LICENCE D'INFORMATIQUE

Sorbonne Université

LU3IN003 – Algorithmique
Cours 6 : Séance de révision

Année 2024-2025

Responsables et chargés de cours
Fanny Pascual
Olivier Spanjaard

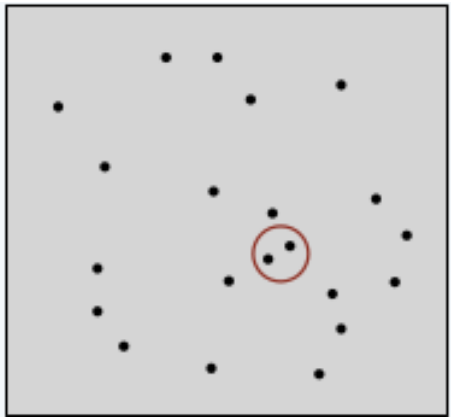
Diviser pour régner

Paire de points les plus proches

Etant donné n points dans un plan, trouver une paire de points les plus proches au sens de la distance euclidienne

Applications : vision, systèmes d'informations géographiques, contrôle aérien, modélisation moléculaire...

Algorithme naïf : tester toutes les paires de points en $\Theta(n^2)$

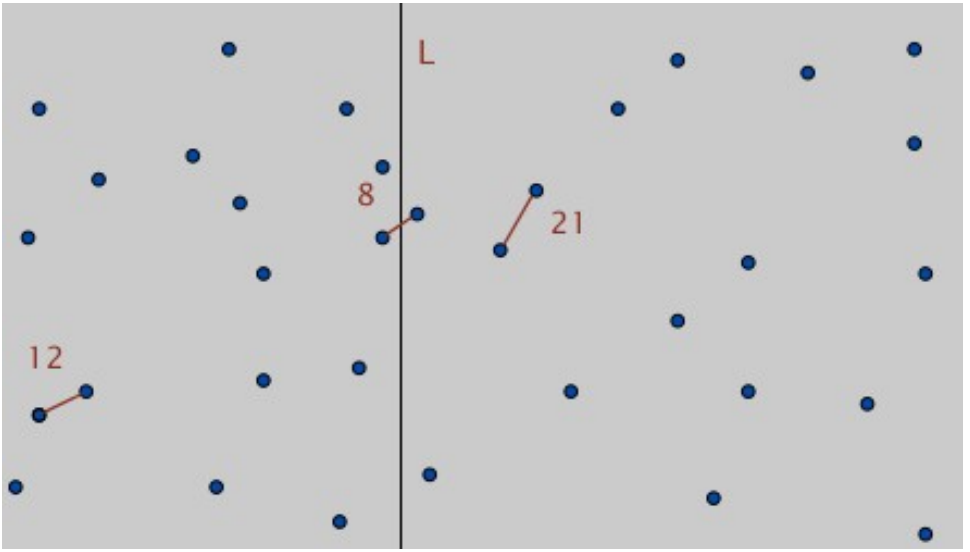


Paire de points les plus proches

Diviser Tracer une droite verticale L de façon à obtenir n/2 points dans chaque sous-région

Régner Trouver la paire de points les plus proches dans chaque sous-région

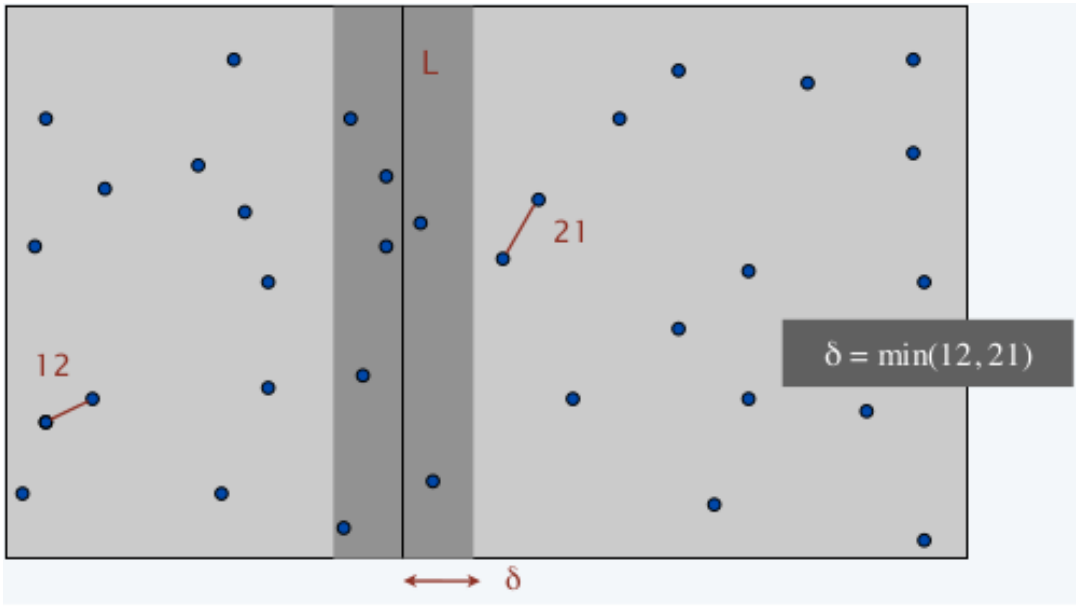
Combiner Trouver la paire de points les plus proches avec un point dans chaque sous-région
Retourner la meilleure des trois solutions trouvées



Paire de points les plus proches

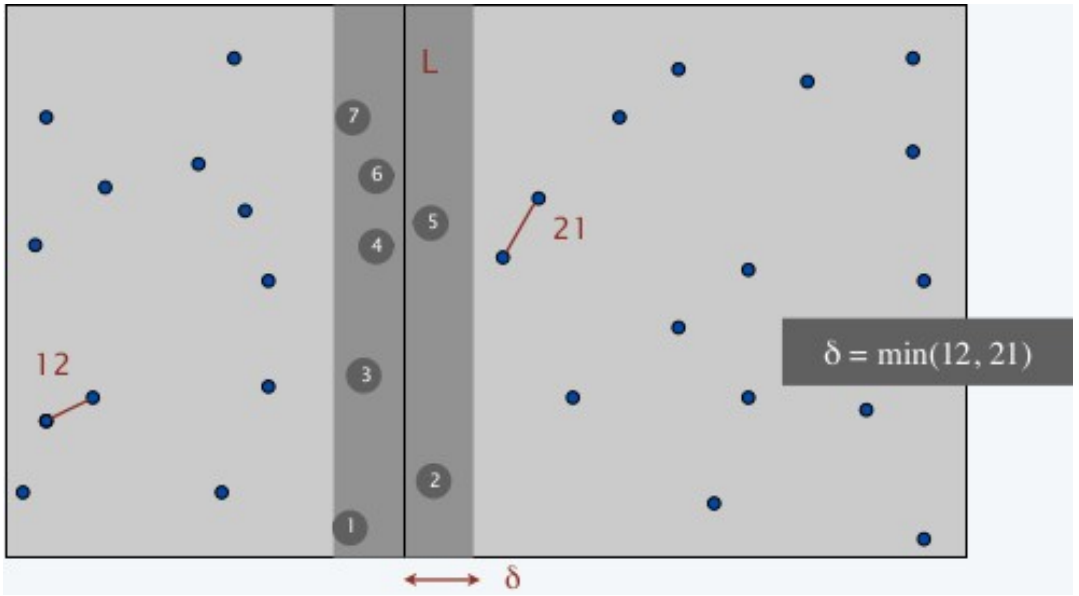
Comment trouver efficacement la paire de points les plus proches avec un point dans chaque sous-région ?

