Théorie des Langages

TD

Bases de Données

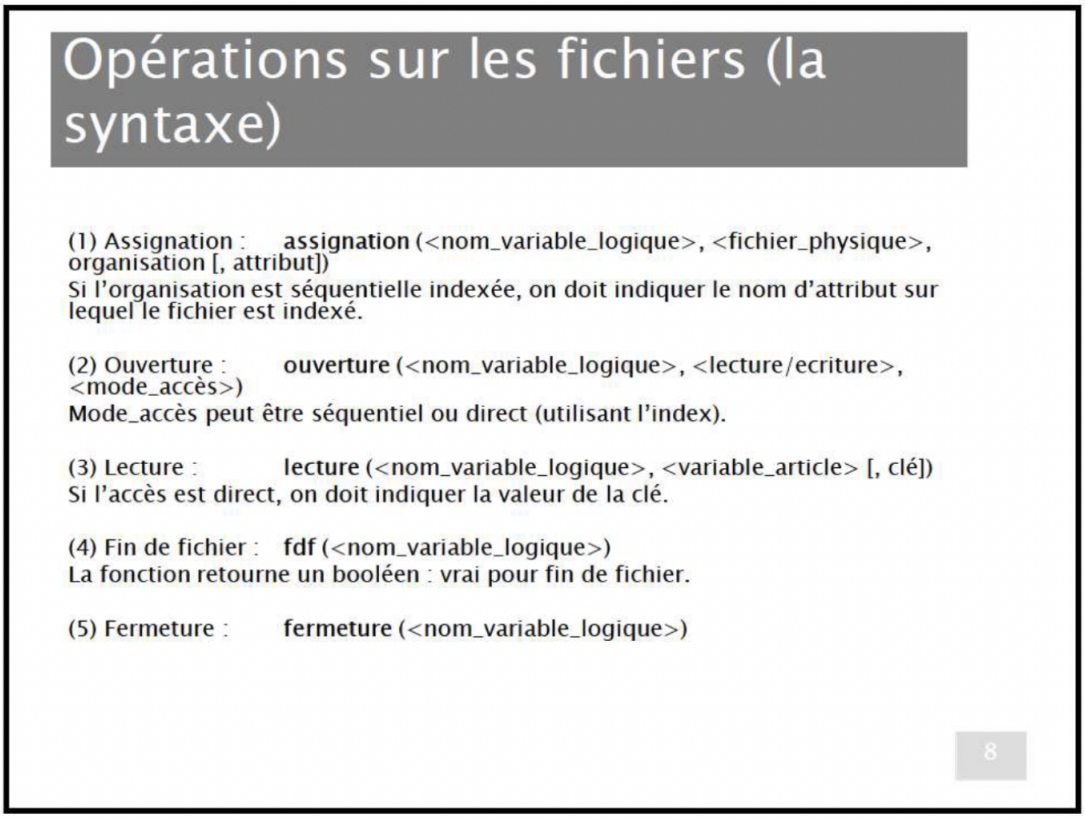
Elana Courtines
courtines.e@gmail.com
https://github.com/irinacake

Séance 1 - 14 novembre 2022
Séance 2 - 15 novembre 2022
Séance 3 - 22 novembre 2022
Séance 4 - 29 novembre 2022

Riad Mokadem - riad.mokadem@univ-tlse3.fr

1 Opérations sur les Fichiers

1.1 Rappels



1.2 Énoncé 1

- Soit 2 fichiers personne.txt et voiture.txt qui contiennent les informations suivantes, respectivement :
  - personne.txt : nom, adresse
  - voiture.txt : num_im, marque, proprio (le nom de la personne)
- Hypothèse :
  - L'organisation du fichier est séquentielle

Note : parce que je trouve leur pseudo-code insupportable, j'ai opté pour un pseudo-code un peu plus proche du Python/C :)

1.3 Exercice 1

Écrivez un programme (pseudo-code) pour trouver la marque de la voicture 31xx31.

```
enregistrement voiture {
    num_im : number(4)
    marque : varchar(20)
    proprietaire : varchar(20)
} v

assignation(fv, "C:\...\voiture.txt", sequentielle)

ouverture(fv, lecture , sequentiel)

while (!FDF(fv)):
    lecture(fv,v)
    if (v.num_im == '31xx31'):
        afficher(v.marque)

fermeture(fv)
```

Cette opération est équivalente à un Select en SQL.

