UNIVERSITÉ TOULOUSE III PAUL SABATIER
Université de Toulouse
Faculté des Sciences et d'Ingénierie

Méthodologie

Algorithmique, Vérification, Structures de Données

Algorithmique
Structures de Données
Exercices

Semestre 2 - 2019-2020

Motivations et objectifs

- *Connaître le langage Python n'est pas un but mais un moyen*
- A travers des exercices assurer une meilleur maîtrise des notions suivantes :
  - Structures de Données : identification, abstraction, typage et implémentation
  - Utilisation de différentes Structures de données dans le développment d'un programme
  - Utilisation des Stratégies pour construire et améliorer les algorithmes

Problème 1

L'élément majoritaire dans une liste

Soit L une liste de taille n. Un élément x de cette liste est dit majoritaire si x apparaît strictement plus que n/2 fois dans L.

Exemples :

- L=[ ] Il n'y a pas de majoritaire
- L=[4] Le majoritaire est 4
- L=[1,4,1,1,1,4,3,1] Le majoritaire est 1
- L=[1,1,1,2,1,2,1,2,2,2] Il n'y a pas de majoritaire

Problème 1 : L'élément majoritaire

Solution Naïve

Cette fonction pourrait servir pour vérifier les résultats des fonctions demandées.

```
1  def FB(L):
2    n, max=len(L),0
3    for i in L:
4      nboc=0
5      for j in L:
6        if i==j:
7          nboc+=1
8      if max < nboc:
9        max, maj = nboc, i
10   if max>n//2:
11     return maj
12   else:
13     return None
```

Problème 1 :L'élément majoritaire

Solution 1

Dans la solution naïve, nous calculons dans la variable max le nombre d'occurrences, le plus grand. Ensuite nous le comparons à n/2 pour constater le majoritaire s'il existe. Ceci consiste à parcourir la liste L, n^2 fois. (n étant le nombre des éléments de L).

Q1. Pour améliorer cette solution, de point de vu efficacité, on évite de calculer le max chaque fois (glouton). Ecrire la fonction **FBG(L)** qui consiste à faire :

Pour chaque élément de la liste :

- Dés que le nombre de ses occurrences devienne $> n/2$ la fonction retourne cet élément.
- Si tous les éléments sont traités et aucun a donné un nombre d'occurrences $> n/2$, on retourne None.

Solution 2

La solution, proposée dans (Q1.) nous a permis d'améliorer l'efficacité qui varie désormais entre $n/2$ et n^2 fois.

- Si la liste contient un majoritaire et il se trouve au début de la liste, on répète n/2 fois
- s'il n'y a pas de majoritaire, on répète n^2 fois

Q2. Nous proposons d'écrire une fonction **MajL(Linit)** qui évite de refaire le même traitement pour la même occurrence :

- travailler sur une copie de la liste initiale
- retirer lors du comptage toutes les occurrences de l'élément concerné de la liste.
- On recommence jusqu'à que la longueur de la liste soit $<= n/2$ ou le nombre d'occurrence de l'élément en cours est$> n/2$.

Problème 1 : L'élément majoritaire

Solution 2 :Exemples

```
1  Exemple 1: Linit=[2,3,1,1,2,2,1,1], long=8,
2  L=[2,3,1,1,2,2,1,1]
3  candidat = 2 ==> L=[3,1,1,1,1]  nboc (de 2)=3
       <= long//2 et reste 5 > long//2
4  candidat=3 ==> L=[1,1,1,1]  nboc(de 3) =1
       <=long//2  et reste 4 <= long//2
5  pas de majoritaire
6  Exemple 2: Linit=[2,3,1,1,1,3,1,1], long=8,
7  L=  [2,3,1,1,1,3,1,1]
8  candidat = 2 ==> L=[3,1,1,1,3,1,1] nboc (de
       2)=1 <= long//2
9  candidat = 3 ==> L=[1, 1, 1, 1, 1] nboc (de
       3)=2 <= long//2
10 candidat=1 ==> L=[] nboc (de 1) =5>long//2
11 1 est le Majoritaire
```

