CHAPTER 21. Graphs and Combinatorial Optimization

Sec. 21.1 Graphs and Digraphs

Problem Set 21.1. Page 1014

7. **Adjacency matrix of a digraph.** The given figure shows 4 vertices, denoted by 1, 2, 3, 4, and 4 edges e_1, e_2, e_3, e_4. This is a digraph (directed graph), not just a graph, because each vertex has a direction, indicated by an arrow head. Thus, edge e_1 goes from vertex 1 to vertex 2, and so on. And there are two edges connecting vertices 1 and 3. These have opposite directions (e_2 from vertex 1 to vertex 3, and e_3 from vertex 3 to vertex 1). In a graph there cannot be two edges connecting the same pair of vertices because we have excluded this (as well as edges going from a vertex back to the same vertex, as well as isolated vertices; see Fig. 446). An adjacency matrix **A** of a graph or digraph is always square, $n \times n$, where n is the number of vertices. This is the case, regardless of the number of edges. Hence in the present problem, **A** is 4×4. The definition on p. 1013 shows that in our case, $a_{12} = 1$ because the digraph has an edge (namely, e_1), which goes from vertex 1 to vertex 2. Now comes a point about which you should think a little. a_{12} is the entry in Row 1 and Column 2. Since e_{12} goes *from* 1 *to* 2, by definition, the row number is the number of the vertex at which an edge *begins*, and the column number is the number of the vertex at which the edge *ends*. Think this over and look at the matrix in Example 2 on p. 1013. Since there are three edges that begin at 1 and end at 2, 3, 4, and since there is no edge that begins at 1 and ends at 1 (no loop), the first row of **A** is

$$0 \quad 1 \quad 1 \quad 1.$$

Since the digraph has 4 edges, the matrix **A** must have 4 ones, the three we have just listed and a fourth resulting from the edge that goes from 3 to 1. Obviously, this gives the entry $a_{31} = 1$. In this way you obtain the matrix shown in the answer on p. A44 of the book.

11. **Graph for a given adjacency matrix.** The matrix is 3×3. Hence the graph has 3 vertices. The diagonal entries are always zero since there are no loops. Since all the other entries are 1, every vertex is joined to every other vertex; such a graph is called *complete* (see also Prob. 16). Hence the present graph looks like a triangle. It has 3 edges, in agreement with the fact that each edge in an adjacency matrix creates 2 ones, and our matrix has $3 \cdot 2 = 6$ ones.

19. **Incidence matrix $\widetilde{\mathbf{B}}$ of a digraph.** The incidence matrix of a graph or digraph is an $n \times m$ matrix, where n is the number of vertices and m is the number of edges. Each row corresponds to one of the vertices, and each column to one of the edges. Hence each column contains two ones (in the case of a graph), or a one and a minus one (in the case of a digraph). In Prob. 19 the matrix is square, but just by chance because the number of edges equals the number of vertices. The first column corresponds to the edge e_1, which goes from vertex 1 to vertex 2. Hence, by definition, $\widetilde{b}_{11} = -1$ and $\widetilde{b}_{21} = 1$. Similarly for the other columns shown on p. A45 of the book.

Sec. 21.2 Shortest Path Problems. Complexity

Problem Set 21.2. Page 1019

1. **Shortest path.** s is a vertex that belongs to a hexagon. s has two adjacent vertices, which get the label 1. Each of the latter has one adjacent vertex, which gets the label 2. These two vertices now labeled 2 are adjacent to the last still unlabeled vertex of the hexagon, which thus gets the label 3. This leaves five vertices still unlabeled. Two of these five are adjacent to the vertex labeled 3 and thus get the label 4. The left one labeled 4 is adjcent to the vertex t, which thus gets labeled 5, provided that there is no shorter way for reaching t. There is no shorter way. You could reach t from the right. But the adjacent vertex to the

right of t gets the label 4 because the vertex below it is labeled 3 since it is adjacent to a vertex of the hexagon labeled 2.

9. **Hamiltonian cycle.** For instance, start at s downward, take 3 more vertices on the hexagon H, then the vertex outside H labeled 4, then the vertex inside H, then t, then the vertex to the right of t, then the vertex below it. Then return to H, taking the remaining two vertices of H and return to s.

