Queues

D.E ZEGOUR
École Supérieure d'Informatique
ESI

Queues

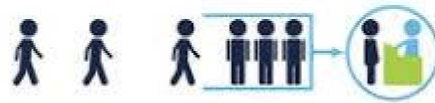
Definition, Principle, Application Domains

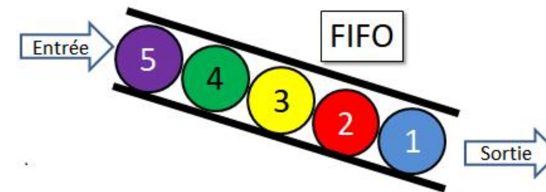
Collection of items in which

- any new item is inserted at the end, and

- any item can only be removed from the beginning.

Principle : FIFO,
First In, First Out
first entered, first served





Used
- in computer operating systems,
- simulation problems.

Also used for traversing trees and solving many other problems

Queues

Abstract Machine

CREATEQUEUE (Q) :

Create an empty queue.

EMPTY_QUEUE (Q) :

Test if queue Q is empty.

ENQUEUE (Q, Val) :

Enqueue (add to tail) value Val to queue Q.

DEQUEUE (Q, Val) :

Dequeue (remove from the head) a value from queue Q and put it in Val.

Queues

Implementation

Static : using arrays (by stream - by shifting - by a circular array)

Dynamic : using linked lists

Queues

Static Implementation by stream

Initially, an empty zero-based array with a maximum of Max elements

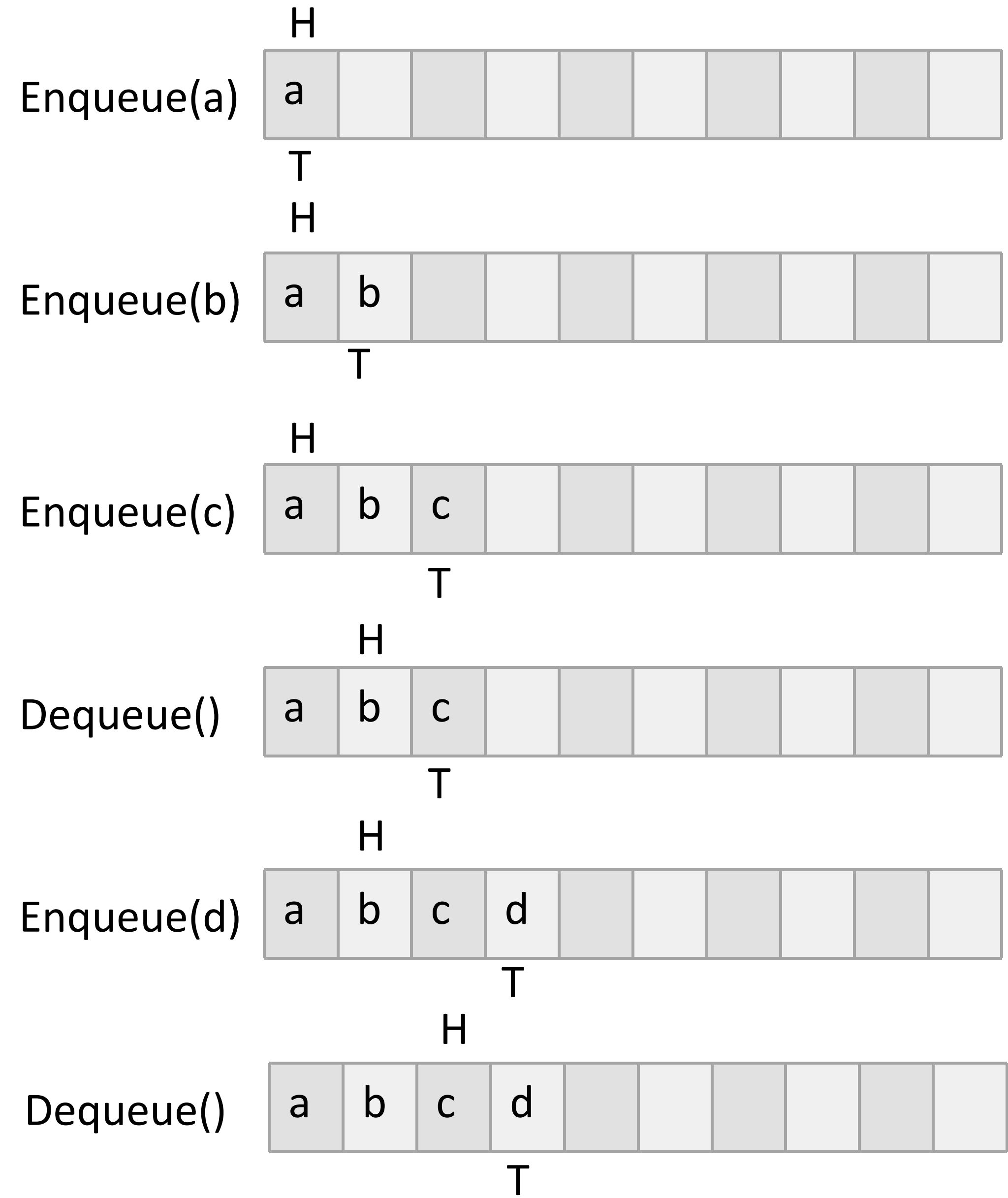
- The number of elements is equal to T - H + 1