Idée pour une troisième Solution

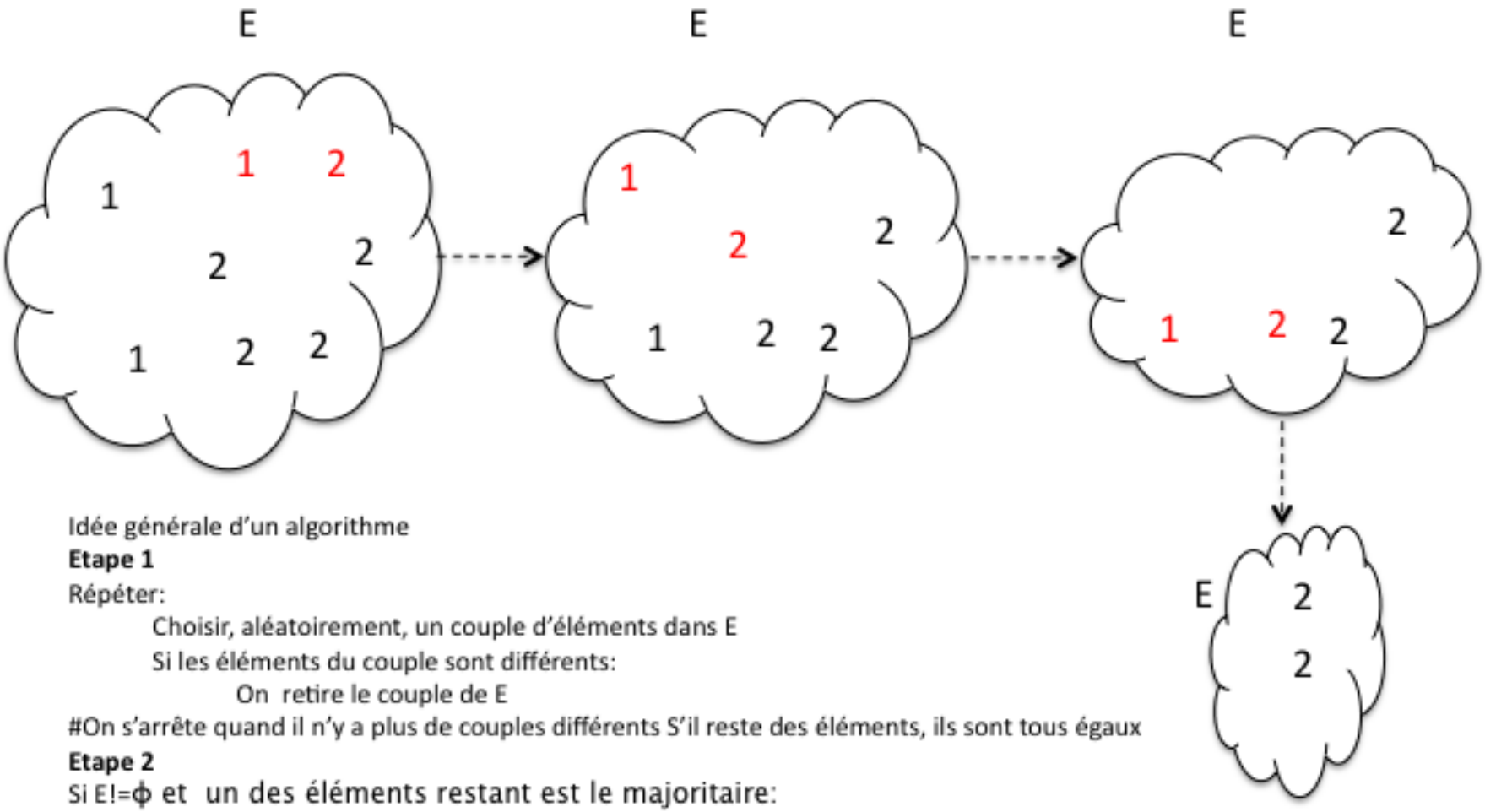
La solution 2 bien qu'elle améliore modestement la précédente, reste non stable car la liste(ce qui reste) est parcouru à nouveau pour chaque nouveau candidat. On répète entre $n/2$ et $(n^2 + 2n)/4$ fois. En suivant une stratégie plus intuitive, qui se fait en 2 étapes :

1. Éliminer dans une collection de données tous les couples composés de deux éléments différents.
2. Ce qui reste après élimination :
   - soit une collection vide, dans ce cas on n'a pas de majoritaire
   - soit une collection d"élément (tous égaux car les différents sont éliminés) ce qui nécessite un parcours de la liste initiale pour savoir s'il s'agit du majoritaire