17. **Postman problem.** The idea is that the postman goes through all the streets (the edges) and returns to the vertex (the post office) from where he came. In the present case the situation is rather obvious. Either he serves vertices 1, 2, 3 first, and then 4, 5, 6 (this corresponds to the solution given on p. A45 in Appendix 2), or conversely. In either case he has to traverse 3 - 4 twice in order to return to the point he started from. This is vertex 1 in the given solution, but any other vertex will do it and will give a shortest walk of the same length (see Prob. 16).

19. **The symbol O.** By definition, an algorithm that is $O(m)$ involves $am + b$ operations. The symbol O is used for conveniently characterizing the behavior of algorithms for large m. Hence you can drop b as being small compared to am. Let k be the number of operations per hour done by the old computer. Then $am = k$ or $m = m_1 = k/a$. The new computer does $100k$ operations per hour. Hence for it, $am = 100k$ or $m = m_2 = 100k/a = 100m_1$, as claimed. For an algorithm that is $O(m^2)$ you can consider cm^2, dropping the term in m and the constant term possibly present. Then you have $cm^2 = k$ for the old computer, hence $m = m_1 = \sqrt{k/c}$. For the new computer you have $cm^2 = 100k$, hence

$$m = m_2 = \sqrt{100k/c} = 10\sqrt{k/c} = 10m_1.$$

Similarly in the other cases.

Sec. 21.3 Bellman's Optimality Principle. Dijkstra's Algorithm

Problem Set 21.3. Page 1023

1. **Shortest path.** By inspection:

Drop 20 because 6 + 14 does the same.
Drop 18 because 6 + 8 is shorter.
Drop 14 because 8 + 4 is shorter.

Dijkstra's algorithm runs as follows. (Sketch the figure yourself and keep it handy while you are working.)

Step 1

1. $L_1 = 0, \tilde{L}_2 = 6, \tilde{L}_3 = 20, \tilde{L}_4 = 18$. Hence $\mathcal{PL} = \{1\}$, $\mathcal{TL} = \{2,3,4\}$. No ∞ appears because each of the vertices 2, 3, 4 is adjacent to 1, that is, is connected to vertex 1 by a single edge.
2. $L_2 = \min(\tilde{L}_2, \tilde{L}_3, \tilde{L}_4) = \min(6, 20, 18) = 6$. Hence $k = 2$, $\mathcal{PL} = \{1,2\}$, $\mathcal{TL} = \{3,4\}$. Thus you started from vertex 1, as always, and added to the set $\mathcal{PL}$ the vertex which is closest to vertex 1, namely vertex 2. This leaves 3 and 4 with temporary labels. These must now be updated. This is Operation 3 of the algorithm (see the table on p. 1022).
3. Update the temporary label $\tilde{L}_3$ of vertex 3,

$$\tilde{L}_3 = \min(20, 6 + l_{23}) = \min(20, 6 + 14) = 20.$$

20 is the old temporary label of vertex 3, and 14 is the distance from vertex 2 to vertex 3, to which you have to add the distance 6 from vertex 1 to vertex 2, which is the permanent label of vertex 2.
Update the temporary label $\tilde{L}_4$ of vertex 4,

$$\tilde{L}_4 = \min(18, 6 + l_{24}) = \min(18, 6 + 8) = 14.$$

18 is the old temporary label of vertex 4, and 8 is the distance from vertex 2 to vertex 4. Vertex 2 belongs to the set of permanently labeled vertices, and 14 shows that vertex 4 is now closer to this set $\mathcal{PL}$ than it had been before. This is the end of Step 1.

Step 2

1. Extend the set $\mathcal{PL}$ by including that vertex of $\mathcal{TL}$ that is cloest to a vertex in $\mathcal{PL}$, that is, add to $\mathcal{PL}$ the vertex with the smallest temporary label. Now vertex 3 has the temporary label 20, and vertex 4 has the temporary label 14. Accordingly, include vertex 4 in $\mathcal{PL}$. Its permanent label is

   $L_4 = \min(\tilde{L}_3, \tilde{L}_4) = \min(20, 14) = 14.$

   Hence you now have $k = 4$, so that $\mathcal{PL} = \{1,2,4\}$ and $\mathcal{TL} = \{3\}$.
2. Update the temporary label $\tilde{L}_3$ of vertex 3,

   $\tilde{L}_3 = \min(20, 14 + l_{43}) = \min(20, 14 + 4) = 18.$

   20 is the old temporary label of vertex 3, and 4 is the distance from vertex 4 (which alredy belongs to $\mathcal{PL}$) to vertex 3.

