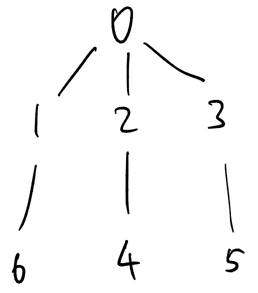
Lecture 25

Breadth-first search (BFS) and priority-first search (PFS)

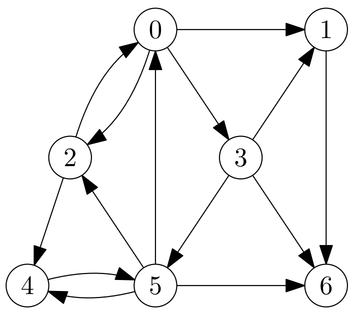
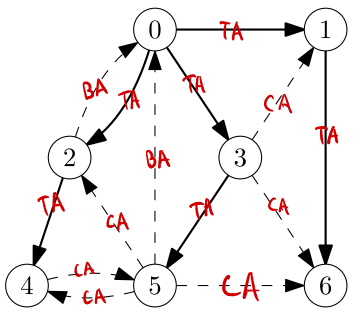
Definition 25.1. In ***breadth-first search*** (BFS) the new grey node chosen is the one that has been grey for the **longest** time.

BFS takes us away from the root node as slowly as possible. First we visit the root, then all its neighbours, then all neighbours of its neighbours and so on. The root is the first node to turn black.

As with DFS, the running time of BFS is in $\Theta(n+m)$ when implemented using adjacency lists.



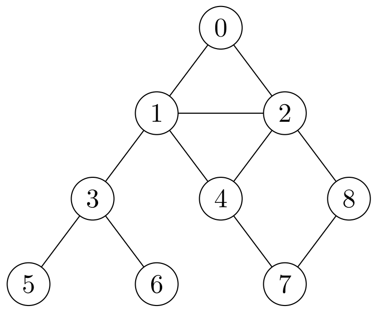
Example 25.2. A digraph and its BFS search tree, rooted at node 0. The dashed arcs indicate the original arcs that are not part of the BFS search tree.

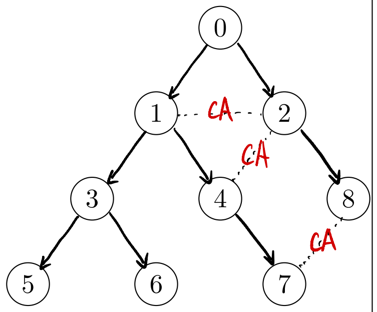



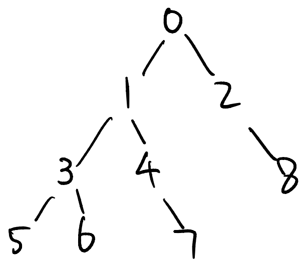
Find a tree arc, cross arc, forward arc and back arc (or say if that type of arc does not exist for that traversal).

No forward-arcs

Example 25.3. Use the nodes on the right to draw the search tree you obtain by running BFS on the graph on the left, starting at vertex 0. Use dashed edges to indicate edges that are not arcs in the search tree.







Find a tree arc, cross arc, forward arc and back arc (or say if that type of arc does not exist for that traversal).

No forward/back arcs

25.1 Pseudocode for breadth-first search

The first-in first-out processing of the grey nodes in BFS is ideally handled by a queue. The pseudocode for `BFS` and `BFSvisit` is in Algorithms 6 and 7. The timestamps `seen` and `done` of DFS are of less use here. It is more useful to record the number of steps from the root in the $\boxed{\text{array } d.}$

$\downarrow$

$d[x] = d(r, x)$

(distance)

