

Graph Algorithms Midterm Exam – 28/10/2025

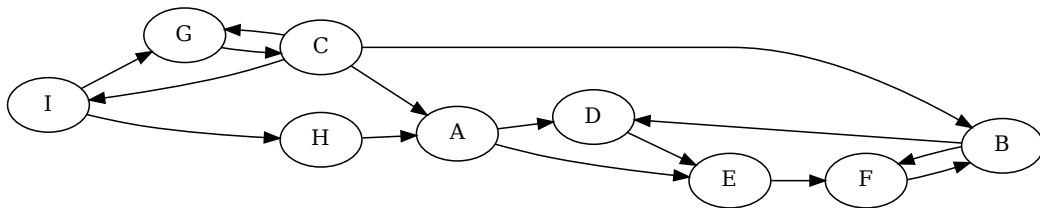
Guidelines

- You may use all algorithms and theorems we have seen in class (including the TDs) without proof. Other than that, you need to provide full justifications for all your answers.
- When asked to design an algorithm, you may do so in **pseudocode** or in **natural language**. Using a programming language (e.g. Python) is **unnecessary** and strongly discouraged.
- Provide proofs of correctness for your algorithms.
- Provide complexity analyses for your algorithms.
- Upper bounds on the running times of your algorithms must be given in terms of n (for adjacency matrix representation) or n, m (for adjacency list representation).
- **Algorithmic efficiency** is a major criterion in grading your answer. However, sub-optimal but correct algorithms may receive partial credit (depending on performance).
- An appendix with a reminder of some of the algorithms we have seen in class is given in the end.
- In your analysis, you may assume that arithmetic operations take $O(1)$ time, **except for exercise 5**, where you need to supply an upper bound on the cost of the arithmetic operations of your algorithm.
- **Write clearly!**
- **Don't panic!**
- **Good luck!**

1 DFS and SCC algorithm execution (5 points)

Consider the digraph given below, where we assume that all vertices and adjacency lists are ordered alphabetically. This exercise relies on the DFS algorithm, a reminder of which is given in the appendix.

1. Compute the output of DFS on this digraph. You should give a table with the discovery and finish times of all vertices. Furthermore, you should give a list of tree arcs, a list of backward arcs, a list of forward arcs, and a list of cross arcs. (3 points)
2. As you recall, executing DFS is the first step of the algorithm we saw in class for computing strongly connected components. Show how the execution of this SCC algorithm continues on the given digraph and the strongly connected components it outputs. (2 points)



2 Disconnecting by Removing (3 points)

For each of the following statements, indicate if it is true or false and give an appropriate justification (proof or counter-example).

1. If $G = (V, E)$ is an undirected connected graph, $v \in V$ a vertex, then $G - v$ has at most 2 connected components.
2. If $G = (V, E)$ is an undirected connected graph, $e \in E$ an edge, then $G - e$ has at most 2 connected components.
3. If $G = (V, A)$ is a strongly connected directed graph, $e \in A$ an arc, then $G - e$ has at most 2 strongly connected components.
4. If $G = (V, E)$ is an undirected graph and $e \in E$ an edge contained in a cycle, then e is not a bridge.
5. If $G = (V, E)$ is an undirected graph and $v \in V$ a vertex contained in a cycle, then v is not an articulation point.
6. If $G = (V, E)$ is an undirected graph, $v_1, v_2 \in V$ are articulation points, and v_1, v_2 are adjacent, then $v_1 v_2$ is a bridge.

In all cases, $G - v$ or $G - e$ indicates the graph obtained after removing v or e respectively from G . Recall that a bridge is an edge e such that $G - e$ has more components than G , and an articulation point is a vertex v such that $G - v$ has more components than G . (0.5 points per question)

3 2-Kings (4 points)

A 2-king in a directed graph $G = (V, A)$ is a vertex v such that for all other vertices u there is a path of length at most 2 from v to u . In other words, v 2-dominates a vertex u if $vu \in A$ or there exists w such that $vw, wu \in A$; and v is a 2-king if it 2-dominates every other vertex.

