

1 Data Structure

1.1 dsu class

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 class DSU{
5 public:
6     DSU(int n){
7         this->n = n;
8         reset();
9     }
10    int n;
11    vector<int> boss;
12    vector<int> rank;
13    vector<int> size;
14
15    void reset(){
16        this->boss.resize(n);
17        this->rank.resize(n,0);
18        this->size.resize(n,0);
19        for(int i =0;i<n;i++){
20            boss[i] = i;
21        }
22    }
23    int find(int x){
24        if(boss[x]!= x){
25            boss[x] = find(boss[x]);
26        }
27        return boss[x];
28    }
29    int get_size(int x){
30        return size[find(x)];
31    }
32    void merge(int x, int y){
33        int a = find(x);
34        int b = find(y);
35        if(a==b){
36            if(rank[a]<rank[b]){
37                boss[a] = b;
38                size[b] += size[a];
39            }else if (rank[a]<rank[b]){
40                boss[b] = a;
41                size[a] += size[b];
42            }else{
43                boss[a] = b;
44                size[b] += size[a];
45                rank[b]++;
46            }
47        }
48    }
49    bool aresame(int a,int b){
50        return find(a)==find(b);
51    }
52 };

```

1.2 monotonic queue

```

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

```

```

2 using namespace std;
3
4 class Monotonic_queue{
5 private:
6     deque<int> qu;
7 public:
8     void push(int n){
9         while(!qu.empty()&&qu.back()<n){
10             qu.pop_back();
11         }
12         qu.push_back(n);
13     }
14     int max(){
15         return qu.front();
16     }
17     int min(){
18         return qu.back();
19     }
20     int size(){
21         return qu.size();
22     }
23     void pop(){
24         qu.pop_front();
25     }
26 };

```

1.3 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 };

```

1.4 segment tree simple add

```

1 #include <bits/stdc++.h>
2 using namespace std;
3

```

```

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     int res = 0;
41     int left = segment_tree[x].left;
42     int right = segment_tree[x].right;
43     int mid = (left+right)/2;
44     if(segment_tree[x].left==i && segment_tree[x].right ==j){
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 }

```

1.5 monotonic stack

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 vector<int> monotonic_stack(vector<int> nums){
5     int n = nums.size();
6     vector<int> res(n);
7     stack<int> st;
8     for(int i = n-1;i>=0;i--){
9         while(!st.empty() && st.top()<=nums[i]){
10             st.pop();
11         }
12         if(st.empty())res[i] = -1;

```

```

13     else res[i] = st.top();
14     st.push(nums[i]);
15 }
16 return res;
17 }

```

2 Flow

2.1 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;

```

```

51    int flow = 0;
52    int df;
53    while (bfs()){
54        memset(cur, 0, sizeof(cur));
55        while (df = dfs(s, INF)) {
56            flow += df;
57        }
58    }
59    return flow;
60 }
61 };
62 int main(){
63     int n = 4, m = 6;
64     MaxFlow maxflow;
65     for (int i = 0; i < m; i++){
66         int a, b, cap;
67         cin >> a >> b >> cap;
68         maxflow.add_edge(a, b, cap);
69     }
70     cout << maxflow.Maxflow(1, 3) << endl;
71 }

```

3 Gaph

3.1 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 }

```

```

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     , (part A)
45     vector<int> k33b = {3, 4, 5}; // Vertices of K
46     , (part B)
47
48     if (containsSubgraph(graph, k5) || containsSubgraph(graph,
49         k33a) || containsSubgraph(graph, k33b)) {
50         return false; // The graph is non-planar
51     }
52     return true; // The graph is planar
53 }
54
55 int main() {
56     int vertices, edges;
57     Graph graph(vertices);
58     for (int i = 0; i < edges; ++i) {
59         int u, v; cin >> u >> v;
60         graph.addEdge(u, v);
61     }
62     if (isPlanar(graph)) {
63         cout << "The graph is planar." << endl;
64     } else {
65         cout << "The graph is non-planar." << endl;
66     }
67 }

```

3.2 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             }
29         }
30     }
31 }

```

```

27         pq.push({dis[from]+weight,to});
28     }
29 }
30 }
31 }
32 int main(){
33     int startpoint;
34     dijkstra(startpoint);
35     //dis and parent
36 }

