Dijkstra's Algorithm and Floyd-Warshall's Algorithm

Nawal Ahmed

Fall 2018

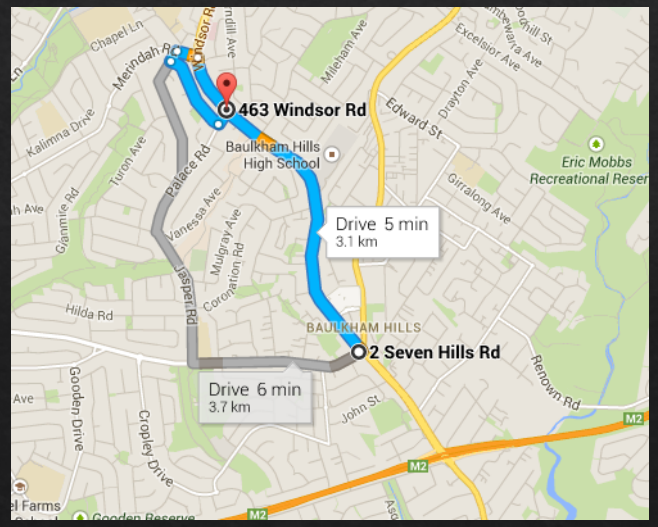
Video Presentation Link: https://youtu.be/xbQ0ewgNNKA

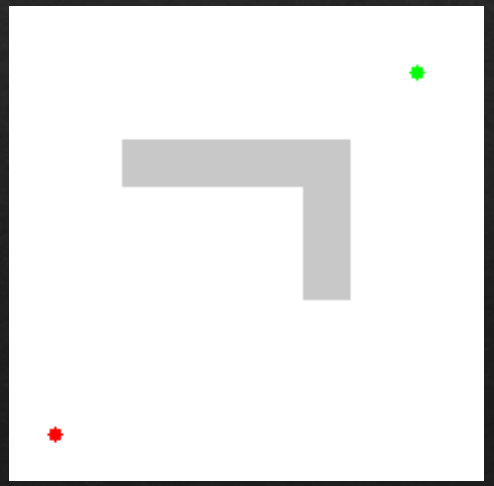
Path finding algorithms

- Both are a type of path finding algorithm tailored for its own purpose.
- Pathfinding algorithms decrease the time it takes to receive an answer instead of brute forcing it.
- Pathfinding algorithms must determine:
  - 1) A solution does exist
  - 2) A method of confirming that the chosen answer is correct.
- Used in Geographic Information Systems, Satellite Navigations Systems, Network Routing, etc.

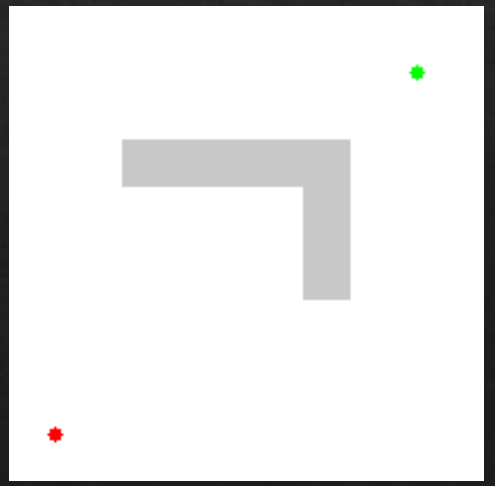
Examples

- Google Maps uses a proprietary algorithm.
- Almost certainly more advance than Dijkstra's algorithm – most likely their own version of the A* algorithm.





