CS302 Data Structures

Spring 2010 – Dr. George Bebis

Homework 6 - Solutions

1. Exercise 9 (page 599)

(a) The highest-priority element is the one with the smallest time stamp

(b)

Enqueue

Assign the next largest time stamp to the new element Put new element in the queue (position is unimportant)

```
template <typename ItemType>
void QueType<ItemType>::Enqueue( ItemType newItem )
{
    QueNode<ItemType> tmp;
    tmp.data = newItem;
    tmp.priority = nextTimestamp;
    nextTimestamp++;
    PQType<ItemType>::Enqueue( tmp );
}
```

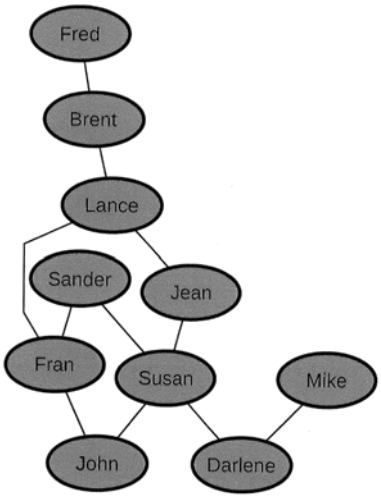
Dequeue

Find the element with the smallest time stamp Assign this element to item (to be returned) Remove this element from the queue

```
template <typename ItemType>
void QueType<ItemType>::Dequeue( ItemType item )
{
    QueNode<ItemType> tmp;
    PQType<ItemType>::Dequeue( tmp );
    item = tmp.data;
}
```

(c) `Enqueue`, like those in Chapter 5, has 0(1). `Dequeue` has O(N), as compared with the 0(1) operations developed in Chapter 5. If the priority queue is implemented using a heap with the smallest value the highest priority, `Enqueue` has O(1) and `Dequeue` has O($\log_2 N$).

2. Exercise 11, 12, 13 (pages 600, 601) To get credit, demonstrate DFS and BFS like in the examples we did in class.



[0]	Brent
[1]	Darlene
[2]	Fran
[3]	Fred
[4]	Jean
[5]	John
[6]	Lance
[7]	Mike
[8]	Sander
[9]	Susan

Table 1: `EmployeeGraph` adjacentcy matrix

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
[0]	0	0	0	1	0	0	1	0	0	0
[1]	0	0	0	0	0	0	0	1	0	1
[2]	0	0	0	0	0	1	1	0	1	0
[3]	1	0	0	0	0	0	0	0	0	0
[4]	0	0	0	0	0	0	1	0	0	1
[5]	0	0	1	0	0	0	0	0	0	1
[6]	1	0	1	0	1	0	0	0	0	0
[7]	0	1	0	0	0	0	0	0	0	0
[8]	0	0	1	0	0	0	0	0	0	1
[9]	0	1	0	0	1	1	0	0	1	0

Iter:	
1	Init: E9
2	D9 M9 E1,4,5,8
3	D1 M1 E7
4	D4 M4 E6
5	D5 M5 E2
6	D8 M8 E2
7	D7 M7 EØ
8	D6 Stop

←F			R→
9			
1	4	5	8
4	5	8	7
5	8	7	6
8	7	6	2
7	6	2	2
6	2	2	
2	2		

BFS

Iter:	1	2	3	4	5
	Initialize stack	Pop 9 Mark 9 Push 1, 4, 5, 8	Pop 8 Mark 8 Push 2	Pop 2 Mark 2 Push 5, 6	Pop 6 Done
	9	8 5 4 1	2 5 4 1	6 5 5 4 1	5 5 4 1

DFS

3. Exercice 19 (page 599)

V(StateGraph) = {Oregon, Alaska, Texas, Hawaii, Vermont, NewYork, California}
E(StateGraph) = {(Alaska, Oregon), (Hawaii, Alaska), (Hawaii, Texas), (Texas, Hawaii), (Hawaii, California), (Hawaii, New York), (Texas, Vermont), (Vermont, California), (Vermont, Alaska)}

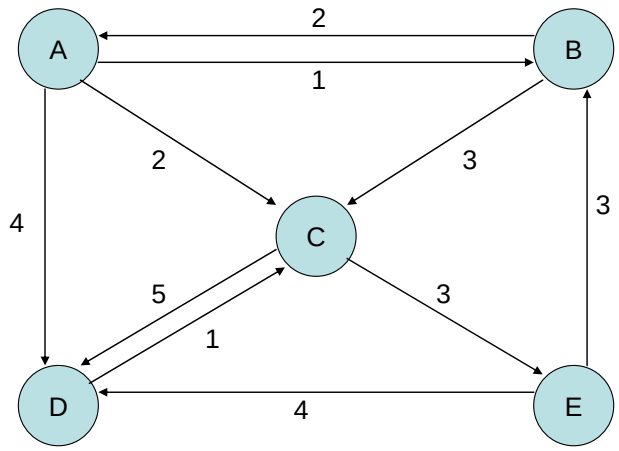
Note: The order of the elements in the sets in not important.

