Université de Caen Normandie, UFR des Sciences
Licence 3

Module "Protection de données"
TP - algorithmes probabilistes ; durée : 2h30

Le TP a pour objectif d'implémenter plusieurs algorithmes probabilistes.

1 Estimation de π

Nous avons vu en cours et en TD un algorithme probabiliste pour estimer le nombre π. Cet algorithme se base sur l'expérience aléatoire suivante :

Tirer uniformément deux réel X et Y dans l'intervalle $[-1, 1]$, poser Z la variable aléatoire qui vaut 1 si $X^2 + Y^2 \leq 1$, 0 sinon.

La variable aléatoire Z suit une loi de Bernoulli de paramètre $\pi/4$. Son espérance et sa variance sont

$$\mathbb{E}[Z] = \frac{\pi}{4}, \qquad \mathbb{V}(Z) = \frac{\pi}{4}\left(1 - \frac{\pi}{4}\right).$$

Si $Z_1, \ldots, Z_n$ sont n réalisations indépendantes de l'expérience précédente, alors la loi des grands nombres dit que $T_n = (Z_1 + \cdots + Z_n)/n$ converge presque sûrement vers $\pi/4$. Les bornes de Chernoff nous donnent en plus

$$\forall \epsilon > 0, \qquad \mathbb{P}\left(\left|T_n - \frac{\pi}{4}\right| > \epsilon\right) \leq 2e^{-\epsilon^2 n}.$$

Question 1 : Ecrire une fonction `get_n(epsilon,certitude)` qui retourne le plus petit n tel que

$$2e^{-\epsilon^2 n} < 1 - \texttt{certitude}.$$

Remarque : avec un tel `n`, l'estimateur T_n sera une valeur approchée de $\pi/4$ à ϵ près avec une probabilité supérieure à `certitude`.

Question 2 : Ecrire une fonction `estime_pisur4(epsilon,certitude)` qui retourne une estimation de $\pi/4$ à ϵ près avec une probabilité supérieure à `certitude`.

2 Algorithmes de sélection de la médiane

Soit un ensemble S de n entiers présenté sous forme d'un tableau $S[1..n]$ *non trié* dont les n éléments (entiers) sont **supposés tous distincts pour simplifier**. On appelle *médiane* de S l'élément m de S tel que le nombre d'éléments de S qui sont strictement plus grands que m est exactement $\lfloor n/2 \rfloor$ (où $\lfloor x \rfloor$ désigne la partie entière de x) ; autrement dit, m est le k-ième élément de S si $n = 2k$ (ce qu'on notera $rang_S(m) = k$) et le $k+1$-ième si $n = 2k+1$. Cela revient aussi à dire que $rang_S(m) = \lceil \frac{n}{2} \rceil$ (où $\lceil x \rceil$ désigne le plus petit entier supérieur ou égal à x).

Cet exercice vise à comparer 3 algorithmes de sélection de la médiane

1. un algorithme basé sur le tri de python,
2. un algorithme l'algorithme QuickSelect qui s'inspire fortement de QuickSort.
3. l'algorithme probabiliste vu en cours/TD

Sélection de la médiane avec un tri

L'approche la plus simple pour sélectionner la médiane est de trier le tableau est d'aller chercher directement la médiane.

Question 3 : Ecrire une fonction `select_mediane_tri(S)` qui retourne la médiane en triant `S`. Attention, `S` ne doit pas être modifié (utilisez `sorted`).

Entrées: Un ensemble S constitué de n éléments
Sorties: la médiane de S ou ECHEC

1. Soit R un ensemble de $\lfloor n^{3/4} \rfloor$ éléments de S tirés aléatoirement et uniformément avec remise (possibilité d'avoir des doublons)
2. Trier R et trouver a et b tels que

$$rang_R(a) = \max\left(1, \lfloor \frac{n^{3/4}}{2} - \sqrt{n} \rfloor\right), \qquad rang_R(b) = \min\left(\lfloor \frac{n^{3/4}}{2} + \sqrt{n} \rfloor, \lfloor n^{3/4} \rfloor\right)$$

où $rang_R(a) = t$ si a est le t-ième élément le plus petit de R.
3. En comparant chaque élément de S avec a et b, calculer en un seul parcours de S :

$$rang_S(a), \quad rang_S(b), \quad P = \{x \in S,\, a \leq x \leq b\} \quad \text{et} \quad \text{Card(P)}.$$

