Graphs and Connectivity

https://cs.pomona.edu/classes/cs140/

Outline

Topics and Learning Objectives

- Discuss the basics of graphs
- Introduce graph searching

Exercise

• Graph search

Extra Resources

• Introduction to Algorithms, 3rd, Chapter 22

Graphs

Represent pairwise relationships

Tons of uses

- Physical connections : roads (driving directions), network routing (phone), ...
- Relationship groups : social networks, similar purchases, ...
- Problem solving : each vertex may represent a partial part of the problem, and each edge is a step/move (e.g., Sudoku)

Tons of algorithms

• Cuts, clustering, searching, partitioning, contracting, ...

Graphs

For many reasons, graph algorithms are extremely important.

They are a ubiquitous tool for solving many engineering problems

- Signal traces on a PCB
- Balancing the load on a server
- Balancing the load across cores on a computer
- Scheduling the delivery of packages via drone
- Scheduling the path of an automated robot that is grabbing your Amazon purchase from shelves in a warehouse
- Topological networks
- Data mirroring across a network
- Modeling an ecology
- Modeling the nervous system
- The list goes on and on

For this reason, you will often be asked graph-related questions during interviews













BFS vs Dijkstra's vs A*

https://www.redblobgames.com/pathfinding/a-star/introduction.html

G = (V, E)

G is the standard symbol representing a graph

V is the standard symbol representing a set of graph vertices (|V| = n)

- Vertices are also sometimes referred to as nodes
- E is the standard symbol representing a set of graph edges (|E| = m)
- Each edge contains pointers to two vertices, for example: (v1, v2)
- The order of the vertices may or may not matter

Directed and Undirected

Notation for Edges



Graph Search and Connectivity

Goals:

- Find everything that is findable (a "path" from the start node exists)
- Don't explore anything twice (don't waste time)
- These operations are done in linear time,
- Note: it is often useful to consider O(n) algorithms as being "free"
 - (when compared to more complex tasks)



























General Algorithm

```
has been found and the other
FUNCTION Connectivity(G, start vertex)
                                               vertex has not been found.
  found = {v: FALSE FOR v IN G.vertices}
  found[start_vertex] = TRUE
  LOOP
      (vFound, vNotFound) = get_valid_edge(G.edges, found)
      IF vFound == NONE | vNotFound == NONE
                                                                         g
         BREAK
      ELSE
         found[vNotFound] = TRUE
                                                                    h
  RETURN found
                                                       n
```

Find an edge where one vertex

General Algorithm Outline

Claim: at the end of this algorithm

- if v is found
- Then there exists a path from s to v

Proof by contradiction

- Suppose the graph G has a path $\stackrel{\text{}}{\text{}}$ from the vertex s to the vertexs v
- \bullet Also suppose that upon completion of the algorithm ${\bf v}$ was not found
- Thus, we have an edge (u, w) such that u is found, and w is not found
- This is **contradictory** to the termination condition of the algorithm

Contradiction

found

S

Suppose G has a path p from s to v

W

Also suppose that upon completion of the algorithm v was not found Thus we have an edge (u, w) such that u is found and w is not found This is <u>contradictory</u> to the termination condition of the algorithm

not found

General Algorithm

```
has been found and the other
FUNCTION Connectivity(G, start vertex)
                                               vertex has not been found.
  found = {v: FALSE FOR v IN G.vertices}
  found[start_vertex] = TRUE
  LOOP
      (vFound, vNotFound) = get_valid_edge(G.edges, found)
     IF vFound == NONE VNotFound == NONE
                                                                        g
         BREAK
      ELSE
         found[vNotFound] = TRUE
                                                                   h
  RETURN found
```

Find an edge where one vertex

How do we choose the <u>next</u> edge?



Two common (and well studied) options

Breadth-First Search

- Explore the graph in layers
- "Cautious" exploration
- Use a FIFO data structure (can you think of an example?)

Depth-First Search

- Explore recursively
- A more *"aggressive"* exploration (we backtrack if necessary)
- Use a LIFO data structure (or recursion)