Corrigé : couplage dans un graphe biparti équilibré (d'après Mines 2012)

Partie I. Généralités

Question 1. Il existe dans G_0 un couplage de cardinal 3, par exemple $\{0_A, 1_B\}$, $\{1_A, 3_B\}$ et $\{2_A, 0_B\}$.
En revanche, il n'existe pas de couplage de cardinal 4 ; en effet, si un tel couplage existait les sommets 1_A et 3_A seraient tous deux couplés à 3_B et les deux arêtes correspondantes seraient incidentes.

Question 2. Pour chaque nouveau couplage rencontré il faut vérifier si l'arête correspondante existe dans le graphe et si elle n'est pas incidente à une arête déjà rencontrée. Pour cela on utilise un tableau `dejaVu` qui marque les sommets de B dès lors qu'ils sont couplés à un sommet de A.

```
def verifie(g, c):
    n = len(c)
    dejaVu = [False] * n
    for i in range(n):
        if c[i] != -1:
            if not g[i][c[i]] or dejaVu[c[i]]:
                return False
            else:
                dejaVu[c[i]] = True
    return True
```

L'accès aux éléments d'une liste est de complexité constante donc la complexité de cette fonction est un $O(n)$.

Question 3. Il suffit de dénombrer le nombre de valeurs du tableau c qui ne sont pas égales à -1 :

```
def cardinal(c):
    s = 0
    for x in c:
        if x != -1:
            s += 1
    return s
```

La complexité de cette fonction est bien évidemment un $O(n)$.

Partie II. Un algorithme pour déterminer un couplage maximal

Question 4. Dans le graphe initial toutes les arêtes ont une somme des degrés des extrémités égale à 4, 5 ou 6. Seules deux ont une somme égale à 4 : les arêtes $\{1_A, 3_B\}$ et $\{3_A, 3_B\}$. On choisit par exemple l'arête $\{1_A, 3_B\}$.
Une fois les arêtes incidentes éliminées, il ne reste que des arêtes de somme 5 ; on choisit l'arête $\{0_A, 0_B\}$ qu'on retire ainsi que les arêtes incidentes.
Il ne reste alors que deux arêtes : $\{2_A, 1_B\}$ et $\{2_A, 2_B\}$; on choisit la première et les deux arêtes restantes sont alors éliminées : l'algorithme se termine en renvoyant le couplage $\{\{1_A, 3_B\}, \{0_A, 0_B\}, \{2_A, 1_B\}\}$.

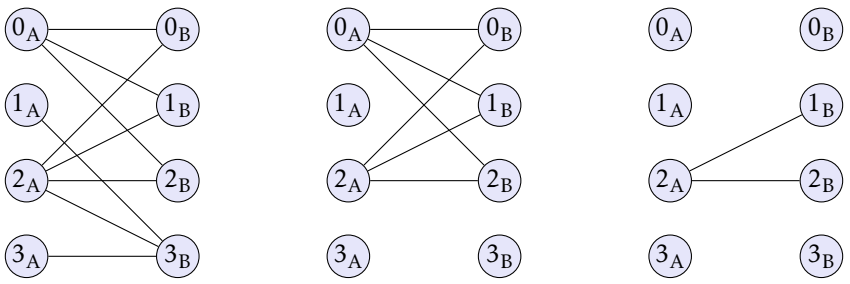
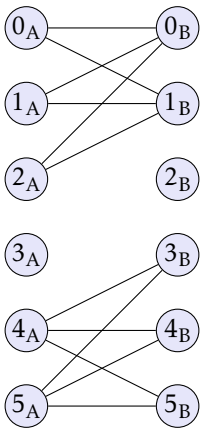


Figure 1 – Les trois étapes d'élimination de l'algorithme.