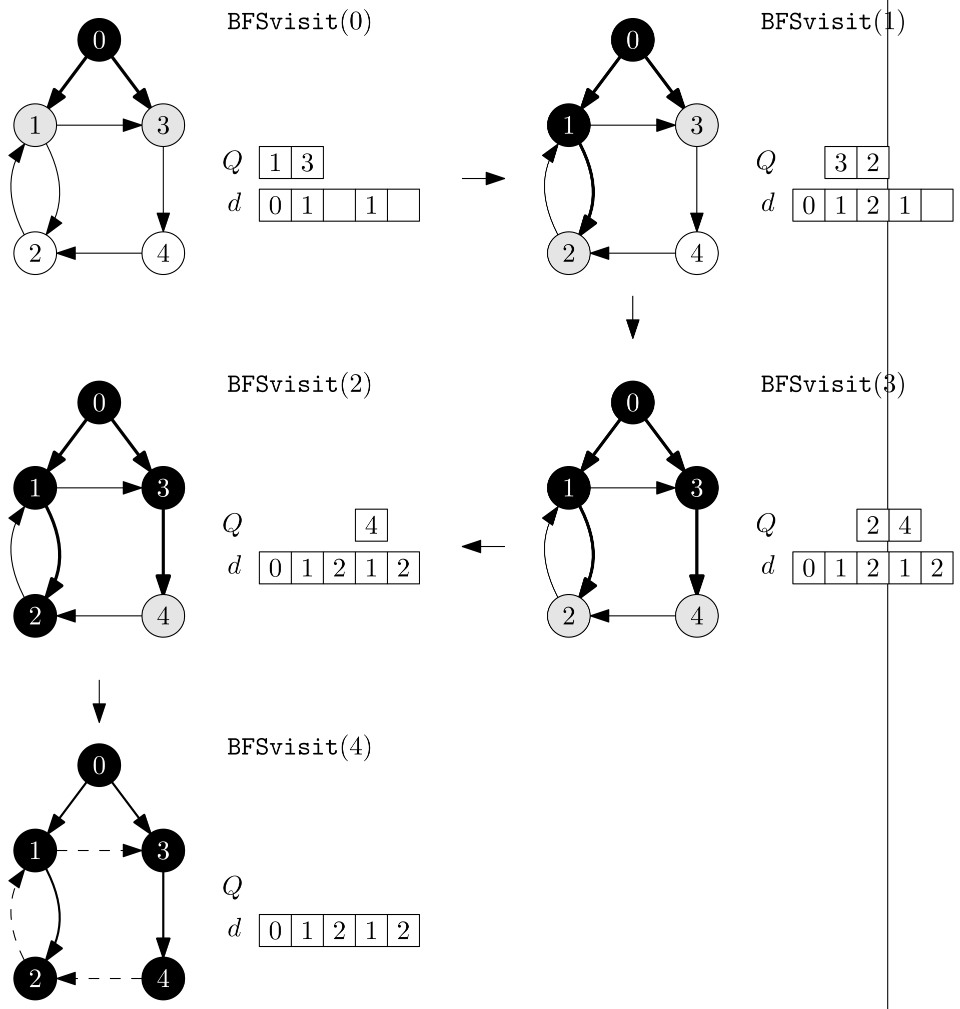
Algorithm 6 Breadth-first search algorithm.

```
function BFS(digraph $G$)
    queue $Q$
    array colour[$0..n-1$], pred[$0..n-1$], $d[0..n-1]$
    for $u \in V(G)$ do
        colour[$u$] $\leftarrow$ WHITE; pred[$u$] $\leftarrow$ null        ▹ initialise arrays
    for $s \in V(G)$ do
        if colour[$s$] = WHITE then        ▹ find a WHITE node
            BFSvisit($s$)        ▹ start traversal from $s$
    return pred, $d$
```

