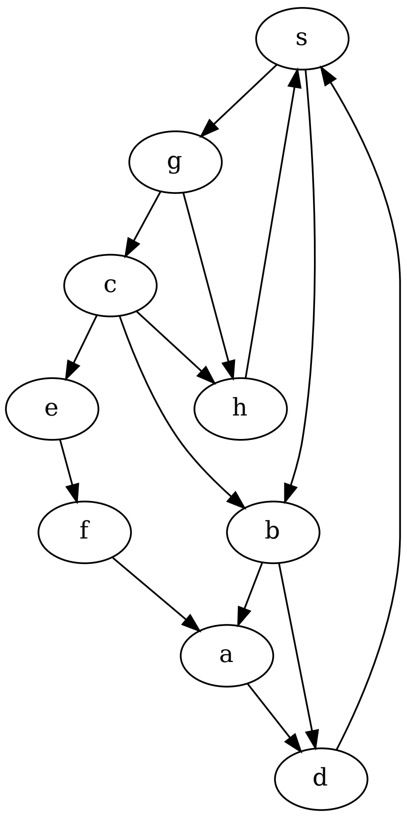
TD 3: DFS

1 Execute DFS

Execute the DFS algorithm on the directed graph below, starting from vertex s. You can assume that adjacency lists are ordered alphabetically. Show the discovery and finish times of all vertices, the tree calculated, and classify the arcs as tree, forward, backward, and cross.

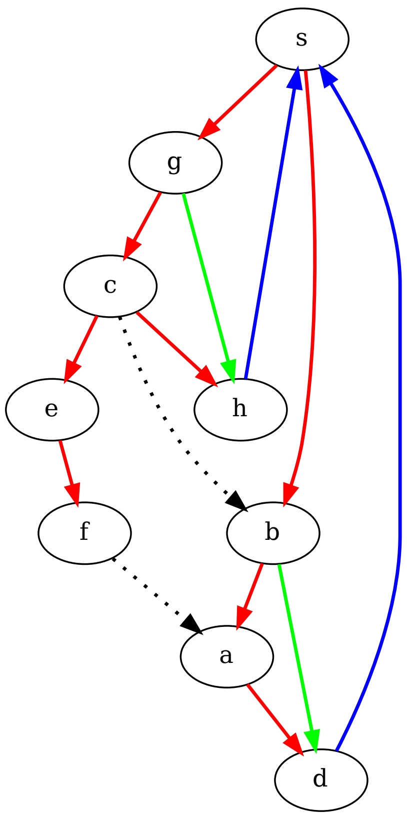


Times:

Vertex	Discovery	Finish
s	1	18
a	3	6
b	2	7
c	9	16
d	4	5
e	10	13
f	11	12
g	8	17
h	14	15

Solution:

DFS tree:



Red arcs are tree arcs, Blue arcs are backward, Green arcs are forward, Dotted arcs are cross.

2 Connected Components

Modify the DFS algorithm to identify the connected components of an undirected graph G. More precisely, your algorithm should take as input an undirected graph G and output an integer C equal to the number of connected components of G. Furthermore, it should compute for each $v \in V$ an attribute $v.cc$ such that for all $u, v \in V$ we have $u.cc = v.cc$ if and only if u, v are in the same connected component.

Solution:

```
1:  procedure CONNECTED-COMPS($G = (V, E)$)
2:      C←0
3:      for $v \in V$ do                                    ▷ initialize
4:          Color $v$ White, Parent of $v \leftarrow$ NULL
5:          v.cc←0
6:      end for
7:      $t = 0$                                             ▷ universal time variable
8:      for $v \in V$ do
9:          if $v$ is White then
10:             $C++$
11:             DFS-Visit($G$,$v$)
12:         end if
13:     end for
14:     procedure DFS-VISIT($G$,$u$)
15:         $t++$
```

```
16:         $u.d = t$, Color $u$ Gray
17:         $u.cc \leftarrow C$
18:         for $v \in N(u)$ do
19:             if $v$ is White then
20:                 Set Parent of $v$ to be $u$
21:                 DFS-Visit($G$,$v$)
22:             end if
23:         end for
24:         $t++$
25:         $u.f = t$, Color $u$ Black
26:     end procedure
27: end procedure
```

3 DAG Detection

A directed graph is called a DAG (Directed Acyclic Graph) if it contains no directed cycle as a subgraph. Show that we can decide if a given directed graph G is a DAG in linear time as follows: execute DFS on G and check whether a backward arc exists. Prove that G is a DAG if and only if no backward arc exists.
Solution:

For one direction, suppose we execute DFS and the output contains the backward arc uv. Since uv is a backward, v is an ancestor of u. However, in a DFS tree there is a directed path from any vertex to all of its descendants, therefore there is a path $v \to u$ which together with the arc uv creates a cycle in G.

