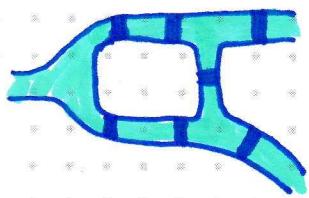


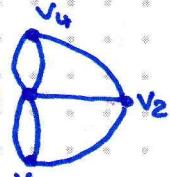
THEORIE DES GRAPHS

1735 : Euler - les 7 ponts de Königsberg



Est-il possible de faire une promenade qui traverse chaque pont exactement une fois?

Spoiler : Non. → Preuve de Euler



Voir ça comme un graphe (multigraph) :

$$G = (V, E)$$
$$V = \{v_1, v_2, v_3, v_4\}$$
$$E = \{(v_3, v_4), (v_2, v_3), (v_1, v_2), (v_2, v_4), (v_1, v_3)\}$$

Ex de chemin : $\sigma = (v_1, v_2)(v_2, v_4)(v_4, v_3) \dots$

À chaque fois qu'on passe par un sommet, on utilise une arête pour rentrer et une arête pour sortir sauf pour les sommets initial et final.

Si un chemin eulerien existe, il faut nécessairement que le nombre de sommet de degré impair soit 0 ou 2
(arrivée = départ) (arrivée ≠ départ)

Ici : $\deg(v_1) = 3$ $\deg(v_2) = 3$ $\deg(v_3) = 5$ $\deg(v_4) = 3$
⇒ Il n'existe pas de solution aux 7 ponts de Königsberg.

Mais c'est pas suffisante. Contre ex :

THM : un (multi)graphe non orienté possède un cycle eulerien si tous ses sommets sont de degré pair et les sommets de degré > 0 sont dans la même composante connexe.

Composante connexe : ensemble de sommets qui sont tous connectés (indirectement)

S'il y a deux sommets de degré impair alors il existe un chemin eulerien.



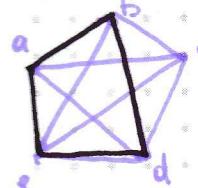
Algo pour produire un cycle eulerien dans un graphe connexe dont tous les sommets sont pairs



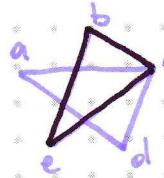
- Choisir un sommet de départ et construire un chemin à partir de ce sommet en effaçant les arêtes au fur et à mesure. On peut avancer jusqu'à retomber sur pt départ.
- Dans les arêtes qui restent, l'un des sommets est forcément sur le chemin créé car graphe connexe.
- On répète l'étape 1 à partir de ce sommet

- insérer ce cycle dans le chemin
- répéter 2, 3, 4 jusqu'à ce qu'il n'y ait plus d'arêtes

Ex avec K_5 :



$a \rightarrow b \rightarrow d \rightarrow e \rightarrow a$



$b \rightarrow c \rightarrow e \rightarrow b$

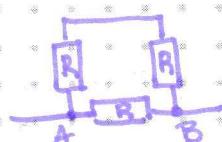


$a \rightarrow c \rightarrow d \rightarrow a$

$\Rightarrow a \rightarrow b \rightarrow c \rightarrow e \rightarrow b \rightarrow d \rightarrow e \rightarrow a$

1847: Gustav Kirchhoff - loi des noeuds, loi des mailles

ex:

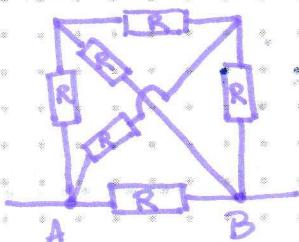


$$= \frac{1}{R'} = \frac{1}{R} + \frac{1}{2R} \Rightarrow R' = \frac{2}{3}R$$

$R' = ???$



A. courants :



contient AB

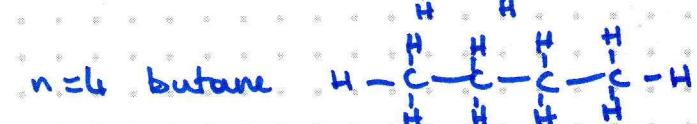


$$\Rightarrow R' = \frac{8}{16}R = \frac{1}{2}R$$

$$\rightarrow R' = \frac{\text{contient } AB}{\text{total}} \times R$$

Arbres couvrants : arbre inclus dans le graphe qui connecte tous les sommets du graphe.

