Les files d'attente

D.E ZEGOUR

École Supérieure d'Informatique

ESI

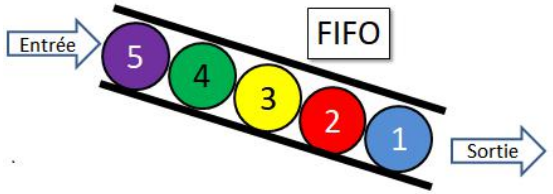
Les files d'attente

Définition, principe, domaine d'application

Collection d'éléments dans laquelle

- tout nouveau élément est inséré à la fin
- tout retrait d'élément ne peut être fait que du début.

Principe FIFO,
"First In, First Out"
Premier entré, Premier servi



Utilisée dans
- les systèmes d'exploitation
- les problèmes de simulation.

Aussi utilisée pour le parcours des arbres et pour résoudre tant d'autres problèmes.

Les files d'attente

Machine abstraite

CréerFile(F)	Créer une file F vide
Enfiler(F,Val)	Ajouter Val en queue de la file F.
Défiler(F,Val)	Retirer dans Val l'élément en tête de la file F
Filevide(F)	Tester si la file F est vide.
Filepleine(F)	Tester si la file F est pleine.

Les files d'attente

Implémentation

Statique : au moyen de tableaux (par flot - par décalage - Tableau circulaire)