Step 3

Since only a single vertex, 3, is left in $\mathcal{TL}$, you finally assign the temporary label 18 as the permanent label to vertex 3.

Hence the remaining roads are

from vertex 1 to vertex 2	Length 6
from vertex 2 to vertex 4	Length 8
from vertex 4 to vertex 3	Length 4.

The total length of the remaining roads is 18 and these roads satisfy the condition that they connect all four communities.

Since Dijkstra's algorithm gives a shortest path from vertex 1 to each other vertex, it follows that these shortest paths also provide paths from any of these vertices to every other vertex, as required in the present problem. The solution agrees with the above solution by inspection.

7. **Dijkstra's algorithm.** The procedure is the same as in Example 1 in Sec. 21.3 and as in Prob. 1 just considered. Make a sketch of the graph and use it during your work.

Step 1

1. Vertex 1 gets the permanent label 0. The other vertices get the temporary labels 2 (vertex 2), ∞ (vertex 3), 5 (vertex 4) and ∞ (vertex 5).

   The further work is an application of Operation 2 (assigning a permanent label to the (or a) vertex closest to $\mathcal{PL}$) and Operation 3 (updating the temporary labels of the vertices that are still in the set $\mathcal{TL}$ of the temporarily labeled vertices), in alternating order.
2. $L_2 = 2$ (the minimum of 2, 5, and ∞).
3. $\tilde{L}_3 = \min(\infty, 2 + 3) = 5$.

   $\tilde{L}_4 = \min(5, 2 + 1) = 3$.

   $\tilde{L}_5 = \min(\infty, \infty) = \infty$.

Step 2

1. $L_4 = \min(5, 3, \infty) = 3$. Thus $\mathcal{PL} = \{1,2,4\}$, $\mathcal{TL} = \{3,5\}$. Two vertices are left in $\mathcal{TL}$; hence you have to make two updatings.
2. $\tilde{L}_3 = \min(5, 3 + 1) = 4.$

   $\tilde{L}_5 = \min(\infty, 3 + 4) = 7$.

Step 3

1. $L_3 = \min(4, 7) = 4$.
2. $\tilde{L}_5 = \min(7, 4 + 2) = 6$.

Step 4

1. $L_5 = \tilde{L}_5 = 6.$

Your result is as follows.

Step	Vertex added to $\mathcal{PL}$	Permanent label	Edge added to the path	Length of edge
1	1,2	0,2	(1,2)	2
2	4	3	(2,4)	1
3	3	4	(4,3)	1
4	5	6	(3,5)	2

The permanent label of a vertex is the length of the shortest path from vertex 1 to that vertex. Mark the shortest path from vertex 1 to vertex 5 in your sketch and convince yourself that you have omitted three edges of lengths 3, 4, and 5, the edges retained being shorter.

Sec. 21.4 Shortest Spanning Trees. Kruskal's Greedy Algorithm

Example 1. Application of Kruskal's algorithm (p. 1025) We reproduce the list of double labels (Table 21.5, obtained from the rather simple Table 21.4) and give some further explanations to it.

Vertex	Choice 1 (3, 6)	Choice 2 (1, 2)	Choice 3 (1, 3)	Choice 4 (4, 5)	Choice 5 (3, 4)
1		(1,0)			
2		(1,1)			
3	(3,0)		(1,2)		
4				(4,0)	(1,3)
5				(4,4)	(1,4)
6	(3,3)		(1,3)		

You can now see what the shortest spanning tree looks like, by going through the table line by line. Do this, simultaneously sketching your findings.

Line 1. (1, 0) shows that 1 is a root.

Line 2. (1, 1) shows that 2 is in a subtree with root 1 and is preceded by 1. (This tree consists of the single edge (1, 2).)

Line 3. (3, 0) means that 3 first is a root, and (1, 1) shows that later it is in a subtree with root 1, and then is preceded by 1, that is, joined to the root by a single edge (1, 3).

Line 4. (4, 0) shows that 4 first is a root, and (1, 3) shows that later it is in a subtree with root 1 and is preceded by 3.

Line 5. (4, 4) shows that 5 first belongs to a subtree with root 4 and is preceded by 4, and (1, 4) shows that later 5 is in a (larger) subtree with root 1 and is still preceded by 4. This subtree actually is the whole tree to be found because we are now dealing with Choice 5.)