1860: Cayley - Enumération des isomères des alcanes



n	1	2	3	4	5	6	7	8	9	10	11	12
#isomères	1	1	1	2	3	5	9	16	35	75	155	355

un alcane : 1 arbre avec n carbones et $2n+2$ hydrogènes

On veut montrer $2n+2 = 2n+2$ (C_nH_{2n+2})

$\rightarrow n+2$ sommets et $n+2-1$ arêtes

Prop: Un arbre de n sommets possède $n-1$ arêtes.

Prop : la somme des degrés de tous les sommets est deux fois le nombre d'arêtes.

$$\sum_{v \in V} \deg(v) = 2 \times |E|$$

Pour les alcunes : $\sum \deg(v) = 2x(n+x-1)$

$$\Leftrightarrow 4n + x = 2n + 2x - 2$$

$$\Leftrightarrow 2n + 2 = x$$

Prop : le nbr de sommets de deg impair est pair.
($\sum \deg(v) = 2|E|$ est pair)

Formule de Cayley :

le nbr d'arbre (sans racine) de sommets numérotés est n^{n-2}

Def : Un arbre est un graphe acyclique et connexe.

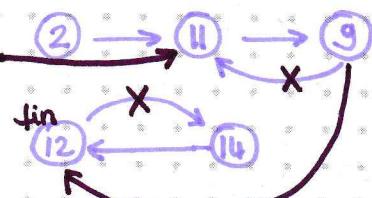
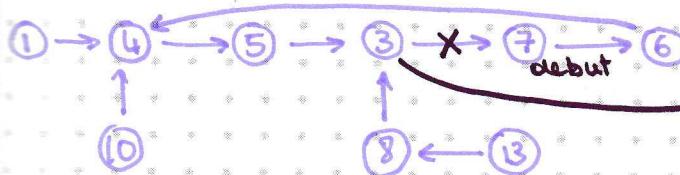
Demo (moderne) de la formule de Cayley :
on montre une bijection entre

{tut de $[1, n]$ vers $[1, n]$ } et {arbres numérotés entre 1 et n avec 2 marques début et fin}

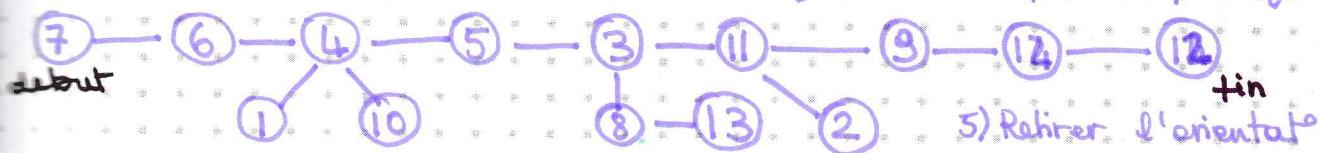
Cardinal : $n^n = T(n) \times n^2$ d'où $T(n) = n^{n-2}$

Ex: $x \parallel 1 2 3 4 5 6 7 8 9 10 11 12 13 14$
 $+x \parallel 4 11 7 5 3 4 6 3 11 4 9 14 8 12$

D On interprète $(x, +x)$ comme un arc



- 2) casser les cycles après leur plus petite valeur
- 3) connecter les cycles dans l'ordre des plus petites valeurs
- 4) Marquer le début du plus petit cycle et la fin du plus grand.

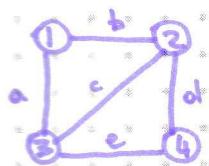


5) Retirer l'orientation

- 5) Remettre les flèches de manière à aller de début vers fin
- 3) Cherche la + pt valeur entre début et fin (fin 1^{er} cycle)
- 2) Répéter pour les cycles suivants
- 4) Supprimer les marques
- 1) trivial.

