Recitation 9

Graph Search: BFS and DFS

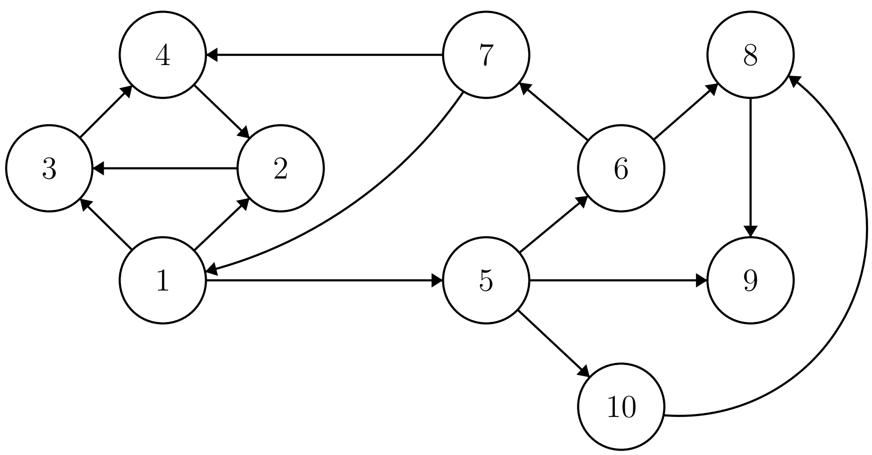
9.1 Announcements

- *BridgeLab* has been released, and is worth 140 points. The due dates are a bit wonky because of Spring Break: the written section is due at Friday at 5pm, while the programming portion is due Sunday night.
- *ShortLab* will be released on **Friday**.

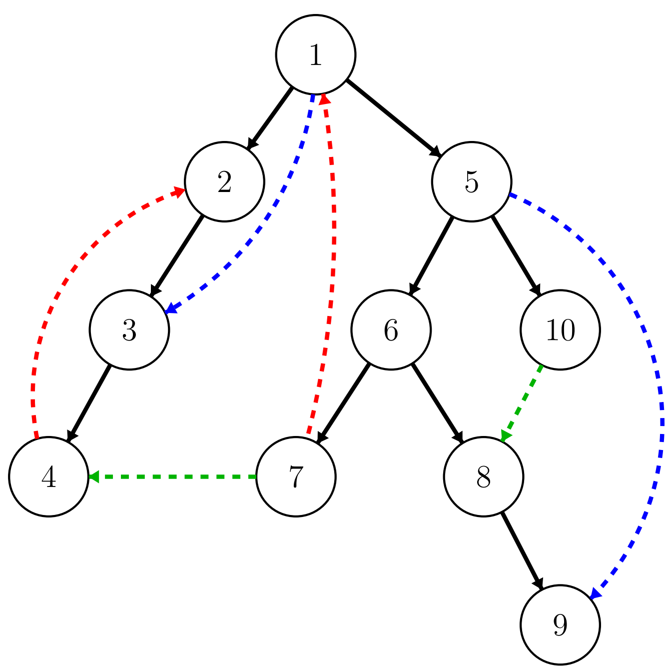
9.2 DFS Trees and Numberings

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.*



In the following diagram, back edges are red, forward edges are blue, and cross edges are green.



Vertex	Discovery	Finish
1	0	19
2	1	6
3	2	5
4	3	4
5	7	18
6	8	15
7	9	10
8	11	14
9	12	13
10	16	17

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.

Write $d[v]$ and $f[v]$ for the discovery and finish time of v, respectively.

Numbering	Possible Edge Type
$d[x] < d[y] < f[y] < f[x]$	tree, forward
$d[y] < d[x] < f[x] < f[y]$	back
$d[y] < f[y] < d[x] < f[x]$	cross

9.2.1 Higher-Order DFS

Recall the following code from the textbook:

Algorithm 9.3. *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 9.4. *Define Σ_0, `revisit`, `discover`, and `finish` to calculate DFS numberings.*

Algorithm 9.5. *Time-stamping with generalized directed DFS.*

```
Σ₀ = ({},{},0)
revisit (Σ,_,_) = Σ
discover ((D,F,c),v,_) = (D ∪ {v ↦ c},F,c+1)
finish (_,(D,F,c),v,_) = (D,F ∪ {v ↦ c},c+1)
```

Task 9.6. *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.)

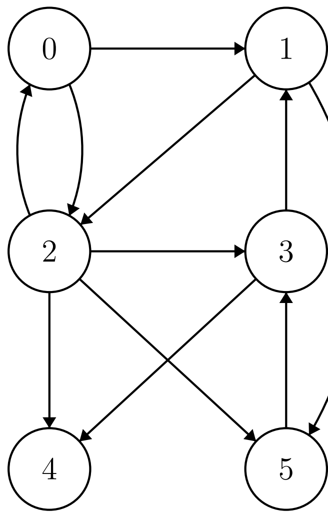
The problem with running the above code on an undirected graph is that every *every child will revisit its parent in the DFS tree, creating m back edges.* Hence, when attempting undirected cycle detection, every edge will be considered a cycle. We can fix this problem by omitting the parent from the neighbors of each child.