4. Si $rang_S(a) > \lceil \frac{n}{2} \rceil$ **retourner** ECHEC
5. Si $rang_S(b) < \lceil \frac{n}{2} \rceil$ **retourner** ECHEC
6. Si $\text{Card}(P) \geq 4n^{3/4}$ **retourner** ECHEC
7. Trier P
8. **retourner** c tel que $rang_P(c) = \lceil \frac{n}{2} \rceil - rang_S(a) + 1$

Sélection de la médiane avec la procédure du cours

Nous redonnons l'algorithme vu en cours/TD (appelé aussi procédure) pour le calcul de la médiane.
Question 4 : Ecrire une fonction `select_mediane_procedure(S)` qui met en oeuvre la procédure ci-dessus et retourne la médiane ou ECHEC. Attention, `S` ne doit pas être modifié.

Question 5 : Ecrire une fonction `select_mediane_vegas(S)` qui s'appuie sur la procédure précédente mais qui retourne toujours la médiane. Attention, `S` ne doit pas être modifié.

Sélection de la médiane avec Quickselect

L'algorithme QuickSelect ressemble très fortement à l'algorithme QuickSort. Son principe est le même. QuickSelect sélectionne un pivot et sépare le tableau en deux sous-tableaux qui dépendent du pivot. Le sous tableau gauche (resp. droite) contient les éléments plus petits (resp. plus grands) que le pivot. Après cette phase de partition, le pivot est à sa position finale c'est à dire son rang dans la liste. Trois cas peuvent alors se produire :

— soit le pivot est en position $\lceil n/2 \rceil$ et le pivot est alors la médiane ;
— soit le pivot est en position $< \lceil n/2 \rceil$ et la médiane se trouve dans le sous-tableau droite ;
— soit le pivot est en position $> \lceil n/2 \rceil$ et la médiane se trouve dans le sous-tableau gauche ;

Dans les deux derniers cas, on applique récursivement la procédure au sous tableau gauche ou droite. Voici le pseudo-code pour la fonction de partition et la fonction QuickSelect.

```
fonction partition(tableau S[1..n], entier gauche, entier droite, entier pivot)
    Val_Pivot := S[pivot]
    échanger S[pivot] et S[droite]  // le pivot est placé à la fin
    pos_insertion := gauche
    pour i de gauche à droite-1
        si S[i] < Val_Pivot
            échanger S[pos_insertion] et S[i]
            incrémenter pos_insertion
    échanger S[droite] et S[pos_insertion]  //  le pivot est à sa place finale
    retourner pos_insertion

Quickselect(tableau S[1..n], entier gauche, entier droite)
si gauche = droite alors retourner S[gauche]
pivot := gauche
    pivot := partition(S, gauche, droite, pivot)
```

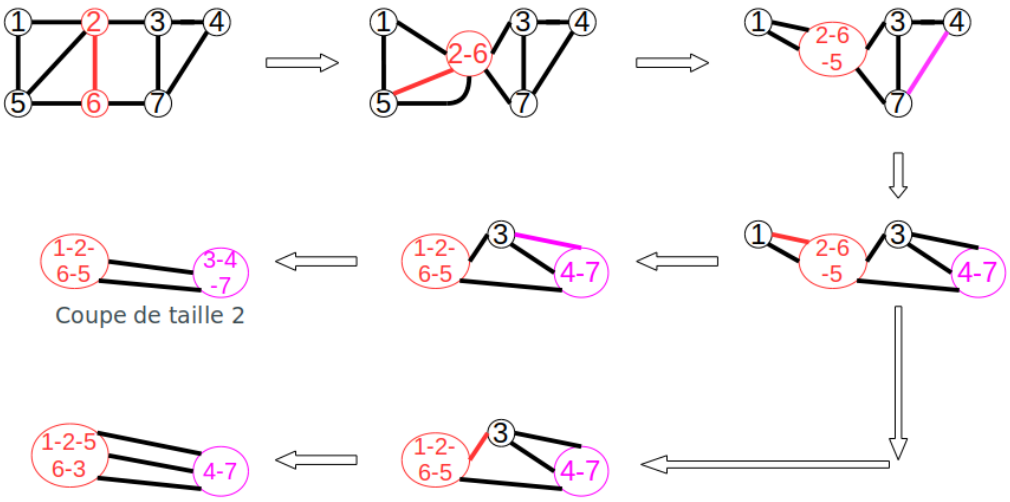
```
si pivot = ceil(n/2) alors retourner S[pivot]
si pivot < ceil(n/2) alors retourner Quickselect(S,pivot+1,droite)
                     sinon retourner Quickselect(S,gauche,pivot-1)
```

