1 Data Structure

1.1 DSU

```cpp
class DSU{
public:
    DSU(int n ){
        this->n = n;
        reset();
    }
    int n;
    vector<int> boss;
    vector<int> rank;
    vector<int> size;
    void reset(){
        this->boss.resize(n);
        this->rank.resize(n,0);
        this->size.resize(n,0);
        for(int i =0;i<n;i++){
            boss[i] = i;
        }
    }
    int find(int x){
        if(boss[x]!= x){
            boss[x] = find(boss[x]);
        }
        return boss[x];
    }
    int get_size(int x){
        return size[find(x)];
    }
    void merge(int x, int y){
        int a = find(x);
        int b = find(y);
        if(a!=b){
            if(rank[a]<rank[b]){
                boss[a] = b;
                size[b] += size[a];
            }else if (rank[a]<rank[b]){
                boss[b] = a;
                size[a] += size[b];
            }else{
                boss[a] = b;
                size[b] += size[a];
                rank[b]++;
            }
        }
    }
    bool aresame(int a,int b){
        return find(a)==find(b);
    }
};
```

1.2 Monotonic Queue

```cpp
class Monotonic_queue{
private:
    deque<int> qu;
public:
    void push(int n){
        while(!qu.empty()&&qu.back()<n){
            qu.pop_back();
        }
        qu.push_back(n);
    }
    int max(){
        return qu.front();
    }
    int min(){
        return qu.back();
    }
    int  size(){
        return qu.size();
    }
    void pop(){
        qu.pop_front();
    }
};
```

1.3 BIT

```cpp
class BIT{
public:
    vector<int> bit;
    int N;
    BIT(int n){
        this->N = n;
        this->bit.resize(n);
    }
    void update(int x,int d){
        while(x<=N){
            bit[x] +=d;
            x +=x&(-x);// lowest bit in x;
        }
    }
    int query(int x){
        int res = 0;
        while(x){
            res+= bit[x];
            x -= x& -x;
        }
        return res;
    }
};
```

1.4 Segment Tree

```cpp
class SegmentTree{
private:
    const int n;
    const vl arr;
    // vl st;
    vl summ;
    vl minn;
    vl maxx;
    vl tag;
    void pull(int l,int r,int v){
        if(r-l==1)
            return;
        // st[v]=st[2*v+1]+st[2*v+2];
        int mid=(l+r)/2;
        push(l,mid,2*v+1);
        push(mid,r,2*v+2);
        summ[v]=summ[2*v+1]+summ[2*v+2];
        // minn[v]=min(minn[2*v+1],minn[2*v+2]);
        // maxx[v]=max(maxx[2*v+1],minn[2*v+2]);
    }
    void push(int l,int r,int v){
        summ[v]+=tag[v]*(r-l);
        if(r-l==1)
            return tag[v]=0,void();
        tag[2*v+1]+=tag[v];
        tag[2*v+2]+=tag[v];
        tag[v]=0;
    }
    void build(int l,int r,int v=0){
        if(r-l==1){
            summ[v]=arr[l];
            // summ[v]=minn[v]=maxx[v]=arr[l];
            return;
        }
        int mid=(l+r)/2;
        build(l,mid,2*v+1);
        build(mid,r,2*v+2);
        pull(l,r,v);
    }

public:
    SegmentTree(vl&_arr,int _n):arr(_arr),n(_n){
        assert(arr.size()==n);
        summ.assign(4*n,0);
        // minn.assign(4*n,1e9);
        // maxx.assign(4*n,-1e9);
        tag.assign(4*n,0);
        build(0,arr.size());
    }
    void modify(int x,int val,int l,int r,int v=0){

    }
    // query sum
    loli query(int L,int R,int l,int r,int v=0){
        // dbn(L,R,l,r,v)
        push(l,r,v);
        if(l==L && R==r){
            return summ[v];
            return minn[v];
            return maxx[v];
        }
        int mid=(l+r)/2;
        if(R<=mid)
            return query(L,R,l,mid,2*v+1);
        else if(mid<=L)
            return query(L,R,mid,r,2*v+2);
        else
            return query(L,mid,l,mid,2*v+1)+query(mid,R,mid,r
                ,2*v+2);
    }
    // plus `val` to every element in [L,R)
    void update(int L,int R,loli val,int l,int r,int v=0){
        // dbn(L,R,l,r,v)
        push(l,r,v);
        if(l==L && R==r){
            tag[v]+=val;
            push(l,r,v);
            return;
        }
        int mid=(l+r)/2;
        if(R<=mid)
            update(L,R,val,l,mid,2*v+1);
        else if(mid<=L)
            update(L,R,val,mid,r,2*v+2);
        else
            update(L,mid,val,l,mid,2*v+1),update(mid,R,val,
                mid,r,2*v+2);
        pull(l,r,v);
    }
};

void solve(){
    int n,q;
    cin>>n>>q;
    vl arr(n);
    for(auto&x:arr)
        cin>>x;
    SegmentTree st(arr,n);
    while(q--){
        int op=0;
        // str op;
        cin>>op;
        if(op&1){
            loli l,r,val;
            cin>>l>>r>>val;
            assert(r>=l);
            st.update(l-1,r,val,0,n);
            // loli k,u;
            // cin>>k>>u;
            // st.update(k-1,k,u-arr[k-1],0,n);
            // arr[k-1]=u;
        }else{
            int x,y;
            cin>>x>>y;
            assert(y>=x);
            cout<<st.query(x-1,y,0,n)<<endl;
        }
    }
}
```

1.5 Sparse Table

```
int a[N], sp[__lg(N) + 1][N]{};
void init(int n) { //0-based
  for (int i = 0; i < n; ++i) {
    sp[0][i] = a[i];
  }
  for (int i = 0; i < __lg(n); ++i) {
    for (int j = 0; j+(1<<i) < n; ++j) {
      sp[i + 1][j] = max(sp[i][j], sp[i][j+(1<<i)]);
    }
  }
}
int query(int l, int r) { //[l, r]
  int p = __lg(r - l + 1);
  return max(sp[p][l], sp[p][r-(1<<p)+1]);
}
```

1.6 Monotonic Stack

```
vector<int> monotonic_stack(vector<int> nums){
    int n = nums.size();
    vector<int> res(n);
    stack<int>  st;
    for(int i = n-1;i>=0;i--){
        while(!st.empty() && st.top()<=nums[i]){
            st.pop();
        }
        if(st.empty())res[i] = -1;
        else res[i] = st.top();
        st.push(nums[i]);
    }
    return res;
}
```

2 Flow

2.1 Dinic

```
#define maxn 2005
#define INF 0x3f3f3f3f
struct MaxFlow{
    struct edge{
        int  to, cap, flow,rev;
        edge( int v, int c, int f,int r) :  to(v), cap(c),
             flow(f),rev(r) {}
      };
    vector<edge> G[maxn];
    int s,t,dis[maxn],cur[maxn],vis[maxn];
    void add_edge(int from,int to,int cap){
        G[from].push_back(edge(to,cap,0,G[to].size()));
        G[to].push_back(edge(from,0,0,G[from].size()-1));
    }
    bool bfs(){
        memset(dis, -1, sizeof(dis));
        queue<int> qu;
        qu.push(s);
        dis[s] = 0;
        while (!qu.empty()) {
            int from = qu.front();
            qu.pop();
            for (auto &e: G[from]) {
                if (dis[e.to]==-1 && e.cap != e.flow) {
                    dis[e.to] = dis[from] + 1;
                    qu.push(e.to);
                }
            }
        }
        return dis[t]!=-1;
    }
    int dfs(int from,int cap){
        if(from==t ||cap==0)return cap;
        for(int &i = cur[from];i<G[from].size();i++){
            edge &e = G[from][i];
            if(dis[e.to]==dis[from]+1 && e.flow!=e.cap){
                int df = dfs(e.to,min(e.cap-e.flow,cap));
                if(df){
                    e.flow+=df;
                    G[e.to][e.rev].flow-=df;
                    return df;
                }
            }
        }
        dis[from] = -1;
        return 0;
    }
    int Maxflow(int s,int t){
        this->s = s,this->t =t;
        int flow = 0;
        int df;
        while(bfs()){
            memset(cur,0,sizeof(cur));
            while(df = dfs(s,INF)){
                flow +=df;
            }
        }
        return flow;
    }
};
int main(){
    int n = 4,m = 6;
    MaxFlow maxflow;
    for(int i =0;i<m;i++){
        int a,b,cap;
        cin >>a>>b>>cap;
        maxflow.add_edge(a,b,cap);
    }
    cout << maxflow.Maxflow(1,3)<<endl;;
}
```