Remarque On peut se contenter d'examiner seulement les points de distance ≤ δ de la droite L, où $\delta = \min(d_{min}(regG), d_{min}(regD))$



Paire de points les plus proches

Trier les points dans la bande de largeur de largeur 2δ selon leurs ordonnées



Observation examen des distances que pour 8 positions de la liste triée

Paire de points les plus proches

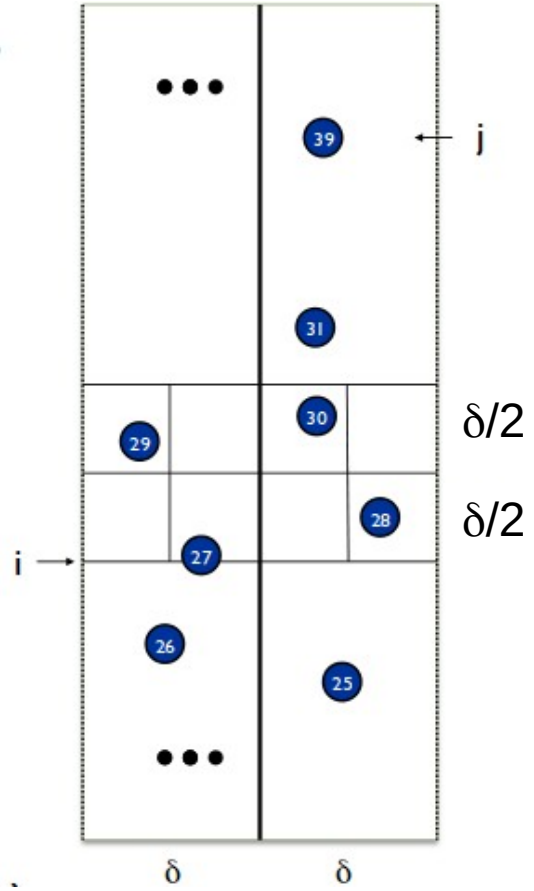
Soit s_i le point de la bande de largeur 2δ de i^e plus petite ordonnée

Propriété Si $|i-j| \geq 8$, alors la distance entre s_i et s_j est $\geq \delta$

Preuve (arguments)

1 seul point dans chaque carré

Deux points séparés par au moins deux bandes horizontales sont distants d'au moins $2(\delta/2)$



Paire de points les plus proches

PairePlusProche(p_1,…,p_n)

Si $n \leq 3$ alors retourner distmin entre les points (∞ si n=1)
Sinon calculer la ligne de séparation L

1. d_{min}(regG) ← PairePlusProche(points regG)
2. d_{min}(regD) ← PairePlusProche(points regD)
3. $\delta = \min(d_{min}(regG), d_{min}(regD))$
4. Supprimer tous les points à distance supérieure à δ de L
5. Examiner les points dans l'ordre de leurs ordonnées croissantes et comparer la distance entre le point courant et les 7 suivants. Si une de ces distances est $\leq \delta$, mettre à jour δ
6. Retourner (δ)

Paire de points les plus proches

On dispose de deux listes des points triés par ordre d'abscisses croissantes et d'ordonnées croissantes (calculées une fois pour toute en O(n log n))

PairePlusProche(p_1,…,p_n)

Si $n \leq 3$ alors retourner distmin entre les points (∞ si n=1)

Sinon calculer la ligne de séparation L

1. d_{min}(regG) ← PairePlusProche(points regG)
2. d_{min}(regD) ← PairePlusProche(points regD)

← 2T(n/2)

3. $\delta = \min(d_{min}(regG), d_{min}(regD))$
4. Supprimer tous les points à distance supérieure à δ de L
5. Examiner les points dans l'ordre de leurs ordonnées croissantes et comparer la distance entre le point courant et les 7 suivants. Si une de ces distances est $\leq \delta$, mettre à jour δ

← O(n)

6. Retourner (δ)

T(n) fonction complexité pire-cas de PairePlusProche(p_1,…,p_n)

Paire de points les plus proches

$T(1)=O(1)$;
$T(n)=T(\lfloor n/2 \rfloor)+T(\lceil n/2 \rceil)+O(n)$

La complexité de l'algorithme diviser pour régner pour le calcul d'une paire de points les plus proches est $O(n \log n)$

Quiz : Fonction d’affichage

Quel est l'ordre de grandeur du nombre de lignes affichées par la fonction f(n) ?

```
fonction f(n)
    si n>1
        afficher("encore une ligne")
        f(n/2)
        f(n/2)
```

A) O(log(n))
B) O(n)
C) O(nlog(n))
D) $O(n^2)$

Quiz : Tri fusion d’un tableau de chaînes

Une liste de n chaînes de caractères, chacune de longueur n, est triée en ordre lexicographique à l'aide de l'algorithme de tri fusion. Quelle complexité obtient-on dans ce cas ?

A) $O(nlog(n))$
B) $O(n^2)$
C) $O(n^2log(n))$
D) $O(n^3)$

Exploration d’un arbre d’énumération

Problème du sac à dos

- Un cambrioleur avec un sac à dos de capacité *W*, et il peut dérober un ensemble *S* de *n* objets
- Chaque objet *i* a un poids $w_i \in \mathbb{N}$ et rapporte $v_i \in \mathbb{N}$
- Problème : quel sous-ensemble d'objets de S dérober pour maximiser le profit ?

