CSE 5311 Homework #2 Solutions

1 MST Using Warshall's Algorithm

The results are as follows:

1	2	0.14
1	4	0.21
3	4	0.12
4	5	0.11
5	6	0.25
6	7	0.21
6	9	0.23
8	9	0.19

2 Problem 24.2-4, p. 510

When the edge weights are integers in the range 1 to $|V|$, then line 4 (p. 505) of Kruskal's algorithm can be done using the counting sort in $O(E)$ time but the disjoint-set forest operations still take $O(E \lg E)$ time. So the total running time remains $O(E \lg E)$.

When the edge weights are integers in the range 1 to W for some constant W, it doesn't change anything. Running time remains $O(E \lg E)$.

3 Problem 24.2-5, p. 510

If the edge weights are integers in the range 1 to W and W is a very small constant (say 3, i.e., all edge weights are either 1, 2 or 3), then the priority queue can be maintained as a doubly-linked list with the operation Extract-Min taking $O(1)$ time and then the running time or Prim's algorithm becomes $O(|V| + |E|)$.

If the edge weights are integers in the range 1 to $|V|$ it doesn't really change anything except when $|V|$ is very small the the W chosen above.

4 Problem 27.2-2, p. 599

Figure 1 gives the results using Edmonds-Karp. Augmenting paths are denoted by dashed lines.