Algorithm 9.7. ***Undirected**, generalized DFS.*

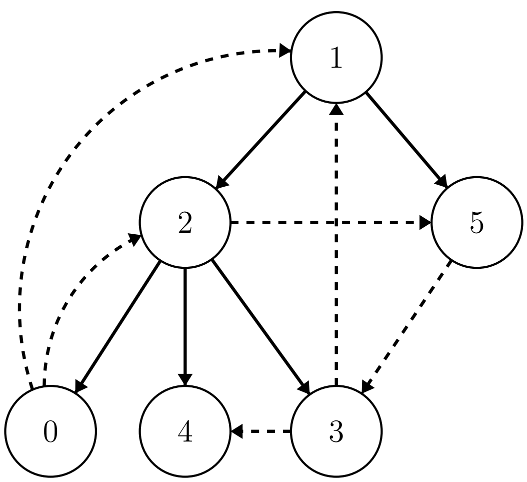
```
undirectedDFS (revisit, discover, finish) (G, Σ0, s) =
  let
    DFS p ((X, Σ), v) =
      if (v ∈ X) then (X, revisit (Σ, v, p)) else
      let
        Σ' = discover (Σ, v, p)
        X' = X ∪ {v}
        (X'', Σ'') = iterate (DFS v) (X', Σ') (N+_G(v) \ p)
        Σ''' = finish (Σ', Σ'', v, p)
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

9.3 BFS

9.3.1 An Example



Task 9.8. *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.*



Note that we could have chosen $(5, 3)$ as a tree edge instead of $(2, 3)$. Either edge is valid; as long as we don't choose *both* as tree edges, we're golden!

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

Algorithm 9.9. *Computing BFS trees on adjacency sequences.*

```
fun BFS (G, s) =
  let
    fun BFS' (X_i, F_i) =
      if |F_i| = 0 then STSeq.toSeq X_i else
      let
        val N_i =
          Seq.flatten ⟨⟨(u, [v]) : u ∈ G[v] | X_i[u] = [ ]⟩ : v ∈ F_i⟩
        val X_{i+1} = STSeq.inject (X_i, N_i)
        val F_{i+1} = ⟨u : (u, v) ∈ N_i | X_{i+1}[u] = [v]⟩
      in
        BFS' (X_{i+1}, F_{i+1})
      end

    val init = STSeq.fromSeq ⟨[ ] : 0 ≤ i < |G|⟩
    val X_0 = STSeq.update (init, (s, [s]))
    val F_0 = ⟨s⟩
  in
    BFS' (X_0, F_0)
  end
```

Task 9.10. *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 $i = 0, 1, 2, 3$.*

i	X_i	F_i	N_i
0	$\langle \boxed{}, \boxed{1}, \boxed{}, \boxed{}, \boxed{}, \boxed{} \rangle$	$\langle 1 \rangle$	$\langle (2, \boxed{1}), (5, \boxed{1}) \rangle$
1	$\langle \boxed{}, \boxed{1}, \boxed{1}, \boxed{}, \boxed{}, \boxed{1} \rangle$	$\langle 2, 5 \rangle$	$\langle (0, \boxed{2}), (4, \boxed{2}), (3, \boxed{2}), (3, \boxed{5}) \rangle$
2	$\langle \boxed{2}, \boxed{1}, \boxed{1}, \boxed{5}, \boxed{2}, \boxed{1} \rangle$	$\langle 0, 4, 3 \rangle$	$\langle\rangle$
3	$\langle \boxed{2}, \boxed{1}, \boxed{1}, \boxed{5}, \boxed{2}, \boxed{1} \rangle$	$\langle\rangle$	*(nonexistent)*

Task 9.11. *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).*

Let's break down the code, line-by-line. We write $||F|| = \sum_{v \in F}(1 + d_G^+(v))$.

- Line 7: $O(||F_i||)$ work, $O(\log n)$ span.
- Line 8: $O(||F_i||)$ work, $O(1)$ span.
- Line 9: $O(||F_i||)$ work, $O(\log n)$ span.
- Line 14: $O(n)$ work, $O(1)$ span.
- Lines 15,16: $O(1)$ work, $O(1)$ span.

There are two important observations to make here:

1. no vertex is ever in a frontier more than once, and
2. the number of rounds of BFS is upper bounded by $d + 1$. (There could be a vertex d hops away from the source, and each round progresses by exactly one hop. The "+1" comes from the final round which verifies that the frontier is empty, then exits).

We can now show that

$$\sum_{i=0}^{d} ||F_i|| \leq \sum_{v} (1 + d_G^+(v)) = n + m.$$

Therefore the total work is

$$O\left(n + \sum_{i=0}^{d-1} ||F_i||\right) = O(n + m)$$

and the span is $O(d \log n)$.

.