Programme d'optimisation

- Les objets ne peuvent pas être divisés
  - Un objet est sélectionné ou abandonné

$$\textbf{Max } \Sigma v_i x_i$$

$$\Sigma w_i x_i \leq W$$

$$x_i \in \{0,1\} \text{ pour } i=1,\ldots,n$$

Si $x_i = 1$, l'objet i est sélectionné
Si $x_i = 0$, l'objet i est abandonné

Fonction récursive d'énumération

Entrée : tableaux v[1...n] et w[1...n], capacité W du sac
Sortie : valeur maximum d'un sous-ensemble S d'objets vérifiant la contrainte de capacité

On pose $\mathbf{M = 1+\Sigma_i v_i}$

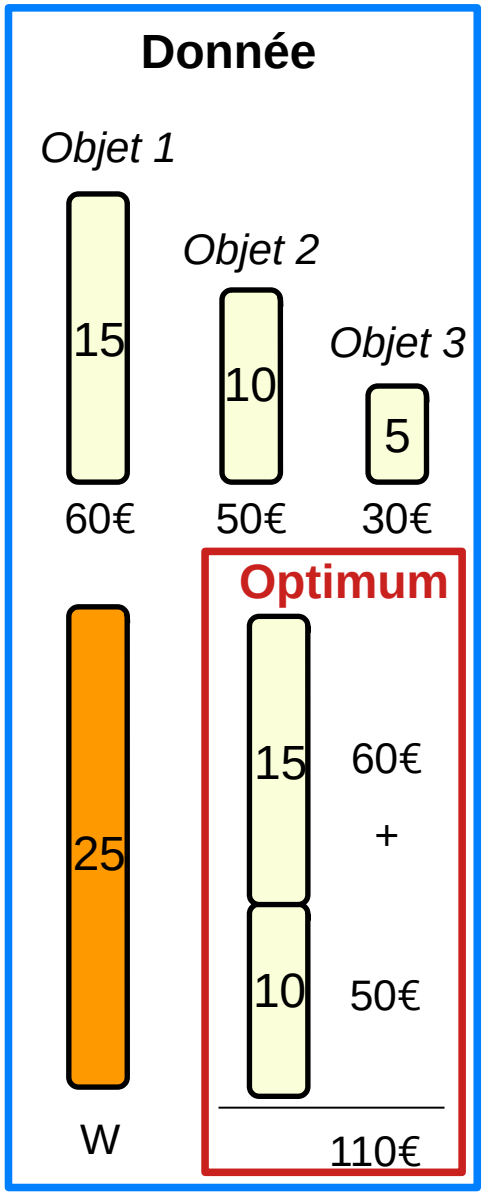
```
fonction Sac(i,w)
    si w<0
            retourner -M
    sinon si w=0 ou i=0
            retourner 0
    sinon
            retourner max{v_i+Sac(i-1,w-w_i),Sac(i-1,w)}
```

Appel initial
Sac(n,W)

On **inclut** l'objet d'indice i dans S

On **n'inclut pas** l'objet d'indice i dans S

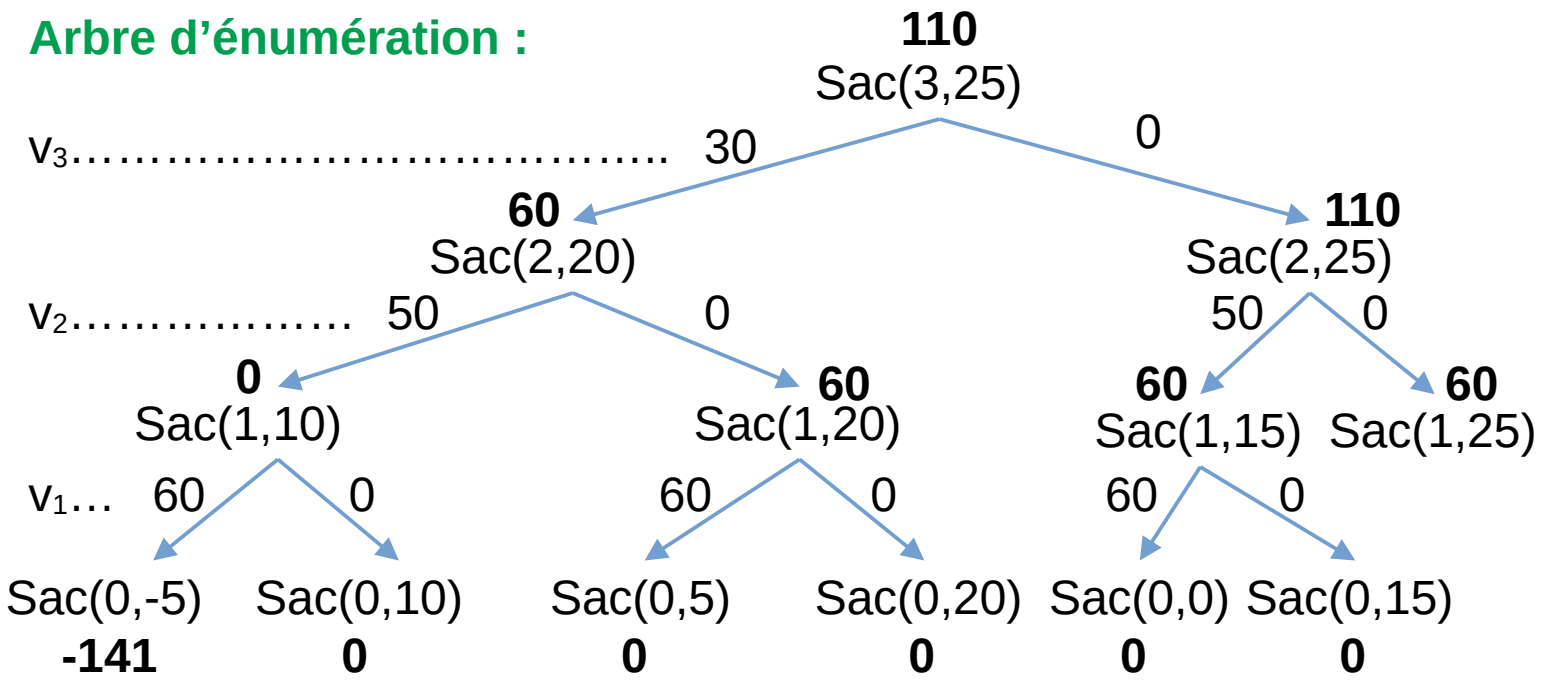
Exemple




```
fonction Sac(i,w)
    si w<0
          retourner -M
    sinon si w=0 ou i=0
          retourner 0
    sinon
          retourner max{v_i+Sac(i-1,w-w_i),Sac(i-1,w)}
```

$M=1+\Sigma_i v_i$

Arbre d'énumération :



Analyse de complexité

```
fonction Sac(i,w)
    si w<0
          retourner -M
    sinon si w=0 ou i=0
          retourner 0
    sinon
          retourner max{v_i+Sac(i-1,w-w_i),Sac(i-1,w)}
```

Nombre d'appels à la fonction **`Sac(i,w)`** en fonction du nombre n d'objets :
A(0) = 1
$A(n) \leq 2A(n-1)+1$

On sait que le terme général de la suite A'(n) obtenue en remplaçant ≤ par = est :
$A'(n) = 2^{n+1}-1$.