```

3.3 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
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     int n;
37     floyd_warshall(n);
38     for(int i=0;i<4;i++){
39         for(int j=0;j<4;j++){
40             cout << dis[i][j]<<" ";
41         }
42         cout << endl;
43     }
44     find_path(0,2);
45 }

```

3.4 2 sat

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 class TwoSAT {
5 public:
6     TwoSAT(int n) : n(n), graph(2 * n), visited(2 * n, false) {}
7     void addClause(int a, int b) { // 0-base;
8         a *= 2;
9         b *= 2;
10        // Add implications (~a => b) and (~b => a)
11        graph[a ^ 1].push_back(b);
12        graph[b ^ 1].push_back(a);
13    }
14    bool solve() { // Find SCCs and check for contradictions
15        for (int i = 0; i < 2 * n; ++i) {
16            if (!visited[i]) {
17                dfs1(i);
18            }
19        }
20        reverse(processOrder.begin(), processOrder.end()); // topological sort
21        for (int i = 0; i < 2 * n; ++i) {
22            visited[i] = false;
23        }
24        for (int node : processOrder) {
25            if (!visited[node]) {
26                scc.clear();
27                dfs2(node);
28                if (!checkSCCConsistency()) {
29                    return false;
30                }
31            }
32        }
33        return true;
34    }
35 private:
36     int n;
37     vector<vector<int>>> graph;
38     vector<bool> visited;
39     vector<int> processOrder;
40     vector<int> scc;
41
42     void dfs1(int node) {
43         visited[node] = true;
44         for (int neighbor : graph[node]) {
45             if (!visited[neighbor]) {
46                 dfs1(neighbor);
47             }
48         }
49         processOrder.push_back(node);
50     }
51
52     void dfs2(int node) {
53         visited[node] = true;
54         scc.push_back(node);
55         for (int neighbor : graph[node]) {
56             if (!visited[neighbor]) {
57                 dfs2(neighbor);
58             }
59         }
60     }
61 }
62
63 bool checkSCCConsistency() {

```

```

65     for (int node : scc) {
66         if (find(scc.begin(), scc.end(), node ^ 1) != scc
67             .end()) {
68             return false; // Contradiction found in the
69                             // same SCC
70         }
71     }
72     return true;
73 }
74 int main() {
75     int n, m; // Number of variables and clauses
76     TwoSAT twoSat(n);
77     for (int i = 0; i < m; ++i) {
78         int a, b;
79         twoSat.addClause(a, b);
80     }
81     if (twoSat.solve()) {
82         cout << "Satisfiable" << endl;
83     } else {
84         cout << "Unsatisfiable" << endl;
85     }
86 }

```

3.5 Kosaraju 2dfs

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int n = 16;
5 vector<vector<int>>> graph;
6 vector<vector<int>>> reverse_graph;
7 int visit[n];
8 int contract[n]; // 每個點收縮到的點
9 vector<vector<int>>> block;
10 vector<int> finish; // fake topological sort
11 // need graph and reverse graph
12 void dfs1(int x){
13     visit[x] = true;
14     for(auto to:graph[x]){
15         if(!visit[to]){
16             dfs1(to);
17         }
18     }
19     finish.push_back(x);
20 }
21 void dfs2(int x,int c){
22     contract[x] = c;
23     block[c].push_back(x);
24     visit[x] = true;
25     for(auto to:reverse_graph[x]){
26         if(!visit[to]){
27             dfs2(to,c);
28         }
29     }
30 }
31 int main(){
32     graph = {};
33     reverse_graph = {};
34
35     for(int i=0;i<n;i++){
36         if (!visit[i])
37             dfs1(i);

```

```

38     }
39     int c = 0;
40     memset(visit, 0, sizeof(visit));
41     for(int i = n-1; i >= 0; i--){
42         if(!visit[finish[i]]){
43             block.push_back({});
44             dfs2(finish[i], c++);
45         }
46     }
47     for(auto t: block){
48         for(auto x: t){
49             cout << x << " ";
50         }
51     }
52 }

```

3.6 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            return false; //分析 4
21        }
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
36    vector<vector<int>> get_match(){
37        vector<vector<int>> res;
38        for(int i = 0; i < x_cnt; i++){
39            if(mx[i] != -1){
40                res.push_back({i, mx[i]});
41            }
42        }
43        return res;
44    }
45
46    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
50 int main(){
51     int n, m;
52     Bipartite_matching bm;
53     for(int i = 0; i < n; i++){
54         int a, b; cin >> a >> b;
55         bm.add_edge(a, b);
56     }
57     bm.init(n);
58     cout << bm.bipartite_matching() << endl;
59     auto match = bm.get_match();
60     for(auto t: match){
61         cout << t[0] << " " << t[1] << endl;
62     }
63 }