1.4 Exercice 2

Écrivez un programme (pseudo-code) pour trouver les adresses des propriétaires des voitures de marque "Renault".

```
enregistrement voiture {
    num_im : varchar(4)
    marque : varchar(20)
    proprietaire : varchar(20)
} v
enregistrement personne {
    nom : varchar(20)
    adresse : varchar(20)
} p

assignation(fv, "C:\...\voiture.txt", sequentielle)
assignation(fp, "C:\...\personne.txt", sequentielle)

ouverture(fv, lecture , sequentiel)

while (!FDF(fv)):
    lecture(fv,v)
    if (v.marque == "Renault"):
        ouverture(fp, lecture , sequentiel)
        while (!FDF(fp)):
            lecture(fp,p)
            if (v.proprietaire == p.nom):
                afficher(p.adresse)
        fermeture(fp)

fermeture(fv)
```

Cette opération est équivalente à une Jointure en SQL.
Cf. fichiers source pour une version "*avec moins de complexité*".

1.5 Algorithme de sélection - Énoncé 2

Ab*onnement* ***T****éléphonique*

- AB-T (<u>nom</u>, prenom, prof, adr, tel)
- Estimez le nombre de pages disques lues pour la question : numéros de téléphone de Monsieur Dupont :

  1) Sans Index
  2) Avec Index

Donnée	taille
Nom	20c
adresse	38c
prenom	18c
tel	12c
prof	12c
Total	100c

Adresse relative	5c

- Nombre de tuples : 200 000
- 1 page = 1000 caractères

1.6 Sans Index :

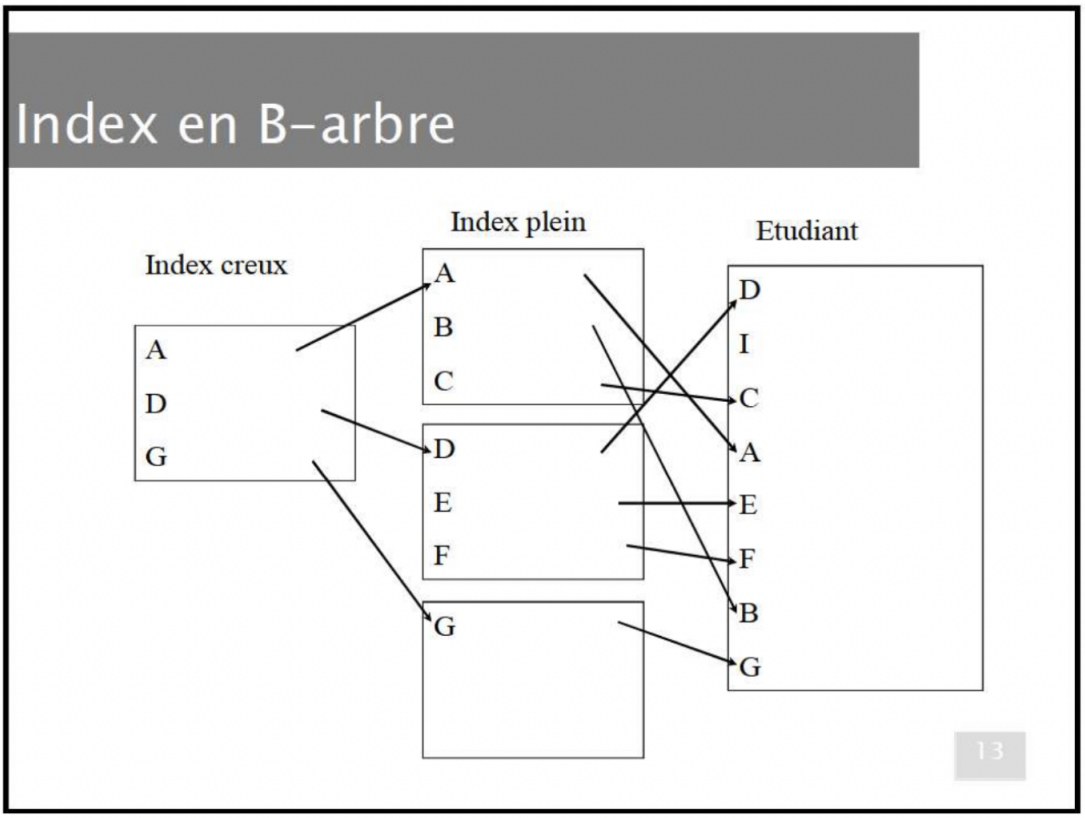
200 000 pages de 100 caractères, d'où :