THM : le nbr d'arbres couvrants d'un graphe G est égal à la valeur absolue de n'importe quel cofacteur de la matrice laplacienne de G.

Matrice laplacienne : $L = D - M$
 (matrice de degré) (matrice d'adjacence)



$$D = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

$$M = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$L = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & 2 & 2 \end{bmatrix}$$

$$\begin{aligned} & \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & 2 & 2 \end{bmatrix} \\ & = 12 - 2 - 2 = 8 \end{aligned}$$

$$M = \begin{bmatrix} a+b & -b & -a & 0 \\ -b & b+c+d & -c & -d \\ -a & -c & a+c+e & -e \\ 0 & -d & -e & d+e \end{bmatrix}$$

$$\begin{aligned} & \begin{vmatrix} a+b & -b & 0 \\ -a & a+c+e & -e \\ 0 & -e & d+e \end{vmatrix} = (a+b)(a+c+e)(d+e) - (d+e)^2 \\ & = acd + ace + aed + bcd + bed + bce = \square \quad \square \quad \square \quad \square \end{aligned}$$

PARCOURS

$$G = (V, E)$$

$$V = \{0, 1, 2, 3, 4, 5, 6\}$$

$$E = \{(0, 1), (0, 3), (1, 2), (1, 5), (3, 4), (4, 5)\}$$

avec la convention que si $(s, d) \in E$ alors $(s, d) = (d, s)$.

On va calculer des complexité en fonction de $|V|$ et $|E|$

$$|E| \leq \binom{|V|}{2} = \frac{|V| \times (|V|-1)}{2} < \frac{|V|^2}{2} \Rightarrow |E| = O(|V|^2)$$

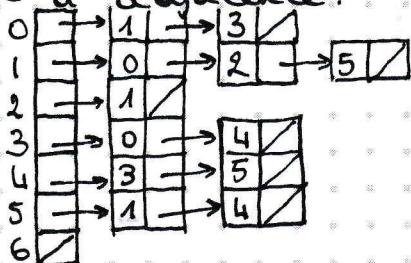
Dans les graphes connexes, il en y au moins $|E| \geq |V| - 1$

$$\Rightarrow |E| = \Omega(|V|)$$

Rappel : $\sum_{v \in V} \deg(v) = 2|E| = \Theta(|E|)$

DFS = depth first search = parcours en profondeur

Liste d'adjacence :



En Python :

`edges = [[1, 3], [0, 2, 5], [1], [0, 4], [3, 5], [1, 4], []]`,

`len(edges)` donne $|V|$

```
def dfs(adj):
    n = len(adj)
    seen = [False] * n
    def rec(start):
        print(start)
        seen[start] = True
        for d in adj[start]:
            if not seen[d]
                rec(d)
```

```
for d in range(n)
    if not seen[d]
        rec(d)
```

Version itérative du DFS :

pile de paires (i, j) où i sommet ($\text{edges}[i]$)
 j successeur ($\text{edges}[i][j]$)

```
def dfs_iter(adj):
    n = len(adj)
    seen = [False] * n
    stack = []
    for start in range(n):
        if seen[start]:
            continue
        stack = [(start, 0)]
        while stack:
            src, pos = stack.pop()
            print(src)
            if pos == 0:
                seen[src] = True
            if pos == len(adj[src]):
                continue
            stack.append((src, pos + 1))
            if not seen[adj[src][pos]]:
                stack.append((adj[src][pos], 0))
```

BFS = Breadth-First Search = parcours largeur

```
def BFS(adj):
    n = len(adj)
    seen = [False] * n
    for start in range(n):
        if seen[start]:
            continue
        queue = [start]
        seen[start] = True
        while queue:
            src = queue.pop(0)
            print(src)
            for dst in adj[src]:
                if not seen[dst]:
                    seen[dst] = True
                    queue.append(dst)
```

\Rightarrow BFS en $\Theta(|V| + |E|)$ \rightarrow linéaire pour un graphe

Calcul de distance depuis un sommet :

```
def distmap(adj, start):
    n = len(adj)
    dist = [None] * n
    queue = deque([start])
    dist[start] = 0
    while queue:
        src = queue.popleft()
        d = dist[src]
        for dst in adj[src]:
```

```
for dst in adj[src]:
    if dist[dst] is None:
        dist[dst] = d + 1
        queue.append(dst)
return dist
```

