The shortest path problem	Season	2
	Episode	22
	Time frame	2 periods

Prerequisites : Vocabulary and basic concepts of graph theory.

Objectives :
- Understand the shortest path problem and its applications.
- Apply Dijkstra's algorithm.

Materials :
- *Task sheet #1 : Easy graphs.*
- *Task sheet #2 : More difficult graphs.*
- *Task shett #3 : From Dover to Oxford.*
- *Beamer.*
- *Lesson.*

1 – Introduction to the problem (task sheet #1) — 1 period

Weighted graphs and the shortest path problem is introduced in relation with the route planning websites and GPS devices.
Students have to solve a few simple examples. Solutions are compared between students.

2 – Dijkstra's algorithm (task sheets #1 and #2) — 1 period

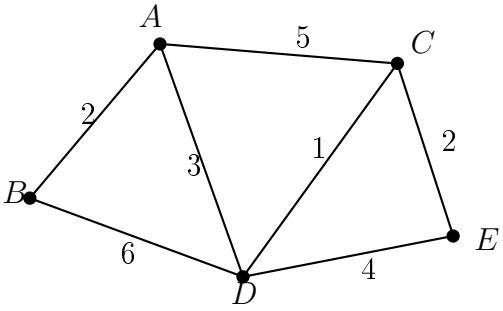
Dijkstra's algorithm is introduced and applied on the simple graphs of part 1. Then, students have to apply it on a map of Manhattan.

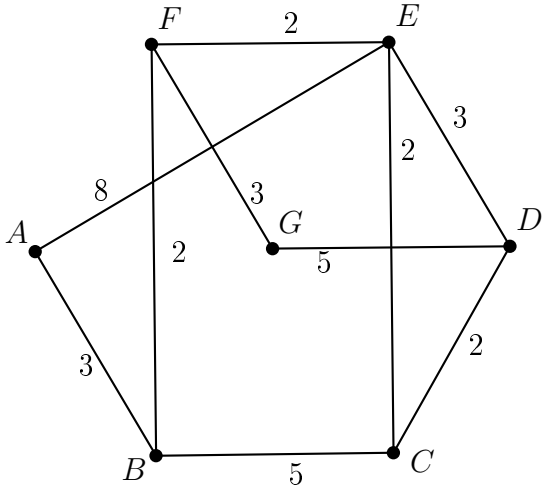
3 – From Dover to Oxford (task sheet #3) — 1 period

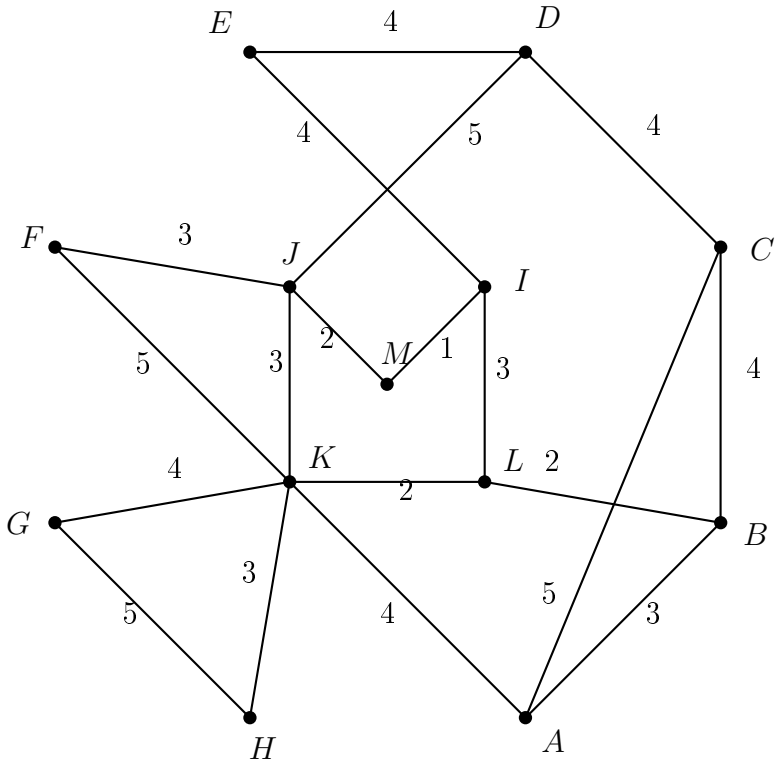
In the computer room, students use the Via Michelin website (http ://www.viamichelin.co.uk/) to fill out two maps, one with the distances between some cities, the other with the travelling times.
Then they have to use Dijskstra's algorithm to find the shortest and quickest routes between Dover and Oxford.
Finally, they compare the quickest and shortest itineraries they found with the ones given by the website.