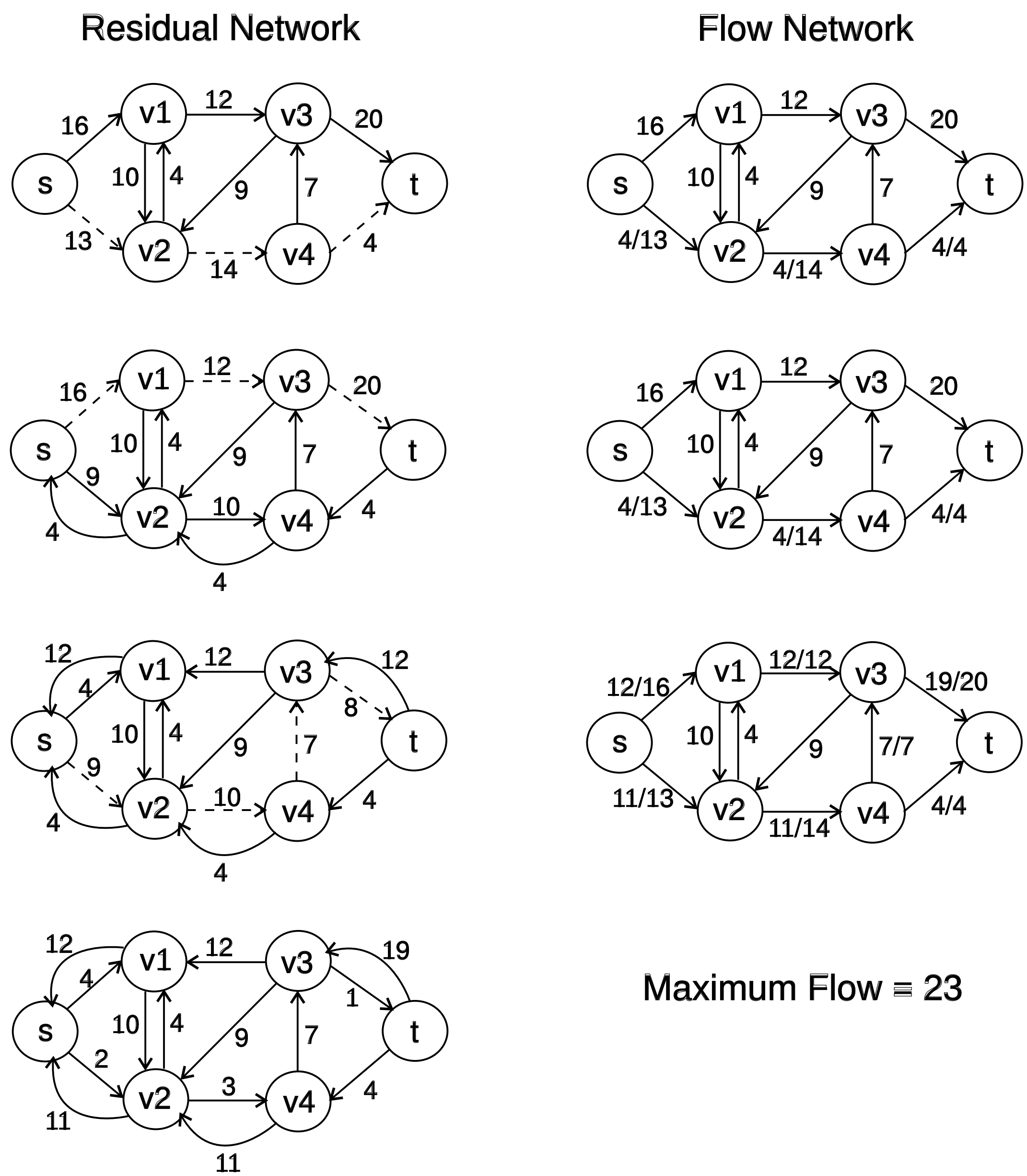

Figure 1: Edmonds-Karp Solution

6	10	
0	5	
0	1	16
0	2	13
1	3	12
1	2	10
2	1	4
2	4	14
3	2	9
3	5	20
4	3	7
4	5	4

Table 1: Input File for `preflowPush.c`

For the preflow-push results you may use `preflowPush.c` located in the public directory (`/public/cse/5311`) on omega. Table 1 gives an input file which encodes the figure from the book into the proper format for that program. The source is assigned a node number of 0 and the sink a node number of 5. The other node numbers correspond to the numbers in the diagram.

5 Lattice for Stable Marriage Problem

Figure 2 shows the male-oriented lattice.

6 Problem 23-1, p. 495

a. Undirected Graph with breadth-first search (BFS):

- Because of the properties of BFS–the way nodes are visited in BFS, there can be no back and forward edges. The edges that constitute as back and forward edges in depth-first search (DFS), have already been visited through the parent and are tree edges in a BFS of an undirected graph.
- Again, because of the way nodes are visited in a BFS, for each tree edge (u, v), $d[u]$ has to be equal to $d[u] + 1$.
- Two kinds of situations may arise where there can be cross edges. These are depicted in figure 3. In one case $d[v] = d[u]$ and in the other case $d[v] = d[u]+1$.

b. Directed Graph with BFS:

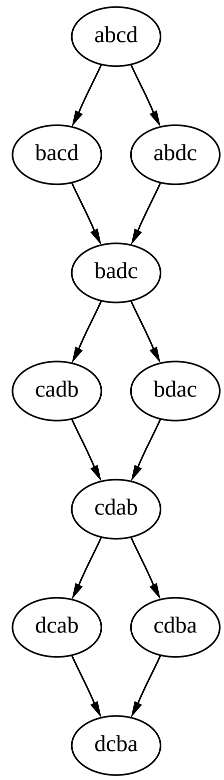

Figure 2: Stable-Marriage Lattice

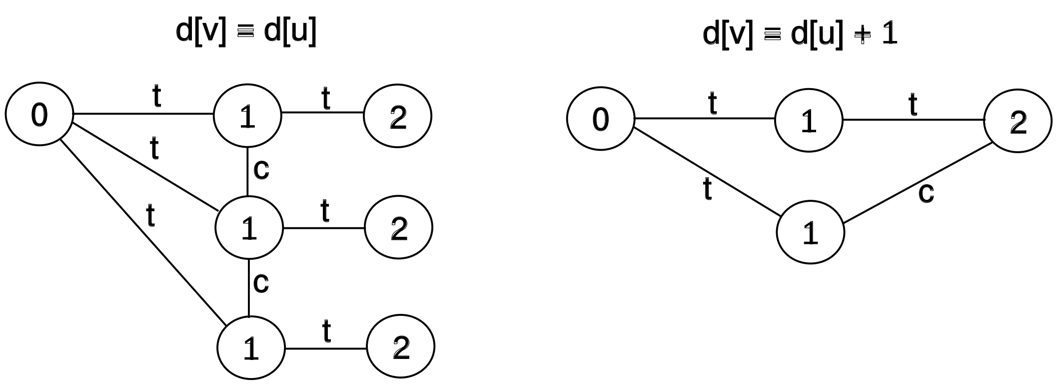

Figure 3: Undirected Graph Using BFS

- Because the graph is directed, it can now have back edges. However, there are still no forward edges, because of the way nodes are visited in a BFS. The forward edges of a DFS are tree edges in a BFS already visited by the parent node.
- Same as a) 2 above.

- Same as a) 2 above. So, $d[v] \leq d[u] + 1$.
- Two examples of a back edge in a BFS of a directed graph are given in figure 4. In one case we have $d[v] < d[u]$ and in the other case we have $0 \leq d[v] < d[u]$.