Question 6 : L'algorithme QuickSelect ci-dessus est une version récursive. Implémentez une version itérative de l'algorithme QuickSelect, `QuickSelect(S)`, qui ne prend que le tableau S comme seul argument. Attention, S ne doit pas être modifié. Vous pourrez utiliser des fonctions annexes comme la fonction partition ou d'autres.
Question 7 : Implémentez un algorithme `compare_algos(n,k)` qui calcule le temps moyen des 3 algorithmes de sélection de la médiane implémentés. Le temps moyen sera calculé en exécutant k fois chaque algorithme sur des tableaux aléatoires de taille n contenant des entiers entre 1 et n (sans répétition). Le résultat doit retourner un triplet de réels représentant dans l'ordre le temps moyen de `select_mediane_tri`, `select_mediane_vegas` et `QuickSelect`.

3 Algorithme de Karger pour la coupe minimale

L'algorithme de Karger est un algorithme probabiliste de type Monte-Carlo qui retourne une coupe dont on espère qu'elle est minimale. Cet algorithme, vu en cours, est basé sur le principe de contraction d'une arête. A chaque étape, l'algorithme contracte une arête au hasard et fusionne ainsi les deux extrémités. Ainsi, le nombre de sommets décroît de 1 à chaque étape. Lorsqu'il ne reste que deux sommets, on obtient une coupe dont on espère quelle est minimale et et la taille de la coupe est le nombre d'arêtes entre les deux sommets restants.

L'illustration ci-dessous, issue du cours, montre deux exécutions possibles de l'algorithme de Karger qui conduisent à une coupe minimale de taille 2 et à une coupe non minimale de taille 3. A chaque fois, l'arête contractée est mise en couleur.



Après chaque étape, les arêtes entre deux sommets peuvent être multiples. On parle alors de multigraphes. Dans notre cas, les multigraphes sont non orientés (les arêtes sont bidirectionnelles) et sans boucle (pas d'arête partant d'un sommet et arrivant au même sommet). Pour représenter un multigraphe non orienté (avec ou sans boucle), nous utiliserons la classe `Multigraphe` qui est composée de 5 champs publics :

- Le champs `n` correspond au nombre total de sommets du multigraphe ;
- Le champs `m` correspond au nombre total d'arêtes du multigraphe ;
- Le champs `labels` est un tableau de chaînes de caractères qui liste tous les labels de tous les sommets. L'élément `labels[i]` contiendra le i-ème label du graphe (voir exemple ci-dessous) ;
- le champs `adjacence` est un tableau bidimensionnel à `n` lignes avec `n` le nombre de sommets. La i-ème ligne est un tableau de i cases et pour $j \leq i$, l'élément `adjacence[i][j]` est un entier correspondant au nombre d'arêtes entre les sommets `labels[i]` et `labels[j]`. Dans notre cas, les multigraphes n'ayant pas de boucle, on aura pour $i = 1 \dots n$, `adjacence[i][i]=0` ;
- le champs `total` est un tableau tel que `total[i]`$=\sum_{j \leq i}$`adjacence[i][j]`.

Le champs `total` servira à choisir aléatoirement une arête parmi celle restantes. Une représentation des multigraphes de l'illustration ci-dessus qui conduisent à la coupe minimale de taille 2 est donnée à la fin du sujet.

Question 8 : Ecrire une fonction `random_multi(m,n)` qui retourne un multigraphe aléatoire à n sommets et m arêtes. Pour cela, il suffit de répéter l'évènement suivant à m reprises : choisir une arête au hasard et l'ajouter au multigraphe (une arête peut être choisie à plusieurs reprises puisque nous considérons des multigraphes).

Ecrire également une fonction `illustration_multi()` qui retourne le premier multigraphe de l'illustration ci-dessus.

Remarque : ces fonctions pourront être utilisées pour tester vos fonctions.

Question 9 : Ecrire une fonction `contraction(multi,i,j)` qui prend en entrée un multigraphe `multi` et qui contracte l'arête entre les sommets étiquetés par `labels[i]` et `labels[j]`. Le multigraphe sera directement modifié (inutile d'en faire une copie). Lors de la concaténation des labels, l'ordre du tableau `labels` est respecté et le nouveau label prend la place du premier label (le deuxième label utilisé disparaissant, cf. exemple ci-dessous). Par exemple si $i < j$, `labels[i]` est remplacé par `labels[i]+"-"+labels[j]` et `labels[j]` est ensuite supprimé de la liste des labels.