- $200\ 000 * 100c \rightarrow 20\ 000\ 000c$
- $1000c \rightarrow 1$ page
- nbpages à lire = 20 000 000/1000 = 20 000
- tps de lecture = $20000 * 15ms = 5mins$

Autrement dit, il faut 5 mins pour trouver le numéro de téléphone d'une personne dans un registre de 200 000 entrées.

1.7 Avec Index :

Rappel :



Dans notre cas, le disque contient des entrées de 100c, tandis que les index contiendront des entrées de 20c+5c (nom+référence).

Nombre de pages par index :

- Index plein : $200\,000 * (20c + 5c)/1000 \rightarrow 5\,000$
- Index creux 1^{er} niveau : $5\,000 * (20c + 5c)/1000 \rightarrow 125$
- Index creux 2^{e} niveau : $125 * (20c + 5c)/1000 = 3.125 \rightarrow 4$ (on arrondit toujours au supérieur pour les données qui "débordent")
- Index creux 3^{e} niveau : $4 * (20c + 5c)/1000 < 1 \rightarrow$ on s'arrête

Il faut donc lire 5 pages (dans cet ordre) :

- Index creux 3^e niveau : 1 lecture
- Index creux 2^e niveau : 1 lecture
- Index creux 1^{er} niveau : 1 lecture
- Index plein : 1 lecture
- lecture effective dans le disque : 1 lecture

D'où un temps d'exécution total de $5 * 15 = 75ms$

Afin de démontrer la puissance de l'indexage, on peut simuler ce qu'il se passerait en doublant la quantité d'entrées.

Nombre de pages par index avec 400 000 enregistrements :

- Index plein : $400\ 000 * (20c + 5c)/1000 \rightarrow 10\ 000$
- Index creux 1er niveau : $10\ 000 * (20c + 5c)/1000 \rightarrow 250$
- Index creux 2e niveau : $250 * (20c + 5c)/1000 = 6.25 \rightarrow 7$
- Index creux 3e niveau : $7 * (20c + 5c)/1000 < 1 \rightarrow$ on s'arrête

En doublant la quantité d'entrées, le nombre de lecture de page par requête n'a pas augmenté.

2 Algorithmes de jointure

2.1 Algorithmes à étudier

- Jointure par Produit Catésien (JPC) par bloc
- Jointure par Hachage simple (JHS)
- Jointure par Tri-Fusion (JTF)

2.2 Énoncé

Soient deux relations : R(X, Y) et S(Y, Z)

Nous allons étudier des différents algorithmes en faisant la jointure suivante :
T = Jointure (R, S, R.Y=S.Y) *jointure entre R et S sur Y*

Relation R

X	Y
1	2
4	5
9	3
6	4
8	7
10	9
5	11
2	15
14	8
13	10
11	13
3	12
7	14
12	6

Relation S

Y	Z
2	1
4	2
15	3
1	4
11	5
7	6
6	7
5	8
3	9
12	10
8	11
13	12
9	13
10	14
14	15
16	16

Pour simplifier, nous supposons qu'une page disque contient 2 tuples, quelle que soit la taille d'un tuple.