Dynamique : au moyen des listes linéaires chaînées

Les files d'attente

Implémentation : Statique par flot

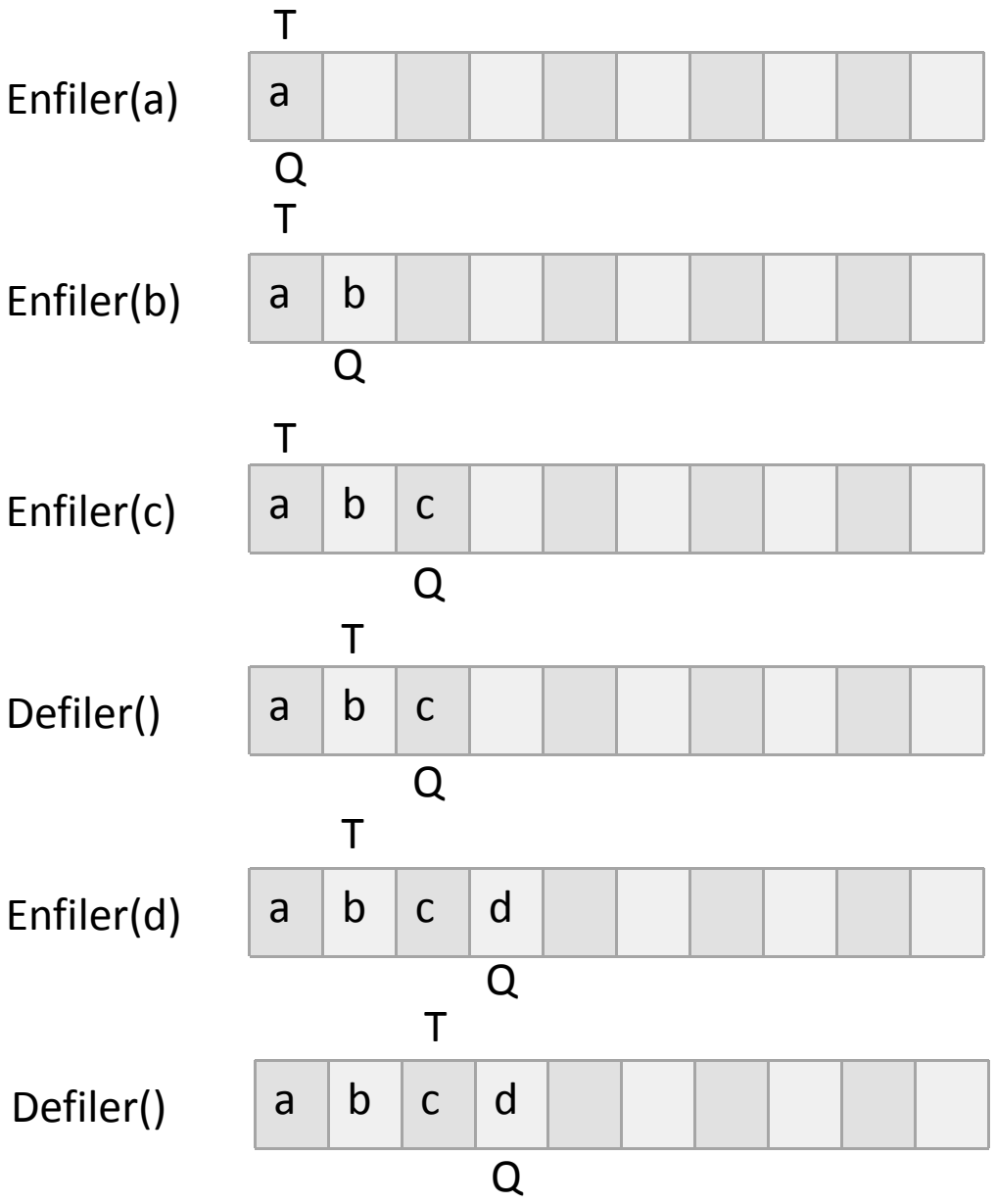
Au départ, tableau T[1..Max] vide

Nombre d'éléments = Q - T + 1
La file n'est pas vide si Q >= T
donc la file est vide si non (Q >= T),
C'est à dire Q < T

Initialisation : T = 1, Q = 0

La file avance en flot
Inconvénient : les éléments avant T
ne sont pas récupérables

La file est pleine si Q = Max



Les files d'attente

Implémentation : Statique par flot (C)

```
#define Max 100
typedef int Bool;
typedef int Typeqq;
struct Typefile
 {
   Typeqq Elements[Max];
    int Tete, Queue;
 }
```

```
void Creerfile ( struct Typefile *F )
  {
    (*F).Tete = 0;
    (*F).Queue = -1;
  }

 Bool Filevide ( struct Typefile F )
  {
    return ( F.Tete > F.Queue );
  }

 Bool Filepleine ( struct Typefile F )
  {
    return ( F.Queue == Max-1 );
  }
```

Les files d'attente

Implémentation : Statique par flot (C)

```
void Enfiler ( struct Typefile *F, Typeqq Val )
  {
    if ( ! Filepleine(*F) )
      {
        (*F).Queue++;
        (*F).Elements[(*F).Queue] = Val;
      }
    else
      printf(" File pleine");
   }
```

```
void Defiler ( struct Typefile *F, Typeqq *Val )
   {
     if ( ! Filevide( *F) )
       {
         *Val = (*F).Elements[(*F).Tete];
          (*F).Tete ++;
       }
     else
       printf(" File vide");
   }
```