3 Geometry

3.1 Sort by Angle

```
bool cmp(pii a, pii b) {
#define is_neg(k) (k.y < 0 || (k.y == 0 && k.x < 0));
  int A = is_neg(a), B = is_neg(b);
  if (A != B)
    return A < B;
  if (cross(a, b) == 0)
    return (a.x*a.x + a.y*a.y) < (b.x*b.x + b.y*b.y);
  return cross(a, b) > 0;
}
```

3.2 Convex Hull

```
using pdd = pair<double, double>;
#define F first
#define S second
pdd operator-(pdd a, pdd b) {
  return {a.F - b.F, a.S - b.S};
}
double cross(pdd a, pdd b) {
  return a.F * b.S - a.S * b.F;
}
void solve() {
  int n;
  cin >> n;
  vector<pdd> pnts;
  for (int i = 0; i < n; ++i) {
    double x, y;
    cin >> x >> y;
    pnts.push_back(x, y);
  }
  sort(iter(pnts));
  vector<pdd> hull;
  for (int i = 0; i < 2; ++i) {
    int t = hull.size();
    for (pdd j: pnts) {
      while(hull.size() - t >= 2 && cross(j - hull[hull.size() - 2], hull.back() - hull[hull.size() - 2]) >= 0)
        hull.pop_back();
      hull.push_back(j);
    }
    hull.pop_back();
    reverse(iter(pnts));
  }
  double area = 0;
  for (int i=0; i < hull.size(); ++i){
    area += cross(hull[i], hull[(i + 1) % hull.size()]);
  }
  area /= 2.0;
}
```

3.3 Point in Polygon

```
const ll inf = 2000000000;
struct Point {
  ll x, y;
  Point(ll x = 0, ll y = 0):x(x), y(y){}
  Point operator+(const Point p) const {
    return Point(x + p.x, y + p.y); }
  Point operator-(const Point p) const {
    return Point(x - p.x, y - p.y); }
  ll operator*(const Point p) const { //dot
    return x * p.x + y * p.y; }
  ll operator^(const Point p) const { //cross
    return x * p.y - y * p.x; }
};
bool onseg(Point a, Point b, Point o) {
  return ((a - o) ^ (b - o)) == 0 && ((a - o) * (b - o)) <= 0;
}
int ori(Point a, Point b, Point o) {
  ll w = (a - o) ^ (b - o);
  return (w ? (w > 0 ? 1 : -1) : 0);
}
bool inters(Point a, Point b, Point c, Point d) {
  if (onseg(a, b, c) || onseg(a, b, d)) return 1;
  if (onseg(c, d, a) || onseg(c, d, b)) return 1;
  if (ori(a, b, c) * ori(a, b, d) < 0 && ori(c, d, a) * ori(c, d, b) < 0) return 1;
  return 0;
}
Point poly[maxn];
void solve(int n, Point p) {
  poly[n] = poly[0];
  int cnt = 0;
  for (int i = 0; i < n; ++i) {
    if (onseg(poly[i], poly[i + 1], p)) {
      cnt = -1;
      break;
    }
    if (inters(poly[i], poly[i + 1], p, Point(inf, p.y))) {
      ++cnt;
    }
    Point hi = (poly[i].y > poly[i + 1].y ? poly[i] : poly[i + 1]);
    if (hi.y == p.y && hi.x > p.x) {
      --cnt;
    }
  }
  if (cnt < 0)
    cout << "BOUNDARY\n";
  else if (cnt % 2)
    cout << "INSIDE\n";
  else
    cout << "OUTSIDE\n";
}
```

3.4 MinCoveringCircle

```
double dis(pdd a, pdd b) {
  double dx = a.x - b.x, dy = a.y - b.y;
  return sqrt(dx*dx + dy*dy);
}
double sq(double x) {
  return x * x;
}
pdd excenter(pdd p1, pdd p2, pdd p3) {
  double a1 = p1.x - p2.x, a2 = p1.x - p3.x;
  double b1 = p1.y - p2.y, b2 = p1.y - p3.y;
  double c1 = (sq(p1.x) - sq(p2.x) + sq(p1.y) - sq(p2.y)) / 2;
  double c2 = (sq(p1.x) - sq(p3.x) + sq(p1.y) - sq(p3.y)) / 2;
  double dd = a1*b2 - a2*b1;
  return {(c1*b2 - c2*b1) / dd, (a1*c2 - a2*c1) / dd};
}
void solve(pdd a[], int n) {
  shuffle(a, a + n, rng);
  pdd center = a[0];
  double r = 0;
  for (int i = 1; i < n; ++i) {
    if (dis(center, a[i]) <= r) continue;
    center = a[i], r = 0;
    for (int j = 0; j < i; ++j) {
      if (dis(center, a[j]) <= r) continue;
      center.x = (a[i].x + a[j].x) / 2;
      center.y = (a[i].y + a[j].y) / 2;
      r = dis(center, a[i]);
      for (int k = 0; k < j; ++k) {
        if (dis(center, a[k]) <= r) continue;
        center = excenter(a[i], a[j], a[k]);
        r = dis(center, a[i]);
      }
    }
  }
  cout << fixed << setprecision(10) << r << '\n';
  cout << center.x << ' ' << center.y << '\n';
}
```

4 Graph

4.1 Bipartite Matching

```
const int MAXN = 100;

struct Bipartite_matching{
    int mx[MAXN], my[MAXN], vy[MAXN]; //matchX, matchY, visitY
    vector<int> edge[MAXN]; //adjcent list;
    int x_cnt;
    bool dfs(int x){
        for(auto y: edge[x]){ //對 x 可以碰到的邊進行檢查
            if(vy[y] == 1) continue; //避免遞F error

            vy[y] = 1;
            if(my[y] == -1 || dfs(my[y])){ //分析 3
                mx[x] = y;
                my[y] = x;
                return true;
            }
        }
        return false; //分析 4
    }

    int bipartite_matching(){
        memset(mx, -1, sizeof(mx)); //分析 1,2
        memset(my, -1, sizeof(my));
        int ans = 0;
        for(int i = 0; i < x_cnt; i++){  //對每一個 x 節點進行 DFS(最大匹配)
            memset(vy, 0, sizeof(vy));
            if(dfs(i)) ans++;
        }
        return ans;
    }
    vector<vector<int>> get_match(){
        vector<vector<int>> res;
        for(int i =0 ;i<x_cnt;i++){
            if(mx[i]!=-1){
                res.push_back({i,mx[i]});
            }
        }
        return res;
    }
    void add_edge(int i,int j){
        edge[i].push_back(j);
    }
    void init(int x){
        x_cnt = x;
    }
};
int main(){
    int n,m;
    Bipartite_matching bm;
    for(int i = 0;i<m;i++){
        int a , b;cin >>a>>b;
        bm.add_edge(a,b);
    }
   bm.init(n);
    cout << bm.bipartite_matching()<<endl;
    auto match = bm.get_match();
    for(auto t: match){
        cout << t[0]<<" "<<t[1]<<endl;
    }
}
```

4.2 Tarjan SCC

```
const int n = 16;
vector<vector<int>> graph;
int visit[n], low[n],t = 0;
int st[n], top =0;
bool instack[n];
int contract[n]; // 每個點收縮到的點
vector<vector<int>> block;
void dfs(int x,int parent){
    // cout <<x<<endl;
    visit[x] = low[x] = ++t;
  st[top++] = x;
  instack[x] = true;
    for(auto to: graph[x]){
        if(!visit[to])
            dfs(to,x);

        if(instack[to])
            low[x] = min(low[x],low[to]);
    }
    if(visit[x]==low[x]){ //scc 🄵最早拜訪的
        int j;
        block.push_back({});
        do{
            j = st[--top];
            instack[j] = false;
            block[block.size()-1].push_back(j);
            contract[j] =x;
        }while(j!=x);
    }
}
int main(){
    for(int i =0;i<n;i++){
        if (!visit[i])
      dfs(i, i);
    }
    for(auto t: block){
        for(auto x:t){
            cout << x <<" ";
        }cout <<endl;
    }
}
```

4.3 Bridge

```
const int n = 9;
vector<vector<int>> graph;
vector<int> visit(n, 0);
vector<int> trace(n, 0);
vector<vector<int>> bridge;
int t = 0;
void dfs(int x, int parent){
    visit[x] = ++t;
    trace[x] = x; // 最高祖先預設🄵自己
    for (auto to : graph[x]){
        if (visit[to]){ // back edge
            if (to != parent){
                trace[x] = to;
            }
        }
        else{ // treeedge
            dfs(to, x);
            if (visit[trace[to]] < visit[trace[x]])
                trace[x] = trace[to];

            // 子樹回不到祖先暨自身。
            if (visit[trace[to]] > visit[x])
                bridge.push_back({x, to});
        }
    }
}//call for()dfs(i,-1)
int main(){
    for(int i =0;i<9;i++){
        if(!visit[i])
            dfs(i,-1);
    }
    for(auto x: bridge){
        cout << x[0]<<" "<< x[1]<<endl;
    }
}
```

