Piles et Files

Ivan Noyer

Lycée Thiers

1 Piles
- Généralités

2 Files

Crédits

- Wikipédia : en français et (plus complet mais en anglais) ici

Crédits

- Wikipédia : en français et (plus complet mais en anglais) ici
- La documentation sur le module Stack de OCaml

Crédits

- Wikipédia : en français et (plus complet mais en anglais) ici
- La documentation sur le module Stack de OCaml
- La documentation sur le module Queue de OCaml

1. Piles
   - Généralités

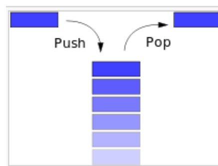
2. Files

1. Piles
   - Généralités
2. Files

Pile

- Une *pile* (en anglais stack) est une *structure de données* fondée sur le principe « dernier arrivé, premier sorti » (ou LIFO pour Last In, First Out), ce qui veut dire que les derniers éléments ajoutés à la pile seront les premiers à être récupérés.

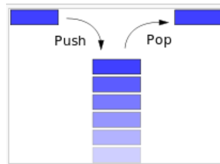
FIGURE – Empiler et dépiler



Pile

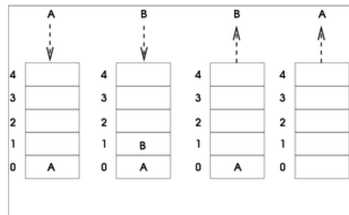
- Une *pile* (en anglais stack) est une *structure de données* fondée sur le principe ≪ dernier arrivé, premier sorti ≫ (ou LIFO pour Last In, First Out), ce qui veut dire que les derniers éléments ajoutés à la pile seront les premiers à être récupérés.
- Le fonctionnement est celui d'une pile d'assiettes : on ajoute des assiettes sur la pile, et on les récupère dans l'ordre inverse, en commençant par la dernière ajoutée.

FIGURE – Empiler et dépiler



Exemple

Figure – Empilement de A puis B, dépilement de B puis A (LIFO)



Historique

- Turing 1946. Description théorique d'appel et de retour de sous-routines.

Historique

- Turing 1946. Description théorique d'appel et de retour de sous-routines.
- Klaus Samelson et Friedrich L. Bauer de l'université Technique de Münich généralisent l'idée en 1955 en présentant une modélisation du fonctionnement des ordinateurs à base de piles.

Historique

- Turing 1946. Description théorique d'appel et de retour de sous-routines.
- Klaus Samelson et Friedrich L. Bauer de l'université Technique de Münich généralisent l'idée en 1955 en présentant une modélisation du fonctionnement des ordinateurs à base de piles.
- Même concept, indépendamment développé par l'australien Charles Leonard Hamblin en 1957.

Les primitives indispensables

- « Empiler » : ajoute un élément sur la pile. Terme anglais correspondant : « Push ».

Les primitives indispensables

- « Empiler » : ajoute un élément sur la pile. Terme anglais correspondant : « Push ».
- « Dépiler » : enlève l'élément au sommet de la pile et le renvoie. Terme anglais correspondant : « Pop ».

Les primitives indispensables

- « Empiler » : ajoute un élément sur la pile. Terme anglais correspondant : « Push ».
- « Dépiler » : enlève l'élément au sommet de la pile et le renvoie. Terme anglais correspondant : « Pop ».
- « La pile est-elle vide ? » : renvoie vrai si la pile est vide, faux sinon. « isempty »

Autres primitives

Elles peuvent être obtenues à partir des 3 premières :

- « Nombre d'éléments de la pile » : renvoie le nombre d'éléments dans la pile. Complexité : Selon les implémentations, peut être fait soit en temps constant soit en temps linéaire.

