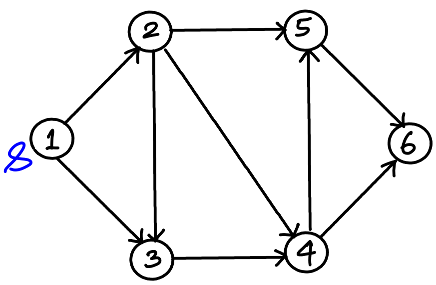
MATH 566: Lecture 8 (09/12/2024)

Today: * BFS, DFS examples
* complexity of search
* topological ordering

Example

slightly modified version of AMO Fig 3.5.



order
(i)

Outarc lists

$A(1) = \{(1,2),(1,3)\}$

$A(2) = \{(2,3),(2,4),(2,5)\}$

$A(3) = \{(3,4)\}$

$A(4) = \{(4,5),(4,6)\}$

$A(5) = \{(5,6)\}$

$A(6) = \emptyset$

Initialization

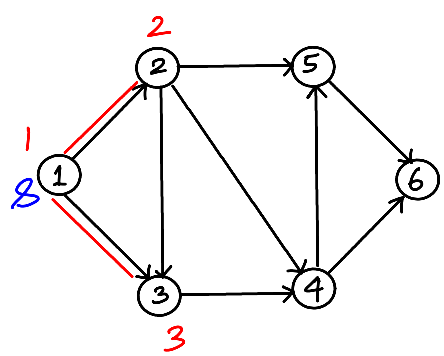
$$
\begin{array}{rcl}
 & & \quad 1 \;\; 2 \;\; 3 \;\; 4 \;\; 5 \;\; 6 \\
\text{Mark} & = & [\cancel{0}\,1 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0] \\
\text{Pred} & = & [\cancel{M}\,0 \;\; M \;\; M \;\; M \;\; M \;\; M] \\
\text{Order} & = & [\cancel{0}\,1 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0]
\end{array}
$$

LIST = $\cancel{\emptyset}$ → [s]

next = 1

Mark(s) = 1 ← mark s

pred(s) = 0



the search tree is shown in red (after Iteration 2 here)

Iteration 1

i=1. Look at $\overset{(i,j)}{(1,2)}$ ← current arc

it is admissible as Mark(i)=1 & Mark(j)=0.

$$
\begin{array}{rcl}
 & & \quad 1 \;\; 2 \;\; 3 \;\; 4 \;\; 5 \;\; 6 \\
\text{Mark} & = & [1 \;\; \cancel{0}\,1 \;\; 0 \;\; 0 \;\; 0 \;\; 0] \\
\text{Pred} & = & [0 \;\; \cancel{M}\,1 \;\; M \;\; M \;\; M \;\; M] \\
\text{Order} & = & [1 \;\; \cancel{0}\,2 \;\; 0 \;\; 0 \;\; 0 \;\; 0]
\end{array}
$$

next → 2

LIST = [1 2]

in general, we set pred(j) = i;

Iteration 2

i=1 → taken from front of LIST

(1,2) is now inadmissible, so move on.