Remarque : avant d'implémenter cette fonction, faites un schéma des changement à faire dans la matrice d'adjacence.

Question 10 : Ecrire une fonction `random_arete(multi)` qui prend en entrée un multigraphe `multi` à n sommets et qui retourne un couple aléatoire (i, j) avec $0 \leq j \leq i < n$ (le couple est de type python tuple). La probabilité du couple (i, j) sera proportionnelle à `adjacence[i][j]`. Vous pourrez utiliser le champs `total` de la classe multigraphe pour éviter de parcourir toutes les lignes de la matrice d'adjacence. La fonction retourne $(i, j) = (-1, -1)$ s'il n'y a pas d'arête.

Question 11 : Ecrire une fonction `karger(multi)` qui applique l'algorithme de Karger sur le multigraphe `multi` et qui retourne la taille de la coupe trouvée. Si le processus se passe bien, à la fin de la fonction, le multigraphe `multi` doit être composé de 2 sommets.

Remarque 1 : si le multigraphe en entrée contient plus de 3 composantes connexes, l'algorithme de Karger se termine avant qu'il ne reste deux sommets. Dans ce cas, le multigraphe obtenu à la fin contient autant de sommets que de composantes connexes et aucune arête. La taille de la coupe minimale est 0.

Remarque 2 : ne dupliquez pas le multigraphe multi. Travaillez directement avec.

Question 12 : En cours, nous avons étudié la probabilité que l'algorithme de Karger retourne une coupe de taille minimale. A partir du cours, implémentez une fonction `karger_certitude(multi,certitude)` qui retourne un multigraphe à deux sommets représentant une coupe du multigraphe `multi`. La coupe obtenue sera une coupe minimale avec une probabilité supérieure à `certitude`.

Remarque : si le multigraphe en entrée contient plus de 3 composantes connexes, l'algorithme `karger_certitude`, qui se base sur la fonction karger précédente, retournera un multigraphe qui contient autant de sommets que de composantes connexes et aucune arête. La taille de la coupe minimale sera 0.

Voici un exemple d'exécution de l'algorithme de Karger sur la structure de donnée.

n=7, m=10, labels = | '1' | '2' | '3' | '4' | '5' | '6' | '7' | total = | 0 | 1 | 1 | 1 | 2 | 2 | 3 |

adjacence[0] =	0							(pas d'arête entre '1' et '1')
adjacence[1] =	1	0						(1 arête entre '2' et '1')
adjacence[2] =	0	1	0					(1 arête entre '3' et '2')
adjacence[3] =	0	0	1	0				(1 arête entre '4' et '3')
adjacence[4] =	1	1	0	0	0			(1 arête entre '5' et '1' puis '5' et '2')
adjacence[5] =	0	1	0	0	1	0		(1 arête entre '6' et '2' puis '6' et '5')
adjacence[6] =	0	0	1	1	0	1	0	(1 arête entre '7' et '3','4' et '6')

n=6, m=9, labels = | '1' | '2-6' | '3' | '4' | '5' | '7' | total = | 0 | 1 | 1 | 1 | 3 | 3 |

adjacence[0] = | 0 | (pas d'arête entre '1' et '1')
adjacence[1] = | 1 | 0 | (1 arête entre '2-6' et '1')
adjacence[2] = | 0 | 1 | 0 | (1 arête entre '3' et '2-6')
adjacence[3] = | 0 | 0 | 1 | 0 | (1 arête entre '4' et '3')
adjacence[4] = | 1 | 2 | 0 | 0 | 0 | (1 arête entre '5' et '1' puis 2 arêtes entre '5' et '2-6')
adjacence[5] = | 0 | 1 | 1 | 1 | 0 | 0 | (1 arête entre '7' et '2-6', '3' et '4')

n=5, m=7, labels = | '1' | '2-6-5' | '3' | '4' | '7' | total = | 0 | 2 | 1 | 1 | 3 |
adjacence[0] = | 0 | (pas d'arête entre '1' et '1')
adjacence[1] = | 2 | 0 | (2 arêtes entre '2-6-5' et '1')
adjacence[2] = | 0 | 1 | 0 | (1 arête entre '3' et '2-6-5')
adjacence[3] = | 0 | 0 | 1 | 0 | (1 arête entre '4' et '3')
adjacence[4] = | 0 | 1 | 1 | 1 | 0 | (1 arête entre '7' et '2-6-5', '3' et '4')