The queue is not empty if T ≥ H,
so it is empty if not (T ≥ H), meaning T < H.

- The queue is full if T = Max - 1.

- Initialization is done as follows: H = 0, T = -1.

The queue advances in a stream, and therefore, the elements before the head H are not retrievable



Queues

C Static Implementation by stream

```
#define Max 100
typedef int Bool;
typedef int Anytype;
struct Typequeue
 {
   Anytype Elements[Max];
   int Head, Tail;
 };
void Createqueue ( struct Typequeue *Q )
  {
    (*Q).Head = 0;
    (*Q).Tail = -1;
  }
Bool Empty_queue ( struct Typequeue Q )
  {
    return ( Q.Head > Q.Tail);
  }
Bool Full_queue ( struct Typequeue Q )
  {
    return ( Q.Tail== Max-1 );
  }
```

```
  void Enqueue ( struct Typequeue *Q, Anytype Val )
   {
     if ( ! Full_queue(*Q) )
       {
          (*Q).Tail++;
          (*Q).Elements[(*Q).Tail] = Val;
       }
     else   printf( " %s", "Overflow");
    }
 void Dequeue ( struct Typequeue *Q, Anytype *Val )
   {
     if ( ! Empty_queue(*Q) )
       {
         *Val = (*Q).Elements[(*Q).Head];
          (*Q).Head++-;
       }
     else   printf(" %s", "Underflow");
 }
  int main(int argc, char *argv[])
   {    system("PAUSE");      return 0;  }
```

Queues

Static Implementation by shifting

Initially, an empty Array Table(0..Max]

- The number of elements is equal to T + 1.

