

Algorithmique

Programmation Objet

Python



Andrea G. B. Tettamanzi

Université de Nice Sophia Antipolis

Département Informatique

andrea.tettamanzi@unice.fr

CM - Séance 7

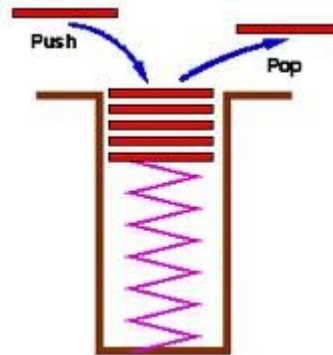
Piles, files et dèques

Plan

- Piles
- Files
- Dèques
- Files de priorité

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)
 - Les derniers éléments ajoutés à la pile seront les premiers à être récupérés.
- Pile d'assiettes : on ajoute des assiettes sur la pile, et on les récupère dans l'ordre inverse, en commençant par la dernière ajoutée.



Pile : opérations

- Sommet(P) :
 - renvoie le dernier élément ajouté et non encore retiré : le sommet (top en anglais)
- Empiler(P,elt) :
 - comme insérer;
 - place l'élément au sommet de la pile P (push en anglais)
- Dépiler(P) :
 - comme supprimer
 - retire de la pile le sommet (pop en anglais)
- estVide(P) :
 - Renvoie vrai si la pile est vide et faux sinon (empty)

Pile

- On considérera qu'à l'instar d'un tableau, une pile passée en paramètre est systématiquement passée en entrée/sortie
- Une pile passée en paramètre sera globalement modifiée si elle est localement modifiée
- Dépiler(P) : modifie effectivement la pile P

Pile : utilisation

- Une des structures de données les plus fondamentales en informatique : très simple et puissante
- Son utilisation la plus importante : gérer l'appel de sous-programmes (procédures, fonctions, méthodes)
- Pour chaque appel de fonction/méthode, la pile contient
 - Les paramètres d'appel des procédures ou fonctions.
 - Les variables locales.
 - Le point de retour.
- La plupart des microprocesseurs gèrent nativement une pile

