COMP20230: Data Structures & Algorithms
Lecture 17: Graphs (2)

Dr Andrew Hines

Office: E3.13 Science East
School of Computer Science
University College Dublin

andrew.hines@ucd.ie

Outline

Today

- Graph Traversal: DFS and BFS Complexity
- Weighted Graphs
- Shortest Path – Dijkstra's Algorithm

Take home message

Graphs can be weighted along edges to better represent and model some scenarios.
Dijkstra's algorithm is a shortest path algorithm.

Recap: Graphs

A collection of vertices (nodes), joined together by edges.
No definite beginning or end.

Representations of graphs

Two common methods: adjacency lists and adjacency matrices

Searching

Many algorithms begin by searching (e.g. DFS, BFS) their input graph to obtain this structural information.

BFS Complexity

Worst case is you need to traverse and explore **every** vertex and **every** edge.

Time Complexity for $|V|$ vertices and $|E|$ edges

$\mathcal{O}(|V| + |E|)$

where, depending on sparsity, $\mathcal{O}(|E|)$ is between $\mathcal{O}(1)$ and $\mathcal{O}(|V|^2)$

Breadth-first search is **complete**, but depth-first search is not, i.e. the input to breadth-first search is assumed to be a finite graph, represented explicitly (e.g. by an adjacency list).

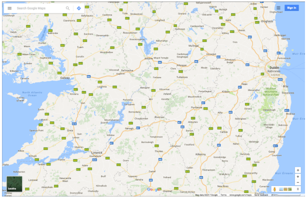
Weighted Graphs

Sometimes graphs have a weights on edges.

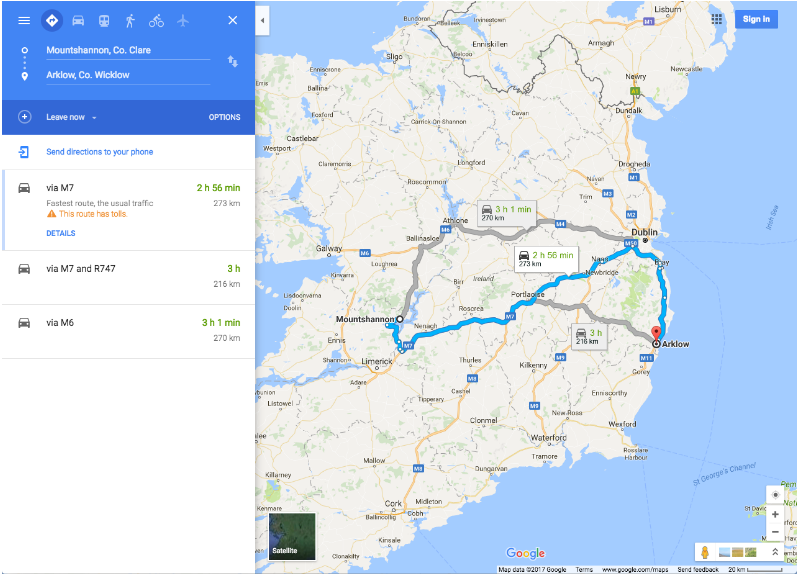
Examples

A road network with the distance between two cities

A computer network – hops between routers for a streaming video packet with time between nodes (e.g. think `ping` and `traceroute`).



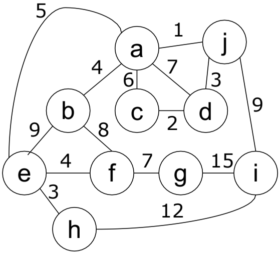
Routing



Mountshannon, Co. Clare
Arklow, Co. Wicklow
Leave now
OPTIONS
Send directions to your phone
via M7
2 h 56 min
Fastest route, the usual traffic
273 km
This route has tolls.
DETAILS
via M7 and R747
3 h
216 km
via M6
3 h 1 min
270 km
Sign in
Dublin
Mountshannon
Arklow
Galway
Limerick
Athlone
Portlaoise
Cork
Waterford
Wexford
Kilkenny
3 h 1 min
270 km
2 h 56 min
273 km
3 h
216 km
Satellite
Map data ©2017 Google

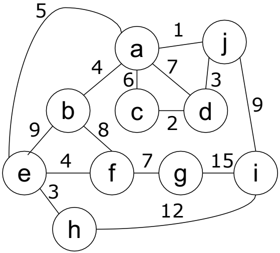
Distance Matrix (Jacob Stockdale, 1805)

Weighted Graph Adjacency List



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Weighted Graph Adjacency Matrix



	a	b	c	d	e	f	g	h	i	j
a		4	6	7	5					1
b	4				9	8				
c	6			2						
d	7		2							3
e	5	9				4		3		
f		8			4		7			
g						7			15	
h					3				12	
i							15	12		9
j	1			3					9	

Unweighted Graph Shortest Path

Shortest Path – Simple Case

How would we calculate the shortest path if there were no weights?

