

# Recitation 9

## Graph Search: BFS and DFS

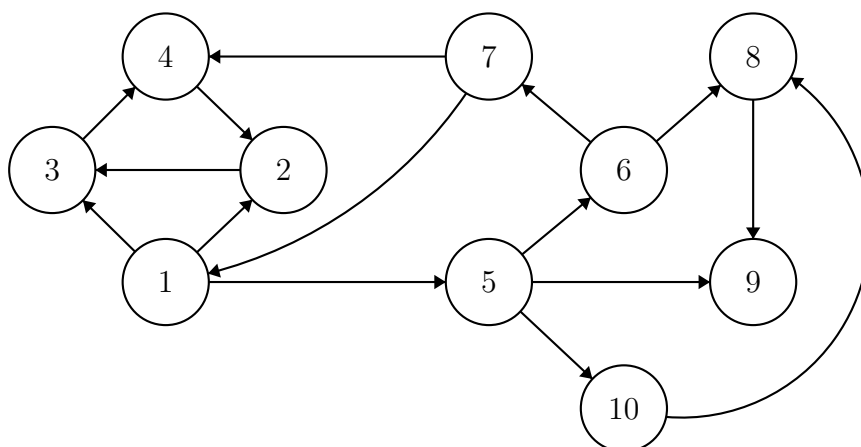
### 9.1 Announcements

- *BridgeLab* has been released, and is worth 140 points. The written section is due at Friday at 5pm, while the programming portion is due Sunday night.
- *ShortLab* will be released on **Friday**.
- Please fill out a midterm survey about the course so that we may improve. The link to the survey is on Piazza.

## 9.2 DFS Trees and Numberings

**Task 9.1.** Starting at vertex 1, execute DFS on the following graph, visiting vertices in increasing order. Trace the process by doing each of the following.

1. Draw the resulting DFS tree. Draw tree edges as solid lines, and include non tree edges in your drawing as dashed lines.
2. Classify each non tree edge as one of **forward**, **back**, or **cross**.
3. Label each vertex with its discovery and finish times.



**Task 9.2.** Suppose DFS is run on a directed graph, and consider some edge  $(x, y)$ . Using the discovery and finish times of  $x$  and  $y$ , attempt to classify this edge as one of tree, forward, back, or cross.

### 9.2.1 Higher-Order DFS

Recall the following code from the textbook:

Built: October 24, 2016

**Algorithm 9.3.** *Directed, generalized DFS.*

```

1  directedDFS (revisit, discover, finish) (G,  $\Sigma_0, s$ ) =
2  let
3    DFS p ((X,  $\Sigma$ ), v) =
4    if (v  $\in$  X) then (X, revisit ( $\Sigma, v, p$ )) else
5    let
6       $\Sigma'$  = discover ( $\Sigma, v, p$ )
7      X' = X  $\cup$  {v}
8      (X'',  $\Sigma''$ ) = iterate (DFS v) (X',  $\Sigma'$ ) ( $N_G^+(v)$ )
9       $\Sigma'''$  = finish ( $\Sigma', \Sigma'', v, p$ )
10   in
11     (X'',  $\Sigma'''$ )
12   end
13 in
14   DFS s (({ },  $\Sigma_0$ ), s)
15 end

```

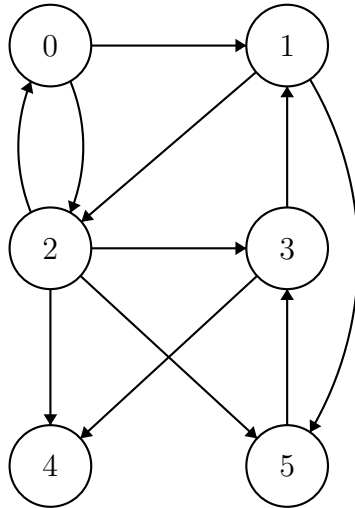
**Task 9.4.** Define  $\Sigma_0$ , revisit, discover, and finish to calculate DFS numberings.

**Task 9.5.** Modify the given generalized DFS code to work with undirected graphs.

(Hint: We only want to traverse each edge once! Try implementing undirected cycle detection with the above algorithm and see where it fails.)

## 9.3 BFS

### 9.3.1 An Example



**Task 9.6.** Run BFS on the example graph above, starting at vertex 1. Draw the resulting BFS tree. Draw tree edges as solid lines and non-tree edges as dashed lines.

### 9.3.2 Implementation

Consider the following code, which computes the BFS tree of an enumerated graph represented by an adjacency sequence. For brevity, we'll write NONE as  $\square$  and (SOME  $x$ ) as  $\boxed{x}$ .

**Algorithm 9.7.** *Computing BFS trees on adjacency sequences.*

```

1 fun BFS (G, s) =
2   let
3     fun BFS' (Xi, Fi) =
4       if |Fi| = 0 then STSeq.toSeq Xi else
5       let
6         val Ni =
7           Seq.flatten <<(u,  $\boxed{v}$ ) : u ∈ G[v] | Xi[u] =  $\square$ > : v ∈ Fi>
8         val Xi+1 = STSeq.inject (Xi, Ni)
9         val Fi+1 = <u : (u, v) ∈ Ni | Xi+1[u] =  $\boxed{v}$ >
10      in
11        BFS' (Xi+1, Fi+1)
12      end
13
14    val init = STSeq.fromSeq < $\square$  : 0 ≤ i < |G|>
15    val X0 = STSeq.update (init, (s,  $\boxed{s}$ ))
16    val F0 = <s>
17  in
18    BFS' (X0, F0)
19  end

```

**Task 9.8.** *Execute this code on the example graph given in the first section, starting with vertex 1 as the source. Trace the process by writing down the values  $X_i$ ,  $F_i$ , and  $N_i$  for  $i = 0, 1, 2, 3$ .*

**Task 9.9.** *Analyze the work and span of this implementation in terms of  $n$  (the number of vertices),  $m$  (the number of edges), and  $d$  (the diameter of the graph).*