     for (int i=1; i<logn; i++)// 上兩輩祖先、上四輩祖先、上八輩
        祖先、.....
22     for (int x=0; x<n; ++x)
23         ancestor[x][i] = ancestor[ancestor[x][i-1]][i-1];
24 }
25
26 int kth_ancestor(int x, int k){
27     for (int i=0; i<logn; i++)// k 拆解成二進位位數，找到第k祖
        先。不斷上升逼近之。
28         if (k & (1<<i))
29             x = ancestor[x][i];
30     return x;
31 }
32
33 void rooted_tree(int root){// build the tree with root at "
    root"
34     ancestor[root][0] = root;
35     dfs(root);
36     table();
37 }
38 int LCA(int x, int y){
39     if (is_ancestor(x, y)) return x;
40     if (is_ancestor(y, x)) return y;
41     for (int i=logn-1; i>=0; i--)
42         if (!is_ancestor(ancestor[x][i], y))
43             x = ancestor[x][i];
44     return ancestor[x][0];
45 }
46 int main(){
47     graph = {
48         {1,2},
49         {3},
50         {5,6},
51         {7},
52         {},
53         {},
54         {},
55         {8},
56         {4},

```

```

57     };
58     n = 9;
59     logn = ceil(log2(n));
60     ancestor.resize(n, vector<int>(logn));
61     tin.resize(n);
62     tout.resize(n);
63
64     rooted_tree(0);
65     while(true){
66         int a, b;
67         cin >> a >> b;
68         cout << LCA(a, b) << endl;
69     }
70 }
71 int main(){
72     n = 9;
73     logn = ceil(log2(n));
74     ancestor.resize(n, vector<int>(logn));
75     tin.resize(n);
76     tout.resize(n);
77     rooted_tree(0);
78     while(true){
79         int a, b;
80         cin >> a >> b;
81         cout << LCA(a, b) << endl;
82     }
83 }

```

10.2 Diameter

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 vector<vector<int>> graph;
5 int diameter = 0;
6 int dfs(int start, int parent){
7     int h1 = 0, h2 = 0;
8     for (auto child : graph[start]){
9         if (child != parent){
10             int h = dfs(child, start) + 1;
11             if (h > h1){
12                 h2 = h1;
13                 h1 = h;
14             }
15             else if (h > h2){
16                 h2 = h;
17             }
18         }
19     }
20     diameter = max(diameter, h1 + h2);
21     return h1;
22 }
23 // call diameter
24 int main(){
25     dfs(0, -1);
26     cout << diameter << endl;
27 }

```

10.3 Radius

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 // Perform DFS to find the farthest node and its distance
    from the given node
4 pair<int, int> dfs(int node, int distance, vector<bool> &
    visited, const vector<vector<int>> & adj_list){
5     visited[node] = true;
6     int max_distance = distance;
7     int farthest_node = node;
8
9     for (int neighbor : adj_list[node]){
10         if (!visited[neighbor]){
11             auto result = dfs(neighbor, distance + 1, visited
                , adj_list);
12             if (result.first > max_distance){
13                 max_distance = result.first;
14                 farthest_node = result.second;
15             }
16         }
17     }
18
19     return make_pair(max_distance, farthest_node);
20 }
21
22 // Calculate the radius of the tree using DFS
23 int tree_radius(const vector<vector<int>> & adj_list){
24     int num_nodes = adj_list.size();
25     vector<bool> visited(num_nodes, false);
26
27     // Find the farthest node from the root (node 0)
28     auto farthest_result = dfs(0, 0, visited, adj_list);
29
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,
        adj_list).first;
35
36     return radius;
37 }
38 int main() {
39     vector<vector<int>> adj_list;
40     int radius = tree_radius(adj_list);
41     cout << "Tree radius: " << radius << endl;
42     return 0;
43 }

```

NYCU_SEGMENTTREE CODEBOOK

Contents

1 Data Structure	1	3.3 Kosaraju 2dfs	3	7 Original_Code/Graph	7
1.1 DSU	1	3.4 Bridge	3	7.1 planar	7
1.2 Monotonic Queue	1	3.5 Dijkstra	3	7.2 Dijkstra	8
1.3 BIT	1	3.6 2_sat	4	7.3 Floyd_Warshall	8
1.4 Segment Tree	1	3.7 Floyd Warshall	4	7.4 2_sat	8
1.5 Monotonic Stack	2	3.8 Articulation Vertex	4	7.5 bipartite_matching	9
2 Flow	2	3.9 Topological Sort	5	7.6 tarjan-SCC	9
2.1 Dinic	2	3.10 Planar	5	7.7 topological_sort	9
3 Gaph	2	4 Math	5	8 Original_Code/Math	10
3.1 Bipartite Matching	2	4.1 extgcd	5	8.1 extgcd	10
3.2 Tarjan SCC	3	5 Original_Code/Data_Structure	5	9 Original_Code/Tree	10
		5.1 dsu-class	5	9.1 LCA	10
		5.2 monotonic-queue	6	9.2 diameter	10
		5.3 BIT	6	9.3 radius	11
		5.4 segment-tree-simple-add	6	9.4 bridge	11
		5.5 monotonic-stack	7	9.5 Articulation_vertex	11
		6 Original_Code/Flow	7	10 Tree	12
		6.1 dicnic	7	10.1 LCA	12
				10.2 Diameter	12
				10.3 Radius	12