

Programmation avancée

Listes chaînées : variantes

Walter Rudametkin

Walter.Rudametkin@polytech-lille.fr
<https://rudametw.github.io/teaching/>

Bureau F011
 Polytech'Lille

29 février 2016

1/24

Listes chaînées : variantes

Maintenir la longueur

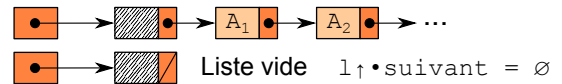
- Pour accès par position : $k < \text{longueur}(L)$

Maintenir un pointeur sur la dernière cellule

- Accès et modifications courantes en queue

Introduction d'une tête fictive

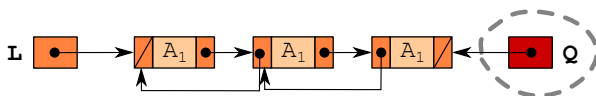
- Pour simplifier ajout / suppression en tête



2/24

Listes symétriques (ou doublement chaînées)

- Facilitent parcours symétriques (dans les 2 sens)
- Ajout/retrait sans nécessiter le prec



Action $\text{supp}(P)$

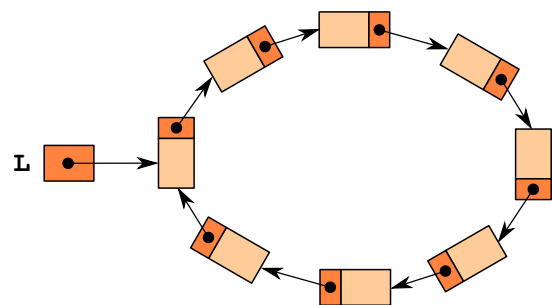
$D/R : P : \text{Liste_contiguë}$
 $P \uparrow \cdot \text{prec} \uparrow \cdot \text{suiv} \leftarrow P \uparrow \cdot \text{suiv}$
 $P \uparrow \cdot \text{suiv} \uparrow \cdot \text{prec} \leftarrow P \uparrow \cdot \text{prec}$
 libérer (P)

Faction

3/24

Listes circulaires (sans tête)

- Permet accès à tous les éléments à partir de n'importe quelle cellule



4/24

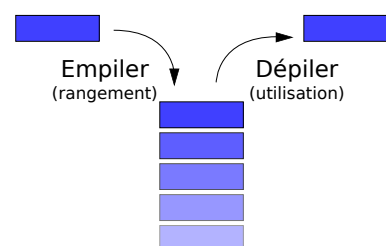
Listes à fonctionnalités particulières

- Limitation de l'accès aux éléments en fonction d'utilisations particulières (accès privilégié)
- Piles (Last In First Out—LIFO)
- Files d'attente (First In First Out—FIFO)

5/24

Les piles

Accès réduit : uniquement en tête



Ordre chronologique inverse

- Dernière information rangée
- Première utilisée

**Last In First Out
LIFO**

6/24

Les piles: exemples

- ▶ Pile de cartes
- ▶ Recherche d'un chemin sur une carte
 - ▶ Aller de i en j : `empiler(i)`
 - ▶ Revenir de j en i : `dépiler(i)`
 - ▶ Quand la destination est rencontrée, le chemin recherché est dans la pile
- ▶ Pile d'exécution de sous programmes
 - ▶ Gérée automatiquement par le langage pour sauvegarder les contextes d'exécution (restaurés dans l'ordre inverse des appels)
 - ▶ Permet la récursivité

7/24

Les piles: définition

- ▶ P : de type Pile [de $\langle T \rangle$]

Opérations

- ▶ `empiler(P,V)` : action qui ajoute un élément en sommet de pile
- ▶ `dépiler(P,V)` : action qui retire l'élément au sommet de pile et le range dans V
- ▶ `sommet(P)` : fonction qui retourne la valeur au sommet de pile sans la dépiler
- ▶ `pile_vide(P)` : fonction qui teste si la pile est vide

8/24

Les piles

- ▶ `init_pile(P)` : action qui initialise la pile P à vide avant toute utilisation
- ▶ `pile_pleine(P)` : fonction qui teste si la pile est pleine (quand elle est de taille bornée)

Remarques

- ▶ `pile_vide(P) \Rightarrow sommet(P) et dépiler(P,V) invalides !`
- ▶ `pile_pleine(P) \Rightarrow empiler(P) invalide !`

9/24

Les piles : choix d'implantation

type abstrait \rightarrow implantation

- ▶ List dont on restreint l'accès
 - ▶ chaînée
 - ▶ contiguë

10/24

Les piles : implantation par liste chaînée



- ▶ `type Pile = Liste chaînée`
- ▶ `dépiler \rightarrow supp_tête`
- ▶ `empiler \rightarrow ajout_tête`

11/24

Les piles : implantation par liste chaînée

Dépiler

Action `dépiler(P, V)`
 $\underline{D/R}$: P : Pile
 \underline{R} : V : $\langle T \rangle$
 $V \leftarrow P \uparrow \bullet \text{valeur}$
`supp_tête(P)`
Faction

Sommet

Fonction `sommet(P) : $\langle T \rangle$`
 \underline{D} : P : Pile
 Retourner ($P \text{ valeur}$)
Ffonction

Empiler

Action `empiler(P, V)`
 $\underline{D/R}$: P : Pile
 \underline{D} : V : $\langle T \rangle$
`ajout_tête(P, V)`
Faction

12/24

Les piles : implantation par liste contiguë

- ▶ type Pile = Liste_contiguë

Accès au dernier

sommet(P) : P.espace[P.dernier]

empiler(P, V) : P.dernier \leftarrow P.dernier + 1
P.espace[P.dernier] \leftarrow V

dépiler(P, V) : V \leftarrow P.espace[P.dernier]
P.dernier \leftarrow P.dernier - 1