Autres primitives

Elles peuvent être obtenues à partir des 3 premières :

- « Nombre d'éléments de la pile » : renvoie le nombre d'éléments dans la pile. Complexité : Selon les implémentations, peut être fait soit en temps constant soit en temps linéaire.
- « Quel est l'élément de tête ? » : renvoie l'élément de tête sans le désempiler. Terme anglais correspondant : « Peek ».

Autres primitives

Elles peuvent être obtenues à partir des 3 premières :

- « Nombre d'éléments de la pile » : renvoie le nombre d'éléments dans la pile. Complexité : Selon les implémentations, peut être fait soit en temps constant soit en temps linéaire.
- « Quel est l'élément de tête ? » : renvoie l'élément de tête sans le désempiler. Terme anglais correspondant : « Peek ».
- « Vider la pile » : dépiler tous les éléments. Complexité : Selon l'implémentation, cela peut être fait en temps constant ou linéaire. Terme anglais correspondant : « Clear ».

Autres primitives

Elles peuvent être obtenues à partir des 3 premières :

- « Nombre d'éléments de la pile » : renvoie le nombre d'éléments dans la pile. Complexité : Selon les implémentations, peut être fait soit en temps constant soit en temps linéaire.
- « Quel est l'élément de tête ? » : renvoie l'élément de tête sans le désempiler. Terme anglais correspondant : « Peek ».
- « Vider la pile » : dépiler tous les éléments. Complexité : Selon l'implémentation, cela peut être fait en temps constant ou linéaire. Terme anglais correspondant : « Clear ».
- « Dupliquer l'élément de tête » et « Échanger les deux premiers éléments » : existe sur les calculatrices fonctionnant en notation polonaise inverse (style HP). Termes anglais correspondants : « Dup » et « Swap » respectivement.

Un peu de maths

En matière de structures abstraites, on peut considérer qu'une pile est un *monoïde libre*, c'est-à-dire un ensemble muni d'une loi de composition interne (la concaténation) associative et possédant un élément neutre (la pile vide).

Applications

- La plupart des microprocesseurs gèrent nativement une pile. Elle correspond alors à une zone de la mémoire, et le processeur retient l'adresse du dernier élément.

Applications

- La plupart des microprocesseurs gèrent nativement une pile. Elle correspond alors à une zone de la mémoire, et le processeur retient l'adresse du dernier élément.
- Dans un navigateur web, une pile sert à mémoriser les pages Web visitées. L'adresse de chaque nouvelle page visitée est empilée et l'utilisateur désempile l'adresse de la page précédente en cliquant le bouton ≪ Afficher la page précédente ≫.

Applications

- La plupart des microprocesseurs gèrent nativement une pile. Elle correspond alors à une zone de la mémoire, et le processeur retient l'adresse du dernier élément.
- Dans un navigateur web, une pile sert à mémoriser les pages Web visitées. L'adresse de chaque nouvelle page visitée est empilée et l'utilisateur désempile l'adresse de la page précédente en cliquant le bouton « Afficher la page précédente ».
- L'évaluation des expressions mathématiques en notation post-fixée (ou polonaise inverse) utilise une pile.

Applications

- La fonction « Annuler la frappe » (en anglais Undo) d'un traitement de texte mémorise les modifications apportées au texte dans une pile.

Applications

- La fonction « Annuler la frappe » (en anglais Undo) d'un traitement de texte mémorise les modifications apportées au texte dans une pile.
- Un algorithme de recherche *en profondeur d'abord* utilise une pile pour mémoriser les nœuds visités.

Applications

- La fonction « Annuler la frappe » (en anglais Undo) d'un traitement de texte mémorise les modifications apportées au texte dans une pile.
- Un algorithme de recherche *en profondeur d'abord* utilise une pile pour mémoriser les nœuds visités.
- Inversion d'un tableau ou d'une chaîne de caractères.