Il y a donc $O(2^n)$ nœuds dans l'arbre des appels.

Si l'on suppose chaque appel en O(1), la complexité de **`Sac(n,W)`** est $O(2^n)$.

Remarque importante

Supposons que l'on mémorise dans une table **`M[i,w]`** le résultat des appels à **`Sac`** pour les différents couples de paramètres **`(i,w)`**, et que l'on retourne directement le résultat lorsque l'on réalise un appel à **`Sac(i,w)`** pour lequel **`M[i,w]`** a déjà été calculé.

On parle de ***mémoïsation***.

```
fonction Sac(i,w)
    si M[i,w] est connu
        retourner M[i,w]
    sinon si w<0
            retourner -M
    sinon si w=0 ou i=0
            retourner 0
    sinon
            M[i,w]=max{v_i+Sac(i-1,w-w_i),Sac(i-1,w)}
            retourner M[i,w]
```

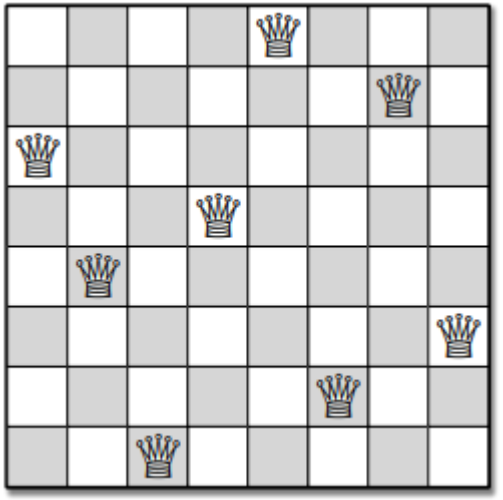
Il y a alors de l'ordre de *nW* appels récursifs non-mémoïsés (autant que de couples de paramètres **`(i,w)`** possibles), et la complexité devient O(nW).

Cette complexité est **pseudopolynomiale** car le paramètre W est encodé sur $\log_2 W$ bits.

Il s'agit d'un algorithme de programmation dynamique (vu plus tard dans le semestre).

Le problème des *n* dames

- On cherche à énumérer toutes les dispositions possibles de *n* dames sur un échiquier de manière à ce qu'aucune paire de dames ne se menacent l'une l'autre.
- On rappelle qu'aux échecs, une dame peut atteindre n'importe quelle case sur sa rangée, sa colonne, ou chacune de ses deux diagonales.
- Ci-dessous une disposition possible pour *n*=8 (taille standard d'un échiquier) :



Deux premières stratégies

- ***Tester les $(n^2)^n = n^{2n}$ dispositions possibles*** des *n* dames sur l'échiquier, et tester pour chacune si elle est réalisable (pas plus d'une dame par case) et sans aucunes dames qui se menacent
  → pour $n = 8$, cela fait environ $2{,}815 \times 10^{14}$ dispositions à tester !
- En remarquant que toute solution s'obtiendra en plaçant ***une dame par rangée***, et qu'on peut même s'arranger pour que les dames placées ne soient pas sur les colonnes déjà occupées au moment où on les place, il y a *n* façons de placer la première dame dans la première rangée, $n - 1$ façons de placer la deuxième dame sur la deuxième rangée, etc.
  → pour $n = 8$, cela ne fait plus que 8! = 40 320 dispositions à tester.

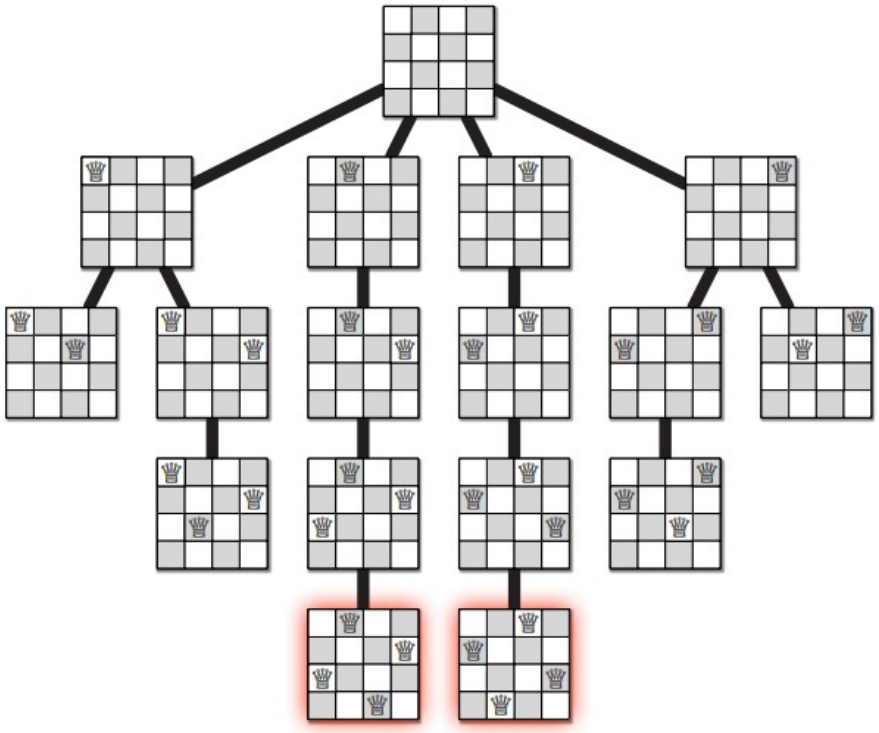
On peut faire mieux en remarquant que, lorsqu'on place les dames une à une comme dans la deuxième stratégie, tester si les dames placées se menacent permet de détecter rapidement qu'on est en train de construire une disposition qui échouera.

Stratégie récursive de résolution

Rappel : il ne peut n'y avoir qu'une et une seule dame par rangée (sinon les dames se menaceraient) dans une disposition valide.

