Mathieu Vellozzi
Ilyas El Habachi
Mathieu Corne

Compte Rendu Algorithmie

Mathieu Vellozzi
Ilyas El Habachi
Mathieu Corne

Nous avons été contacter par l'aérodrome de Grenoble Alpes Isère, pour leurs faire un programme leur permettent de répondre à plusieurs fonctionnalités :

- Affichage des informations sur les vols sur les tableaux dédiés
- Recherche d'un vol à partir du nom de la compagnie, de la destination ou de l'heure de décollage
- Affichage de la liste des passagers sur les écrans des salles d'embarquement
- Gestion des retards et des annulations des vols
- Maximiser l'utilisation de la piste

Pour répondre à leur demande nous avant tout d'abord pensée aux algorithmes nécessaires comme :

- Algorithme de tri
- Algorithme de recherche
- Algorithme fonctionnalité

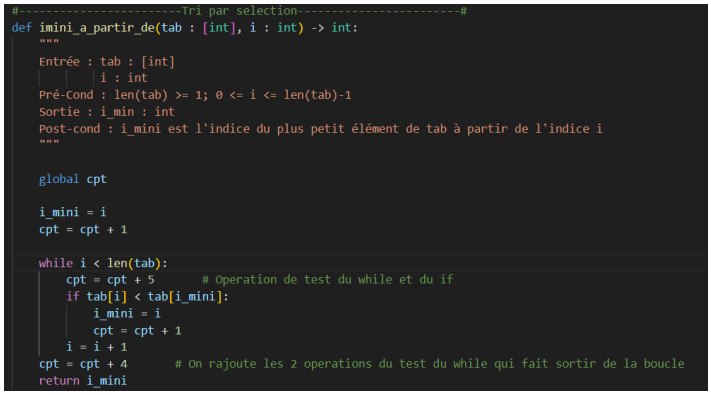
En second temps, nous somme passer à la réalisation de ces derniers.

1. Algorithme de Tri

- Algorithme tri sélections
- Algorithme tri insertions
- Algorithme tri bulle
- Algorithme tri rapide
- Algorithme de tri fusion

A. Algorithme tri par sélections

Cet algorithme consiste à diviser le tableau en deux parties : une partie triée et une partie non triée. À chaque itération, l'algorithme trouve le plus petit élément dans la partie non triée et l'échange avec le premier élément de la partie non triée. Cela continue jusqu'à ce que tout le tableau soit trié.
Pour nous faciliter la tâche nous avons coder une fonction « imini_a_partire_de » qui trouve l'indice du plus petit élément du tableau à partir de l'indice i .



```python
#-----------------------Tri par selection-----------------------#
def imini_a_partir_de(tab : [int], i : int) -> int:
    """
    Entrée : tab : [int]
             i : int
    Pré-Cond : len(tab) >= 1; 0 <= i <= len(tab)-1
    Sortie : i_min : int
    Post-cond : i_mini est l'indice du plus petit élément de tab à partir de l'indice i
    """

    global cpt

    i_mini = i
    cpt = cpt + 1

    while i < len(tab):
        cpt = cpt + 5       # Operation de test du while et du if
        if tab[i] < tab[i_mini]:
            i_mini = i
            cpt = cpt + 1
        i = i + 1
    cpt = cpt + 4       # On rajoute les 2 operations du test du while qui fait sortir de la boucle
    return i_mini
```

Mathieu Vellozzi
Ilyas El Habachi
Mathieu Corne

```python
def tri_selection(tab : [int]):
    """
    Entrée / Sortie : tab : [int]
    Pré-Cond : len(tab) > 1
    Post-Cond : Le tableau est triée
    """

    global cpt

    i = 0
    cpt = cpt + 1

    while i < len(tab)-1:
        i_mini = imini_a_partir_de(tab, i)
        temp = tab[i]
        tab[i] = tab[i_mini]
        tab[i_mini] = temp
        i = i + 1
        cpt = cpt + 13      # Opreations dans le while + Operations de test du while
    cpt = cpt + 3           # On rajoute les 2 operations du test du while qui fait sortir de la boucle
```