Les files d'attente

Implémentation : Statique par décalage

Au départ, tableau T[1..Max] vide

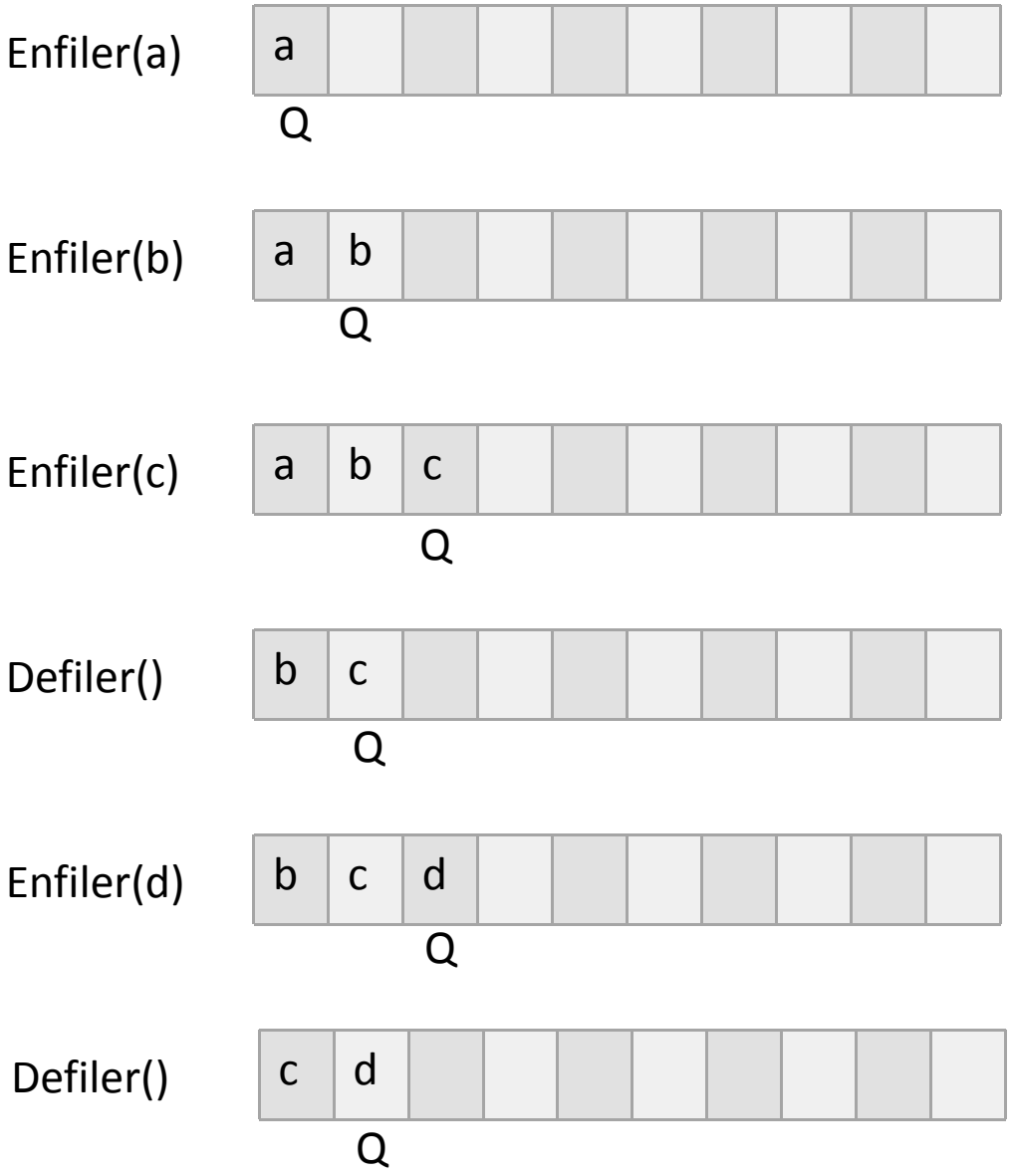
Nombre d'éléments = Q
La file n'est pas vide si Q >= 1
donc la file est vide si Q < 1

Initialisation : Q = 0

On a pas besoin de la tete

La file est pleine si Q = Max

Inconvénient : Pour chaque défilement, on fait un décalage



Les files d'attente

Implémentation : Statique par décalage (C)

```
#define Max 100
 typedef int Bool;
 typedef int Typeqq;
  struct Typefile
  {
    Typeqq Elements[Max];
     int Queue;
   }
  struct Typefile F
```

```
void Creerfile ( struct Typefile *F )
 {
   (*F).Queue = -1;
 }

Bool Filevide ( struct Typefile F )
 {
    return ( F.Queue == -1 );
 }

Bool Filepleine ( struct Typefile F )
 {
    return ( F.Queue == Max-1 );
 }
```

Les files d'attente

Implémentation : Statique par décalage (C)

```
void Enfiler ( struct Typefile *F, Typeqq Val )
 {
   if ( ! Filepleine(*F) )
    {
       (*F).Queue++;
       (*F).Elements[(*F).Queue] = Val;

    }
   else
    printf(" File pleine");
 }
```

```
void Defiler ( struct Typefile *F, Typeqq *Val )
  {
   int I;
   if ( ! Filevide(*F) )
    {
     *Val = (*F).Elements[0];
      for(I=0; I<=(*F).Queue-2; I++)
       (*F).Elements[I] =(*F).Elements[I + 1];
      (*F).Queue --;
    }
   else
    printf(" File vide");
  }
```

Les files d'attente

Implémentation : Statique par tableau circulaire

Revenons à la solution par flot (situation file pleine) et essayons d'utiliser le tableau de façon circulaire.