On peut choisir les couples d'une façon arbitraire (on peut donc considérer, par exemple, à tout instant qu'un couple est constitué par deux éléments voisins de la collection)

Problème 1 : L'élément majoritaire

Solution abstraite 3



Idée générale d'un algorithme

Etape 1

Répéter:

Choisir, aléatoirement, un couple d'éléments dans E

Si les éléments du couple sont différents:

On retire le couple de E

#On s'arrête quand il n'y a plus de couples différents S'il reste des éléments, ils sont tous égaux

Etape 2

Si E!=ϕ et un des éléments restant est le majoritaire:

Retourner cet élément

Sinon:

Retourner None

Solution 3 : Différentes Mise en oeuvre

Dans les différentes mises en oeuvres qu'on souhaite développer, nous avons besoin, en deuxième étape, si on a un candidat de vérifier s'il est le majoritaire.

Q3. Ecrire la fonction **EstMaj(x, Linit)** qui retourne True si x est le majoritaire sinon elle retourne Fals// Les différentes mises en oeuvres concernant l'étape 1 :

- choisir un couple d'éléments quelconques :
  - si les deux éléments sont différents on les élimine
  - Dans le cas contraire, le choix étant arbitraire, on risque de retomber un nombre indéterminé de fois sur ce même couple. D'où il est nécessaire de proposer un choix efficace efficace.
- une solution utilisant des structures comme les piles (2 piles) ou les files (2 files) en plus que la liste initiale
- une solution avec une liste copie de la liste initiale
- une solution utilisant des compteurs

