02-613 Week 3

Algorithms and Advanced Data Structures

Aidan Jan

September 10, 2025

Graph Traversal Algorithms

Depth First Search (DFS)

- Start from the root node and go down as deep as possible into a branch, and record nodes along the way.
- If there are no more child nodes, backtrack the path taken, and go down a different branch.
- Since each node in the graph is visited exactly once, this is a O(n) operation.
- Typically implemented using recursion or a stack.

Theorem

If $(x, y) \in E$, either x is an ancestor of y or x is a descendant of y. Proof:

- Without loss of generality, x is an ancestor of y.
- All nodes between initially seeing x or leaving x are descendents of x.
- y must be explored before leaving x

Breadth First Search (BFS)

- Very similar to DFS but rather than going down all one branch all the way, it visits every child in order of level.
- Start from the root and record it plus every child. Repeat for each child in order.
- Also O(n), since each node in the graph is visited exactly once.
- Typically implemented using a queue.

Theorem

If $(x, y) \in E$, then $|\text{layer}(x) - \text{layer}(y)| \leq 1$. Proof:

- Without loss of generality, assume that layer(x) $<$ layer(y) - 1.
- All neighbors of x are added in or before layer$(x) + 1$

$$\text{layer}(y) \leq \text{layer}(x) + 1$$
$$\text{layer}(y) > \text{layer}(x) + 1$$

- The above is a contradiction.

- Basically, every time a node is visited, all its neighbors are added to the list to be iterated in the next level iteration.

The following is an example implementation of BFS.

```
TreeGrowing(graph G, node V, function findNext):
    T = ({v}, {})
    S = set of nodes incident to V
    while S != {}
        e = nextEdge(G, S)
        T += e
        S = updateFrontier(G, S, e)
```

Aside: Stacks and Queues

A queue is an array that views items in a First-in, First-out (FIFO) order. Basically elements added into the queue first will also be visited first.

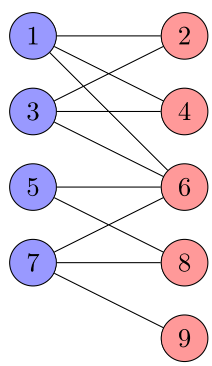
- `Dequeue()`: removes the first element of the queue, and moves the head pointer to the next element.
- `Enqueue(e)`: adds the element e to the end of the list.

A stack is an array that views items in a Last-in, First-out (LIFO) order. Basically elements added into the queue last will be visited first.

- `Pop(e)`: removes the last element of the list and decrements the tail pointer to the previous element.
- `Push(e)`: adds an element e to the end of the list.

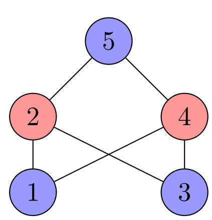
Bipartite Graphs

A **bibartite graph** is one with nodes in two sets such that there are no edges within a partition.



- $G = (U \cup V, E)$ such that $U \cap V = \emptyset$
- For a graph to be bipartite, it must have a two-coloring.
- It must also have no odd-length cycles.

Another example:



It is easy to tell that this graph is bipartite just by looking at it and coloring it in. However, what if the graph was bigger?

Breadth-First-Search Strategy

1. First, pick a random node, and BFS through the graph. Notice that every tree is bipartite since there are no cycles.

2. Now, insert in all the non-tree-edges. If there are no edges within the same level, then it is bipartite. (No "monolayer" edegs.)

```
def determineIfBipartite(G):
    T = BFS(G)
    for each layer:
        for each node pair u, v in layer:
            if (u, v) in G.E:
                return False
    return True
```

Proof of Correctness

For the purpose of contradiction, assume that there exists a $(u, v) \in E$ that is monolayer. Let Z be the common ancestor of u and v and let the path length from Z to u and v be l_i. In this case, edge (u, v) will create a cycle with length $2l_i + 1$. This length is always odd for any $l_i \in \mathbb{Z}$, and results in a contradiction since bipartite graphs cannot have odd-length cycles. This algorithm will have a time complexity of $\mathrm{O}(|V| + |E|)$ since it is limited by the BFS step.

Topological Sort

Given a DAG (directed, acyclic graph), find a bijective mapping f from v to $\{1, \ldots, |V|\}$ such that $\forall (u, v) f(u) < f(v)$.