```

3.7 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
23 if(visit[x] == low[x]){ //scc ㊦ 最早拜訪的
24     int j;
25     block.push_back({});
26     do{
27         j = st[--top];
28         instack[j] = false;
29         block[block.size()-1].push_back(j);
30         contract[j] = x;
31     }while(j != x);
32 }
33
34 int main(){
35     for(int i = 0; i < n; i++){
36         if(!visit[i])
37             dfs(i, i);
38     }
39     for(auto t: block){
40         for(auto x: t){

```

```

40         cout << x << " ";
41     }
42     }
43 }

```

3.8 topological sort

```

1
2 #include <bits/stdc++.h>
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){ //reverse(order) is topo
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 } //for() if(!vis[i])DFS(i)
18
19 int main(){
20     for(int i = 0; i < n; ++i){
21         if(!visit[i])
22             DFS(i);
23     }
24     if(cycle)
25         cout << "圖上有環";
26     else
27         for(int i = n-1; i >= 0; --i)
28             cout << order[i];

```

3.9 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; // 最高祖先預設自己
12     for(auto to: graph[x]){
13         if(visit[to]){ // back edge
14             if(to != parent){
15                 trace[x] = to;
16             }
17         }
18         else{ // tree edge
19             dfs(to, x);
20             if(visit[trace[to]] < visit[trace[x]])
21                 trace[x] = trace[to];

```

```

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 } // call for() dfs(i, -1)
29 int main(){
30     for(int i = 0; i < 9; i++){
31         if(!visit[i])
32             dfs(i, -1);
33     }
34     for(auto x: bridge){
35         cout << x[0] << " " << x[1] << endl;
36     }
37 }

```

3.10 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 {
13     visited[node] = true;
14     disc[node] = t;
15     low[node] = t;
16     t++;
17     int children = 0;
18
19     for (int neighbor : graph[node])
20     {
21         if (!visited[neighbor])
22         {
23             children++;
24             parent_array[neighbor] = node;
25             dfs_articulation(neighbor, node);
26             low[node] = min(low[node], low[neighbor]);
27
28             if (low[neighbor] >= disc[node] && parent != -1)
29             {
30                 is_articulation[node] = true;
31             }
32         }
33         else if (neighbor != parent)
34         {
35             low[node] = min(low[node], disc[neighbor]);
36         }
37     }
38
39     if (parent == -1 && children > 1)
40     {
41         is_articulation[node] = true;
42     }
43 } // call for() dfs(i, -1)
44 int main(){

```

```

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

```

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) ?
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);

```

```

74 cout << "Are 0 and 2 connected? " << (dsu.aresame(0, 2) ?
    "Yes" : "No") << endl;
75 cout << "Are 1 and 3 connected? " << (dsu.aresame(1, 3) ?
    "Yes" : "No") << endl;
76
77 return 0;
78 }

```

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();
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     }cout << endl;
40     for(int i = 1;i<n;i++){
41         bit.update(i,-1);
42     }
43     for(int i = 1;i<n;i++){
44         cout << bit.query(i)<<" ";
45     }cout << endl;
46 }

```

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+
        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
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 =
        {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         }cout << endl;
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     }
35 }