Problème 1 : L'élément majoritaire

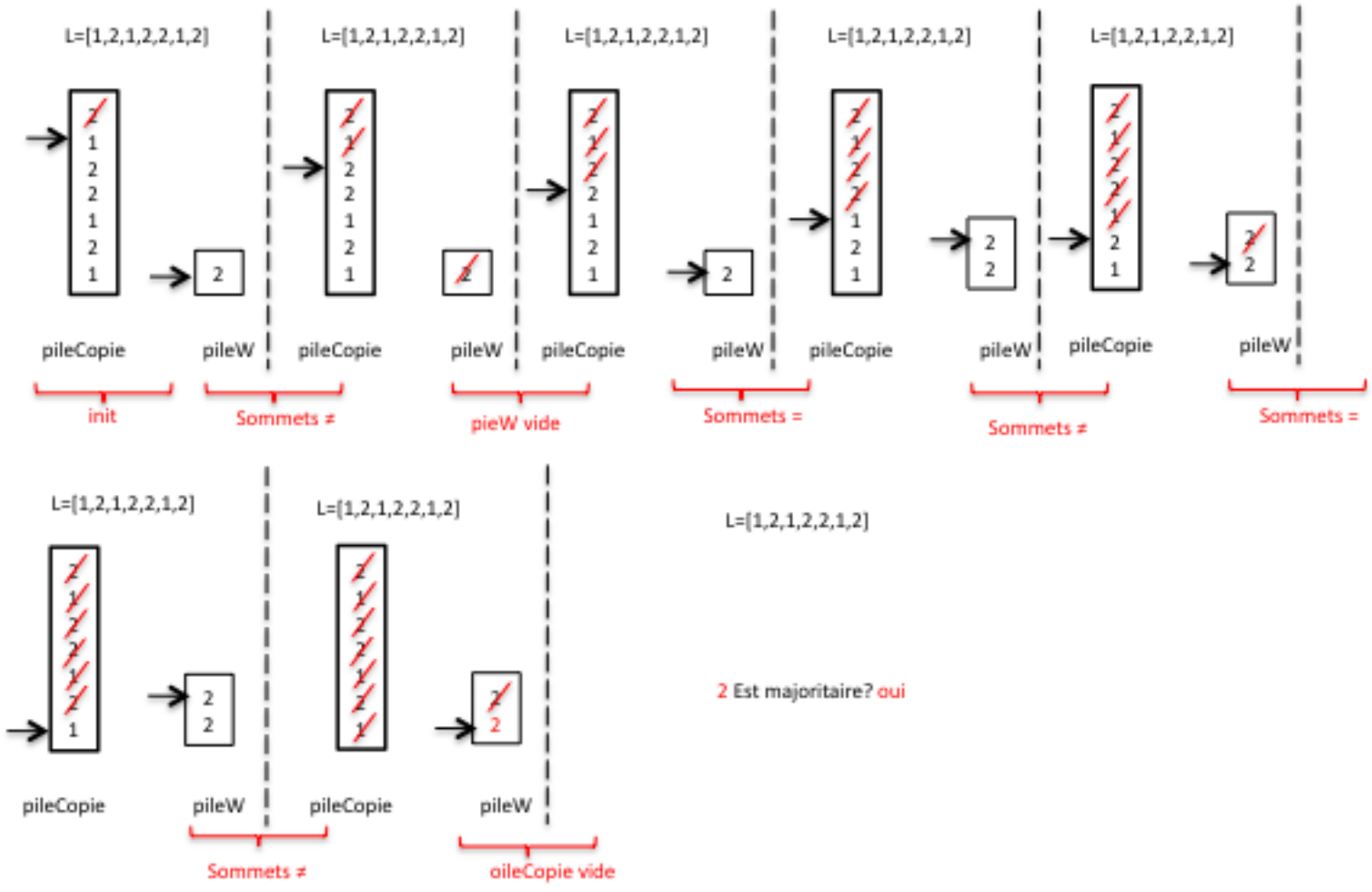
Solution 3 avec 2 piles

- Empiler dans une pile copiePile les éléments de la liste initiale
- créer une pile de travail pileW
- le coupl d'éléments est composé des deux têtes des deux piles
- on répète jusqu'à que la copiPile soit vide
  - pileW est vide on empile la tête de copiePile dans pileW
  - les têtes des deux piles sont égaux, alors on empile la tête de copiePile dans pileW
  - les deux têtes sont différentes, on dépiles les deux tête

  Nous pouvons faire d'une façon similaire la même démarche avec des files
  def EstMaj(x,Linit) :
- **Q4** Écrire une fonction à deux piles permettant de désigner le **candidat**

Problème 1 : L'élément majoritaire

Solution 3 avec 2 piles



Solution 3 : Une liste copie

Les solutions avec piles ou files assurent bien une implémentaition efficace mais à chaque fois nous avons uttilsé deux structures (exemple ci-dessus deux piles) L'utilisation d'une deuxième pile a permis de préserver des éléments en double. Ceci pourrait se faire d'une façon similaire avec une seule liste. Nous proposons d'écrire une fonction, **def MajLOpt(Linit) :**

Cas particuliers

- Linit est vide on retourne None

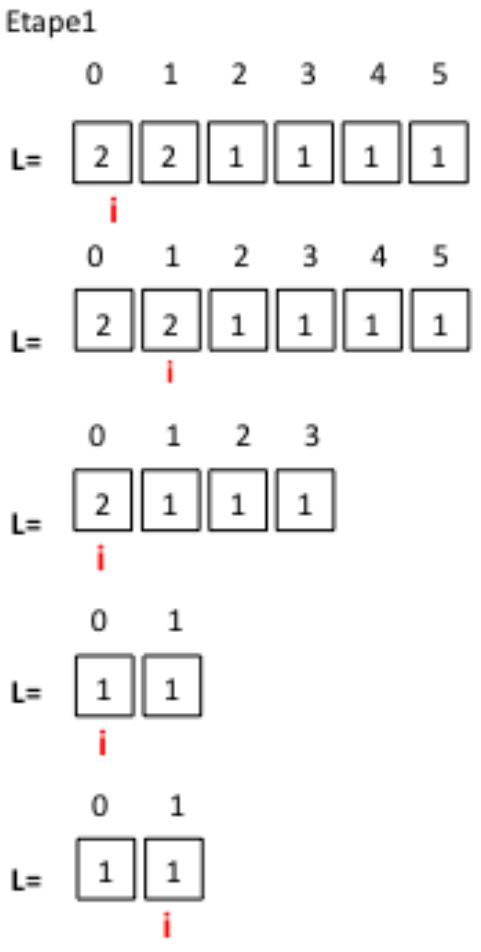
Cas général, en deux étapes :

