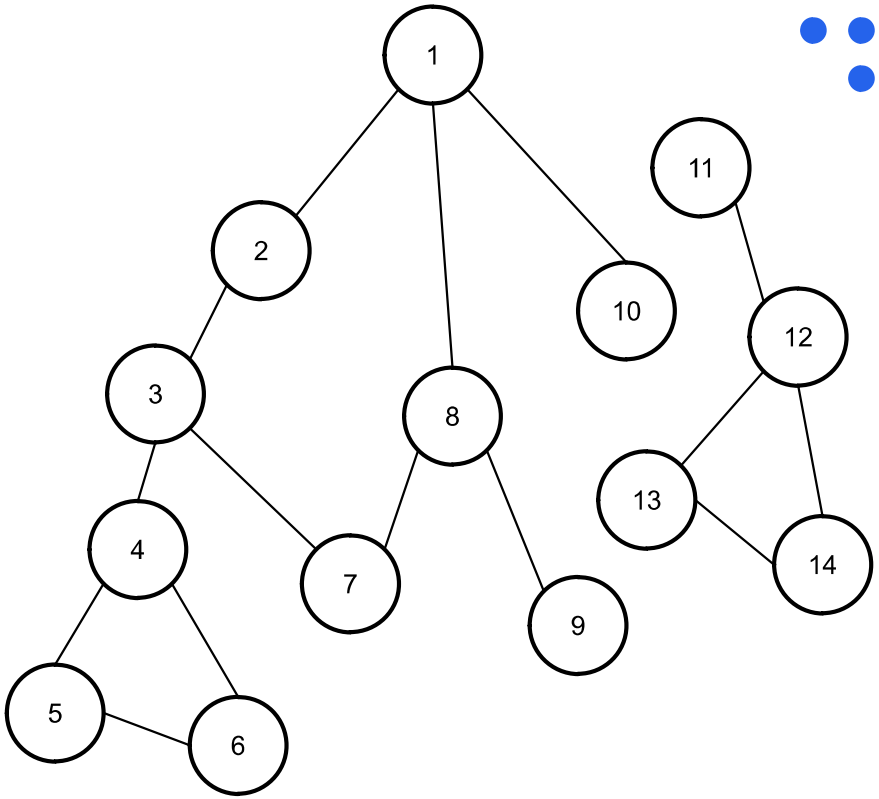
# Depth First Search

https://usaco.guide/silver/dfs

**CP Initiative**
joincpi.org

# Depth First Search

- Recursively traversing a graph.
- Connected component: a maximal set of connected nodes in an undirected graph.
- In other words, two nodes are in the same connected component if and only if they can reach each other via edges in the graph.
- A graph is connected if it is all part of the same connected component.



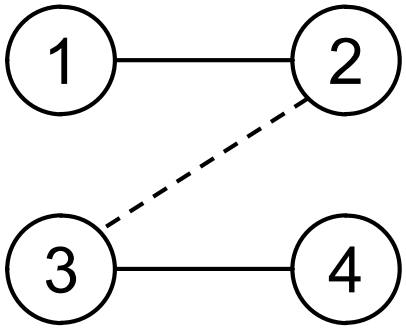
# DFS Example Problem

[CSES - Building Roads]

# DFS Example Solution

CSES - Building Roads

# Building Roads Solution Sketch




- Find connected components.
- The number of new roads needed is equal to n-1 where n is the number of connected components.
- Connect a road between each connected component.

# DFS Challenge Solution

```
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;
#define MAX_N 100000

int N, M;
typedef pair<int,int> pii;
vector<pii> C;
vector<int> nbrs[MAX_N];
int moonet[MAX_N];
struct BB { int x1, x2, y1, y2; };

// Reursively visit cow i in moonet k with bounding box bb
void visit(int i, int k, BB &bb)
{
  moonet[i] = k;
  bb.x1 = min(bb.x1, C[i].first);
  bb.x2 = max(bb.x2, C[i].first);
  bb.y1 = min(bb.y1, C[i].second);
  bb.y2 = max(bb.y2, C[i].second);
  for (int j : nbrs[i])
    if (moonet[j]==0) visit(j, k, bb);
}
```

# DFS Challenge Solution

```
int main(void)
{
  ifstream fin ("fenceplan.in");
  fin >> N >> M;
  pii p;
  for (int i=0; i<N; i++) {
    fin >> p.first >> p.second;
    C.push_back (p);
  }
  for (int i=0; i<M; i++) {
    fin >> p.first >> p.second;
    nbrs[p.first-1].push_back(p.second-1);
    nbrs[p.second-1].push_back(p.first-1);
  }
 int K = 0, best = 999999999;
 for (int i=0; i<N; i++)
    if (moonet[i]==0) {
      BB bb = {999999999,0,999999999,0};
      visit(i, ++K, bb);
      best = min(best, 2*(bb.x2-bb.x1+bb.y2-bb.y1));
    }
  ofstream fout ("fenceplan.out");
  fout << best << "\n";
  return 0;
}
```

# Graph Two-Coloring

**CP Initiative**
joincpi.org

# Graph Two-Coloring

- Traverses Graph similar to DFS.
- Assigns a “color” to each node.
- Bipartite Graph: Each edge connects two node of opposite colors.

# Graph Two-Coloring Example Problem

[CSES - Building Teams]

# Graph Two-Coloring Example Solution

CSES - Building Teams

# Building Teams Solution Sketch

- Arbitrarily label a node and then run DFS.
- Every time we visit a new (unvisited) node, we set its color based on the edge rule.
- When we visit a previously visited node, check to see whether its color matches the edge rule.

# Graph Two-Coloring: Challenge Problem

USACO: The Great Revegetation

# Graph Two-Coloring: Challenge Solution

```
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

int N, M, answer;
int L[100001];
vector<int> S_nbrs[100001], D_nbrs[100001];
bool impossible;

void visit(int x, int v)
{
  L[x] = v;
  for (auto n : S_nbrs[x]) {
    if (L[n] == 3-v) impossible = true;
    if (L[n] == 0) visit(n, v);
  }
  for (auto n : D_nbrs[x]) {
    if (L[n] == v) impossible = true;
    if (L[n] == 0) visit(n, 3-v);
  }
}
```

```
int main(void)
{
  ifstream fin ("revegetate.in");
  fin >> N >> M;
  for (int i=0; i<M; i++) {
    int a, b;
    string s;
    fin >> s >> a >> b;
    if (s=="S") { S_nbrs[a].push_back(b); S_nbrs[b].push_back(a); }
    if (s=="D") { D_nbrs[a].push_back(b); D_nbrs[b].push_back(a); }
  }

  for (int i=1; i<=N; i++)
    if (!L[i]) { visit(i,1); answer++; }

  ofstream fout ("revegetate.out");
  if (impossible) fout << "0\n";
  else {
    fout << "1";
    for (int i=0; i<answer; i++) fout << "0";
    fout << "\n";
  }
  return 0;
}
```