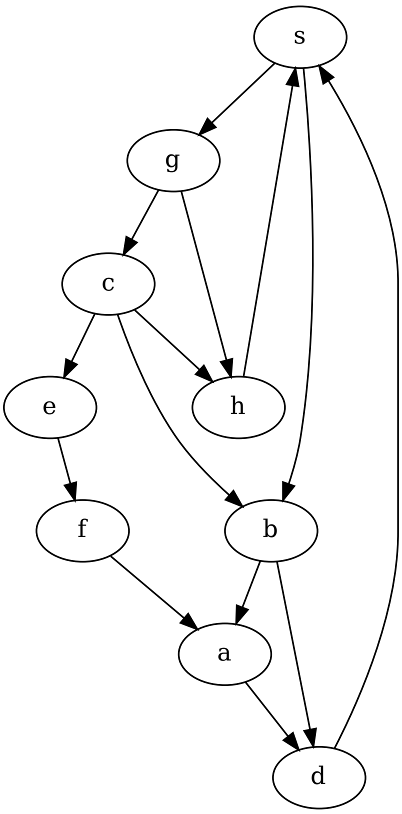
TD 2: BFS

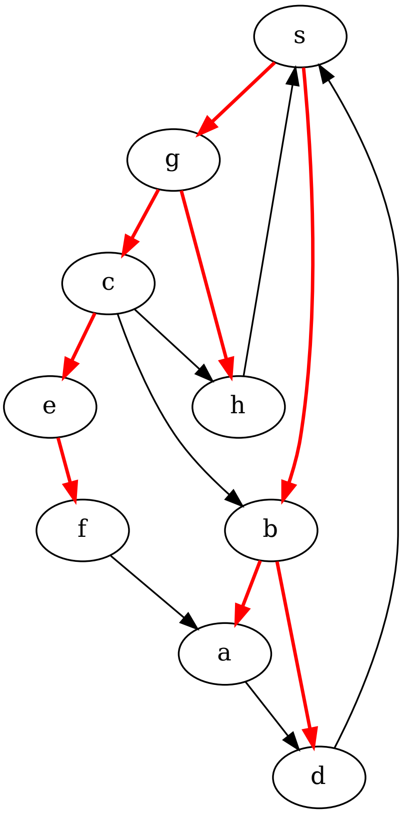
1 Execute BFS

Execute the BFS algorithm on the directed graph below, starting from vertex s. You can assume that adjacency lists are ordered alphabetically. Show the contents of the queue at every iteration, the distances calculated, and the edges of the BFS tree.



Solution:

BFS tree:



Distances:

Vertex	distance
a	2
b	1
c	2
d	2
e	3
f	4
g	1
h	2

Queue contents:

Time	Contents
1	s
2	b, g
3	g,a,d
4	a,d,c,h
5	d,c,h
6	c,h
7	h,e
8	e
9	f
10	$\emptyset$

2 BFS layers

Show that if we execute BFS on a graph $G = (V, E)$ starting from s, for all $uv \in E$ we have $|d_{BFS}(u) - d_{BFS}(v)| \leq 1$. (Recall that $d_{BFS}(v)$ is the distance computed by BFS for v and we have shown that $d_{BFS}(v) = \mathrm{dist}(s, v)$. Furthermore, we have shown that any two vertices which are simultaneously in the BFS queue have d_{BFS} values which differ by at most 1.).

Solution:

Suppose without loss of generality that u is added to the queue first. At the moment when u exits the queue we have two cases:

- v is currently White. Then, v will be added to the queue and $d_{BFS}(v) = d_{BFS}(u) + 1$.
- v is currently Gray. Then, u, v where in the queue at the same time, so their distances differ by at most 1.

Alternative solution: We use the fact that $d_{BFS}(v) = \mathrm{dist}(s, v)$ and prove that if $uv \in E$, then $|\mathrm{dist}(s, u) - \mathrm{dist}(s, v)| \leq 1$. Without loss of generality, suppose $\mathrm{dist}(s, u) \leq \mathrm{dist}(s, v)$. Then, $\mathrm{dist}(s, v) \leq \mathrm{dist}(s, u) + 1$, because one path from s to v can be constructed by taking a shortest path from s to u and appending the edge uv. We conclude that $\mathrm{dist}(s, u) \leq \mathrm{dist}(s, v) \leq \mathrm{dist}(s, u) + 1$, which implies the desired statement.