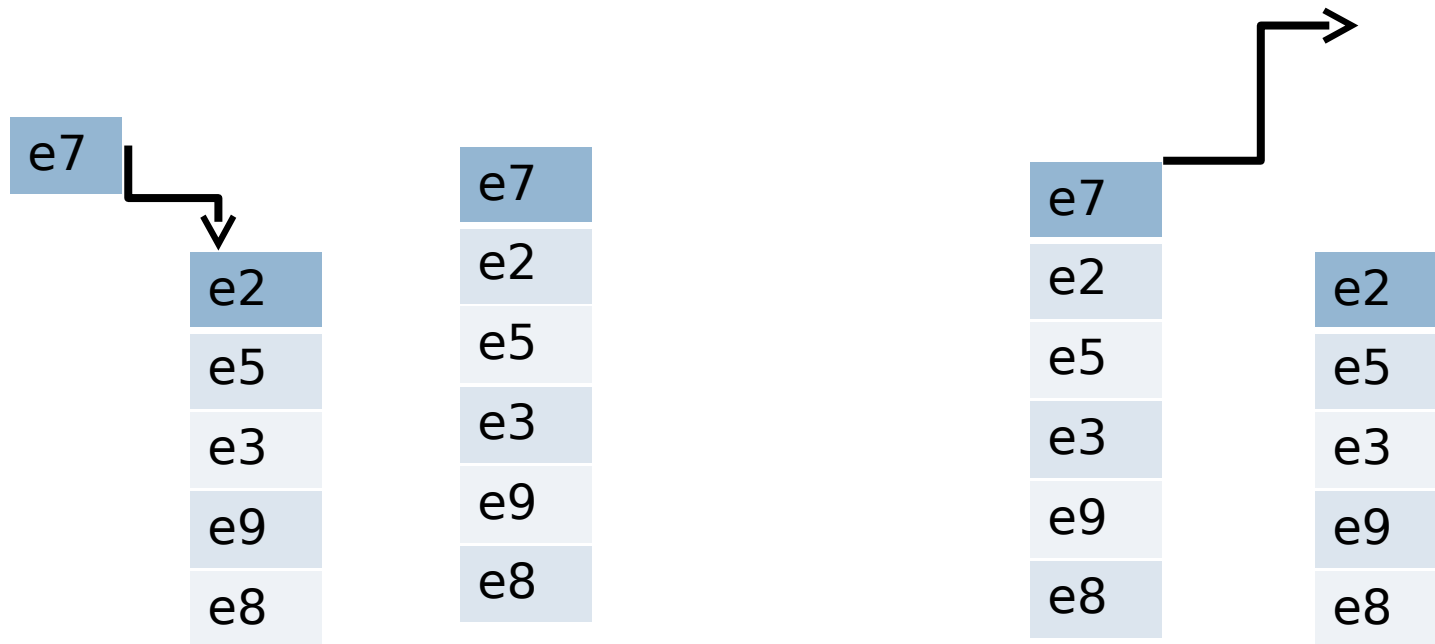
Pile dans les microprocesseurs

- Exemple : dans le microprocesseurs Intel X86 :
 - Le registre ESP sert à indiquer l'adresse du sommet d'une pile dans la RAM.
 - Les opcodes « push » et « pop » permettent respectivement d'empiler et de dépiler des données.
 - Les opcodes "call" et "ret" utilisent la pile pour appeler une fonction et la quitter par la suite en retournant à l'instruction suivant immédiatement l'appel.
 - En cas d'interruption, les registres EFLAGS, CS et EIP sont automatiquement empilés.

Pile : utilisation

- La fonction « Annuler la frappe » (en anglais *Undo*) mémorise les modifications apportées au texte dans une pile.
- Parseur d'expressions XML, des pages web
- Un algorithme de recherche en profondeur dans un graphe utilise une pile pour mémoriser les nœuds visités.
- Les algorithmes récursifs utilisent implicitement une pile d'appels

Pile : représentation



Empiler e7

Dépiler

Pile : vérification expression XML

- `<livre>`
 - `<chapitre>`
 - `<section>`
 - `</section>`
 - `<section>`
 - `<sous-section>`
 - `</sous-section>`
 - `<sous-section>`
 - `</section>`
 - `<section>`
 - `<sous-section>`
 - `</sous-section>`
 - `<sous-section>`
 - `</sous-section>`
 - `</chapitre>`
- `</livre>`

Pile : vérification expression XML

- Balise de début : <section>
- Balise de fin : </section>
- Doit être bien équilibré

- On rencontre une balise :
 - si balise de début, alors on l’empile
 - si balise de fin, alors le sommet doit être la balise de début correspondant sinon erreur. Si ok, alors on dépile

Pile : vérification expression XML

- `<livre>`
 - `<chapitre>`
 - `<section>`
 - `</section>`
 - `<section>`
 - `<sous-section>`
 - `</sous-section>`
 - `<sous-section>`
 - `</section>`
 - `<section>`
 - `<sous-section>`
 - `</sous-section>`
 - `<sous-section>`
 - `</sous-section>`
 - `</section>`
 - `</chapitre>`
- `</livre>`

← PROBLEME ICI

Pile : vérification expression XML

texteOk(TextXML) : booléen

créer pile P

parcourir le texte

pour chaque balise b faire

 si b est une balise de début alors empiler(P,b)

 sinon b' ← sommet(P)

 si b' n'est pas la balise de début de b

 alors erreur(« b et b' incompatible »)

 retourner faux

 sinon dépiler(P)

si P est vide alors retourner vrai

sinon retourner faux

Pile : réalisation

- A l'aide de tableaux (stack overflow)
- A l'aide de liste chaînée

Pile : réalisation par un tableau

- Attributs de la classe :
 - un tableau (T)
 - taille courante (s)
- Créer(n) : créer T de taille n; $s \leftarrow 0$
- Sommet() : renvoyer T[s]
- Empiler(elt) : $s \leftarrow s + 1$; $T[s] \leftarrow \text{elt}$
- Dépiler() : $s \leftarrow s - 1$
- estVide() : renvoyer $s = 0$ (on compte à partir de 1)
- Attention :
 - Dépiler : s ne doit pas devenir négatif
 - Empiler : *stack overflow* = dépassement de la taille de T

Pile : réalisation par une liste chaînée

- Attributs de la classe :
 - une liste chaînée (L)
- Créer(n) : créer une liste L vide
- Sommet() : renvoyer tête de la liste L
- Empiler(elt) : ajouter elt en tête à la liste L
- Dépiler() : supprimer tête de la liste L
- estVide() : L.estVide()
- Attention :
 - Dépiler : donnera une erreur si L est vide
 - Empiler : pas de *stack overflow*, la seule limite est l'espace de mémoire disponible

Files

- une **file** (en anglais *queue*) est une structure de données basée sur le principe « premier arrivé, premier sorti », en anglais FIFO (First In, First Out),
 - Les premiers éléments ajoutés à la file seront les premiers à être récupérés.
- Le fonctionnement ressemble à une file d'attente : les premières personnes à arriver sont les premières personnes à sortir de la file.



File : opérations

- Début(F) :
 - renvoie le premier élément ajouté et non encore retiré : le début ou le premier (*front* en anglais)
- Enfiler(F,elt) :
 - comme insérer;
 - place l'élément à la fin de la file F (*enqueue* en anglais)
- Défiler(F) :
 - comme supprimer
 - retire de la file le premier (*dequeue* en anglais)
- estVide(F) :
 - Renvoie vrai si la file est vide et faux sinon (*empty*)

File

- On considérera qu'à l'instar d'un tableau ou d'une pile, une file passée en paramètre est systématiquement passée en entrée/sortie
- Une file passée en paramètre sera globalement modifiée si elle est localement modifiée
- `F.défiler()` : modifie effectivement la file `F`

File : applications

- Application principale : les **buffers** (mémoire tampon = espace de mémorisation temporaire)
 - Les serveurs d'impression, qui doivent traiter les requêtes dans l'ordre dans lequel elles arrivent, et les insèrent dans une file d'attente.
 - 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'un graphe utilise une file pour mémoriser les nœuds visités.