Stratégie récursive pour énumérer les dispositions, où l'on place les dames rangée par rangée, du haut vers le bas :

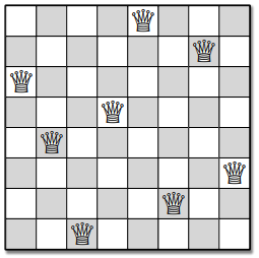
- Pour placer la *r*-ième dame, on essai méthodiquement les *n* cases de la rangée *r* de gauche à droite.
- Si une dame placée antérieurement menace une case, alors on ignore cette case ;
- sinon, on place provisoirement une dame sur la case considérée, et l'on cherche récursivement des placements cohérents des dames dans les rangées suivantes.



L'arbre des appels récursifs pour *n*=4 est représenté ci-dessus. Les deux feuilles de l'arbre en rouge correspondent aux deux solutions symétriques.

Plus formellement

Une solution aux problèmes des *n* dames peut être représentée en machine sous la forme d'un tableau D[1..n] où D[i]=j si la dame de la rangée i est positionnée en colonne j.



D=[5,7,1,4,2,8,6,3]

La dame de la rangée i menace la case (r,j) si D[i]=j (même colonne) ou D[i]=j+r-i (même diagonale montante) ou D[i]=j-r+i (diagonale descendante).

```
fonction PlacerLesDames(D[1..n],r)
  si r=n+1 afficher D[1..n]
  sinon
        pour j=1 à n faire
             valide=vrai
             pour i=1 à r-1 faire
                  si D[i]=j ou D[i]=j+r-i ou D[i]=j-r+i
                     valide=faux
             si valide
                  D[r]=j
                  PlacerLesDames(D[1..n],r+1)
```

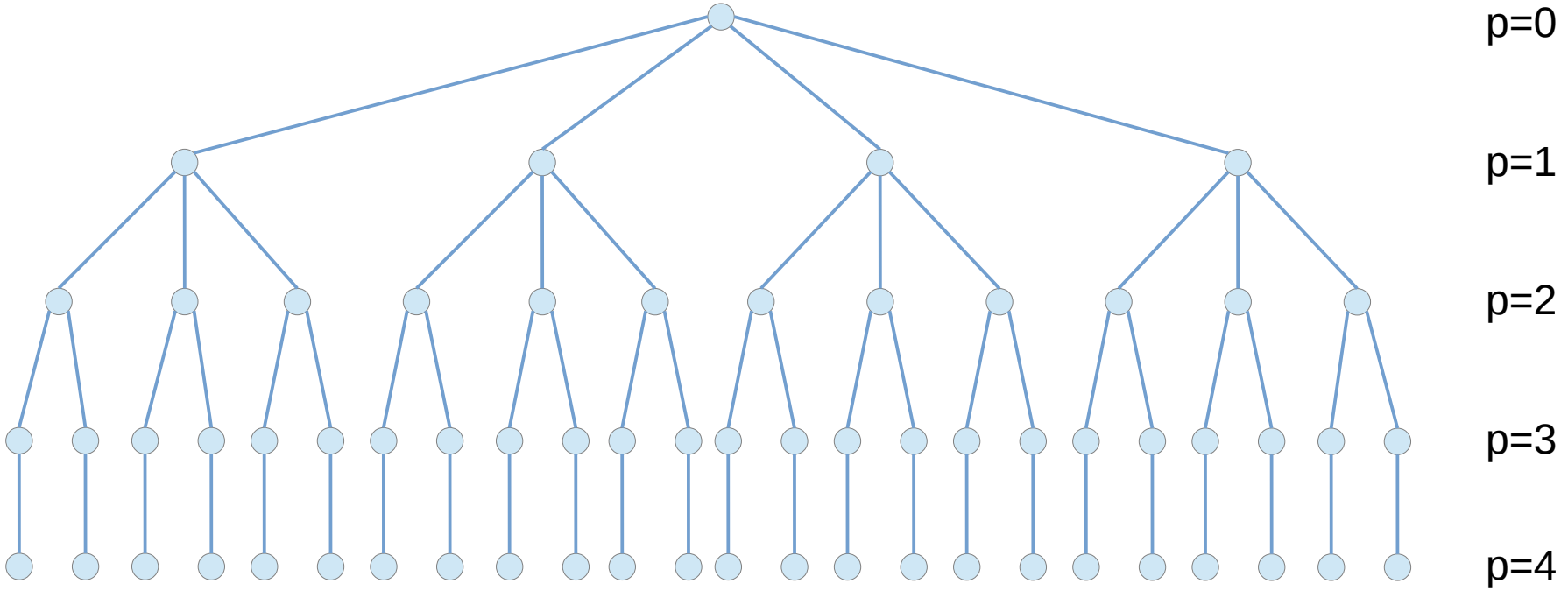
Appel initial
PlacerLesDames(D[1..n], 1)

Analyse de complexité (1/2)

Montrons que **le nombre de nœuds de l'arbre des appels récursifs, si l'on ne tenait compte que des colonnes menacées, est majoré par trois fois le nombre de feuilles**.

Soit I(p) le nombre de nœuds internes si l'on « coupe » l'arbre à la profondeur p, et F(p) le nombre de nœuds à la profondeur p.
Montrons par récurrence que $F(p) \geq I(p)$ pour $p \in \{1,\ldots,n-1\}$ (HR_p).
Cas de base. Pour p=1, $F(1)=n \geq 1=I(1)$.
Etape inductive. $F(p+1) \geq 2F(p)$ car le facteur de branchement est au moins 2 pour tout nœud de profondeur $p \in \{0,\ldots,n-2\}$. Ce qui se réécrit $F(p+1) \geq F(p)+F(p)$.
D'après HR_p, $F(p) \geq I(p)$. D'où $F(p+1) \geq I(p)+F(p)$. Or $I(p)+F(p)=I(p+1)$. D'où $F(p+1) \geq I(p+1)$.

Résultat : On a $I(n)=F(n-1)+I(n-1) \leq 2F(n-1)=2F(n)$, et donc $I(n)+F(n) \leq 2F(n)+F(n)=3F(n)$.



Analyse de complexité (2/2)

Comme le facteur de branchement est d'abord n, puis n-1, puis n-2, etc., le nombre de feuilles dans l'arbre des appels récursifs est n!

Le nombre total d'appel récursif est donc en O(F(n))=O(n!).

De plus, les deux boucles pour imbriquées sont en $O(n^2)$ pour chaque appel (car $r \leq n+1$).