Unweighted Graph Shortest Path

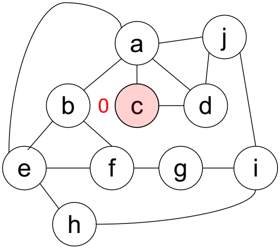
Shortest Path – Simple Case

How would we calculate the shortest path if there were no weights?

Ignore weights on edges

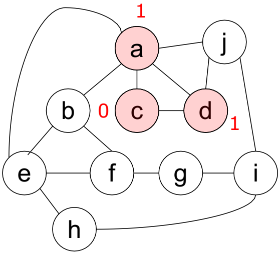
Breadth First Search will give us shortest path

Breadth First Search – Unweighted Shortest Path



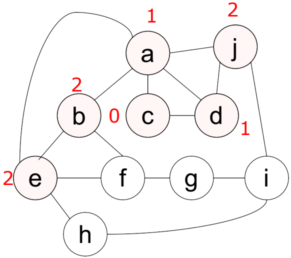
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Breadth First Search – Unweighted Shortest Path



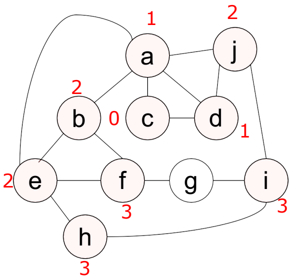
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Breadth First Search – Unweighted Shortest Path



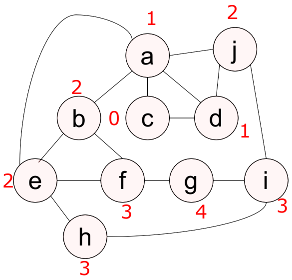
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Breadth First Search – Unweighted Shortest Path



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Breadth First Search – Unweighted Shortest Path



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Searching a weighted graph

Dijkstra's algorithm

Find the shortest paths between nodes in a graph.

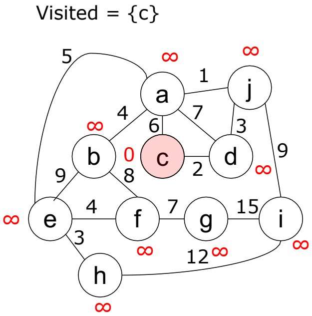
Dijkstra Variations

Original is to finds the shortest path between two given nodes
Variant fixes a vertex as the origin and finds shortest paths to all other nodes in the graph (a shortest-path tree)

Dijkstra's Algorithm

- Pick start and end vertices (from c to g)
- Mark start vertex as 0 distance and all other vertices at ∞
- Repeat until we reach end vertex:
  1. Calculates total distance to each unvisited neighbour for the current vertex
  2. Update neighbour's distance if smaller
  3. Pick the vertex with the lowest marked distance and set as current
  4. Mark current as visited

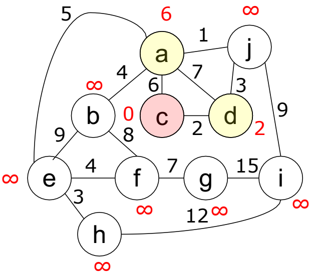
Dijkstra's Algorithm



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

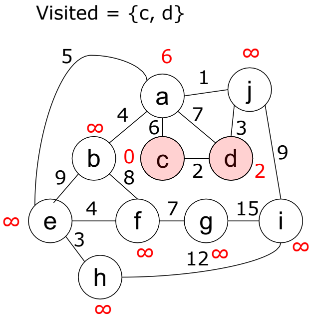
Dijkstra's Algorithm

Visited = {c}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

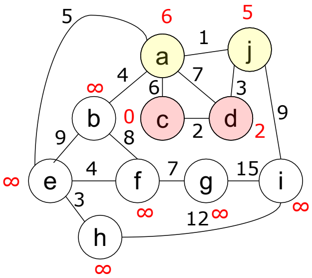
Dijkstra's Algorithm



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Dijkstra's Algorithm

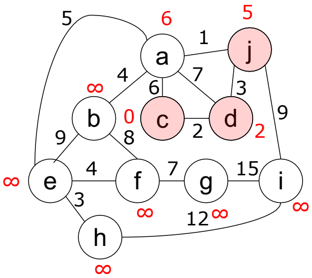
Visited = {c, d}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

