Notes, 4(a)

ECE 606

Design Strategy 1: Incremental

Some problems lend themselves to the incremental strategy: in each iteration, the algorithm shrinks the problem-size by a constant, e.g., 1.

Example: given an array of integers, find the minimum and maximum.

```
MinAndMax($A[1, \dots, n]$)
$min \leftarrow A[1]$, $max \leftarrow A[1]$, $i \leftarrow 2$
while $i < n$ do
    if $A[i] \leq A[i+1]$ then
        if $A[i] < min$ then $min \leftarrow A[i]$
        if $A[i+1] > max$ then $max \leftarrow A[i+1]$
    else
        if $A[i+1] < min$ then $min \leftarrow A[i+1]$
        if $A[i] > max$ then $max \leftarrow A[i]$
    $i \leftarrow i + 2$
if $i = n$ then
    if $A[i] < min$ then $min \leftarrow A[i]$
    if $A[i] > max$ then $max \leftarrow A[i]$
return $\langle min, max \rangle$
```

The above algorithm adopts the incremental strategy: it “eliminates” two entries of the array in a single iteration of the **while** loop.

Aside: the above algorithm is somewhat smarter than the naive algorithm: that one performs $2(n-1)$ comparisons; this one performs $\left\lceil \frac{3}{2}(n-1) \right\rceil$ comparisons only.

Another example: insertion sort, recursive version.

```
RecursiveInsertionSort($A, n$)
1 if $n = 1$ then return
2 RecursiveInsertionSort($A, n - 1$)
3 foreach $j$ from $1$ to $n - 1$ do
4     if $A[n] < A[j]$ then
5         $tmp \leftarrow A[n]$
6         foreach $k$ from $n$ downto $j + 1$ do
7             $A[k] \leftarrow A[k - 1]$
8         $A[j] \leftarrow tmp$
```

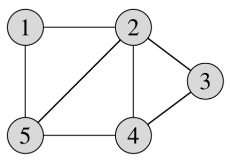
The number of comparisons, $f(n)$:

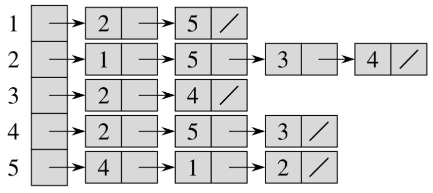
$$f(n) = \begin{cases} 0 & \text{if } n = 1 \\ (n-1) + f(n-1) & \text{otherwise} \end{cases}$$

$$\begin{aligned} f(n) &= (n-1) + f(n-1) \\ &= (n-1) + (n-2) + f(n-2) \\ &\dots \\ &= (n-1) + (n-2) + \ldots + 1 + 0 \\ &= \frac{n(n-1)}{2} = \Theta(n^2) \end{aligned}$$

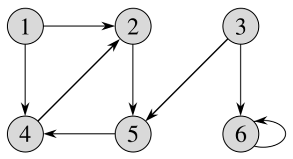
The basic graph algorithms, Breadth and Depth First Search, can be seen as incremental: we process one vertex at a time.

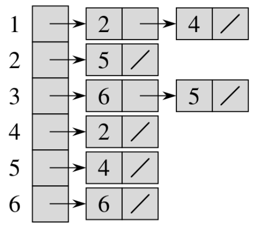
First: two conventional ways of representing graphs: adjacency list and adjacency matrix.