n=4, m=6, labels = | '1' | '2-6-5' | '3' | '4-7' | total = | 0 | 2 | 1 | 3 |
adjacence[0] = | 0 | (pas d'arête entre '1' et '1')
adjacence[1] = | 2 | 0 | (2 arêtes entre '2-6-5' et '1')
adjacence[2] = | 0 | 1 | 0 | (1 arête entre '3' et '2-6-5')
adjacence[3] = | 0 | 1 | 2 | 0 | (1 arête entre '4-7' et '2-6-5', 2 arêtes entre '4-7' et '3')

n=3, m=4, labels = | '1-2-6-5' | '3' | '4-7' | total = | 0 | 1 | 3 |
adjacence[0] = | 0 | (pas d'arête entre '1-2-6-5' et '1-2-6-5')
adjacence[1] = | 1 | 0 | (1 arête entre '1-2-6-5' et '3')
adjacence[2] = | 1 | 2 | 0 | (1 arête entre '4-7 et '1-2-6-5' et 2 arêtes entre '4-7' et '3')

n=2, m=2, labels = | '1-2-6-5' | '3-4-7' | total = | 0 | 2 |
adjacence[0] = | 0 | (pas d'arête entre '1-2-6-5' et '1-2-6-5')
adjacence[1] = | 2 | 0 | (2 arêtes entre '1-2-6-5' et '3-4-7')