B. Algorithme tri par insertions

Cet algorithme consiste à diviser le tableau en deux parties : une partie triée et une partie non triée. À chaque itération, l'algorithme prend un élément de la partie non triée et l'insère à la bonne position dans la partie triée, en déplaçant les éléments plus grands vers la droite. Comme pour le cas précèdent nous avons créé un code secondaire « inserer » qui permet de placer un élément a ça bonne place.

```python
#----------------------Tri par Insertion----------------------#
def inserer(tab : [float], i : int):
    """
    Entrée / Sortie: Tab : [float]
    Entrée: i : int
    Pré-Cond: len(tab) >= 1; 0 <= i <= len(tab)-1
              Tab est trié jusqu'à l'indice i-1
    Post-Cond: Tab est trié jusqu'à l'indice i
    """

    global cpt

    while i>0 and tab[i] < tab[i-1]:
        temp = tab[i-1]
        tab[i-1] = tab[i]
        tab[i] = temp
        i = i - 1
        cpt = cpt + 16      # Opreations dans le while + Operations de test du while
    cpt = cpt + 5           # On rajoute les 2 operations du test du while qui fait sortir de la boucle

def tri_insertion(tab : [float]):
    """
    Entrée / Sortie: Tab : [float]
    Pré-Cond: len(tab) >= 1
    Post-Cond: Les elements de tab sont trié en ordre croissant
    """

    global cpt

    i = 0
    cpt = cpt + 1
    while i<len(tab):
        inserer(tab, i)
        i = i + 1
        cpt = cpt + 4       # Opreations dans le while + Operations de test du while
    cpt = cpt + 2           # On rajoute les 2 operations du test du while qui fait sortir de la boucle
```

C. Algorithme tri a bulle

Cet algorithme compare les éléments adjacents du tableau et les échange s'ils ne sont pas dans l'ordre, répétant ce processus jusqu'à ce que le tableau soit entièrement trié.
Nous avons la aussi coder une fonction secondaire « permuter_jusqu_a » c'est une revisitassions de l'algorithme « inserer ».

Mathieu Vellozzi
Ilyas El Habachi
Mathieu Corne

```python
def permuter_jusqu_a(tab : [float], i : int):
    """
    Entrée / Sortie: Tab : [float]
    Entrée: i : int
    Pré-Cond: len(tab) >= 1; 1 <= i <= len(tab)-1
              Tab est trié jusqu'à l'indice i+1
    Post-Cond: Tab est trié à partir de l'indice i
    """

    global cpt

    j = 0
    cpt = cpt + 1
    while j < i:
        cpt = cpt + 7       # Operations du test du if + Operations du test de la boucle
        if tab[j] > tab[j+1]:
            temp = tab[j]
            tab[j] = tab[j+1]
            tab[j+1] = temp
            cpt = cpt + 9   # Operations dans le if
        j = j + 1
    cpt = cpt + 3           # On rajoute les 2 operations du test du while qui fait sortir de la boucle

def tri_a_bulles(tab : [float]):
    """
    Entrée / Sortie: Tab : [float]
    Pré-Cond: len(tab) >= 1
    Post-Cond: Les elements de tab sont trié en ordre croissant
    """

    global cpt

    i = len(tab)-1
    cpt = cpt + 2
    while i > 0:
        permuter_jusqu_a(tab, i)
        i = i - 1
        cpt = cpt + 3       # Opreations dans le while + Operations de test du while
    cpt = cpt + 1           # On rajoute les 2 operations du test du while qui fait sortir de la boucle
```

