11

June 16, 2016

1 Graph Algorithms

A *graph* is $G = (V, E)$. $m = |E|$, $n = |V|$, $n - 1 < m \leq n^2$.

Adjacency matrix: $A[u, v] = 1$ iff $(u, v) \in E$, else 0. $O(n^2)$ space.

Adjacency list: for each $u \in V$, store linked list $Adj[u] = v : (u, v) \in E$. $O(n + m)$ space.

Use DFS/BFS to find a path from s to t.

back edge: an edge that goes to a parent in a traversal.

forward edge: an edge that goes to a child (?).

cross edge: an edge that connects nodes on different branches.

No forward edges in BFS. For undirected graphs, forward edges same as back, no cross edges in DFS.

BFS and DFS: $\Theta(n + m)$ runtime.

Useful for:

1. Are 2 vertices connected?
2. Is undirected graph connected?
3. Does graph contain a cycle?
   - for undirected, run BFS/DFS on graph

BFS gives unweighted shortest path.

1.1 Bipartiteness/"2-colouring"

Given undirected graph G, determine whether it can be 2-coloured.

Solution: just DFS/BFS out from a point, colour all neighbouring point of the current point different from the current point. If we find a contradiction, then it is not possible. Else we would've coloured all the points.

Bipartite iff there is no odd-length cycle.

1.2 Topological Sort

Given *directed* graph, return a vertex order s.t. $\forall (u, v) \in E \implies u$ appears before v. This is always possible if we do not have a cycle (aka. a DAG).

Solution: DFS, then reverse discovery order, $O(m + n)$ time.

```
In [ ]:
```