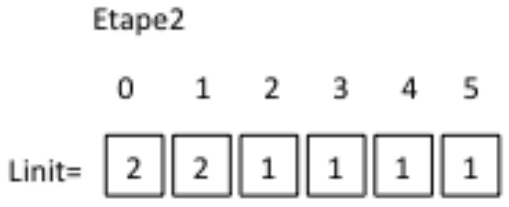
1. Trouver, en un seul parcours de la liste, le seul élément candidat à être majoritaire dénoté **candidat**. Ceci peut se faire en éliminant tous les couples d'éléments différents de la liste.
2. Vérifier en un seul parcours de la liste si **candidat** est le majoritaire

L'élément majoritaire :Solution 3

Eliminer des éléments conduit à travailler sur une **Copie L de Linit**

Un choix aléatoire de couple ne permet pas d'optimiser
On considère les couples :
(L[i],L[i+1])





Candidat: L[0) est majoritaire dans Linit

L'élément majoritaire :Solution 3
Démarche en 2 étapes utilisant une liste

1. Trouver **candidat**
   - **On répète tant que L contient des éléments différents :**
     - un couple d'éléments égaux : on avance d'un cran dans **L**.
     - si les éléments du couple sont différents, : on élimine les deux éléments de **L**.
   - Quand on s'arrête, soit **L** est vide, soit tous ses éléments sont égaux ce qui permettra de désigner **L[0]** comme **candidat**
     - **L** est vide : on retourne **None**
     - **L** n'est pas vide,on retourne **L[0]** s'il est majoritaire dans **Linit**, sinon on retourne **None**

- **Q5.** Écrire une fonction, **def MajLOpt(Linit) :**, qui en seul parcours, si la liste n'est pas vide désigne, le **candidat**.

Problème 2

Chemin le plus court. Modélisation des données

Il s'agit de trouver le chemin le plus court dans une image.

- Une image pourrait se présenter par une **matrice carré**.
- Nous supposons que les données en entrées se trouvent dans une liste de **L** de n^2 éléments ce qui permettra de construire une matrice carré avec n lignes et n colonnes.
- Les points significatifs sont représentés par des valeurs égales à 1 et ceux qui ne sont pas significatifs par O.
- Un chemin entre deux points est une succession de voisins égaux à 1.

Problème 2

Chemin le plus court. . Modélisation des données

- Un élément dans la matrice est repéré par ses coordonnées (Ligne, Colonne)
- Un élément a au plus 4 voisins (sur les bords de la matrice on a 2 ou trois voisins. Exemple d'un élément à 4 voisins M[i][j] a comme voisins : M[i-1][j], M[i+1][j], M[i][j-1], M[i][j+1]
- Un élément vaut 0 est un élément non accessible, sinon l'élément vaut 1 et il est accessible.

Problème 2

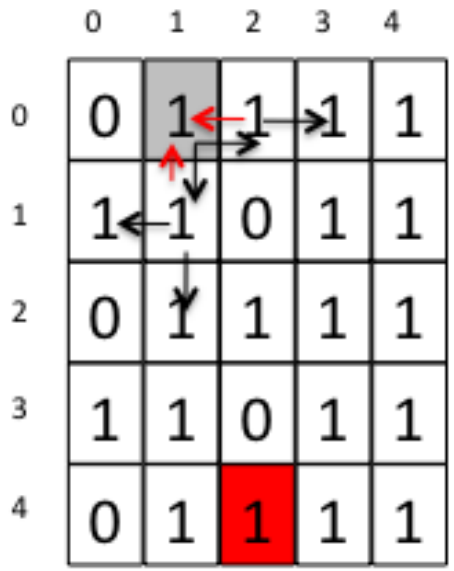
Chemin le plus court. Parcours en largeur

Pour optimiser la recherche, nous effectuons un parcours en en largeur en partant de point de départ qu'on désigne **sd**. Ceci consiste à visiter les voisins immédiats accessibles du point courant. On répète cette démarche jusqu'à rencontrer le point arrivé désigné par **sf**. Si nous n'arrivons pas à avancer à partir des voisins, nous pouvons constater l'inexistence de tel chemin.

