# Pontificia Universidad Católica del Perú - FCI

XieXieLucas Notebook - Froz/Phibrain/Ands

November 9, 2017

## Contents



# 1 2SAT

```
//2-SAT
//Conditions from 0 to 2*number of nodes, i and i^1 are reciprocal
//That means, ~0 is 1, ~1 is 0, ~2 is 3, ~3 is 2, etc
//When adding an edge, make sure to fix values
//For example, node from a to b (a,b >= 1)
//aa = (a-1)*2, bb = (b-1)*2, then a has "aa" as true and aa^1 as false
//Same to b
//To return to the main state, divide by 2 and sum 1

struct TwoSAT{
   int n;
   vector< vi> g, adj;
   vi d, low, scc, ans, lev;
   vector<bool> stacked, ok;
   stack<int> s;
   int ticks, current_scc;

   TwoSAT(int N):
      n(N),ticks(0), current_scc(0), g(N), adj(N), d(N), low(N), scc(N),
          ans(N), lev(N),
      stacked(N), ok(N){}

   void initialize(){
      REP(i,0,n){
         stacked[i] = false;
         d[i] = -1;
         scc[i] = -1;
         ok[i] = false;
         current_scc = ticks = 0;
      }
   }

   void addEdge(int a, int b){
      g[a].pb(b);
   }

   void tarjan(int u){
     d[u] = low[u] = ticks++;
     s.push(u);
     stacked[u] = true;
     const vector<int> &out = g[u];
     for (int k=0, m=out.size(); k<m; ++k){
       const int &v = out[k];
       if (d[v] == -1){
         tarjan(v);
         low[u] = min(low[u], low[v]);
       }else if (stacked[v]){
         low[u] = min(low[u], low[v]);
       }
     }
```

```
    if (d[u] == low[u]){
      int v;
      do{
        v = s.top();
        s.pop();
        stacked[v] = false;
        scc[v] = current_scc;
      }while (u != v);
      current_scc++;
    }
  }

  bool consistent(){
     for(int i = 0; i < n; i+=2){
        if(scc[i] == scc[i^1]){
           return false;
        }
     }
     return true;
  }

  void build(){
     REP(i,0,n){
        REP(j,0,sz(g[i])){
           int v = g[i][j];
           if(scc[i] != scc[v]){
              adj[i].pb(v); lev[v]++;
           }
        }
     }
  }

  void toposort(){
     queue<int> q;
     REP(i,0,current_scc){
        if(lev[i] == 0) q.push(i);
     }
     int x = 1;
     while(!q.empty()){
        int u = q.front(); q.pop();
        ans[u] = x ++;
        REP(i,0,sz(adj[u])){
           int v = adj[u][i];
           lev[v]--;
           if(lev[v] == 0) q.push(v);
           }
        }
     }

  void solve(){
     for(int i = 0; i<n; i+=2){
        if(ans[scc[i]] < ans[scc[i^1]]){
           ok[i] = false; ok[i^1] = true;
        }
        else{
           ok[i] = true; ok[i^1] = false;
        }
     }
  }

  bool go(){
     REP(i,0,n){
        if(scc[i] == -1) tarjan(i);
     }
     if(!consistent()) return false;
     else{
        build();
        toposort();
        solve();
        return true;
     }
  }

};

int main(){
  fastio;
  int n,m; cin >> n >> m;
  TwoSAT TS = TwoSAT(2*n);
  TS.initialize();

  //TO DO: ADD EDGES

  bool res = TS.go();
  if(!res) cout << "Impossible" << endl;
  else{
     for(int i = 0; i < 2*n; i+=2){
        int state = i/2 + 1;
        if(TS.ok[i]) //state is true
        else //state is false
```

```
    }
  return 0;
}
```

## 2 Biconnected Components

```
//Finds Biconnected Components

bool usd[1005];
int low[1005], d[1005], prev[1005], cnt;
vector <int> adj[1005];
stack <ii> S;

void Outcomp( int u , int v ){
	printf("New Component\n");
	ii e;
	do{
		e = S.top(); S.pop();
		cout << e.fst << " " << e.snd << endl;
	} while( e != mp( u , v ) );
}

void dfs( int u ){
	usd[u] = 1; cnt++;
	low[u] = d[u] = cnt;
	REP(i,0,sz(adj[u])){
		int v = adj[u][i];
		if( !usd[v] ){
			S.push( mp( u , v ) );
			prev[v] = u; dfs( v );
			if( low[v] >= d[u] ) Outcomp( u , v );
			low[u] = min( low[u] , low[v] );
		}
		else if( prev[u] != v and d[v] < d[u] ){
			S.push( mp( u , v ) );
			low[u] = min( low[u] , d[v] );
		}
	}
}

int main(){
	int n, m;
	cin >> n >> m;
	REP(i,0,m){
		int a , b;
		cin >> a >> b;
		adj[a].pb(b);
		adj[b].pb(a);
	}
	cnt = 0;
	memset(usd,0,sizeof(usd));
	memset(prev,-1,sizeof(prev));
	REP(i,0,n){
		if( !usd[i] ) dfs(i);
	}
	return 0;
}
```

