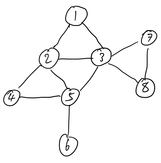
Lec05-BFS-DFS

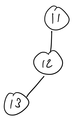
Thursday, September 3, 2026 3:07 PM

KT Ch. 3

Suppose we have a graph $G = (V, E)$

Can we find a path from a node s to a node t? (Connectivity)

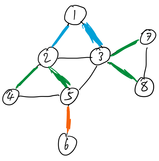


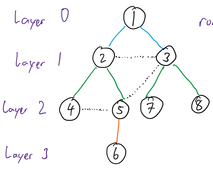


3 connected components

Breadth-First Search (BFS)

Explore outwards in layers defined by distance from root.





Layer 0 — root

Layer 1

Layer 2

Layer 3

notice, edges never jump layers

Thm If $(x,y) \in E$, then $|\text{layer}(x) - \text{layer}(y)| \leq 1$.

proof. Suppose not. Then x & y must be at least 2 layers apart.

WLOG, $\text{layer}(x) \leq \text{layer}(y) - 2$.

But all neighbors of x will be found no later than $\text{layer}(x) + 1$.

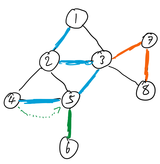
$\Rightarrow \text{layer}(y) \leq \text{layer}(x) + 1$

$\Rightarrow \text{layer}(x) \geq \text{layer}(y) + 1$

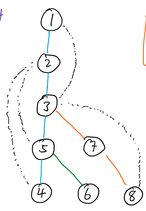
Contradiction $\rightarrow\!\leftarrow$ ▨

Depth-First Search (DFS)

Go as far as possible in a direction, and backtrack only as much as needed.



root



Both DFS & BFS create search trees

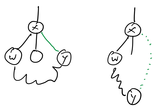
(y is a descendant of v) x is an ancestor of y if the path from y to the root goes through x

Thm If $(x,y) \in E$, then either x is an ancestor of y, or y is an ancestor of x in DFS tree T_G.

proof. WLOG, x is reached first in DFS. (so y is unexplored when x is reached)

All nodes explored between initially reaching x and leaving x for the last time (by backtracking past x) are descendants of x in T_G.

y must be explored before leaving x for last time because $(x,y) \in G$ (though it might be explored through a child) ▨



Both BFS/DFS are tree-growing algorithms:

- Let T be the current tree
- Maintain list of frontier edges that go from T to $V - T$.
- Repeatedly choose a frontier edge (somehow) and add it to T.

Question Which MST algorithm is most similar?

Prim's Kruskal Reverse-Delete None

Question: BFS/DFS returns an MST in which of the following circumstances?

Always

If all the edges have unique weight, and we break ties by choosing the lower-weight edge.

If all the edges have the same weight.

None of the above.

Tree-growing Pseudocode

```
TreeGrowing (graph G, vertex v, func nextEdge):
    T = ({v}, ∅)
    S = set of edges incident to v
    While S ≠ ∅:
        e = nextEdge(G, S)
        T = T + e          (and add relevant node)
        S = updateFrontier(G, S, e)
    return T
```

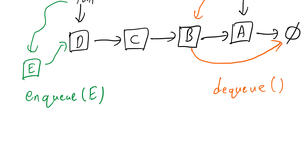
Runtime depends on nextEdge and updateFrontier.

Prim's is also an example of a tree growing algorithm

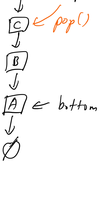
What's nextEdge for BFS? Select least-recently discovered edge. Queue

What's nextEdge for DFS? Select most-recently discovered edge. Stack

Queue: (for BFS) FIFO (First-In-First-Out)



Stack (for DFS) LIFO (Last-In-First-Out)



Runtime of BFS/DFS:

Every edge & node gets added to frontier once. $O(|V| + |E|)$ } overall $O(|V| + |E|)$

nextEdge takes $O(1)$ time, and happens $|E|$ times $O(|E|)$ }

Alternate recursive DFS

```
procedure DFS(G, u):
    while ∃ v an unvisited neighbor of u:
        mark v visited
        DFS(G, v)
```

Could we do a recursive BFS in the same way? Why or why not