Chemin le plus court : départ

L=[0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1

Sd=(0,1)



Sf=(4,2)

Point de départ sd(0,1) dans la matrice

Visitez les voisins immédiats accessibles

Pas 1: les voisins visités sont:

(1,1) et (0,2)
(O,O) n'est pas accessible non visité

On répète, la même chose,
dans l'ordre pour chaque voisin

De (1,1) on visite (1,0) , (2,1) et (0,1)
(1,2) n'est pas accessible non visité

De (0,2) on visite (0,3) et (0,1)
(1,2) n'est pas accessible non visité
:::::::::

Jusqu'à rencontrer (4,2) ou impossible d'avancer

Remarques:
- Pas,de problème d'ordre entre les voisins de même niveau
- Problème 1: assurer l'ordre entre les voisin de niveau différents
(niveau i est traité avant le niveau i+1
- Problème2: repasser par des points déjà visités conduit à un cycle

Problème 2

Chemin le plus court. Enrichir le modèle

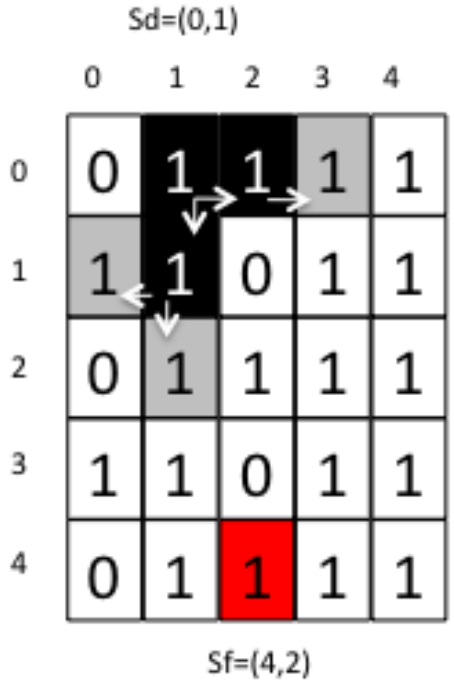
Dans ette recherche d'un chemin nous soulévons les problèmes suivants :

- Comment visiter d'abord les voisins immédiats ? Chaque voisin visité pour la première fois est inséré dans une file FIFO.
- comment éviter de visiter un voisin déjà visité ? Nos devons associer une information à tout élément pour savoir s'i s'agit de la première visite
- une fois on arrive à **sf**, comment retrouver le chemin parcouru depuis **sd** ? Nous devons associer à chaque élément les coordonnées du voisin de là où on vient.

Problème 2

Chemin le plus court :voisins et couleurs :

L=[0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1



Solution:

Problème 1: assurer l'ordre entre les voisin de niveau différents (niveau i est traité avant le niveau i+1

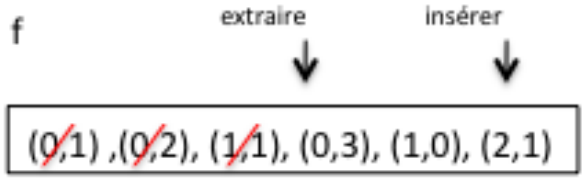
A la première visite, le voisin est inséré dans une file

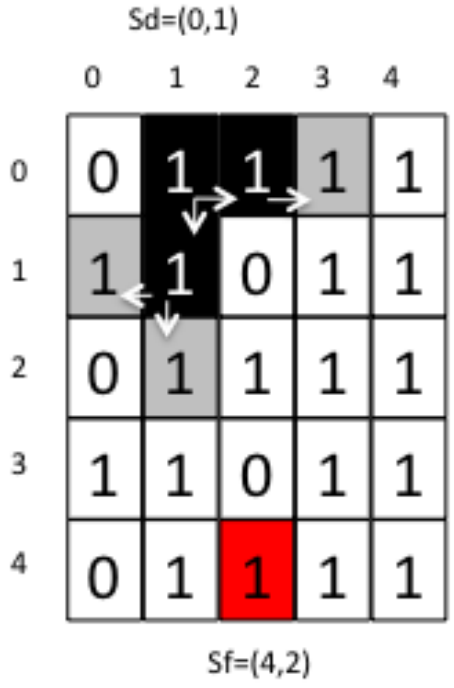
Problème2: repasser par des points déjà visités conduit à un cycle
Colorier toutes les cases en blan au départ
Donner la couleur gris à la case à la première visite
Colorier une case en noir une fois tous ses voisins immédiats sont en couleur gris

Problème 2

Chemin le plus court : voisins et File :

L=[0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1





Solution:

Problème 1: assurer l'ordre entre les voisin de niveau différents (niveau i est traité avant le niveau i+1

