Recitation 1

Graph Search: BFS and DFS

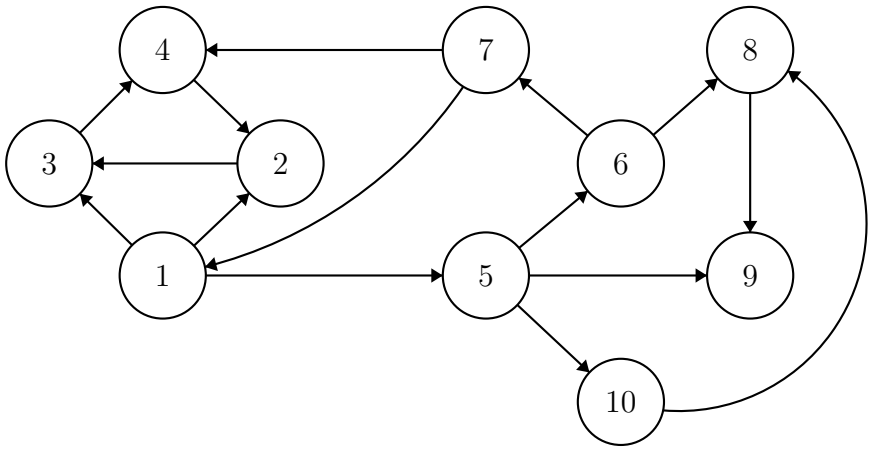
1.1 Announcements

- *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.

1.2 DFS Trees and Numberings

Task 1.1. *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 1.2. *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.

1.2.1 Higher-Order DFS

Recall the following code from the textbook:

Algorithm 1.3. *Directed, generalized DFS.*

```
directedDFS (revisit,discover,finish) (G,Σ0,s) =
  let
    DFS p ((X,Σ),v) =
      if (v ∈ X) then (X,revisit (Σ,v,p)) else
      let
        Σ' = discover (Σ,v,p)
        X' = X ∪ {v}
        (X'',Σ'') = iterate (DFS v) (X',Σ') (N_G^+(v))
        Σ''' = finish (Σ',Σ'',v,p)
      in
        (X'',Σ''')
      end
  in
    DFS s (({},Σ0),s)
  end
```

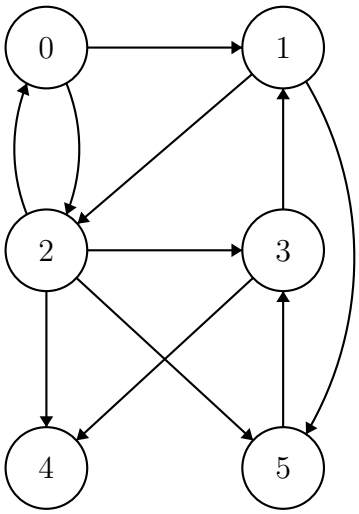
Task 1.4. *Define* Σ_0, `revisit`, `discover`, *and* `finish` *to calculate DFS numberings.*

Task 1.5. *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.)

1.3 BFS

1.3.1 An Example



Task 1.6. *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.*

1.3.2 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 $\boxed{\ }$ and (`SOME` x) as $\boxed{x}$.

Algorithm 1.7. *Computing BFS trees on adjacency sequences.*

```
1  fun BFS (G, s) =
2    let
3      fun BFS' (X_i, F_i) =
4        if |F_i| = 0 then STSeq.toSeq X_i else
5        let
6          val N_i =
7            Seq.flatten ⟨⟨(u, v) : u ∈ G[v] | X_i[u] = ⟩ : v ∈ F_i⟩
8          val X_{i+1} = STSeq.inject (X_i, N_i)
9          val F_{i+1} = ⟨u : (u, v) ∈ N_i | X_{i+1}[u] = v⟩
10       in
11         BFS' (X_{i+1}, F_{i+1})
12       end
13
14     val init = STSeq.fromSeq ⟨ : 0 ≤ i < |G|⟩
15     val X_0 = STSeq.update (init, (s, s))
16     val F_0 = ⟨s⟩
17   in
18     BFS' (X_0, F_0)
19   end
```

Task 1.8. *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 1.9. *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).*

.