D. Algorithme tri rapide

Cet algorithme fait appel à la récursivité, il sépare le tableau en deux avec un pivot et il met tous les élément plus petit avant et les plus grand après.
Nous avons cette fois ci coder 2 fonctions secondaire :
- une fonction « echange » qui échange 2 éléments
- une fonction « repartition » qui permet de placer les éléments autour du pivot

```python
def echange(tab: [float], i: int, j: int):
    """
    Échange les éléments aux positions i et j dans le tableau tab.
    """
    global cpt
    temp = tab[i]
    tab[i] = tab[j]
    tab[j] = temp
    cpt = cpt + 3                   #cpt+3 pour les 3 affectations
```

```python
def repartition(tab: [float], i_debut: int, i_fin: int) -> int:
    """
    Entrée / Sortie: Tab : [float]
    Pré-Cond: len(tab) >= 1; 0 <= i_debut <= i_fin <= len(tab)-1
    Sortie: i : int
    Post-Cond: tab[i] vaut tab[i_debut]
               les éléments tab[i_debut] à tab[i-1] sont inférieurs à tab[i],
               les éléments de tab[i+1] à tab[i_fin] sont supérieurs à tab[i]
    """
    global cpt

    pivot = tab[i_debut]
    i = i_debut + 1
    j = i_fin
    cpt = cpt + 5

    while i <= j:
        cpt = cpt + 6  # Operations du test du if + Operations du test de la boucle
        if tab[i] > pivot and tab[j] <= pivot:
            echange(tab, i, j)
            i = i + 1
            j = j - 1
            cpt = cpt + 2  # Operations dans le if
        else:
            cpt = cpt + 2  # Operations du test du if
            if tab[i] <= pivot:
                i = i + 1
                cpt = cpt + 2  # Operations dans le if
            cpt = cpt + 2  # Operations du test du if
            if tab[j] > pivot:
                j = j - 1
                if i <= j:  # Ajout de cette condition pour éviter que j devienne négatif
                    cpt = cpt + 2  # Operations dans le if
    cpt = cpt + 1  # On rajoute une operation du test du while qui fait sortir de la boucle
    echange(tab, i - 1, i_debut)

    return i - 1
```

Mathieu Vellozzi
Ilyas El Habachi
Mathieu Corne

```python
def tri_rapide_rec(tab, i_debut, i_fin):
    global cpt

    cpt = cpt + 2           # Operations du test du if
    if i_debut + 1 == i_fin:
        cpt = cpt + 3       # Operations du test du if
        if tab[i_fin] < tab[i_debut]:
            echange(tab, i_debut, i_fin)
    else:
        cpt = cpt + 2       # Operations du test du if
        if i_debut < i_fin:
            i_pivot = repartition(tab, i_debut, i_fin)
            tri_rapide_rec(tab, i_debut, i_pivot - 1)
            tri_rapide_rec(tab, i_pivot + 1, i_fin)
            cpt = cpt + 1
```

E. Algorithme tri fusion

Cet algorithme fait appel à la récursivité, il calcule le milieu et sépare le tableau en deux tableaux temporaires jusqu'à ceux qu'il y est que des tableaux de 1 élément grâce à la récursivité puis il fusion les différant tableau en les trient. Pour faciliter la création du code nous avons fait une fonction secondaire « fusionner » qui fusionne les tableau temporaires dans le tableau de base.

```python
def fusionner(tab : [int], i_debut : int, i_millieu : int, i_fin : int):
    global cpt
    temp = numpy.zeros(i_fin-i_debut+1, float)
    i = i_debut
    j = i_millieu + 1
    k = 0
    cpt = cpt + 5
    while i <= i_millieu and j <= i_fin:
        cpt = cpt + 6       # Operations du test du if + Operations du test de la boucle
        if tab[i] < tab[j]:
            temp[k] = tab[i]
            i = i + 1
            cpt = cpt + 2
        else:
            temp[k] = tab[j]
            j = j + 1
            cpt = cpt + 5
        k = k + 1
        cpt = cpt + 5       # On rajoute les 3 operations du test du while qui fait sortir de la boucle
    while i <= i_millieu:
        temp[k] = tab[i]
        i = i + 1
        k = k + 1
        cpt = cpt + 8       # Operations dans la boucle + Une operation de test de la boucle
    cpt = cpt + 1           # On rajoute une operation du test du while qui fait sortir de la boucle
    while j <= i_fin:
        temp[k] = tab[j]
        j = j + 1
        k = k + 1
        cpt = cpt + 8       # Operations dans la boucle + Une operation du test de la boucle
    k = 0
    cpt = cpt + 3           # On rajoute une operation du test du while qui fait sortir de la boucle
    while k < len(temp):
        tab[i_debut + k] = temp[k]
        k = k + 1
        cpt = cpt + 8       # Operations dans la boucle + Une operation du test de la boucle
    cpt = cpt + 2           # On rajoute les 2 operations du test du while qui fait sortir de la boucle
```

```python
def tri_fusion_rec(tab : [int], i_debut, i_fin):
    """
    Entrée / Sortie: tab : [float]
    Pré-Cond: len(tab) >= 1
    Post-Cond: Les elements de tab sont trié en ordre croissant
    """

    global cpt

    i_millieu = (i_debut + i_fin) // 2
    cpt = cpt + 4
    if i_debut < i_fin:
        tri_fusion_rec(tab, i_debut, i_millieu)
        tri_fusion_rec(tab, i_millieu+1, i_fin)
        fusionner(tab, i_debut, i_millieu, i_fin)
        cpt = cpt + 1
```