13/24

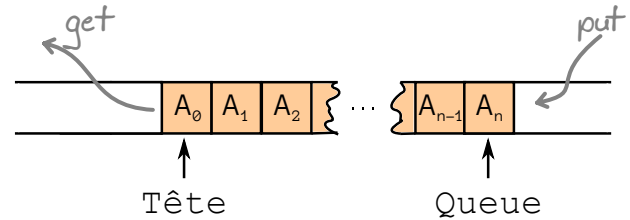
Les files d'attente

- ▶ Liste où les éléments sont utilisés dans l'ordre chronologique de leur rangement

- ▶ 1ère information rangée

- ▶ 1ère information traitée

**First In First Out
FIFO**



14/24

Les files d'attente : exemples

- ▶ Stock de données périssables
- ▶ Caisses de supermarché
- ▶ File d'attente de travaux d'impression sur imprimante de façon générale, file d'attente d'utilisation d'une ressource partagée

Définition du type

- ▶ F : file d'attente de <T>
- ▶ F : FIFO de <T>

15/24

Les files d'attente : primitives

- ▶ init_fifo(F) : action qui initialise la FIFO F à vide (avant toute utilisation)
- ▶ fifo_vide(F) : booléen : fonction qui teste si F est vide
- ▶ fifo_pleine(F) : booléen : fonction qui teste si F est pleine {si la file est de taille bornée}
- ▶ first(F) : <T> : fonction qui rend la valeur de l'élément de F sans l'extraire
- ▶ put(F, X) : action qui range X en queue de file
- ▶ get(F, X) : action qui extrait de la file l'élément de tête et le range dans X

16/24

Les files d'attente : implantation chaînée

- ▶ Fortement dynamique
- ▶ Sans estimation aisée de la taille max

get et first

- ▶ Accès en tête aisé au travers du pointeur de tête

put et last

- ▶ Parcours séquentiel jusqu'au dernier : coûteux !!
- ▶ Besoin d'un accès privilégié en queue !
- ▶ Solution \Rightarrow Maintenir un 2ème pointeur de queue

17/24

Les files d'attente : définition du type FIFO chaînée

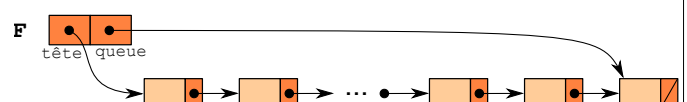
type Ptcellule = pointeur de Cellule

type Cellule = structure
valeur : <T>
suivant : Ptcellule

fin

type Fifo = structure
tête, queue : Ptcellule

fin



18/24

Les files d'attente : implantation chaînée

Init

```

Action init_fifo(F)
  D/R : F : Fifo de <T>
  F.tête ← NULL
  F.queue ← NULL
Faction

```

First

```

Fonction first(F) : <T>
  D : F : Fifo de <T>
  retourner(F.tête↑.valeur)
FFonction

```

Get

```

Action get(F, X)
  D/R : F : Fifo de <T>
  R : X : <T>
  X ← F.tête↑.valeur
  supp_tête(F.tête)
Faction

```

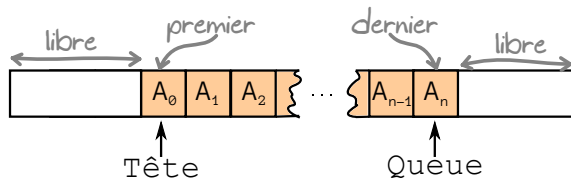
19/24

Les files d'attente : implantation contiguë

- Taille peu variable ou estimation de max est aisée
- put et last :
 - Accès en queue
 - Aisé au travers de l'indice dernier
- first : accès en tête
- get :
 - Compactage systématique : *cher*
 - Maintenir un indice premier et gérer un espace libre devant ?
 - Solution : boucler sur l'espace

20/24

Les files d'attente : implantation contiguë



Définition

```

type Fifo = structure
  espace : vecteur [0..MAX-1] de <T>
  premier, dernier : -1..MAX-1 {-1 si file vide}
fin

```

21/24

Les files d'attente : quelques primitives

Init

```

Action init_fifo(F)
  D/R : F : Fifo de <T>
  F.dernier ← -1
  F.premier ← -1
Faction

```

Vide

```

Fonction fifo_vide(F) : booléen
  D : F : Fifo de <T>
  retourner(F.premier = -1)
FFonction

```

Pleine

```

Fonction fifo_pleine(F) : booléen
  D : F : Fifo de <T>
  retourner(
    F.premier = (F.dernier+1)%MAX
  )
FFonction

```

First

```

Fonction first(F) : <T>
  D : F : Fifo de <T>
  retourner(F.espace[F.premier])
FFonction

```

22/24

Les files d'attente : implantation contiguë put

```

Action put(F, X)
  D/R : F : Fifo de <T>
  D : X : <T>

  {valide si non fifo_pleine(F)}

  Si F.dernier = -1 Alors
    F.premier = 0
  Fsi

  F.dernier ← (F.dernier+1) % MAX
  F.espace[F.dernier] ← X
Faction

```

23/24

Les files d'attente : implantation contiguë get

```

Action get(F, X)
  D/R : F : Fifo de <T>
  R : X : <T>

  {valide si non fifo_vide(F)}

  X ← F.espace[F.premier]

  Si F.premier = F.dernier Alors
    F.premier ← F.dernier ← -1
  Sinon
    F.premier ← (F.premier+1) % MAX
  Fsi
Faction

```

24/24