The shortest path problem	Season	2
	Episode	22
	Document	Task sheet #1

Find the shortest distance and shortest path from the vertex A to all the other vertices in the following graphs.







The shortest path problem	Season Episode Document	2 22 Task sheet #2

Below is a subway map of Manhattan.

1. Chelsea International Hostel.
2. Battery Park.
3. Times Square.
4. The Empire State Building
5. Columbus Circle.
6. City Hall.
7. Brooklin Bridge.
8. Museum of Modern Art

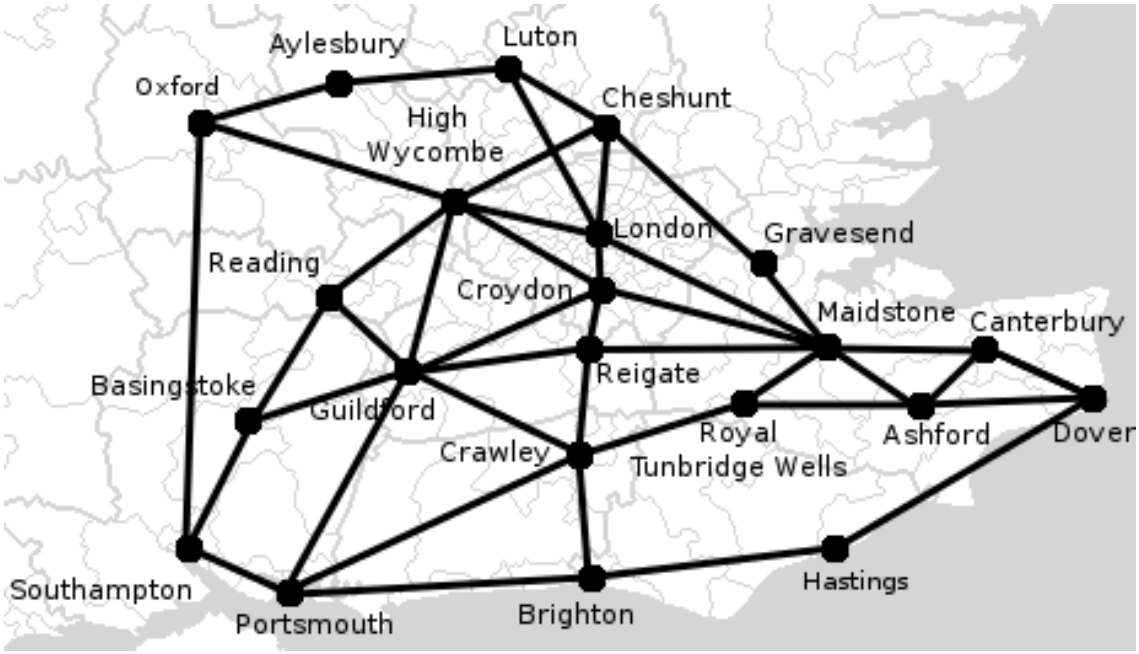
First, use the internet to find the closest subway station to each of these locations. Then, use the subway map as a graph to find the shortest route between Union Square and each location. To do so, we will consider that the average time between two subway stations is 5 minutes, and we will not take into account the possible switches from one line to another, except

