Introduction to Graph Theory

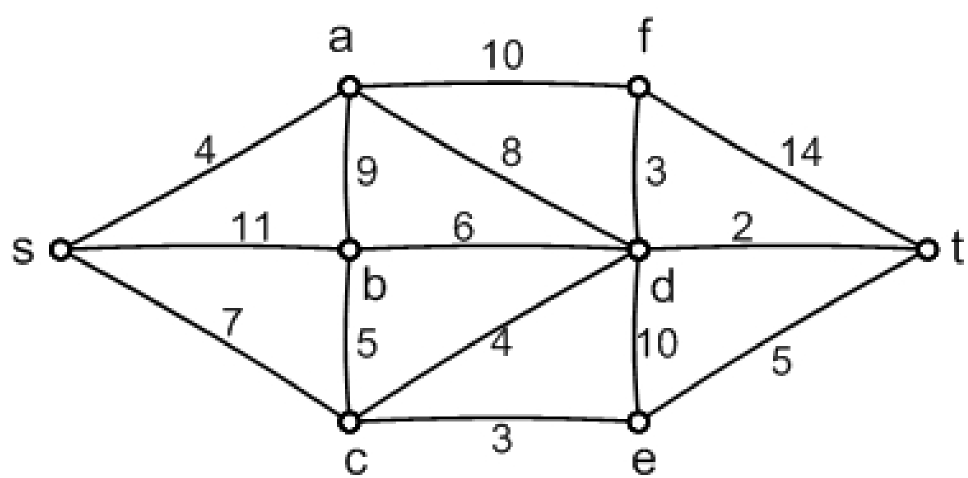
Lecture 27

Menu

- Shortest path algorithm to determine the shortest path between two vertices of a weighted graph

Example 1

- The weighted graph shown below represents a communication network with weights indicating the delays associated with each edge.
- Find the minimum delay path from *s* to *t*.



Solution - Stage 1:

- Begin at the start vertex *s*. This is the reference vertex for stage 1.
- Label all the adjacent vertices with the lengths of the paths using only one edge.
- Mark all other vertices with a very large number (larger than the sum of all the weights in the graph). In this case we choose 100. This is shown in the diagram.
- At the same time, start to form a table as shown in Table 1.
- The lengths of paths using only 1 edge from *s*

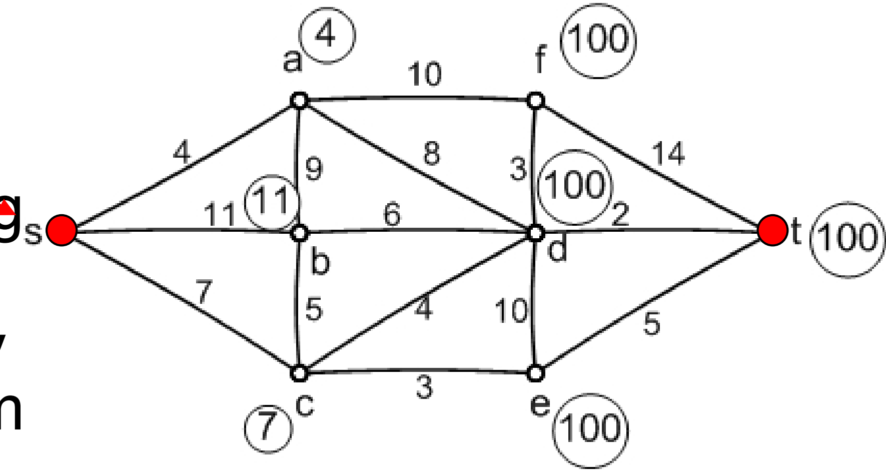


Table 1

	a	*b*	*c*	*d*	*e*	*f*	*t*
s	4	11	7	100	100	100	100

Solution - Stage 2:

- Choose as the reference vertex for stage 2 the vertex with the ***smallest label*** that has not already been a reference vertex. This is vertex ***a***.
- Consider any vertex adjacent to the new reference vertex and mark it with the length of the path from *s* via *a* to this vertex if this is less than the current label on the vertex. This gives the labels shown right.
- We also add a new line to Table 1 to give Table 2, noting that as vertex *a* has been made a reference vertex the label of *s* becomes permanent and is marked with an underline in the table.
- The lengths of paths using up to 2 edges from *s*

