Graph Traversals

Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Fall 2023

Objectives

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

Basics

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

Code

```
typedef pair<int, int> ii;
typedef vector<ii> vii; // edge is (neighbor, weight) pair
typedef vector<int> vi;

vi dfs_num;

void dfs(int u) {
  // we mark the vertex as visited
  dfs_num[u] = VISITED; // == 1,  UNVISITED == -1
  for (auto it = AdjList[u].begin();
            it != AdjList[u].end(); ++it) {
    if (dfs_num[it->first] == UNVISITED)
       dfs(it->first);
} }
```

BSF Basics

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

BFS Code

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

Connected Components

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

Flood Fill

```
int dr[] = {1,1,0,-1,-1,-1, 0, 1};
int dc[] = {0,1,1, 1, 0,-1,-1,-1};

int floodfill(int r, int c, char c1, char c2) {
  if (r < 0 || r >= R || c < 0 || c >= C) return 0;
  if (grid[r][c] != c1) return 0;
  int ans = 1;
  grid[r][c] = c2;
  for (int d = 0; d < 8; d++)
      ans += floodfill(r + dr[d], c + dc[d], c1, c2);
  return ans;
}
```

Topological Sorting

```
vi ts;  // the toposort vector

void toposort(int u) {
    dfs_num[u] = VISITED;
    for (int j = 0; j < (int)AdjList[u].size(); j++) {
        ii v = AdjList[u][j];
        if (dfs_num[v.first] == UNVISITED)
            toposort(v.first);
    }
    ts.push_back(u);
}
```

Topologial Sorting, ctd.

```
// in main

ts = vector(UNVISITED,dfs_num.size())
for (int i = 0; i < V; i++)
    if (dfs_num[i] == UNVISITED)
       dfs2(i);
```

Bipartite Graphs

```
groups = vector(-1,dfs_num.size())
bool checkBipartite(int u,int group) {
  if (dfs_num[u] == VISITED) {
    if (groups[u] != group)
      return false;
  } else {
    dfs_num[u] = VISITED;
    groups[u] = group;
    for(auto it = AdjList[u].begin(); it != AdjList[u].end
      if (! checkBipartite(*it,2-group))
        return false;
  }
  return true;
}
```