Question 5. Dans G_1 toutes les arêtes ont une somme des degrés des extrémités égale à 4, 5 ou 6. Une seule a une somme égale à 4 : l'arête $a_1 = \{3_A, 2_B\}$. Une fois celle-ci éliminée ainsi que les arêtes incidentes il reste le graphe :



On constate que le graphe obtenu n'est plus connexe ; chacune de ses deux composantes peut posséder en son sein au plus deux couplages (il est d'ailleurs facile de constater que c'est effectivement le cas) donc l'algorithme retournera un couplage de cardinal inférieur ou égal à 5. Or il existe un couplage de cardinal 6 : $\{\{0_A, 0_B\}, \{1_A, 1_B\}, \{2_A, 2_B\}, \{3_A, 3_B\}, \{4_A, 4_B\}, \{5_A, 5_B\}\}$; l'algorithme ne garantit donc pas d'obtenir un couplage de cardinal maximal.

Question 6. On commence par rédiger une fonction qui calcule les degrés des sommets de A et des sommets de B :

```
def calcule_degres(g):
    n = len(g)
    degA = [0] * n
    degB = [0] * n
    for i in range(n):
        for j in range(n):
            if g[i][j]:
                degA[i] += 1
                degB[j] += 1
    return degA, degB
```

Cette première fonction est bien évidemment de complexité quadratique.
Il reste ensuite à partir à la recherche de l'arête minimale :

```
def arete_min(g):
    n = len(g)
    degA, degB = calcule_degres(g)
    maxi = 2 * n + 1
    for i in range(n):
        for j in range(n):
            if g[i][j] and degA[i] + degB[j] < maxi:
                maxi = degA[i] + degB[j]
                a = (i, j)
    if maxi == 2 * n + 1:
        return False, None
    return True, a
```

On notera que la somme des degrés des extrémités d'une arête ne peut excéder $2n$, ce qui explique la valeur initiale de la variable `maxi`.
Cette fonction est de complexité quadratique $O(n^2)$.

Question 7. La fonction demandée fait passer à `False` toutes les valeurs de la ligne d'indice i et de la colonne d'indice j :

```
def supprimer(g, a):
    n = len(g)
    i, j = a
    for k in range(n):
        g[i][k] = False
        g[k][j] = False
```

Cette fonction est de complexité linéaire.

Question 8. On définit enfin la fonction principale de l'algorithme *algo_approche* :

```
def algoA_approche(g):
    n = len(g)
    gg = [[g[i][j] for j in range(n)] for i in range(n)]
    c = [-1] * n
    b, a = arete_min(gg)
    while b:
        c[a[0]] = a[1]
        supprimer(gg, a)
        b, a = arete_min(gg)
    return c
```

Notez la manière de dupliquer une matrice sur le première ligne de ce code.
Sachant que la complexité de la fonction `arete_min` est un $O(n^2)$ et qu'un couplage maximal comporte au maximun n couplages, la complexité totale de cette fonction est en $O(n^3)$.

Partie III. Recherche exhaustive d'un couplage de cardinal maximum

Question 9.

```
def une_arete(g):
    n = len(g)
    for i in range(n):
        for j in range(n):
            if g[i][j]:
                return (True, (i, j))
    return (False, None)
```

Question 10. Lorsque le graphe G contient au moins une arête a, on réalise deux copies G_1 et G_2 de G. Dans la première on supprime l'arête a et dans la seconde l'arête a ainsi que toutes les arêtes incidentes.
Tout couplage C_1 de G_1 est un couplage de G ne contenant pas l'arête a ; tout couplage C_2 de G_2 à qui on ajoute a est un couplage de G contenant l'arête a. Ceci conduit à l'algorithme suivant :

```
def meilleur_couplage(g):
    n = len(g)
    b, a = une_arete(g)
    if not b:
        return [-1] * n
    g1 = [[g[i][j] for j in range(n)] for i in range(n)]
    g1[a[0]][a[1]] = False
    g2 = [[g[i][j] for j in range(n)] for i in range(n)]
    supprimer(g2, a)
    c1 = meilleur_couplage(g1)
    c2 = meilleur_couplage(g2)
    c2[a[0]] = a[1]
    if cardinal(c1) > cardinal(c2):
        return c1
    else:
        return c2
```