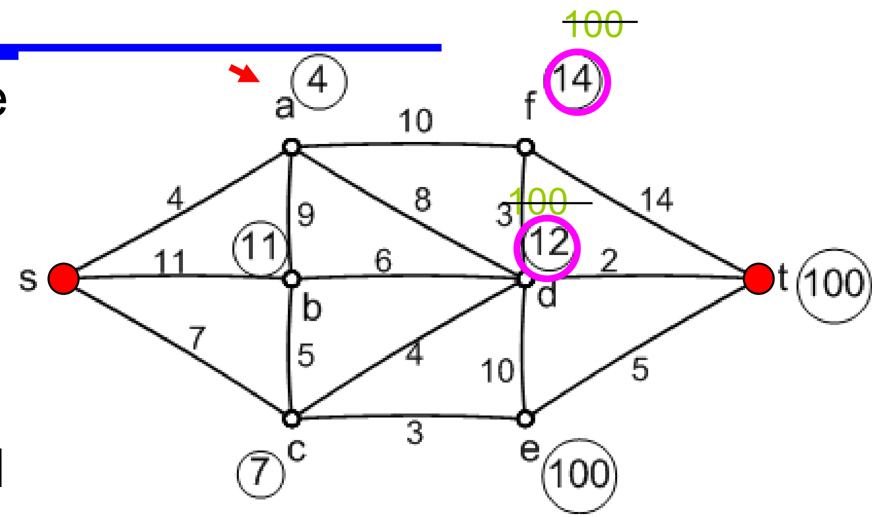
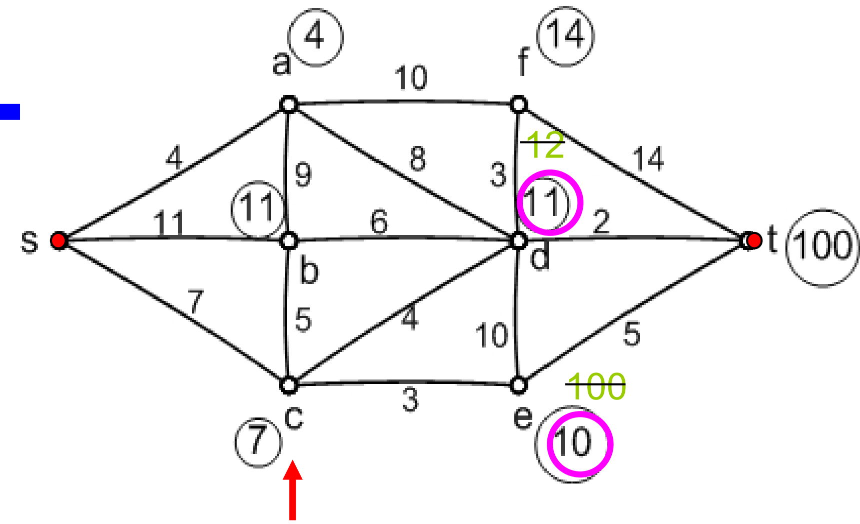


Table 2

	a	*b*	*c*	*d*	*e*	*f*	*t*
s	<u>4</u>	11	7	100	100	100	100
a		**11**	**7**	**12**	**100**	**14**	**100**

Solution - Stage 3:

- Choose as the reference vertex the vertex with the ***smallest label*** that has not already been a reference vertex. From table 2 we see that **c** is the reference vertex for stage 3.
- Consider any vertex adjacent to *c* that does not have a permanent label and calculate the length of the path from *s* via *c* to this vertex. If it is less than the current label on the vertex mark the vertex with this length. This gives us the labels shown right.
- We also add a new line to Table 2 to give Table 3. Note that the third line of Table 3 does not have an entry for *a* as this has already been a reference vertex.



- The lengths of paths using up to 3 edges from *s*

Table 3

	a	*b*	*c*	*d*	*e*	*f*	*t*
s	4	11	7	100	100	100	100
a		11	7	12	100	14	100
c		**11**		**11**	**10**	**14**	**100**

Solution - Stage 4:

- Proceeding as before, the reference vertex for stage 4 is, by inspection of the third line of Table 3, vertex **e**.
- Again we calculate the lengths of the paths from *s* via *e* to any vertices adjacent to *e* that do not have permanent labels and replace the labels on those vertices with the relevant path lengths if this is less than the existing label.
- This gives the labels shown right and Table 4.