4.4 2 SAT

```
class TwoSAT{
public:
    TwoSAT(int n) : n(n), graph(2 * n), visited(2 * n, false) {}
    void addClause(int a, int b) {// 0-base;
        a *=2;
        b *=2;
        // Add implications (~a => b) and (~b => a)
        graph[a ^ 1].push_back(b);
        graph[b ^ 1].push_back(a);
    }
    bool solve() {// Find SCCs and check for contradictions
        for (int i = 0; i < 2 * n; ++i) {
            if (!visited[i]) {
                dfs1(i);
            }
        }
        reverse(processingOrder.begin(), processingOrder.end());//topological sort
        for (int i = 0; i < 2 * n; ++i) {
            visited[i] = false;
        }
        for (int node : processingOrder) {
            if (!visited[node]) {
                scc.clear();
                dfs2(node);
                if (!checkSCCConsistency()) {
                    return false;
                }
            }
        }

        return true;
    }

private:
    int n;
    vector<vector<int>> graph;
    vector<bool> visited;
    vector<int> processingOrder;
    vector<int> scc;

    void dfs1(int node) {
        visited[node] = true;
        for (int neighbor : graph[node]) {
            if (!visited[neighbor]) {
                dfs1(neighbor);
            }
        }
        processingOrder.push_back(node);
    }

    void dfs2(int node) {
        visited[node] = true;
        scc.push_back(node);
        for (int neighbor : graph[node]) {
            if (!visited[neighbor]) {
                dfs2(neighbor);
            }
        }
    }

    bool checkSCCConsistency() {
        for (int node : scc) {
            if (find(scc.begin(), scc.end(), node ^ 1) != scc.end()) {
                return false; // Contradiction found in the same SCC
            }
        }
        return true;
    }
};
int main() {
    int n, m;// Number of variables and clauses
    TwoSAT twoSat(n);
    for (int i = 0; i < m; ++i) {
        int a, b;
        twoSat.addClause(a, b);
    }
    if (twoSat.solve()) {
        cout << "Satisfiable" << endl;
    } else {
        cout << "Unsatisfiable" << endl;
    }
}
```

4.5 Kosaraju 2DFS

```
const int n = 16;
vector<vector<int>> graph;
vector<vector<int>> reverse_graph;
int visit[n];
int contract[n]; // 每個點收縮到的點
vector<vector<int>> block;
vector<int> finish;//fake topological sort
```

```
// need graph and reverse praph
void dfs1(int x){
    visit[x] = true;
    for(auto to:graph[x]){
        if(!visit[to]){
            dfs1(to);
        }
    }
    finish.push_back(x);
}
void dfs2(int x,int c){
    contract[x] = c;
    block[c].push_back(x);
    visit[x] = true;
    for(auto to:reverse_graph[x]){
        if(!visit[to]){
            dfs2(to,c);
        }
    }
}
int main(){
    graph = {};
    reverse_graph = {};

    for(int i =0;i<n;i++){
        if (!visit[i])
      dfs1(i);
    }
    int c =0;
    memset(visit,0,sizeof(visit));
    for(int i =  n-1;i>=0;i--){
        if(!visit[finish[i]]){
            block.push_back({});
            dfs2(finish[i],c++);
        }
    }
    for(auto t: block){
        for(auto x:t){
            cout << x <<" ";
        }cout <<endl;
    }
}
```

4.6 Dijkstra

```
#define maxn 200005
vector<int> dis(maxn,-1);
vector<int> parent(maxn,-1);
vector<bool> vis(maxn,false);
vector<vector<pair<int,int>>> graph;
void dijsktra(int source){
    dis[source] =0 ;

    priority_queue<pair<int,int>,vector<pair<int,int>>, greater<pair<int,int>>> pq;
    pq.push({0,source});
    while(!pq.empty()){
        int from  = pq.top().second;
        pq.pop();
        // cout <<vis[from]<<endl;
        if(vis[from])continue;
        vis[from] = true;
        for(auto next : graph[from]){
            int to = next.second;
            int weight = next.first;
            // cout <<from<<' ' <<to<<' ' <<weight;
            if(dis[from]+weight< dis[to] || dis[to]==-1){
                dis[to] = dis[from]+weight;
                parent[to] = from;
                pq.push({dis[from]+weight,to});
            }
        }
    }
}
int main(){
    int startpoint;
    dijsktra(startpoint);
    //dis and parent
}
```

4.7 Floyd Warshall

```
#define maxn 2005
vector<vector<int>> dis(maxn,vector<int>(maxn,9999999));
vector<vector<int>> mid(maxn,vector<int>(maxn,-1));
vector<vector<pair<int,int>>> graph;

void floyd_warshall(int n ){ // n is n nodes
  for(int i =0;i<n;i++){
        for(auto path:graph[i]){
            dis[i][path.second] = path.first;
        }
    }
  for (int i=0; i<n; i++)
    dis[i][i] = 0;
  for (int k=0; k<n; k++){
    for (int i=0; i<n; i++){
      for (int j=0; j<n; j++){
        if (dis[i][k] + dis[k][j] < dis[i][j] || dis[i][j]==-1){
          dis[i][j] = dis[i][k] + dis[k][j];
          mid[i][j] = k;  // 由i點走到j點經過了k點
        }
            }
        }
    }
}
void find_path(int s, int t){   // 印出最短路徑
  if (mid[s][t] == -1) return;  // 🄵有中繼點就結束
  find_path(s, mid[s][t]);     // 前半段最短路徑
  cout << mid[s][t];         // 中繼點
  find_path(mid[s][t], t);     // 後半段最短路徑
}
int main(){
    int n;
    floyd_warshall(n);
    for(int i =0;i<4;i++){
        for(int j = 0 ; j <4;j++)
            cout << dis[i][j]<<" ";
        cout << endl;
    }
    find_path(0,2);
}
```

4.8 Articulation Vertex

```
const int n = 9;
int t = 0;
vector<int> disc(n, -1);         // Discovery time
vector<int> low(n, -1);          // Low time
vector<int> parent_array(n, -1); // Parent in DFS tree
vector<bool> visited(n, false);
vector<bool> is_articulation(n, false);
vector<vector<int>> graph;
void dfs_articulation(int node, int parent){
    visited[node] = true;
    disc[node] = t;
    low[node] = t;
    t++;
    int children = 0;

    for (int neighbor : graph[node])
    {
        if (!visited[neighbor])
        {
            children++;
            parent_array[neighbor] = node;
            dfs_articulation(neighbor, node);
            low[node] = min(low[node], low[neighbor]);

            if (low[neighbor] >= disc[node] && parent != -1)
            {
                is_articulation[node] = true;
            }
        }
        else if (neighbor != parent)
        {
            low[node] = min(low[node], disc[neighbor]);
        }
    }

    if (parent == -1 && children > 1)
    {
        is_articulation[node] = true;
    }
}//call for() dfs(i,-1)
int main(){
    for (int i = 0; i < n; ++i) {
        if (!visited[i]) {
            dfs_articulation(i, -1);
        }
    }
    cout << "Articulation Points: ";
    for (int i = 0; i < n; ++i) {
        if (is_articulation[i]) {
            cout << i << " ";
        }
    }cout << endl;
}
```

4.9 Topological Sort

```
vector<vector<int>> graph;
vector<int> visit(10,0);
vector<int> order;
int n;
```

```
bool cycle;   // 記□DFS的過程中是否偵測到環
void DFS(int i){ //reverse(order) is topo
  if (visit[i] == 1) {cycle = true; return;}
  if (visit[i] == 2) return;
  visit[i] = 1;
  for(auto to :graph[i])
        DFS(to);
  visit[i] = 2;
    order.push_back(i);
}//for() if(!vis[i])DFS(i)
int main() {
  for (int i=0; i<n; ++i){
    if (!visit[i])
      DFS(i);
    }
  if (cycle)
    cout << "圖上有環";
  else
    for (int i=n-1; i>=0; --i)
      cout << order[i];
}
```