$$\begin{array}{c|ccccc|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}$$





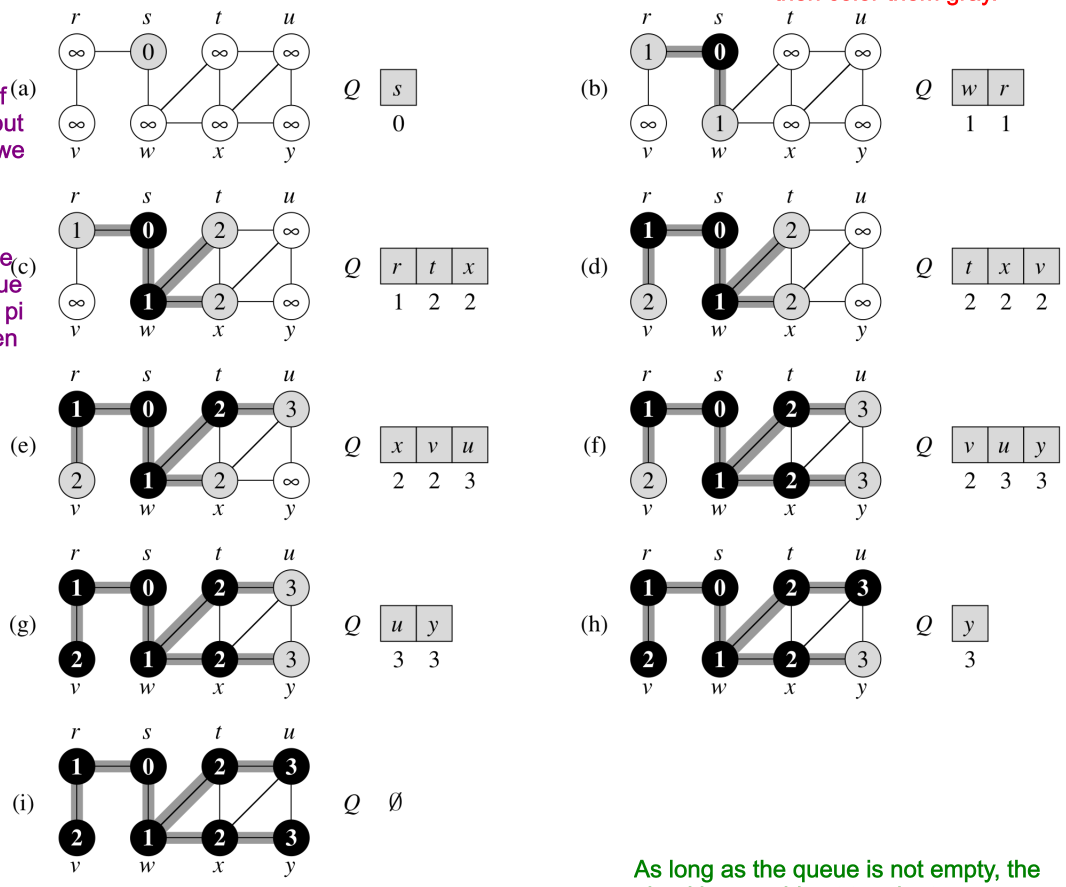
$$\begin{array}{c|cccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
\end{array}$$

BFS via example

And for (b), the pi value, which is the predecessor vertex, would be s.

As soon as we visit a white vertex, we enqueue it, we update the distance estimate to be one more than the vertex we are currently processing, in case (b), this would be s, we set the distance value for those vertices, and then color them gray.

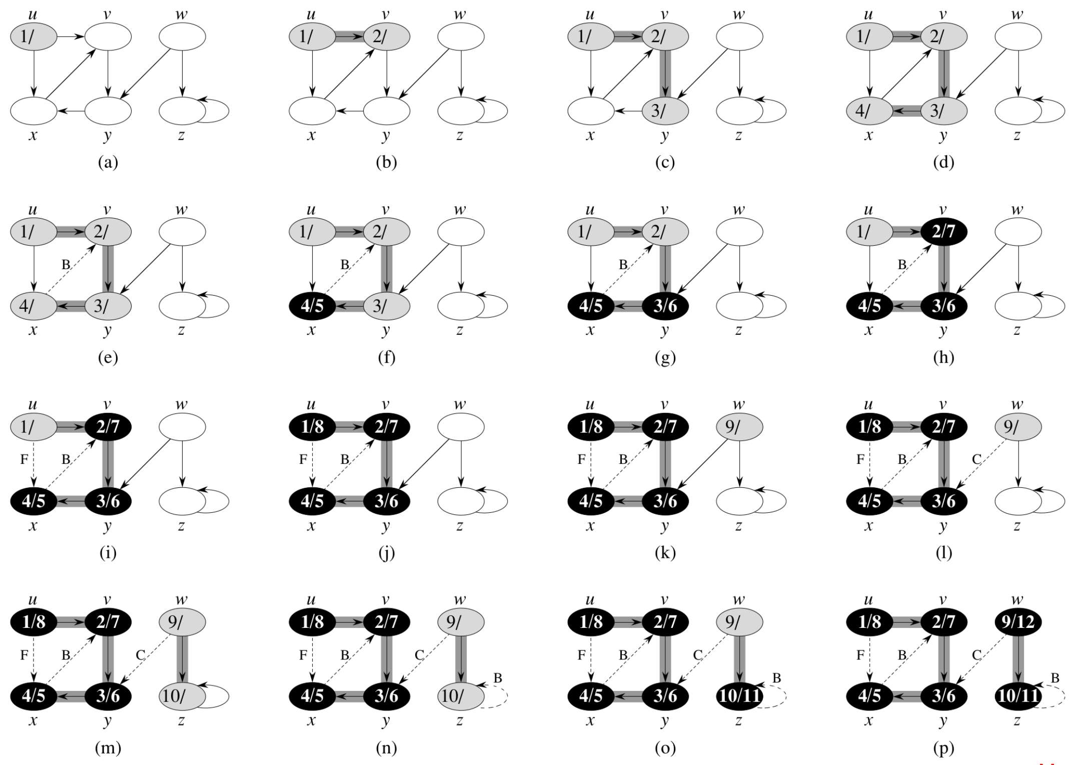
For (c), we would dequeue w, one of its neighbor is s, but s is not white, so we ignore it and enqueue t and x. We then update their d values to be 1 more than d value of w, and set their pi value to be w. Then color them gray.



As long as the queue is not empty, the algorithm would not terminate.