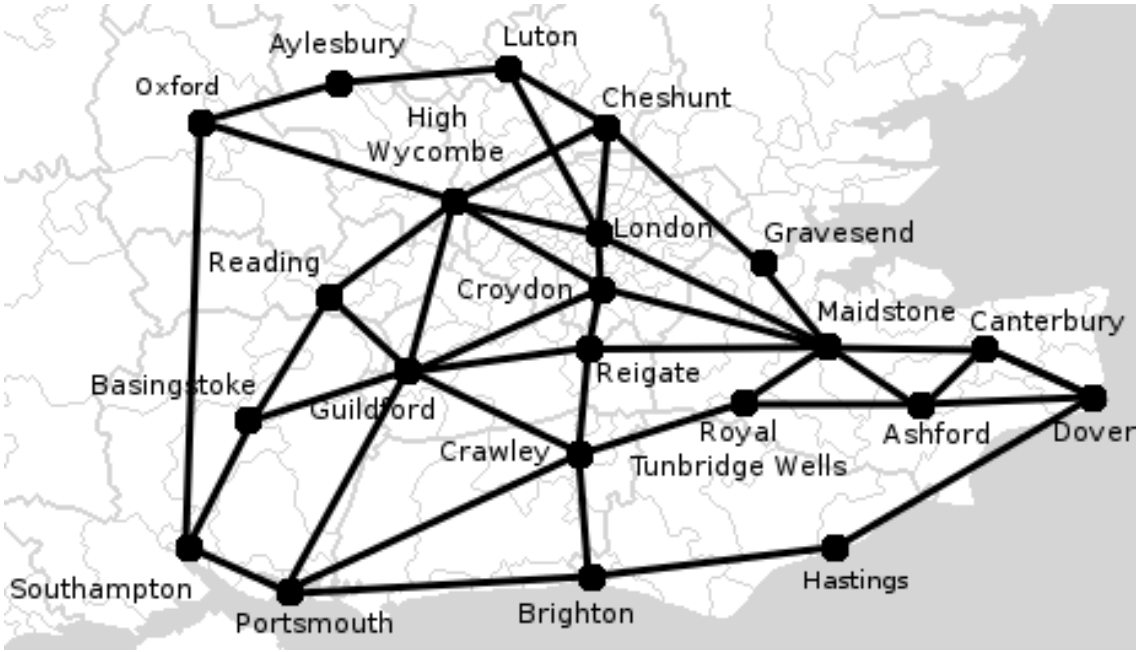
- 14 St 1239 to 6 Av L ;
- Port Authority ACE to Times Square 42 St ;
- Chambers St to Park Place ;

which will be accounted for 5 minutes each.

The shortest path problem	Season	2
	Episode	22
	Document	Task sheet #3

Below are two maps of the South-East of England. We want to plan a trip from Dover to Oxford. We will do so in two different ways, that's why there are two copies of the map.





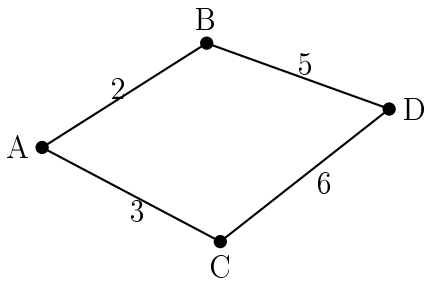
1. Use the route planning websites `http://maps.google.co.uk/` or `http://www.viamichelin.co.uk/` to find the distances and travel times for all adjacent towns on the map. Make sure that no itinerary passes through one other town. For example, from Ashford to Royal Tunbridge Wells, the website may advise you to drive around Maidstone on the M20 but you should take the distance and time along the direct route starting on the A28. Write down the distances on one map and the travel times on the other.
2. Use Dijkstra's algorithm to find the shortest route between Dover and Oxford.
3. Use Dijkstra's algorithm to find the quickest route between Dover and Oxford.
4. Is the shortest route also the quickest ? Explain why it is so.
5. Compare the itineraries you found with the ones given by the route planning website. Explain any difference you may notice.

The shortest path problem	Season	2
	Episode	22
	Document	Lesson

Weighted graphs and the shortest path problem

Definition 1

A *weighted* graph is a graph where a real number, a *weight* is assigned to each edge. The weight, or length, of a path on this graph is the sum of the weights of its constituent edges.