La complexité est donc en $O(n^2.n!)$.

```
fonction PlacerLesDames(D[1..n],r)
  si r=n+1 afficher D[1..n]
  sinon
        pour j=1 à n faire
             valide=vrai
             pour i=1 à r-1 faire
                  si D[i]=j ou D[i]=j+r-i ou D[i]=j-r+i
                     valide=faux
             si valide
                  Q[r]=j
                  PlacerLesDames(D[1..n],r+1)
```

Appel initial
PlacerLesDames(D[1..n], 1)

Compléments sur ce problème

- L’arbre des appels récursifs pour $n = 8$ comporte 13 756 feuilles, et on trouve 92 dispositions valides des dames sur l’échiquier.
- La plus grande taille pour laquelle il a été possible d’énumérer toutes les dispositions valides est $n = 26$ (en utilisant une méthode particulièrement optimisée).
- Cela a été réalisé en 2009 par des chercheurs de l’Université de Dresde, en Allemagne.
- Le calcul a duré dix mois et a abouti à environ 22 millions de milliards de dispositions valides pour le problème des 26 dames.

Quiz : Analyse de complexité

On considère un algorithme récursif où le nombre A(n) de nœuds dans l'arbre des appels récursifs est caractérisé par :

$A(1)=1$
$A(2)=1$
$A(n)=1+2A(n-1)+A(n-2)$

Les opérations réalisées à chaque appel se font en temps constant. Quelle est la complexité de l'algorithme ?

A) $O(1.42^n)$
B) $O(2^n)$
C) $O(2.42^n)$
D) $O(3^n)$

Quiz : Relations de récurrence

On considère une liste d'algorithmes récursifs et une liste de relations de récurrence comme indiquées ci-dessous. Chaque relation de récurrence correspond à exactement un algorithme et permet de déduire sa complexité. Quelle est le bon appariement entre algorithmes et relations de récurrence ?

R. Recherche dichotomique
T. Tri fusion
S. Suite de Fibonacci

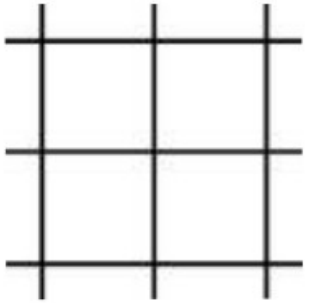
I. $T(n)=T(n-1)+T(n-2)+O(n)$
II. $T(n)=2T(n/2)+O(n)$
III. $T(n)=T(n/2)+O(1)$

A) (R,I) (T,II) (S,III)
B) (R,II) (T,I) (S,III)
C) (R,II) (T,III) (S,I)
D) (R,III) (T,II) (S,I)

Modélisation par les graphes

Quiz : Segments et intersections

Ci-dessous un exemple de 6 segments de droites ayant chacun une intersection avec 3 des 5 autres segments



Peut-on tracer 7 segments de droites ayant chacune une intersection avec 3 des 6 autres segments ?

A) Oui
B) Non

Quiz : Malversations financières

Un inspecteur de police enquête sur des soupçons de malversations financières. Il dispose d'une liste de 7 suspects et aimerait mieux connaître les activités professionnelles qui lient ces 7 personnes. Pour ce faire il présente sa liste aux suspects et demande à chacun d'indiquer le nombre de personnes parmi les 6 autres avec qui ils ont déjà été en relation d'affaires. Chacun répond que ce nombre est égal à 3.

Est-il possible qu'ils disent tous la vérité ?

A) Oui
B) Non

Handball

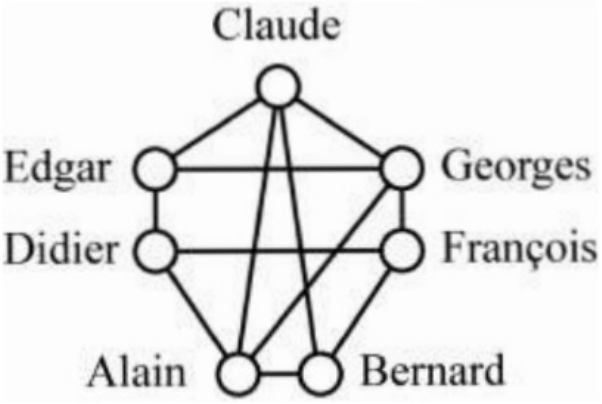
- Dans le cadre de la préparation physique d'une équipe de handball, **deux ateliers** sont organisés en parallèle pour les **7 coéquipiers**, pris en charge par deux intervenants extérieurs.
- Au terme de l'entraînement, chaque intervenant informe l'entraîneur que **3 coéquipiers** ont pris part à leur atelier.

  → Un joueur n'a pas donc pas participé à l'entraînement !
- Les intervenants, peu physionomistes, ne sont pas capables d'indiquer quels sont les joueurs qui ont participé à leur atelier.

Modélisation par un graphe

- L'entraîneur interroge les 7 joueurs.
- Chaque joueur donne les noms de deux coéquipiers qui n'ont pas participé au même atelier que lui (tableau de gauche)
- On en déduit le graphe de droite, dont **les sommets sont les joueurs** et où **une arête** figure entre deux sommets **si l'un au moins dit ne pas avoir vu l'autre** dans son atelier.

	n'était pas dans le même atelier que	
Alain	Bernard	Didier
Bernard	Claude	François
Claude	Alain	Georges
Didier	Edgar	François
Edgar	Claude	Georges
François	Bernard	Georges
Georges	Alain	Edgar



Alain Hertz. L'enseignement de la théorie des graphes à l'aide d'intrigues policières. Bulletin de l'APMEP 499, 2012.

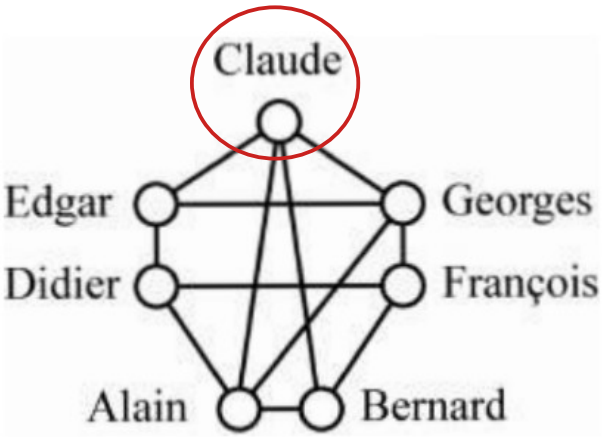
Propriété des graphes bipartis

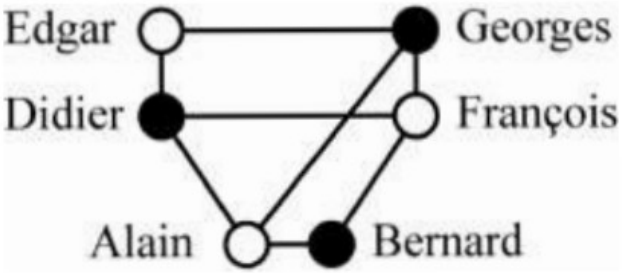
- Si les coéquipiers avaient tous participé, le graphe devrait être biparti :
  Joueurs ayant participé à un même atelier | Joueurs ayant participé à l'autre atelier.