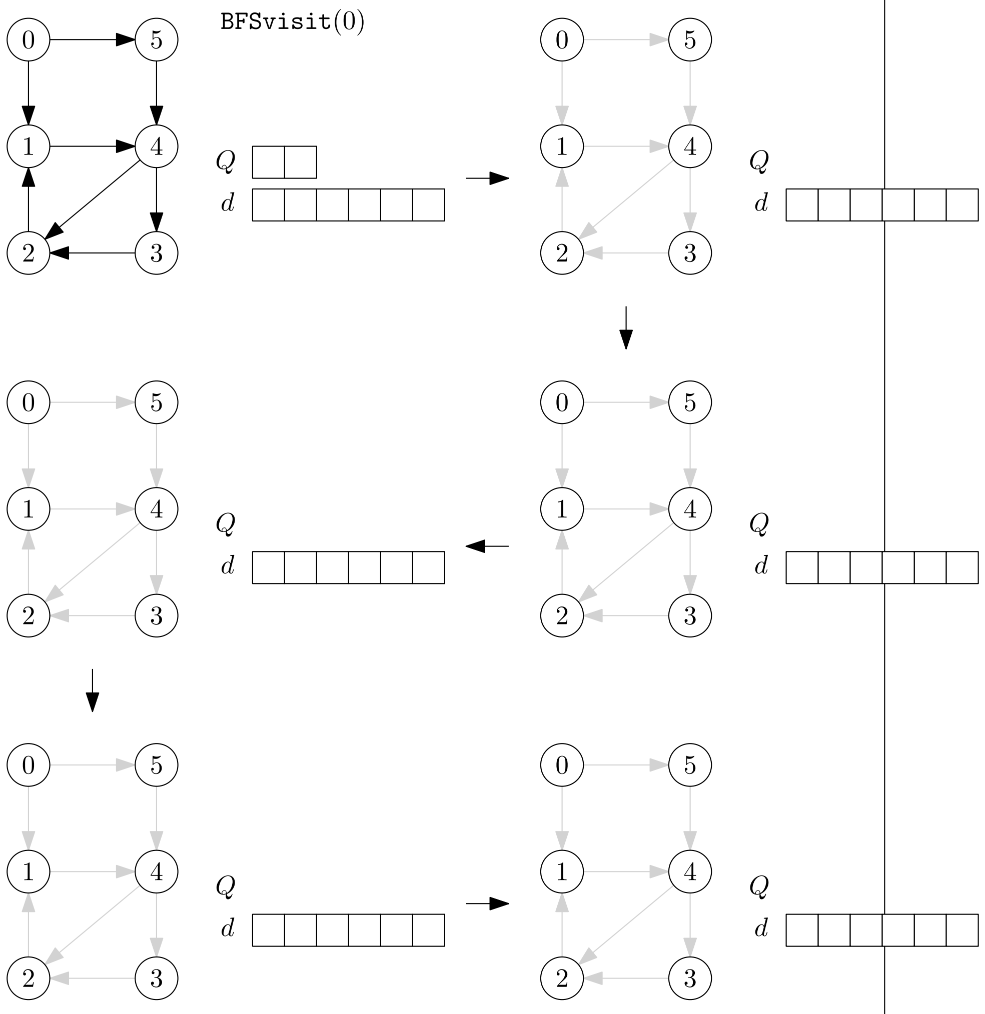
Algorithm 7 Breadth-first search visit algorithm.

```
function BFSvisit(node $s$)
    colour[$s$] $\leftarrow$ GREY; $d[s] \leftarrow 0$
    $Q$.insert($s$)        ▹ put $s$ in queue
    while not $Q$.isEmpty() do
        $u \leftarrow Q$.peek()        ▹ get node at front of queue
        for each $v$ adjacent to $u$ do
            if colour[$v$] = WHITE then        ▹ find white neighbours
                colour[$v$] $\leftarrow$ GREY; pred[$v$] $\leftarrow u$; $d[v] \leftarrow d[u]+1$
                $Q$.insert($v$)    ▹ update colour, tree, and depth, put in queue
        $Q$.delete()        ▹ done with this node
        colour[$u$] $\leftarrow$ BLACK
```

Example 25.4. BFS on a digraph.



Example 25.5. Execute BFS starting at 0 and following Example 25.4 fill out the values for Q and d, and highlight the BFS search tree after each step.



25.2 BFS useful results and facts

It is rather obvious that BFS processes all nodes at distance 1, then all nodes at distance 2, etc, from the root. The formal theorem stating this is given below without proof (see book for a simple inductive proof).

Theorem 25.6. Suppose we run `BFS` on a digraph G. Let $v \in V(G)$, and let r be the root of the search tree containing v. Then $d[v] = d(r, v)$.

We can classify arcs, but the answer is not as nice as with DFS.

Theorem 25.7. Suppose that we are performing `BFS` on a digraph G. Let $(v, w) \in E(G)$ and suppose that we have just chosen the grey node v. Then

- if (v, w) is a tree arc then `colour`$[w]$ = WHITE, $d[w] = d[v] + 1$;
- if (v, w) is a back arc, then `colour`$[w]$ = BLACK, $d[w] \le d[v] - 1$;
- there are no forward arcs; and
- if (v, w) is a cross arc then one of the following holds:
  - $d[w] < d[v] - 1$, and `colour`$[w]$ = BLACK;
  - $d[w] = d[v]$, and `colour`$[w]$ = GREY;
  - $d[w] = d[v]$, and `colour`$[w]$ = BLACK;
  - $d[w] = d[v] - 1$, and `colour`$[w]$ = GREY;
  - $d[w] = d[v] - 1$, and `colour`$[w]$ = BLACK.

Example 25.8. Explain why there are no forward arcs when performing BFS on a digraph.

If (v,w) is a forward arc then v is ancestral to w in F but v is not a tree arc. $d[w] \geq d[v]+2$

But $d[v]+2 \leqslant d[w] = d[r,w] \leqslant d[v]+1$

$\therefore$ No forward-arcs

In the special case of graphs we can say more.

Theorem 25.9. Suppose that we have performed BFS on a graph G. Let $\{v,w\} \in E(G)$. Then exactly one of the following conditions holds.

- $\{v,w\}$ is a tree edge, $|d[w] - d[v]| = 1$;
- $\{v,w\}$ is a cross edge, $d[w] = d[v]$;
- $\{v,w\}$ is a cross edge, $|d[w] - d[v]| = 1$.