Line 6. (3, 3) shows that 6 is first in a subtree with root 3 and is preceded by 3, and then later is in a subtree with root 1 and is still preceded by 3.

Problem Set 21.4. Page 1027

1. **Kruskal's algorithm.** Trees constitute a very important type of graph. Kruskal's algorithm is very straightforward. It begins by ordering the edges of a given graph G in ascending order of length. The length of an edge (i, j) is denoted by l_{ij}. Arrange the result in a table similar to Table 21.4 on p. 1026. The

given graph G has $n = 5$ vertices. Hence a spanning tree in G has $n - 1 = 4$ edges, so that you can terminate your table when 4 edges have been chosen.

Edge	Length	Choice
(2, 3)	1	1st
(2, 5)	2	2nd
(1, 3)	3	3rd
(2, 4)	4	4th

The list of double labels looks as follows.

Vertex	Choice 1	Choice 2	Choice 3	Choice 4
	(2, 3)	(2, 5)	(1, 3)	(2, 4)
1			(2, 3)	
2	(2, 0)			
3	(2, 2)			
4				(2, 2)
5		(2, 2)		

Explanation:

Choice 1. (2, 0) because 2 is a root of the subtree consisting of edge (2, 3). Furthermore, (2, 2) because 2 is the root of the subtree to which 3 belongs, and 2 is the predecessor of 3 in this subtree.

Choice 2. (2, 2) because 2 is the root of the subtree to which 5 belongs, and 2 is the predecessor of 5 in this subtree. (Sketch it.)

Choice 3. (2, 3) because 2 is the root of the subtree to which 1 belongs, and 3 is the predecessor of 1 in this subtree. (Sketch it.)

Choice 4. (2, 2) because 2 is the root of the subtree (actually, the tree to be determined) to which 4 now belongs, and 2 is the predecessor of 4 in this subtree. Sketch the tree obtained and compute its length (the sum of the lengths of its edges).

15. Trees that are paths. Let T be a tree with exactly two vertices of degree 1. Suppose that T is not a path. Then it must have at least one vertex v of degree $d \geqq 3$. Each of the d edges incident with v will eventually lead to a vertex of degree 1 (at least one such vertex) because T is a tree, so it cannot have cycles (definition on p. 1015). This contradicts the assumption that T has but *two* vertices of degree 1.

Sec. 21.5 Prim's Algorithm for Shortest Spanning Trees

Problem Set 21.5. Page 1030

7. Shortest spanning tree obtained by Prim's algorithm. In each step, U is the set of vertices of the tree T to be grown, and S is the set of edges of T. The beginning is at vertex 1, as always. The table is similar to that in Example 1 on p. 1030. It contains the initial labels and then in each column the effect of relabeling. Explanations follow after the table.

Vertex	Initial	Relabeling		
		(I)	(II)	(III)
2	$l_{12} = 16$	$l_{24} = 4$	$l_{24} = 4$	-
3	$l_{13} = 8$	$l_{34} = 2$	-	-
4	$l_{14} = 4$	-	-	-
5	$l_{15} = \infty$	$l_{45} = 14$	$l_{35} = 10$	$l_{35} = 10$

