Centre for Information Technology and Electronics

DTU Diplom
Lautrupvang 15, 2750 Ballerup

Dijkstra Graph Traversal

Finding shortest paths to all vertices in a non-directional, weighted graph.

Supervisor(s): **Roger Munck-Fairwood**

Student name	**Student number**	**Signature**
James Testmann	S071954	
Aimo Suikkanen	S082577	
Lukas Janocko	S135272	
Shicheng Dai	S133342	
Sudhir Kumar Chaurasiya	S137239	

Title

Graphs: shortest path

Perspective

Modeling problems from the real physical world will in many cases call for a graph, such as:

- Find the cheapest plane ticket from Copenhagen to Rome.
- Find the quickest route from Lautrupvang to Tivoli by bike.
- Determine the cheapest way to supply pipes for fresh water to houses under construction.
- Find the most efficient route through a complex data network.
- In a speech recognition system find the most likely word in a stream of spoken words.

The choice of implementation of a graph together with associated algorithms is highly dependent on the problem to be solved. Some attempts have been made to supply graph libraries to toolboxes of different compilers, but they are often over-killed with functionality and very often cannot satisfy your actual needs.

Answer: do it yourselves!

Knowledge of some very basic graph-searching algorithms is mandatory. They are fundamental to most of the more advanced algorithms.

Assignment

Dijkstra's algorithm is a graph traversal method for finding the shortest path from one vertex to any other non-isolated vertex.

A graph is any set of objects, some or all of which may be connected by links (objects and links are called "vertices" and "edges" respectively). It is a supertype of other structures like trees.

If two vertices are connected by an edge, this edge may be directional or not, weighted or not, or even contain loops and multiple paths between the same verts (called a "quiver" graph).

In this assignment we use a weighted, non-directional graph with no loops and no isolated subsets.

Each vertex starting with the graph's "start vertex" is visited deterministically (the vertex with the currently lowest score). Then all non-visited neighbours of the visited vertex are given a score which is updated only if it is greater than the current path to that neighbour. We can describe it in pseudo-code as such:

```
if (neighbour's score > (current score + "cost" to neighbour))
{
        neighbour's score = current score + "cost" to neighbour
} else
{
        neighbour's score = neighbour's score.
}
```

Alongside a score - each vertex also has a return value. This value is the vertex to backtrack to, to get the shortest path.

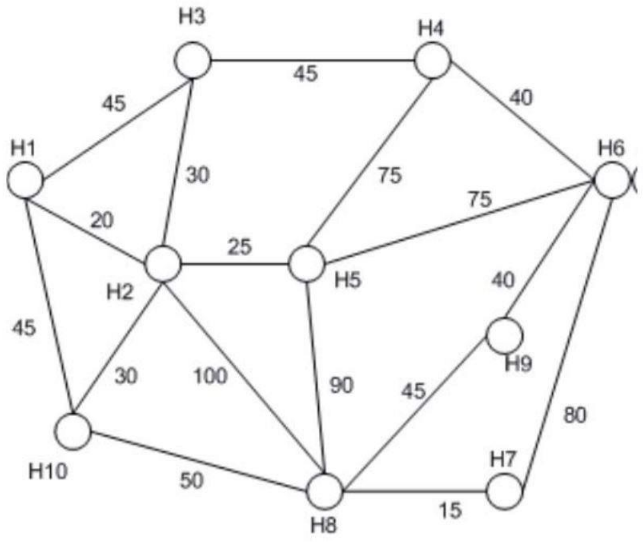


Figure 1- Connection graph (e.g. connections between houses or transport stops)

Run Dijkstra's Shortest Path algorithm by hand on the figure above.

- *Use H1 and H5 as two different source (starting) vertices (and others, if you wish).*

Start H1	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
Dist	**0**									
Pred										
Dist		**20**	45							45
Pred		**H1**	H1							H1
Dist			**45**		45			120		45
Pred			**H1**		H2			H2		H1
Dist				90	**45**			120		45
Pred				H3	**H2**			H2		H1
Dist				90		120		120		**45**
Pred				H3		H5		H2		**H1**
Dist				**90**		120		95		
Pred				**H3**		H5		H10		
Dist						120		**95**		
Pred						H5		**H10**		
Dist						120	**110**		140	
Pred						H5	**H8**		H8	
Dist						**120**			140	
Pred						**H5**			H8	
Dist									**140**	
Pred									**H8**	

Goal	Cost	Path (List of vertices)		
H1	0			
H2	20	H1		
H3	45	H1		
H4	90	H3	H1	
H5	45	H2	H1	
H6	120	H5	H2	H1
H7	110	H8	H10	H1
H8	95	H10	H1	
H9	140	H8	H10	H1
H10	45	H1		

