Overview

• Depth First Search
• Topical sorting
• Intro Shortest Path Routing
Objective

- Understand and be able to apply the depth first search (DFS) algorithm
- Apply Topological Sorting as graph algorithm
- Understand the problem of Shortest Path Routing

Knight’s Tour - Analysis

- Very sensitive to method used to select next vertex
- Example
  - 5 x 5 board, calculate path in 1.5 second
  - 8 x 8 board, up to ½ hour
- Reason: $O(k^N)$, $N$ is number of squares, $k$ is small constant
Knight’s Tour - Analysis

• Root is starting point of search tree
• Then checks each move knight can make
  • 2 legal moves in corner
  • 3 in squares adjacent to corners
  • 8 in middle of board

Knight’s Tour - Analysis

• Figure shows number of possible moves on board
• Next level of tree has again 2 – 8 next possible moves
• Number of possible positions to examine corresponds to number of nodes in search tree
Knight’s Tour - Analysis

• Number of nodes in binary tree is $2^{N+1} - 1$
• Number much larger for tree with up to 8 nodes
• Use average branch factor to estimate number of child nodes: $k^{N+1} - 1$, $k$ is average branching factor
• Example:
  • 5 x 5 board, tree is 25 levels deep => $N=24$
  • $k=3.8 \Rightarrow 3.8^{25}-1 = 3.12 \times 10^{14}$

Knight’s Tour - Analysis

• Way to speed up 8 x 8 case => runs in less than 1 second
• `orderbyAvail` will be called used instead of `u.getConnections` (shown in previous code)
• Line 10 is critical one, it ensures to select vertex that has *fewest* available moves
• But why not select node that has *most* available moves?
**Knight’s Tour - Analysis**

```python
def orderByAvail(n):
    resList = []
    for v in n.getConnections():
        if v.getColor() == 'white':
            c = 0
            for w in v.getConnections():
                if w.getColor() == 'white':
                    c = c + 1
            resList.append((c, v))
    resList.sort(key=lambda x: x[0])
    return [y[1] for y in resList]
```

- Problem with using vertex with most available moves => tends to have knight visit middles squares early on
  - Easy for night to get stranded on one side of board and cannot reach other side.
- Visiting squares with fewest available moves first pushes knight to visit squares around edges
- Using intuition is called *heuristic*!
General Depth First Search

- Implementation extends graph class by adding:
  - Time instance variable and methods `dfs` and `dfsvisit`
  - `dfs` method iterates over all vertices in graph calling `dfsvisit` on white nodes
  - This ensures all nodes in graph are considered and no vertices are left out of depth first forest

```python
from Graph import Graph, Vertex
class DFSGraph(Graph):
    def __init__(self):
        super().__init__()
        self.time = 0

    def dfs(self):
        for aVertex in self:
            aVertex.setColor('white')
            aVertex.setPred(-1)
        for aVertex in self:
            if aVertex.getColor() == 'white':
                self.dfsvisit(aVertex)

    def dfsvisit(self, startVertex):
        startVertex.setColor('gray')
        self.time += 1
        startVertex.setDiscovery(self.time)
        for nextVertex in startVertex.getConnections():
            if nextVertex.getColor() == 'white':
                nextVertex.setPred(startVertex)
                self.dfsvisit(nextVertex)
        startVertex.setColor('black')
        self.time += 1
        startVertex.setFinish(self.time)
```
General Depth First Search

- DFS method starts with single vertex `startVertex` and explores all neighboring white vertices as deeply as possible.
- `dfsvisit` is almost identical to `bfsexcept`.
- `dfsvisit` uses a stack where `bfsexcept` uses queue.
  - Not visible in code but implicit for `dfsvisit`.

General Depth First Search

- Following sequence of figures illustrated DFS in action.
- Dotted lines indicate checked edges but node on other end of edge has already been added to DFS tree.
- In the code this is realized by checking that color of the other node is non-white.
General Depth First Search

- Search begins at vertex A
- Since all vertices are white algorithm visits vertex A
  1. Set color of vertex A gray => vertex is being explored
  2. Discovery time is set to 0
  3. Neighbors B and D need to be visited as well
  4. Arbitrary decision to visit adjacent nodes in alphabetic order

General Depth First Search

- Vertex B is visited next
  1. Its color is set to gray
  2. Discovery time is set to 2
  3. B is adjacent to C and D
  4. Visit vertex C next
General Depth First Search

