Dynamic Programming

For a given weighted network (di-graph) there is the problem of finding the shortest (or longest) path from one vertex to another. What is required is an algorithm that computes the shortest path. Any such algorithm is called a greedy algorithm. Dynamic programming is a greedy algorithm for finding the shortest path through a network.

Another algorithm for finding the shortest path is Dijkstra's algorithm. Dijkstra's algorithm is quicker (more efficient) than Dynamic Programming, but does not have as broad scope

(1) Dijkstra's algorithm cannot be applied to cases where there are negatively weighted edges

(2) Dynamic programming is more readily adapted to other problems – e.g. the problem of finding the longest path through a network.

The dynamic programming algorithm:

This algorithm works by systematically labelling vertices with two numbers;
N = the stage at which the vertex was labelled, L = the length of the shortest path of that vertex from the starting vertex.

The symbol $(N; L)$ will denote the stage (N) at which the vertex was labelled and the minimum length (L) of that vertex from the starting vertex.

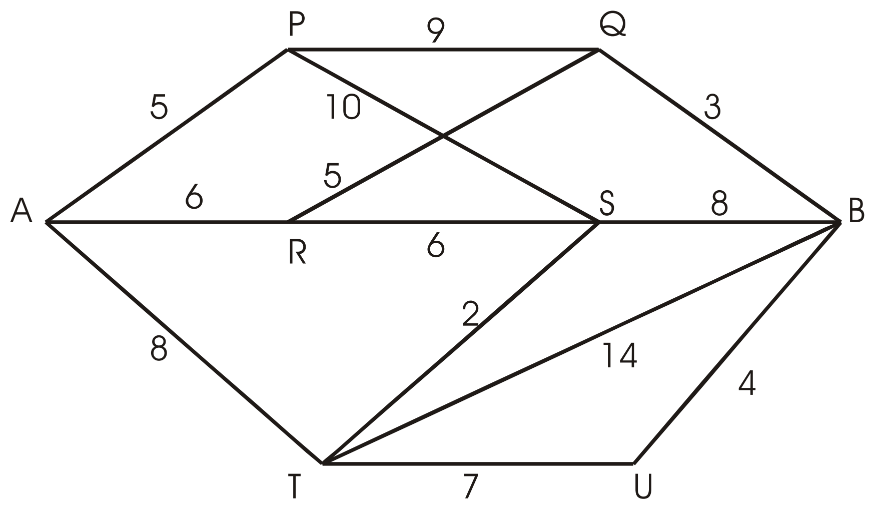
STEP 1: Assign the label (0;0) to the starting vertex

STEP 2: For a given stage, N, identify all vertices with labels (N; L) that have been already reached at that stage. For each such vertex (N; L) find all vertices that can be reached from that vertex by a movement along a single edge. Compute the length L + edge weight (L + W) for each such vertex. Assign the temporary label (N + 1; L + W) to each such vertex. Repeat the process until all vertices at the same stage, N, have been examined.

STEP 3: Assign to each vertex with stage, label N + 1 the length L' where L' = minimum of all the temporary length labels. Label the vertex (N + 1, L') and delete all the temporary labels. If a vertex has a label (N', L) where N' < N + 1 and L < L' delete that label and replace it by the label (N + 1; L').