2. Algorithme recherche

- Recherche par dicotomie
- Recherche vols à partir de

A. Recherche par dicotomie

Cet algorithme fait appel à la récursivité, il calcule la moitié du tableau Pui regarde si l'élément et plus petit ou plus grand et réitère avec la partie du bas si le numéro et plus petit ou dans la partie haut si le numéro et plus grand

Mathieu Vellozzi
Ilyas El Habachi
Mathieu Corne

```python
def rech_dicotomie(tab : [int], valeur : int) -> bool:
    """
    Pré-cond :
    Sortie : bool
    Post-cond : Trouve vaut False si "valeur" n'est pas dans "tab"
                Trouve vaut True si "valeur" est dans "tab"
    """
    global cpt
    i_debut = 0
    i_fin = len(tab)-1
    touve = False
    cpt = cpt + 4                           #cpt+4 pour les 3 affectation et 1 soustractions
    while i_debut <= i_fin and not touve:
        i_millieu = (i_debut + i_fin) // 2
        cpt = cpt + 5                       #cpt+5 pour 2 comparaison, 1 affectation et 2 operations elementaire
        if tab[i_millieu] == valeur:
            touve = True
            cpt = cpt + 2                   #cpt+2 pour 1 comparaison et 1 affectation
        else:
            if tab[i_millieu] > valeur:
                i_fin = i_millieu - 1
                cpt = cpt + 3               #cpt+3 pour 1 comparaison, 1 affectation et 1 operation elementaire
            else:
                i_debut = i_millieu + 1
                cpt = cpt + 2               #cpt+2 pour 1 affectation et 1 opperation elementaire
    cpt = cpt + 2                           #cppt+2 pour 2 teste de sortie de la boucle while
    return touve
```

B. Recherche vols à partir de

La fonction « recherche_vol_a_partir_de » recherche tous les indices dans le tableau tab où la valeur est égale à la valeur fournie.

```python
def recherche_vol_a_partir_de(tab, valeur):
    """
    entré= tab[int] et valeur(int)
    pre-cond = len(tab)>0 and 0<tab[]<5 000
    sortie = tab_ind
    post-cond = tab_ind contient tout les indices des elément de tab qui sont égaux à valeur
    """

    global  cpt

    tab_ind = numpy.zeros(len(tab),int)
    i = 0
    k = 1
    cpt = cpt+3                             #cpt+3 pour 3 affectations
    while i < len(tab):
        cpt = cpt + 3                       #cpt+3 pour 1 comparaison, 1 affectation et 1 adition
        if tab[i] == valeur:
            tab_ind[k] = i
            k = k + 1
            cpt = cpt + 4                   #cpt+4 pour 1 comparaison, 2 affectations et 1 adition
        i = i + 1
    cpt = cpt + 1                           #cpt+1 pour la sortie de la boucle while
    return tab_ind
```

3. Algorithme demander

- algorithme N°1 afficher 3h
- algorithme N°2 recherches vol à partir de
- algorithme N°3 afficher passager
- algorithme N°4 vols avec retard

A. Afficher 3h

Le code parcours tab et regarde quand l'heure est identique et l'affiche (l'affichage et en commentaire pour pas saturer l'affichage)

```python
def affichage_vols_3h(tab, heure):
    global cpt
    i=0
    h_fin=heure + 300
    cpt = cpt + 3                           #cpt+3 pour 2 affectation et 1 adition
    while i < len(tab):
        cpt = cpt + 3                       #cpt+3 pour 1 comparaison, une affectation et 1 aditions
        if tab[i]>=heure and tab[i]<=h_fin:
#           print("le vols décole a:",tab[i]) #il y a plus de print pour l'affichage de toout les information d'un vol mais pour la complexiter je en n'ai mie que 1
            cpt = cpt + 3                   #cpt+3 pour 3 comparaison (je ne compte pas la complexiter des print)
        i = i + 1
    cpt = cpt + 1                           #cpt+1 pour l la sortie de la boucle
```