Applications

- La fonction « Annuler la frappe » (en anglais Undo) d'un traitement de texte mémorise les modifications apportées au texte dans une pile.
- Un algorithme de recherche *en profondeur d'abord* utilise une pile pour mémoriser les nœuds visités.
- Inversion d'un tableau ou d'une chaîne de caractères.
- Les algorithmes récursifs admis par certains langages (`LISP`, `C`, `Python`,etc...) utilisent implicitement une pile d'appel.

Listes OCaml et piles

- En OCaml, les listes ont un comportement de piles (fonctionnelles). Il n'y a donc pas besoin d'importer de module particulier pour bénéficier d'une structure de pile.

Listes OCaml et piles

- En OCaml, les listes ont un comportement de piles (fonctionnelles). Il n'y a donc pas besoin d'importer de module particulier pour bénéficier d'une structure de pile.
- Les listes OCaml ne sont pas des structures impératives : il n'y a pas d'effets de bord.
  Pour une structure de pile impérative préférer le module `Stack`.

En OCaml

```
open Stack;;(*charger le module Stack (pile)*)
let s = create ();;(*création d'une pile vide*)
push 3 s; push 6 s; push 7 s;;
(*-> ajouter 3 nombres dans la pile*)
let s' = copy(s);;(*faire une copie de la pile*)
for i = 0 to 2 do
  (*notre première boucle for CAML*)
  (*pos s : retire le sommet*)
  Printf.printf "%d; " (pop s);
done ;;
try
  (*On sait que le code suivant risque
  de générer une exception...
  ...Mais on 'essaye' quand même (try) !*)
  print_int (pop s);
  (*on risque depiler une pile vide!!*)
  with Empty ->
    print_string "pile vide : t ouf???\n";;
```

Rendu (jusqu'aux empilements)

```
1  # open Stack;;
2  # let s = create ();;
3  val s : '_a Stack.t = <abstr>
4  # push 3 s; push 6 s; push 7 s;;
5  - : unit = ()
6  # let s' = copy(s);;
7  val s' : int Stack.t = <abstr>
8  # for i = 0 to 2 do
9    Printf.printf "%d; " (pop s);
10 done;;
11     7; 6; 3; - : unit = ()
12 # push 3 s; push 6 s; push 7 s;;
13 - : unit = ()
```

Rendu (fin)

```
1  # let s' = copy(s);;
2  val s' : int Stack.t = <abstr>
3  # for i = 0 to 2 do
4     (*notre première boucle for CAML*)
5     Printf.printf "%d; " (pop s);
6  done;;
7          7; 6; 3; - : unit = ()
8  # try
9     (*On sait que le code suivant risque de générer une
          exception*)
10    (*mais on 'essaye' quand même (try)*)
11    print_int (pop s);(*on risque dedépiler une pile
          vide*)
12 with Empty -> print_string "vous avez voulu dépiler
       une pile vide\n";;
13           vous avez voulu dépiler une pile vide
14 - : unit = ()
```

Des piles en C
À partir de tableaux redimensionnables

Dans le fichier `vector.c` (cf. cours listes), ajoutons :

```
1  // renvoie le sommet SANS dépiler
2  int vector_top(vector *v) {
3    // renvoie l'élément en haut de pile SANS dépiler
4    int n = vector_size(v) - 1;
5    assert(0 <= n);
6    return vector_get(v, n);
7  }
8
9  // Dépile le sommet et le renvoie
10 // décrémente v->size + redimensionnement possible
11 int vector_pop(vector *v) {
12   int n = vector_size(v) - 1;
13   assert(0 <= n);
14   int r = vector_get(v, n);
15   vector_resize(v, n);//
16   return r;
17 }
```

Des piles en C

À partir de listes chaînées

Dans le fichier `tp1c.c` (cf TP sur les listes chaînées) ajoutons :

```
Liste *create(){// crée une liste vide
  Liste *liste = malloc(sizeof(Liste));
  liste->first = NULL;
  return liste;
}

void push(Liste *liste, int x){// empiler
  ajouterDebut(liste,x);
}

int pop(Liste *liste){// dépiler
  return supprimer(liste,0);
}

```

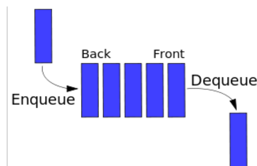
1 Piles
- Généralités

2 Files

Définition

- En informatique, une *file* (*queue* en anglais) est une structure de données basée sur le principe du Premier entré, premier sorti, en anglais FIFO (First In, First Out), ce qui veut dire que les premiers éléments ajoutés à la file seront les premiers à être récupérés.