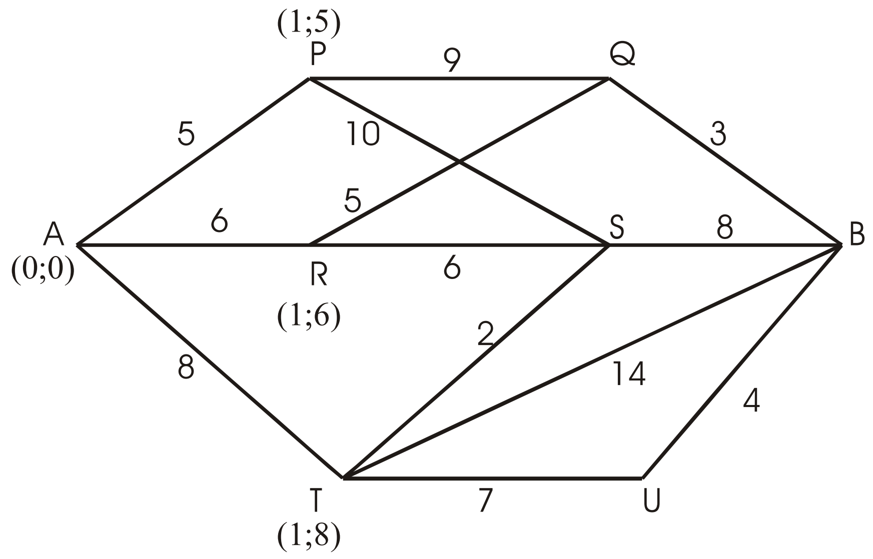
Example

The following network represents distances between towns in km.

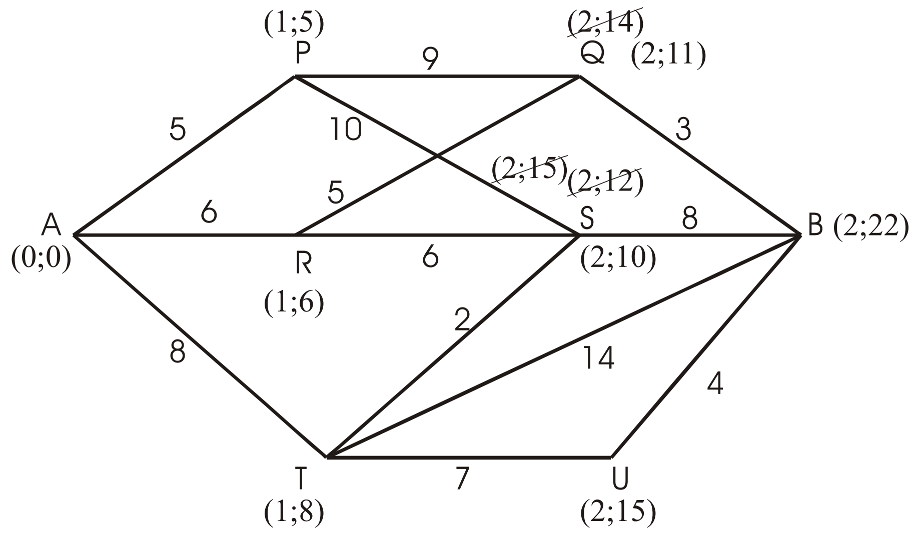


We will apply dynamic programming to find the shortest path from A to B.

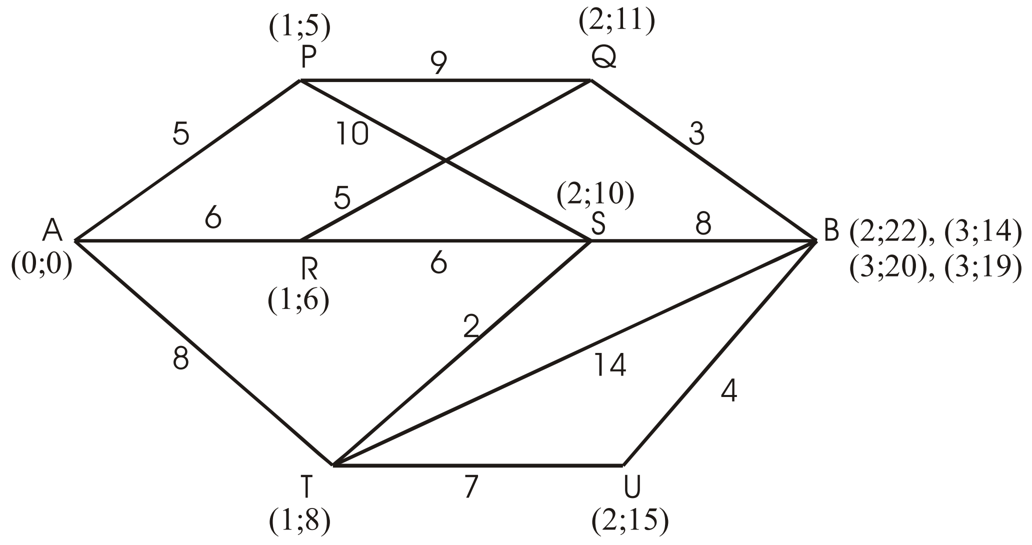
Stage 1:



Stage 2:



Stage 3:



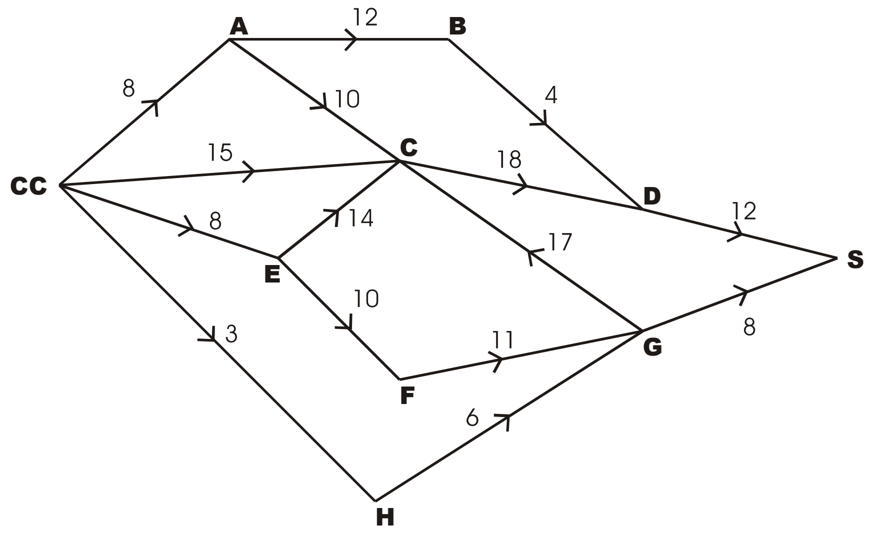
So the shortest path from A to B has length 14 km and tracing backwards this is along the route: A → R → Q → B.

Just a word about the application of the algorithm to the above example. In this example the network is not a directed network, thus, strictly speaking it is possible to trace backwards along an edge. For example, vertex R at stage 3 should acquire the temporary label (3;16) by tracing back from Q with (2;11) to R along the edge with weight 5. However, labels that obviously increase the distance have simply been omitted from the diagram. They would be deleted anyway when temporary labels with distances longer than the minimum are deleted.

