Collège de Candolle

ÉPREUVE REGROUPÉE

Discipline: INFORMATIQUE **Date:** 13/12/2024
Cours: 2 IN DP 03 / 04 / 05 **Durée:** 105 minutes

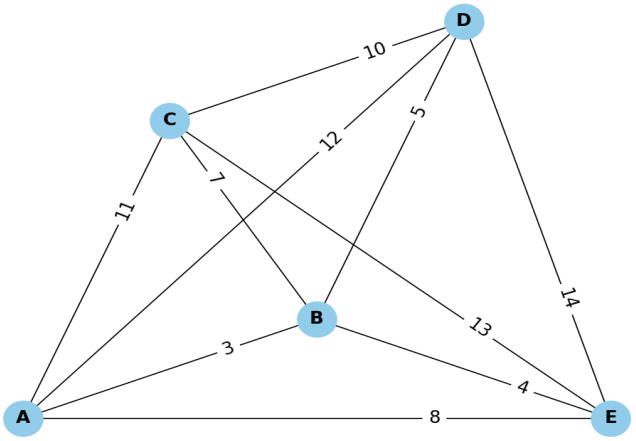
Maîtres: JC Barros
Matériel autorisé: aucun

Consignes de travail:
- **Ecriture lisible et présentation soignée**
- **Mise en évidence des résultats**
- **Il y a 10 pages dont 2 d'aide-mémoire**

Nom et prénom: **Total: 50 points**

Ex 1: *(/ 5 pts)* Voyageur de commerce glouton

Voici un graphe représentant 5 villes et les distances qui les séparent.



Un voyageur de commerce nommé Edsger et habitant dans la ville "E" souhaite partir de chez lui (ville E), passer par chaque autre ville une seule fois et revenir chez lui.

Pour l'aider à trouver l'itinéraire le plus court, on demande d'utiliser un algorithme glouton.

a. (1pt) Proposer un critère glouton pour choisir, à chaque étape, la prochaine ville à visiter:

b. (2pts) Suivre cet algorithme pour déterminer l'itinéraire que devrait suivre Edsger:

i. Itinéraire glouton:

ii. Distance totale:

c. (2pts) Est-ce que cet itinéraire est vraiment le meilleur possible?

i. Si oui, justification:

ii. Sinon, itinéraire meilleur avec sa distance totale:

Ex 2: *(/ 3 pts)* Stratégies algorithmiques

Donner le nom des trois stratégies algorithmiques vues en classe:

a. (1pt) Cette stratégie consiste à explorer toutes les possibilités d'un problème avant de donner une réponse:

b. (1pt) Cette stratégie fait un choix optimal à chaque étape en écartant toutes les autres possibilités. Ceci conduit à une résolution beaucoup plus rapide mais elle peut parfois ne pas être idéale:

c. (1pt) Cette stratégie réduit la complexité d'un algorithme en divisant son entrée en plusieurs parties puis en résolvant chaque sous-partie avant de combiner leurs solutions pour trouver une solution globale:

Ex 3: *(/ 8 pts)* Algorithmes de tri...

Voici une liste désordonnée de nombres: **`[ 1, 8, 0, 5, 3 ]`**

a. (4pts) Choisir un algorithme de tri simple (parmi sélection, insertion ou bulles)

Écrire chaque étape dans la résolution de cet algorithme, c'est-à-dire la liste des nombres après chaque échange effectué dans l'algorithme choisi. De plus, souligner les nombres triés par l'algorithme au fur et à mesure de l'avancement du tri.

- Algorithme choisi:
- Etapes de l'algorithme pour trier la liste:

b. (4pts) Faire de même avec un algorithme avancé (tri par fusion ou tri par pivot)

Chaque étape peut contenir plusieurs listes.