3 Destroying connectivity

Suppose that in an n-vertex connected undirected graph G, two (given) vertices s, t are at distance strictly greater than $n/2$.

- Prove that there exists a vertex x, such that if we delete x from the graph, then we destroy all paths from s to t.
- Give an algorithm that finds x in time $O(m + n)$ (assuming G is given in the form of adjacency lists).

Solution:

Execute BFS on G starting from vertex s. We obtain as a result the distance of each vertex from s and confirm that $\mathrm{dist}(s, t) > n/2$. Consider now the sets of vertices $D_i = \{v \in V \mid \mathrm{dist}(s, v) = i\}$, for $i \in \{1, \ldots, \mathrm{dist}(s, t) - 1\}$. We claim that there exists i such that $|D_i| = 1$.

Proof of claim: (Pigeonhole principle) Suppose that $|D_i| \neq 1$ for all i. We observe that for all i, $|D_i| > 0$, because otherwise there would be no path from s to t. If for all i, $|D_i| \geq 2$, then the graph has at least $2(\mathrm{dist}(s, t) - 1) + 2 > 2(\frac{n}{2} - 1) + 2 = n$ vertices, contradiction, where we have also counted s, t in the vertices of the graph. Hence, for some i, $|D_i| = 1$. We set x to be the unique vertex at distance i from s.

We now observe that removing x from G destroys all paths from s to t, because every such path must traverse a vertex at distance i from s, and the only such vertex was x.

In order to determine the vertex x, we can sort the vertices of the graph according to their distance from s. This can be done in $O(n)$ time, as all distances are between 0 and n. We now traverse the sorted array and find a vertex whose distance from s is different from that of its previous and next element.

4 Different BFS trees

For simplicity, we usually assume that adjacency lists are alphabetically ordered. However, using lists in a different order may affect the tree output by the BFS algorithm.

1. Give an example of a graph and two orderings of the vertices such that executing BFS with each ordering produces different trees. Does the ordering of the vertices affect the d_{BFS} values computed?
2. Give an example of a graph $G = (V, E)$, a vertex $s \in V$, and an edge $e \in E$, such that no matter how we order the vertices in the adjacency lists, e will never be part of the tree output by BFS.

3. Give an example of a graph $G = (V, E)$, a vertex $s \in V$, and a set of edges $E_\pi \subseteq E$, such that (i) E_π is a shortest-path tree from s in G (ii) no possible ordering can make BFS output the tree E_π.

Solution:

1. Consider a C_4, with vertices s, a, b, c in this order (so s is not adjacent to b). If we order alphabetically, the output BFS tree will have b as a child of a; otherwise b will be a child of c.

2. Consider a K_3, with vertices s, a, b. The edge ab is never part of the output of BFS starting from s, as it is not part of any shortest path starting from s.

3. Consider a path on 5 vertices $a_2 - a_1 - s - b_1 - b_2$ and let the edges of this path be E_π. Add to the graph the edges a_1b_2 and a_2b_1. Now, if we execute BFS from s, we can either explore a_1 or b_1 first. In the first case, this will add use in the tree the edge a_1b_2 and therefore not use b_1b_2. In the second case, this will use b_1a_2, and therefore not use a_1a_2.

5 Find a cycle through an edge

Give an algorithm which takes as input a graph $G = (V, E)$, and an edge $uv \in E$ and decides in linear time whether there exists a cycle in G that traverses the edge uv.

Solution:

Remove the edge(*) uv from G and execute BFS starting from u. If the algorithm finds a path from u to v (that is, the distance computed for v is not ∞), then the path together with the edge uv form a cycle in G. If, on the other hand, no such path exists, then no cycle containing uv exists in G, because such a cycle would contain a path from u to v avoiding the edge uv.

(*) Removing an edge from a graph can be done in $O(1)$ time in adjacency matrix form, or $O(m)$ time in adjacency list form. That being said, it may be preferable not to modify the graph. In this case, removing a single edge is still possible: we execute the algorithm and every time an edge e is about to be traversed, we check if $e = uv$ and only proceed if $e \neq uv$.