2.3 Exercice 1

- Donner la trace de l'algorithme de jointure par produit cartésien (par bloc d'une page).
- Hypothèse : il y a 3 pages en mémoire :
  - 1 pour lire R
  - 1 page pour lire S
  - 1 page pour eécrire le résultat dans T

Exemple :

- On prend la page 0 de R $\begin{matrix}1 & 2\\4 & 5\end{matrix}$
- On la compare à la page 0 de S $\begin{matrix}2 & 1\\4 & 2\end{matrix}$
- On vérifie les 4 "matchs" possibles :
  - 1 2 - 2 1
  - 1 2 - 4 2
  - 4 5 - 2 1
  - 4 5 - 4 2
- 1 2 - 2 1 correspond, donc on écrit [1 2 2 1] dans le résultat de la jointure

Résultat :

- 7 lectures de page disque pour R
- 7*8 lectures de page disque pour S
- 7 écritures de page disque pour T

  → 70 lectures/écritures disque

Relation T

1	2	2	1
4	5	5	8
6	4	4	2
9	3	3	9
8	7	7	6
10	9	9	13
2	15	15	3
5	11	11	5
14	8	8	11
13	10	10	14
3	12	12	10
11	13	13	12
12	6	6	7
7	14	14	15

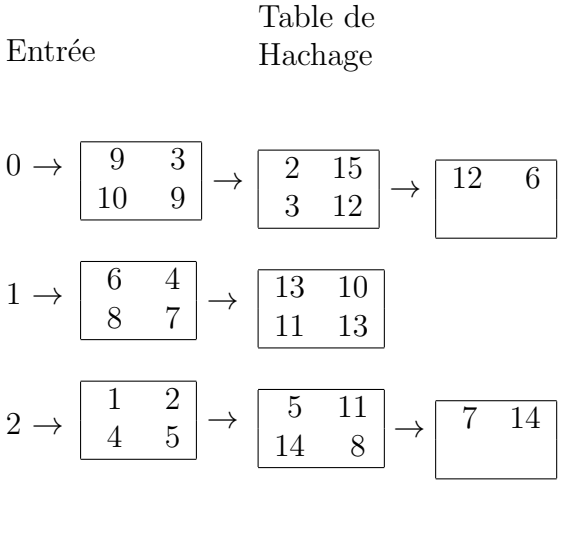
2.4 Exercice 2

- Donner la trace de l'algorithme de jointure par hachage simple ;
- Hypothèse :
  - Il y a 3 entrées dans le tableau de hachage (proposer une fonction de hachages pour faire cela) ;
  - Le tableau de hachage peut rester en mémoire ;
  - En plus, il y a 2 pages en mémoire :
    - 1 pour lire S ;
    - 1 pour écrire le résultat dans T;

Le démarrage de l'algorithme (Phase 1) - "Build"

On crée une table de hachage à partie la relation R. Ici on prendre comme fonction "mod 3".

X	Y	%3
1	2	2
4	5	2
9	3	0
6	4	1
8	7	1
10	9	0
5	11	2
2	15	0
14	8	2
13	10	1
11	13	1
3	12	0
7	14	2
12	6	0



L'algorithme - Phase 2, Probe

Exemples d'exécution :

- Prendre le premier tuple de la page S.P0 = (2,1) ;
- Prendre le mod3 du Y : 2%3 = 2 ;
- Dans la table de hachage, rechercher un tuple pour lequel Y vaut 2 uniquement dans l'entrée "2" de la table ;
- on trouve (1,2), donc on écrit (1,2,2,1) dans T
- ...
- Prendre le deuxième tuple de la page S.P2 = (1,4) ;
- Prendre le mod3 du Y : 1%3 = 1 ;
- Dans la table de hachage, rechercher un tuple pour lequel Y vaut 1 uniquement dans l'entrée "1" de la table ;
- Il n'y a pas de match, on arrête la recherche (sans vérifier les autres entrées)
- ...