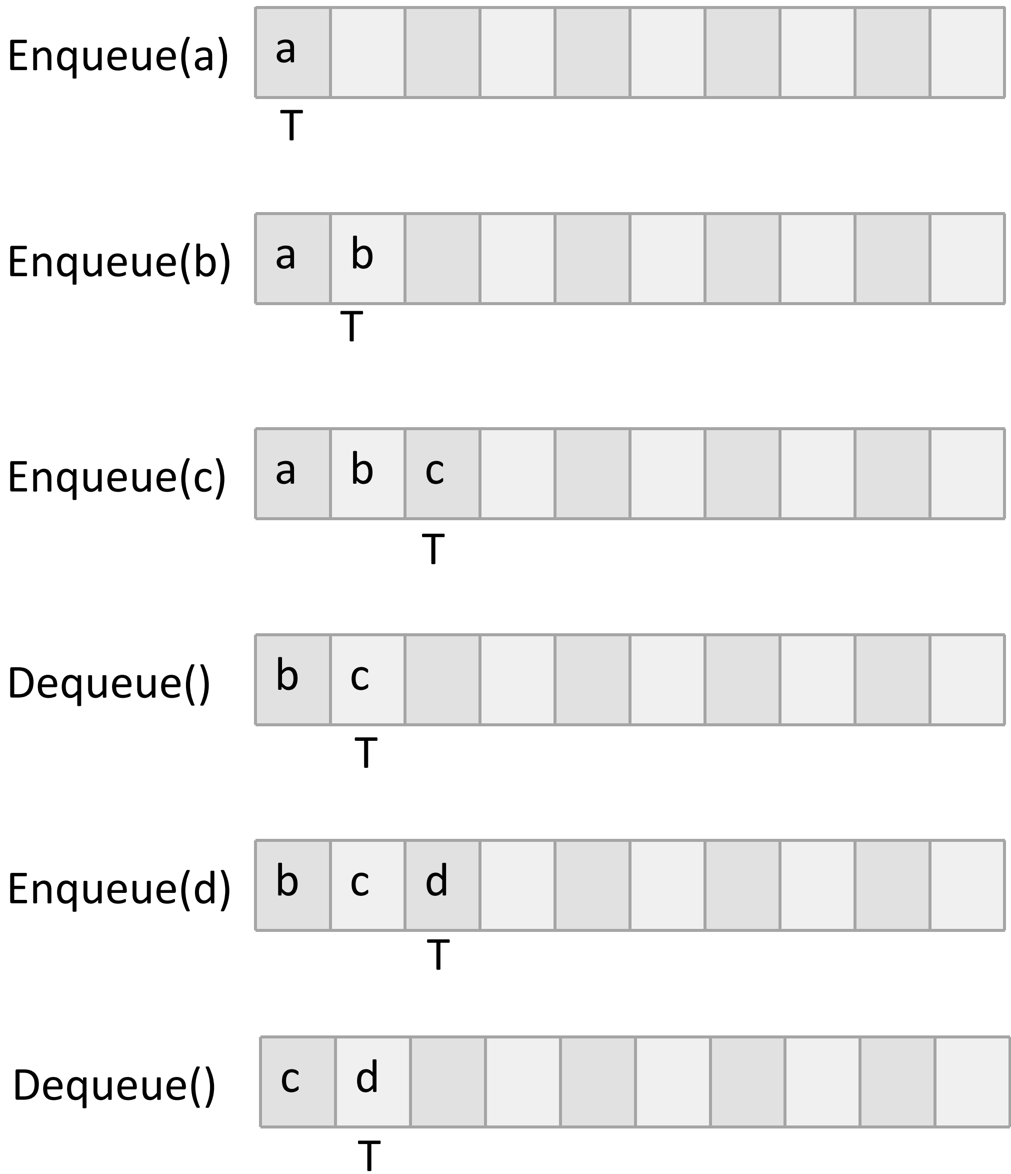
- The queue is not empty if T ≥ 1, so it is empty if T < 1.

- Initialization is done by setting T := - 1.

We don’t need the head

- The queue is full if T is equal to Max - 1.

Drawback : for each dequeue operation, we perform a shift of T-1 elements.



Queues

C Static Implementation by shifting

```
#define Max 100
typedef int Bool;
typedef int Anytype;
 struct Typequeue
 {
   Anytype Elements[Max];
    int Tail;
  } ;
struct Typequeue Q;
void Createqueue ( struct Typequeue *Q)
  {
     (*Q).Tail= -1;
  }
Bool Empty_queue ( struct Typequeue Q)
  {
      return ( Q.Tail== -1 );
  }
Bool Full_queue ( struct Typequeue Q)
  {
      return ( Q.Tail== Max-1 );
  }
```

```
void Enqueue ( struct Typequeue *Q, Anytype Val )
  {
     if ( ! Full_queue(*Q) )
       {
           (*Q).Tail++;
           (*Q).Elements[(*Q).Tail] = Val;
       }
     else    printf(" %s", "Overflow");
   }
 void Dequeue ( struct Typequeue *Q, Anytype *Val )
   {int I;
     if ( ! Empty_queue (*Q) )
       {
         *Val = (*Q).Elements[0];
          for(I=0; I<=(*Q).Tail-2; I++)
            (*Q).Elements[I] =(*Q).Elements[I + 1];
          (*Q).Tail--;
       }
     else   printf(" %s", "Underflow");
}
int main(int argc, char *argv[])
   {   system("PAUSE");    return 0;  }
```

Queues

Static Implementation by a circular array

Let's go back to the stream-based solution
and try to use the array in a circular manner.

How to initialize the queue ?

