1 Graph Traversals

Generally have three sets of nodes

- Nodes that have not yet been discovered
- Working Set nodes we are currently processing in some way
- Nodes that we have finished with

1.1 General view

Processing a node will generally mean looking at its neighbours and (generally) adding them to the working set

- The working set is stored in some data structure
- Need a policy to pick which node of the working set is next selected for processing: FIFO? LIFO? something else?
- Once selected, in some algorithms, the node might be moved to a data structure storing finished nodes
- Usually continue until the working set is empty

2 DFS vs BFS

2.1 DFS

The aim of DFS algorithm is to traverse the graph in such a way that it tries to **go far** from the root node. Stack is used in the implementation of the depth first search.
Steps

- Step 1: Push the root node in the Stack
- Step 2: Loop until stack is empty
- Step 3: Peek the node of the stack
- Step 4: If the node has unvisited child nodes, get the unvisited child node, mark it as traversed and push it on stack.
- Step 5: If the node does not have any unvisited child nodes, pop the node from the stack.

2.2 BFS

This is a very different approach for traversing the graph nodes. The aim of BFS algorithm is to traverse the graph as close as possible to the root node.
Steps

- Step 1: Push the root node in the Queue
- Step 2: Loop until the queue is empty
- Step 3: Remove the node from the Queue
- Step 4: If the removed node has unvisited child nodes, mark them as visited and insert the unvisited children in the queue.

2.3 Complexity

- Need a queue/stack of size $|V|$ (the number of vertices).
- Space complexity $O(V)$.

2.4 Space complexity in trees

If the graph has special properties then these complexities can be reduced. Example: suppose the graph is a tree, and we search from the root. In **DFS** the stack will be $O(height)$, this can be as good as $O(\log n)$. In **BFS** we still need to store **all nodes** of a level, hence is (generally) still $O(n)$. Hence in trees, DFS can be a lot more **space efficient** than BFS

2.5 Time complexity

- In terms of the number of vertices V: two nested loops over V, hence at worst $O(V^2)$.
- More useful complexity estimate is in terms of the number of edges.

2.6 Complexity of breadth-first search

- Assume an adjacency list representation, V is the number of vertices, E the number of edges.
- Each vertex is enqueued and dequeued at most **once**.
- Scanning for all adjacent vertices takes $O(|E|)$ time, since sum of lengths of adjacency lists is $|E|$.
- Gives a $O(|V| + |E|)$ time complexity.

2.7 Complexity of depth-first search

- Each vertex is pushed on the stack and popped at most once.
- For every vertex we check what the next unvisited neighbour is.
- In our implementation, we traverse the adjacency list only once. This gives $O(|V| + |E|)$ again.

Reference section

placeholder