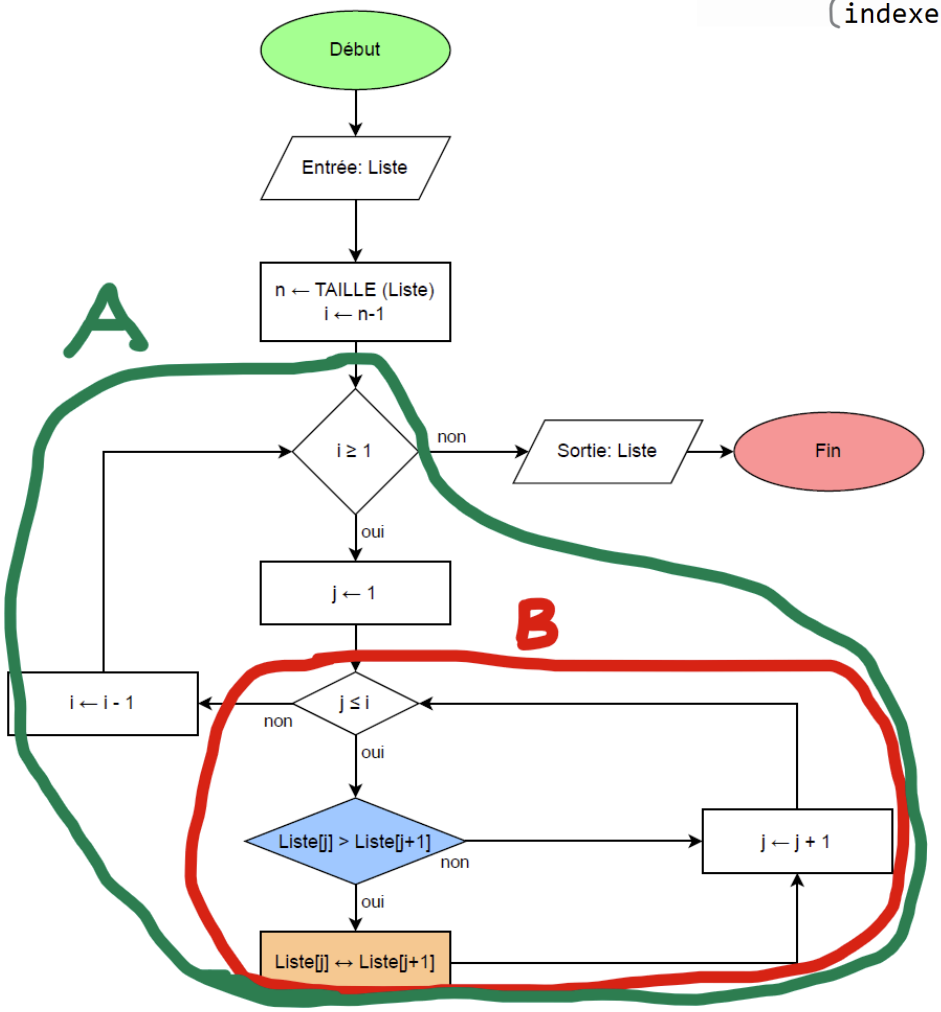
Nouvelle liste désordonnée de nombres: **`[ 6, 8, 0, 3, 5, 2, 9, 4 ]`**

- Algorithme choisi:
- Etapes de l'algorithme pour trier la liste:

Ex 4: *(/ 7 pts)* Logigramme

Voici le logigramme d'un algorithme de tri (les index des éléments de la liste vont de 1 à n) :

```
exemple: liste = [5, 4, 8, 0]
         (indexes: 1  2  3  4)
```



a. (1pt) De quel algorithme s'agit-il?

b. (2pts) Observer ce que fait chaque instruction de la boucle B (entourée en rouge) et résumer en une phrase le travail effectué par cette boucle.

c. (2pts) Faire de même pour la boucle A (entourée en vert).

d. Avec la liste exemple ([5, 4, 8, 0]), combien de fois va se répéter...
   i. (1pt) la boucle verte?

   ii. (1pt) la boucle rouge?