A la première visite, le voisin est inséré dans une file

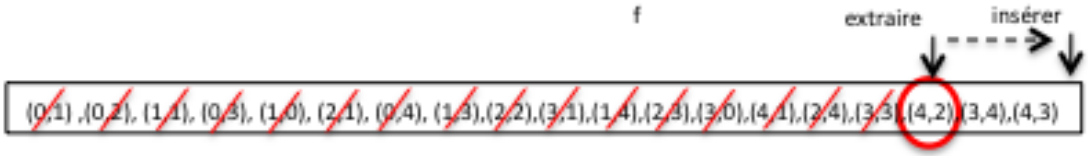
Problème2: repasser par des points déjà visités conduit à un cycle
Colorier toutes les cases en blanc au départ
Donner la couleur gris à la case à la première visite
Colorier une case en noir une fois tous ses voisins immédiats sont en couleur gris

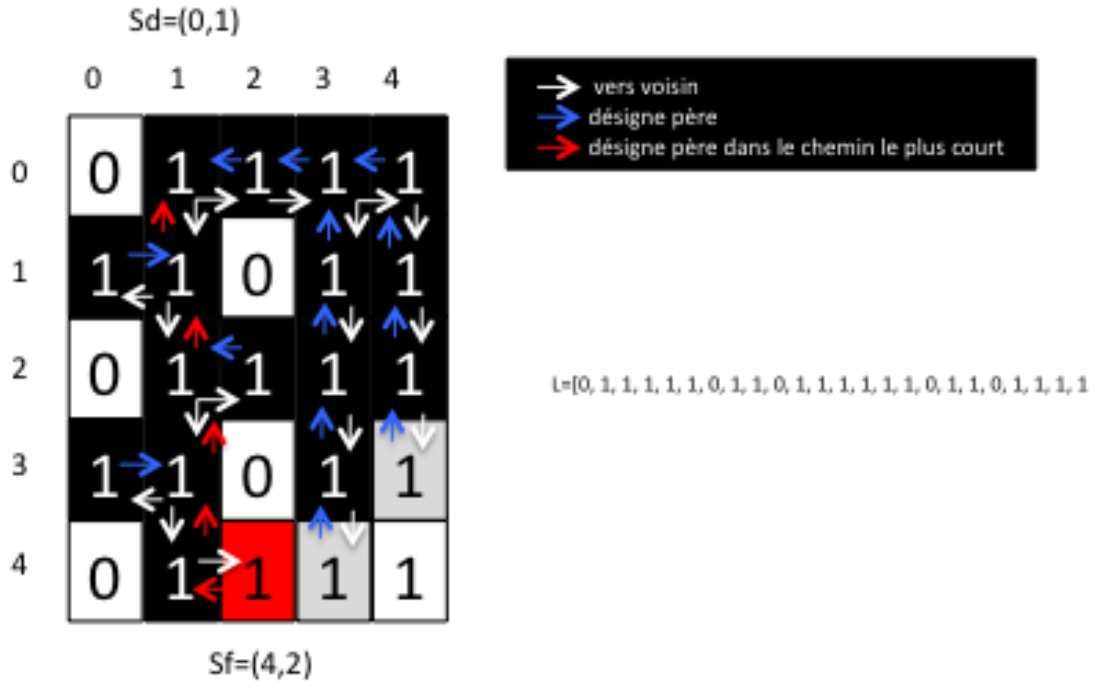
Problème 2

Chemin le plus court. Enrichir le modèle

- un champ couleur(pour éviter les cycles)
  - 'b' : élément non visité
  - 'g' : élément visité pour la première fois
  - 'n' : il est 'g' et tous ses voisins sont visités
- un champ père permettant de reconstruire le chemin trouvé au départ initialisé à (-1,-1)
- Un voisin est accessible par un élément courant, si sa couleur est blanche (n'a pas été visité) et s'il est égal à 1. Son père devient l'élément courant et sa couleur devient grise.
- quand tous les voisins d'un élément sont visités sa couleur devient noir.