4.10 Planar

```
class Graph {
public:
    int V;
    vector<vector<int>> adj;
    Graph(int vertices) : V(vertices), adj(vertices) {}
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
};

bool containsSubgraph(const Graph& graph, const vector<int>& subgraph) {
    unordered_set<int> subgraphVertices(subgraph.begin(), subgraph.end());
    for (int vertex : subgraphVertices) {
        for (int neighbor : graph.adj[vertex]) {
            if (subgraphVertices.count(neighbor) == 0) {
                bool found = true;
                for (int v : subgraph) {
                    if (v != vertex && v != neighbor) {
                        if (graph.adj[v].size() < 3) {
                            found = false;
                            break;
                        }
                    }
                }
                if (found)
                    return true;
            }
        }
    }
    return false;
}

bool isPlanar(const Graph& graph) {
    // Subgraphs isomorphic to K and K ,
    vector<int> k5 = {0, 1, 2, 3, 4};       // Vertices of K
    vector<int> k33a = {0, 1, 2};           // Vertices of K, (part A)
    vector<int> k33b = {3, 4, 5};           // Vertices of K, (part B)

    if (containsSubgraph(graph, k5) || containsSubgraph(graph, k33a) || containsSubgraph(graph, k33b)) {
        return false; // The graph is non-planar
    }
    return true; // The graph is planar
}
int main() {
    int vertices, edges;
    Graph graph(vertices);
    for (int i = 0; i < edges; ++i) {
        int u, v;cin >> u >> v;
        graph.addEdge(u, v);
    }
    if (isPlanar(graph)) {
        cout << "The graph is planar." << endl;
    } else {
        cout << "The graph is non-planar." << endl;
    }
}
```

4.11 Heavy Light Decomposition

```
int dep[N], pa[N], sz[N], nxt[N];
int id[N], rt[N];
int dfs(int u, int lst, int d = 0) {
  dep[u] = d;
  pa[u] = lst;
  sz[u] = 1;
  nxt[u] = -1;
  for (int v: g[u]) {
    if (v == lst) continue;
    sz[u] += dfs(v, u, d + 1);
    if (nxt[u] == -1 || sz[v] > sz[nxt[u]]) {
      nxt[u] = v;
    }
  }
  return sz[u];
}
int tn = 0;
void mapId(int u, int lst, int root) {
  id[u] = ++tn;
  rt[u] = root;
  if (~nxt[u]) mapId(nxt[u], u, root);
  for (int v: g[u]) {
    if (v == lst || v == nxt[u]) continue;
    mapId(v, u, v);
  }
}
void solve() {
  while (rt[a] != rt[b]) {
    if (dep[rt[a]] > dep[rt[b]]) swap(a, b);
    //...
    b = pa[rt[b]];
  }
  if (a != b) {
    if (id[a] > id[b]) swap(a, b);
    //...
  } else {
    //...
  }
}
```

4.12 Centroid Decomposition

```
int sz[maxn]{};
bool ok[maxn]{};
int get_subtree_size(int u, int lst) {
  sz[u] = 1;
  for (int v: g[u]) {
    if (v == lst || ok[v]) continue;
    sz[u] += get_subtree_size(v, u);
  }
  return sz[u];
}
int get_centroid(int u, int lst, int tree_size) {
  for (int v: g[u]) {
    if (v == lst || ok[v]) continue;
    if (2 * sz[v] >= tree_size) {
      return get_centroid(v, u, tree_size);
    }
  }
  return u;
}
void centroid_decomp(int u = 1) { //1-based
  int centroid = get_centroid(u, u, get_subtree_size(u, u));
  //...
  ok[centroid] = 1;
  for (int v: g[centroid]) if (!ok[v]) {
    centroid_decomp(v);
  }
}
```

5 Math

5.1 fpow

```
ll fpow(ll b, ll p, ll mod) {
  ll res = 1;
  while (p) {
    if (p & 1) res = res * b % mod;
    b = b * b % mod, p >>= 1;
  }
  return res;
}
```

5.2 extgcd

```
#include<bits/stdc++.h>
using namespace std;

int extgcd(int a,int b,int &x,int &y){//a*x +b*y = 1
    if(b==0){
        x = 1;
        y = 0;
        return a;  //到達遞歸邊界開始向上一層返回
    }
    int r = extgcd(b,a%b,x,y);
    int temp=y;    //把x y變成上一層的
    y = x - (a / b) * y;
    x = temp;
    return r;      //得到a b的最大公因數
}
int main(){
    int a = 55,b = 80;
    int x,y;//a*x+b*y = 1;
    int GCD = extgcd(a,b,x,y);
}
```

5.3 EulerTotientFunction

```
ll phi[maxn];
for (int i = 0; i < maxn; ++i) {
  phi[i] = i;
}
for (int i = 2; i < maxn; ++i) if (phi[i] == i) {
  phi[i] = i - 1; //prime
  for (int j = 2; i*j < maxn; ++j) { //overflow
    phi[i*j] = (phi[i*j] / i) * (i - 1);
  }
}
```

5.4 FFT

```
//OI Wiki
#include <complex>
using cd = complex<double>;
const double PI = acos(-1);
void change(vector<cd> &y) {
  vector<int> rev(y.size());
  for (int i = 0; i < y.size(); ++i) {
    rev[i] = rev[i >> 1] >> 1;
    if (i & 1) {
      rev[i] |= y.size() >> 1;
    }
  }
  for (int i = 0; i < y.size(); ++i) {
    if (i < rev[i]) {
      swap(y[i], y[rev[i]]);
    }
  }
}
void fft(vector<cd> &y, bool inv) {
  change(y);
  for (int h = 2; h <= y.size(); h <<= 1) {
    cd wn(cos(2 * PI / h), sin(2 * PI / h));
    for (int j = 0; j < y.size(); j += h) {
      cd w(1, 0);
      for (int k = j; k < j + h / 2; ++k) {
        cd u = y[k];
        cd t = w * y[k + h / 2];
        y[k] = u + t;
        y[k + h / 2] = u - t;
        w = w * wn;
      }
    }
  }
  if (inv) {
    reverse(begin(y) + 1, end(y));
    for (int i = 0; i < y.size(); ++i) {
      y[i] /= y.size();
    }
  }
}
void solve() {
  int n;
  int m = 1 << (__lg(n) + 1); //power of 2
  vector<cd> a(m), b(m);
  //...
  fft(a, 0);
  fft(b, 0);
  vector<cd> c(m);
  for (int i = 0; i < m; ++i) {
    c[i] = a[i] * b[i];
  }
  fft(c, 1);
  for (auto p: c) {
    int ans = int(p.real() + 0.25);
  }
}
```

6 Misc

6.1 pbds

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template<typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;

int32_t main() {
  ordered_set<int64_t> rbt;
  // .insert(x); .erase(x)
  // .lower_bound(x); .upper_bound(x): iter
  // .find_by_order(k): find k-th small value(iter)
  // .order_of_key(x): return x is k-th big
  // .join(rbt2): merge with no mutiple same element
  // .split(key, rbt2): rbt keeps value <= key, others to rbt2
}
```

6.2 Misc

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
int randint(int lb, int ub) {
  return uniform_int_distribution<int>(lb, ub)(rng);
} //static unsigned x = 19; ++(x *= 0xdefaced);

#define SECs ((double)clock() / CLOCKS_PER_SEC)

struct KeyHasher {
  size_t operator()(const Key& k) const {
    return k.first + k.second * 100000;
} };
typedef unordered_map<Key,int,KeyHasher> map_t;

__lg
__gcd

__builtin_popcount  // 二進位有幾個1
__builtin_clz       // 左起第一個1之前0的個數
__builtin_parity    // 1的個數的奇偶性
```

6.3 Mo'sAlgorithm

```
struct Query {
  int L, R;
  //...
};
vector<Query> query;
void solve() { //K = n / sqrt(q)
  sort(iter(query), [&](Query &a, Query &b) {
    if (a.L / K != b.L / K) return a.L < b.L;
    return a.L / K % 2 ? a.R < b.R : a.R > b.R;
  });
  int L = 0, R = 0;
  for (auto x: query) {
    while (R < x.R) add(arr[++R]);
    while (L > x.L) add(arr[--L]);
    while (R > x.R) sub(arr[R--]);
    while (L < x.L) sub(arr[L++]);
    //...
  }
}
```

7 String

7.1 Hashing

```
const ll P = 401, M = 998244353;

ll hashes[10005], modp[10005];
ll hashp(string s, bool saveval) {
  ll val = 0;
  int index = 0;
  for (char c: s) {
    val = ((val * P) % M + c) % M;
    if (saveval) hashes[index++] = val;
  }
  return val;
}
void init(int base, int exp) {
  ll b = 1;
  modp[0] = 1;
  for (int i = 0; i < exp; i++) {
    b = (b * base) % M;
    modp[i + 1] = b;
  }
}
ll subseq(int l, int r) { //[l, r]
  if (l == 0) return hashes[r];
  return ((hashes[r] - hashes[l-1] * modp[r-l+1]) % M + M) % M;
}
```