B. Recherche vol à partir de

Vue juste au-dessus pour les algorithmes de recherche (code utilise car il n'a pas besoin de tier le tableau avant

C. Afficher passager

Fait un premier tri pour mettre les plus jeune au plus vieux. Ensuit parcours le début du tableau pour avoir les personnes de plus de 12 ans et refait un tri par rapport au prix du billet que on inverser pour avoir le plus cher en premier. Pour finir re parcours le tableau pour vérifier que les tris sois parfait

Mathieu Vellozzi
Ilyas El Habachi
Mathieu Corne

```python
def affichage_passager(tab : [[int]],nb_ligne):
    global cpt
    i = 0
#----------pour afficher le tableau avant trie----------#
#     k = 0
#     print("avant")
#     while k< nb_ligne:
#         print("age:",tab[k][0])
#         print("prix:",tab[k][1])
#         print("n° initial:",tab[k][2])
#         k = k + 1
    tri_fusion_rec2D(tab,0,nb_ligne-1,0)
    trouver = False
    j = 0
    temps = numpy.zeros((len(tab),3), dtype=int)
    cpt = cpt + 4
    while i < len(tab) and not trouver:
        cpt = cpt + 3
        if tab[i][0] <= 12:
            i = i + 1
            cpt = cpt + 3
        else:
            trouver=True
            cpt = cpt + 2
    while j < len(tab)-1:
        cpt = cpt + 4
        if tab[j][0]==tab[j+1][0]:
            cpt = cpt + 2
            if tab[j][1]<tab[j+1][1]:

                temps[j][0]=tab[j][0]
                tab[j][0]=tab[j+1][0]
                tab[j+1][0]=temps[j][0]

                temps[j][1]=tab[j][1]
                tab[j][1]=tab[j+1][1]
                temps[j][0]=tab[j][0]
                tab[j][0]=tab[j+1][0]
                tab[j+1][0]=temps[j][0]

                temps[j][1]=tab[j][1]
                tab[j][1]=tab[j+1][1]
                tab[j+1][1]=temps[j][1]

                temps[j][2]=tab[j][2]
                tab[j][2]=tab[j+1][2]
                tab[j+1][2]=temps[j][2]
                cpt = cpt + 17
        j = j + 1

    tri_fusion_rec2D(tab,i,nb_ligne-1,1)
    inversions_a_partir_de(tab,i)

    while i < len(tab)-1:
        cpt = cpt + 4
        if tab[i][1]==tab[i+1][1]:
            cpt = cpt + 2
            if tab[i][2]>tab[i+1][2]:

                temps[i][0]=tab[i][0]
                tab[i][0]=tab[i+1][0]
                tab[i+1][0]=temps[i][0]

                temps[i][1]=tab[i][1]
                tab[i][1]=tab[i+1][1]
                tab[i+1][1]=temps[i][1]

                temps[i][2]=tab[i][2]
                tab[i][2]=tab[i+1][2]
                tab[i+1][2]=temps[i][2]
                cpt = cpt + 17
#----------pour afficher le tableau trier----------#
#     k = 0
#     print("")
#     print("après")
#     while k< nb_ligne:
#         print("age:",tab[k][0])
#         print("prix:",tab[k][1])
#         print("n° initial:",tab[k][2])
#         k = k + 1
        i = i + 1
```

D. Vols avec retard

Nous avons fait 4 fonctions pour séparer tout cette fonction :