1. $i(k) = 1$, $U = \{1\}$, $S = \emptyset$. Vertices 2, 3, 4 are adjacent to vertex 1. This gives their inital labels equal to the length of the edges connecting them with vertex 1 (see the table). Vertex 5 gets the initial label ∞ because the graph has no edge (1, 5); that is, vertex 5 is not adjacent to vertex 1.
2. $\lambda_4 = l_{14} = 4$ is the smallest of the initial labels. Hence include vertex 4 in U and edge (1, 4) as the first edge of the growing tree T. Thus, $U = \{1,4\}$, $S = \{(1,4)\}$.
3. Each time you include a vertex in U (and the corresponding edge in S) you have to update labels. This gives the three numbers in column (I) because vertex 2 is adjacent to vertex 4, with $l_{24} = 4$ (the length of edge (2, 4)), and so is vertex 3, with $l_{34} = 2$ (the length of edge (3, 4)) . Vertex 5 is also adjacent to vertex 4, so that ∞ is now gone and replaced by $l_{45} = 14$ (the length of edge (4, 5)).
2. $\lambda_3 = l_{34} = 2$ is the smallest of the labels in (I). Hence include vertex 3 in U and edge (3, 4) in S. You now have $U = \{1,3,4\}$ and $S = \{(1,4),(3,4)\}$.
3. Column (II) shows the next updating. $l_{24} = 4$ remains because vertex 2 is not closer to the new vertex 3 than to vertex 4. Vertex 5 is closer to vertex 3 than to vertex 4, hence the update is $l_{35} = 10$, replacing 14.
2. The end of the procedure is now quite simple. l_{24} is smaller than l_{35} in column (II), so that you set $\lambda_2 = l_{24} = 4$ and include vertex 2 in U and edge (2, 4) in S. You thus have $U = \{1,2,3,4\}$ and $S = \{(1,4),(3,4),(2,4)\}$.
3. Updating gives no change because vertex 5 is closer to vertex 3, whereas it is not even adjacent to vertex 2.
2. $\lambda_5 = l_{35} = 10$. $U = \{1,2,3,4,5\}$, so that your spanning tree T consists of the edges $S = \{(1,4),(3,4),(2,4),(3,5)\}$.

13. **Prim's algorithm** gives the same tree as Kruskal's algorithm did in Sec. 21.4, but the edges appear in a different order, namely,

Prim	(1,3)	(2,3)	(2,5)	(2,4)
Kruskal	(2,3)	(2,5)	(1,3)	(2,4) .

The table for Prim's algorithm is as follows.

Vertex	Initial	Relabeling		
	label	(I)	(II)	(III)
2	$l_{12} = 5$	$l_{12} = 5$	-	-
3	$l_{13} = 3$	-	-	-
4	$l_{14} = 6$	$l_{14} = 6$	$l_{24} = 4$	$l_{24} = 4$
5	$l_{15} = \infty$	$l_{15} = \infty$	$l_{25} = 2$	-

This table is obtained by the algorithm in the following steps (whose explanation is similar to that in Prob. 1).

1. $i(k) = 1$, $U = \{1\}$, $S = \emptyset$
2. $\lambda = l_{13} = 3$, $U = \{1,3\}$, $S = \{(1,3)\}$
3. No change, continue using the initial labels for vertices 2, 4, 5; see column (I)

2. $\lambda_2 = l_{23} = 1$, $U = \{1,2,3\}$, $S = \{(1,3),(2,3)\}$
3. $l_{24} = 4$, $l_{25} = 2$; see column (II)
2. $\lambda_5 = l_{25} = 2$, $U = \{1,2,3,5\}$, $S = \{(1,3),(2,3),(2,5)\}$
3. $l_{24} = 4$ remains; see column (III)
2. $\lambda_4 = l_{24} = 4$. This gives the vertex set U and the edge set S of the spanning tree obtained, namely, $U = \{1,2,3,4,5\}$, $S = \{(1,3),(2,3),(2,5),(2,4)\}$.

Sec. 21.6 Networks. Flow Augmenting Paths

Problem Set 21.6. Page 1037

1. **Cut set, capacity.** $S = \{1,2,3\}$ is given and T consists of the other vertices, that is, $T = \{4,5,6\}$. Now look at the figure. Draw a curve that separates S from T. Then you will see that the curve cuts the edge (1,4), whose capacity is 10, the edge (5,2), which is a backward edge, the edge (3,5), whose capacity is 5, and the edge (3,6), whose capacity is 13. By definition, the capacity cap(S, T) is the sum of the capacities of the three forward edges,

$$\text{cap}(S,T) = 10 + 5 + 13 = 28.$$

The edge (5, 2) is indeed a backward edge because it goes from vertex 5, which belongs to T, to vertex 3, which is an element of S. And backward edges are not included in the capacity of a cut set, by definition, for reasons given in the text.

11. **Flow augmenting paths** for a given flow in a given network exist only if the given flow is not maximum. 1 - 2 - 3 - 6 is not augmenting because the edge (2, 3) is used to capacity. Similarly, 1 - 4 - 5 - 6 is not augmenting since the edge (5, 6) is used to capacity. For such a small network you can find flow augmenting paths by trial and error (if they exist). For large networks you need an algorithm, such as that of Ford and Fulkerson in the next section.