4. Exercise 24 (page 603)

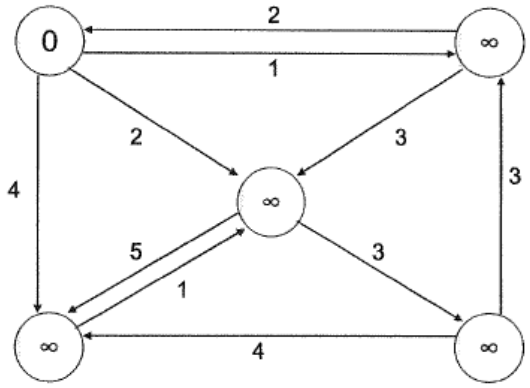
Deleting a vertex is more complicated than deleting an edge for two reasons. First, in addition to removing the vertex from the set of vertices, we must also remove the edges to all its adjacent vertices in the set of edges. Second, we must decide what to do about the now unused vertex number. The best solution is to keep a list of returned vertex numbers and assign new numbers from there first.

```
template <typename ItemType>
void GraphType<ItemType>::DeleteVertex( ItemType vert )
{
    int i=0, j;
    while( i<numVertices && (not (vert==vertices[i])) )
    {
        i++;
    }
    if ( i!=numVertices )
    {
        vertices[i] = vertices[numVertices-1];
        for (j=0; j<maxVertices-1; j++)
        {
            edges[i][j] = edges[maxVertices-1][j];
            edges[j][i] = edges[j][maxVertices-1];
        }
        numVertices--;
    }
}
```

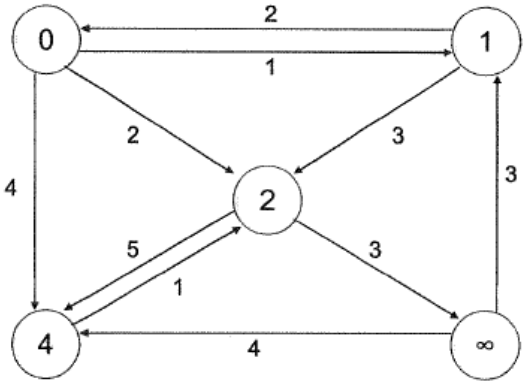
5.