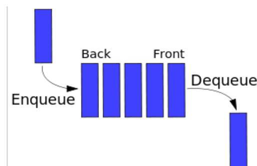
FIGURE – Une pile (FIFO)



Définition

- En informatique, une *file* (*queue* en anglais) est une structure de données basée sur le principe du Premier entré, premier sorti, en anglais FIFO (First In, First Out), ce qui veut dire que les premiers éléments ajoutés à la file seront les premiers à être récupérés.
- Le fonctionnement ressemble à une file d'attente à la poste : les premières personnes à arriver sont les premières personnes à sortir de la file.

FIGURE – Une pile (FIFO)



Applications

- Usage : Mémoriser temporairement des transactions qui doivent attendre pour être traitées *dans l'ordre d'arrivée*.

Applications

- Usage : Mémoriser temporairement des transactions qui doivent attendre pour être traitées *dans l'ordre d'arrivée*.
- Les serveurs d'impression, qui doivent traiter les requêtes dans l'ordre dans lequel elles arrivent, et les insèrent dans une file d'attente (ou une queue).

Applications

- Usage : Mémoriser temporairement des transactions qui doivent attendre pour être traitées *dans l'ordre d'arrivée*.
- Les serveurs d'impression, qui doivent traiter les requêtes dans l'ordre dans lequel elles arrivent, et les insèrent dans une file d'attente (ou une queue).
- Certains moteurs multitâches, dans un système d'exploitation, qui doivent accorder du temps-machine à chaque tâche, sans en privilégier aucune.

Applications

- Usage : Mémoriser temporairement des transactions qui doivent attendre pour être traitées *dans l'ordre d'arrivée*.
- Les serveurs d'impression, qui doivent traiter les requêtes dans l'ordre dans lequel elles arrivent, et les insèrent dans une file d'attente (ou une queue).
- Certains moteurs multitâches, dans un système d'exploitation, qui doivent accorder du temps-machine à chaque tâche, sans en privilégier aucune.
- Un algorithme de *parcours en largeur d'abord* dans un graphe utilise une file pour mémoriser les nœuds visités.

Applications

- Usage : Mémoriser temporairement des transactions qui doivent attendre pour être traitées *dans l'ordre d'arrivée*.
- Les serveurs d'impression, qui doivent traiter les requêtes dans l'ordre dans lequel elles arrivent, et les insèrent dans une file d'attente (ou une queue).
- Certains moteurs multitâches, dans un système d'exploitation, qui doivent accorder du temps-machine à chaque tâche, sans en privilégier aucune.
- Un algorithme de *parcours en largeur d'abord* dans un graphe utilise une file pour mémoriser les nœuds visités.
- Pour créer toutes sortes de mémoires tampons (en anglais *buffers*).

Primitives

Voici les primitives communément utilisées pour manipuler des files. Il n'existe pas de normalisation pour les primitives de manipulation de file. Leurs noms sont donc indiqués de manière informelle.

- « Mettre dans la file » : ajoute un élément dans la file. Terme anglais correspondant : « Enqueue ».

Primitives

Voici les primitives communément utilisées pour manipuler des files. Il n'existe pas de normalisation pour les primitives de manipulation de file. Leurs noms sont donc indiqués de manière informelle.

- « Mettre dans la file » : ajoute un élément dans la file. Terme anglais correspondant : « Enqueue ».
- « Défiler » : renvoie le prochain élément de la file, et le retire de la file. Terme anglais correspondant : « Dequeue ».