DFS via example

The label B in (e) means that the edge exists in the graph, but it will not be a tree edge in the DFS tree, but rather, it'll end up being a "back edge", which is the edge from a node to one of its ancestors in the tree.



Here is a special case of Back Edge, which is from a vertex to itself.

F is the "Forward edge", meaning it's the edge pointing from the ancestor to the descendent.

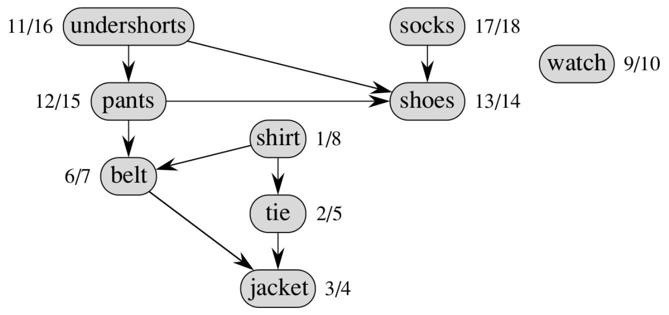
C is the "Cross edge", because w and y are in 2 different trees, and c is the cross edge that connects them.

If we start the DFS with w, then what we would end up with is every vertex besides u will be in the tree rooted at w, and u would be is another tree rooted at itself.

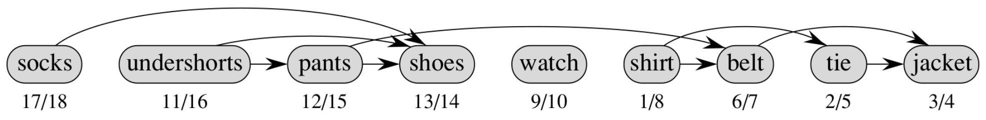
Topological Sort via example

Directed, acyclic graph.
(Directed Graph with no cycles)

(a)



(b)

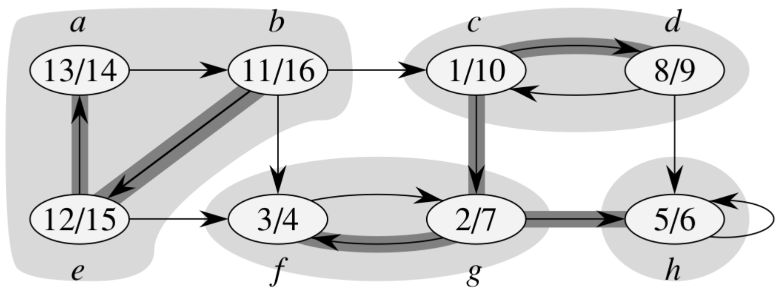


Strongly connected components of a directed graph.

SCC via example

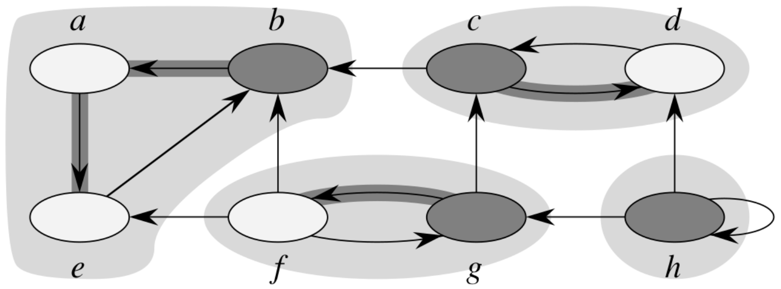
This graph may have cycles.

(a)



a,b,e in one SCC.
c,d in one SCC.
f,g in one SCC.
h in one SCC.

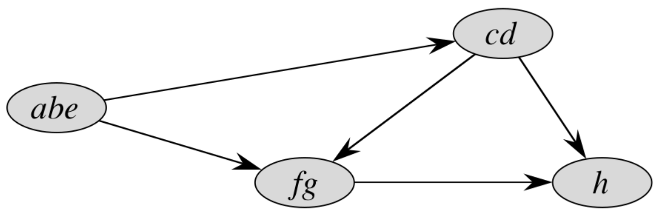
(b)



Given a directed acyclic graph, the SCC of the graph is unique. Meaning that not only does the vertex belong to some SCC, but also, it belongs to exactly one SCC.

The transpose of a graph is just to take each edge, and change its direction.

(c)



To find SCC of a graph, first run DFS on it, then, compute the transpose of the graph, and run DFS again. But this time, we are going to provide the vertices in decreasing finish time from the original DFS.

So, for the second DFS, B will be the first source vertex for the transposed graph, and a and e will be in the tree rooted at B. Then, c would be the vertex with highest finishing time, and d would in its tree.

Finally, once we are done running the DFS on the tranposed graph, the algorithm outputs each DFS tree of the transposed graph as a SCC.

Strongly connected components is a subset of the vertices such that in the graph, given any pair of distinct vertices in the subset, I can reach one from the other.

So given <u, v>, <u, v> is a strongly connected component of G, if and only if there exists a path from u to v, and a path from v to u.