Propriété. Soit G un graphe biparti entre S_1 et S_2 :
a) s'il existe une chaîne de longueur paire (en nombre d'arêtes) dans G entre deux sommets x et y alors ces deux sommets x et y appartiennent nécessairement au même sous-ensemble (S_1 ou S_2) de la partition.
b) s'il existe une chaîne de longueur impaire dans G entre deux sommets x et y alors l'un appartient à S_1 et l'autre à S_2.

- Soient x,y deux sommets d'un cycle C de longueur impaire alors $\{x,y\}$ est une chaîne de longueur 1 impaire et l'autre chaîne qui relie x et y dans C est paire.

- **Claude** appartient à tous les cycles de longueur impaire. C'est donc la personne recherchée, et le graphe devient bien biparti en supprimant son sommet.





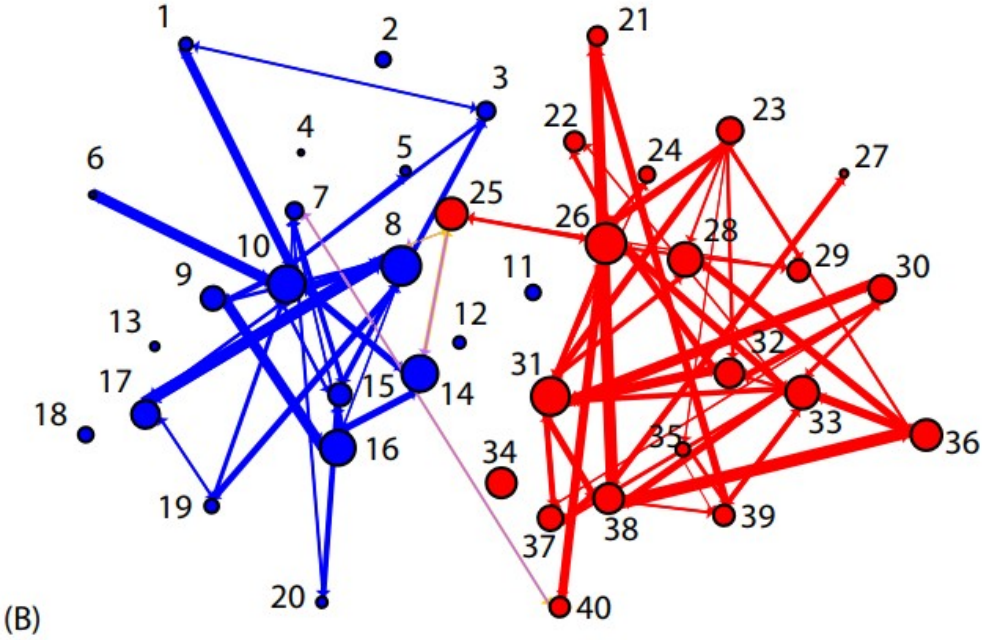
Application des parcours en profondeur

Détection de communautés

L'identification de composantes fortement connexes dans un graphe orienté permet de détecter des communautés.

Exemple : réseaux sociaux
On considère le graphe dont les sommets sont les utilisateurs du réseau social, et on met un arc d'un utilisateur auteur d'une publication vers un utilisateur qui y a répondu.

Une grande composante fortement connexe indique la présence d'une communauté où de nombreux utilisateurs interagissent, directement ou indirectement.



Analyse des interactions dans la blogosphère politique américaine lors de l'élection présidentielle de 2004 (Adamic et Glance, 2005).

Rappel : composantes connexes

Soit $L=(s_1,s_2,\ldots,s_n)$ un parcours de G.

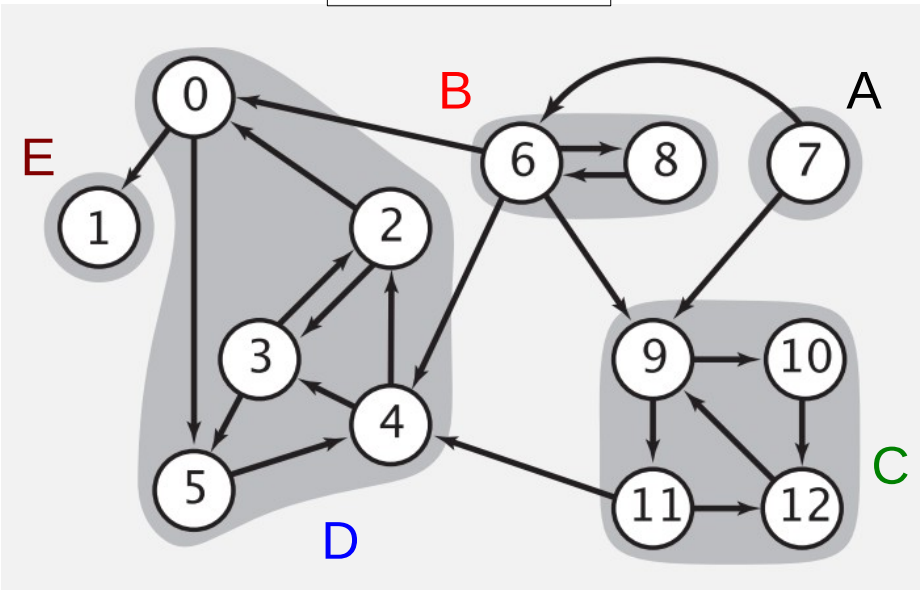
Soit $(i_1,\ldots,i_r)$ les indices des points de régénération de L.

$G[i_1..i_2\text{-}1]$, $G[i_2..i_3\text{-}1]$,…, $G[i_r..n]$ sont les sous-graphes induits par les **composantes connexes** de G ;
$L[i_1..i_2\text{-}1]$, $L[i_2..i_3\text{-}1]$,…, $L[i_r..n]$ sont des **parcours des sous-graphes induits par les composantes connexes** de G.