Dijkstra

$\text{dijkstra}(G = (V, E), \text{start}) :$

$\forall v \in V, D[v] \leftarrow \infty$
 $D[\text{start}] \leftarrow 0$
 $\text{queue} \leftarrow \{\text{start}\}$
 $\text{while } \text{queue} \neq \emptyset$

```

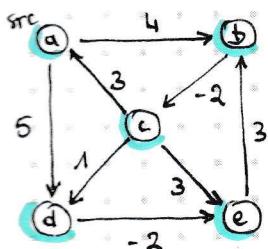
    min ← queue.pop_min()
    d ← D[min]
    for y ∈ successor(min)
        d' ← d + w(min, y)
        old ← D[y]
        D[y] ← min(old, d')
        if dd = ∞
            queue.insert(y)
        else
            queue.update(y)
    return D.

```

Algo de Bellman-Ford :

PCC ds graphes avec poids $\in \mathbb{R}$, sans cycle négatif
(1 source, n destination)

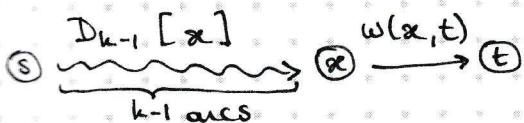
approche par prog. dyn^a



Supposons que $(s) \rightarrow (o) \rightarrow (o) \rightarrow (x) \rightarrow (t)$
+ court chemin de s à t
alors $(s) \rightarrow (o) \rightarrow (o) \rightarrow (x)$ optimal pour aller
de s à x en k arcs.

Notons $D_k[t]$ la longueur du PCC allant
de s à t en $\leq k$ arcs.

$$\begin{cases} D_1[t] = w(s, t) \\ D_k[x] = \min \{D_{k-1}[x] + w(x, t) \mid x \in V\} \quad \forall k \in [2, |V|-1] \end{cases}$$



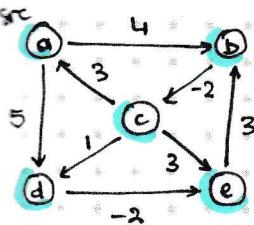
Bellman-Ford (V, E, w, s)

$\forall x \in V, D_1[x] \leftarrow w(s, x)$
for $k \leftarrow 2$ to $|V|-1$
for $t \in V$
 m ← ∞
 for $x \in V$
 m ← $\min(m, D_{k-1}[x] + w(x, t))$
 D_k[t] ← m

return $D_{|V|-1}$

$\Theta(|V|)$
 $\Theta(|V|)$
 $\Theta(|V|^2)$
"
 $\Theta(|V|^3)$
"
 $\Theta(|V|^2)$

 $\Theta(|V|^3)$



	a	b	c	d	e
D ₁	0	4	∞	5	∞
D ₂	0	4	2	5	3
D ₃	0	4	2	3	3
D ₄	0	4	2	3	1

$\leftarrow w$

- Opti 1 : Remplacer D_1, D_{k-1} et D_k par D .
Avec k à l'itérat° k , D peut contenir des PCC utilisant + que k arcs.
- Opti 2 : Arrêter la boucle sur k après une itérat° sans mise à jour de valeur $\rightarrow O(|V|^3)$.
- Opti 3 : Tester uniquement les arêtes qui existent
 \Rightarrow bellmanford (V, E, w, s)

```

 $\forall x \in V, D[x] \leftarrow w(s, x)$ 
for  $k \leftarrow 2$  to  $|V|-1$ 
  for  $(x, t) \in E$ 
     $D[t] \leftarrow \min(D[t], D[x] + w(x, t))$ 
return  $D$ 

```

$$\begin{aligned}
&\Theta(|V|) \\
&\Theta(|V|) \\
&\Theta(|V| \times |E|) \\
&\Theta(|V| \times |E|) \\
\hline
&\Theta(|V| \times |E|)
\end{aligned}$$

Pour retrouver les PCC correspondants aux distances, il faut sauvegarder le x qui donne le min.