File : réalisation

- A l'aide de tableaux
- A l'aide de liste chaînée

File : réalisation par un tableau

- Enfiler : on met après le dernier : ok
- Défiler : on retire le premier : le tableau se décale vers la droite !
 - 2 3 x x x
 - Enfiler 4 : 2 3 4 x x
 - Défiler : x 3 4 x x
 - Enfiler 1 : x 3 4 1 x
 - Défiler : x x 4 1 x
- On doit gérer un début et une fin de tableau
- Que faire lorsqu'on atteint le borne droite ?
- On devient circulaire

File : réalisation par un tableau

- Une file est implémentée par une mémoire tampon circulaire.
- Physiquement, on garde la structure de tableau, mais on considère que l'indice suivant le dernier (i.e. n) est 1 (celui du début) et que l'indice précédant le premier (i.e. 1) est la fin (i.e. n).
 - On utilisera un indice de début (d) et un indice de fin (f).
Au début $d=f=1$.
 - Quand on ajoute un élément on le met à la place de f et on incrémente f
 - Quand on supprime un élément on incrémente d
 - Incrémenter(x) : si $(x=n)$ {renvoyer 1}sinon{renvoyer $x + 1$ }
 - On laissera aussi une case vide

File : réalisation par un tableau

- Attributs de la classe :
 - Tableau T
 - Intier d (= Début de la file)
 - Intier f (= Fin de la file)
- Méthode auxiliaire :
 - incrémenter(entier n) : entier
renvoyer $n + 1$ (modulo T.taille)

File par tableau : méthodes

- début()
renvoyer $T[d]$
- enfiler(elt)
 $T[f] \leftarrow \text{elt}$
 $f \leftarrow \text{incrémenter}(f)$
- défiler()
 $d \leftarrow \text{incrémenter}(d)$
- estVide()
renvoyer $d = f$
- EstPleine()
renvoyer $\text{incrémenter}(f) = d$
- On ne gère pas défiler avec une file vide et enfiler avec une file pleine

File : réalisation par une liste chaînée

- Attributs
 - Liste L
 - Référence au dernier élément de la liste, d
- Méthodes
 - début() : renvoyer tête de L
 - enfiler(elt) : insérer elt après d ; $d \leftarrow elt$
 - défiler() : supprimer tête de L
 - estVide() : renvoyer L.estVide()
 - EstPleine : renvoyer faux

Deque

- une *double-ended queue* (abrégé **deque** et prononcé « deck ») est une structure de données qui réalise une file pour laquelle les éléments peuvent être ajoutés au début et en fin.
- Elle est souvent appelée *head-tail linked list*.

Deque : opérations

- `front(D)` : renvoie le premier
- `push_front(D,elt)` : ajoute au début
- `pop_front(D)` : supprime le premier

- `back(D)` : renvoie le dernier
- `push_back(D,elt)` : ajoute en fin
- `pop_back()` : supprime le dernier

- `estVide(D)` :
 - Retourne vrai si la deque est vide et faux sinon (*empty*)

Deque

- On considérera qu'à l'instar d'un tableau, d'une pile, ou d'une file, une deque passée en paramètre est systématiquement passée en entrée/sortie
- Une deque passée en paramètre sera globalement modifiée si elle est localement modifiée
- `D.Pop_front()` : modifie effectivement la deque D

Deque: réalisation

- Par un tableau
- Par une liste chaînée

Files de priorité

- Une **file de priorité** est un type abstrait élémentaire qui manipule des éléments, chacun ayant une clé (sa « priorité »), sur laquelle on peut effectuer trois opérations :
 - insérer un élément
 - lire puis supprimer l'élément ayant la plus grande clé
 - tester si la queue de priorité est vide ou pas.
- On ajoute parfois à cette liste l'opération
 - augmenter la clé d'un élément

File de priorité

- Une des structures de données les plus étudiées
- A donné naissance à des tas très complexes (vraiment très complexes)
- Souvent on impose que la queue soit monotone
 - La valeur du maximum ne fait que décroître
 - La valeur du minimum ne fait que croître

Liste de priorité : réalisation

- Une des implémentations les plus souples est d'utiliser un tas binaire
 - Augmenter ou diminuer la clé est possible
 - On peut ajouter des éléments
 - On peut demander le maximum (ou le minimum)
 - Toutes les opérations sont $O(\log(n))$

Merci de votre attention