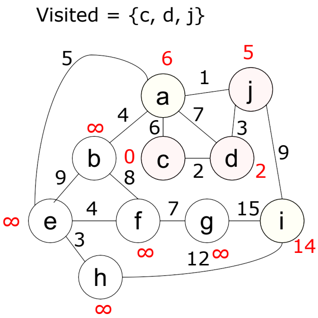
Dijkstra's Algorithm

Visited = {c, d, j}



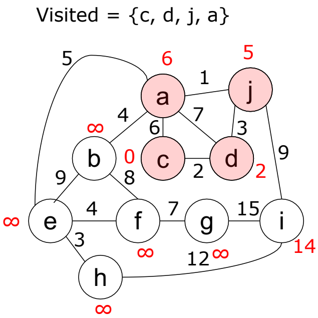
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Dijkstra's Algorithm



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

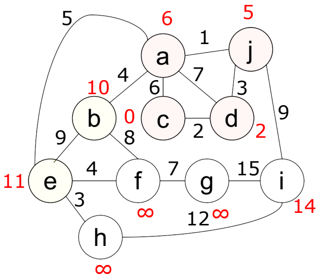
Dijkstra's Algorithm



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Dijkstra's Algorithm

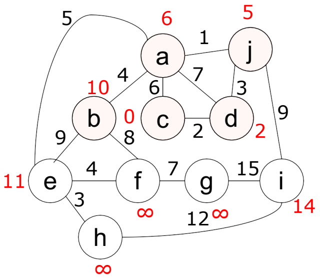
Visited = {c, d, j, a}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

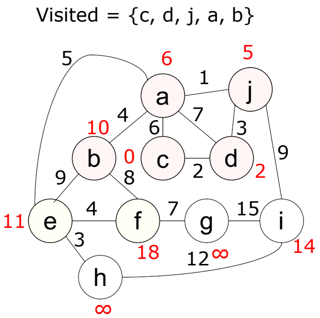
Dijkstra's Algorithm

Visited = {c, d, j, a, b}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

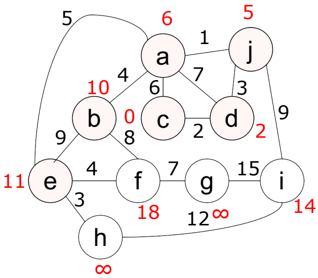
Dijkstra's Algorithm



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Dijkstra's Algorithm

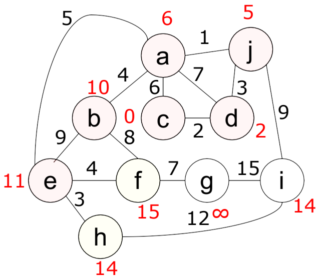
Visited = {c, d, j, a, b, e}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

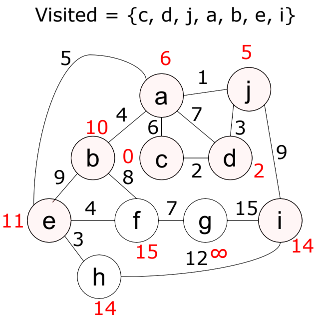
Dijkstra's Algorithm

Visited = {c, d, j, a, b, e}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

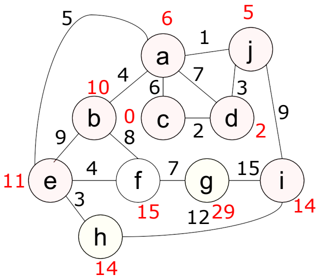
Dijkstra's Algorithm



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Dijkstra's Algorithm

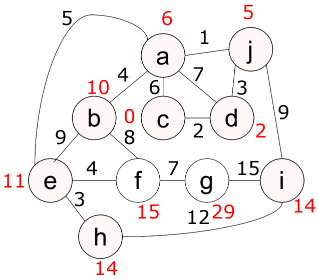
Visited = {c, d, j, a, b, e, i}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Dijkstra's Algorithm