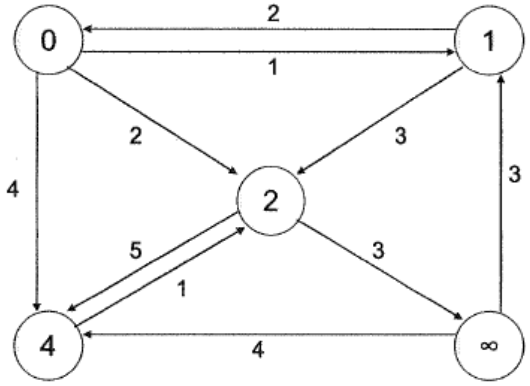
S={} Q={A,B,C,D,E}



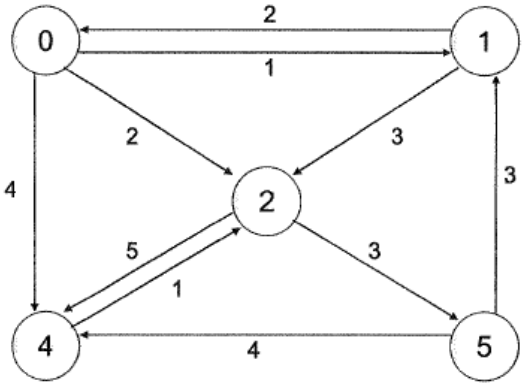
S={A} Q={B,C,D,E}



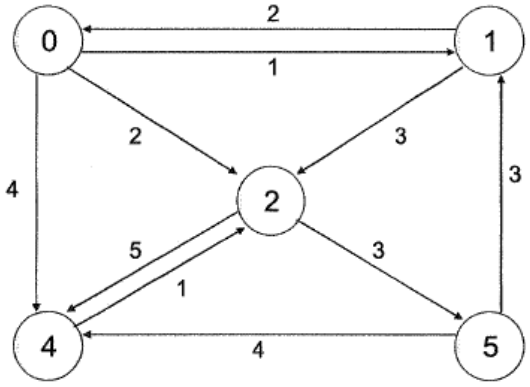
S={A,B} Q={C,D,E}



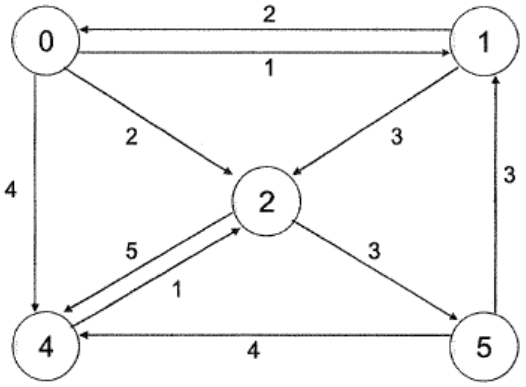
S={A,B,C} Q={D,E}



S={A,B,C,D} Q={E}

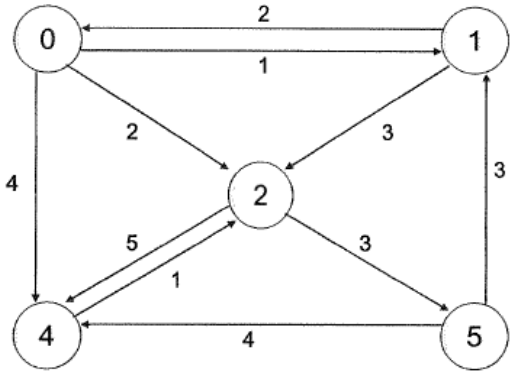


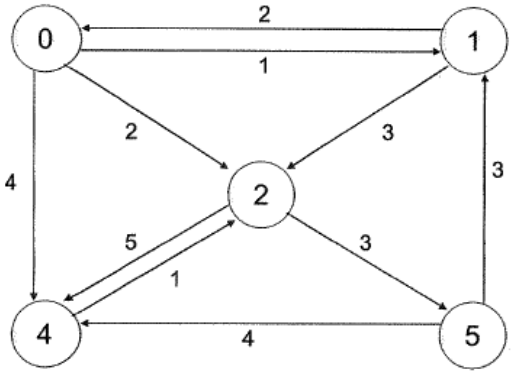
S={A,B,C,D,E} Q={}

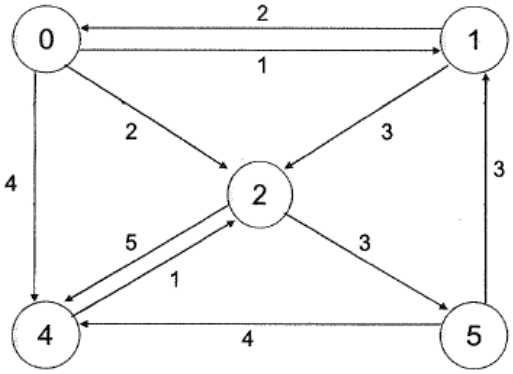


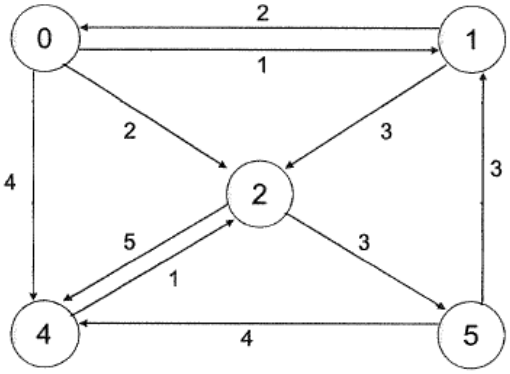
Dijkstra's Algorithm

(A,B), (A;C), (A,D), (B,A), (B,C), (C,D), (C,E), (D,C), (E,B), (E,D)

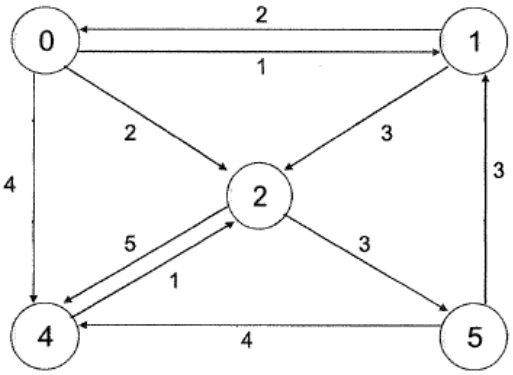








Stable!



Bellman Ford's Algorithm

- 4 passes (since there are 5 vertices)
- One more pass to check for negative cycles
- Note that results depend on the order of relaxing the edges.