Start/Source H5	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
Dist					**0**					
Pred										
dist		**25**		75		75		90		
pred		**H5**		H5		H5		H5		
dist	**45**		55	75		75		90		55
pred	**H2**		H2	H5		H5		H5		H2
dist			**55**	75		75		90		55
pred			**H2**	H5		H5		H5		H2
dist				75		75		90		**55**
pred				H5		H5		H5		**H2**
dist				**75**		75		90		
pred				**H5**		H5		H5		
dist						**75**		90		
pred						**H5**		H5		
dist							155	**90**	115	
pred							H6	**H5**	H6	
dist							**105**		115	
pred							**H8**		H6	
dist									**115**	
pred									**H6**	

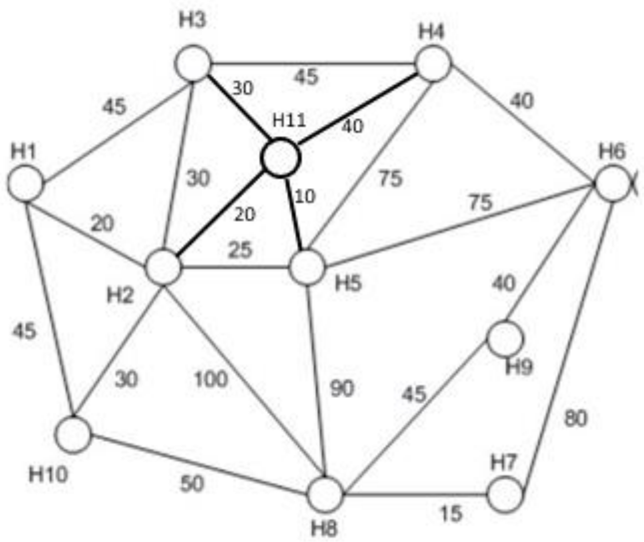
Goal	Cost	Path (List of vertices)		
H1	45	H2	H5	
H2	25	H5		
H3	55	H2	H5	
H4	75	H5		
H5	0			
H6	75	H5		
H7	105	H8	H5	
H8	90	H5		
H9	115	H6	H5	
H10	55	H2	H5	

- *The connection between H8 and H10 is now disconnected. Run the algorithm again with H1 as the source vertex.*

(no connection between H8 and H10)										
Start H1	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
Dist	**0**									
Pred										
Dist		**20**	45							45
Pred		**H1**	H1							H1
Dist			**45**		45			120		45
Pred			**H1**		H2			H2		H1
Dist				90	**45**			120		45
Pred				H3	**H2**			H2		H1
Dist				90		120		120		**45**
Pred				H3		H5		H2		**H1**
Dist				**90**		120		120		
Pred				**H3**		H5		H2		
Dist						**120**		120		
Pred						**H5**		H2		
Dist							200	**120**	130	
Pred							H6	**H2**	H6	
Dist							135		**130**	
Pred							H8		**H6**	
Dist							**135**			
Pred							**H8**			

Goal	Cost	Path (List of vertices)			
H1	0				
H2	20	H1			
H3	45	H1			
H4	90	H3	H1		
H5	45	H2	H1		
H6	120	H5	H2	H1	
H7	135	H8	H2	H1	
H8	120	H2	H1		
H9	130	H6	H5	H2	H1
H10	45	H1			

- *An extra vertex (H11) is added.*



Start H1	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11
dist	**0**										
pred											
dist		**20**	45							45	
pred		**H1**	H1							H1	
dist			45		45			120		45	**40**
pred			H1		H2			H2		H1	**H2**
dist			**45**	80	45			120		45	
pred			**H1**	H11	H2			H2		H1	
dist				80	**45**			120		45	
pred				H11	**H2**			H2		H1	
dist				80		120		120		**45**	
pred				H11		H5		H2		**H1**	

Centre for Information Technology and Electronics

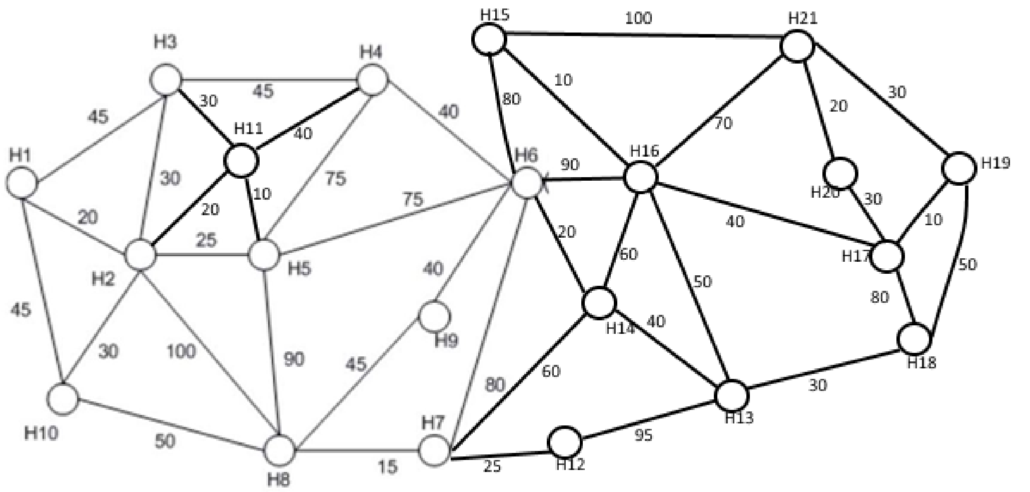
DTU Diplom

Lautrupvang 15, 2750 Ballerup

dist				**80**		120		95			
pred				**H11**		H5		H10			
dist						120		**95**			
pred						H5		**H10**			
dist						120	**110**		140		
pred						H5	**H8**		H8		
dist						**120**			140		
pred						**H5**			H8		
dist									**140**		
pred									**H8**		

