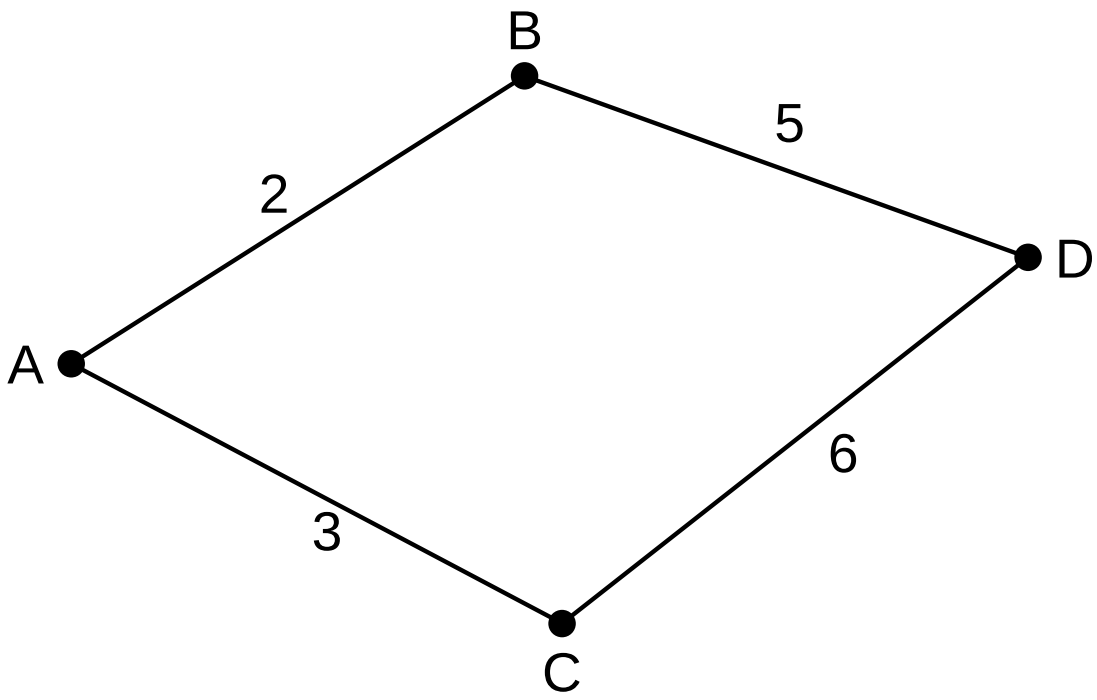
Episode 22 – The shortest path problem

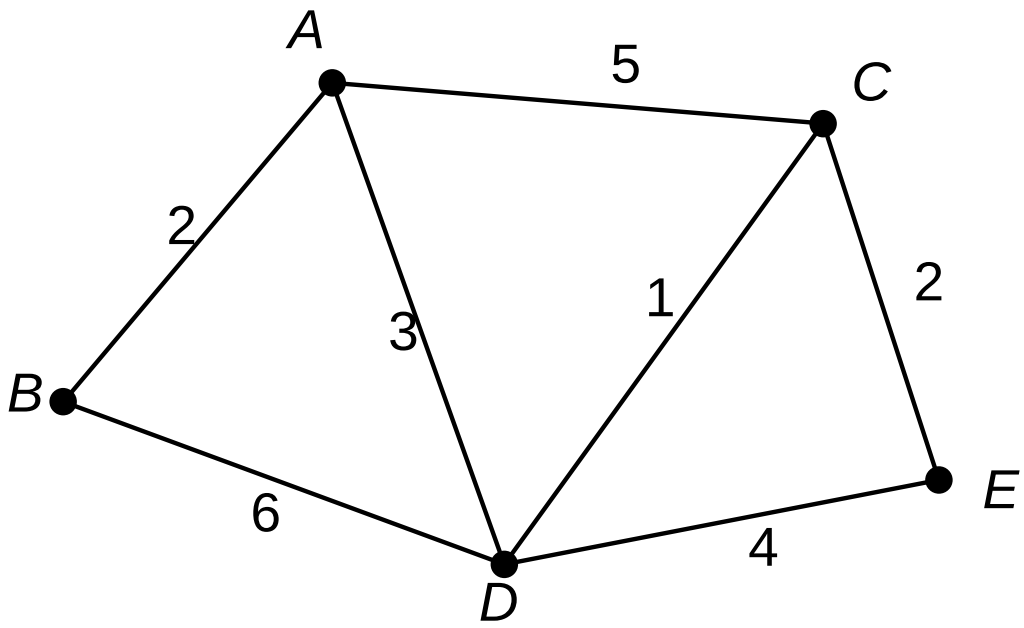
European section – Season 2

A weighted graph

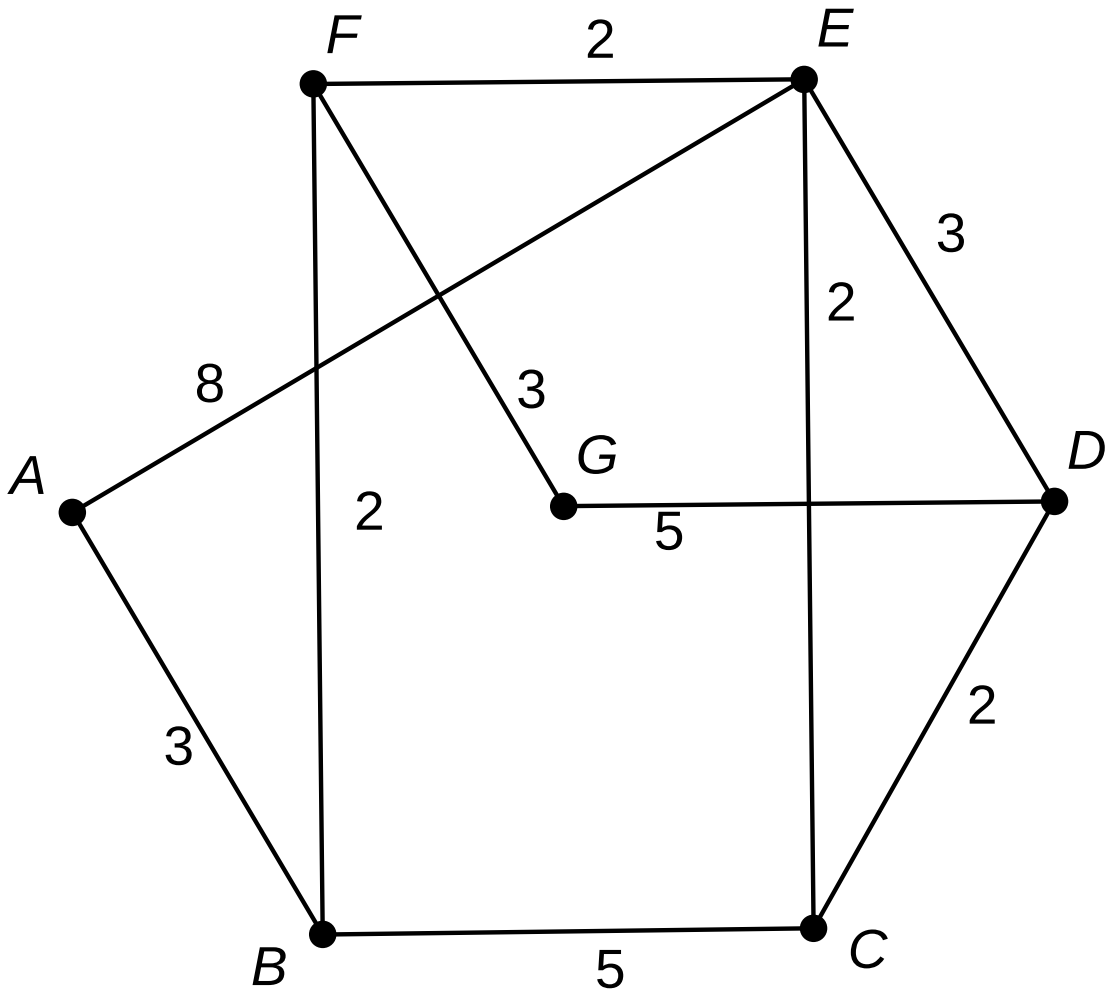


The shortest path problem

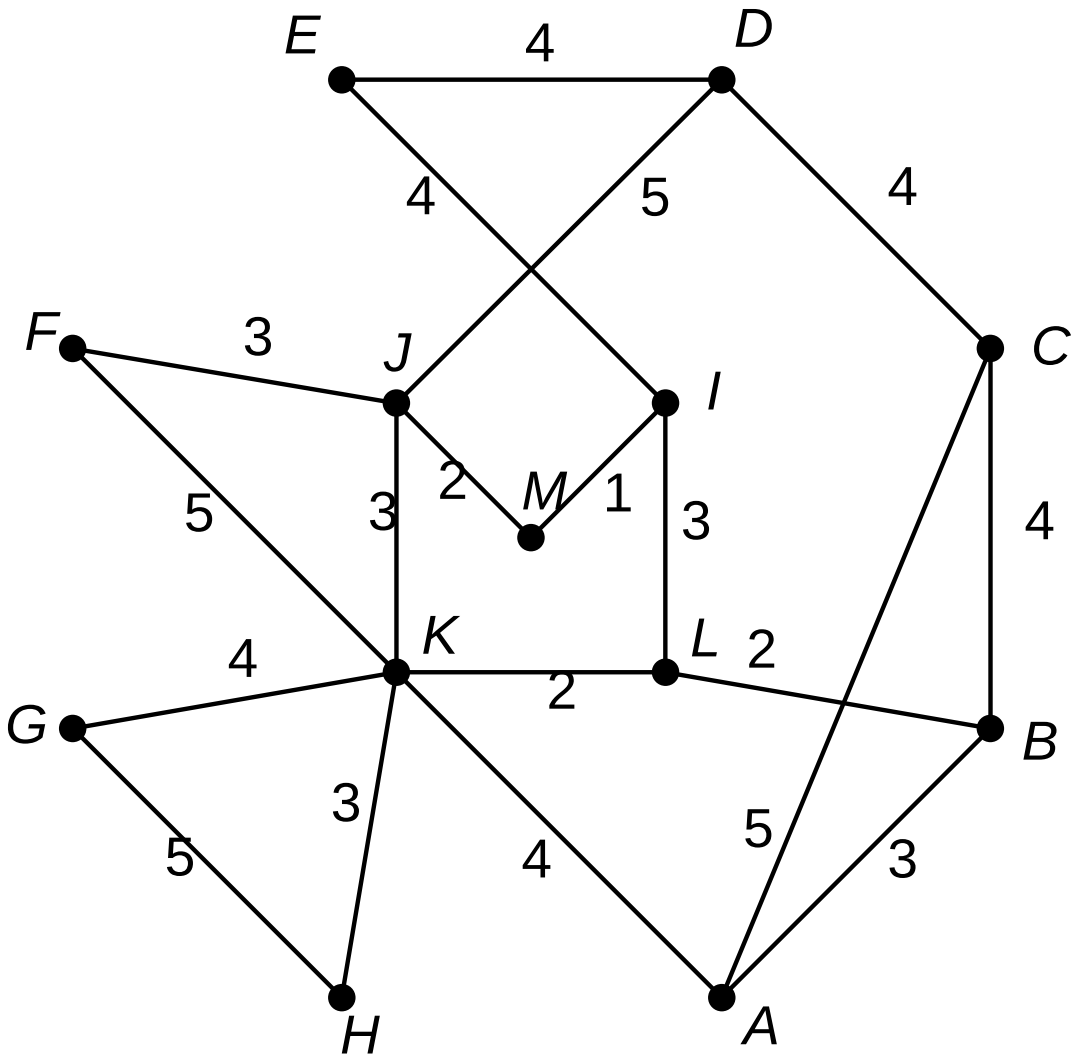
Task sheet #1



Task sheet #1



Task sheet #1



Dijkstra's algorithm

Vertex we are starting from : *initial vertex*.
Distance of a vertex Y : distance from the initial vertex to it.

Dijkstra's algorithm

Vertex we are starting from : *initial vertex*.
Distance of a vertex Y : distance from the initial vertex to it.

1. Assign to every vertex a distance value : 0 for the initial vertex and ∞) for all other vertices.

Dijkstra's algorithm

Vertex we are starting from : *initial vertex*.
Distance of a vertex Y : distance from the initial vertex to it.

1. Assign to every vertex a distance value : 0 for the initial vertex and ∞) for all other vertices.
2. Mark all vertices as unvisited. Set initial vertex as current.

Dijkstra's algorithm

Vertex we are starting from : *initial vertex*.
Distance of a vertex Y : distance from the initial vertex to it.

1. Assign to every vertex a distance value : 0 for the initial vertex and ∞) for all other vertices.
2. Mark all vertices as unvisited. Set initial vertex as current.
3. For current node, consider all its unvisited neighbours and calculate their distance. If this distance is less than the previously recorded distance, overwrite the distance.

Dijkstra's algorithm

Vertex we are starting from : *initial vertex*.
Distance of a vertex Y : distance from the initial vertex to it.