- Visiting C brings alg. to end of branch of tree
  1. Color node gray and set discovery time to 3
  2. No adjacent vertices to C
  3. Color vertex black, set finish time to 4

General Depth First Search

- Now return to B and explore nodes adjacent to it
- Only addition vertex is D
  - Visit D and continue search
  - Results in exploring E, which has adjacent vertices B and F
  - B is already colored, thus explore F
General Depth First Search

- F has only adjacent vertex C
  - C already colored black
  - Nothing else to explore
  - Reached end of branch
- Algorithm works its way back to first node
  - Setting finish times and
  - Coloring vertices black
**General Depth First Search**

- Start and finishing times are called *parentheses property*
- All children of particular node in DFS
  - Have later discovery time than parent
  - Have earlier finish time than parent
- Figure shows final tree constructed by DFS algorithm

**General Depth First Search**

- General running time:
  - Loops in dfs run in $O(V)$, since executed once for each vertex in graph
  - Since dfsvisit only called recursively if vertex is white, loop will execute max. once for every edge in graph => $O(E)$
- Total time for DFS is $O(V+E)$
Topological Sorting

• Demonstrate that almost anything can be turned into a graph problem
• Consider problem of stirring up batch of pancakes
• Recipe: 1 egg, 1 cup of pancake mix, 1 tablespoon oil and ¾ cup of milk
• Heat griddle, mix all ingredients together, and spoon mix onto hot griddle
• When pancakes start bubbling, turn them over
• Heat up syrup

Here the process is illustrated as a graph
Topological Sorting

- Problem: Know what to do first
- Start by heating griddle or adding any of ingredients to pancake mix
- To make that decision we turn to algorithm called \textit{topological sort}

Topological Sorting

- Topological sort takes DAG and produces linear ordering of all vertices such that
  - If graph contains edge \((v,w)\) then vertex \(v\) comes before vertex \(w\).
- Other examples besides pancakes:
  - project schedules
  - Multiplying matrices
Topological Sorting

- Algorithm for Topological Sort (adaptation of DFS):
  1. Call $\text{dfs}(g)$ for some graph $g$. Main reason, call finish times for each vertex
  2. Store vertices in a list in decreasing order of finish time
  3. Return the ordered list as the result of the topological sort

Topological Sorting

- Tree constructed by DFS
Topological Sorting

- Result of applying topological sorting to graph
- Now we know exactly order in which to make pancakes

Shortest Path Problems

- When you surf the web, send email, or chat, lots of work is going on behind the scenes
- Specific details on how computer networks and the Internet work, is part of a different class
- However, we’ll learn just enough to understand another very important graph algorithm
Shortest Path Problems

• When web page is requested from server, request travels over local network on to the Internet via a router

• Request travels over Internet and eventually arrives at router that connect local area network where web server is located

• Web page travels same route back to client

traceroute to www.heise.de (193.99.144.85), 64 hops max, 52 byte packets
1 128.119.232.1 (128.119.232.1) 5.001 ms 1.859 ms 1.879 ms
2 core1-rt-xe-0-1-gw.umass.edu (128.119.0.237) 1.843 ms 1.929 ms 1.811 ms
3 border1-rt-ct-5-0-gw.umass.edu (192.80.83.102) 1.817 ms 2.185 ms
   border2-rt-ct-5-0-gw.umass.edu (192.80.83.110) 2.013 ms
4 5-2-19.ear3.newyork1.level3.net (4.71.230.233) 7.528 ms
   border2-rt-ct-4-0-1-gw.umass.edu (192.80.83.97) 2.042 ms 2.105 ms
5 5-2-19.ear3.newyork1.level3.net (4.71.230.233) 7.413 ms
   ae-1-3110.edge4.frankfurt1.level3.net (4.69.163.106) 84.297 ms
   5-2-19.ear3.newyork1.level3.net (4.71.230.233) 7.824 ms
6 ae-1-3110.edge4.frankfurt1.level3.net (4.69.163.106) 84.635 ms 84.664 ms 84.442
Shortest Path Problems

- Represent network of routers as graph with weighted edges
- Problem to solve:
  - Find path with smallest total weight
- Similar to BFS but here we are concerned with the total weight of the path, rather than number of nodes in path
- If all weight are equal, problem is the same
Next Steps

- Next lecture on Thursday: Dijkstra’s Algorithm