COMP2421—DATA STRUCTURES AND ALGORITHM

Queues

Dr. Radi Jarrar
Department of Computer Science
Birzeit University

Queues

- Queues are lists in which insertion is done at the end, and deletion is done at the beginning
- Elements are inserted and removed according to FIFO (First-In, First-Out) principle
- Elements can be inserted in queue at anytime. Only the element that has been in the queue the longest can be removed

Queue Operations

- Operations
  - Enqueue: Insert element at the end (called Rear)
  - Dequeue: Delete and return the element at the beginning (called Front)

Implementing Queues

- It can be noticed that operations in Queues and Stacks give O(1) running time (we are not traversing elements)
- Linked List implementation of Queue is quite straightforward:
  - Insert at the end of the list
  - Remove from beginning

Array implementation of Queues

- Implementing queue using arrays
  - Array queue[]
  - Position front
  - Position rear
  - Number of elements in the queue: q_size
- When enqueue is called, increment q_size and rear. Set queue[rear]=X

Array implementation of Queues (2)

- How to keep track of the front and rear elements?
  - Let queue[0] is the front element. Issues?

- Shift all elements every time we do dequeue. This means it takes O(n) running time. We need to achieve constant time

Array implementation of Queues (3)

- To avoid shifting elements once they are inserted, keep track of three variables: **front**, **rear**, and **size.**
- Front: the index where the first element in the queue is stored.
- Rear: the index of the last element in the queue.
- Size: returns the number of elements in the queue.

Array implementation of Queues (4)

- Initialise the queue: set front = rear = 0, size = 0

Circular Queue

- The solution is to use circular queue
- In circular queue: set front = 1, rear = 0, and size = 0
- Every time we increment front or rear, we simply compute the increment as “(front + 1) mod N” or “(rear +1) mod N”

Implementation of Queues

```
#include<stdio.h>
#define MinQueueSize 5

struct QueueRecord{
    int Capacity;
    int Front;
    int Rear;
    int Size;
    int *Array;
};
```

Implementation of Queues (2)

```
typedef struct QueueRecord *Queue;


int IsEmpty( Queue Q ){
    return Q->Size == 0;
}
```

Implementation of Queues (3)

```
int IsFull( Queue Q ){
    return Q->Size == Q->Capacity;
}
```

Implementation of Queues (4)

```
Queue CreateQueue( int MaxElements ){
      Queue q;

      if( MaxElements < MinQueueSize )    printf("Queue size is too small\n);

      Q = (Queue)malloc(sizeof( struct QueueRecord ));
      if( Q == NULL)    printf("Out of space");

      Q->Array = (int*)malloc(sizeof(int) * MaxElements);

      if( Q->Array == NULL )  printf("Out of space");

      Q->Capacity = MaxElements;
      MakeEmpty( Q );
      return Q;}
```

Implementation of Queues (5)

```
void MakeEmpty( Queue Q){
    Q->Size = 0;
    Q->Front = 1;
    Q->Rear = 0;
}
```

Implementation of Queues (6)

```
void DisposeQueue( Queue Q ){
    if( Q != NULL ){
        free( Q->Array );
        free( Q );
    }
}
```

Implementation of Queues (7)

```
int Succ( int Value, Queue Q ){
    if( ++Value == Q->Capacity )
        Value = 0;

    return Value;
}
```

Implementation of Queues (8)

```
void Enqueue( int X, Queue Q ){
    if( IsFull( Q ) )
        printf( "Full Queue" );
    else{
        Q->Size++;
        Q->Rear = Succ( Q->Rear, Q );
        Q->Array[ Q->Rear ] = X;
    }
}
```

Implementation of Queues (9)

```
int Front( Queue Q ){
    if( !IsEmpty( Q ) )
        return Q->Array[ Q->Front ];

    printf("Empty Queue!");

    return 0;
}
```

Implementation of Queues (10)

```
void Dequeue( Queue Q ){
    if( IsEmpty( Q ) )
        printf("Empty Queue!");
    else{
        Q->Size--;
        Q->Front = Succ( Q->Front, Q );
    }
}
```

Implementation of Queues (11)

```
int FrontAndDequeue( Queue Q ){
     int X = 0;

     if( IsEmpty( Q ) )
          printf("Empty Queue!");
     else{
          Q->Size--;
          X = Q->Array[ Q->Front ];
          Q->Front = Succ( Q->Front, Q );
     }
     return X;
}
```

Implementation of Queues (12)

```
int main( ){
      Queue q;
      int i;

      q = CreateQueue( 12 );

      for( i=0; i<10; i++ )    Enqueue(i, q);
      while( !IsEmpty( q )){
            printf("%d\n", Front( q ));
            Dequeue( q );
      }

      DisposeQueue ( q );
      return 0;
}
```