7.2 Trie

```
struct node {
  int ch[26]{};
  int cnt = 0;
};
struct Trie {
  vector<node> t;
  void init() {
    t.clear();
    t.emplace_back(node());
  }
  void insert(string s) {
    int ptr = 0;
    for (char i: s) {
      if (!t[ptr].ch[i - 'a']) {
        t[ptr].ch[i - 'a'] = (int)t.size();
        t.emplace_back(node());
      }
      ptr = t[ptr].ch[i - 'a'];
    }
    t[ptr].cnt++;
  }
} trie;
```

7.3 Zvalue

```
vector<int> Zvalue(string &s) { //t + # + s
  vector<int> Z(s.size());
  int x = 0, y = 0;
  for (int i=0; i<s.size(); ++i) {
    Z[i] = max(0, min(y - i + 1, Z[i - x]));
    while (i + Z[i] < s.size() && s[Z[i]] == s[i + Z[i]])
      x = i, y = i + Z[i], ++Z[i];
  }
  return Z;
}
```

7.4 KMP

```
int F[maxn]{};
vector<int> match(string& s, string& t) {
  int p = F[0] = -1;
  for (int i = 1; i < t.size(); ++i) {
    while (p != -1 && t[p + 1] != t[i]) p = F[p];
    if (t[p + 1] == t[i]) ++p;
    F[i] = p;
  }
  p = -1;
  vector<int> v;
  for (int i = 0; i < s.size(); ++i) {
    while (p != -1 && t[p + 1] != s[i]) p = F[p];
    if (t[p + 1] == s[i]) ++p;
    if (p == t.size() - 1) v.push_back(i - p), p = F[p];
  }
  return v; //0-based
}
```

7.5 Manacher

```
int z[maxn * 2]{};
int manacher(string& s) {
  string t = "#";
  for (char c: s) t += c, t += '#';
  int l = 0, r = 0, ans = 0; //l: mid, r: right
  for (int i = 1; i < t.size(); ++i) {
    z[i] = (r > i ? min(z[2 * l - i], r - i) : 1);
    while (i - z[i] >= 0 && i + z[i] < t.size()) {
      if (t[i - z[i]] == t[i + z[i]])
        ++z[i];
      else
        break;
    }
    if (i + z[i] > r) r = i + z[i], l = i;
  }
  for (int i = 1; i < t.size(); ++i) ans = max(ans, z[i] - 1);
  string res;
  for (int i = 1; i < t.size(); ++i) if (ans == z[i] - 1) {
    for (int j = i - ans + 1; j < i + ans; ++j) if (t[j] != '#') {
      res += t[j];
    }
    break;
  }
  return ans;
}
```

8 Tree

8.1 LCA

```
int n, logn,t=0;
vector<vector<int>> graph;
vector<vector<int>> ancestor;
vector<int> tin, tout;
void dfs(int x){
    tin[x] = t++;
  for(auto y:graph[x]){
        if(y!= ancestor[x][0]){
            ancestor[y][0] = x;
            dfs(y);
        }
    }
    tout[x] = t++;
}
bool is_ancestor(int x, int y){
  return tin[x] <= tin[y] && tout[x] >= tout[y];
}
void table(){
  for (int i=1; i<logn; i++)// 上兩輩祖先、上四輩祖先、上八輩祖先、……
    for (int x=0; x<n; ++x)
      ancestor[x][i] = ancestor[ancestor[x][i-1]][i-1];
}

int kth_ancestor(int x, int k){
  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。
    if (k & (1<<i))
      x = ancestor[x][i];
  return x;
}

void rooted_tree(int root){// build the tree with root at "root"
  ancestor[root][0] = root;
  dfs(root);
  table();
}
int LCA(int x,int y){
    if (is_ancestor(x, y)) return x;
  if (is_ancestor(y, x)) return y;
    for (int i=logn-1; i>=0; i--)
    if (!is_ancestor(ancestor[x][i], y))
      x = ancestor[x][i];
  return ancestor[x][0];
}
int main(){
    graph = {
        {1,2},
        {3},
        {5,6},
        {7},
        {},
        {},
        {},
        {8},
        {4},
    };
    n = 9;
    logn = ceil(log2(n));
    ancestor.resize(n,vector<int>(logn));
    tin.resize(n);
    tout.resize(n);

    rooted_tree(0);
    while(true){
        int a,b;
        cin >>a>>b;
        cout <<LCA(a,b)<<endl;;
    }
}
int main(){
    n = 9;
    logn = ceil(log2(n));
    ancestor.resize(n,vector<int>(logn));
```

```cpp
    tin.resize(n);
    tout.resize(n);
    rooted_tree(0);
    while(true){
        int a,b;
        cin >>a>>b;
        cout <<LCA(a,b)<<endl;;
    }
}
```

8.2 Diameter

```cpp
vector<vector<int>> graph;
int diameter = 0;
int dfs(int start, int parent){
    int h1 = 0, h2 = 0;
    for (auto child : graph[start]){
        if (child != parent){
            int h = dfs(child, start) + 1;
            if (h > h1){
                h2 = h1;
                h1 = h;
            }
            else if (h > h2){
                h2 = h;
            }
        }
    }
    diameter = max(diameter, h1 + h2);
    return h1;
}
// call diameter
int main(){
    dfs(0,-1);
    cout << diameter<<endl;
}
```

8.3 Radius

```cpp
// Perform DFS to find the farthest node and its distance from the given node
pair<int, int> dfs(int node, int distance, vector<bool> &visited, const vector<vector<int>> &adj_list){
    visited[node] = true;
    int max_distance = distance;
    int farthest_node = node;

    for (int neighbor : adj_list[node]){
        if (!visited[neighbor]){
            auto result = dfs(neighbor, distance + 1, visited, adj_list);
            if (result.first > max_distance){
                max_distance = result.first;
                farthest_node = result.second;
            }
        }
    }

    return make_pair(max_distance, farthest_node);
}

// Calculate the radius of the tree using DFS
int tree_radius(const vector<vector<int>> &adj_list){
    int num_nodes = adj_list.size();
    vector<bool> visited(num_nodes, false);

    // Find the farthest node from the root (node 0)
    auto farthest_result = dfs(0, 0, visited, adj_list);

    // Reset visited array
    fill(visited.begin(), visited.end(), false);

    // Calculate the distance from the farthest node
    int radius = dfs(farthest_result.second, 0, visited, adj_list).first;

    return radius;
}
int main() {
    vector<vector<int>> adj_list;
    int radius = tree_radius(adj_list);
    cout << "Tree radius: " << radius << endl;
    return 0;
}
```

9 Z_Original_Code/Data_Structure

9.1 dsu-class

```cpp
#include <bits/stdc++.h>

using namespace std;

class DSU{
    public:
    DSU(int n ){
        this->n = n;
        reset();
    }
    int n;
    vector<int> boss;
    vector<int> rank;
    vector<int> size;

    void reset(){
        this->boss.resize(n);
        this->rank.resize(n,0);
        this->size.resize(n,0);
        for(int i =0;i<n;i++){
            boss[i] = i;
        }
    }
    int find(int x){
        if(boss[x]!= x){
            boss[x] = find(boss[x]);
        }
        return boss[x];
    }
    int get_size(int x){
        return size[find(x)];
    }
    void merge(int x, int y){
        int a = find(x);
        int b = find(y);
        // if(a!=b){
        //     boss[a] = b;
        //     size[b] += size[a];
        // }
        if(a!=b){
            if(rank[a]<rank[b]){
                boss[a] = b;
                size[b] += size[a];
            }else if (rank[a]<rank[b]){
                boss[b] = a;
                size[a] += size[b];
            }else{
                boss[a] = b;
                size[b] += size[a];
                rank[b]++;
            }
        }
    }
    bool aresame(int a,int b){
        return find(a)==find(b);
    }
};
int main(){
    DSU dsu(10);

    dsu.merge(0, 1);
    dsu.merge(2, 3);
    dsu.merge(4, 5);
    dsu.merge(6, 7);

    cout << "Are 0 and 1 connected? " << (dsu.aresame(0, 1) ? "Yes" : "No") << endl;
    cout << "Are 2 and 3 connected? " << (dsu.aresame(2, 3) ? "Yes" : "No") << endl;
    cout << "Are 4 and 5 connected? " << (dsu.aresame(4, 5) ? "Yes" : "No") << endl;
    cout << "Are 6 and 7 connected? " << (dsu.aresame(6, 7) ? "Yes" : "No") << endl;
    cout << "Are 1 and 2 connected? " << (dsu.aresame(1, 2) ? "Yes" : "No") << endl;

    dsu.merge(1, 2);

    cout << "Are 0 and 2 connected? " << (dsu.aresame(0, 2) ? "Yes" : "No") << endl;
    cout << "Are 1 and 3 connected? " << (dsu.aresame(1, 3) ? "Yes" : "No") << endl;

    return 0;
}
```