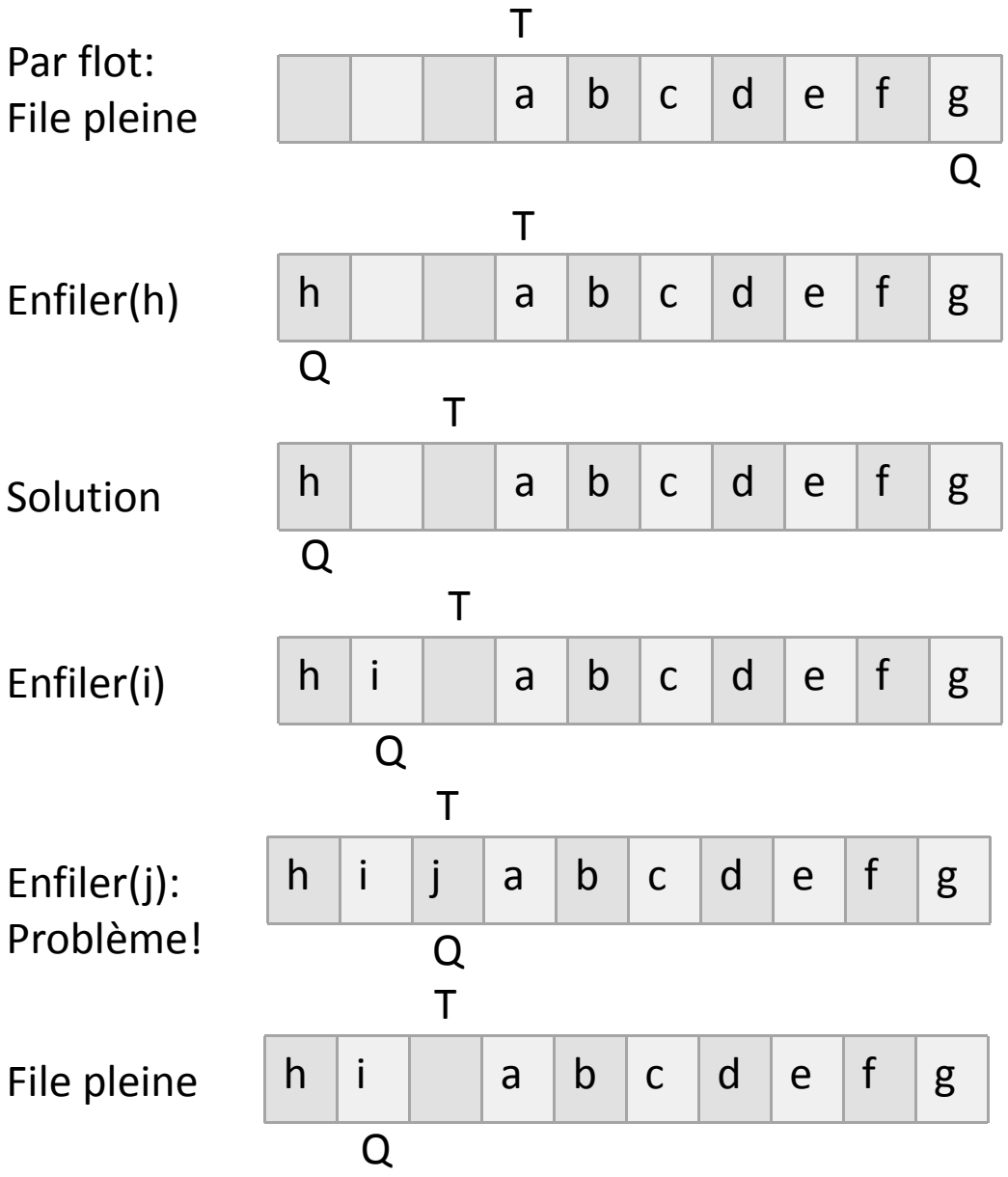
Comment initialiser la file ?

- Q < T impossible (contre exemple)
- Q > T impossible (contre exemple)
- T = Q impossible (cas ou il reste un élément)

Solution :

- T : pointe l'élément qui précède le premier.
- Q: pointe le dernier élément.
- (T= Q) : cas file vide.

(T= (Q Mod Max) + 1) : cas file pleine.