bellmanford (V, E, w, s)

```

 $\forall x \in V, D[x] \leftarrow w(s, x)$ 
 $\forall x \in V, Father[x] \leftarrow s$ 
for  $k \leftarrow 2$  to  $|V|-1$ 
  for  $(x, t) \in E$ 
     $m \leftarrow D[x] + w(x, t)$ 
    if  $D[t] > m$ 
       $D[t] \leftarrow m$ 
      Father[t]  $\leftarrow x$ 
return  $D, Father$ 

```

Avec l'exemple, on a :

	a	b	c	d	e
D	0	4	2	3	1
Father	a	a	b	c	d

Généralisons à n sources et à n destinations.

\rightarrow On veut construire une matrice de distance
Pour programmat° dyn° $\textcircled{s} \rightsquigarrow \textcircled{x} \rightsquigarrow \textcircled{t}$

Notons $D_k[s, t]$ la longueur du PCC reliant s à t avec $\leq k$ arcs

$$1. \quad D_1[s, t] = w(s, t)$$

$$D_k[s, t] = \min \{D_{k-1}[s, t] + w(x, t) \mid x \in V\} \quad k \geq 1$$

$$2. \quad D_0[s, t] = \begin{cases} 0 & \text{si } s = t \\ \infty & \text{si } s \neq t \end{cases}$$

$$D_k[s, t] = \min \{D_{k-1}[s, t] + w(x, t) \mid x \in V\} \quad k \geq 1$$

$\text{dist}(V, E, w)$

- $s \in V, \forall t \in V, D[s, t] \leftarrow w(s, t)$

- $k \leftarrow 2 \text{ to } |V|-1$

for $s \in V$

for $t \in V$

$m \leftarrow \infty$

for $x \in V$

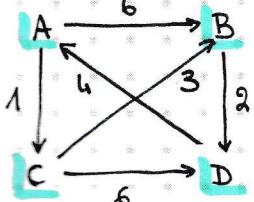
$m \leftarrow \min(m, D[s, x] + w(x, t))$

$D'[s, t] \leftarrow m$

$D \leftarrow D'$

return D

$\rightarrow \Theta(|V|^4)$



$$w = D_1 =$$

A	B	C	D
0	6	1	∞
∞	0	∞	2
∞	3	0	6
4	∞	∞	0

$$D_2 =$$

	D	B	C	A
D	0	4	1	7
6	0	∞	2	
10	3	0	5	
4	10	5	0	

$$D_3 =$$

	D	B	C	A
D	0	4	1	6
6	0	7	2	
9	3	0	5	
4	8	5	0	

$$\min \begin{cases} 4+6 \\ 10+0 \\ 5+3 \\ 0+\infty \end{cases}$$

$$D_3[D, B] = \min \{ D_2[D, \frac{B}{2}] + w[\frac{B}{2}, B] \}$$

$$D_1 = w, D_2 = D_1 \otimes w = D_1 \otimes D_1, D_3 = D_2 \otimes w = D_1 \otimes D_1 \otimes D_1 = w^3$$

→ élévation de matrice à la puissance $|V|-1$ du semi-anneau $(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$

Note: $D[|V|+1] = D[|V|-1]$

~~SlowDist~~ (V, e, w)

- $s \in V, \forall t \in V, D[s, t] \leftarrow w(s, t)$

for $k \leftarrow 2 \text{ to } \lceil \log(|V|-1) \rceil$

for $s \in V$

for $t \in V$

$m \leftarrow \infty$

for $x \in V$

$m \leftarrow \min(m, D[s, x] + D[x, t])$

$D'[s, t] \leftarrow m$

$D \leftarrow D'$

return D

$\leftarrow \Theta(|V|^3 \log |V|)$

~~Floyd-Warshall~~

$D[k+1] =$ longueur du PCC de
s à t qui n'utilise que
les $k+1$ premiers sommets du graphe