Résultat :

- 7 lectures de page disque pour R
- 8 lectures de page disque pour S
- 7 écritures de page disque pour T

  → 22 lectures/écritures disque

2.5 Exercice 3

- Donner la trace de l'algorithme de jointure par tri-fusion ;
- Hypothèse :
  - Pour simplifier, nous supposons que les deux relations sont déjà triées ($\rightarrow$ R' et S') ;
  - Il y a 3 pages en mémoire :
    - 1 pour lire R' ;
    - 1 pour lire S' ;
    - 1 pour écrire le résultat dans T ;

Relation R'

X	Y
1	2
9	3
6	4
4	5
12	6
8	7
14	8
10	9
13	10
5	11
3	12
11	13
7	14
2	15

Relation S'

Y	Z
1	4
2	1
3	9
4	2
5	8
6	7
7	6
8	11
9	13
10	14
11	5
12	10
13	12
14	15
15	3
16	16

Les premières lignes de la trace :

- Lire R'.P0
- Lire S'.P0
  - comparer le tuple (1,2) de R' avec le tuple (1,4) de S' : rien n'est produit.
    Puisque R'.Y (2) ¿ S'.Y(1), avancer le cureur dans S'
  - comparer le tuple (1,2) de R' avec le tuple (2,1) de S' : un tuple (1,2,2,1) est produit $\rightarrow$ ajouter le tuple dans la page en mémoire pour T.
    Puisque R'.Y (2) == S'.Y(2), et que nous supposons qu'il n'y a pas de doublon, avancer le cureur dans R' et S'

Résultat :

- 7 lectures de page disque pour R
- 8 lectures de page disque pour S
- 7 écritures de page disque pour T

  $\rightarrow$ 22 lectures/écritures disque

Relation T

1	2	2	1
9	3	3	9
6	4	4	2
4	5	5	8
12	6	6	7
8	7	7	6
14	8	8	11
10	9	9	13
13	10	10	14
5	11	11	5
3	12	12	10
11	13	13	12
7	14	14	15
2	15	15	3

3 Optimisation logique

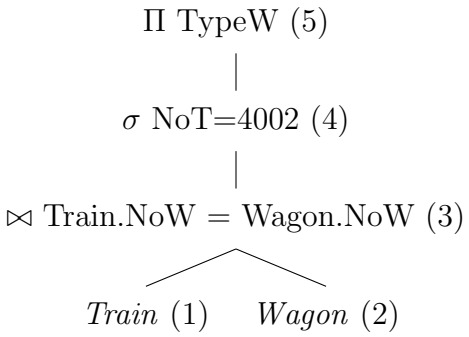
3.1 Énoncé

- Train(<u>NoT, NoW</u>)
- Wagon (<u>NoW</u>, TypeW, PoidsVide, Capa, etat gare)
- Hypothèse : répartition homogène des wagons dans les trains
- Donner 2 arbres algébriques associés à la question : Types de wagon du train 4002

Estimation du volume de données manipulées par chaque arbre algébrique :

Relation	Quantité de tuples	Longueur d'un tuple
Train	60 000	10c
Wagon	200 000	30c

Constituant	Nb val possibles	Longueur en C
NoT	2 000	4c
NoW	200 000	6c
TypeW	200	2c



