L13

Pathfinding in Graphs

Many real-world problems can be solved by finding specific paths in graphs. (~~NavTomTom~~, FB friends-of-friends, internet routing, garbage collection, etc.)

① Does ~~there exist~~ a path exist?

Input: Graph G {
Vertices s, t

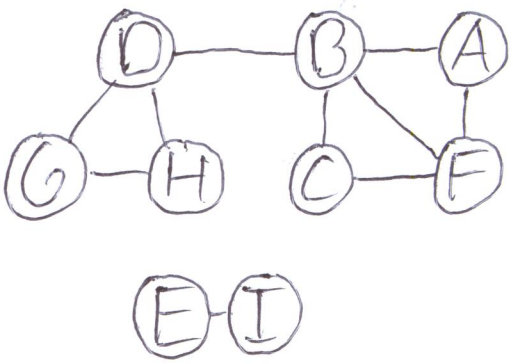
Output: Yes, if G contains a path from s to t.

We explore G using depth-first search (DFS).

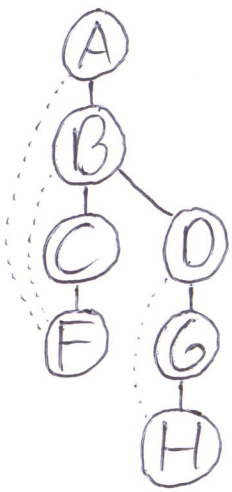
Algorithm DFS(G, v):
mark v
for each edge (v, w) do
if w is not marked then
DFS(G, w)

Algorithm PathExists (G, s, t):
DFS(G, s)
if t is marked then
return true
else
return false

Example:



DFS(G, A):



Solid edges (—) correspond to DFS calls and are called tree edges.
Others (····) are back edges.

- Correct?

Claim: After DFS(G, v), u is marked if and only if there is a path from v to u.

Proof: ⇒: The algorithm only marks neighbours of marked vertices. Since we start at v, all these vertices are reachable from v.

⇐: By contradiction. Suppose a path exists, but u is not marked. Let w be the first unmarked vertex on the path:

But when its predecessor on the path is marked, it will call DFS(G, w) which marks w. Therefore there cannot be a first unmarked vertex and all vertices - including u - are marked. □

- Terminates? Yes - each call the number of unmarked vertices decreases.

- Efficient? DFS is called once for each vertex. Each edge is inspected at most twice.

⇒ $O(|V|+|E|)$

Many problems about the structure of a graph can be solved by (modifying) DFS.

② What is the shortest path?

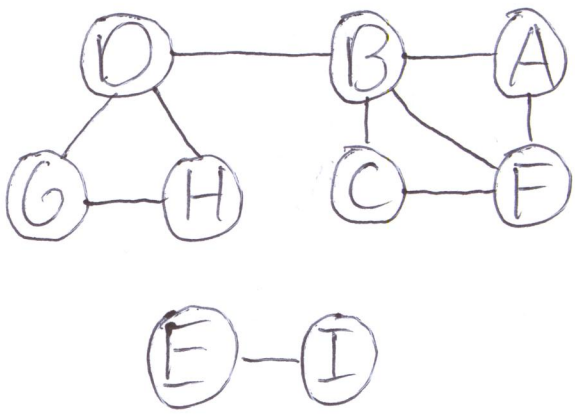
Input: Graph G
Vertices s, t

Output: Minimum number of edges on any s-t-path.

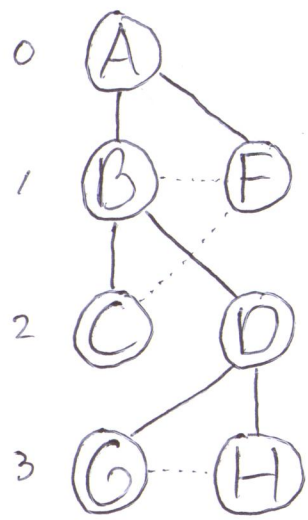
We explore G using Breadth-first search. (BFS)

Algorithm BFS(G, v, t):
  mark v ; $d[v] \leftarrow 0$
  $Q \leftarrow [v]$
  while Q is not empty do
    $x \leftarrow$ dequeue from Q
    for each edge (x, w) do
      if w is not marked then
        mark w ; $d[w] \leftarrow d[x] + 1$
        enqueue w in Q
  return $d[t]$

Example:



BFS(G, A, ..):



Tree edges (solid) form the shortest path from each vertex to A.

- Terminates? Yes - every vertex is added to Q at most once and one vertex is removed per step.
- Efficient? $O(|V|+|E|)$ again.
- Correct?

Loop Invariant: If $Q=[v_1, v_2, \dots, v_i]$, then

$$d[v_1] \leq d[v_2] \leq \dots \leq d[v_i] \leq d[v_1]+1.$$

- Initially, $Q=[v]$ and $d[v]=0 \leq 1 = d[v]+1$ ✓
- $Q=[v_1, v_2, \dots, v_i]$ $\quad d[v_1] \leq d[v_2] \leq \dots \leq d[v_i] \leq d[v_1]+1$

  $Q'=[v_2, \dots, v_i, \dots, v_j]$ $\quad d[v_2] \leq \dots \leq d[v_i] \leq \dots \leq d[v_j] \leq d[v_2]+1$

  $= d[v_1]+1 \leq d[v_2]+1$ ✓

Claim: After $BFS(G,v,t)$, $d[w]=\delta(v,w)$ $\forall w \in V$

$\delta(v,w)$: length of the shortest v-w-path

Proof: By induction on $\delta(v,w)$.

Base case ($\delta(v,w)=0$): Then $w=v$ and $d[v]=0$. ✓

IH: $d[w]=\delta(v,w)$ for all vertices with $\delta(v,w)<k$.

Step ($\delta(v,w)=k$): By the loop invariant, all vertices x with $d[x]<d[w]$ are processed before w. By induction, this includes all vertices with $\delta(v,x)=k-1$. Let u be the neighbour of w with $\delta(v,u)=k-1$ that is processed first. Then it enqueues w and sets $d[w]=d[u]+1=k-1+1=k=\delta(v,w)$. □