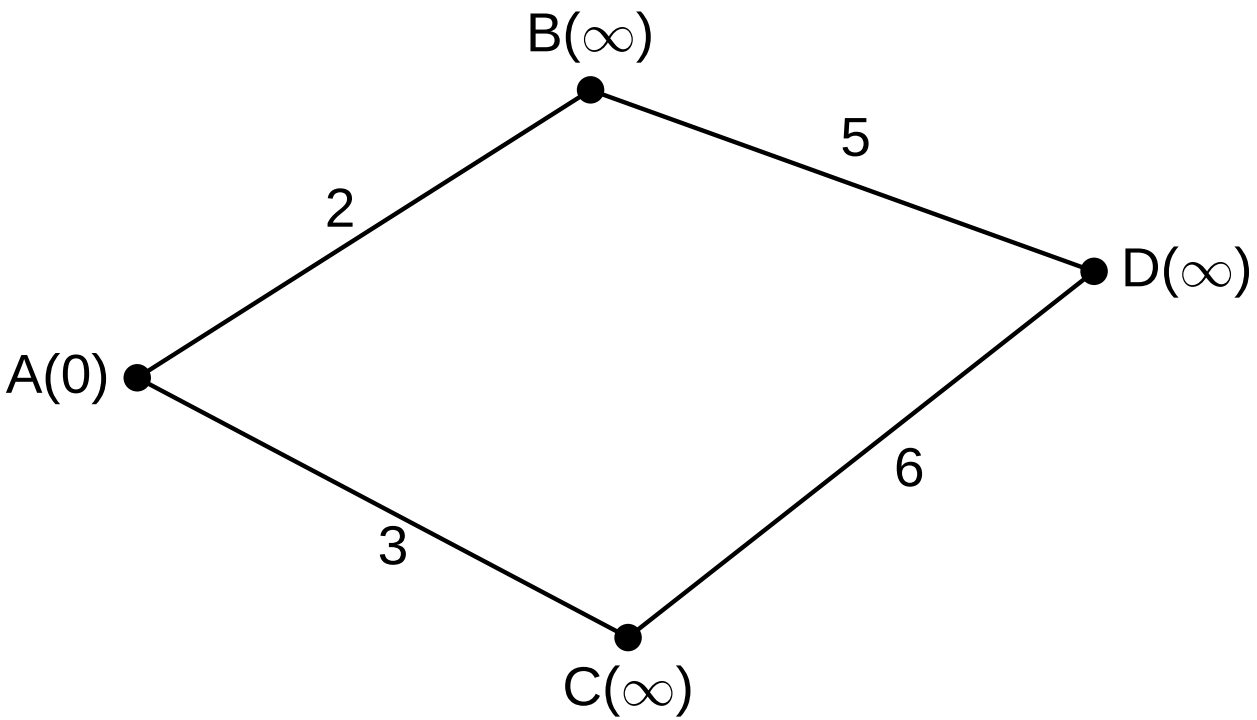
1. Assign to every vertex a distance value : 0 for the initial vertex and ∞) for all other vertices.
2. Mark all vertices as unvisited. Set initial vertex as current.
3. For current node, consider all its unvisited neighbours and calculate their distance. If this distance is less than the previously recorded distance, overwrite the distance.
4. When all neighbours of the current node have been considered, mark it as visited.

Dijkstra's algorithm

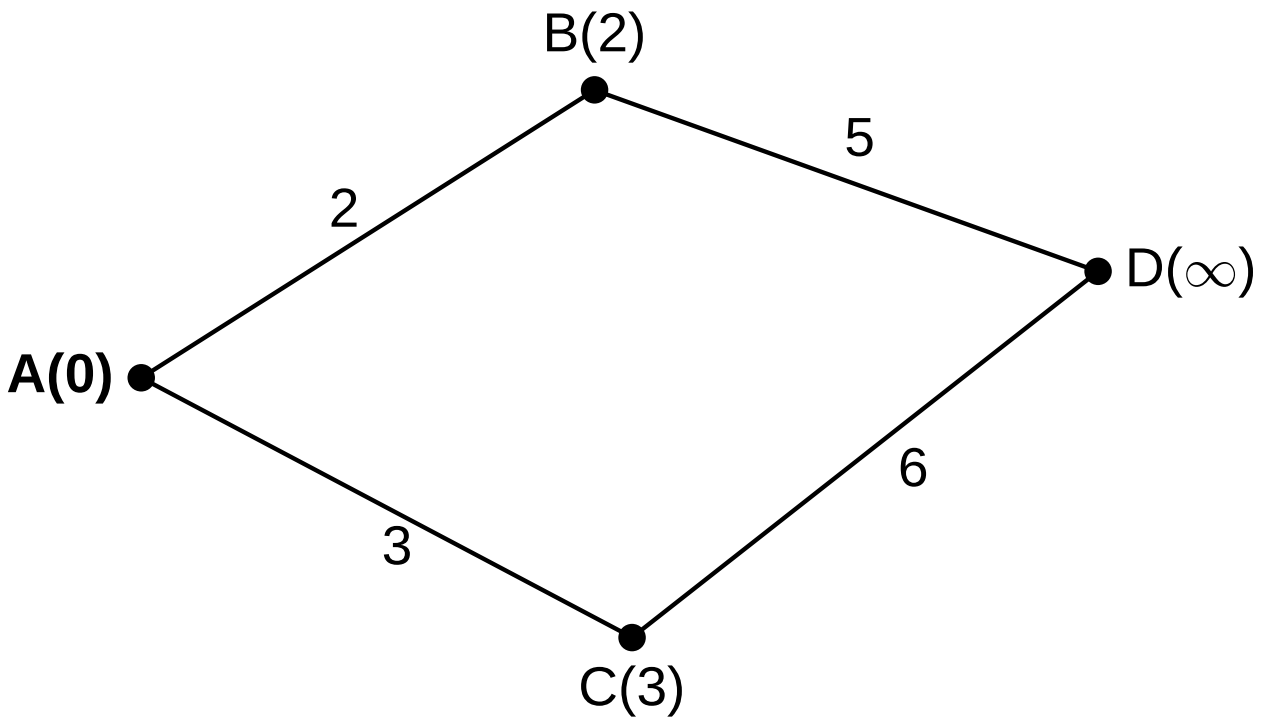
Vertex we are starting from : *initial vertex*.
Distance of a vertex Y : distance from the initial vertex to it.