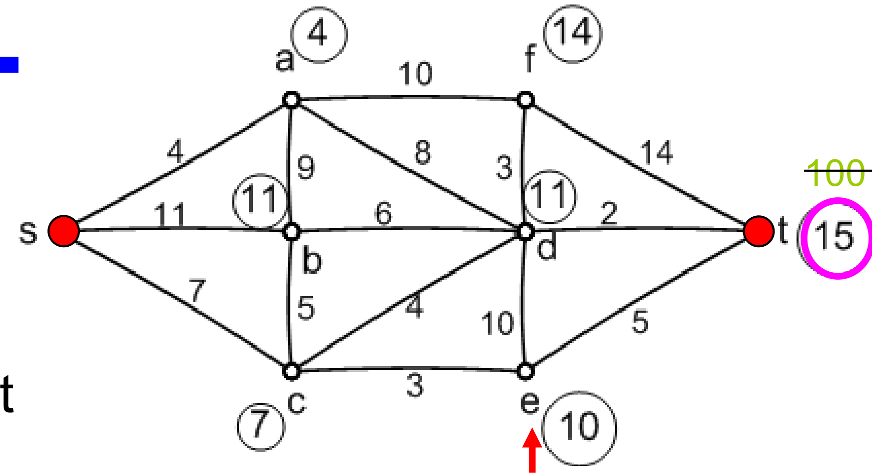


Table 4

	a	*b*	*c*	*d*	*e*	*f*	*t*
s	4	11	7	100	100	100	100
a		11	7	12	100	14	100
c		11		11	**10**	14	100
e		**11**		**11**		**14**	**15**

- The lengths of paths using up to 4 edges from *s*

Solution - Stage 5:

- Choose ***b*** as the new reference vertex (we could have chosen d instead but this would make no difference to the final result).
- Compare paths from *s* via *b* to the labels on any adjacent vertices with temporary labels and re-label if the paths are found to be shorter.
- The result of stage 5 is that the labels remain as in stage 4, but that the label on b becomes permanent giving Table 5.

