

Graph Traversals

Dr. Mattox Beckman

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
DEPARTMENT OF COMPUTER SCIENCE

Spring 2024

Objectives

- ▶ Implement DFS and BFS
- ▶ Show how to use these to solve some classic graph problems:
 - ▶ connected components
 - ▶ bipartite graph detection
 - ▶ topological sort
 - ▶ flood fill

Basics

- ▶ Step 1: Mark self as visited
- ▶ Step 2: Visit all unvisited children
- ▶ Step 3: ???
- ▶ Step 4: Profit!

Code

```
1  typedef pair<int, int> ii;
2  typedef vector<ii> vii; // edge is (neighbor, weight) pair
3  typedef vector<int> vi;
4
5  vi dfs_num;
6
7  void dfs(int u) {
8      // we mark the vertex as visited
9      dfs_num[u] = VISITED; // == 1, UNVISITED == -1
10     for (auto it = AdjList[u].begin();
11          it != AdjList[u].end(); ++it) {
12         if (dfs_num[it->first] == UNVISITED)
13             dfs(it->first);
14     } }
```

BSF Basics

- ▶ Mark self as visited
- ▶ Enqueue all unvisited children
- ▶ Dequeue next child and visit
- ▶ ???
- ▶ Profit!

BFS Code

```
1  vi d(V, INF); d[s] = 0; // initialize source distance
2  queue<int> q; q.push(s); // start from source
3  while (!q.empty()) {
4      int u = q.front(); q.pop();
5      for (int j = 0; j < (int)AdjList[u].size(); j++) {
6          int v = AdjList[u][j];
7          if (d[v.first] == INF) {
8              d[v.first] = d[u] + 1;
9              q.push(v.first);
10 } } }
```

Connected Components

```
1 numCC = 0;
2 dfs_num.assign(V, UNVISITED);
3 for (int i = 0; i < V; i++)
4     if (dfs_num[i] == UNVISITED) {
5         printf("CC %d:", ++numCC);
6         dfs(i);
7         printf("\n");
8     }
```

Flood Fill

```
1  int dr[] = {1,1,0,-1,-1,-1, 0, 1};
2  int dc[] = {0,1,1, 1, 0,-1,-1,-1};
3
4  int floodfill(int r, int c, char c1, char c2) {
5      if (r < 0 || r >= R || c < 0 || c >= C) return 0;
6      if (grid[r][c] != c1) return 0;
7      int ans = 1;
8      grid[r][c] = c2;
9      for (int d = 0; d < 8; d++)
10         ans += floodfill(r + dr[d], c + dc[d], c1, c2);
11     return ans;
12 }
```


Topological Sorting

```
1  vi ts; // the toposort vector
2
3  void toposort(int u) {
4      dfs_num[u] = VISITED;
5      for (int j = 0; j < (int)AdjList[u].size(); j++) {
6          ii v = AdjList[u][j];
7          if (dfs_num[v.first] == UNVISITED)
8              toposort(v.first);
9      }
10     ts.push_back(u);
11 }
```

Topological Sorting, ctd.

```
1  // in main
2
3  ts = vector(UNVISITED,dfs_num.size())
4  for (int i = 0; i < V; i++)
5      if (dfs_num[i] == UNVISITED)
6          dfs2(i);
```

Bipartite Graphs

```
1 groups = vector(-1,dfs_num.size())
2 bool checkBipartite(int u,int group) {
3     if (dfs_num[u] == VISITED) {
4         if (groups[u] != group)
5             return false;
6     } else {
7         dfs_num[u] = VISITED;
8         groups[u] = group;
9         for(auto it = AdjList[u].begin(); it != AdjList[u].end()
10            if (! checkBipartite(*it,2-group))
11                return false;
12     }
13     return true;
14 }
```