9.2 monotonic-queue

```cpp
//ref:leetcode
#include<bits/stdc++.h>

using namespace std;

class Monotonic_queue{
private:
    deque<int> qu;
public:
    void push(int n){
        while(!qu.empty()&&qu.back()<n){
            qu.pop_back();
        }
        qu.push_back(n);
    }
    int max(){
        return qu.front();
    }
    int min(){
        return qu.back();
    }
    int  size(){
        return qu.size();
    }
    void pop(){
        qu.pop_front();
    }
};

vector<int> maxSlidingWindow(vector<int> nums, int k) {
    Monotonic_queue window;
    vector<int> res;
    for (int i = 0; i < nums.size(); i++) {
        if (i < k - 1) {
            window.push(nums[i]);
        } else {
            window.push(nums[i]);
            res.push_back(window.max());
            if(window.max() == nums[i-k+1]){
                window.pop();
            }

        }
    }
    return res;

}
int main(){
    vector<int> nums = {1,3,-1,-3,5,3,6,7};
    int k = 3;
    vector<int> res = maxSlidingWindow(nums,k);
    for (auto r:res)cout <<r <<" ";
}
```

9.3 BIT

```cpp
#include <bits/stdc++.h>
using namespace std;

class BIT{
public:
    vector<int> bit;
    int N;
    BIT(int n){
        this->N = n;
        this->bit.resize(n);
    }
    void update(int x,int d){
        while(x<=N){
            bit[x] +=d;
            x +=x&(-x);// lowest bit in x;
        }
    }
    int query(int x){
        int res = 0;
        while(x){
            res+= bit[x];
            x -= x& -x;
        }
        return res;
    }
};
// Driver program to test above functions
int main()
{
    vector<int> freq =  {0, 2, 1, 1, 3, 2, 3, 4, 5, 6, 7, 8, 9};
    int n = freq.size();
    BIT bit(n);
    for(int i = 1;i<n;i++){
        bit.update(i,freq[i]);
    }
    for(int i = 1;i<n;i++){
        cout << bit.query(i)<<" ";
    }cout << endl;
    for(int i = 1;i<n;i++){
        bit.update(i,-1);
    }
    for(int i = 1;i<n;i++){
        cout << bit.query(i)<<" ";
    }cout << endl;
}
```

9.4 segment-tree-simple-add

```cpp
#include <bits/stdc++.h>

using namespace std;
struct node{
    int left;
    int right;
    int value;
};
vector<node> segment_tree;
void build(int left,int right,int x ,vector<int> & nums){
    segment_tree[x].left = left;
    segment_tree[x].right = right;
    // cout <<left <<" "<<right<<" "<<x<<endl;
    if(left == right){ // here is leaf
        segment_tree[x].value = nums[left];
        return;
    }
    int mid = (left+right)/2;
    build(left,mid,x<<1,nums);
    build(mid+1,right,x<<1|1,nums);
    segment_tree[x].value = segment_tree[x<<1].value+ segment_tree[x<<1|1].value;
}
void modify(int position ,int x,int value){
    if(segment_tree[x].left == position && segment_tree[x].right ==position){ // here is leaf
        segment_tree[x].value = value;
        return;
    }
    int mid = (segment_tree[x].left+segment_tree[x].right)/2;

    if(position<=mid){
        modify(position,x<<1,value);
    }else{
        modify(position,x<<1|1,value);
    }
    segment_tree[x].value = segment_tree[x<<1].value+ segment_tree[x<<1|1].value;
}
int query(int i,int j,int x){
    // cout <<i <<" "<<j <<" "<<segment_tree[x].left<<" " << segment_tree[x].right<<endl;
    int res = 0;
    int left = segment_tree[x].left;
    int right = segment_tree[x].right;
    int mid = (left+right)/2;
    if(segment_tree[x].left==i && segment_tree[x].right ==j){

        return segment_tree[x].value;
    }
    if(i>mid)return query(i,j,x*2+1);
    if(mid>=j)return query(i,j,x*2);
    return query(i,mid,x*2)+ query(mid+1,j,x*2+1);
}
int main(){
    vector<int> nums = {1,10,5,148,78,2,56,231,5,64,65,32,1,8};
    int n  = nums.size();
    segment_tree.resize(n*4);
    build(0,n-1,1,nums);
    modify(5,1,100);
    // cout << "++++++++++\n";
    for(int i =0;i<n;i++){
        for(int j = i ;j<n;j++){
            cout << query(i,j,1)<<" ";
        }cout << endl;
    }
}
```

9.5 monotonic-stack

```cpp
/*
input: array A
ouput: array B
bi is the value aj such that j>i and aj>bi   (j)
ex:
A = [2,1,2,4,3]
B = [4,3,4,-1,-1
*/
#include<bits/stdc++.h>

using namespace std;

vector<int> monotonic_stack(vector<int> nums){
    int n = nums.size();
    vector<int> res(n);
    stack<int>  st;
    for(int i = n-1;i>=0;i--){
        while(!st.empty() && st.top()<=nums[i]){
            st.pop();
            // we want the value greater than nums[i], so we pop the value smaller and equal nums[i]
        }
        if(st.empty())res[i] = -1;
        else res[i] = st.top();
        st.push(nums[i]);
    }
    return res;
}

int main(){
    vector<int> res = monotonic_stack({2,1,2,4,3});
    for(auto r:res){
        cout << r<<" ";
    }
}
```

10 Z_Original_Code/Flow

10.1 dicnic

```cpp
#include <bits/stdc++.h>
#define maxn 2005
#define INF 0x3f3f3f3f
using namespace std;
struct MaxFlow{
    struct edge{
        int  to, cap, flow,rev;
        edge( int v, int c, int f,int r) :  to(v), cap(c), flow(f),rev(r) {}
      };
    vector<edge> G[maxn];
    int s,t,dis[maxn],cur[maxn],vis[maxn];
    void add_edge(int from,int to,int cap){
        G[from].push_back(edge(to,cap,0,G[to].size()));
        G[to].push_back(edge(from,0,0,G[from].size()-1));
    }
    bool bfs(){
        memset(dis, -1, sizeof(dis));
        queue<int> qu;
        qu.push(s);
        dis[s] = 0;
        while (!qu.empty()) {
            int from = qu.front();
            qu.pop();
            for (auto &e: G[from]) {
                if (dis[e.to]==-1 && e.cap != e.flow) {
                    dis[e.to] = dis[from] + 1;
                    qu.push(e.to);
                }
            }
        }
        return dis[t]!=-1;
    }
    int dfs(int from,int cap){
        if(from==t ||cap==0)return cap;
        for(int &i = cur[from];i<G[from].size();i++){
            edge &e = G[from][i];
            if(dis[e.to]==dis[from]+1 && e.flow!=e.cap){
                int df = dfs(e.to,min(e.cap-e.flow,cap));
                if(df){
                    e.flow+=df;
                    G[e.to][e.rev].flow-=df;
                    return df;
                }
            }
        }
        dis[from] = -1;
        return 0;
    }
    int Maxflow(int s,int t){
        this->s = s,this->t =t;
        int flow = 0;
        int df;
        while(bfs()){
            memset(cur,0,sizeof(cur));
            while(df = dfs(s,INF)){
                flow +=df;
            }
        }
        return flow;
    }
};
```

11 Z_Original_Code/Graph

11.1 planar

```cpp
#include <iostream>
#include <vector>
#include <unordered_set>

using namespace std;

class Graph {
public:
    int V;
    vector<vector<int>> adj;
    Graph(int vertices) : V(vertices), adj(vertices) {}
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
};

bool containsSubgraph(const Graph& graph, const vector<int>& subgraph) {
    unordered_set<int> subgraphVertices(subgraph.begin(), subgraph.end());
    for (int vertex : subgraphVertices) {
        for (int neighbor : graph.adj[vertex]) {
            if (subgraphVertices.count(neighbor) == 0) {
                bool found = true;
                for (int v : subgraph) {
                    if (v != vertex && v != neighbor) {
                        if (graph.adj[v].size() < 3) {
                            found = false;
                            break;
                        }
                    }
                }
                if (found)
                    return true;
            }
        }
    }
    return false;
}

bool isPlanar(const Graph& graph) {
    // Subgraphs isomorphic to K  and K ,
    vector<int> k5 = {0, 1, 2, 3, 4};       // Vertices of K
    vector<int> k33a = {0, 1, 2};           // Vertices of K , (part A)
    vector<int> k33b = {3, 4, 5};           // Vertices of K , (part B)

    if (containsSubgraph(graph, k5) || containsSubgraph(graph, k33a) || containsSubgraph(graph, k33b)) {
        return false; // The graph is non-planar
    }
    return true; // The graph is planar
}

int main() {
    int vertices, edges;
    cin >> vertices;
    cin >> edges;

    Graph graph(vertices);
    for (int i = 0; i < edges; ++i) {
        int u, v;
        cin >> u >> v;
        graph.addEdge(u, v);
    }
    if (isPlanar(graph)) {
        cout << "The graph is planar." << endl;
    } else {
        cout << "The graph is non-planar." << endl;
    }

    return 0;
}
```