$$D[k+1] = w(s, t)$$

$$D[k+1] = \min(D_k[s, t], D_{k-1}[s, k] + D_k[k, t])$$

$k > 0$

Floyd-Warshall

$$D \leftarrow w$$

for $k \in V$

for $s \in V$

for $t \in V$

$$D'[s, t] \leftarrow \min(D[s, t], D[s, k] + D[k, t])$$

$D \leftarrow D'$

return D

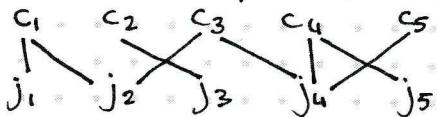
$\rightarrow O(|V|^3)$

COUPLAGE

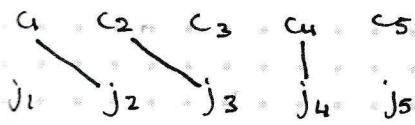
Couplage (Matching en anglais)

- Pb1 : • travailleurs candidats c_1, c_2, c_3, c_4, c_5 avec compétences variées
 • Tâches j_1, j_2, \dots demandant des compétences particulières.

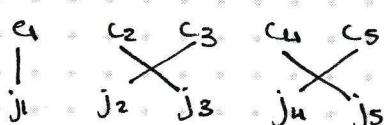
Graphes de compatibilité :



Q: Comment peut-on exécuter le plus de tâches à la fois ?



affectation non optimale

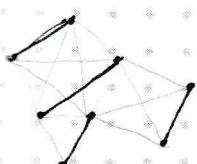


affectation optimale

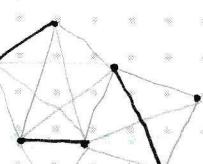
- Pb2 : armée multinationale

- chaque soldat parle ≥ 1 langue
- il faut 2 soldats parlant la même langue par tranche.

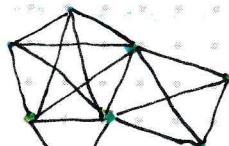
Q: Maximiser le nombre de soldat



Optimal



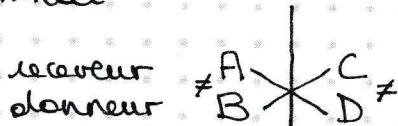
Non optimal



Graphes de compatibilité

- français
- italien
- allemand

- Pb3 : greffe d'organe

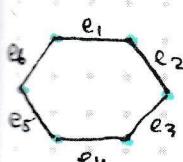


Q: Maximiser le nombre d'échanges.

Def: Un couplage d'un graphe $G = (V, E)$ est un sous ensemble $M \subseteq E$ d'arêtes tq M ne contient pas 2 arêtes voisines.

Un couplage maximal est un couplage M qui n'est pas contenu dans un couplage strictement plus grand.

Un couplage maximum est un couplage M tq \forall couplage M' , $|M'| \leq |M|$



$$E = \{e_1, e_2, \dots, e_6\}$$

$M_1 = \emptyset$ couplage "trivial"

$M_2 = \{e_1, e_4\}$ couplage maximal mais non maximum

$M_3 = \{e_1, e_3\}$ pas maximal

$M_4 = \{e_1, e_3, e_5\}$ maximum

$M_5 = \{e_2, e_4, e_6\}$ maximum (et dc maximal)

Note: Un couplage maximum est maximal

Dans un graphe de $|V|$ sommets, tout couplage M vérifie $|M| \leq \lfloor \frac{|V|}{2} \rfloor$

Un couplage M est parfait si $|M| = \frac{|V|}{2}$ (ts les sommets sont couplés)

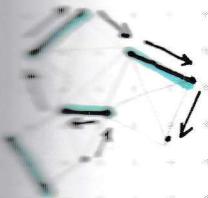
écrire un couple maximal ?

$\emptyset \leftarrow \emptyset$

Number 1

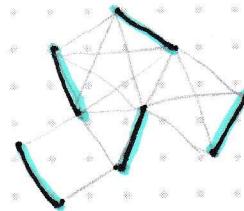
- tirer une arête (x, y) au hasard
 - ajouter (x, y) à M
 - supprimer x, y et leur arêtes incidentes.

→ Trouver un coupleage maximum?