13. **Flow augmenting path.** The given answer is

$$\Delta_{13} = 8 - 5 = 3, \quad \Delta_{35} = 11 - 7 = 4, \quad \Delta_{56} = 13 - 9 = 4.$$

From this you see that the path 1 - 3 - 5 - 6 is flow augmenting and admits an additional flow

$$\Delta = \min(3,4,4) = 3.$$

Similarly, the path 1 - 4 - 6 is flow augmenting with

$$\Delta = \min(6-3, 4-1) = \min(3,3) = 3.$$

Another flow augmenting path is 1 - 2 - 4 - 6, with

$$\Delta = \min(5-2, 4-2, 4-1) = \min(3,2,3) = 2.$$

Another flow augmenting path is 1 - 2 - 4 - 5 - 6, with

$$\Delta = \min(5-2, 4-2, 5-2, 13-9) = \min(3,2,3,4) = 2.$$

Another flow augmenting path is 1 - 4 - 5 - 6, with

$$\Delta = \min(6-3, 5-2, 13-9) = \min(3,3,4) = 3.$$

Any path containing (4, 3) as a forward edge is not flow augmenting because this edge is used to capacity.

17. **Maximum flow.** The given flow in the network in Prob. 13 is 10, as you can see by looking at the two edges (4, 6) and (5, 6) that go into the target t (the sink 6), or by looking at the three edges leaving vertex 1 (the source s), whose flow is $2 + 5 + 3 = 10$. To find the maximum flow in Prob. 13 by inspection, note the following. Each of the three edges outgoing from vertex 1 could carry an additional flow of 3. Hence you may augment the given flow by 3 by using the path 1 - 4 - 5 - 6. Then the edges(1, 4) and (4, 5) are used to capacity . This increases the given flow from 10 to 13. Next you can use the path 1 - 2 - 4 - 6, whose capacity is

$$\Delta = \min (5 - 2, 4 - 2, 4 - 1) = 2.$$

This increases the flow from 13 to 15. For this new increased flow the capacity of the path 1 - 3 - 5 - 6 is

$$\Delta = \min (3, 4, 13 - 12) = 1$$

because the first increase increased the flow in the edge (5, 6) from 9 to 12. Hence you can increase our flow from 15 to 16. Finally, consider the path 1 - 3 - 4 - 6. The edge (4, 3) is a backward edge in this path. By decreasing the existing flow in this edge from 2 to 1 you can push a flow 1 through this path. Then the edge (4, 6) is used to capacity, whereas the edge (1, 3) is still not fully used. But since both edges going to vertex 6, namely, (4, 6) and (5, 6), are now used to capacity, you cannot augment the flow further, so that you have reached the maximum flow 17. The flows in the edges are

$$f_{12} = 4, \quad f_{13} = 7, \quad f_{14} = 6$$
$$f_{24} = 4$$
$$f_{35} = 8$$
$$f_{43} = 1, \quad f_{45} = 5, \quad f_{46} = 4$$
$$f_{56} = 13.$$

Sketch the network with the new flow and check that Kirchhoff 's law is satisfied at every vertex.

Sec. 21.7 Ford-Fulkerson Algorithm for Maximum Flow

Problem Set 21.7. Page 1040

1. **Maximum flow.** Example 1 in the text shows how you can proceed in applying the Ford-Fulkerson algorithm for obtaining flow augmenting paths until the maximum flow is reached. No algorithms would be needed for the modest problems in our problem sets. Hence the point of this and similar problems is to obtain familiarity with the most important algorithms for basic tasks in this chapter, as they will be needed for solving large-scale real-life problems. Keep this in mind,. to avoid misunderstandings. From time to time look at Example 1 in the text, which is similar and may help you to see what to do next.
   1. The given initial flow is $f = 6$. This can be seen by looking at the flows 2 in edge (1, 2), 1 in edge (1, 3), and 3 in edge (1, 4), whose sum is 6, or, more simply, by looking at the flows 5 and 1 in the two edges (2, 5) and (3, 5), respectively, that end at vertex 5 (the target t).
   2. Label s (= 1) by $\emptyset$. Mark the other edges 2, 3, 4, 5 "unlabeled."
   3. Scan 1. This means labeling the vertices 2, 3, and 4 adjacent to vertex 1 as explained in Step 3 of Table 21. 8 (the table of the Ford-Fulkerson algorithm), which in the present case amounts to the following. $j = 2$ is the first unlabeled vertex in this process, which corresponds to the first part of Step 3 in Table 21.8. You have $c_{12} > f_{12}$ and compute

      $$\Delta_{12} = c_{12} - f_{12} = 4 - 2 = 2 \quad \text{and} \quad \Delta_2 = \Delta_{12} = 2.$$

      Label 2 with the forward label $(1^+, \Delta_2) = (1^+, 2)$.

