GM3 Nathalie Chaignaud

DS2 - Algorithmes et Structures de Données

# Vendredi 8 Janvier 2021

Durée 3H – Documents non autorisés

## 1. Pile – 5 pts

`quick-select` effectue la recherche du $k^{ème}$ plus petit entier `e` dans le tableau d'entiers `t` entre les indices `inf` et `sup`.

```
Const max = 10
Type tab = tableau [1..max] d'entier

Procédure quickSelect(E/S t : tab ; E k, inf, sup : entier ; S e : entier)
Var p : entier
Début
  Si inf=sup
    Alors e←t[inf]
    Sinon
      partitionner(t,inf,sup,p)  (* où p est l'indice du pivot *)
      Si k=p
        Alors e←t[p]
        Sinon Si k<p Alors quickSelect(t,k,inf,p-1,e) {@1}
                     Sinon quickSelect(t,k,p+1,sup,e) {@2}
              FinSi
      FinSi
  FinSi
Fin
```

t : 4 10 8 18 12 2 16 0 14 6

**Simuler la pile** sur l'appel `quickSelect(t,7,1,max,e)` {@0} en donnant aussi les valeurs intermédiaires de t après l'appel de `partitionner` (donné dans le résumé).

## 2. Représentations d'un polynôme – 5 pts

Pour manipuler des polynômes, on peut utiliser plusieurs structures de données, dont les tableaux et les listes chaînées. Par exemple, on peut utiliser ces 2 représentations :

- ```
  Const degmax = 20
  Type poly1 = tableau [0..degmax] de réel
  ```
  où la valeur d'une case du tableau représente le coefficient (réel) du terme dont le degré (entier ≤ `degmax`) est donné par l'indice de la case.
- ```
  Type terme = Enregistrement
                    coef : réel
                    deg : entier
                    suiv : ^terme
               FinEnregistrement
       poly2 = ^terme
  ```
  où les termes du polynôme sont chaînés et triés dans l'ordre croissant des degrés.

On souhaite pouvoir passer d'une représentation à l'autre.

**2.1.** Ecrire en pseudo-langage une fonction `poly1To2` qui fait la conversion du type `poly1` vers le type `poly2`.

**2.2.** Ecrire en pseudo-langage une fonction `poly2To1` qui fait la conversion du type `poly2` vers le type `poly1`.

**3. Gestion de trains – 10 pts**

On souhaite pouvoir gérer des trains avec leurs différents arrêts aux stations et leurs horaires. On propose la structure de données suivante pour représenter la liste de trains :

```
Type  arret = Enregistrement
                  nom, horaire : chaîne
                  suiv : ^arret
              FinEnregistrement
      train1 = Enregistrement
                  num : chaine
                  la : ^arret
                  suiv : ^train1
             FinEnregistrement
      ltrain1 = ^train1
```

**3.1.** Faire un dessin de la structure de données avec les données suivantes : numéro ; arrêt/horaire

346 ; Paris(8h40), Rouen(10h05), Yvetot(10h26), Bréauté(10h40), LeHavre(10h54)

348 ; Paris(18h41), Rouen(20h07), Yvetot(20h28), Bréauté(20h42), LeHavre(20h56)

355 ; LeHavre(8h02), Bréauté(8h18), Yvetot(8h31), Rouen(8h56), Paris(10h23)

357 ; LeHavre(17h02), Bréauté(17h17), Yvetot(17h31), Rouen(17h56), Paris(19h23)

402 ; LeHavre(7h28), Bréauté(7h46), Yvetot(8h02), Pavilly(8h14), Barentin(8h17), Rouen(8h30)

405 ; LeHavre(19h24), Bréauté(19h43), Yvetot(19h58), Pavilly(20h10), Barentin(20h13), Rouen(20h26)

410 ; Rouen(7h26), Barentin(7h37), Pavilly(7h40), Yvetot(7h51), Bréauté(8h06), LeHavre(8h25)

413 ; Rouen(17h10), Barentin(17h22), Pavilly(17h25), Yvetot(17h37), Bréauté(17h51), LeHavre(18h08)

**3.2.** Ecrire en pseudo-langage la procédure `supprimeTrain` qui supprime de la liste `lt1` un train existant donné par son numéro, ainsi que tous les arrêts. **Faire un dessin.**

```
Procédure supprimeTrain(E num : chaîne ; E/S lt1 : ltrain1)
```

**3.3.** Ecrire en pseudo-langage la fonction `donneTrainStation` qui renvoie un pointeur sur une liste de trains (non vide) passant par une station (existante et donnée par son nom). **Faire un dessin.**

```
Fonction donneTrainStation(lt1 : ltrain1, nom : chaîne) : ltrain2
```

```
Type  train2 = Enregistrement
                  num : chaîne
                  suiv : ^train2
               FinEnregistrement
      ltrain2 = ^train2
```

**3.4.** Ecrire en pseudo-langage la fonction `donneTrains` qui renvoie un pointeur sur une liste de trains passant successivement par la station1 puis la station2 (données respectivement par `nom1` et `nom2`). La liste peut être égale à `nil`. **Faire un dessin.**

```
Fonction donneTrains(lt1 : ltrain1, nom1, nom2 : chaîne) : ltrain2
```

**3.5.** Ecrire en pseudo-langage une procédure `créeTabStation` qui, à partir d'un tableau de `n` chaines représentant les noms de station (`tch`), crée un tableau de stations (`tst`) où chacune d'elles pointe sur la liste des trains qui y passent. On utilisera la fonction `donneTrainStation` de la question 3.2. **Faire un dessin.**

```
Procédure créeTabStation(E lt1 : ltrain1, tch : tab-ch, n : entier ; S tst :
tab-st)
```

```
Const max = 100
Type  station = Enregistrement
                  nom : chaîne
                  lt : ltrain2
                FinEnregistrement
      tab-st = tableau[1..max] de station
      tab-ch = tableau[1..max] de chaine
```

**3.6.** Ecrire en pseudo-langage la fonction `donneStationMaxTrain` qui renvoie le nom de la station (la première trouvée) par laquelle passent le plus de trains. V**otre fonction doit être optimisée.**

```
Fonction donneStationMaxTrain(tst : tab-st, n : entier) : chaîne
```