UQCS Competitive Programming Group
Week 3
Graphs

Matthew Low

UQ Computing Society

Welcome to CPG!

UQCS **Competitive Programming Group**.

Algorithms, data structures and, most of all, problem solving.

We will try to be beginner-friendly, but basic programming knowledge is expected. COMP3506 (or any algorithms course) would also be helpful for tackling these problems.

Last week!

We learnt about data structures (stacks, trees and 2D arrays).

This week!

We'll be covering **three** (maybe four) problems dealing with **graphs** and searches in graphs (breadth-first search and depth-first search).

You are highly encouraged to try out more of these problems for yourself! Look in CP3 for more.

Want more specifics?

- COMP3506: Algorithms and Data Structures
- COMP4500: *Advanced* Algorithms and Data Structures

We'll just be doing problems here. If you haven't done either (or are doing them at the moment), don't worry.

Graphs

A graph data structure consists of a finite (and possibly mutable) set of vertices (also called nodes or points), together with a set of unordered pairs of these vertices for an undirected graph or a set of ordered pairs for a directed graph. These pairs are known as edges (also called links or lines), and for a directed graph are also known as arrows.[1]

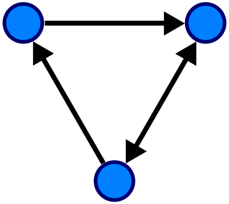

Figure 1: A graph.

[1]https://en.wikipedia.org/wiki/Graph_(abstract_data_type)

Breadth-first and depth-first search

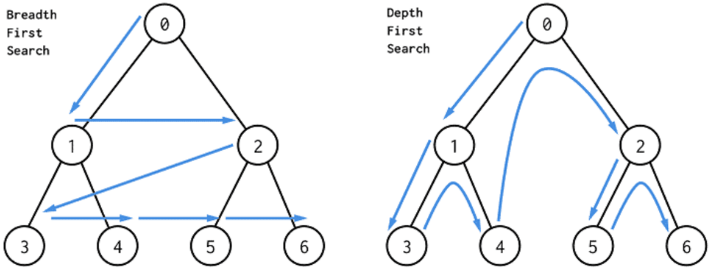

Figure 2:
`https://dev.to/danimal92/difference-between-depth-first-search-and-breadth-first-search-6om`

DFS in Python

```python
def dfs(graph, root):
    visited = set()
    stack = [root]
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            visited.add(vertex)
            stack.extend(graph[vertex] - visited)
    return visited
```

BFS in Python

```python
def bfs(graph, root):
    visited = set()
    queue = [root]
    while queue:
        vertex = queue.pop(0)
        print(vertex)
        for neighbour in graph[vertex]:
            if neighbour not in visited:
                visited.add(neighbour)
                queue.append(neighbour)
```

Example

```
graph = {0: set([1,2]),
         1: set([3,4]),
         2: set([5,6]),
         3: set(),
         4: set(),
         5: set(),
         6: set()}

print(bfs(graph, 0))
print(dfs(graph, 0))
```

Example

```
$ python bfs-dfs.py
0
1
2
3
4
5
6
None
0
2
6
5
1
4
3
None
```

Problem 1: Flood Fill

An image is represented by a 2-D array of integers, each integer representing the pixel value of the image (from 0 to 65535).

Given a coordinate `(sr, sc)` representing the starting pixel (row and column) of the flood fill, and a pixel value `newColor`, "flood fill" the image.

To perform a "flood fill", consider the starting pixel, plus any pixels connected 4-directionally to the starting pixel of the same color as the starting pixel, plus any pixels connected 4-directionally to those pixels (also with the same color as the starting pixel), and so on. Replace the color of all of the aforementioned pixels with the `newColor`.

At the end, return the modified image.

Problem 1: Flood Fill

```
Input:
image = [[1,1,1],[1,1,0],[1,0,1]]
sr = 1, sc = 1, newColor = 2
Output: [[2,2,2],[2,2,0],[2,0,1]]
Explanation:
From the center of the image
(with position (sr, sc) = (1, 1)),
all pixels connected by a path of the same color
as the starting pixel are colored with the new color.
Note the bottom corner is not colored 2,
because it is not 4-directionally connected
to the starting pixel.
```

Problem 2: Number of Islands

Given a 2d grid map of `'1'`s (land) and `'0'`s (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Problem 3: Course Schedule

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses-1`.

Some courses may have prerequisites, for example to take course `0` you have to first take course `1`, which is expressed as a pair: `[0,1]`

Given the total number of courses and a list of prerequisite pairs, is it possible for you to finish all courses?

Problem 4: Bus Routes

We have a list of bus routes. Each `routes[i]` is a bus route that the `i`-th bus repeats forever. For example if `routes[0] = [1, 5, 7]`, this means that the first bus (`0`-th indexed) travels in the sequence `1->5->7->1->5->7->1->...` forever.

We start at bus stop `S` (initially not on a bus), and we want to go to bus stop `T`. Travelling by buses only, what is the least number of buses we must take to reach our destination? Return `-1` if it is not possible.