```

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             return dis[t]!=-1;
32         }
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         }
60         return flow;
61     }
62 };

```

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 K and K ,
44     vector<int> k5 = {0, 1, 2, 3, 4}; // Vertices of K
45     vector<int> k3a = {0, 1, 2}; // Vertices of K
46     , (part A)
47     vector<int> k3b = {3, 4, 5}; // Vertices of K
48     , (part B)
49
50     if (containsSubgraph(graph, k5) || containsSubgraph(graph,
51         k3a) || containsSubgraph(graph, k3b)) {
52         return false; // The graph is non-planar
53     }
54     return true; // The graph is planar
55 }
56
57 int main() {
58     int vertices, edges;
59     cin >> vertices;
60     cin >> edges;
61
62     Graph graph(vertices);
63     for (int i = 0; i < edges; ++i) {
64         int u, v;
65         cin >> u >> v;
66         graph.addEdge(u, v);
67     }
68     if (isPlanar(graph)) {
69         cout << "The graph is planar." << endl;
70     } else {
71         cout << "The graph is non-planar." << endl;
72     }
73     return 0;
74 }

```

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; // 圖中有中繼點就結束
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     }
48     find_path(0,2);

```

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         reverse(processingOrder.begin(), processingOrder.end());
28         // topological sort
29         for (int i = 0; i < 2 * n; ++i) {
30             visited[i] = false;
31         }
32         for (int node : processingOrder) {
33             if (!visited[node]) {
34                 scc.clear();
35                 dfs2(node);
36                 if (!checkSCCConsistency()) {
37                     return false;
38                 }
39             }
40         }
41         return true;
42     }
43 private:
44     int n;
45     vector<vector<int>> graph;
46     vector<bool> visited;
47     vector<int> processingOrder;
48     vector<int> scc;
49
50     void dfs1(int node) {
51         visited[node] = true;
52         for (int neighbor : graph[node]) {
53             if (!visited[neighbor]) {
54                 dfs1(neighbor);
55             }
56         }
57         processingOrder.push_back(node);
58     }
59
60     void dfs2(int node) {
61         visited[node] = true;
62         scc.push_back(node);
63         for (int neighbor : graph[node]) {
64             if (!visited[neighbor]) {
65                 dfs2(neighbor);
66             }
67         }
68     }
69
70     bool checkSCCConsistency() {
71         for (int node : scc) {
72             if (find(scc.begin(), scc.end(), node ^ 1) != scc
73                 .end()) {
74                 return false; // Contradiction found in the
75                                 // same SCC
76             }
77         }
78         return true;
79     };
80
81     int main() {
82         int n, m;
83         cin >> n >> m; // Number of variables and clauses
84         TwoSAT twoSat(n);

```



```

86     for (int i = 0; i < m; ++i) {
87         int a, b;
88         cin >> a >> b;
89         twoSat.addClause(a, b);
90     }
91
92     if (twoSat.solve()) {
93         cout << "Satisfiable" << endl;
94     } else {
95         cout << "Unsatisfiable" << endl;
96     }
97
98     return 0;
99 }
100 }

```

7.5 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    }

```

```

43    void add_edge(int i, int j){
44        edge[i].push_back(j);
45    }
46    void init(int x){
47        x_cnt = x;
48    }
49 };
50 int main(){
51     /*
52     0 3
53     0 4
54     1 3
55     1 5
56     2 3
57     2 4
58     2 5
59     */
60     Bipartite_matching bm;
61     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 }

```

7.6 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 int main(){
34     graph = {
35         {1},
36         {3, 4, 5},
37         {6},
38         {2},
39         {7},
40         {11, 15},
41         {2, 3},
42         {4, 6, 9},
43         {},
44         {},
45         {},
46         {15},
47         {14},
48         {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         }cout << endl;
60     }
61 }

```

7.7 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
20
21 int main() {
22     graph = {
23         {1, 2},
24         {3},
25         {3, 4},
26         {4},
27         {}
28     };

```

```

29     n = 5;
30     cycle = false;
31     for (int i=0; i<n; ++i){
32         if (!visit[i])
33             DFS(i);
34     }
35     if (cycle)
36         cout << "圖上有環";
37     else
38         for (int i=n-1; i>=0; --i)
39             cout << order[i];
40 }

```

8 Original Code/Math

8.1 extgcd