## 3 Bridges and Articulation Points

```
//Finding bridges and articulation points

int low[N],id[N],parent[N];
bool art[N];
vi adj[N];
vi bridge[N];
int curr_id =0;
int root, rootchild;

void dfs(int u) {
	low[u] = id[u] = curr_id++;
	REP(j,0,sz(adj[u])) {
		int v = adj[u][j];
		if (id[v] == -1) {
			parent[v] = u;
			if (u == root) rootchild++;
			dfs(v);
			if (low[v] >= id[u]) art[u] = true;
			if (low[v] > id[u]){
				bridge[u].pb(v);
				bridge[v].pb(u); //store bridges in a sub
				    graph
			}
			low[u] = min(low[u], low[v]);
```

```
            }
            else if (v != parent[u]) low[u] = min(low[u], id[v]);
        }
}

//inside int main()
REP(i,0,n){
        if (id[i] == -1) {
                root = i; rootchild = 0; dfs(i);
                art[root] = (rootchild > 1);
        }
}
```

# 4 Eulerian Path

```
// Finds Eulerian Path (visits every edge exactly once)
// CYCLE exists iff all edges even degree, all edges in
// same connected component.
// PATH exists iff cycle exists and once edge removed
// [ Hamiltonian (all vertices) is NP complete ]
struct Edge;
typedef list<Edge>::iterator iter;
struct Edge
{
        int next_vertex;
        iter reverse_edge;
        Edge(int next_vertex) :next_vertex(next_vertex) { }
};
const int max_vertices = ;
int num_vertices;
list<Edge> adj[max_vertices];       // adjacency list
vector<int> path;

void find_path(int v)
{
        while(adj[v].size() > 0)
        {
                int vn = adj[v].front().next_vertex;
                adj[vn].erase(adj[v].front().reverse_edge);
                adj[v].pop_front();
                find_path(vn);
        }
        path.push_back(v);
}
void add_edge(int a, int b)
{
        adj[a].push_front(Edge(b));
        iter ita = adj[a].begin();
        adj[b].push_front(Edge(a));
        iter itb = adj[b].begin();
        ita->reverse_edge = itb;
        itb->reverse_edge = ita;
}
```

# 5 Maximal Cliques

```
// Bron-Kerbosch algorithm for finding all the
// maximal cliques of a graph in O(3^(n/3))
// 3 ^ 13 = 1.6e6

// Call them using clique(0, (1LL << n) - 1, 0)
// n vertexs
ll adj[65];
// This algorithm finds all the maximal cliques containing an edge
// The cliques are found explicitly (the vertex of the cliques)
void clique(ll r, ll p, ll x) {
   if (p == 0 && x == 0) {
      /* r is a maximal clique */
      /* Every 1 in r is a vertex of the clique
      Then, __builtin_popcountll(r) is the size of the clique*/
      return;
      }
   int pivot = -1;
   int menor = INF;
   for (int i = 0; i < n; i++) {
      if ( ((1LL << i) & p) || ((1LL << i) & x) ) {
         int x = __builtin_popcountll(p & (~(adj[i])));
         if (x < menor) {
            pivot = i;
            menor = x;
         }
      }
   }
   for (int i = 0; i < n; i++) {
```

```
        if ((1LL << i) & p) {
            if (pivot != -1 && adj[pivot] & (1LL << i)) continue;
            clique(r | (1LL << i), p & adj[i], x & adj[i]);
            p = p ^ (1LL << i);
            x = x | (1LL << i);
        }
    }
}

// This one has the same idea, but is faster
// However, it only finds the size of the cliques
void clique2(int r, ll p, ll x){
    if(p == 0 && x == 0){
        // r is the size of the clique
    }
    if(p == 0) return;
    int u = __builtin__ctzll(p | x);
    ll c = p & ~ adj[u];
    while(c){
        int v = __builint_ctzll(c); //Number of trailing zeros
        clique(r + 1, p & adj[v], x & adj[v]);
        p ^= (1LL << v);
        x |= (1LL << v);
        c ^= (1LL << v);
    }
}
```

## 6 Tarjan Strongly Connected Components

```
/* Complexity: O(E + V)
 Tarjan's algorithm for finding strongly connected
components.
 *d[i] = Discovery time of node i. (Initialize to -1)
 *low[i] = Lowest discovery time reachable from node
 i. (Doesn't need to be initialized)
 *scc[i] = Strongly connected component of node i. (Doesn't
 need to be initialized)
 *s = Stack used by the algorithm (Initialize to an empty
 stack)
 *stacked[i] = True if i was pushed into s. (Initialize to
 false)
 *ticks = Clock used for discovery times (Initialize to 0)
 *current_scc = ID of the current_scc being discovered
 (Initialize to 0)
*/

//DON'T FORGET TO INITIALIZE d[MAXN] TO -1 !!!!
vector<int> g[MAXN];
int d[MAXN], low[MAXN], scc[MAXN];
bool stacked[MAXN];
stack<int> s;
int ticks, current_scc;
void tarjan(int u){
  d[u] = low[u] = ticks++;
  s.push(u);
  stacked[u] = true;
  const vector<int> &out = g[u];
  for (int k=0, m=out.size(); k<m; ++k){
    const int &v = out[k];
    if (d[v] == -1){
      tarjan(v);
      low[u] = min(low[u], low[v]);
    }else if (stacked[v]){
      low[u] = min(low[u], low[v]);
    }
  }
  if (d[u] == low[u]){
    int v;
    do{
      v = s.top();
      s.pop();
      stacked[v] = false;
      scc[v] = current_scc;
    }while (u != v);
    current_scc++;
  }
}
```