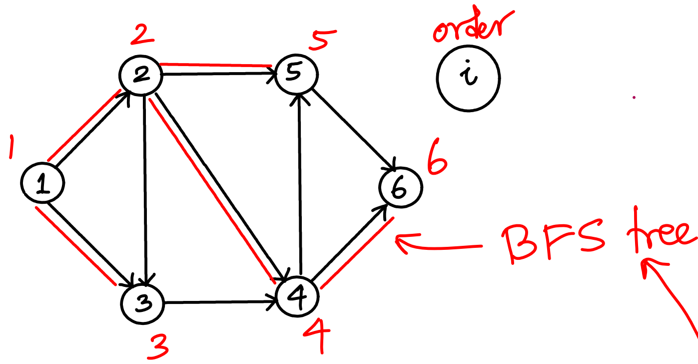
current arc ← (1,3) → admissible

$$
\begin{array}{rcl}
 & & \quad 1 \;\; 2 \;\; 3 \;\; 4 \;\; 5 \;\; 6 \\
\text{Mark} & = & [1 \;\; 1 \;\; \cancel{0}\,1 \;\; 0 \;\; 0 \;\; 0] \\
\text{Pred} & = & [0 \;\; 1 \;\; \cancel{M}\,1 \;\; M \;\; M \;\; M] \\
\text{Order} & = & [1 \;\; 2 \;\; \cancel{0}\,3 \;\; 0 \;\; 0 \;\; 0]
\end{array}
$$

next → 3

LIST = [1 2 3]

BFS Example (continued)



$$\begin{array}{lc} & \begin{array}{cccccc} \overset{s}{1} & 2 & 3 & 4 & 5 & 6 \end{array} \\ \text{Mark} = & [\cancel{0}\,1,\ \cancel{0}\,1,\ \cancel{0}\,1,\ \cancel{0}\,1,\ \cancel{0}\,1,\ \cancel{0}\,1] \\ \text{pred} = & [\cancel{M}\,0,\ \cancel{M}\,1,\ \cancel{M}\,1,\ \cancel{M}\,2,\ \cancel{M}\,2,\ \cancel{M}\,4] \\ \text{order} = & [\cancel{0}\,1,\ \cancel{0}\,2,\ \cancel{0}\,3,\ \cancel{0}\,4,\ \cancel{0}\,5,\ \cancel{0}\,6] \end{array}$$

$$\text{LIST} = [1] \to [1\ 2] \to [\cancel{1}\ 2\ 3] \to [2\ 3] \to [2\ 3\ 4] \to [\cancel{2}\ 3\ 4\ 5] \to [\cancel{3}\ 4\ 5] \to [4\ 5] \to [\cancel{4}\ 5\ 6] \to [\cancel{5}\ 6] \to [\cancel{6}]$$

Iterations 9, 10, 11: delete nodes 4, 5, 6

initialization

Iteration: 0 1 2 3 4 5 6 7 8 9 10 11

(Changes made in each iteration are color-coded accordingly)

We get a search tree, called the breadth-first search (BFS) tree.

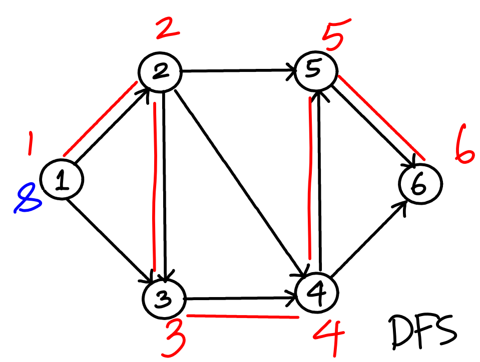
We could stop the algorithm once we have marked all nodes here. But, of course, all nodes may not be reachable from an s in every instance. For completeness, we do want to run the steps till LIST is indeed empty.

Property In the BFS tree, the directed path from s to i contains the smallest number of arcs among all directed s–i paths, i.e., it is <u>a</u> shortest path in terms of # arcs.
→ the SP here may not be unique, but # arcs is minimum

Depth-First Search (DFS)

Maintain LIST as a stack, i.e., in a **last-in first out** (LIFO) order.