```

1  #include<bits/stdc++.h>
2
3  using namespace std;
4
5  int extgcd(int a,int b,int &x,int &y)//擴展歐幾里得算法
6  {
7      if (b==0)
8      {
9          x = 1;
10         y = 0;
11         return a; //到達遞歸邊界開始向上一層返回
12     }
13     int r = extgcd(b,a%b,x,y);
14     int temp=y; //把x y變成上一層的
15     y = x - (a / b) * y;
16     x = temp;
17     return r; //得到a b的最大公因數
18 }
19
20 int main(){
21     int a = 55,b = 80;
22     int x,y;
23     int GCD = extgcd(a,b,x,y);
24     cout << "GCD: " << GCD << endl;
25     cout << x << " " << y << endl;
26     cout << a*x+b*y << endl;
27 }
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拆分成二進位位數，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
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 }

```

9.3 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>& visited, const vector<vector<int>>& adj_list) {
6     visited[node] = true;
7     int max_distance = distance;
8     int farthest_node = node;
9
10     for (int neighbor : adj_list[node]) {
11         if (!visited[neighbor]) {
12             auto result = dfs(neighbor, distance + 1, visited, adj_list);
13             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,
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 }

```

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 }

```

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         }
28     } else if (neighbor != parent) {
29         low[node] = min(low[node], disc[neighbor]);
30     }
31 }

```

```

31 }
32
33 if (parent == -1 && children > 1) {
34     is_articulation[node] = true;
35 }
36 }
37
38 int main(){
39     graph = {
40         {1,2},
41         {3},
42         {5,6},
43         {7},
44         {},
45         {},
46         {},
47         {8},
48         {4},
49     };
50     for (int i = 0; i < n; ++i) {
51         if (!visited[i]) {
52             dfs_articulation(i, -1);
53         }
54     }
55     cout << "Articulation Points: ";
56     for (int i = 0; i < n; ++i) {
57         if (is_articulation[i]) {
58             cout << i << " ";
59         }
60     }
61     cout << endl;
62 }

```

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
28         先。不斷上升逼近之。
29         if (k & (1<<i))
30             x = ancestor[x][i];
31     return x;
32 }
33 void rooted_tree(int root){// build the tree with root at "
34     根
35     ancestor[root][0] = root;
36     dfs(root);
37     table();
38 }
39 int LCA(int x,int y){
40     if (is_ancestor(x, y)) return x;
41     if (is_ancestor(y, x)) return y;
42     for (int i=logn-1; i>=0; i--)
43         if (!is_ancestor(ancestor[x][i], y))
44             x = ancestor[x][i];
45     return ancestor[x][0];
46 }
47 int main(){
48     graph = {
49         {1,2},
50         {3},
51         {5,6},
52         {7},
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
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     return make_pair(max_distance, farthest_node);
22 }
23 // Calculate the radius of the tree using DFS
24 int tree_radius(const vector<vector<int>>> &adj_list){
25     int num_nodes = adj_list.size();
26     vector<bool> visited(num_nodes, false);
27
28     // 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 int main() {
40     vector<vector<int>>> adj_list;
41     int radius = tree_radius(adj_list);
42     cout << "Tree radius: " << radius << endl;
43     return 0;
44 }

```

NYCU_SEGMENTTREE CODEBOOK

Contents

1	Data Structure	1
1.1	dsu class	1
1.2	monotonic queue	1
1.3	BIT	1
1.4	segment tree simple add	1
1.5	monotonic stack	1
2	Flow	2
2.1	dicnic	2
3	Gaph	2
3.1	planar	2
3.2	Dijkstra	2

3.3	Floyd Warshall	3
3.4	2 sat	3
3.5	Kosaraju 2dfs	3
3.6	bipartite matching	4
3.7	tarjan SCC	4
3.8	topological sort	4
3.9	bridge	4
3.10	Articulation vertex	5

4	Math	5
4.1	extgcd	5
5	Original Code/Data Structure	5
5.1	dsu class	5
5.2	monotonic queue	6
5.3	BIT	6
5.4	segment tree simple add	6
5.5	monotonic stack	6

6	Original Code/Flow	7
6.1	dicnic	7

7	Original Code/Graph	7
7.1	planar	7
7.2	Dijkstra	7
7.3	Floyd Warshall	8
7.4	2 sat	8
7.5	bipartite matching	9
7.6	tarjan SCC	9
7.7	topological sort	9

8	Original Code/Math	10
8.1	extgcd	10

9	Original Code/Tree	10
9.1	LCA	10
9.2	diameter	10
9.3	radius	10
9.4	bridge	11
9.5	Articulation vertex	11

10	Tree	11
10.1	LCA	11
10.2	diameter	12
10.3	radius	12