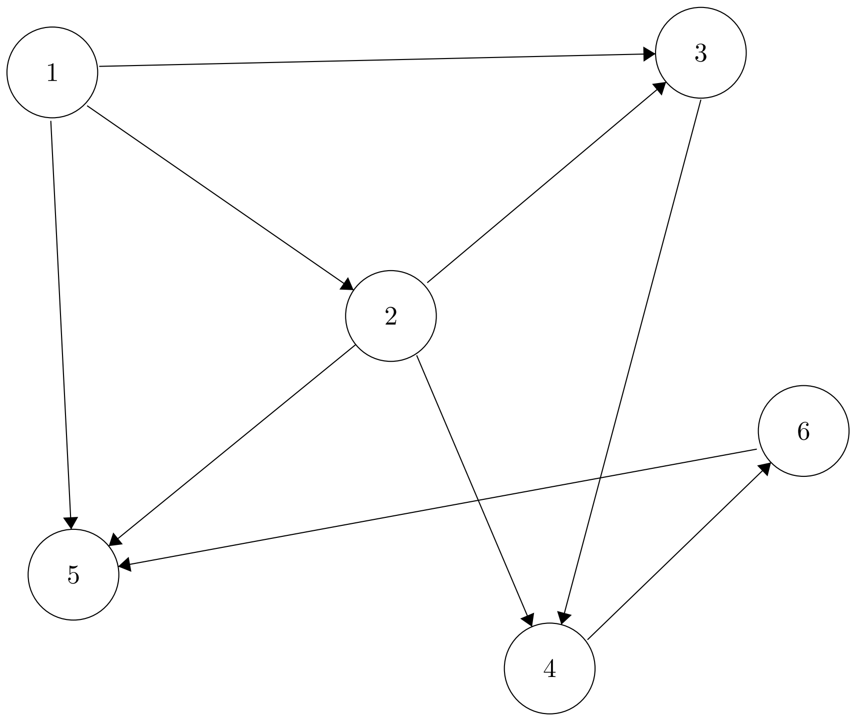
CS61B Spring 2016
Secret Section: Week 6

Tutor Team

April 11, 2016

1 Representations

Consider the graph below.



Construct the following representations for the graph:
• Adjacency Matrix

src dest	1	2	3	4	5	6
1	0	1	1	0	1	0
2	0	0	1	1	1	0
3	0	0	0	1	0	0
4	0	0	0	0	0	1
5	0	0	0	0	0	0
6	0	0	0	0	1	0

• List of Edges

(1,2), (1,3), (1,5), (2,3), (2,4), (2,5), (3,4), (4,6), (6,5)

• Adjacency List

$1 \to [2, 3, 5]$
$2 \to [3, 4, 5]$
$3 \to [4]$
$4 \to [6]$
$5 \to []$
$6 \to [5]$

What are the advantages and disadvantages of the different representations? Are some better than others?
• Adjacency matrices are pretty wasteful data structures. They use $\Theta(V^2)$ memory no matter what, which is the worst case memory usage for both the other representations. This is because the maximum number of edges in a graph is $\Theta(V^2)$. In addition, they take $\Theta(V)$ time to find adjacent vertices (pretty slow), which is a very commonly required operation.
• Lists of edges are the most memory efficient representations of graphs, but they suffer from performance issues, taking $\Theta(E)$ time to both check for the presence of an edge and to get the adjacent vertices of a vertex.
• Adjacency lists strike a balance between adjacency matrices and lists of edges. They have slightly larger memory requirements than a list of edges, but have much better performance, especially when getting adjacent vertices ($\Theta(1)$), which is very commonly required by graph algorithms. On the other hand, they have much lower memory requirements than adjacency matrices in the common case and the same in the worst case, while still outperforming them on the adjacency operation.

2 DFS

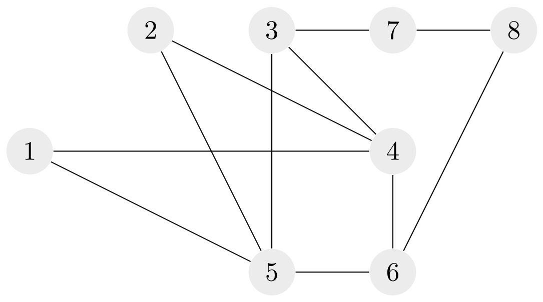
(a) Draw a directed graph whose DFS pre-order traversal is A, B, C, D, E, and whose DFS post-order traversal is C, B, D, E, A. Assume that ties are broken alphabetically.

$$\begin{tikzcd}
A \arrow[r] \arrow[d] \arrow[dr] & B \arrow[r] & C \\
D & E\ ?\ ? &
\end{tikzcd}$$

(b) What is the runtime complexity? How about space complexity?
Time: O(V+E)
Space: O(V)
?

?

3 BFS



? ? Let's say we want to find a path from node 1 to every other reachable node.

(a) In what order does BFS visit the nodes? Break ties numerically.
1, 4, 5, 2, 3, 6, 7, 8 ?

(b) Fill in the missing code:

```java
public class BreadthFirstPaths {
    private boolean[] marked;
    private int[] edgeTo;
    ...

    private void bfs(Graph G, int s) {
        Queue<Integer> fringe = new Queue<Integer>();
        fringe.enqueu(s);
        marked[s] = true;
        while (!fringe.isEmpty()) {
            int v = fringe.dequeu();
            for (int w : G.adj(v)) {
                if (!marked[w]) {
                    ___________________________
                    ___________________________
                    ___________________________
                }
            }
        }
    }
}
```

? fringe.enqueue(w);
marked[w] = true;
edgeTo[w] = v;
?

(c) What is the runtime complexity? How about space complexity?
Time: O(V+E)
Space: O(V)
?