(1) 60 000 * 10c

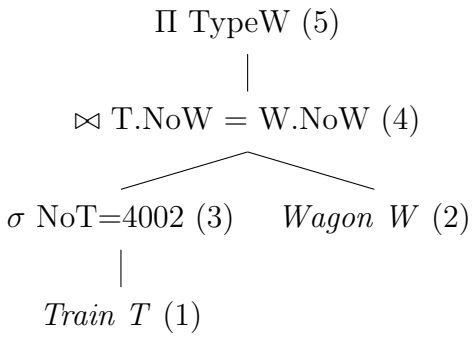
(2) 200 000 * 30c

(3) 60 000 (worst case) * (10c + 30c) = 60 000 * 40c

(4) 60 000 / 2000 * (10c + 30c) = 30 * 40c

(5) 30 * 2c

Au total, 9 001 260 de caractères sont manipulés



(1) 60 000 * 10c

(2) 200 000 * 30c

(3) 60 000 / 2000 * (10c + 30c) = 30 * 10c

(4) 30 * (10c + 30c) = 30 * 40c

(5) 30 * 2c

Au total, 6 601 560 de caractères sont manipulés

3.2 Résultat

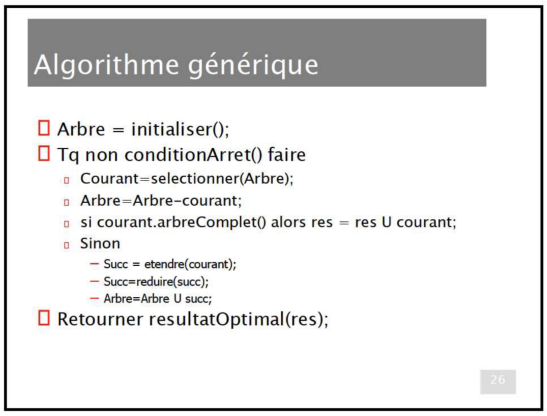
On diminue de $\sim$ 2.4millions (36%) le nombre de caractères manipulés pour cette petite optimisation.

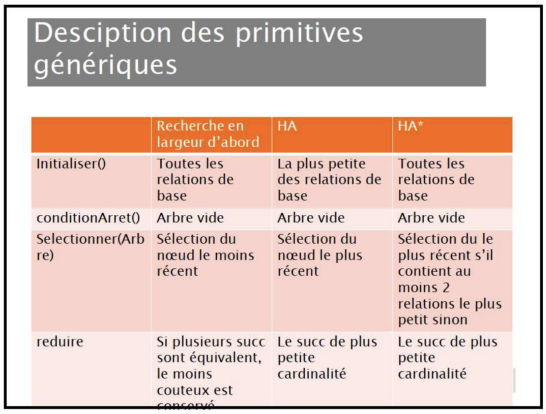
4 Optimisation Physique

Select from R1, R2, R3
where R1.a = R2.b and R2.c = R3.d;

Donner la trace d'exécution de HA (Heuristique Augmentation), avec les hypothèses suivantes:

- R2 est la plus petite relation ;
- R2 est une relation pivot, c'est-à-dire qu'elle est joignable avec R1 et R3 ;
- $|R2 \bowtie R1| < |R2 \bowtie R3|$





étendre = "par quoi on peut faire une jointure"

Stratégie de recherche en profondeur d'abord - HA : $R2 \bowtie R1 \bowtie R3$

0. init : Arbre = {R2} *#la plus petite de relation de base*
1. – courant = selectionner(Arbre) → courant = {R2}
   – arbre = arbre - courant = ∅
   – courant.arbreComplet → false
   – else :
     * succ = etendre(courant) = $\{R2 \bowtie R1, R2 \bowtie R3\}$
     * succ = reduire(succ) = $\{R2 \bowtie R1\}$
     * arbre = arbre ∪ succ = $\{R2 \bowtie R1\}$
2. – courant = selectionner(Arbre) → courant = $\{R2 \bowtie R1\}$
   – arbre = arbre - courant = ∅
   – courant.arbreComplet → false
   – else :
     * succ = etendre(courant) = $\{(R2 \bowtie R1) \bowtie R3\}$
     * succ = reduire(succ) = $\{(R2 \bowtie R1) \bowtie R3\}$
     * arbre = arbre ∪ succ = $\{(R2 \bowtie R1) \bowtie R3\}$
3. – courant = selectionner(Arbre) → courant = $\{(R2 \bowtie R1) \bowtie R3\}$
   – arbre = arbre - courant = ∅
   – courant.arbreComplet → true
   – then : resultat = $\{(R2 \bowtie R1) \bowtie R3\}$

Quelques notes sur les algorithmes :

- Le largeur d'abord produit toujours tous les arbres
- Le HA produit toujours 1 arbre
- Le HA* produit "quelques arbres"