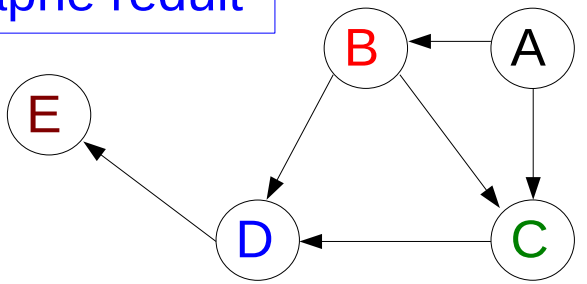
Idée : pour déterminer les composantes fortement connexes (CFC) d'un graphe **orienté**, s'assurer que chaque point de régénération appartient à une **CFC sans successeur dans le graphe réduit du sous-graphe des sommets non visités**, en sorte que $L[i_1..i_2\text{-}1]$, $L[i_2..i_3\text{-}1]$,…, $L[i_r..n]$ sont des parcours des sous-graphes induits par les **composantes fortement connexes** de G.

Composantes fortement connexes

Graphe G



Graphe réduit



Un parcours en profondeur :
(1,0,5,4,2,3,11,12,9,10,6,8,7)
E D C B A

Comment s'assurer que chaque point de régénération appartienne à une CFC sans successeur dans le graphe réduit du sous-graphe des sommets non visités ?

Observation clé : le sommet de plus grand numéro de postvisite à l'issue d'un parcours en profondeur d'un graphe orienté appartient nécessairement à une CFC **sans prédécesseur** dans le graphe réduit.

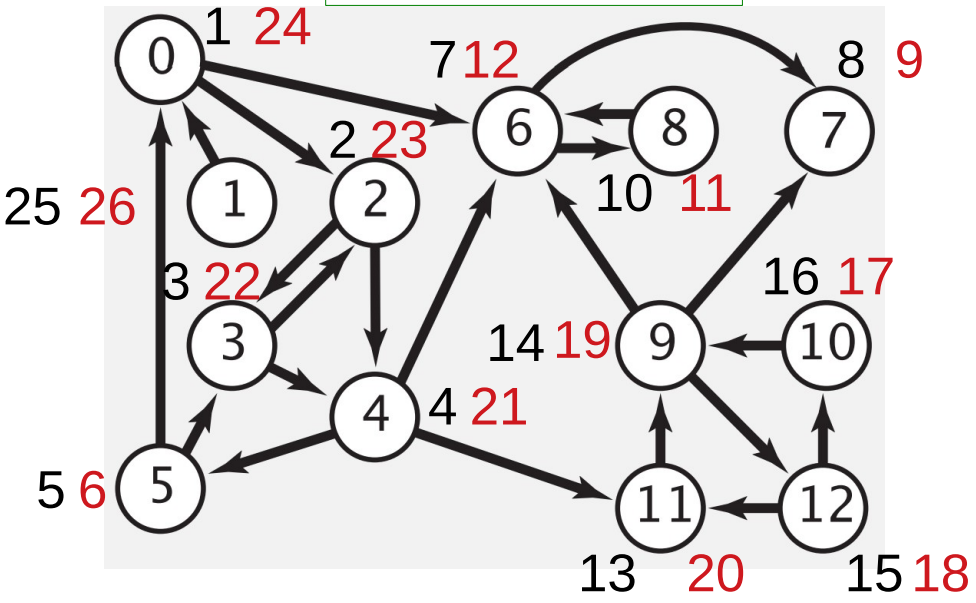
Conséquence : Soit G un graphe orienté et G^{Inv} le graphe obtenu ***en inversant l'orientation des arcs de G.*** Le sommet de plus grand numéro de postvisite à l'issue d'un parcours en profondeur de G^{Inv} appartient nécessairement à une CFC **sans successeur** dans le graphe réduit de G.

Algorithme de Kosaraju-Sharir

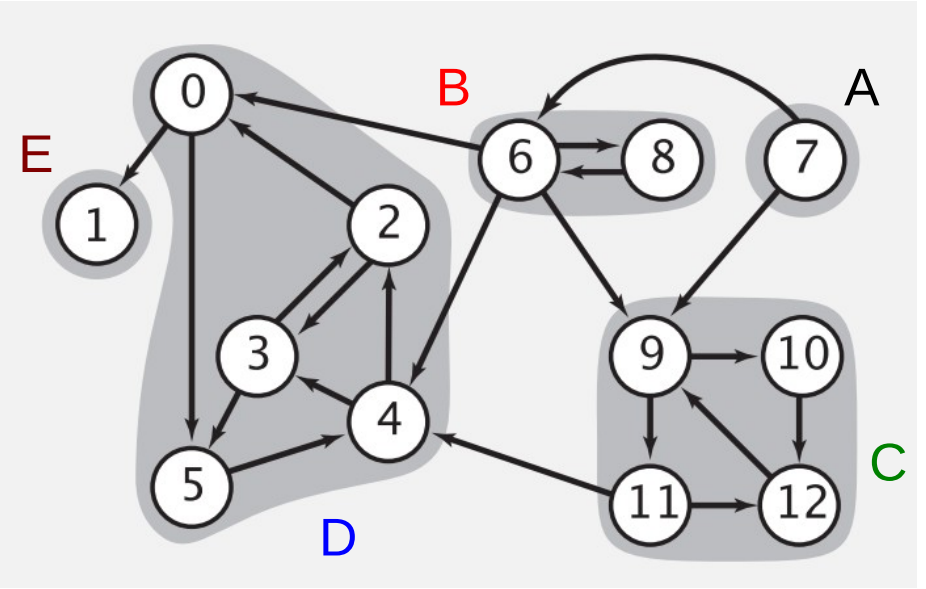
L'algorithme comporte trois étapes pour un graphe G :

1. Déterminer le graphe inverse de G.
2. Faire un parcours en profondeur du graphe inverse en effectuant une numérotation postfixe des sommets.
3. Faire un parcours en profondeur de G en démarrant par le sommet de plus grand numéro postfixe et en choisissant comme point de régénération, quand nécessaire, le sommet de plus grand numéro postfixe parmi les sommets non visités.

Graphe inverse



Graphe G



Parcours : (1,0,5,4,2,3,11,12,9,10,6,8,7)

Analyse de complexité

Etape 1 : La construction de la représentation du graphe inverse de G sous forme de listes de successeurs se fait en Θ(n+m) si G est lui même représenté sous forme de listes de successeurs.

```
Ginverse:=GrapheVide()
Pour tout sommet s de G faire
    Pour tout successeur t de s faire
        insérer s dans la liste de succ de t dans Ginverse
```

Etape 2 : Parcours en profondeur du graphe inverse de G en Θ(n+m).

Etape 3 : Parcours en profondeur de G en Θ(n+m).

L'algorithme de Kosaraju-Sharir est donc de complexité Θ(n+m).