Tree traversals

- Review from CS 134:
  - We usually describe tree traversal recursively.
    - Preorder (visit node before children)
    - Postorder (visit children before node)
    - Inorder
      - For binary trees only
      - Visit left child, node, right child
  - All take *Q(n)* time for an *n*-node tree.

BFS and DFS

- Breadth-first, depth-first search
  - Each search strategy starts at node, and explores the entire connected component.
- General view:
  - Vertices start out coloured white (not visited).
  - A visited vertex is coloured gray (visited, but may still have white neighbors).
  - When all neighbours of a vertex are visited, it is coloured black.

General View of Searching

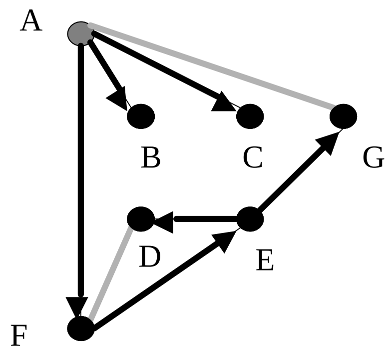
- The gray nodes form a "frontier".
- We choose any non-black neighbour of a gray node to be next visited.

- In general, we want to perform a computation:
  - preprocess when colouring gray
  - postprocess when colouring black
  - analogous to tree traversal uses.

Depth-First Search

- The DFS strategy:
  - Main idea:We use a stack to store gray nodes.
  - The algorithm visits new (white) vertices before dealing with older gray ones.
  - Hence it tends to explore deeply first.
  - We may add a timestamp of colour changes to indicate when a node turned gray ($d[u]$) and black ($f[u]$).
    - We will look at this later…

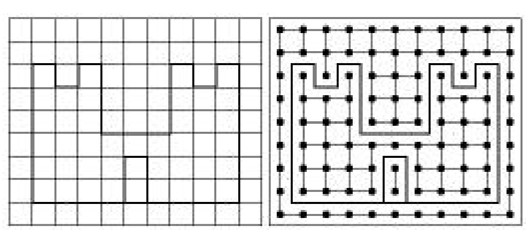
Example of depth-first search:



Exploring Undirected Graphs

- Problem statement:
  - Given an undirected graph G = (V;E), separate the vertices into connected components.
    - In particular, assign each vertex a number so that vertices have the same number iff they are in the same connected component.
    - Eg.: Consider a castle drawn on a grid. There are three areas made up of squares (nodes) that are connected if the squares are adjacent:

The idea is to colour each connected components with the same colour.



DFS Pseudocode

```
function dfs_visit(v,cnum)
   // Pre-condition: v is WHITE vertex
   // Find all vertices reachable from v via white vertices
   status[v] := gray;
   num[v] := cnum;
   for each w in out(v)
        if status[w] = white
               dfs_visit(w, cnum)
   status[v] := black;

// --- main program ---
// Start off with status of all vertices being white
cnum = 0;
for all vertices v in V
   if status[v] = white
      // all vertices in v's component are white; explore v's component
       dfs_visit(v, cnum);
       cnum := cnum + 1;
```

DFS Analysis

- Running time:
  - We call dfs_visit once for each vertex v 2 V.
  - If we ignore recursive calls, then a dfs_visit for vertex v takes $Q(1) + Q(\deg(v))$ time.
  - Thus the total running time is:

$$\Theta(n) + \Theta\left(\sum_{v \in V} \deg(v)\right) = \Theta(n+m)$$

Analysis of DFS

- Recall that we mentioned the use of a stack for a DFS implementation.
  - In the last program, the stack was implicit (it stores parameters for recursive calls)
    - "v on stack" means call to dfs_visit(v) has not finished.
    - dfs_visit is called once on every white node.
  - Each adjacency list is run through once.
  - Running time is $Q(|V|+|E|)$ or $Q(n+m)$.

DFS Pseudocode

- Let us make the stack explicit:

```
function dfs(start_node, A)
init_empty_stack(S);
for each v in A
   status[v] := white;
Push(start_node, S);
while S is non_empty
   x = Pop(S);
   if(status[x] = white) then
      process x;
      status[x] = black;
      for all edges (x, y) leaving x
            Push(y, S);
```

Other DFS Properties

- Discovery time:
  - We assign to every vertex a “timestamp" d(v) that records when it changes colour from white (unexplored) to gray (discovered).
- Finishing time:
  - We give every vertex a “timestamp" f(v) that records when it changed its colour from gray (discovered) to black (finished).
- Tree edges:
  - When we discover vertex w by calling dfs_visit from vertex v, we mark edge (v, w) as a tree edge.
    - We will call v the parent of w.

Other DFS Properties (cont.)

```
function dfs_visit(v, cnum)
  status[v] := gray;
  time := time + 1;    d[u] := time;   // Note
  num[v] := cnum;
  for each w in out(v)
     if status[w] = white
          edge (v,w) is a tree edge;  // Note
          dfs_visit(w, cnum)
  status[v] := black;
  time := time + 1;  f[u] := time;     // Note
```