      $j = 3$ is the second unlabeled vertex adjacent to 1, and you compute

      $$\Delta_{13} = c_{13} - f_{13} = 3 - 1 = 2 \quad \text{and} \quad \Delta_3 = \Delta_{13} = 2.$$

      Label 3 with the forward label $(1^+, \Delta_3) = (1^+, 2)$.

      $j = 4$ is the third unlabeled vertex adjacent to 1, and you compute

      $$\Delta_{14} = c_{14} - f_{14} = 10 - 3 = 7 \quad \text{and} \quad \Delta_4 = \Delta_{14} = 7.$$

      Label 4 with the forward label $(1^+, \Delta_4) = (1^+, 7)$.
   4. Scan 2. This is necessary since you have not yet reached t (vertex 5), that is, you have not yet obtained a flow augmenting path. Adjacent to vertex 2 are the vertices 1, 4, and 5. Vertices 1 and 4 are labeled. Hence the only vertex to be considered is 5. Compute

      $$\Delta_{25} = c_{25} - f_{25} = 8 - 5 = 3.$$

The calculation of Δ_5 differs from the corresponding previous ones. From the table you see that

$$\Delta_5 = \min(\Delta_2, \Delta_{25}) = \min(2,3) = 2.$$

The idea here is that $\Delta_{25} = 3$ is of no help because in the previous edge (1, 2) you can increase the flow only by 2. Label 5 with the forward label $(2^+, \Delta_5) = (2^+, 2)$.

5. You have obtained a first flow augmenting path P: 1 - 2 - 5.
6. You augment the flow by $\Delta_5 = 2$ and set $f = 6 + 2 = 8$.
7. Remove the labels from 2, 3, 4, 5 and go to Step 3. Sketch the given network, with the new flows $f_{12} = 4$ and $f_{25} = 7$. The other flows remain the same as before. You will now obtain a second augmenting path.

3. Scan 1. Adjacent are 2, 3, 4. You have $c_{12} = f_{12}$; edge (1, 2) is used to capacity and is no longer to be considered. For vertex 3 you compute

$$\Delta_{13} = c_{13} - f_{13} = 3 - 1 = 2 \quad \text{and} \quad \Delta_3 = \Delta_{13} = 2.$$

Label 3 with the forward label $(1^+, 2)$. For vertex 4 compute

$$\Delta_{14} = c_{14} - f_{14} = 10 - 3 = 7 \quad \text{and} \quad \Delta_4 = \Delta_{14} = 7.$$

Label 4 with the forward label $(1^+, 7)$.

3. You need not scan 2 because you now have $f_{12} = 4$ so that $c_{12} - f_{12} = 0$; (1, 2) is used to capacity; the condition $c_{12} > f_{12}$ in the algorithm is not satisfied. Scan 3. Adjacent to 3 are the vertices 4 and 5. For vertex 4 you have $c_{43} = 6$ but $f_{43} = 0$, so that the condition $f_{43} > 0$ is violated. Similarly, for vertex 5 you have $c_{35} = f_{35} = 1$, so that the condition $c_{35} > f_{35}$ is violated and you must go on to vertex 4. Scan 4. The only unlabeled vertex adjacent to 4 is 2, for which you compute

$$\Delta_{42} = c_{42} - f_{42} = 5 - 3 = 2$$

and

$$\Delta_2 = \min(\Delta_4, \Delta_{42}) = \min(7,2) = 2.$$

Label 2 with the forward label $(4^+, 2)$.

4. Scan 2. Unlabeled adjacent to 2 is vertex 5. Compute

$$\Delta_{25} = c_{25} - f_{25} = 8 - 7 = 1$$

and

$$\Delta_5 = \min(\Delta_2, \Delta_{25}) = \min(2,1) = 1.$$

Label 5 with the forward label $(2^+, 1)$.