1. Assign to every vertex a distance value : 0 for the initial vertex and ∞) for all other vertices.
2. Mark all vertices as unvisited. Set initial vertex as current.
3. For current node, consider all its unvisited neighbours and calculate their distance. If this distance is less than the previously recorded distance, overwrite the distance.
4. When all neighbours of the current node have been considered, mark it as visited.
5. Set the unvisited vertex with the smallest distance (from the initial node) as the next *current vertex* and continue from step 3.

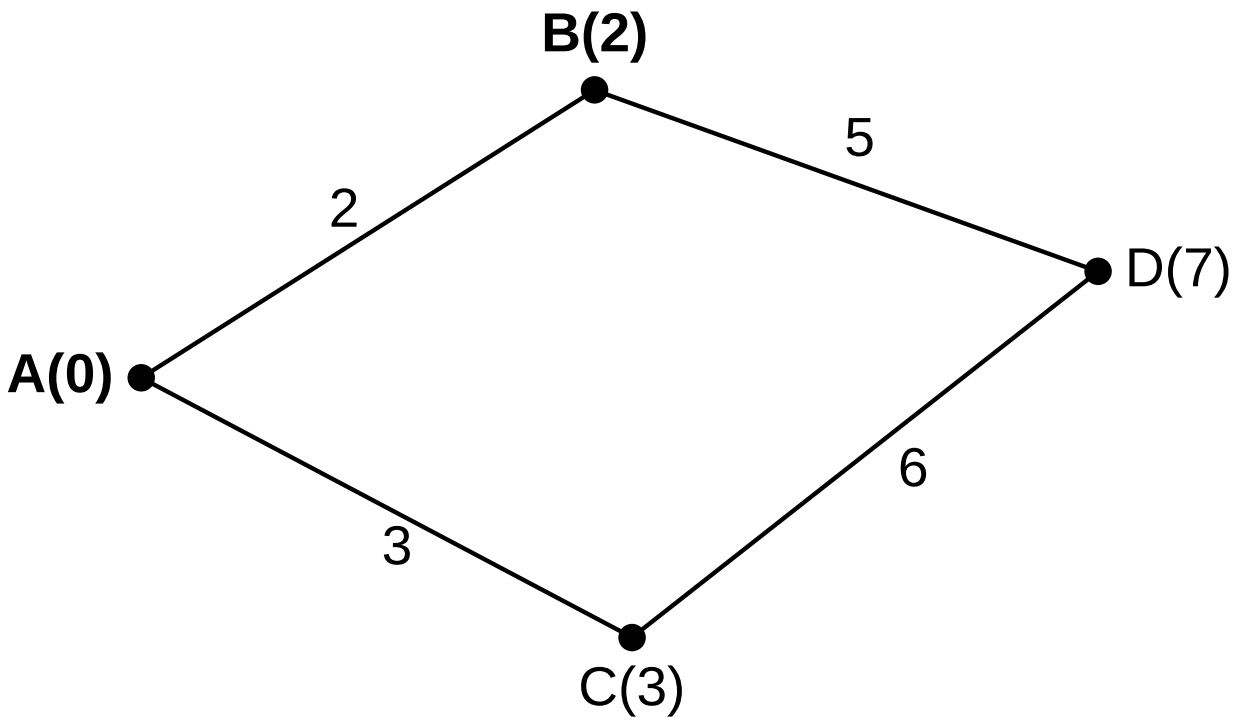
Dijkstra's algorithm : An example



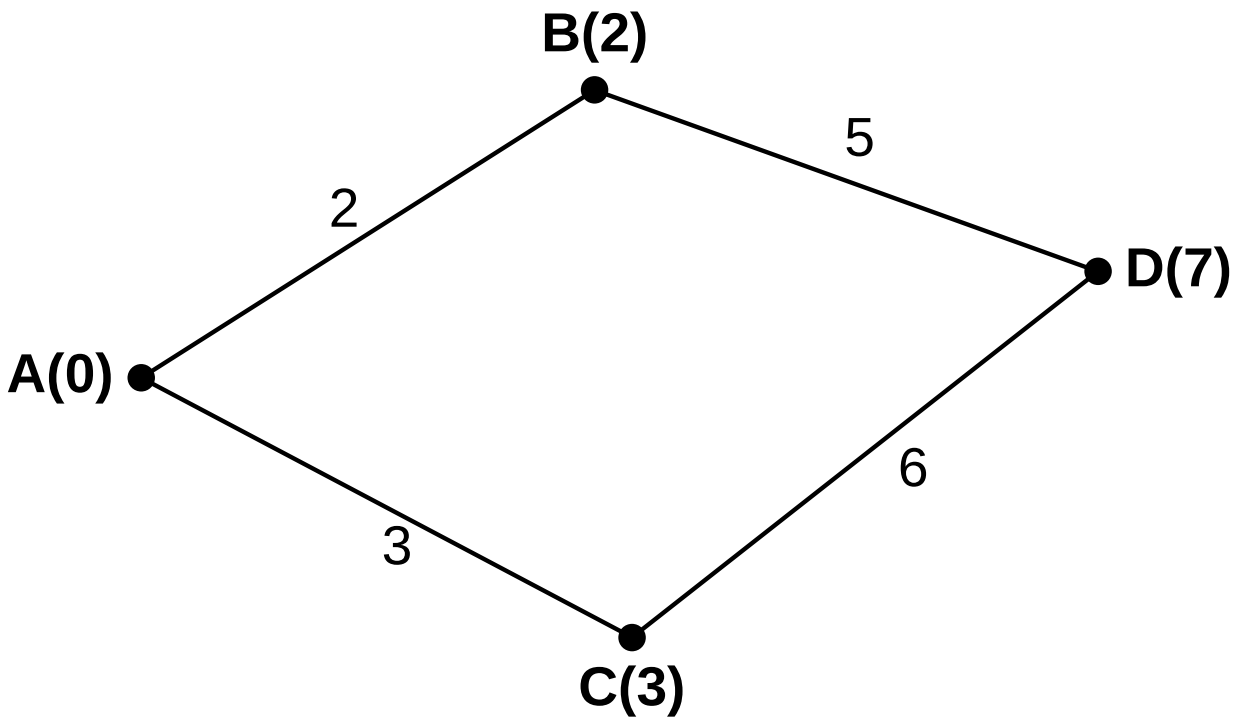
Dijkstra's algorithm : An example



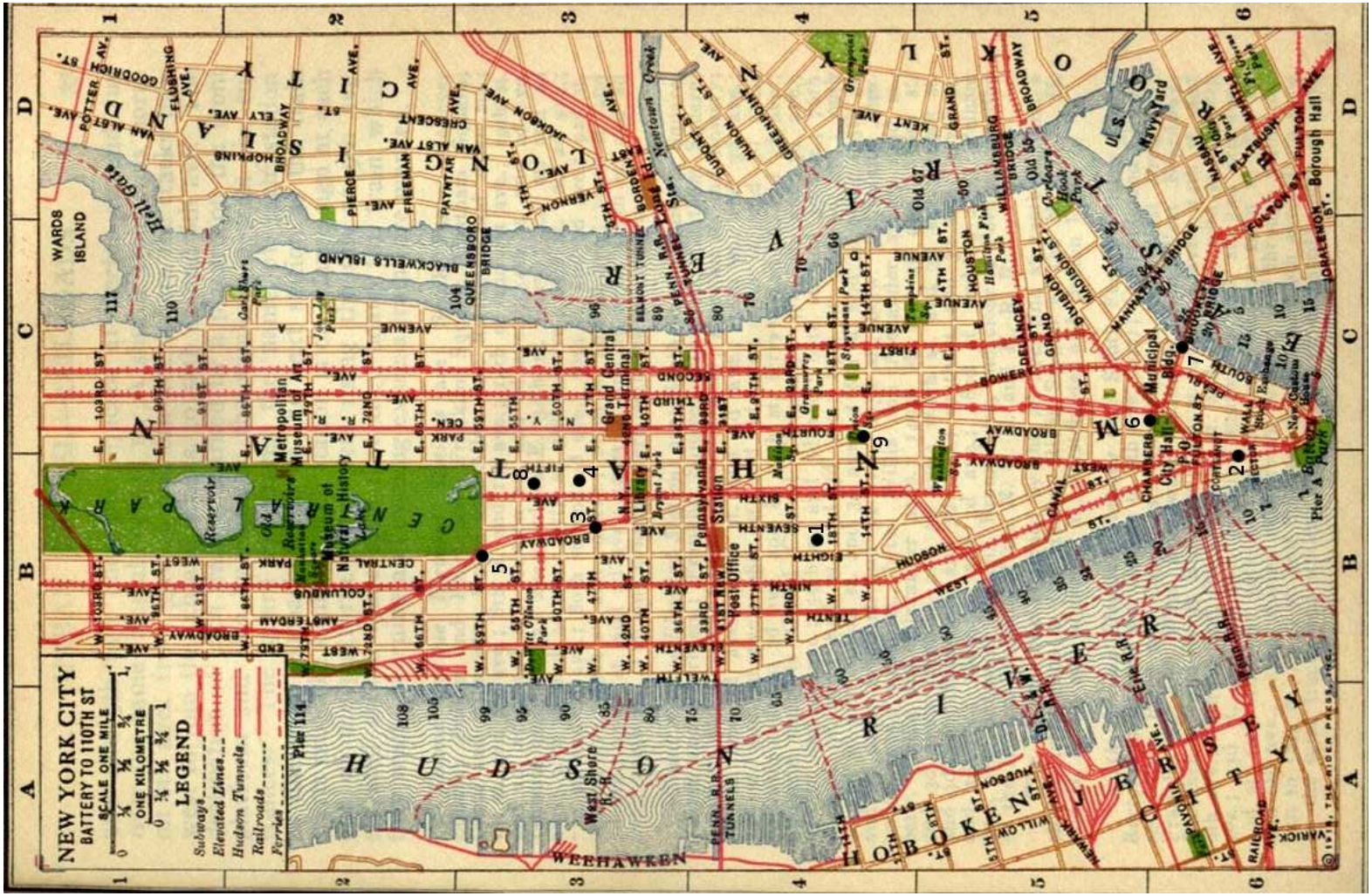