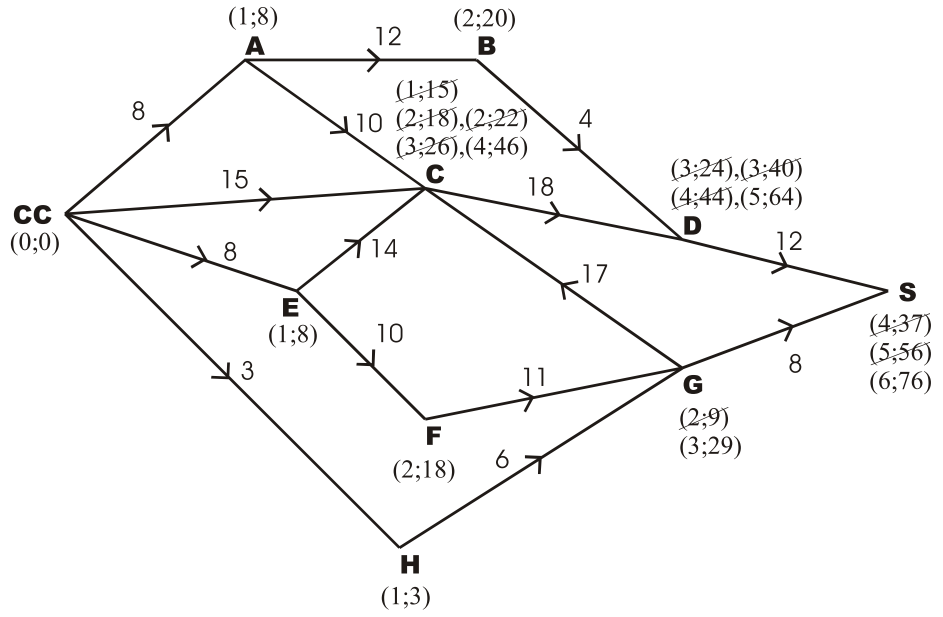
Dynamic programming is a versatile approach to the problems relating to networks.

Example

The following directed network shows potential bus links between a city centre (CC) and a suburb (S). The edge weights indicate the average number of passengers collected along each section of the route. The aim is to maximise the average number of passengers.



Using dynamic programming the solution is:



Tracing backwards, the optimum route is:
CC → E → F → G → C → D → S
with maximum average numbers of passengers = 76

Modelling decision processes

Dynamic programming is particularly suitable for solving problems involving decisions in business or games. But first, information about a decision process must be modelled by a directed network. The model explored here involves labelling vertices by two variables – stage and state to label vertices.

The stage variable corresponds to the number of stages required by the dynamic programming algorithm to reach that vertex. The state variable will designate something of relevance to the problem under investigation – for example, a position on a board in a board game, the number of finished items in a store – and so forth. In the modelling process vertices are first identified and labelled. These represent possible outcomes of the decision making process. Next, directed edges are drawn between the vertices corresponding to the decisions that are possible from one stage to the next. Along the edge a weight variable indicates the outcome of that decision – for example, a profit made as a result of taking that option. The edge may additionally be labelled with the decision taken for the purpose of clarity. The direction of edge runs from one stage to the next.

In a game, a player has at each turn three options – invest one money unit, consume one money unit, or work. The aim of the game is to maximise happiness, which is measured in utility points. (Each stage represents 10 years of time). To consume you spend money points. The utility obtained from consumption depends on the stage reached as follows:

	Stage						
	1	2	3	4	5	6	7
Utility	6	7	6	5	5	4	2

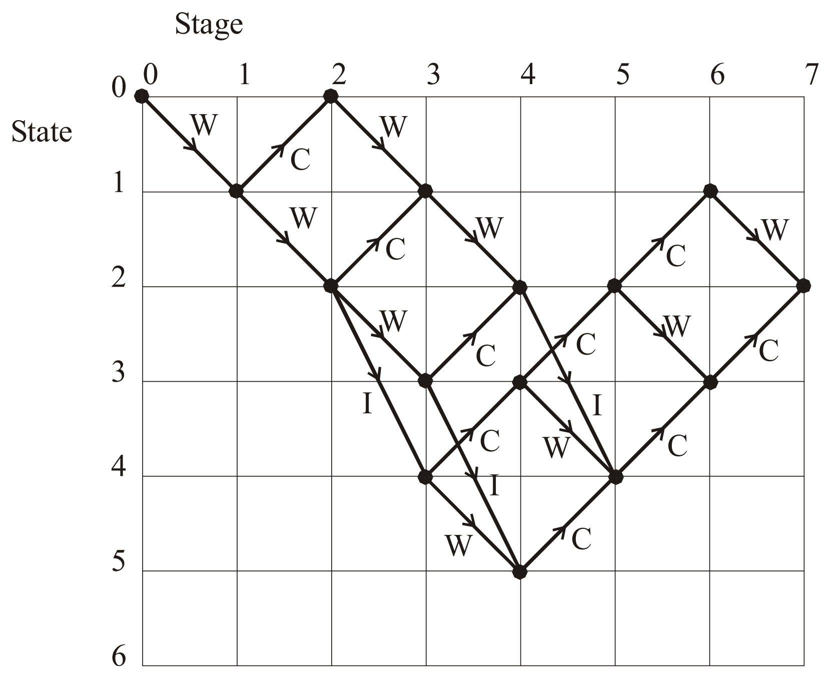
Work produces return of one money unit at every stage.