- « securiter » qui permet de faire des vols tout les 5 min et les décale s'ils sont trop près
- « gestion_retard » qui permet d'ajouter les retards à l'heure de départ
- « couvre_feu » permet seulement d'annuler les vols qui ne sont pas dans la bonne plage horaire
- « affichage retard » permet seulement d'afficher les vols avec soit leurs retard soit leur annulations soit le fait qu'ils soit a leurs

```python
def securiter(tabheur : [int], tabretard : [int]):
    global cpt
    i = 0
    cpt = cpt + 1
    while i<len(tabheur)-1:
        cpt = cpt + 4
        if tabheur[i]%100<5:
            cpt = cpt + 2
            if tabheur[i]-45 > tabheur[i+1]:
                cpt = cpt + 11
                dif=tabheur[i]-45-tabheur[i+1]
                tabheur[i+1]=tabheur[i+1]+dif
                if tabretard[i+1] != -1:
                    cpt = cpt + 6
                    tabretard[i+1]=tabretard[i+1]+dif
        else:
            if tabheur[i]-5 > tabheur[i+1]:
                cpt = cpt + 11
                dif=tabheur[i]-5-tabheur[i+1]
                tabheur[i+1]=tabheur[i+1]+dif
                if tabretard[i+1] != -1:
                    cpt = cpt + 6
                    tabretard[i+1]=tabretard[i+1]+dif
        i = i + 1
```

Mathieu Vellozzi
Ilyas El Habachi
Mathieu Corne

```python
def gestion_retard(tabheur : [int], tabretarde : [int]):
    global cpt
    i = 0
    cpt = cpt + 1
    while i<len(tabheur):
        cpt = cpt + 3
        if tabretarde[i] >= 0:
            cpt = cpt + 1
            if tabheur[i]%100 > 60-tabretarde[i]:
                cpt = cpt + 6
                tabheur[i]=tabheur[i]+40+tabretarde[i]
            else:
                cpt = cpt + 2
                tabheur[i]=tabheur[i]+tabretarde[i]
        i = i + 1
```

```python
def couvre_feu(tabheur : [int], tabretard : [int]):
    global cpt
    i=0
    cpt = cpt + 1
    while i< len(tabheur):
        cpt = cpt + 3
        if tabheur[i]%100>60 and tabheur[i]//100>21 and tabheur[i]//100<6:
            tabretard[i]=-1
            cpt = cpt + 9
        i = i + 1
```

```python
def affichage_retard(tabheur : [int], tabretard : [int]):
    global cpt
    i=0
    cpt = cpt + 1
    while i<len(tabheur):
        cpt = cpt + 3
        if tabretard[i]==-1:
            #print("le vol prevue à : ",tabheur[i]," est annuler")
            cpt = cpt + 1
        else:
            if tabretard[i]==0:
                #print("le vol prevue à : ",tabheur[i]," est a l'heure")
                cpt =cpt + 1
            else :
                #print("le vol prevue à : ",tabheur[i]," est a etait retarde de :",tabretarde[i],"min")
                p=0 #il faut imaginer que le p = 0 n'ai pas la il permet juste d'eviter une erreur car le print est en commentaire pour que les do
        i = i + 1
```

Complexité

Maintenant que on vous a présentait tous nos codes on vas vous dire les quelles nous avons garder et pourquoi

Tris tableaux :

	Tri Selection	Tri Insertion	Tri à Bulle	Tri Rapide	Tri Fusion
Tableau de 10 éléments	447 opérations	509 opérations	583 opérations	154 opérations	872 opérations
Tableau de 100 éléments	27354 opérations	40304 opérations	57042 opérations	1672 opérations	16230 opérations
Tableau de 1000 éléments	2525884 opérations	4022793 opérations	5735451 opérations	16936 opérations	236074 opérations

Comme on peut voir , il y a une véritable différence entre les fonctions récursives et les fonctions interactives quand on dépasse les 100 éléments dans un tableau.

C’est pour cela qu’on a choisi d'implémenter nos fonctions en utilisant le Tri rapide et le Tri Fusion.

Recherche :

	dicotomie	Recherche vol à partir de
Tableau de 10 éléments	32.7	34
Tableau de 100 éléments	57.7	304.4
Tableau de 1000 éléments	74.9	3005.2

Nous avons choisi l’algorithme de « recherche vol à partir de » car même si il est moins efficace le tableau n’a pas besoin d’être trie

Mathieu Vellozzi
Ilyas El Habachi
Mathieu Corne

Algorithme fonctionnalité :

	Affichage vols 3H	Affichage passager trier	Gestion des retard
Tableau de 10 éléments	38.2	1987.1	174.5
Tableau de 100 éléments	336.7	36805.6	1768.6
Tableau de 1000 éléments	3379.6	540023.7	17599.9