Les files d'attente

Implémentation : Statique par tableau circulaire (C)

```
#define Max 100
 typedef int Bool;
 typedef int Typeqq;

 struct Typefile
  {
    Typeqq Elements[Max];
    int Tete, Queue;
  }
 struct Typefile F
```

```
void Creerfile ( struct Typefile *F )
   {
    (*F).Tete = Max - 1;
    (*F).Queue = Max - 1;
   }
```

```
Bool Filevide ( struct Typefile F )
   {
     return ( F.Tete == F.Queue );
   }

  Bool Filepleine ( struct Typefile F )
   {
     return ( F.Tete == F.Queue % (Max-1) + 1 );
   }
```

Les files d'attente

Implémentation : Statique par tableau circulaire (C)

```
void Enfiler ( struct Typefile *F, Typeqq Val )
  {
     if ( ! Filepleine(*F) )
     {
       if ((*F).Queue == Max-1 )
         (*F).Queue = 0;
       else
         (*F).Queue ++ ;
       (*F).Elements[(*F).Queue] = Val;
     }
    else
     printf(" File pleine");
  }
```

```
void Defiler ( struct Typefile *F, Typeqq *Val )
 {
   if ( ! Filevide(*F) )
     {
       if ( (*F).Tete == Max - 1)
         (*F).Tete = 0;
       else
         (*F).Tete ++;
       *Val = (*F).Elements[(*F).Tete];
     }
   else
     printf(" File vide");
  }
```

Les files d'attente

Implémentation : Dynamique

Comme une liste linéaire chaînée

La file est définie par deux pointeurs : Tete et Queue

Enfilement : au debut de la liste
Défilement : en fin de la liste

Les files d'attente

Implémentation : Dynamique (C)

```
/** Implémentation **\: FILE DE ENTIERS**/

typedef int Typeelem_Fi ;
typedef  struct Filedattente_Fi * Pointeur_Fi ;
typedef  struct Maillon_Fi * Ptliste_Fi ;

struct Maillon_Fi
 {
   Typeelem_Fi Val ;
   Ptliste_Fi Suiv  ;
 }

struct Filedattente_Fi
 {
   Ptliste_Fi Tete, Queue ;
 }
```

```
void Creerfile_Fi ( Pointeur_Fi *Fil   )
  {
    *Fil = (struct Filedattente_Fi *)
malloc( sizeof( struct Filedattente_Fi)) ;
    (*Fil)->Tete = NULL ;
    (*Fil)->Queue = NULL ;
  }
```

```
bool Filevide_Fi (Pointeur_Fi Fil  )
 { return  Fil->Tete == NULL ;}
```

Les files d'attente

Implémentation : Dynamique (C)

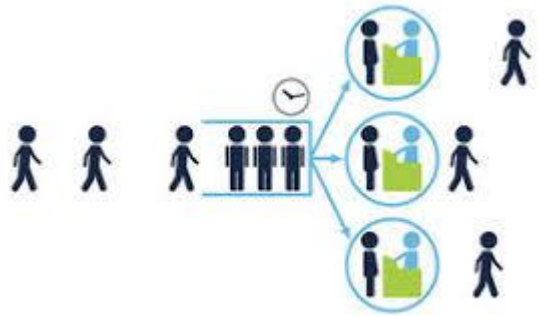
```
void Enfiler_Fi ( Pointeur_Fi Fil , Typeelem_Fi Val   )
  {
    Ptliste_Fi Q;

    Q = (struct Maillon_Fi *) malloc( sizeof( struct Maillon_Fi))   ;
    Q->Val = Val ;
    Q->Suiv = NULL;
    if ( ! Filevide_Fi(Fil) )
      Fil->Queue->Suiv = Q ;
    else Fil->Tete = Q;
    Fil->Queue = Q;
  }
```

```
void Defiler_Fi (Pointeur_Fi Fil, Typeelem_Fi *Val )
 {
   if (! Filevide_Fi(Fil) )
     {
       *Val = Fil->Tete->Val ;
       Fil->Tete =  Fil->Tete->Suiv;
     }
   else printf ("%s \n", "File d'attente vide");
 }
```

Les files d'attente

Application

Simulation du flux de clients via une ligne de paiement



Objectif : réduire le nombre de guichets de sorte que les clients attendent un maximum de x minutes avant d'être servis.

Les files d'attente

Application (Cas d'un seul guichet)

Suppose que:
Chaque minute, 0, 1 ou 2 clients devront être servis sur une ligne de paiement.