Investment at each stage returns three money units for 1 unit invested. You may only invest if you have at least 2 money units.

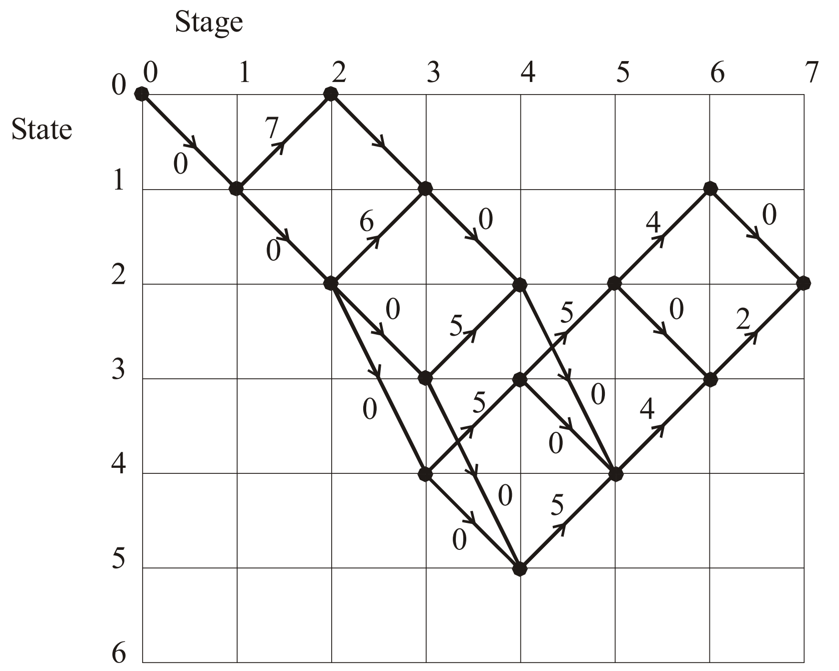
You must finish the game with 2 money units (one to pass on to your wife, and one to pay for your funeral).

