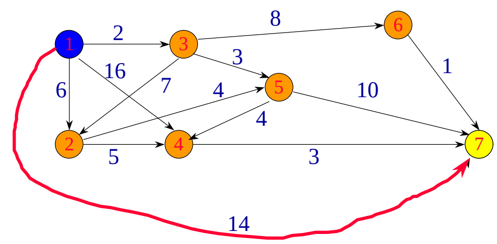
Shortest Path Problems

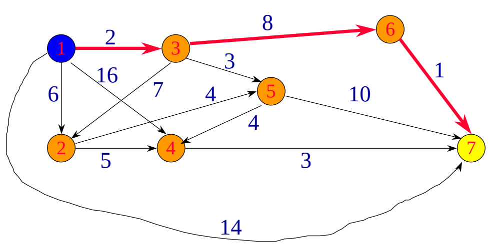
- Directed weighted graph.
- Path length is sum of weights of edges on path.
- The vertex at which the path begins is the source vertex.
- The vertex at which the path ends is the destination vertex.

Example



A path from 1 to 7.
Path length is 14.

Example



Another path from 1 to 7.
Path length is 11.

Shortest Path Problems

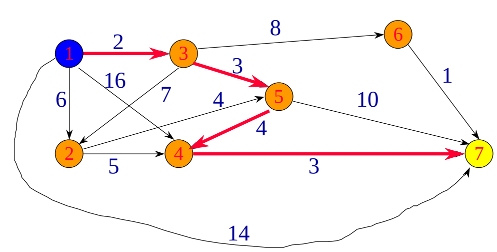
- Single source single destination.
- Single source all destinations.
- All pairs (every vertex is a source and destination).

Single Source Single Destination

Possible greedy algorithm:

- Leave source vertex using cheapest/shortest edge.
- Leave new vertex using cheapest edge subject to the constraint that a new vertex is reached.
- Continue until destination is reached.

Greedy Shortest 1 To 7 Path



Path length is 12.

Not shortest path. Algorithm doesn't work!

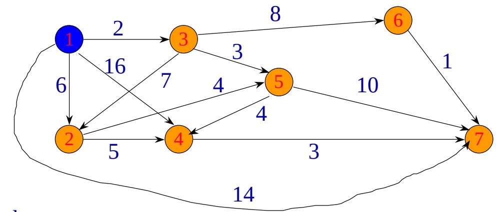
Single Source All Destinations

Need to generate up to n (n is number of vertices) paths (including path from source to itself).

Greedy method:

- Construct these up to n paths in order of increasing length.
- Assume edge costs (lengths) are >= 0.
- So, no path has length < 0.
- First shortest path is from the source vertex to itself. The length of this path is 0.

Greedy Single Source All Destinations



Path	Length
1	0
1 → 3	2
1 → 3 → 5	5
1 → 2	6
1 → 3 → 5 → 4	9
1 → 3 → 6	10
1 → 3 → 6 → 7	11

Greedy Single Source All Destinations

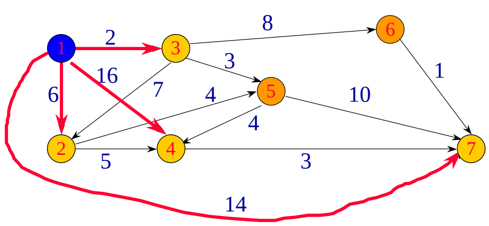
Path	Length
1	0
1 → 3	2
1 → 3 → 5	5
1 → 2	6
1 → 3 → 5 → 4	9
1 → 3 → 6	10
1 → 3 → 6 → 7	11

- Each path (other than first) is a one edge extension of a previous path.
- Next shortest path is the shortest one edge extension of an already generated shortest path.

Greedy Single Source All Destinations

- Let d(i) (distanceFromSource(i)) be the length of a shortest one edge extension of an already generated shortest path, the one edge extension ends at vertex i.
- The next shortest path is to an as yet unreached vertex for which the d() value is least.
- Let p(i) (predecessor(i)) be the vertex just before vertex i on the shortest one edge extension to i.