T < H not possible (counter example)

T > H not possible (counter exemple)

T = H not possible (case where there is one element left in the queue

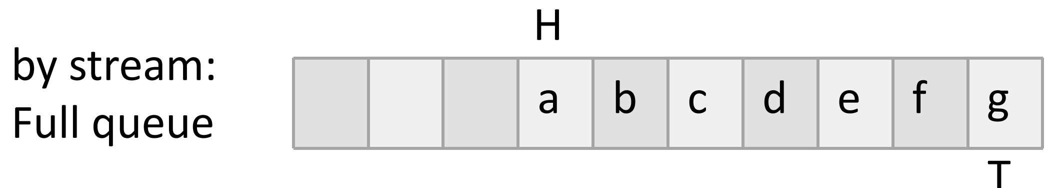
Solution :

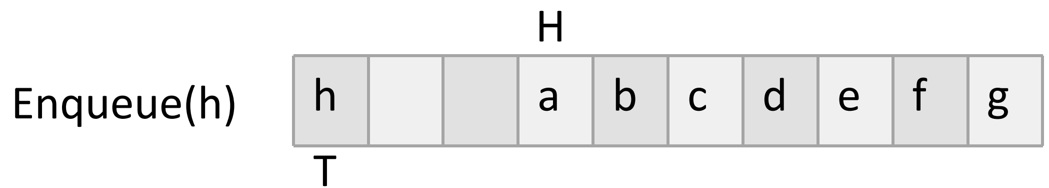
H : points the item that précède the first.

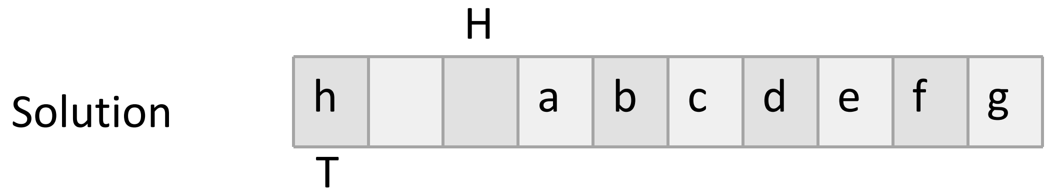
T: points the last item.

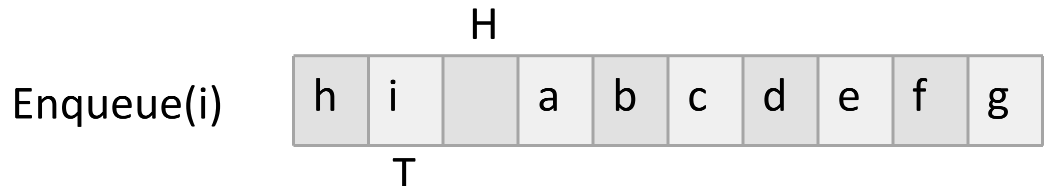
(T= H) : case empty queue.

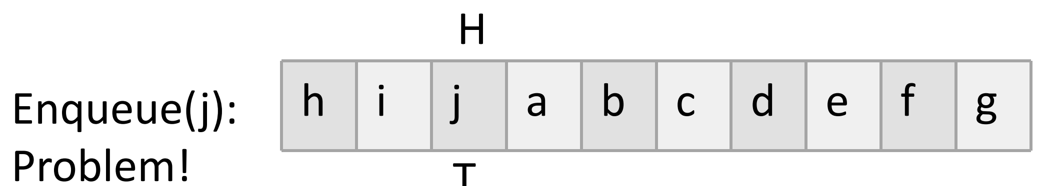
(H= (T Mod Max) + 1) : case full queue.

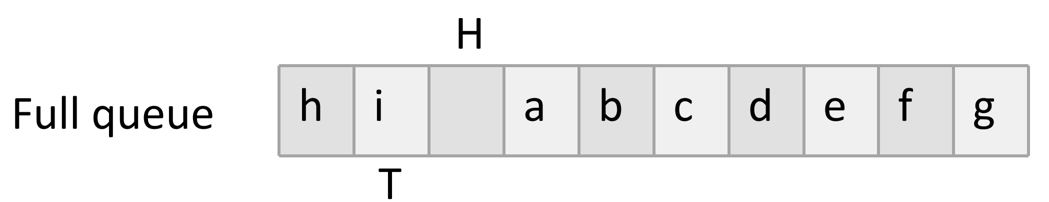












Queues

C Static Implementation using a circular array

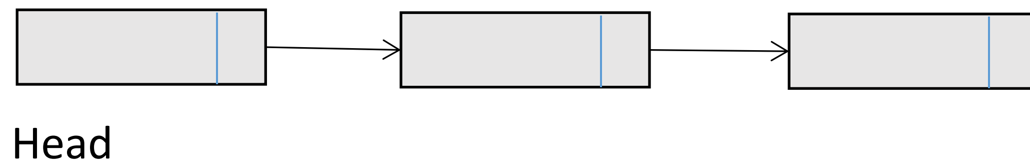
```
#define Max 100
 typedef int Bool;
 typedef int Anytype;
 struct Typequeue
   {
     Anytype Elements[Max];
     int Head, Tail;
   };
 struct Typequeue Q ;
 void Createqueue ( struct Typequeue *Q)
   {
    (*Q).Head = Max - 1;
    (*Q).Tail= Max - 1;
   }
 Bool Empty_queue ( struct Typequeue Q )
   {
     return ( Q.Head == Q.Tail);
   }
 Bool Full_queue ( struct Typequeue Q )
   {
     return ( Q.Head == Q.Tail% (Max-1) + 1 );
   }
```