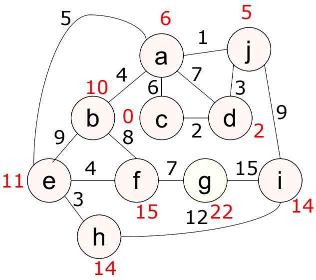
Visited = {c, d, j, a, b, e, i, h}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Dijkstra's Algorithm

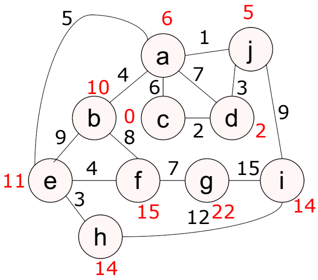
Visited = {c, d, j, a, b, e, i, h, f}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Dijkstra's Algorithm

Visited = {c, d, j, a, b, e, i, h, f, g}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Dijkstra's Algorithm

```
function dijkstra:
Input:  a graph G, a node n
Output:  each node of G gets the shortest distance from n
for each node c of G not n do
    distance[c] ← infinity
endfor
distance[n] ← 0
current ← n
visited ← {n}
while visited does not contain all nodes of G do
    for each node c neighbour of current and not visited do
        # is new route to neighbour better?  min(|n|+|n--c| and |c|)
        distance[c] ← min(distance[n] + weight(current, c),
distance[c])
    endfor
    # set next node to search from - lowest of all non-visited
    current ← non visited node with smallest value
    add current to visited
endwhile
```

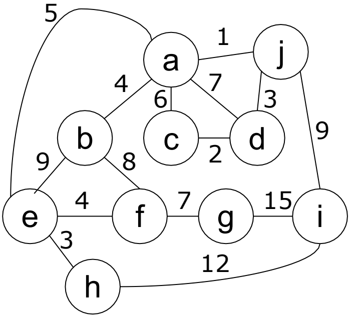
Dijkstra Steps

Shortest Path Tree

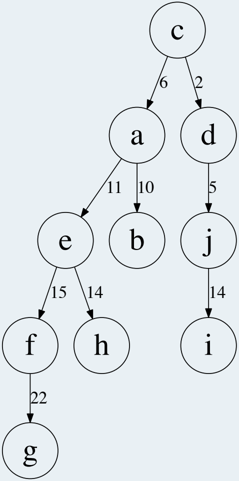
- Visited nodes are in the rows
- Distance from node c in columns for visited nodes
- Best distance from c to a column's node is highlighted in bold

	a	b	c	d	e	f	g	h	i	j
c	6(c-a)	∞		**2(c-d)**	∞	∞	∞	∞	∞	∞
d	6(c-a)	∞			∞	∞	∞	∞	∞	**5(c-d-j)**
j	**6(c-a)**	∞			∞	∞	∞	∞	14(c-d-j-i)	
a		**10(c-a-b)**			11(c-a-e)	∞	∞	∞	14(c-d-j-i)	
b					**11(c-a-e)**	18(c-a-b-f)	∞	∞	14(c-d-j-i)	
e						15(c-a-e-f)	∞	14(c-a-e-h)	**14(c-d-j-i)**	
i						15(c-a-e-f)	29(c-d-j-i)	**14(c-a-e-h)**		
h						**15(c-a-e-f)**	29(c-d-j-i)			
f							**22(c-a-e-f-g)**			

Dijkstra: Shortest Path Tree

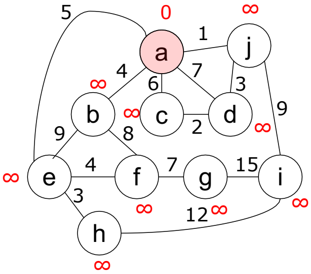


Shortest Path Tree



Exercise: From A to G?

Visited = {a}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Summary

Complexity

Basic Dijkstra is $\mathcal{O}(|V|^2)$
Can be optimised to $\mathcal{O}(|E| + |V| \log |V|)$

Graphs can have weighted edges to better represent and model some scenarios.
Dijkstra's algorithm is a shortest path algorithm to traverse weighted edges.