For the converse direction, we want to show that if G contains a directed cycle C, then DFS will always produce a backward arc. Suppose we execute DFS and $v \in C$ is the first vertex to become Gray. Let u be the vertex **preceding** v in the cycle. By the White-Path theorem, u will become a descendant of v in the DFS tree, as all the vertices of C other than v are White when v became Gray, so there is a White path from v to u at that point in time. Since u precedes v in the cycle, G has the arc uv, and since v is an ancestor of u, this arc becomes a backward arc.

4 Unique Trees

Show that if $G = (V, E)$ is an undirected graph and $s \in V$, we have the following: if the trees produced by executing BFS and DFS on G, s are identical, then G contains no other edges except those of the tree output by the two algorithms.
Solution:

Let E_1 be the set of edges of G output by executing DFS starting from $s \in V$ and suppose that $E_1 \neq E$, therefore there exists $e \in E \setminus E_1$. Let $e = uv$ and without loss of generality suppose that DFS visited u before v ($d_u < d_v < f_v < f_u$).

By the white-path theorem, v is a descendant of u in the DFS/BFS tree. Since the edge e is not in the tree, the path from u to v in the DFS tree has length at least 2, that is u is not the immediate parent of v. However, this leads to a contradiction, because the tree is also a BFS tree and we have shown that in a BFS tree adjacent vertices must be found in the same level or in consecutive levels of the tree.

5 Digraph Random Walks

The objective of this exercise is to understand why the random walk reachability algorithm we saw in class only works for **undirected** graphs. Give an example of a digraph G with n vertices and two vertices s, t so that the probability that a random walk which starts at s manages to reach t is as small as possible (asymptotically, as a function of n). Can you make your example work even on digraphs where all vertices have large outdegree?

Explain why your example has (essentially) the lowest probability possible. Use your argumentation to derive a randomized algorithm (perhaps with exponential expected running time) that decides if a given digraph G has a directed path from s to t.

Solution:

Consider the following digraph G: we have n vertices $v_1, v_2, \dots, v_n$ and a directed path $v_1 \to v_2 \to \dots \to v_n$. We also have n vertices $u_1, \dots, u_n$, which form a bi-directed clique (that is, we have all possible arcs $u_i u_j$). Furthermore, for all i, j we have an arc $v_i u_j$. Let $s = v_1, t = v_n$.

We now observe that the probability that a random walk starting at v_1 reaches v_n is at most $\frac{1}{n+1}^{n-1}$. To see this, observe that each v_i has n arcs going to u_j vertices and only 1 arc leading closer to v_n. Furthermore, if the random walk ever reaches a u_j vertex, it can never reach v_n. Hence, the only way to reach v_n is to select the correct arc $n-1$ times, which happens with probability $\frac{1}{n+1}$ each time.

The probability above is essentially of the form $n^{-\Omega(n)}$ (on a graph with $2n$ vertices). We now observe that the probability that a random walk starting at s reaches t (if an $s \to t$ path exists) is always at least n^{-n}. To see this, observe that if such a path exists, it must have length at most $n-1$, and furthermore at each step we have probability at least $\frac{1}{n}$ of picking the arc of the path to proceed.

A randomized algorithm for directed reachability is now the following:

```
for $i = 1$ to $2n^n$ do
    Perform a random walk from $s$ for $n$ steps
    if $t$ was Reached then
        Return Yes
    end if
end for
Return No
```

Clearly, if no path exists the algorithm will return No. We therefore want to show that the algorithm will answer Yes with some non-trivial probability if a path exists. Suppose a path exists and then each iteration of the loop has probability at least $\frac{1}{n^n}$ of reaching t, therefore the probability that t was not reached in a single iteration is at most $(1 - \frac{1}{n^n})$. The probability that t was not reached in any of the $2n^n$ iterations is then at most $(1 - \frac{1}{n^n})^{2n^n}$. If we use the fact that $\lim_{x\to\infty}(1 - \frac{1}{x})^x = \frac{1}{e}$ the probability that t was not reached tends to at most $\frac{1}{e^2} < \frac{1}{4}$ (as n tends to infinity), so with at least $\frac{3}{4}$ probability we will reach t. Of course, this probability can be increased further by taking more iterations (say $3n^n$) of the for loop. The running time of our algorithm is, however, dominated by the number of iterations, which is exponential in $n \log n$.