- On construit un chemin améliorant
 - On permette les arêtes $\in M$ et $\notin M$
 - Cela fait gagner 1 arête de M

On répète tant que \exists chemin améliorant



Le sommet libre est un sommet dont toutes les arêtes incidentes sont pas de M.

— chemin $P \in E$ est améliorant s'il relie 2 sommets libres alternant des arêtes de $E \setminus M$ et des arêtes de M .

Prop: Si $G = (V, E)$ a un couplage M et un chemin améliorant P alors le couplage $M' = M \oplus P$ a une arête de plus que M .

Ex : 3 chemins amélioarent soit M n'est pas maximum

\Rightarrow evident d'ap la prop préc.
 \Leftarrow on suppose M non maximum, donc il existe M' tq $|M'| > |M|$
 considérons le graphe $G' = (V, M' \oplus M)$ constitué d'arêtes
 de M' sur M mais pas les 2.

les sommets non isolés de G' sont connectés par une alternance d'arêtes de M et M' . À cause de la une composante de G' contient plus d'arêtes de M' et forme un chemin au contraire.

→ constitue un couple maximum ?

M ← Ø

Tant que l'chemin ex

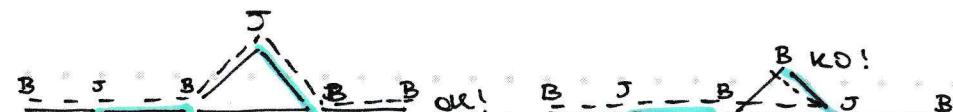
L M ← M

L Retourner M

Recherche de chemin améliorant

- au part d'un sommet libre
 - on constueit un autre alternant arêtes de EIM et arêtes de M
 - quand on tombe sur un sommet libre \rightarrow la branche correspondante est un chemin améliorant.

The diagram illustrates a protein dimerization interface. It features two protein monomers represented by grey shapes. The interface is highlighted by a dashed line and shows several hydrophobic patches, each marked with a cyan circle and labeled 'libre'. One prominent patch is located at the top of the interface, while others are distributed along the interface boundary.

Cas pénible : 

\rightarrow cycles de taille impaire paire peuvent pas = selon le sens ils peuvent être B ou J.

On remplace le cycle de taille impaire par un sommet B.

$B \quad J \quad S \quad B$ On a un chemin améliorant \rightarrow décompresser le bâgeon.

Algo d'Edmonds (recherche de chemin améliorant)

- E(VI) • Retirer les étiquettes de tous les sommets qui en ont
- E(EL) • Marquer tous les arêtes comme non explorées
- O(EL) • Repeter au choix
- O(EL)
 - (A) Trouver un sommet libre non étiqueté $v \in V$ et l'étiqueter $[v, B, v]$
 - (B) Trouver une arête (v, w) non explorée avec v d'étiquette $[r, B, p]$
 - 1) Marquer (v, w) comme explorée
 - 2) Si w n'a pas d'étiquette et est libre
 - $P \leftarrow$ chemin de $r \rightarrow w$
 - break
 - 3) Si w n'a pas d'étiquette et $\exists x \in \pi(w, x) \in M$ étiqueter w par $[r, J, v]$ et x par $[r, B, w]$
 - 4) Si w a l'étiquette $[s, B, q]$ avec $s \neq r$
 - $P \leftarrow$ chemin de $r \rightarrow w +$ celui de $w \rightarrow s$
 - break
 - 5) Si w a l'étiquette $[s, B, q]$ avec $s=r$
 - // cycle impair
 - remplacer les sommets du cycle par un seul sommet x dans le graphe et sauvegarder la correspondance
 - étiqueter x par $[r, B, v]$ sur une pile.
 - 6) Si w a pour étiquette $[s, J, p]$, ne rien faire.
 - E(I)
 - Si ni (A) ni (B) possibles, quitter la boucle avec $P \leftarrow \emptyset$
 - Si $P \neq \emptyset$
 - dépiler la pile des bâgeons et remplacer les occurrences d'état de substitut^o par le chemin de taille paire approprié.

chaque sommet sera étiqueté par $[r, c, p]$
 r : racine de l'arbre EV
 c : couleur B/J
 p : parent de l'arbre EV