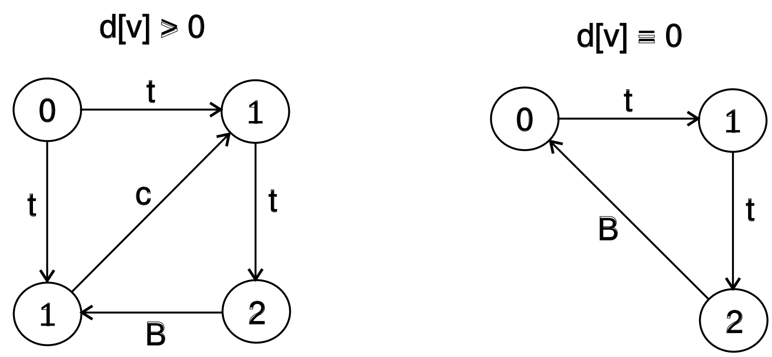

Figure 4: Directed Graph Using BFS

7 Max Flow Using Preflow-Push

The following output was obtain by using `preflowPush.c` located in the public directory (`/public/cse/5311`) on omega. The source is assigned a node number of 0 and the sink a node number of 3. The nodes labelled A and B are given labels of 1 and 2, respectively.

```
debug:  after initialization
  i height excess
  0      4      0
  1      0     10
  2      0     50
  3      0      0
  i   j capacity ...flow
  0   1       10      10
  0   2       50      50
  1   3      100       0
  2   3       20       0
debug:  (1,0) changed minHeight to 4
debug:  (1,3) changed minHeight to 0
debug:  lifting 1 from 0 to 1
debug:  (2,0) changed minHeight to 4
debug:  (2,3) changed minHeight to 0
debug:  lifting 2 from 0 to 1
```

```
debug:
  i height excess
  0      4      0
  1      1     10
  2      1     50
  3      0      0
  i    j capacity ...flow
  0    1       10      10
  0    2       50      50
  1    3      100       0
  2    3       20       0
debug:  pushing 10 units from 1 to 3
debug:  pushing 20 units from 2 to 3
debug:
  i height excess
  0      4      0
  1      1      0
  2      1     30
  3      0     30
  i    j capacity ...flow
  0    1       10      10
  0    2       50      50
  1    3      100      10
  2    3       20      20
debug:  (2,0) changed minHeight to 4
debug:  lifting 2 from 1 to 5
debug:
  i height excess
  0      4      0
  1      1      0
  2      5     30
  3      0     30
  i    j capacity ...flow
  0    1       10      10
  0    2       50      50
  1    3      100      10
  2    3       20      20
debug:  pushing 30 units from 2 to 0
debug:
  i height excess
  0      4     30
```

```
  1      1      0
  2      5      0
  3      0     30
  i   j capacity ...flow
  0   1       10      10
  0   2       50      20
  1   3      100      10
  2   3       20      20
debug:
  i height excess
  0      4     30
  1      1      0
  2      5      0
  3      0     30
  i   j capacity ...flow
  0   1       10      10
  0   2       50      20
  1   3      100      10
  2   3       20      20
final result:
  i height excess
  0      4     30
  1      1      0
  2      5      0
  3      0     30
  i   j capacity ...flow
  0   1       10      10
  0   2       50      20
  1   3      100      10
  2   3       20      20
```

8 KNP Fail Links

The Knuth-Morris-Pratt fails links (both methods) for the search pattern *abracadabra* are given in the table below:

	a	b	r	a	c	a	d	a	b	r	a
Method 1	0	1	1	1	2	1	2	1	2	3	4
Method 2	0	1	1	0	2	0	2	0	1	1	0

9 Complexity of Recursive Matrix Multiplication

Suppose that matrix multiplication is implemented in a recursive decomposition fashion like Strassen's methods. However, instead of using his equations we use the everyday ones, i.e., $C_{ij} = A_{i1} * B_{1j} + A_{i2} * B_{2j}$. What is the asymptotic complexity, based on the number of scalar multiplies and additions/subtractions?

For $C = AB$, we divide each of A, B, C into four $\frac{n}{2} \times \frac{n}{2}$ matrices. Then,

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

Then according to: $C_{ij} = A_{i1} * B_{1j} + A_{i2} * B_{2j}$ we have:

r = ae + bf
s = ag + bh
t = ce + df
u = cg + dh

Each of these four equations specifies two multiplications of $\frac{n}{2} \times \frac{n}{2}$ and addition of $\frac{n}{2} \times \frac{n}{2}$ products. Then the number of multiplies is 8 and the number of additions is 4.

1. Let $M(k)$ be the # of scale multiplies for $n = 2$. Then,

$$\begin{array}{rcl} M(0) & = & 1 \\ M(1) & = & 8 \\ M(k) & = & 8M(k-1) \\ \hline M(n) & = & n^3 \end{array}$$

2. Let $P(k)$ be the # of additions for $n = 2$. Then,

$$\begin{array}{rcl} P(0) & = & 0 \\ P(1) & = & 4 \\ P(k) & = & 8P(k-1) + n^2 \\ \hline P(n) & = & 8P(n/2) + n^2 \\ P(n) & = & \Theta(n^3) \end{array}$$

So, the asymptotic complexity is $\Theta(n^3)$.