Chemin le plus court : Parcours en Largeur





Problème 2

Chemin le plus court

Avant de construire la matrice, on va enrichir chaque information de **L** en produisant une nouvelle liste :

- tuple, pour désigner un point par ses coordonnées : (i,j)
- dictionnaire, pour enrichir un un élément, L[i] devient : LChamps[i]={ "val" :L[i], "coul" :'b', "pere" :(-1,-1) }
- On dispose des structures de données, Matrice et file

Enrichir les éléments

Q1. Transformer une liste L en liste Champs.

```
1
2
3  Exemple :
4
5  L=[0, 1, 1, 1, 1, 1, 1, 0, 1]
6
7  LChamps=
8  [{'val': 0,'coul': 'b', 'pere': (-1, -1)},
9  {'val': 1, 'coul': 'b', 'pere': (-1, -1)},
10 {'val': 1, 'coul': 'b', 'pere': (-1, -1)},
11 {'val': 1, 'coul': 'b', 'pere': (-1, -1)},
12 {'val': 1, 'coul': 'b', 'pere': (-1, -1)},
13 {'val': 1, 'coul': 'b', 'pere': (-1, -1)},
14 {'val': 1, 'coul': 'b', 'pere': (-1, -1)},
15 {'val': 0, 'coul': 'b', 'pere': (-1, -1)},
16 {'val': 1, 'coul': 'b', 'pere': (-1, -1)}]
```

2ème niveau Créer la matrice

Q2. Créer une matrice MG carré à partir de LChamps

```
MG=
[
 [
  {'val': 0, 'coul': 'b', 'pere': (-1, -1)} ,
  {'val': 1, 'coul': 'b', 'pere': (-1, -1)} ,
  {'val': 1, 'coul': 'b', 'pere': (-1, -1)} ,
 ],
 [
  {'val': 1, 'coul': 'b', 'pere': (-1, -1)} ,
  {'val': 1, 'coul': 'b', 'pere': (-1, -1)} ,
  {'val': 1, 'coul': 'b', 'pere': (-1, -1)} ,
 ],
 [
  {'val': 1, 'coul': 'b', 'pere': (-1, -1)} ,
  {'val': 0, 'coul': 'b', 'pere': (-1, -1)} ,
  {'val': 1, 'coul': 'b', 'pere': (-1, -1)} ,
```

Problème 2

Chemin le plus court

Pour écrire la fonction qui qui cherche le chemin le plus cours entre deux points en parcours en Largeur, nous avons beson en entrée :

- **MG** : La matrice créée à partir de LChamps
- **sd** et **sf** sont les tuples représentants les points de départ et d'arrivé.

Algorithme :

- on initialise une file avec le point de départ, qui devient gris.
- on répète, tant qu'on a pas rencontré le point d'arrivée, et il reste des points gris,
  - extraire un point de la file
  - traiter tous ses voisins
  - colorier ce point en noir

Problème 2

Chemin le plus court :Point de

Q3.a Écrire une fonction, qui pour un point traite tous ses voisins :
def TraiterVoisins(f,MG,s) :

- insertion dans la file
- le point devient gris
- Mise à jour de son champ père

Q3.b Ecrire une fonction, retourne la liste qui représente le chemin le plus court. S'il n'existe pas on retourne une liste vide.
def PCourtChemin(MG,sd,sf) :

Files

```
1  # Les files
2  def creer_file_vide():
3    return []
4  def file_vide(une_file) :
5    return nombre_elements(une_file) == 0
6  def nombre_elements(une_file):
7    return len(une_file)
8  # inserer  en fin de la file
9  def inserer(une_file, une_valeur):
10   une_file.append(une_valeur)
11 # extraire: supprime et retourne valeur en tete
12 def extraire(une_file):
13   assert not file_vide(une_file), "extraire␣
       interdit␣sur␣une␣file␣vide"
14   valeur_en_tete = une_file[0]
15   del une_file[0]
16   return valeur_en_tete
```

Matrice

```
1  import math
2  #Les Matriices
3  def creer_matrice_vide():
4    return []
5
6  def initialiser_matrice(L, M):
7    assert int(math.sqrt(len(L)))==
       math.sqrt(len(L), "erreur taille "
8    n=int(math.sqrt(len(L)))
9    for i in range(n):
10     x=[]
11     for j in range(n):
12       x.append(L[i*n+j])
13     M.append(x)
```

Matrice

```
1 def affiche_matrice_carre(M):
2   n=len(M)
3   print("[")
4   for i in range (n):
5     print(M[i],",")
6   print("]")
7
8 def matrice_dim(M):
9   return len(M)
```