Ex 5: *(/ 8 pts)* Pseudocode

Voici le pseudo code d'un algorithme:

```
ENTREE système
ENTREE somme
rendu = [] (liste vide)
TANT_QUE (somme > 0)
    {
     monnaie = système[dernier]   (monnaie de plus haute valeur)
     SI (monnaie > somme)
     ALORS
           {
            ENLEVER monnaie DE système
           }
     SINON
           {
            AJOUTER monnaie A rendu
            somme = somme - monnaie
           }
    }
SORTIE rendu
```

a. (1pt) A quoi sert cet algorithme?

b. (1pt) A quelle stratégie algorithmique appartient-il ?

c. (1pt) Entourer la boucle de cet algorithme en couleur.

d. Avec le système **[1, 2, 5]** et la somme **6**, combien de fois sera exécutée...
   i. (1pt) la ligne surlignée en orange(retrait d'une monnaie de la liste)

   ii. (1pt) la bloc bleu correspondant au TANT_QUE

e. Expliquer brièvement ce que fait l'algorithme dans les deux situations suivantes:
   i. (1pt) système **[1, 2, 5]** et la somme **0**

   ii. (2pts) système **[1, 2, 5]** et la somme **0,5**

Ex 6: *(/ 4 pts)* Code Python

Voici un programme Python:

```
1  def cherche(liste, valeur):
2      trouve = False
3      for element in liste:
4          if element == valeur:
5              trouve = True
6      return trouve
7
8  l = [1, 4, 7, 11, 13, 42, 57]
9  v = 4
10 print(cherche(l, v))
```

a. (1pt) Que fait ce programme?

b. (1pt) Qu'affiche ce programme quand on l'exécute?

c. (2pts) Combien de fois la boucle des lignes 3 à 5 va-t-elle se répéter ?

d. (4pts) Traduire la fonction **cherche( )** en logigramme ou en pseudo code ci-dessous:

Ex 7: *(/ 5 pts)* Récursivité

a. (2pts) Décrire en quelques mots ce qu'est la récursivité en informatique:

b. (2pts) Dans la fonction python récursive ci-dessous, indique à quelle ligne on trouve...

i. la condition de sortie de la récursion

ii. l'appel récursif

iii. la définition de la fonction

iv. le programme principal

c. (1pt) Qu'affiche ce programme quand on l'exécute?

Programme python récursif:

```
1  def compte_a_rebours(n):
2      """
3      affiche un compte à rebours
4      """
5      if n == 0:
6          print(n)
7      else:
8          print(n)
9          compte_a_rebours(n-1)
10
11
12 compte_a_rebours(3)
```

Ex 8: *(/ 6 pts)* CHOISIR **1 SEULE QUESTION** PARMI LES DEUX QUESTIONS SUIVANTES

ENTOURER LE CHOIX CI-APRÈS: Je choisis la partie A. / Je choisis la partie B.

PARTIE A :

Voici une fonction Python récursive:

```
1  def tri_pivot(liste):
2      # si la liste est vide ou contient un seul élément
3      if len(liste) <= 1:
4          return liste
5      else:
6          # pivot: dernier élément de la liste
7          pivot = liste[-1]
8
9          gauche = [x for x in liste if x < pivot]
10         milieu = [x for x in liste if x == pivot]
11         droite = [x for x in liste if x > pivot]
12
13         # Appliquer tri_pivot aux sous-listes et les recombiner
14         return tri_pivot(gauche) + milieu + tri_pivot(droite)
```

a. (2pts) Quelle stratégie algorithmique emploie cet algorithme sous forme de code Python ?
Justifier!

b. (2pts) A quelle ligne se fait l'appel récursif ?
Justifier!

c. (2pts) A quelle ligne s'effectue la combinaison des solutions aux sous-parties de liste à trier?
Justifier!

PARTIE B:

Indiquer dans l'ordre tout ce qui sera affiché dans la console à la fin de l'exécution de chaque programme Python ci-dessous. Justifier avec un tableau de valeurs des variables.