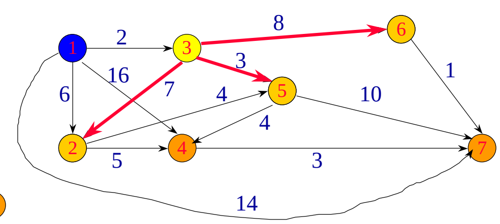
Greedy Single Source All Destinations



1

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	16	-	-	14
p	-	1	1	1	-	-	1

Greedy Single Source All Destinations

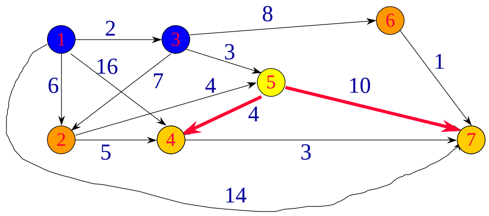


1

1 → 3

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	16	5	10	14
p	-	1	1	1	3	3	1

Greedy Single Source All Destinations



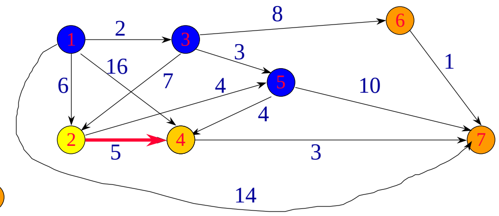
1

1 → 3

1 → 3 → 5

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	9	5	10	14
p	-	1	1	5	3	3	1

Greedy Single Source All Destinations



1

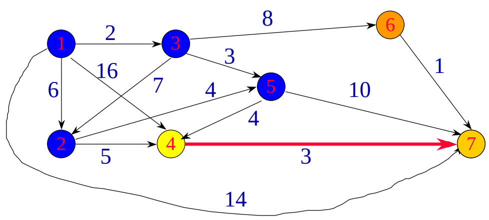
1 → 3

1 → 3 → 5

1 → 2

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	9	5	10	14
p	-	1	1	5	3	3	1

Greedy Single Source All Destinations



1

1 → 3

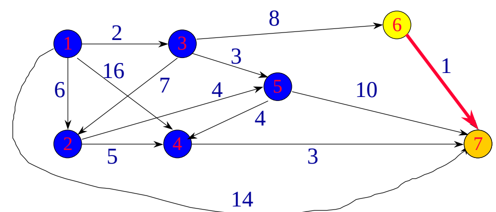
1 → 3 → 5

1 → 2

1 → 3 → 5 → 4

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	9	5	10	12
p	-	1	1	5	3	3	4

Greedy Single Source All Destinations



1 → 3 → 6

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	9	5	10	11
p	-	1	1	5	3	3	6

Greedy Single Source All Destinations

Path	Length
1	0
1 → 3	2
1 → 3 → 5	5
1 → 2	6
1 → 3 → 5 → 4	9
1 → 3 → 6	10
1 → 3 → 6 → 7	11

[1]	[2]	[3]	[4]	[5]	[6]	[7]
0	6	2	9	5	10	11
-	1	1	5	3	3	6

Single Source Single Destination

Terminate single source all destinations greedy algorithm as soon as shortest path to desired vertex has been generated.

Data Structures For Dijkstra's Algorithm

- The greedy single source all destinations algorithm is known as Dijkstra's algorithm.
- Implement d() and p() as 1D arrays.
- Keep a linear list L of reachable vertices to which shortest path is yet to be generated.
- Select and remove vertex v in L that has smallest d() value.
- Update d() and p() values of vertices adjacent to v.

Complexity

- O(n) to select next destination vertex.
- O(out-degree) to update d() and p() values when adjacency lists are used.
- O(n) to update d() and p() values when adjacency matrix is used.
- Selection and update done once for each vertex to which a shortest path is found.
- Total time is $O(n^2 + e) = O(n^2)$.

Complexity

- When a min heap of d() values is used in place of the linear list L of reachable vertices, total time is O((n+e) log n), because O(n) remove min operations and O(e) change key (d() value) operations are done.
- When e is $O(n^2)$, using a min heap is worse than using a linear list.
- When a Fibonacci heap is used, the total time is O(n log n + e).