The *shortest path problem* is the problem of finding a path between two vertices such that the sum of the weights of its constituent edges is minimized. An example is finding the quickest way to get from one location to another on a road map ; in this case, the vertices represent locations and the edges represent segments of road and are weighted by the distance or the time needed to travel that segment.
Different algorithms have been devised to solve this problem. One of the best known is *Dijkstra's algorithm*, named after the Dutch computer scientist that conceived it in 1959.

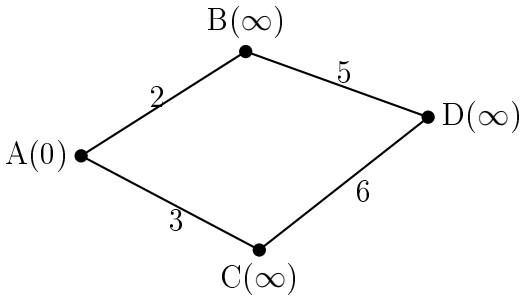
Dijkstra's algorithm

Let's call the vertex we are starting from the *initial vertex*. Let the *distance of a vertex Y* be the distance from the initial vertex to it. Dijkstra's algorithm will assign some initial distance values and will try to improve them step-by-step.

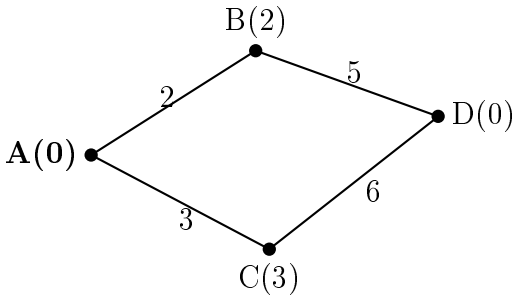
1. Assign to every vertex a distance value. Set it to zero for our initial vertex and to infinity (∞) for all other vertices (written in paper pen, as it will change when the algorithm is applied).
2. Mark all vertices as unvisited. Set initial vertex as current.
3. For current node, consider all its unvisited neighbours and calculate their distance (from the initial node). For example, if current vertex (A) has distance of 6, and an edge connecting it with another vertex (B) is 2, the distance to B through A will be 6+2=8. If this distance is less than the previously recorded distance (infinity in the beginning, zero for the initial node), overwrite the distance.
4. When all neighbours of the current node have been considered, mark it as visited, by circling its distance in green, for example. A visited vertex will not be checked ever again ; its distance recorded now is final and minimal.
5. Set the unvisited vertex with the smallest distance (from the initial node) as the next *current vertex* and continue from step 3.

Example

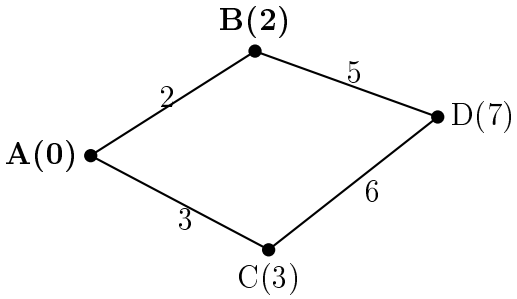
The algorithm begins by initializing any vertex in the graph (vertex A, for example) a permanent label with the value of 0, and all other vertices a temporary label with the value of ∞. Visited vertices are typeset in bold.

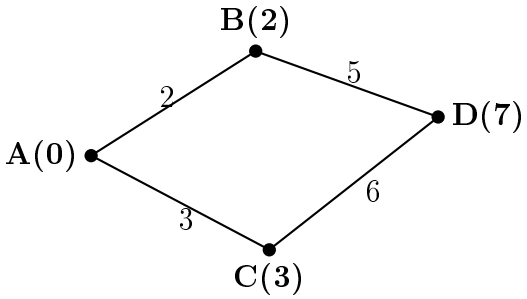


We then pick the non-visited vertex with the lowest label (vertex A in our example), and study all its neighbours. Whenever the distance from one of these neighbours through the current vertex is less then its temporary label, we change the label to this value. Once all neighbours have been visited, the current vertex is marked as visited.



We now apply the same process to vertex B, then to C and D. Notice that nothing is changed when studying C, as the distance from D is greater through C then it is through B. Also, there is nothing to do for vertex D.





The permanent labels of the vertices give the shortest distance from the vertex A.

```
 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[]
```