Goal	Cost	Path (List of vertices)		
H1	0			
H2	20	H1		
H3	45	H1		
H4	80	H11	H2	H1
H5	45	H2	H1	
H6	120	H5	H2	H1
H7	110	H8	H10	H1
H8	95	H10	H1	
H9	140	H8	H10	H1
H10	45	H1		
H11	40	H2	H1	

- *The table is now expanded noticeably*



Centre for Information Technology and Electronics

DTU Diplom
Lautrupvang 15, 2750 Ballerup

Start H1	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12	H13	H14	H15	H16	H17	H18	H19	H20	H21
dist	**0**																				
pred																					
dist		**20**	45							45											
pred		**H1**	H1							H1											
dist			45		45			120		45	**40**										
pred			H1		H2			H2		H1	**H2**										
dist			**45**	80	45			120		45											
pred			**H1**	H11	H2			H2		H1											
dist				80	**45**			120		45											
pred				H11	**H2**			H2		H1											
dist				80		120		120		**45**											
pred				H11		H5		H2		**H1**											
dist				**80**		120		95													
pred				**H11**		H5		H10													
dist						120		**95**													
pred						H5		**H10**													
dist						120	**110**		140												
pred						H5	**H8**		H8												
dist						**120**			140			135		170							
pred						**H5**			H8			H7		H7							
dist									140			**135**		140	200	210					
pred									H8			**H7**		H6	H6	H6					
dist									**140**				230	140	200	210					
pred									**H8**				H12	H6	H6	H6					
dist													230	**140**	200	210					
pred													H12	**H6**	H6	H6					
dist													**180**		200	200					
pred													**H14**		H6	H14					
dist															**200**	200		210			
pred															**H6**	H14		H13			
dist																**200**		210			300
pred																**H14**		H13			H15
dist																	240	**210**		220	270
pred																	H16	**H13**		H16	H16
dist																	240		260	**220**	270
pred																	H16		H18	**H16**	H16
dist																	**240**		260		240
pred																	**H16**		H18		H20
dist																			250		**240**
pred																			H17		**H20**
dist																			**250**		
pred																			**H17**		

Goal	Cost	Path (List of vertices)						
H1	0							
H2	20	H1						
H3	45	H11	H2	H1				
H4	80	H11	H2	H1				
H5	45	H2	H1					
H6	120	H5	H2	H1				
H7	110	H8	H10	H1				
H8	95	H10	H1					
H9	140	H8	H10	H1				
H10	45	H1						
H11	40	H2	H1					
H12	135	H7	H8	H10	H1			
H13	180	H14	H6	H5	H2	H1		
H14	140	H6	H5	H2	H1			
H15	200	H6	H5	H2	H1			
H16	200	H14	H6	H5	H2	H1		
H17	240	H16	H14	H6	H5	H2	H1	
H18	210	H13	H14	H6	H5	H2	H1	
H19	250	H17	H16	H14	H6	H5	H2	H1
H20	220	H16	H14	H6	H5	H2	H1	
H21	240	H20	H16	H14	H6	H5	H2	H1

- *What is the complexity (in big-O notation) of the Dijkstra algorithm?*
  Hint: The implementation of the priority queue may influence your answer.
  *O(|E|+|V|*log|V|)*

To find the efficiency of Dijkstra we need to consider the operations necessary to traverse the graph.

1. For each vertex |V| extract the smallest value.
2. For the extracted vertex, read values of all neighbours |E|.
3. Decrease the values of each neighbour when applicable.

We can then write the efficiency as:

O(|E| * |decrease-values(Q)| + |V| * |extract-min(Q)|)

Where Q is the time it takes for the queue or array to handle the operation.

For an unsorted array, the extract-min operation has a complexity of O(|V|). Each vertex of the graph has to be explored to find the smallest occurrence. The decrease values operation is seen as O(1) because of the fast memory operations with a known index in an array.

Centre for Information Technology and Electronics

DTU Diplom
Lautrupvang 15, 2750 Ballerup

This combined makes an unsorted array's efficiency:

O(|E| + |V|^2)

For a sorted queues we find different efficiencies depending on queue-type implementations.

For Fibonacci and binary heap queues the extract-min operations are:[i]

Binary Heap: O(log |V|)
Fibonacci: O(log |V|)

The increase-value operations are:

Binary Heap: O(log(|V|)
Fibonacci: O(1)

Which makes the efficiency:

Binary Heap: O((|E| + |V|) * log(|V|))
Fibonacci: O(|E| + |V| * log(|V|))

[i] Efficiency numbers collected from http://bigocheatsheet.com/