11.2 Dijkstra

```cpp
#include <bits/stdc++.h>
using namespace std;

#define maxn 200005
vector<int> dis(maxn,-1);
vector<int> parent(maxn,-1);
vector<bool> vis(maxn,false);
vector<vector<pair<int,int>>> graph;
void dijsktra(int source){
    dis[source] =0 ;

    priority_queue<pair<int,int>,vector<pair<int,int>>, greater<pair<int,int>>> pq;
    pq.push({0,source});
    while(!pq.empty()){
        int from  = pq.top().second;
        pq.pop();
        // cout <<vis[from]<<endl;
        if(vis[from])continue;
```

```
            vis[from] = true;
            for(auto next : graph[from]){
                int to = next.second;
                int weight = next.first;
                // cout <<from<<' ' <<to<<' ' <<weight;
                if(dis[from]+weight< dis[to] || dis[to]==-1){
                    dis[to] = dis[from]+weight;
                    parent[to] = from;
                    pq.push({dis[from]+weight,to});
                }
            }
        }

}
int main(){
    graph = {
        {{4,1},{5,3}},
        {{3,3}},
        {{}},
        {{4,0},{2,1},{7,2}}
    };
    dijsktra(0);
    for(int i =0;i<4;i++){
        cout << dis[i]<<" ";
    }
    for(int i =0;i<4;i++){
        cout << parent[i]<<" ";
    }
}
```

11.3 Floyd_Warshall

```
#include <bits/stdc++.h>
using namespace std;

#define maxn 2005
vector<vector<int>> dis(maxn,vector<int>(maxn,9999999));
vector<vector<int>> mid(maxn,vector<int>(maxn,-1));
vector<vector<pair<int,int>>> graph;

void floyd_warshall(int n ){ // n is n nodes
  for(int i =0;i<n;i++){
        for(auto path:graph[i]){
            dis[i][path.second] = path.first;
        }
  }
  for (int i=0; i<n; i++)
    dis[i][i] = 0;
  for (int k=0; k<n; k++){
    for (int i=0; i<n; i++){
      for (int j=0; j<n; j++){
        if (dis[i][k] + dis[k][j] < dis[i][j] || dis[i][j]==-1){
          dis[i][j] = dis[i][k] + dis[k][j];
          mid[i][j] = k;  // 由i點走到j點經過了k點
        }
            }
        }
    }
}
void find_path(int s, int t){   // 印出最短路徑
  if (mid[s][t] == -1) return;  // F有中繼點就結束
  find_path(s, mid[s][t]);    // 前半段最短路徑
  cout << mid[s][t];       // 中繼點
  find_path(mid[s][t], t);    // 後半段最短路徑
}
int main(){
    graph = {
        {{4,1},{5,3}},
        {{3,3}},
        {{}},
        {{4,0},{2,1},{7,2}}
    };
    floyd_warshall(4);
    for(int i =0;i<4;i++){
        for(int j = 0 ; j <4;j++)
            cout << dis[i][j]<<" ";
        cout << endl;
    }
    find_path(0,2);
}
```

11.4 2_sat

```
#include <iostream>
#include <vector>
#include <stack>
#include <algorithm>

using namespace std;

class TwoSAT {
public:
    TwoSAT(int n) : n(n), graph(2 * n), visited(2 * n, false) {}

    void addClause(int a, int b) {// 0-base;
        a *=2;
        b *=2;
        // Add implications (~a => b) and (~b => a)
        graph[a ^ 1].push_back(b);
        graph[b ^ 1].push_back(a);
    }

    bool solve() {
        // Find SCCs and check for contradictions
        for (int i = 0; i < 2 * n; ++i) {
            if (!visited[i]) {
                dfs1(i);
            }
        }
        reverse(processingOrder.begin(), processingOrder.end());//topological sort
        for (int i = 0; i < 2 * n; ++i) {
            visited[i] = false;
        }
        for (int node : processingOrder) {
            if (!visited[node]) {
                scc.clear();
                dfs2(node);
                if (!checkSCCConsistency()) {
                    return false;
                }
            }
        }

        return true;
    }

private:
    int n;
    vector<vector<int>> graph;
    vector<bool> visited;
    vector<int> processingOrder;
    vector<int> scc;

    void dfs1(int node) {
        visited[node] = true;
        for (int neighbor : graph[node]) {
            if (!visited[neighbor]) {
                dfs1(neighbor);
            }
        }
        processingOrder.push_back(node);
    }

    void dfs2(int node) {
        visited[node] = true;
        scc.push_back(node);
        for (int neighbor : graph[node]) {
            if (!visited[neighbor]) {
                dfs2(neighbor);
            }
        }
    }

    bool checkSCCConsistency() {
        for (int node : scc) {
            if (find(scc.begin(), scc.end(), node ^ 1) != scc.end()) {
                return false; // Contradiction found in the same SCC
            }
        }
        return true;
    }
};

int main() {
    int n, m;
    cin >> n >> m; // Number of variables and clauses

    TwoSAT twoSat(n);

    for (int i = 0; i < m; ++i) {
        int a, b;
        cin >> a >> b;
        twoSat.addClause(a, b);
    }

    if (twoSat.solve()) {
        cout << "Satisfiable" << endl;
    } else {
        cout << "Unsatisfiable" << endl;
    }

    return 0;
}
```

11.5 bipartite_matching

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100;


struct Bipartite_matching{
    int mx[MAXN], my[MAXN], vy[MAXN]; //matchX, matchY, visitY
    vector<int> edge[MAXN]; //adjcent list;
    int x_cnt;
    bool dfs(int x){
        for(auto y: edge[x]){ //對 x 可以碰到的邊進行檢查
            if(vy[y] == 1) continue; //避免遞F error

            vy[y] = 1;
            if(my[y] == -1 || dfs(my[y])){ //分析 3
                mx[x] = y;
                my[y] = x;
                return true;
            }
        }
        return false; //分析 4
    }

    int bipartite_matching(){
        memset(mx, -1, sizeof(mx)); //分析 1,2
        memset(my, -1, sizeof(my));
        int ans = 0;
        for(int i = 0; i < x_cnt; i++){  //對每一個 x 節點進行 DFS(最大匹配)
            memset(vy, 0, sizeof(vy));
            if(dfs(i)) ans++;
        }
        return ans;
    }
    vector<vector<int>> get_match(){
        vector<vector<int>> res;
        for(int i =0 ;i<x_cnt;i++){
            if(mx[i]!=-1){
                res.push_back({i,mx[i]});
            }
        }
        return res;
    }
    void add_edge(int i,int j){
        edge[i].push_back(j);
    }
    void init(int x){
        x_cnt = x;
    }
};
int main(){
    /*
    0 3
    0 4
    1 3
    1 5
    2 3
    2 4
    2 5
    */
    Bipartite_matching bm;
    for(int i = 0;i<7;i++){
        int a , b;
        cin >>a>>b;
        bm.add_edge(a,b);
    }
    bm.init(3);
    cout << bm.bipartite_matching()<<endl;
    auto match = bm.get_match();
    for(auto t: match){
        cout << t[0]<<" "<<t[1]<<endl;
    }

}
```

11.6 tarjan-SCC

```
#include <bits/stdc++.h>
using namespace std;
const int n = 16;
vector<vector<int>> graph;
int visit[n], low[n],t = 0;
int st[n], top =0;
bool instack[n];
int contract[n]; // 每個點收縮到的點
vector<vector<int>> block;
void dfs(int x,int parent){
    // cout <<x<<endl;
    visit[x] = low[x] = ++t;
  st[top++] = x;
  instack[x] = true;
    for(auto to: graph[x]){
        if(!visit[to])
            dfs(to,x);

        if(instack[to])
            low[x] = min(low[x],low[to]);
    }
    if(visit[x]==low[x]){ //scc F最早拜訪的
        int j;
        block.push_back({});
        do{
            j = st[--top];
            instack[j] = false;
            block[block.size()-1].push_back(j);
            contract[j] =x;
        }while(j!=x);
    }
}
int main(){
    graph = {
        {1},
        {3,4,5},
        {6},
        {2},
        {7},
        {11,15},
        {2,3},
        {4,6,9},
        {},
        {},
        {},
        {15},
        {14},
        {13,5},
        {15},
        {10,12,13}
    };
    for(int i =0;i<n;i++){
        if (!visit[i])
      dfs(i, i);
    }
    for(auto t: block){
        for(auto x:t){
            cout << x <<" ";
        }cout <<endl;
    }
}
```

11.7 topological_sort

