CS 341, Spring 2020
Semih Salihoglu

## Assignment 5 (due Wed, July 22nd, midnight EST)

**Instructions:**

- Hand in your assignment using Crowdmark. Detailed instructions are on the course website.
- Give complete legible solutions to all questions.
- Your answers will be marked for clarity as well as correctness.
- For any algorithm you present, you should justify its correctness (if it is not obvious) and analyze the complexity.

1. [*15 marks*] Let $G = (V, E)$ be a weighted, directed acyclic graph (DAG) with $n$ vertices and $m$ edges, where $m \geq n$. Let $s, t \in V$. We say that a path $v_1, v_2, ..., v_l$ is monotone if $w(v_1, v_2) \leq w(v_2, v_3) \leq \ldots \leq w(v_{l-1}, v_l)$. We want to compute a shortest monotone path from $s$ to $t$, i.e., a monotone path of the smallest total weight. Give an $O(mn)$-time algorithm using dynamic programming. Your algorithm should return the optimal path in addition to the optimal value. If no such path exists, you can just return "no".

**Algorithm 1:** ShortestMonotonePath($G(V, E), s, t, W$)

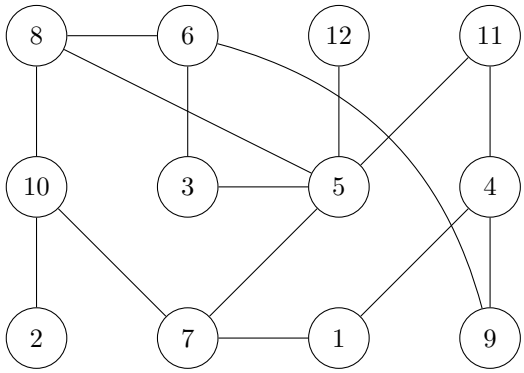
```
P[n, n − 1]
P[s, 0] = (0, −∞)
P[v ≠ s, 0] = (∞, −∞)
path = {t}
for i = 1, ..., n − 1 do
    for v ∈ V do
        P[v, i] = P[v, i − 1] for u ∈ inNbr(v) do
            if P[u, i − 1].second ≤ w(u, v) then
                P[v, i].first = min(P[v, i].first, P[u, i − 1].first + w(u, v))
                P[v, i].second = w(u, v)
                path = path ∩ {u}
if P[t][n − 1].thrid = ∅ then
    return NO
return (path, YES)
```

**Proof of Correctness:** Since $G$ is a DAG, it does not have negative weight cycles. So we can apply Bellman Ford DP algorithm to find a shortest path from $s$ to $t$.
Same as BF algorithm, For a path from $s$ to a vertex $v$ with length $i$, the shortest monotone path $p_{(v,i)}$ is the minimum value between $p_{(v,i-1)}$ and $p_{(u,i-1)+w(u,v)}$ for some $v$'s in-neighbour $u$

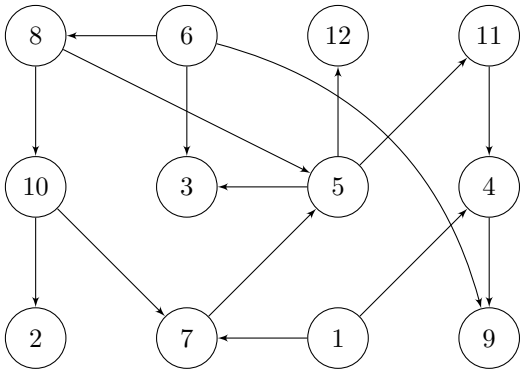
and $w(u, v)$ is greater than or equal to the weight of the edge that is incident to $u$. Meanwhile, the path form $s$ to $u$ is also monotone and the shortest.

**Runtime Analysis:** For each $i$, the runtime of computing each entry is $|in_neighbour(v)|$. Since we check all the vertices of $G$, the inner loop takes $O(m)$ times. Hence $T(n) = O(mn)$.