Here is how DFS will proceed on the same instance:



DFS tree

LIST: [1] → [2 1] → [3 2 1] → [4 3 2 1] → [5 4 3 2 1] → [~~6 5 4 3 2 1~~] → ϕ

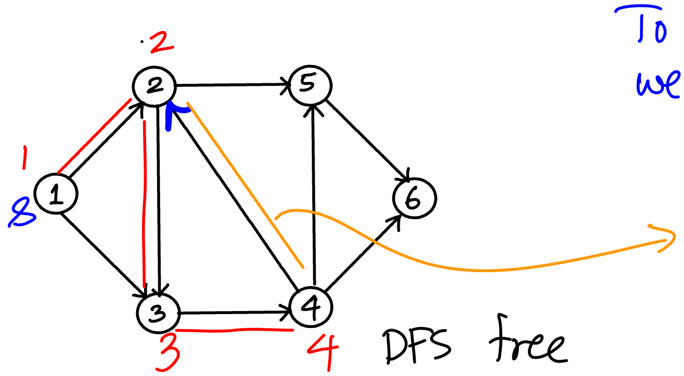
delete node in this order

Note that while the DFS tree is different from the BFS tree, order vectors turn out to be the same. But that is just a coincidence on this small network!

For instance, the search path to node 6 in DFS has 5 arcs, while the corresponding path in BFS is only 3 arcs long.

DFS identifies directed cycles quickly.

To include a directed cycle (for illustration), we replace (2,4) with (4,2).



first inadmissible arc

DFS tree

The first inadmissible arc in DFS identifies a directed cycle. BFS could identify the directed cycle as well, but it could take longer than DFS.

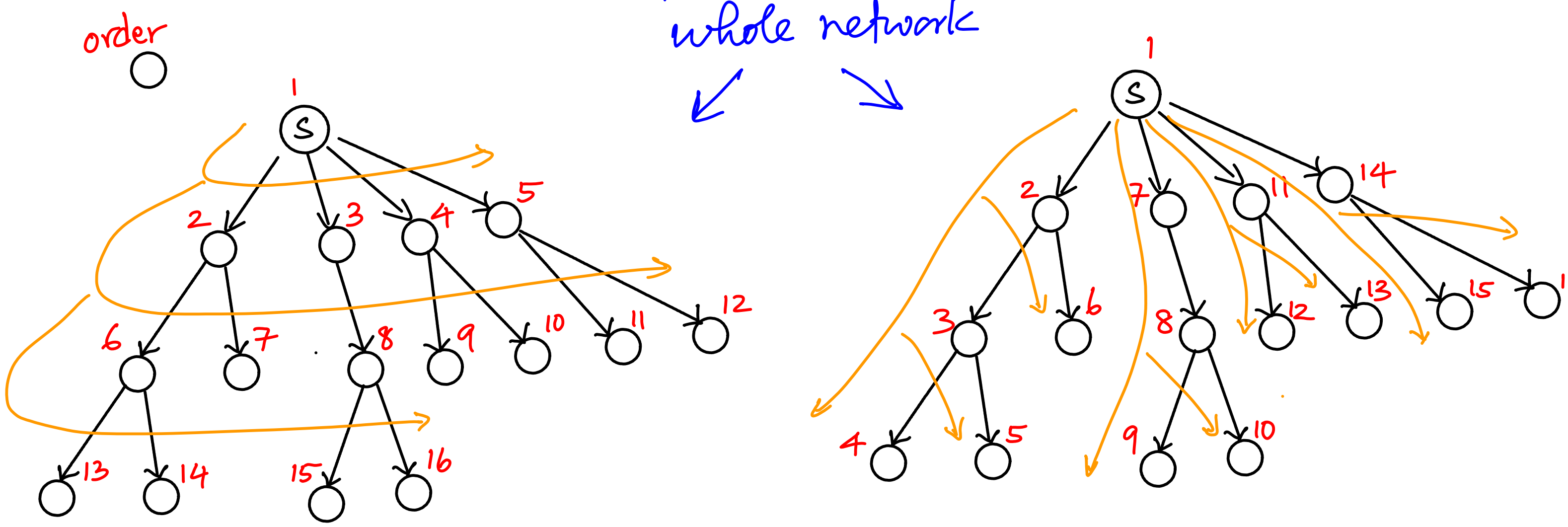
Q. How do BFS and DFS compare on large networks in general?