Le temps de service prévu pour un client est d' 1 minute.

Il y a une ligne de paiement disponible.

Trouvez le nombre de clients non servis après n minutes de service.

Les files d'attente

Application (Cas d'un seul guichet)

```
Creerfile (F)
Pour i = 1,  n
   Si Non filevide(F)
     Defiler un client; Servir client
   Fsi
   Générer un nombre aléatoire K entre 0 et 2
   Si K=1 : Enfiler un nouveau client
   Sinon
      Si k=2 : Enfiler deux nouveaux clients Fsi
   Fsi
Fpour
Afficher le nombre de clients non servis
(cardinalité de F)
```

_____Minute 1
+ Client 1 ; + Client 2
_____Minute 2
Client 1 servi ; + Client 3 ; + Client 4
_____Minute 3
Client 2 servi
_____Minute 4
Client 3 servi ; + Client 5 ; + Client 6
_____Minute 5
Client 4 servi
_____Minute 6
Client 5 servi ; + Client 7
_____Minute 7
Client 6 servi ; + Client 8
_____Minute 8
Client 7 servi ; + Client 9
_____Minute 9
Client 8 servi ; + Client 10 ; + Client 11
_____Minute 10
Client 9 servi ; + Client 12
Nombre de clients non servis : 3 (10, 11 et 12)

Les files d'attente

Application (Algorithme général)

Nb_guichets : nombre de guichets
Nb_clients : Nombre maximal de clients par minutes (générés aléatoirement)
N : nombre de minutes d'observation
Max_mn_servi : Attente maximale (en mn)
Un transaction dure 1 minute

- Trouver le nombre de clients non servis après N minutes de service
- Trouver l'attente maximal

Déduire le nombre de guichets de sorte que les clients attendent un maximum de x minutes avant d'être servis.

```
Creerfile ( F ) ;
Cl := 0 ; Max_mn_servi := 0;
Pour I := 1 , N
   J := 1 ; // Traitement parallèle sur tous les guichets
   Tq Non Filevide ( F ) ET ( J <= Nb_guichets )
      Defiler un client (Cl2, I2)
      SI (I -I2) >  Max_mn_servi  :  Max_mn_servi := I2 Fsi
      J := J + 1 ;
   Ftq ;
    Générer un nombre aléatoire K entre 0 et Nb_clients
    Pour J:=1, K
        Cl := Cl + 1;
        Enfiler un nouveau client (Cl, I)
   Fpour
Fpour
Afficher le nombre de clients non servis (cardinalité de F)
Afficher l'attente maximale (Max_mn_servi)
```

Les files d'attente

Application (Algorithme général : exemple)

N	G	CM	CNS	AM
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

- Temps d'observation (N)
- Nombre de guichets (G)
- Nombre de clients qui arrivent par minute (CM)

- Nombre de clients non servis (CNS)
- Attente maximale (AM)

Les files d'attente

Temps d'observation : 10
Nombre de guichets : 3
Nombre de clients qui arrivent par minute : 5

Application (Algorithme général : exemple)

_____Minute 1
+ Client 1 1
+ Client 2 1
+ Client 3 1
_____Minute 2
Client 1 1 servi après 1
Client 2 1 servi après 1
Client 3 1 servi après 1
+ Client 4 2
_____Minute 3
Client 4 2 servi après 1
+ Client 5 3
+ Client 6 3
+ Client 7 3
_____Minute 4
Client 5 3 servi après 1
Client 6 3 servi après 1
Client 7 3 servi après 1
_____Minute 5
+ Client 8 5
+ Client 9 5
_____Minute 6
Client 8 5 servi après 1
Client 9 5 servi après 1
+ Client 10 6
+ Client 11 6
+ Client 12 6
_____Minute 7
Client 10 6 servi après 1
Client 11 6 servi après 1
Client 12 6 servi après 1
+ Client 13 7
_____Minute 8
Client 13 7 servi après 1
+ Client 14 8
_____Minute 9
Client 14 8 servi après 1
+ Client 15 9
+ Client 16 9
_____Minute 10
Client 15 9 servi après 1
Client 16 9 servi après 1

Nombre de clients non servis : 0
Maximum d'attente : 1