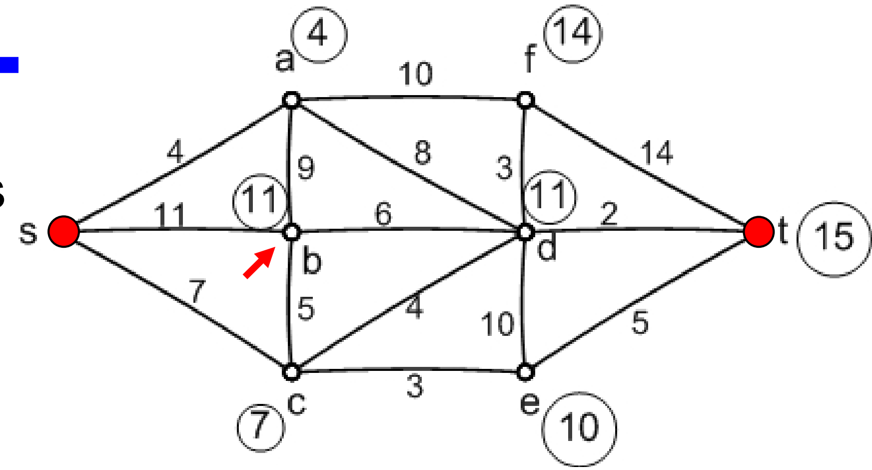


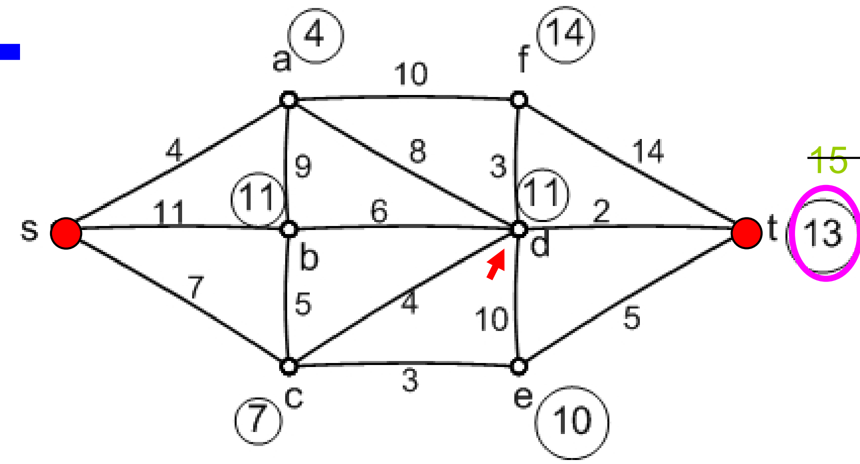
Table 5

	a	*b*	*c*	*d*	*e*	*f*	*t*
s	4	11	7	100	100	100	100
a		11	7	12	100	14	100
c		11		11	10	14	100
e		**11**		11		14	15
b				**11**		**14**	**15**

- The lengths of paths using up to 5 edges from *s*

Solution - Stage 6:

- Choose ***d*** as the new reference vertex.
- The only vertices left without permanent labels are now *f* and *t*.
- The path from *s* via *d* to *t* gives *a* smaller value than the current label of 15. Hence we change the label to 11+2=13.
- The new labels are shown right together with Table 6.

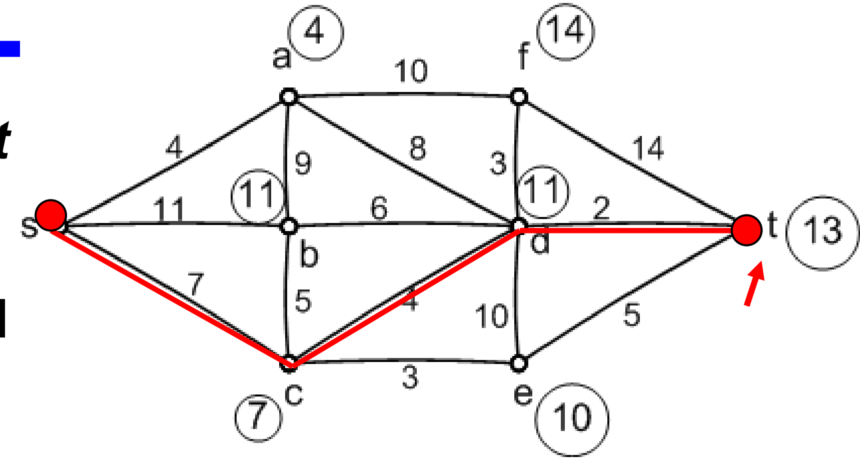


- The lengths of paths using up to 6 edges from *s*

Table 6

	a	*b*	*c*	*d*	*e*	*f*	*t*
s	4	11	7	100	100	100	100
a		11	7	12	100	14	100
c		11		11	10	14	100
e		11		11		14	15
b				**11**		14	15
d						**14**	**13**

Solution - Stage 7:



- The remaining vertex with the ***smallest label*** is ***t***.
- We therefore give *t* the permanent label of 13.
- As soon as *t* receives a permanent label the algorithm stops as this label is the length of the shortest path from *s* to *t*.
- To find the actual path with this length we ***move backwards*** from *t* looking for *consistent* labels.
- This gives *t d c s*. That is, the path is *s c d t*.

Dijkstra's Shortest Path Algorithm (SPA)

- Let the node at which we are starting be called the **initial node**. Let the **distance of node *Y*** be the distance from the **initial node** to *Y*. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.
- Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
- Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.
- For the current node, consider all of its unvisited neighbors and calculate their *tentative* distances. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one. For example, if the current node *A* is marked with a distance of 9, and the edge connecting it with a neighbor *B* has length 4, then the distance to *B*(through *A*) will be 9 + 4 = 13. If B was previously marked with a distance greater than 13 then change it to 13. Otherwise, keep the current value.
- When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.
- If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
- Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

Why is SPA optimal?

- Why SPA gives us the shortest path?
- What is the complexity of SPA?
- Can SPA be generalized for related shortest path problems?

Summary

- Demonstrated the algorithm to determine the shortest path between two vertices of a weighted graph

Readings

- [Mar07] Read 9.3
- [Mar13] Read 9.3