Stratégie de recherche en profondeur d'abord - HA* : $R2 \bowtie R1 \bowtie R3$

0. init : Arbre = {R1, R2, R3}
1. – courant = selectionner(Arbre) → courant = {R2}
   – arbre = arbre - courant = {R1,R3}
   – courant.arbreComplet → false
   – else :
   * succ = etendre(courant) = $\{R2 \bowtie R1, R2 \bowtie R3\}$
   * succ = reduire(succ) = $\{R2 \bowtie R1\}$
   * arbre = arbre ∪ succ = {R1, R3, $R2 \bowtie R1$}
2. – courant = selectionner(Arbre) → courant = $\{R2 \bowtie R1\}$
   – arbre = arbre - courant = {R1,R3}
   – courant.arbreComplet → false
   – else :
   * succ = etendre(courant) = $\{(R2 \bowtie R1) \bowtie R3\}$
   * succ = reduire(succ) = $\{(R2 \bowtie R1) \bowtie R3\}$
   * arbre = arbre ∪ succ = $\{R1, R3, (R2 \bowtie R1) \bowtie R3\}$
3. – courant = selectionner(Arbre) → courant = $\{(R2 \bowtie R1) \bowtie R3\}$
   – arbre = arbre - courant = {R1,R3}
   – courant.arbreComplet → true
   – then : resultat = $\{(R2 \bowtie R1) \bowtie R3\}$
4. – courant = selectionner(Arbre) → courant = {R1} *3e Hypothèse* $|R1| < |R3|$
   – Arbre = arbre - courant = {R3}
   – courant.arbreComplet → false
   – else :
   * succ = etendre(courant) = $\{R1 \bowtie R2\}$
   * succ = reduire(succ) = $\{R1 \bowtie R2\}$
   * arbre = arbre ∪ succ = $\{R3, R1 \bowtie R2\}$

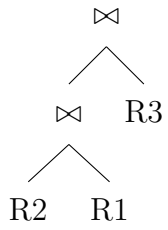
5. – courant = selectionner(Arbre) → courant = $\{R1 \bowtie R2\}$
   – arbre = arbre - courant = {R3}
   – courant.arbreComplet → false
   – else :
     * succ = etendre(courant) = $\{(R1 \bowtie R2) \bowtie R3\}$
     * succ = reduire(succ) = $\{(R1 \bowtie R2) \bowtie R3\}$
     * arbre = arbre ∪ succ = $\{R3, (R2 \bowtie R1) \bowtie R3\}$

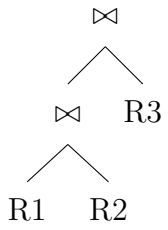
6. – courant = selectionner(Arbre) → courant = $\{(R1 \bowtie R2) \bowtie R3\}$
   – arbre = arbre - courant = {R3}
   – courant.arbreComplet → true
   – then : resultat = $\{(R2 \bowtie R1) \bowtie R3 \cup (R1 \bowtie R2) \bowtie R3\}$

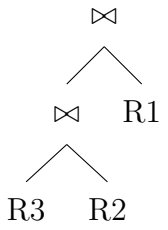
7. – courant = selectionner(Arbre) → courant = {R3}
   – arbre = arbre - courant = ∅
   – courant.arbreComplet → false
   – else :
     * succ = etendre(courant) = $\{R3 \bowtie R2\}$
     * succ = reduire(succ) = $\{R3 \bowtie R2\}$
     * arbre = arbre ∪ succ = $\{R3 \bowtie R2\}$

8. – courant = selectionner(Arbre) → courant = $\{R3 \bowtie R2\}$
   – arbre = arbre - courant = ∅
   – courant.arbreComplet → false
   – else :
     * succ = etendre(courant) = $\{(R3 \bowtie R2) \bowtie R1\}$
     * succ = reduire(succ) = $\{(R3 \bowtie R2) \bowtie R1\}$
     * arbre = arbre ∪ succ = $\{(R3 \bowtie R2) \bowtie R1\}$

9. – courant = selectionner(Arbre) → courant = $\{(R3 \bowtie R2) \bowtie R1\}$
   – arbre = arbre - courant = ∅
   – courant.arbreComplet → true
   – then : resultat = $\{(R2 \bowtie R1) \bowtie R3 \cup (R1 \bowtie R2) \bowtie R3, (R3 \bowtie R2) \bowtie R1\}$