```

#include <bits/stdc++.h>
using namespace std;
vector<vector<int>> graph;
vector<int> visit(10,0);
vector<int> order;
int n;
bool cycle;   // 記FDFS的過程中是否偵測到環
void DFS(int i)
{
  if (visit[i] == 1) {cycle = true; return;}
  if (visit[i] == 2) return;
  visit[i] = 1;
  for(auto to :graph[i])
        DFS(to);
  visit[i] = 2;
    order.push_back(i);
}


int main() {
    graph = {
        {1, 2},
        {3},
        {3, 4},
        {4},
        {}
    };
    n = 5;
  cycle = false;
  for (int i=0; i<n; ++i){
    if (!visit[i])
      DFS(i);
    }
  if (cycle)
    cout << "圖上有環";
  else
    for (int i=n-1; i>=0; --i)
      cout << order[i];
}
```

12 Z_Original_Code/Math

12.1 extgcd

```

#include<bits/stdc++.h>

using namespace std;

int extgcd(int a,int b,int &x,int &y)//擴展歐幾里得算法
{
    if(b==0)
    {
        x = 1;
        y = 0;
        return a;  //到達遞歸邊界開始向上一層返回
    }
    int r = extgcd(b,a%b,x,y);
    int temp=y;    //把x y變成上一層的
    y = x - (a / b) * y;
    x = temp;
    return r;     //得到a b的最大公因數
}

int main(){
    int a = 55,b = 80;
    int x,y;
    int GCD = extgcd(a,b,x,y);
    cout << "GCD: "<<GCD<<endl;;
    cout <<x<<" "<<y<<endl;
    cout <<a*x+b*y<<endl;
}
```

13 Z_Original_Code/Tree

13.1 LCA

```
#include<bits/stdc++.h>
using namespace std;
int n;
int logn;
vector<vector<int>> graph;
vector<vector<int>>ancestor;
vector<int> tin,tout;
int t = 0;
void dfs(int x){
    tin[x] = t++;
  for(auto y:graph[x]){
        if(y!= ancestor[x][0]){
            ancestor[y][0] = x;
            dfs(y);
        }
    }
    tout[x] = t++;
}
bool is_ancestor(int x, int y){
  return tin[x] <= tin[y] && tout[x] >= tout[y];
}
void table(){
    // 上兩輩祖先、上四輩祖先、上八輩祖先、……
  for (int i=1; i<logn; i++)
    for (int x=0; x<n; ++x)
      ancestor[x][i] = ancestor[ancestor[x][i-1]][i-1];
}

int kth_ancestor(int x, int k){
  // k拆解成二進位位數，找到第k祖先。不斷上升逼近之。
  for (int i=0; i<logn; i++)
    if (k & (1<<i))
      x = ancestor[x][i];
  return x;
}

void rooted_tree(int root){
  ancestor[root][0] = root;
  dfs(root);
  table();
}
int LCA(int x,int y){
    if (is_ancestor(x, y)) return x;
  if (is_ancestor(y, x)) return y;
    for (int i=logn-1; i>=0; i--)
    if (!is_ancestor(ancestor[x][i], y))
      x = ancestor[x][i];
  return ancestor[x][0];
}
int main(){
    graph = {
        {1,2},
        {3},
        {5,6},
        {7},
        {},
        {},
        {},
        {8},
        {4},
    };
    n = 9;
    logn = ceil(log2(n));
    ancestor.resize(n,vector<int>(logn));
    tin.resize(n);
    tout.resize(n);

    rooted_tree(0);
    while(true){
        int a,b;
        cin >>a>>b;
        cout <<LCA(a,b)<<endl;;
    }
}
```

13.2 diameter

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

using namespace std;

vector<vector<int>> graph;
int diameter = 0;
int dfs(int start, int parent){
    int h1 = 0,h2 = 0;
    for(auto child: graph[start]){
        if(child!= parent){
            int h = dfs(child,start)+1;
            if(h>h1){
                h2= h1;
                h1 = h;
            }
            else if(h>h2){
                h2 = h;
            }
        }
    }
    diameter = max(diameter,h1+h2);
    return h1;
}

int main(){
    graph = {
        {1,3},
        {0},
        {3},
        {0,2,4},
        {3}
    };
    dfs(0,-1);
    cout << diameter<<endl;
}
```

13.3 radius

```
#include<bits/stdc++.h>
using namespace std;
// Perform DFS to find the farthest node and its distance from the given node
pair<int, int> dfs(int node, int distance, vector<bool>& visited, const vector<vector<int>>& adj_list) {
    visited[node] = true;
    int max_distance = distance;
    int farthest_node = node;

    for (int neighbor : adj_list[node]) {
        if (!visited[neighbor]) {
            auto result = dfs(neighbor, distance + 1, visited, adj_list);
            if (result.first > max_distance) {
                max_distance = result.first;
                farthest_node = result.second;
            }
        }
    }

    return make_pair(max_distance, farthest_node);
}

// Calculate the radius of the tree using DFS
int tree_radius(const vector<vector<int>>& adj_list) {
    int num_nodes = adj_list.size();
    vector<bool> visited(num_nodes, false);

    // Find the farthest node from the root (node 0)
    auto farthest_result = dfs(0, 0, visited, adj_list);

```

```cpp
    // Reset visited array
    fill(visited.begin(), visited.end(), false);

    // Calculate the distance from the farthest node
    int radius = dfs(farthest_result.second, 0, visited,
         adj_list).first;

    return radius;
}

int main() {
    vector<vector<int>> adj_list = {
        {1, 2},
        {0, 3, 4},
        {0, 5},
        {1},
        {1},
        {2}
    };

    int radius = tree_radius(adj_list);
    cout << "Tree radius: " << radius << endl;

    return 0;
}
```

13.4 bridge

```cpp
#include<bits/stdc++.h>
using namespace std;
const int n = 9;
vector<vector<int>> graph;
vector<int> visit(n,0);
vector<int> trace(n,0);
vector<vector<int>> bridge;
int t = 0;
void dfs(int x,int parent){
    visit[x] = ++t;
    trace[x] =x;// 最高祖先預設🄵自己
    for(auto to:graph[x]){
        if(visit[to]){ //back edge
            if(to!=parent  ){
                trace[x] = to;
            }
        }else{  //treeedge
            dfs(to,x);
            if (visit[trace[to]] < visit[trace[x]])
          trace[x] = trace[to];

        // 子樹回不到祖先暨自身。
        if (visit[trace[to]] > visit[x])
          bridge.push_back({x,to});
        }
    }
}
int main(){
    graph = {
        {1,2},
        {3},
        {5,6},
        {7},
        {},
        {},
        {},
        {8},
        {4},
    };
    for(int i =0;i<9;i++){
        if(!visit[i])
            dfs(i,-1);
    }
    for(auto x: bridge){
        cout << x[0]<<" "<< x[1]<<endl;
    }
}
```

13.5 Articulation_vertex

```cpp
#include<bits/stdc++.h>
using namespace std;
const int n = 9;
int t =0;
vector<int> disc(n,-1); // Discovery time
vector<int> low(n,-1); // Low time
vector<int> parent_array(n,-1); // Parent in DFS tree
vector<bool> visited(n,false);
vector<bool> is_articulation(n,false);
vector<vector<int>> graph;
void dfs_articulation(int node, int parent) {
    visited[node] = true;
    disc[node] = t;
    low[node] = t;
    t++;
    int children = 0;

    for (int neighbor : graph[node]) {
        if (!visited[neighbor]) {
            children++;
            parent_array[neighbor] = node;
            dfs_articulation(neighbor, node);
            low[node] = min(low[node], low[neighbor]);

            if (low[neighbor] >= disc[node] && parent != -1)
                 {
                is_articulation[node] = true;
            }
        } else if (neighbor != parent) {
            low[node] = min(low[node], disc[neighbor]);
        }
    }

    if (parent == -1 && children > 1) {
        is_articulation[node] = true;
    }
}
int main(){
    graph = {
        {1,2},
        {3},
        {5,6},
        {7},
        {},
        {},
        {},
        {8},
        {4},
    };
    for (int i = 0; i < n; ++i) {
        if (!visited[i]) {
            dfs_articulation(i, -1);
        }
    }
    cout << "Articulation Points: ";
    for (int i = 0; i < n; ++i) {
        if (is_articulation[i]) {
            cout << i << " ";
        }
    }
    cout << endl;
}
```

NYCU_Segmentree Codebook

Contents