2. [*12 marks*]

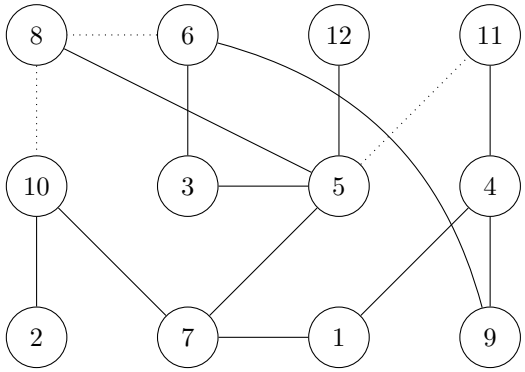



$G_1$




$G_2$

(a) [*2 marks*] Run the breadth-first search on the undirected graph $G_1$, starting from vertex 1, and show its final output. When you have to choose which vertex to process next (and that choice is not otherwise specified by the BFS algorithm), use the one with the smallest label. Show the BFS tree by indicating tree edges with solid lines and non-tree edges with dashed lines, and also trace the algorithm (i.e., indicate the action that is taken at every time step).




$G_1$

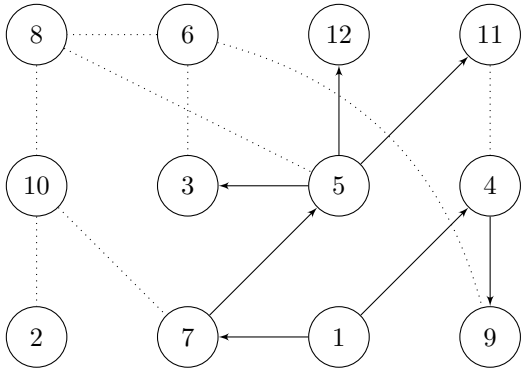
1. Add 1 to the queue. Remove 1 form the queue, mark 4, 7 as visited and add them to the queue. Mark 1 as finished. $\{4, 7\}$
2. Remove 4 form the queue, mark 9, 11 as visited and add them to the queue. Mark 4 finished. $\{7, 9, 11\}$
3. Remove 7 form the queue, mark 5, 10 as visited and add them to the queue. Mark 7 finished. $\{9, 11, 5, 10\}$
4. Remove 9 form the queue, mark 6 as visited and add it to the queue. Mark 9 finished. $\{11, 5, 10, 6\}$
5. Remove 11 form the queue. Mark 11 finished. $\{5, 10, 6\}$
6. Remove 5 form the queue, mark 3, 8, 12 as visited and add them to the queue. Mark 5 finished. $\{10, 6, 3, 8, 12\}$

7. Remove 10 form the queue, mark 2 as visited and add it to the queue. Mark 10 finished. $\{6, 3, 8, 12, 2\}$
8. Remove 6 form the queue. Mark 6 finished. $\{3, 8, 12, 2\}$
9. Remove 3 form the queue. Mark 3 finished. $\{8, 12, 2\}$
10. Remove 8 form the queue. Mark 8 finished. $\{12, 2\}$
11. Remove 12 form the queue. Mark 12 finished. $\{2\}$
12. Remove 2 form the queue. Mark 2 finished. $\{\}$

(b) [*2 marks*] Is $G_1$ bipartite? Justify your answer with reference to the BFS tree.

No. Since we can bipartite a tree by putting vertices in odd levels into one set and even levels into another, BFS tree is bipartite. But 5 and 11 are in the same level and $(5, 11) \in E(G)$. Hence G is not biparitite.

(c) [*2 marks*] Repeat part (a) on the directed graph $G_2$.