Dijkstra's algorithm : An example



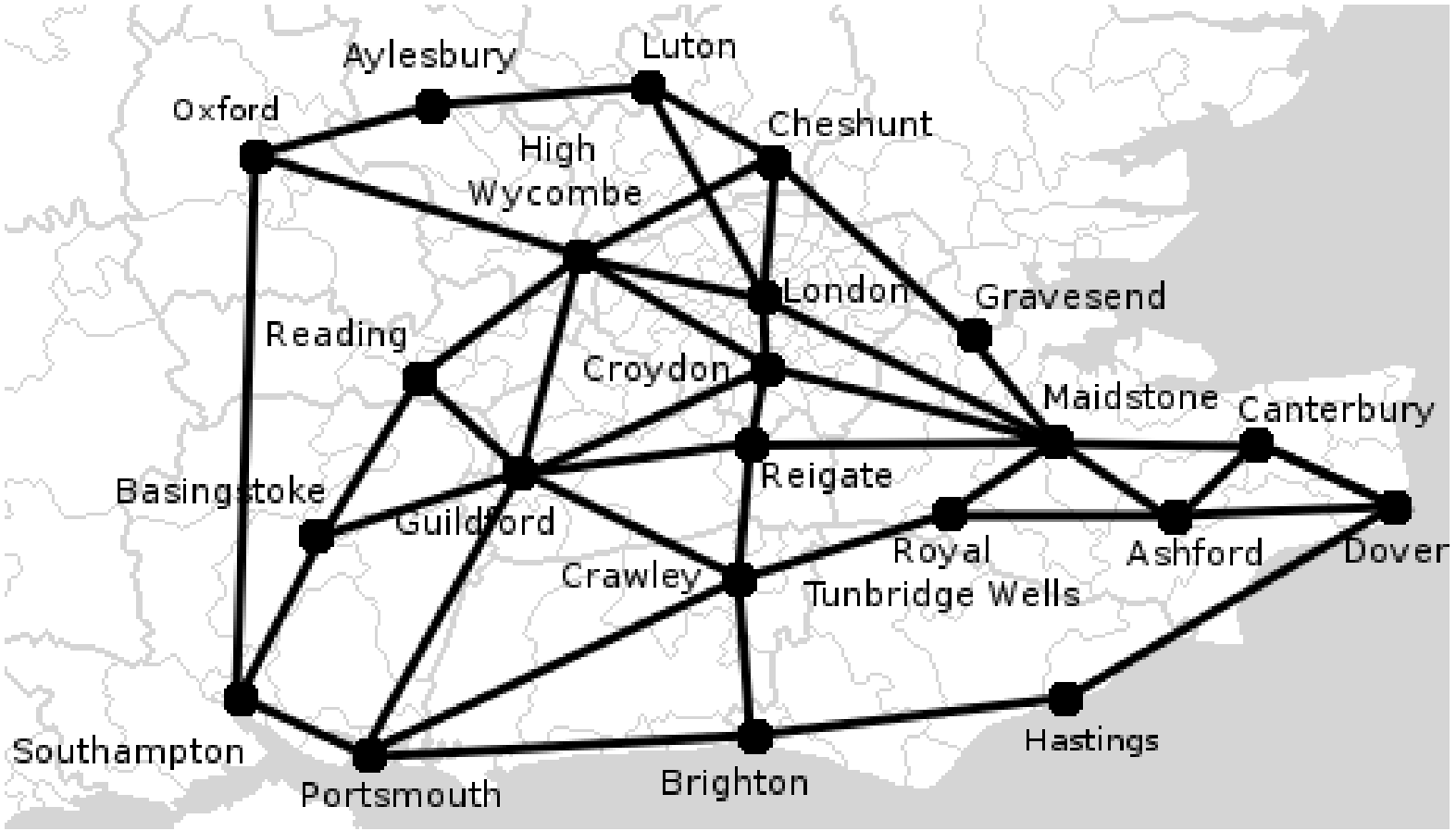
Dijkstra's algorithm : An example



Task sheet #2


NEW YORK CITY
BATTERY TO 110TH ST
SCALE ONE MILE
ONE KILOMETRE
LEGEND
Subways
Elevated Lines
Hudson Tunnels
Railroads
Ferries
HUDSON RIVER
EAST RIVER
NEW YORK BAY
HOBOKEN
WEEHAWKEN
CENTRAL PARK
MANHATTAN
BROOKLYN
QUEENSBORO BRIDGE
WILLIAMSBURG BRIDGE
MANHATTAN BRIDGE
BROOKLYN BRIDGE
Grand Central Terminal
Pennsylvania Station
Metropolitan Museum of Art
Museum of Natural History
Municipal Bldg.
City Hall
P.O.
Post Office
U.S. Navy Yard
Borough Hall
WARDS ISLAND
BLACKWELLS ISLAND
Hell Gate
Pier 114
West Shore R.R.
PENN. R.R. TUNNELS
BROADWAY
FIFTH AVENUE
PARK AVENUE
LEXINGTON AVENUE
MADISON AVENUE
WEST STREET