1. Give an efficient algorithm that finds a 2-king in a given digraph (or decides correctly that none exists), assuming the digraph is given in adjacency matrix form. (1 point)
2. Same as before, but assume that the digraph is given in adjacency list form. (1 point)
3. Suppose that we are given as input a digraph $G = (V, A)$ where for each $u, v \in V$ exactly one of the arcs uv or vu is in A (in adjacency matrix form). Give an algorithm for finding a 2-king in this case which is more efficient than your general algorithm. Prove that a 2-king always exists for this class of digraphs. (2 points)

4 Connectivity-Preserving Orientation (4 points)

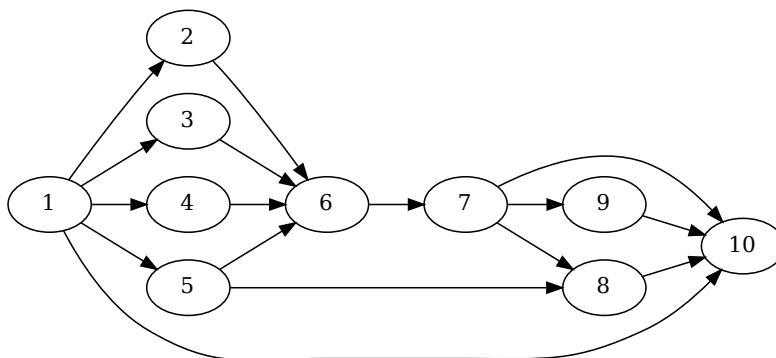
We are given a connected undirected graph $G = (V, E)$. An orientation of G is a digraph obtained by replacing every edge $uv \in E$ with either the arc uv or the arc vu (but not both). A connectivity-preserving orientation is an orientation with the property that the resulting digraph is strongly connected.

1. Give an example of a connected graph G on 5 vertices where no connectivity-preserving orientation exists. (1 point)
2. Give an algorithm that takes as input a connected undirected graph G in adjacency list form and outputs a connectivity-preserving orientation or correctly decides that none exists. (3 points)

5 Most critical arc (4 points)

We are given as input a DAG $G = (V, A)$, in adjacency list form, with the vertex set topologically ordered so that $V = \{1, \dots, n\}$. The **most critical arc** of G is defined as the arc $ij \in A$ such that removing ij from G **minimizes** the number of distinct paths from 1 to n .

1. What is the most critical arc of the graph below? Why? (1 point)
2. Design an efficient algorithm that takes as input a DAG G topologically ordered $\{1, \dots, n\}$ and outputs the most critical arc of G (or one of the most critical arcs, if there is a tie). (3 points)



A Algorithms

A.1 DFS

```

1: procedure DFS( $G$ )
2:   for  $v \in V \setminus \{s\}$  do                                     ▷ initialize
3:     Color  $v$  White, Parent of  $v \leftarrow \text{NULL}$ 
4:   end for
5:    $t = 0$                                                          ▷ universal time variable
6:   for  $v \in V$  do
7:     if  $v$  is White then
8:       DFS-Visit( $G, v$ )
9:     end if
10:  end for
11:  procedure DFS-VISIT( $G, u$ )
12:     $t++$ 
13:     $u.d = t$ , Color  $u$  Gray
14:    for  $v \in N(u)$  do
15:      if  $v$  is White then
16:        Set Parent of  $v$  to be  $u$ 
17:        DFS-Visit( $G, v$ )
18:      end if
19:    end for
20:     $t++$ 
21:     $u.f = t$ , Color  $u$  Black
22:  end procedure
23: end procedure

```

A.2 BFS

```

1: for  $v \in V \setminus \{s\}$  do                                     ▷ initialize
2:   Color  $v$  White, Parent of  $v \leftarrow \text{NULL}$ ,  $\text{dist}(s, v) \leftarrow \infty$ 
3: end for
4:  $\text{dist}(s, s) \leftarrow 0$ , Parent of  $s \leftarrow \text{NULL}$ , Color  $s$  Gray
5: Enqueue( $s$ )                                                     ▷ end of initialization
6: while Queue not empty do                                       ▷ main loop
7:    $v \leftarrow \text{Dequeue}$ 
8:   for each  $u \in N(v)$  do
9:     if  $u$  is White then
10:       $\text{dist}(s, u) \leftarrow \text{dist}(s, v) + 1$ ,  $v$  becomes parent of  $u$ 
11:      Color  $u$  Gray
12:      Enqueue( $u$ )
13:    end if
14:  end for
15:  Color  $v$  Black
16: end while

```