Here is a comparison of how these two searches visit nodes in a typical network. Notice how DFS dives deep fast.

BFS tree

only the search trees are shown here — not the whole network

DFS tree



BFS spreads at each "level" (or depth, in terms of # arcs from s) completely before descending to next level. DFS, on the other hand, dives to the bottom-most level along each "branch" before spreading wide to the next branch.

Complexity (Running time) of (Generic) Search algorithm

```
algorithm search;
begin
    unmark all nodes in N;   → O(n)
    mark node s;
    pred(s) : = 0;
    next : = 1;                          } constant # operations
    order(s): = 1; → next;
    LIST : = {s}
    while LIST ≠ Ø do
    begin
        select a node i in LIST;
        if node i is incident to an admissible arc (i, j) then   |A(i)| times
        begin
            mark node j;
            pred( j) : = i;
            next : = next + 1;           } constant time ops
            order( j) : = next;
            add node j to LIST;
        end
        else delete node i from LIST;
    end;
end;
```

Figure 3.4 Search algorith

Within the **while** loop, a node gets added and deleted from LIST once, giving $2n$ operations, i.e, $O(n)$ time.

We examine, in the worst case, $\sum_{i\in N} |A(i)| = m$ arcs for admissibility, taking $O(m)$ time.

$\Rightarrow$ The overall running time is $O(m+n) = O(m)$, as $m \geq n$ typically.

We assume $m > n$ (typically). Given n nodes, we could have up to n^2 directed arcs. If there are more nodes than arcs, many nodes might be isolated, and hence would not affect algorithms as much as the connected components.

Notice that we do not have to look at all arcs repeatedly within the while loop. Indeed, each arc has to be examined for admissibility only once. If an arc becomes inadmissible at some point, it will never become admissible again. We could hence just run through $A(i)$ for each node i, examining the arcs just once each.

In practice, we can examine all outarcs of current node i in a unified manner, identify the admissible arcs from among them, and mark their head nodes in a unified manner as well.

We now look at a variation and an application of search.

<u>Reverse Search</u> Find all nodes from which one can <u>reach</u> a node t.
via directed path

In generic search, start with LIST = [t] (instead of [s]), and examine AI(·) lists. From node j, (i,j) is **admissible** if j is marked and i is unmarked. Rest of the search process applies in this case as well.

<u>Strong Connectivity</u> Recall, $G=(N,A)$ is strongly connected if there exists a directed path from every node i to every node j.

Pick any node s and apply forward search from s and reverse search to s. If we get a spanning tree in both cases, then the network is strongly connected.

$\Rightarrow$ Strong connectivity can be tested in $O(m)$ time.
Two searches from any one node $\Rightarrow O(2m) = O(m)$ time.

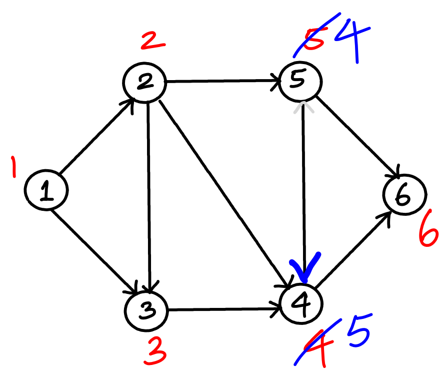
It is indeed sufficient to do this paired search from a single node (s). For any node pair i, j, we could go from i to s and s to j.

Topological Ordering

We use order(i) as labels for nodes. It is often desirable to assign order(·) such that every arc goes from a lower order node to a higher order node.

Def A labeling order(·) is a **topological ordering** if $\forall (i,j) \in A$, we have order(i) < order(j).

Consider the example we used to illustrate BFS and DFS:



The ordering we got for BFS is a topological ordering here.

But if we had (5,4) instead of (4,5) the ordering is not topological. But if we swap order(4) and order(5) now, we again get a topological ordering.