CANAL STREET
HUDSON STREET
CENTRAL PARK WEST
COLUMBUS AVENUE
AMSTERDAM AVENUE
WEST END AVENUE
RIVERSIDE DRIVE
FLATBUSH AVENUE
FULTON ST.
ATLANTIC AVE.
CLARKSON AVE.
LINCOLN PLACE
VAN ALST AVE.
QUEENS
BROOKLYN
STATEN ISLAND
JERSEY CITY
1
2
3
4
5
6
7
8
9

Task sheet #3


Aylesbury
Luton
Oxford
Cheshunt
High
Wycombe
London
Gravesend
Reading
Croydon
Maidstone
Canterbury
Basingstoke
Reigate
Guildford
Royal
Tunbridge Wells
Ashford
Dover
Crawley
Hastings
Southampton
Portsmouth
Brighton

Dijkstra's algorithm in pseudo-code

```
 1  function Dijkstra(Graph, source):
 2      for each vertex v in Graph: // Initializations
 3          dist[v] := infinity
        // Unknown distance function from source to v
 4          previous[v] := undefined
        // Previous node in optimal path from source
 5      dist[source] := 0
        // Distance from source to source
 6      Q := the set of all nodes in Graph
        // All nodes in the graph are unoptimized - thus are in Q
 7      while Q is not empty:       // The main loop
 8          u := vertex in Q with smallest dist[]
 9          if dist[u] = infinity:
10              break
        // all remaining vertices are inaccessible
11          remove u from Q
12          for each neighbor v of u:
        // where v has not yet been removed from Q.
13              alt := dist[u] + dist_between(u, v)
14              if alt < dist[v]:    // Relax (u,v,a)
15                  dist[v] := alt
16                  previous[v] := u
17      return previous[]
```