5. You have obtained a second flow augmenting path P: 1 –4 –2 –5.
6. Augment the existing flow 8 by $\Delta_5 = 1$ and set $f = 8 + 1 = 9$.
7. Remove the labels from 2, 3, 4, 5 and go to Step 3. Sketch the given network with the new flows, write the capacities and flows in each edge, obtaining edge (1, 2): (4, 4), edge (1, 3): (3, 1), edge (1, 4): (10, 4), edge (2, 5): (8, 8), edge (3, 5): (1, 1), edge (4, 2): (5, 4), and edge (4, 3): (6, 0). You see that the two edges going into vertex 5 are used to capacity, hence the flow is maximum. Indeed, the algorithm shows that vertex 5 can no longer be reached.

Sec. 21.8 Assignment Problems, Bipartite Matching

Problem Set 21.8. Page 1045

3. **A graph that is not bipartite.** Proceed in the order of the numbers of the vertices. Put vertex 1 into S and its adjacent vertices 2, 3, 4 into T. Then consider 2, which is now in T. Hence for the graph to be bipartite, its adjancent vertices 1 and 4 should be in S. But vertex 4 has just been put into T. This contradicts and shows that the graph is not bipartite.

5. **Bipartite graph.** Since graphs can be graphed in different ways, one cannot see immediately what is going on. Hence in the present problem you have to proceed systematically.

1. Put vertex 1 into S and all its adjacent vertices 2, 4, 6 into T.
2. Vertex 2 is in T. Put its adjacent vertices 1, 3, 5 into S.
3. Vertex 3 is in S. For the graph to be bipartite, its adjacent vertices 2, 4, 6 should be in T, as is the case.
4. Vertex 4 is in T. Its adjacent vertices 1, 3, 5 are in S, as it should be for a bipartite graph.
5. Vertex 5 is in S. Hence for the graph to be bipartite, its adjacent vertices 2, 4, 6 should be in T, as is the case.
6. Vertex 6 is in T, and its adjacent vertices 1, 3, 5 are in S. Since none of these six steps gave any contradiction, conclude that the given graph is bipartite. Sketch this graph in a form that one can see immediately that it is bipartite. Try to write this method in the form of an algorithm and apply it to Prob. 3.

11. **Algorithm for maximum cardinality matching.** Let $S = \{1,2,3\}$. Then $T = \{4,5,6\}$. (You could equally well let $S = \{4,5,6\}$ and $T = \{1,2,3\}$.) Use the algorithm in Table 21.9 on p. 1044. The given matching is $M = \{(1,4),(3,6)\}$.
    1. An exposed vertex is one that is not endpoint of an edge in M. Hence the vertices 2 and 5 are exposed. Vertex 2 is in S and gets the label $\emptyset$.
    2. Vertex 1 is in S and (1, 5) is not in M and 5 is unlabeled. Hence label 5 with 1. Vertex 2 is in S and (2, 5), (2, 6) are not in M and 6 is unlabeled (whereas 5 is labeled). Hence label 6 with 2. Vertex 3 is in S and (3, 4), (3, 5) are not in M and 4 is unlabeled. Hence label 4 with 3.
    3. Vertices 4 and 6 are nonexposed in T. Hence label 1 with 4 (recall that (1, 4) is an edge in M), and label 3 with 6 (recall that (3, 6) is an edge in M).
    4. Now comes backtracking. The only exposed vertex in T is 5. Its label is 1. Hence add (1, 5) as the first edge of the augmenting path P to be obtained. Now vertex 1 has the label 4, and you add (1, 4) to P as the next edge. (Recall that (1, 4) is in M). Vertex 4 has the label 3. Hence add (3, 4) to P. Vertex 3 has the label 6 and you add (3, 6) (which is in M) to P. Vertex 6 has the label 2 and you add (2, 6) to P. The augmenting path obtained is
    $$P : (2,6),(3,6),(3,4),(1,4),(1,5).$$
    5. You now augment the given matching by removing its edges (3,6) and (1,4) from P and taking the other three edges of P, namely,
    $$(2,6),(3,4),(1,5),$$
    as your new matching. You now have a matching of cardinality 3. You need not look for further augmenting paths because this is the maximum possible cardinality if S or T consists of 3 vertices each. Since no vertex is left exposed in the new matching, this is a complete matching, a term mentioned on p. 1042.

19. K_4 ***is planar*** because you can graph it as a square A,B,C,D, then add one diagonal, say, A,C, inside, and then join B,D not by a diagonal inside (which would cross) but by a curve outside the square.