$G_2$

1. Add 1 to the queue. Remove 1 form the queue, mark 4, 7 as visited and add them to the queue. Mark 1 as finished.
2. Remove 4 form the queue, mark 9 as visited and add it to the queue. Mark 4 finished.
3. Remove 7 form the queue, mark 5 as visited and add it to the queue. Mark 7 finished.
4. Remove 9 form the queue. Mark 9 finished.
5. Remove 5 form the queue, mark 3, 11, 12 as visited and add them to the queue. Mark 5 finished.
6. Remove 3 form the queue. Mark 3 finished.
7. Remove 11 form the queue. Mark 11 finished.
8. Remove 12 form the queue. Mark 12 finished.

(d) [*2 marks*] Run the depth-first search algorithm on the undirected graph $G_1$, starting from vertex 1, and trace the algorithm (i.e., indicate the action that is taken at every time step). When you have to choose which vertex to process next (and that choice is not otherwise specified by the DFS algorithm), use the one with the smallest label. (You do not need to draw the DFS tree.)

1. Mark 1 as visited. Call DFS($G_1$, 4).
2. Mark 4 as visited. Call DFS($G_1$, 9).
3. Mark 9 as visited. Call DFS($G_1$, 6).
4. Mark 6 as visited. Call DFS($G_1$, 3).
5. Mark 3 as visited. Call DFS($G_1$, 5).
6. Mark 5 as visited. Call DFS($G_1$, 8).
7. Mark 8 as visited. Call DFS($G_1$, 10).
8. Mark 10 as visited. Call DFS($G_1$, 2).
9. Mark 2 as visited. Mark 2 as finished.
10. Call DFS($G_1$, 7).
11. Mark 7 as visited. Mark 7 as finished.
12. Mark 10 as finished.
13. Mark 8 as finished.
14. Call DFS($G_1$, 11).
15. Mark 11 as visited. Mark 11 as finished.
16. Call DFS($G_1$, 12).
17. Mark 12 as visited. Mark 12 as finished.
18. Mark 5 as finished.
19. Mark 3 as finished.
20. Mark 6 as finished.
21. Mark 9 as finished.
22. Mark 4 as finished.
23. Mark 1 as finished.

(e) [*2 marks*] Repeat (d) on the directed graph $G_2$.

1. Mark 1 as visited. Call DFS($G_1$, 4).
2. Mark 4 as visited. Call DFS($G_1$, 9).
3. Mark 9 as visited. Mark 9 as finished.
4. Mark 4 as finished.
5. Call DFS($G_1$, 7).
6. Mark 7 as visited. Call DFS($G_1$, 5).
7. Mark 5 as visited. Call DFS($G_1$, 3).
8. Mark 3 as visited. Mark 3 as finished.
9. Call DFS($G_1$, 11).
10. Mark 11 as visited. Mark 11 as finished.
11. Mark 12 as visited. Mark 12 as finished.
12. Mark 5 as finished.
13. Mark 7 as finished.
14. Mark 1 as finished.

(f) [*2 marks*] Is $G_2$ strongly connected? justify your answer with reference to a search tree (or trees).

No. Since the the FDS tree does not contain all the vertices in $G_2$, there is no path form 1 to 6. Hence it is not strongly connected.