A*

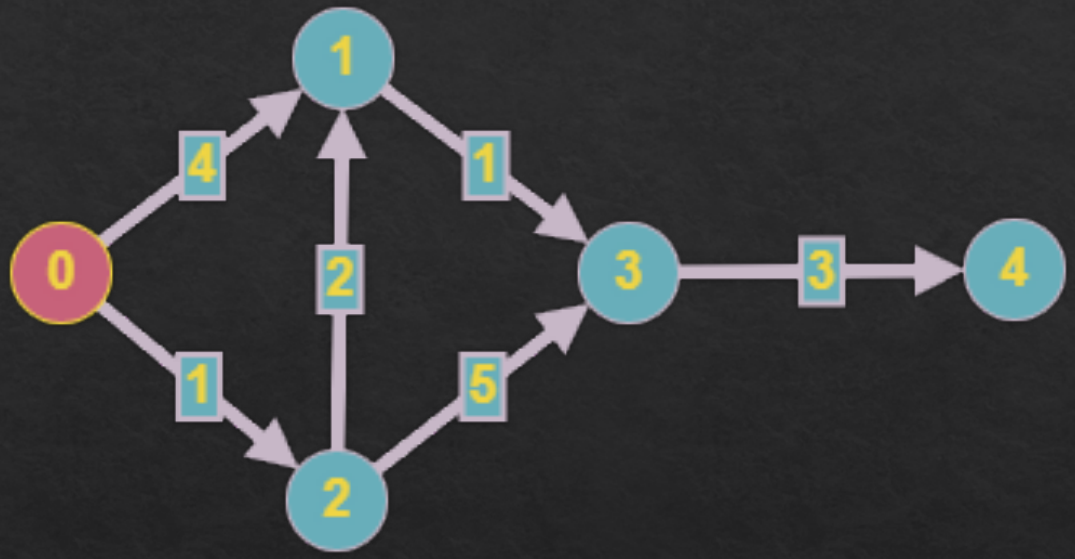


Dijkstra's Algorithm

Dijkstra's algorithm

- Invented by Edsger Dijkstra in 1956.
- Greedy algorithm, explores in increasing orders of cost.
  - Will always make the most optimal decision at the current time.
- Tells the shortest path from one node to every other node.
- Relatively flexible
  - Many implementations.
  - Many variations.
- Base (aka Lazy approach): $O(V^2)$
- Eager approach: $O(ElogV)$

Dijkstra's algorithm cont.



0	1	2	3	4
0	∞	∞	∞	∞

0	1	2	3	4
0	**4**	**1**	∞	∞

0	1	2	3	4
0	**3**	**1**	∞	∞

0	1	2	3	4
0	**3**	**1**	**4**	**7**

Select a starting node, and examine all of edges leaving the node and count their weights. Then, select the edge with the lowest weight of which the vertex that has not yet been chosen. Continue until all nodes have been visited.

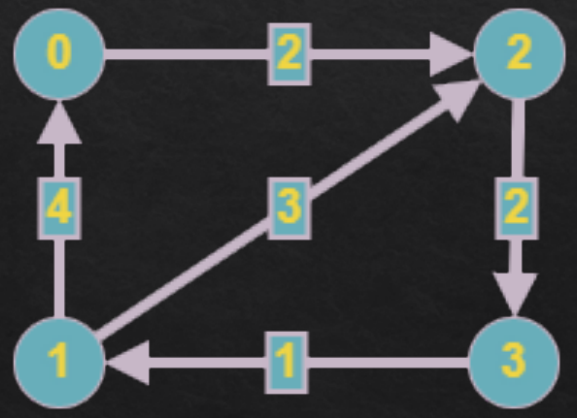
Floyd-Warshall's algorithm

- Invented coincidentally in the same year 1962 by Robert Floyd and Stephen Warshall.
  - Independent from each other, yet very similar.
- All-Pairs Shortest Path algorithm.
  - Gradually build up intermediate routes between two select nodes for the optimal path.
- Find the shortest path between all pairs of nodes, and negative edges are allowed.
  - None allow negative cycles.
- Not as flexible.
- $O(V^3)$

Stephen Warshall

Floyd-Warshall's algorithm cont.

- There are 3 counter variables to be aware of: i,j, and k.
- The counters are used to go through each square of a matrix and determine if it needs updating.
- if matrix[i][j] > matrix[i][k] + matrix[k][j]
  - matrix[i][j] ← matrix[i][k] + matrix[k][j]



	0	1	2	3
0	0		2	
1	4	0	3	
2			0	2
3		1		0

	0	1	2	3
0	0	0	2	0
1	4	0	3	0
2	0	0	0	2
3	0	1	0	0

Implementation

	Dijkstra's Algorithm	**Floyd-Warshall's Algorithm**
Time Complexity (Base)	$O(V^2)$	$O(V^3)$
Graph Size	Medium-Large	Small
All-Pairs Shortest Path	OK	Good
Shortest Path	Good	Bad
Implementation	Depends	Good

Program

```
Node--Dist
0------0
1------3
2------8
3------4
4------8
5------7
6------6
7------4
8------14
9------14
10------13
11------10
12------5

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Left is Dijkstra and bottom is Floyd-Warshall.
Same graph. Dijkstra took half the time.
Floyd-Warshall creates a table of information. Dijkstra creates a specific list.

*note: In the video commentary, I make an error in saying "Dijkstra is half as fast." What I meant to say was Dijkstra took half time, meaning it was twice as fast. The disparity in time increases the larger the graph is.

```
        0       3       8       4       7       7       6       4       14      14      12      10      5
        3       0       5       4       4       4       3       1       11      11      9       7       2
        8       5       0       9       9       9       8       6       6       6       10      2       3
        4       4       9       0       5       5       7       5       14      14      10      11      6
        4       4       9       0       0       5       2       4       9       9       5       11      6
        4       4       9       0       0       0       2       3       9       9       5       11      6
        6       3       8       2       2       5       0       2       11      11      7       10      5
        4       1       6       3       3       3       2       0       12      12      8       8       3
        13      11      6       9       9       14      11      12      0       6       4       4       9
        13      11      6       9       9       14      11      12      6       0       4       4       9
        9       9       10      5       5       10      7       9       4       4       0       8       11
        10      7       2       11      11      11      10      8       4       4       8       0       5
        5       2       3       6       6       6       5       3       9       9       11      5       0

Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.
```

Questions?