4 Trace d'exécution

```
********** début des tests **************
****** question 1: get_n ******
( 0.1 , 0.85 , 260 ),( 0.1 , 0.9 , 300 ),( 0.1 , 0.95 , 369 ),( 0.01 , 0.85 , 25903 ),
( 0.01 , 0.9 , 29958 ),( 0.01 , 0.95 , 36889 ),( 0.001 , 0.85 , 2590268 ),
( 0.001 , 0.9 , 2995733 ),( 0.001 , 0.95 , 3688880 ),( 0.0001 , 0.85 , 259026717 ),
( 0.0001 , 0.9 , 299573228 ),( 0.0001 , 0.95 , 368887946 ),
get_n(0.1,0.95)= 369
get_n(0.01,0.95)= 36889
get_n(0.001,0.95)= 3688880
****** question 2: estime_pisur4 ******
pi/4= 0.7853981633974483
estime_pisur4(0.1,0.95)-pi/4= 0.04658015638574953
estime_pisur4(0.01,0.95)-pi/4= 0.002560306607160112
estime_pisur4(0.001,0.95)-pi/4= 0.00015219335039928783
random_tab(10)= [9, 8, 2, 6, 5, 4, 3, 7, 1, 0]
****** question 3: select_mediane_tri ******
select_mediane_tri([3, 1, 2])= 2
[3, 1, 2]
select_mediane_tri([3,1,2,0])= 1
****** question 4: select_mediane_procedure ******
select_mediane_procedure([3])= 3
[3]
```

```
select_mediane_procedure([3,1,2,0])= ECHEC
True
True
****** question 5: select_mediane_vegas ******
select_mediane_vegas([3, 1, 2])= 2
[3, 1, 2]
select_mediane_vegas([3,1,2,0])= 1
True
True
****** question 6: QuickSelect ******
QuickSelect([3, 1, 2])= 2
[3, 1, 2]
QuickSelect([3,1,2,0])= 1
True
True
****** question 7: compare_algos ******
n= 1000    k= 10 : (8.411407470703125e-05, 0.0002266645431518554, 0.0003585338592529297)
n= 10000    k= 10 : (0.0011191844940185546, 0.0016451120376586915, 0.003748154640197754)
n= 100000    k= 10 : (0.015353655815124512, 0.011879754066467286, 0.03637681007385254)
****** question 8: random_multi ******
n=5
m=15
labels=['0', '1', '2', '3', '4']
  adjacence[0]=[0]
  adjacence[1]=[2, 0]
  adjacence[2]=[0, 0, 0]
  adjacence[3]=[1, 0, 0, 0]
  adjacence[4]=[3, 3, 3, 3, 0]
total=[0, 2, 0, 1, 12]
n=7
m=10
labels=['1', '2', '3', '4', '5', '6', '7']
  adjacence[0]=[0]
  adjacence[1]=[1, 0]
  adjacence[2]=[0, 1, 0]
  adjacence[3]=[0, 0, 1, 0]
  adjacence[4]=[1, 1, 0, 0, 0]
  adjacence[5]=[0, 1, 0, 0, 1, 0]
  adjacence[6]=[0, 0, 1, 1, 0, 1, 0]
total=[0, 1, 1, 1, 2, 2, 3]
****** question 9: contraction ******
Multigraphe(n=7, m=10,labels=['1', '2', '3', '4', '5', '6', '7'],
            adjacence=[[0], [1, 0], [0, 1, 0], [0, 0, 1, 0], [1, 1, 0, 0, 0],
                       [0, 1, 0, 0, 1, 0], [0, 0, 1, 1, 0, 1, 0]],
            total=[0, 1, 1, 1, 2, 2, 3])
Multigraphe(n=6, m=9,labels=['1', '2-6', '3', '4', '5', '7'],
            adjacence=[[0], [1, 0], [0, 1, 0], [0, 0, 1, 0], [1, 2, 0, 0, 0],
                        [0, 1, 1, 1, 0, 0]],
            total=[0, 1, 1, 1, 3, 3])
Multigraphe(n=5, m=7,labels=['1', '2-6-5', '3', '4', '7'],
            adjacence=[[0], [2, 0], [0, 1, 0], [0, 0, 1, 0], [0, 1, 1, 1, 0]],
            total=[0, 2, 1, 1, 3])
Multigraphe(n=4, m=6,labels=['1', '2-6-5', '3', '4-7'],
            adjacence=[[0], [2, 0], [0, 1, 0], [0, 1, 2, 0]],
            total=[0, 2, 1, 3])
Multigraphe(n=3, m=4,labels=['1-2-6-5', '3', '4-7'],
```

```
            adjacence=[[0], [1, 0], [1, 2, 0]],
            total=[0, 1, 3])
Multigraphe(n=2, m=2,labels=['1-2-6-5', '3-4-7'],
            adjacence=[[0], [2, 0]],
            total=[0, 2])
****** question 10: random_arete ******
[5, 1] ,[6, 3] ,[4, 1] ,[2, 1] ,[6, 3] ,[1, 0] ,[3, 2] ,[4, 0] ,[1, 0] ,[6, 5] ,[6, 2] ,[2, 1] ,[3, 2]
****** question 11: karger ******
n=2
m=5
labels=['1-5-6-7', '2-3-4']
  adjacence[0]=[0]
  adjacence[1]=[5, 0]
total=[0, 5]
-------
n=2
m=3
labels=['1-5', '2-3-4-6-7']
  adjacence[0]=[0]
  adjacence[1]=[3, 0]
total=[0, 3]
-------
n=2
m=2
labels=['1-2-5-6-7-3', '4']
  adjacence[0]=[0]
  adjacence[1]=[2, 0]
total=[0, 2]
-------
n=2
m=3
labels=['1-2-5', '3-4-6-7']
  adjacence[0]=[0]
  adjacence[1]=[3, 0]
total=[0, 3]
-------
****** question 12: karger_certitude ******
n=2
m=2
labels=['1-2-5-6', '3-4-7']
  adjacence[0]=[0]
  adjacence[1]=[2, 0]
total=[0, 2]
-------
coupe trouvée:
E1= 0-9-14-17-18-3-8-10-6-19-4-5-2-11-12-13-15-1-16
-
E2= 7
taille= 4
-------
Test de Karger avec deux cliques reliées par 3 arêtes
n=10
m=23
labels=['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
  adjacence[0]=[0]
  adjacence[1]=[1, 0]
```

```
  adjacence[2]=[1, 1, 0]
  adjacence[3]=[1, 1, 1, 0]
  adjacence[4]=[1, 1, 1, 1, 0]
  adjacence[5]=[1, 0, 0, 0, 0, 0]
  adjacence[6]=[0, 1, 0, 0, 0, 1, 0]
  adjacence[7]=[0, 0, 1, 0, 0, 1, 1, 0]
  adjacence[8]=[0, 0, 0, 0, 0, 1, 1, 1, 0]
  adjacence[9]=[0, 0, 0, 0, 0, 1, 1, 1, 1, 0]
total=[0, 1, 2, 3, 4, 1, 2, 3, 3, 4]
coupe trouvée:
E1= 0-2-4-1-3
-
E2= 5-9-6-7-8
taille= 3
n=2
m=3
labels=['0-2-4-1-3', '5-9-6-7-8']
  adjacence[0]=[0]
  adjacence[1]=[3, 0]
total=[0, 3]
-------
```