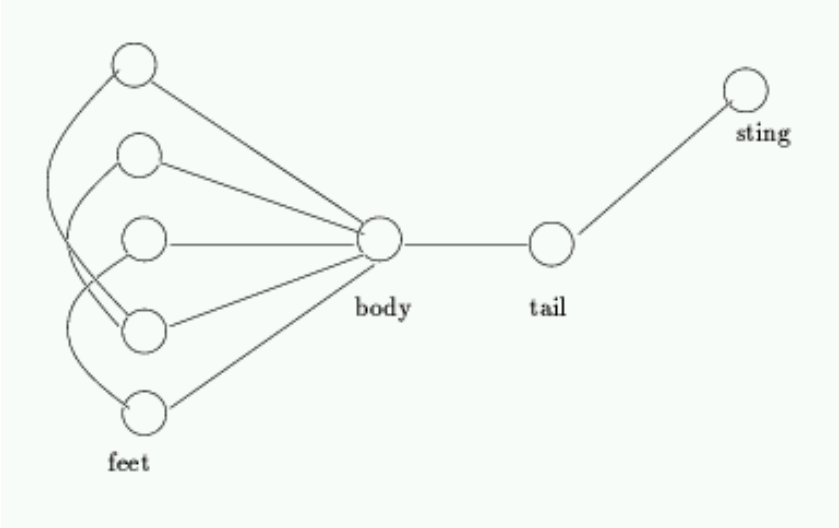



Figure 1: A scorpion graph

3. [*15 marks*] A scorpion graph is an undirected graph with $n$ nodes, where one node (the body) has degree $n-2$, and is connected to $n-3$ nodes (each called a feet) and to the tail. The tail has degree 2, and connects to sting, which has degree 1. A feet is not connected to either tail or sting, and may or may not be connected to other feet. See Figure 1. For parts (a)-(c) below, assume were given an undirected graph $G$ in an *adjacency matrix* representation (*not* an adjacency list).

   (a) [*3 marks*] Suppose we are given an undirected graph $G$ and node $s$ as a candidate sting. Give an O(n)-time procedure that returns the following: return YES if $s$ is indeed a sting and $G$ is a scorpion graph; return NO otherwise.
   Let $degree(v)$ reuturns the number of 1's in $G[v]$ for some $v \in V(G)$.

**Algorithm 2:** IsScorpionSting($G, s$)

```
1  if n < 3 then
2      return NO
3  if degree(s) ≠ 1 then
4      return NO
5  Let tail be s's only neighbour
6  if degree(tail) ≠ 2 then
7      return NO
8  Let body be tail's neighbour that is not s
9  if degree(body) ≠ n − 2 then
10     return NO
11 return YES
```

**Proof of Correctness:** We first check whether $degree(s) = 1$. If not $G$ is not scorpion. Otherwise, we check if $degree(tail) = 2$. If not just return NO. Since we choose *tail* from $s$'s neighbour set and $G$ is undirected. Hence $(tail, s) \in E(G)$. We then take the other neighbour of *tail* that is not $s$. We already know that *body* is not a neighbour of *string* since $s$'s only neighbour is *tail*. We also know that $(body, tail) \in E(G)$. We have checked the degree of *tail* and *sting*, so for all $feet$, $(feet, tail) \in E(G)$ and $(feet, sting) \in E(G)$. Thus we only need to check if $degree(body) = n - 2 = |V(G)| - 2$.

**Runtime Analysis:** Since *degree*() takes $O(n)$ time, $T(n) = 3O(n) + O(1) = O(n)$

(b) [*2 marks*] Suppose we are given an undirected graph $G$ and a node $b$ as a candidate body. Give an O(n)-time procedure that returns the following: return YES if $b$ is indeed a body and $G$ is a scorpion graph; return NO otherwise.

**Algorithm 3:** IsScorpionBody$(G, b)$

```
1 if degree(b) ≠ n − 2 then
2 |   return NO
3 Let sting be a vertex that is not neighbour of b
4 if degree(sting) ≠ 1 then
5 |   return NO
6 Let tail be sting's only neighbour
7 if degree(tail) ≠ 2 then
8 |   return NO
9 return YES
```

**Proof of Correctness:** If the number of neighbours of $b$ is not $n - 2$. Then $b$ is not a body. If $G$ is a scorpion, the only vertex that is not neighbour of $b$ should be *sting*. We then check if the candidate of *sting* is actually a *sting* by calculating the degree of the candidate. Lastly, we check the candidate of *tail*.

**Runtime Analysis:** Since *degree*() takes $O(n)$ time, $T(n) = 3O(n) + O(1) = O(n)$.

(c) [*10 marks*] Design an O(n)-time algorithm that decides if $G$ is a scorpion graph or not. [Hint: Think about how you can identify a single sting or body node in O(n) time.].