6 Undirected random walk cover times

In class we claimed that in any n-vertex undirected graph, a random walk starting from any vertex will reach all other vertices in $O(n^3)$ steps in expectation. However, for some graphs it may actually be much faster to reach all vertices. Consider the following:

1. Show that if G is a clique K_n and s, t any two distinct vertices, if we start a random walk at s, the expected number of steps before we reach t is $\Theta(n)$.
2. Show that if G is a path P_n and s, t are its endpoints, if we start a random walk at s, the expected number of steps before we reach t is $\Theta(n^2)$.
3. Show that, despite the above, it is not correct to conclude that graphs with more edges make random walks faster. Construct a graph with n vertices, $\Omega(n^2)$ edges, such that the expected number of steps for a random walk starting at s to reach t is $\Omega(n^3)$.

Solution:

For the first case, let $h(v)$ be the expected number of steps for a walk starting at a vertex v to reach t. We have $h(t) = 0$ and observe that, by symmetry, $h(u) = h(v)$ for all other vertices $u, v \in V$. Therefore, $h(v) = 1 + h(v)\frac{n-2}{n-1}$, because with probability $\frac{1}{n-1}$ the walk will reach t in one step and with probability $\frac{n-2}{n-1}$

the walk will first go to another vertex, and have expected length $1 + h(v)$. Solving this, we get $h(v) = n - 1$, so $h(v) = \Theta(n)$.

For the second case, let the vertices of the path be $v_0, v_1, \dots, v_{n-1}$, where $s = v_0$ and $t = v_{n-1}$. Let $h(i)$ be the expected number of steps to reach t, if we start a walk at v_i, and we are looking for $h(0)$. We have that:

$$\begin{aligned} h(0) &= 1 + h(1) \\ h(n-1) &= 0 \\ h(i) &= 1 + \frac{h(i-1) + h(i+1)}{2} \end{aligned}$$

In the last equation we assume that $i \in \{1, \dots, n-2\}$. These equations follow because (i) if we are at v_0 the random walk will definitely proceed to v_1 (ii) if we are at v_{n-1} we are done (iii) in the general case, the random walk has probability $1/2$ of moving from i to $i-1$ or to $i+1$.

We now **guess** that $h(i) = ai^2 + bi + c$ and try to determine a, b, c which satisfy the above. In the last equation, b, c are simplified and we get $a = -1$. So, $h(i) = -i^2 + bi + c$. The first equation the gives $c = b + c$, therefore $b = 0$. Finally, the second equation gives $-(n-1)^2 + c = 0$ therefore $c = (n-1)^2$. We get, $h(i) = (n-1)^2 - i^2$. We can now check that this formula indeed satisfies the equations. So, $h(0) = (n-1)^2 = \Theta(n^2)$.

Finally, an example for the last question is the lollipop graph, constructed by taking the two previous graphs (a clique and a path) and adding an edge from an endpoint of the path to a vertex of the clique. Suppose we have a clique and a path of size n and let v be the vertex of the clique adjacent to the path. When the walk is at v is has $\frac{1}{n}$ probability of advancing to the path. Then, once in the path, it has $\frac{1}{n}$ probability of reaching the end before returning to v (why?). Hence, if a walk starts at n it will (i) with probability $\frac{1}{n^2}$ reach the end of the path (ii) with probability $\frac{n-1}{n}$ move to another vertex of the clique (iii) with probability $\frac{1}{n} - \frac{1}{n^2}$ move to the path but return to v without reaching the end. If we move to a different clique vertex, the walk will take $n-1$ steps in expectation to return to v. We have:

$$h(v) \geq \frac{n-1}{n}(n - 1 + h(v)) + \frac{n-1}{n^2} h(v)$$

This implies that $\frac{h(v)}{n^2} \geq \frac{(n-1)^2}{n}$, therefore $h(v) = \Omega(n^3)$.