Code	Justification (table des valeurs pour chaque variable)	Affichage

`m = 5`
`n = 3`
`while n<6 :`
`    print(n)`
`    n=n+1`
`print(n*10)`
 	
(3pts)		

`for i in range(3, 5):`
`   for j in range(2):`
`        print(i * j)`
 	
(3pts)		

FIN DE L'ÉPREUVE (aide-mémoire suit)

Aide-mémoire informatique

2ème année, 1er semestre

Algorithmes

Stratégies:

Force brute

Consiste à calculer toutes les possibilités d'un problème puis à trouver la (les) solution(s) optimale(s). On trouve toujours la meilleure solution mais selon l'algorithme et la taille de l'entrée, il est possible que le temps de calcul soit rédhibitoire.

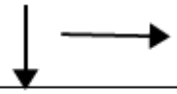
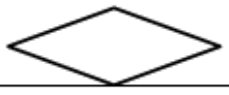
Glouton

On choisit, à chaque étape de l'exécution de l'algorithme, une possibilité qu'on estime optimale (localement). Il faut donc un critère pour faire ce choix. On laisse tomber toutes les autres et on gagne ainsi en efficacité.
Par contre, on n'est pas sûr de trouver la solution optimale (globalement).

Diviser pour régner

On divise un problème complexe en plusieurs sous-problèmes moins complexes. Une fois que les sous-problèmes sont suffisamment petits, on les résout. On finit par combiner les sous-solutions afin d'obtenir la solution globale au problème.

Rappel pour les logigrammes:

Symbole	Nom	Rôle
	Flèche	Indique la séquence du programme
	Terminaison	Débute ou termine un programme
	Processus	Exécute une tâche
	Décision	Génère une décision (oui, non, etc.)
	Entrée/Sortie	Accepte une entrée ou produit une sortie

Programmation (Python)

Calculs :
opérations de base: + - * / **
reste et quotient de division euclidienne: % //
var % div vaut 0 si var est divisible par div
ex: 5 ** 2 → 25 5 // 2 → 2 5 % 2 → 1

Variables et affectations :
types: int, float, str, bool, list ...
int(), float() pour **convertir** en int ou en float
affectation simple : `ma_variable = -5`
permutation de variables: `a, b = b, a`
affectations combinées: +=, -=, *=, etc.
ex: x += 1 est comme x = x + 1

Listes :
liste: `L = [-1, 3.2, "abc"]`
liste vide : `[]`
multiplication: `[0]*3 -> [0, 0, 0]`
élément d'une liste: `L[1] = 5`
doit déjà exister, l'indice commence à 0.
ajouter un élément à une liste :
`L.append(-2.5)`
supprimer élément à la position i :
`del L[i]` ou `L.pop(i)`
supprimer élément de valeur v dans L :
`L.remove(v)`
longueur d'une liste L :
`len(L)`
somme des valeurs d'une liste:
`sum(liste)`
slice: `L[i:j]` (i inclus ! j exclu)

Chaînes de caractères :
chaîne: `txt="bonjour"` ou `txt='bonjour'`
concaténation: txt + " est un texte"
élément, slice d'une chaîne: texte[i]
comme listes mais non modifiable

Entrées/sorties :
sortie: print(valeur,... , end='\n')
valeur de type quelconque, end optionnel
entrée: texte = input("message")
Attention: retourne toujours un str

Commentaires :
sur une ligne : # texte
plusieurs lignes : '''texte''' ou """texte"""

Structures de contrôle :
1ère ligne finit par ':' puis indentation des lignes du bloc
tests simples: ==, !=, <, <=, >, >=, in
tests combinés: and, or, not, a < b < c

conditionnelle
```
if condition1:
    instructions
elif condition2:
    # elif =contraction de else if
    instructions
else:
    instructions
```

boucle while (boucle non bornée) :
```
while condition:
    instructions
```

boucle for (boucle bornée) :
```
for i in liste: # boucle parcourt
la liste
    instructions
for i in range(5):#pour i de 0 à 4
    instructions
```

range(i,j,k): de i (inclus) à j (exclu) par pas de k
sauter, quitter une boucle : continue, break

Fonctions:
définition de la fonction f à deux paramètres:
```
def f(x, y):
    reponse = x**2 + y**2
    return reponse
```

appel de la fonction: f()
variable = f(3, 5)
print(f(2, 6))
etc.