Solution

The completed network, with the decisions relevant to each edge, is as follows

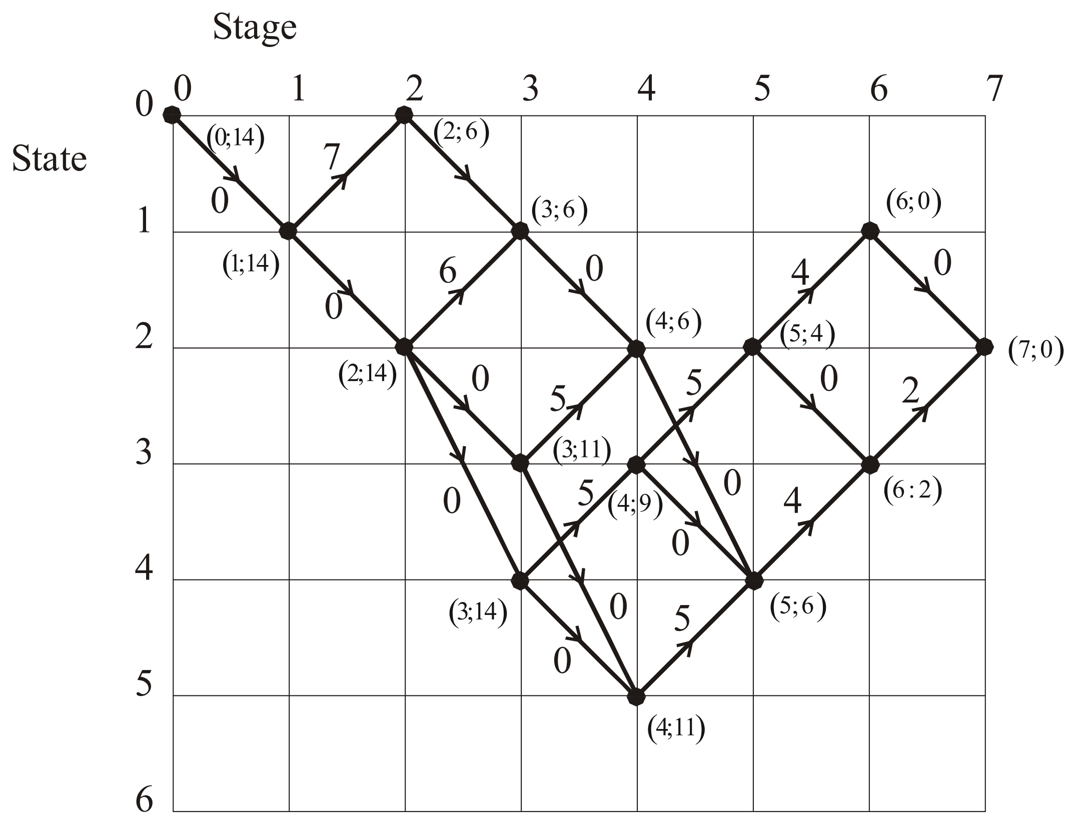


The following digraph shows the edges with their weightings – that is, with the pay offs in terms of utility.



We will now apply dynamic programming to find the optimum path from stage 0 to stage 7.

In this process it is regarded as correct to work backwards from the goal stage and state to the starting stage and state. It is more efficient to do so. We commence by labelling the final vertex (7 ; 0). Here the stage is 7 and the pay-off is 0. Working backwards, and applying the dynamic programming algorithm we obtain the following solution.



The maximum utility is 14 points; working forwards this is produced by the choices

Stage	Decision
0	work
1	work
2	invest
3	consume
4	consume
5	consume
6	work

You are expected to be able to display the entire solution in a tabular form representing every calculation including those of the temporary labels and the resultant choice of permanent labels at each stage. The table partly represents what a computer would be doing at each stage of the calculation. It is advisable to perform the calculation on the network by hand first and then transfer the solution to the table.

Stage	Current state	Previous decision	Previous state	Pay-off	Permanent label
7	2	WORK	1	0	YES
		CONSUME	3	2	YES
6	1	CONSUME	2	$0+4=4$	YES
	3	WORK	2	$2+0=2$	
		CONSUME	4	$2+4=6$	YES
5	2	CONSUME	3	$4+5=9$	YES
	4	INVEST	2	$6+0=6$	YES
	4	WORK	3	$6+0=6$	
	4	CONSUME	5	$6+5=11$	YES
4	2	WORK	1	$6+0=6$	YES
	2	CONSUME	3	$6+5=11$	YES *
	3	CONSUME	4	$9+5=14$	YES
	5	INVEST	3	$11+0=11$	YES *
	5	WORK	4	$11+0=11$	
3	1	WORK	0	$6+0=6$	YES
	1	CONSUME	2	$6+6=12$	
	3	WORK	2	$11+0=11$	
	4	INVEST	2	$14+0=14$	YES
2	0	CONSUME	1	$6+7=13$	
	2	WORK	1	$14+0=14$	YES
1	1	WORK	0	$14+0=14$	YES

* There are two possible lines of action at this stage giving the same utility – however, neither is a solution to the problem as a whole.