Primitives

Voici les primitives communément utilisées pour manipuler des files. Il n'existe pas de normalisation pour les primitives de manipulation de file. Leurs noms sont donc indiqués de manière informelle.

- « Mettre dans la file » : ajoute un élément dans la file. Terme anglais correspondant : « Enqueue ».
- « Défiler » : renvoie le prochain élément de la file, et le retire de la file. Terme anglais correspondant : « Dequeue ».
- « La file est-elle vide ? » : renvoie « vrai » si la file est vide, « faux » sinon.

Primitives

Voici les primitives communément utilisées pour manipuler des files. Il n'existe pas de normalisation pour les primitives de manipulation de file. Leurs noms sont donc indiqués de manière informelle.

- ≪ Mettre dans la file ≫ : ajoute un élément dans la file. Terme anglais correspondant : ≪ Enqueue ≫.
- ≪ Défiler ≫ : renvoie le prochain élément de la file, et le retire de la file. Terme anglais correspondant : ≪ Dequeue ≫.
- ≪ La file est-elle vide ? ≫ : renvoie ≪ vrai ≫ si la file est vide, ≪ faux ≫ sinon.
- ≪ Nombre d'éléments dans la file ≫ : renvoie le nombre d'éléments dans la file.

Implémentation en OCaml

- On peut utiliser une liste comme une file FIFO.

Implémentation en OCaml

- On peut utiliser une liste comme une file FIFO.
- Pas efficace : alors que les ajouts et suppressions en fin de liste sont rapides, les opérations d'insertions ou de retraits en début de liste sont lentes (car tous les autres éléments doivent être décalés d'une position).

Implémentation en OCaml

- On peut utiliser une liste comme une file FIFO.
- Pas efficace : alors que les ajouts et suppressions en fin de liste sont rapides, les opérations d'insertions ou de retraits en début de liste sont lentes (car tous les autres éléments doivent être décalés d'une position).
- Pour implémenter une file, utiliser le module OCaml `Queue` qui a été conçue pour fournir des opérations d'ajouts et de retraits rapides dans la file.

En OCaml

```
1  open Queue;;(*importer le module*)
2  let q = create ();;(*file vide*)
3  let q' = copy(q);;(*copy de q*)
4  push 3 q; push 6 q; push 7 q;;(*ajout de 3 6 7*)
5  for i = 0 to 2 do (*vider q*)
6    print_int (take q)
7  done ;;(*defiler 3 fois*)
8  try
9    print_int (take q);(*défiler une file vide*)
10 with Empty -> print_string "file vide\n";;
```

Une exception `Empty` est soulevée quand on essaye de défiler (`take`) une file vide.

Rendu

```
# open Queue;;
# let q = create ();;
val q : '_a Queue.t = <abstr>
# let q' = copy(q);;
val q' : '_a Queue.t = <abstr>
# push 3 q; push 6 q; push 7 q;;
- : unit = ()
# for i = 0 to 2 do (*vider q*)
  print_int (take q); (*défiler*)
done;;
    367- : unit = ()
# try
  print_int (take q);(*défiler une file vide*)
with Empty -> print_string "file vide\n";;
    file vide
- : unit = ()
```

Implémentations

En C, on peut créer les structures suivantes :

```
// ce qu'on met dans la file
typedef struct _element{// on y met ce qu'on veut :
  int v;// valeur (mais autres champs possibles)
}elt;

typedef struct _m{// liste dblt chaînée (why not ?)
  elt val;
  struct _m* next;// suivant
  struct _m* prev; // précédent
}maillon;

typedef struct _p{// structure de contrôle
  maillon * first;// pointe vers le 1er maillon
  maillon * last; // pointe vers le dernier
  // autres infos, par exemple :
  int size;// incr avec push, decr avec take
} grip;
```