25.3 Priority-first search

Priority-first search is a more general and sophisticated form of traversal that encompasses both BFS and DFS (and others).

- Each grey node has associated with it an integer ***key***.
- The interpretation of the key is of a priority: the smaller the key, the higher the priority.
- The rule for choosing a new grey node is to choose one with the smallest key.

- The key can either be assigned once when the node is first seen and then left unchanged, or could be updated at other times. We concentrate on unchanging keys here.
- To mimic BFS, set the key for the node v to be the time that v turns grey. It will always remain as the lowest key until it turns black. seen[V]
- To mimic DFS, set the key for node v to be $-$`seen`$[v]$, so that the most recently seen node has the lowest key.
- The running time of PFS depends on how long it takes to find the minimum key value.
- The rules described here implemented using an array take $\Omega(n)$ to find the lowest key, so the algorithm is $\Theta(n^2)$. Contrast with with $\Theta(m+n)$ for standard traversal.
- PFS is best described via the priority queue ADT, which has more efficient implementations than using a standard array.

Pseudocode `PFS` and `PFSvisit` demonstrating the PFS is presented in Algorithms 8 and 9. The subroutine `setKey` there is the rule for giving the key value when a node is inserted. We do not include any code for `setKey`.

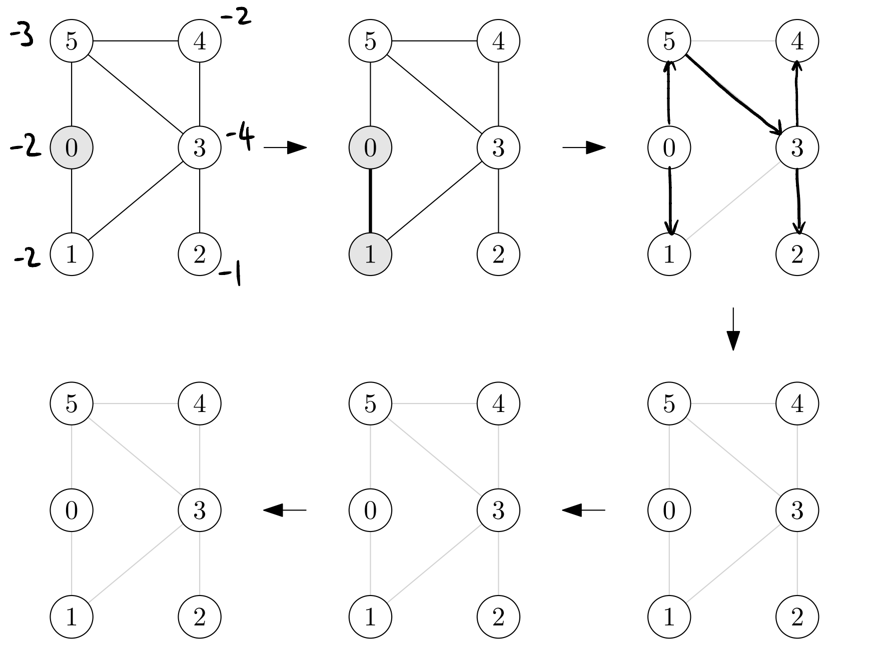
Algorithm 8 Priority-first search algorithm.

```
1: function PFS(digraph $G$)
2:     priority queue $Q$
3:     array colour[$0..n-1$], pred[$0..n-1$]
4:     for $u \in V(G)$ do
5:         colour[$u$] $\leftarrow$ WHITE; pred[$u$] $\leftarrow$ null
6:     for $s \in V(G)$ do
7:         if colour[$s$] = WHITE then
8:             PFSvisit($s$)
9:     return pred
```

Algorithm 9 Priority-first visit algorithm.

```
function PFSvisit(node $s$)
    colour[$s$] $\leftarrow$ GREY
    $Q$.insert($s$, setKey($s$))
    while not $Q$.isEmpty() do
        $u \leftarrow Q$.peek()
        if $u$ has a neighbour $v$ with colour[$v$] = WHITE then
            colour[$v$] $\leftarrow$ GREY
            $Q$.insert($v$, setKey($v$))
        else
            $Q$.delete()
            colour[$u$] $\leftarrow$ BLACK
```

Example 25.10. Execute PFS on the graph below so that nodes with $\boxed{\text{higher degree have higher priority}}$ (recall that a lower key corresponds to a higher priority).
Start the traversal at vertex 0 and break ties by choosing the vertex with the lower index first.
At each step add one vertex and edge to the search tree.



Set Key(v) = −degree [v]

Q: 0 1 5 3 2 4
P: -2 -2 -3 -4 -1 -2