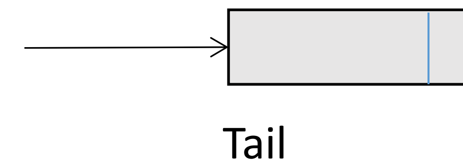
```
void Enqueue ( struct Typequeue *Q, Anytype Val )
 {
     if ( ! Full_queue(*Q) )
     {
       if ((*Q).Tail== Max-1 )
         (*Q).Tail= 0;
       else
         (*Q).Tail++ ;
       (*Q).Elements[(*Q).Tail] = Val;
     }
   else    printf(" %s", "Overflow");   }
void Dequeue ( struct Typequeue *Q, Anytype *Val )
 {
   if ( ! Empty_queue (*Q) )
     {
       if ( (*Q).Head == Max - 1)
         (*Q).Head = 0;
       else
         (*Q).Head ++;
       *Val = (*Q).Elements[(*Q).Head];
     }
   else      printf(" %s", "Undeflow"); }
```

```
int main(int argc, char *argv[])
  {
    system("PAUSE");
    return 0;
  }
```

Queues

Dynamic Implementation





As a linked list

The queue is defined by two pointers : Head et Tail

Enqueuing : at the beginning of the linked list
Dequeuing : at the end of the linked list

Queues

C Dynamic Implementation

```
#include <stdio.h>
#include <stdlib.h>

typedef int bool ;
#define True 1
#define False 0

/** -Implementation- **\: Queue OF INTEGERS**/
/** Queues **/
typedef int Typeelem_Qi ;
typedef  struct Queue_Qi * Pointer_Qi ;
typedef  struct Cell_Qi * Ptliste_Qi ;

struct Cell_Qi
  {
    Typeelem_Qi Val ;
    Ptliste_Qi Next  ;
  };
struct Queue_Qi
  {      Ptliste_Qi Head, Tail ;      };
```

```
  void Createqueue_Qi ( Pointer_Qi *A_queue   )
    {
      *A_queue = (struct Queue_Qi *) malloc( sizeof( struct
Queue_Qi)) ;
      (*A_queue)->Head = NULL ;
      (*A_queue)->Tail = NULL ;
    }
  bool Empty_queue_Qi (Pointer_Qi A_queue  )
    { return  A_queue->Head == NULL ;}

  void Enqueue_Qi ( Pointer_Qi A_queue , Typeelem_Qi Val   )
    {
      Ptliste_Qi Q;
      Q = (struct Cell_Qi *) malloc( sizeof( struct Cell_Qi))   ;
      Q->Val = Val ;
      Q->Next = NULL;
      if ( ! Empty_queue_Qi(A_queue) )
        A_queue->Tail->Next = Q ;
      else A_queue->Head = Q;
        A_queue->Tail = Q;
    }
```

Queues

C Dynamic Implementation

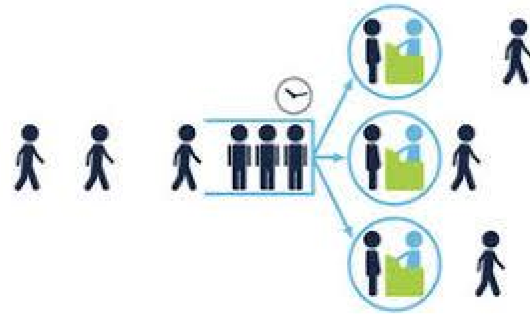
```
void Dequeue_Qi (Pointer_Qi A_queue, Typeelem_Qi *Val )
  {
    Ptliste_Qi Save;
    if (! Empty_queue_Qi(A_queue) )
      {
        Save = A_queue->Head;
        *Val = A_queue->Head->Val ;
        A_queue->Head =  A_queue->Head->Next;
        free(Save);
      }
    else printf ("%s \n", "Queue is empty");
  }
 /** Variables of main program **/
 Pointer_Qi Q=NULL;

 int main(int argc, char *argv[])
  {
    system("PAUSE");
    return 0;
  }
```

Queues

Application

Simulation of a Customer Flow through a Post Office



But : determine the minimum number of counters so that customers wait a maximum of x minutes before being served

Queues

Application (Single Counter Case)

Suppose that
- each minute, 0, 1, or 2 customers arrive at the post office

- the expected service time for a customer is one minute.