Stratégie de recherche en largeur d'abord : $R2 \bowtie R1 \bowtie R3$

0. init : Arbre = {R1, R2, R3}
1. – courant = selectionner(Arbre) → courant = {R1}
   – arbre = arbre - courant = {R2,R3}
   – courant.arbreComplet → false
   – else :
     * succ = etendre(courant) = $\{R1 \bowtie R2\}$
     * succ = reduire(succ) = $\{R1 \bowtie R2\}$
     * arbre = arbre ∪ succ = {R2, R3, $R1 \bowtie R2$}
2. – courant = selectionner(Arbre) → courant = $\{R2\}$
   – Arbre = arbre - courant = $\{R3, R1 \bowtie R2\}$
   – courant.arbreComplet → false
   – else :
     * succ = etendre(courant) = $\{R2 \bowtie R1, R2 \bowtie R3\}$
     * succ = reduire(succ) = $\{R2 \bowtie R1, R2 \bowtie R3\}$
     * arbre = arbre ∪ succ = $\{R3, R1 \bowtie R2, R2 \bowtie R1, R2 \bowtie R3\}$
3. – courant = selectionner(Arbre) → courant = $\{R3\}$
   – Arbre = arbre - courant = $\{R1 \bowtie R2, R2 \bowtie R1, R2 \bowtie R3\}$
   – courant.arbreComplet → false
   – else :
     * succ = etendre(courant) = $\{R3 \bowtie R2\}$
     * succ = reduire(succ) = $\{R3 \bowtie R2\}$
     * arbre = arbre ∪ succ = $\{R1 \bowtie R2, R2 \bowtie R1, R2 \bowtie R3, R3 \bowtie R2\}$
4. arbre = arbre ∪ succ = $\{R2 \bowtie R1, R2 \bowtie R3, R3 \bowtie R2, (R1 \bowtie R2) \bowtie R3\}$
5. arbre = arbre ∪ succ = $\{R2 \bowtie R3, R3 \bowtie R2, (R1 \bowtie R2) \bowtie R3, (R2 \bowtie R1) \bowtie R3\}$
6. arbre = arbre ∪ succ = $\{R3 \bowtie R2, (R1 \bowtie R2) \bowtie R3, (R2 \bowtie R1) \bowtie R3, (R2 \bowtie R3) \bowtie R1\}$
7. arbre = arbre ∪ succ = $\{(R1 \bowtie R2) \bowtie R3, (R2 \bowtie R1) \bowtie R3, (R2 \bowtie R3) \bowtie R1, (R3 \bowtie R2) \bowtie R1\}$
8. – courant = selectionner(Arbre) → courant = $\{(R1 \bowtie R2) \bowtie R3\}$
   – arbre = arbre - courant = $\{(R2 \bowtie R1) \bowtie R3, (R2 \bowtie R3) \bowtie R1, (R3 \bowtie R2) \bowtie R1\}$
   – courant.arbreComplet → true
   – then : resultat = $\{(R1 \bowtie R2) \bowtie R3\}$
9. – courant = selectionner(Arbre) → courant = $\{(R2 \bowtie R1) \bowtie R3\}$
   – arbre = arbre - courant = $\{(R2 \bowtie R3) \bowtie R1, (R3 \bowtie R2) \bowtie R1\}$
   – courant.arbreComplet → true
   – then : resultat = $\{(R1 \bowtie R2) \bowtie R3, (R2 \bowtie R1) \bowtie R3\}$

10. – courant = selectionner(Arbre) → courant = $\{(R2 \bowtie R3) \bowtie R1\}$
    – arbre = arbre - courant = $\{(R3 \bowtie R2) \bowtie R1\}$
    – courant.arbreComplet → true
    – then : resultat = $\{(R1 \bowtie R2) \bowtie R3, (R2 \bowtie R1) \bowtie R3, (R2 \bowtie R3) \bowtie R1\}$

11. – courant = selectionner(Arbre) → courant = $\{(R3 \bowtie R2) \bowtie R1\}$
    – arbre = arbre - courant = $\varnothing$
    – courant.arbreComplet → true
    – then : resultat = $\{(R1 \bowtie R2) \bowtie R3, (R2 \bowtie R1) \bowtie R3, (R2 \bowtie R3) \bowtie R1, (R3 \bowtie R2) \bowtie R1\}$