**Algorithm 4:** IsScorpion($G$)

```
randomly pick a vertex v
d = degree(v)
if d = n − 2 then
    return IsScorpionBody(G, v)
else if d = 1 then
    return IsScorpionSting(G, v)
else if d = 2 then
    Let u_1 and u_2 be the neighbours of v
    return IsScorpionBody(G, u_1) or IsScorpionBody(G, u_2)
else
    Let Nbr be the set of neighbours of v
    NonNbr = V(G)\Nbr
    i = 0; j = 0
    while i < |Nbr| and j < |NonNbr| do
        if G[Nbr[i]][NonNbr[j]] = 1 then
            j = j + 1
        else
            i = i + 1
    if j < |NonNbr| then
        return IsScorpionSting(G, NonNbr[j])
    return NO
```

**Proof of Correctness:** If $degree(v) = n-2, 1$, then we know $v$ is a candidate for being a body or sting respectively. We can just using the algorithm in part (a) or (b) to check if $G$ is a scorpion. If $degree(v) = 2$, $v$ is a foot or tail. Either a foot or tail is neighbour of body, we can call the algorithm in part (b) to check if $G$ is a scorpion.

If $v$ is a foot with $degree(v) \geq 2$. If there exists a vertex $u$ in the set $NonNbr = V(G)\backslash Nbr$ that is not adjacent to any vertices in $Nbr$, $u$ is a candidate for being a sting.If a vertex is a foot or tail form $NonNbr$, then at some iteration, $G[Nbr[i]][NonNbr[j]] = 1$. Since a foot or tail is adjacent to body which is in $Nbr$. Meanwhile, we increment $i$ or $j$ at each iteration, so the loop will always terminates. Hence we only need to check if $NonNbr[j]$ is a sting and $G$ is a scorpion.

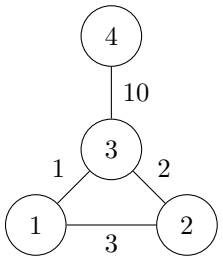
**Runtime Analysis:** By part (a) and (b) $IsScorpionBody()$ and $IsScorpionSting()$ takes $O(n)$ time. And $|Nbr| + |NonNbr| = n$, so while loop takes $O(n)$ time. $T(n) = 5O(n) = O(n)$

4. [*15 marks*] Let $G = (V, E)$ be a graph with $n$ vertices and positive edge weights $c_e > 0$ on the edges, where the weights on the edges are distinct. Let $T = (V, E')$ be a spanning tree of $G$. Let $e_h \in T$ be the edge with the maximum edge weight. We call $e_h$ the heaviest edge of $T$. Let a spanning tree of $G$ with the minimum heaviest edge be called the minimum heaviest-edge spanning tree (MHEST).

   (a) [*10 marks*] Prove that the MST of $G$ is also an MHEST of $G$.

   Assume for a contradiction that there exists a graph $G$ where the $MST$ of $G$ is not an $MHEST$ of $G$. Let edge $(x, y) \in V(MST)$ where $w((x, y))$ is maximum. Clearly, $(x, y) \notin V(MHEST)$. There is another path $p$ from $x$ to $y$ in $MHEST$. There exists an edge $e$ on $p$ in MHEST such that $e \notin E(MST)$, otherwise, there is a cycle in $MST$. Since $w((x, y))$ is greater than all edge weights in MHEST by assumption. Let $T = E(MST) - (x, y) + e$, $w(T) < w(MST)$ contradicting the definition of $MST$. Hence the MST of $G$ is also an MHEST of $G$.

   (b) [*5 marks*] Give a counterexample to the claim that an MHEST is always an MST.




   Let $E(MST) = \{(1, 3), (2, 3), (3, 4)\}$, $E(MHEST) = \{(1, 3), (1, 2), (3, 4)\}$.
   $w(MST) = 1 + 2 + 10 = 13$, but, $w(MHEST) = 1 + 3 + 10 = 14 < w(MST)$.