Find the number of unserved customers after n minutes of service.

Queues

Application (Single Counter Case)

```
CreateQueue (Q)
For i = 1 to n
   If Not Empty_Queue(Q)
      Dequeue a customer; Serve customer
   EndIf
   Generate a random number K between 0 and 2
   If K = 1: Enqueue a new customer
   Else
      If K = 2: Enqueue two new customers
      EndIf
   EndIf
EndFor
Display the number of unserved customers (size of Q)
```

______________________Minute 1
+ Customer 1; + Customer 2
______________________Minute 2
Customer 1 served; + Customer 3; + Customer 4
______________________Minute 3
Customer 2 served
______________________Minute 4
Customer 3 served; + Customer 5; + Customer 6
______________________Minute 5
Customer 4 served
______________________Minute 6
Customer 5 served; + Customer 7
______________________Minute 7
Customer 6 served; + Customer 8
______________________Minute 8
Customer 7 served; + Customer 9
______________________Minute 9
Customer 8 served; + Customer 10; + Customer 11
______________________Minute 10
Customer 9 served; + Customer 12
Number of unserved customers: 3 (Customers 10, 11, and 12)

Queues

Application (General algorithm)

- Nb_counters: the number of counters
- Nb_customers: the maximum number of customers per minute
- N : number of observation minutes
- Max_mn_servi : maximal wait (in mn)

A transaction takes 1 minute

- Find the number of unserved customers after N minutes of service
- Find the maximum wait time

Deduce the number of counters so that customers wait a maximum of x minutes before being served.

```
CreateQueue(Q)
Cl := 0; Max_wait_time := 0;
For I := 1 to N
   J := 1; // Parallel processing at all counters
   While Not Empty_Queue(Q) and (J <= Nb_counters)
      Dequeue a customer (Cl2, I2)
      If (I - I2) > Max_wait_time: Max_wait_time := I2 Endif
      J := J + 1
   EndWhile;
   Generate a random number K between 0 and Nb_customers
   For J := 1 to K
      Cl := Cl + 1
      Enqueue a new customer (Cl, I)
   EndFor
EndFor
Display the number of unserved customers (Size of Q)
Display the maximum wait time (Max_wait_time)
```

Queues

Application (General algorithm)

N	Nb_counters	Nb_customers	Number of unserved customers	Maximal wait
20	3	5	5	2
30	5	8	1	2
20	2	3	37	13
20	3	8	9	4
20	4	8	4	2
10	3	5	0	1

Queues

Observation time : 10
Number of counters : 3
Number of clients that arrive per minute : 5

Application (General algorithm) Trace for N=10, Nb_counters=3, and Nb_customers=5

______________________Minute 1
+ Customer 1 (1)
+ Customer 2 (1)
+ Customer 3 (1)

______________________Minute 2
Customer 1 (1) served after 1
Customer 2 (1) served after 1
Customer 3 (1) served after 1
+ Customer 4 (2)

______________________Minute 3
Customer 4 (2) served after 1
+ Customer 5 (3)
+ Customer 6 (3)
+ Customer 7 (3)

______________________Minute 4
Customer 5 (3) served after 1
Customer 6 (3) served after 1
Customer 7 (3) served after 1

______________________Minute 5
+ Customer 8 (5)
+ Customer 9 (5)

______________________Minute 6
Customer 8 (5) served after 1
Customer 9 (5) served after 1
+ Customer 10 (6)
+ Customer 11 (6)
+ Customer 12 (6)

______________________Minute 7
Customer 10 (6) served after 1
Customer 11 (6) served after 1
Customer 12 (6) served after 1
+ Customer 13 (7)

______________________Minute 8
Customer 13 (7) served after 1
+ Customer 14 (8)

______________________Minute 9
Customer 14 (8) served after 1
+ Customer 15 (9)
+ Customer 16 (9)

______________________Minute 10
Customer 15 (9) served after 1
Customer 16 (9) served after 1

Number of unserved customers: 0
Maximum wait time: 1