Indexation

Structures arborescentes et hachage

P. Rigaux

CNAM Paris

May 3, 2011

Sommaire

- Concepts de base pour l'indexation : index denses, non denses, index multi-niveaux
- Arbre-B, structure et algorithmes
- Hachage
- Index *bitmap*
- Exemples avec Oracle

Pourquoi l'indexation

En l'absence d'un index, seules solutions :

- Parcours séquentiel (complexité linéaire)
- Recherche par dichotomie si fichier trié (complexité logarithmique)

Avec un index :

- Parcours de l'index, puis *accès direct à l'enregistrement*
- Mais attention : mises à jour plus coûteuses !!

Exemple 1 : une table

titre	année	...	titre	année	...
Vertigo	1958	...	Annie Hall	1977	...
Brazil	1984	...	Jurasic Park	1992	...
Twin Peaks	1990	...	Metropolis	1926	...
Underground	1995	...	Manhattan	1979	...
Easy Rider	1969	...	Reservoir Dogs	1992	...
Psychose	1960	...	Impitoyable	1992	...
Greystoke	1984	...	Casablanca	1942	...
Shining	1980	...	Smoke	1995	...

Exemple 2

La table des films, avec :

- 1 000 000 (un million) de films
- Un enregistrement = 120 octets
- Un bloc = 4K => 34 enregistrements par bloc
- environ 30 000 blocs, 120 Mo

Clé de recherche, opérations

Clé de recherche = une *liste* d'un ou plusieurs attributs sur lesquels portent des critères.

Types de recherche :

- Recherche par clé : on associe une valeur à chaque attribut de la clé
- Recherche par intervalle : on associe un intervalle de valeurs à chaque attribut de la clé
- Recherche par préfixe : on associe une valeur à un préfixe de la clé

Exemples

Clés de recherche :

- Le titre du film (c'est aussi la clé primaire)
- l'année du film
- le titre plus l'année

Opérations :

- Rechercher *Vertigo*
- Rechercher les films parus entre 1960 et 1975
- Rechercher les films commençant par 'V'

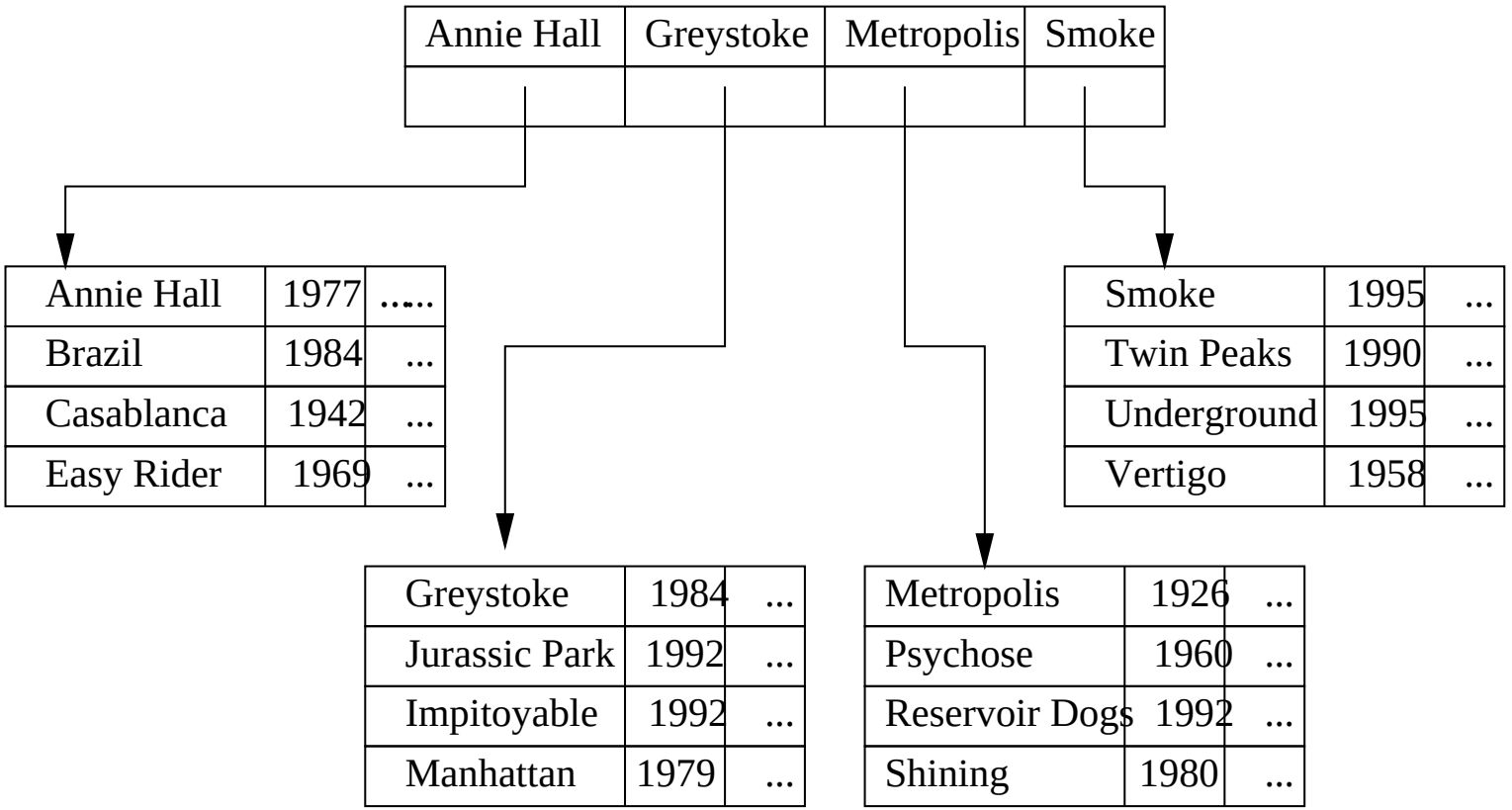
Index non-dense

Hypothèse : fichier trié sur la clé

L'index :

- C'est un fichier !
- Les enregistrements (ou *entrées*) de l'index sont de la forme `[valeur, Addr]`
- Le fichier est trié sur `valeur`
- Un seul enregistrement par bloc du fichier de données est représenté dans le fichier d'index

Exemple


Annie Hall
Greystoke
Metropolis
Smoke
Annie Hall 1977 .....
Brazil 1984 ...
Casablanca 1942 ...
Easy Rider 1969 ...
Greystoke 1984 ...
Jurassic Park 1992 ...
Impitoyable 1992 ...
Manhattan 1979 ...
Metropolis 1926 ...
Psychose 1960 ...
Reservoir Dogs 1992 ...
Shining 1980 ...
Smoke 1995 ...
Twin Peaks 1990 ...
Underground 1995 ...
Vertigo 1958 ...

Opérations

Recherches :

- **Par clé** : par dichotomie sur l'index
- **Par intervalle** : recherche de la borne inférieure, accès au fichier, parcours séquentiel (exemple [J,P])
- **Par préfixe** : cas particulier de la recherche par intervalle

Exemple concret

Sur notre fichier de 120 Mo

- En supposant qu'un titre occupe 20 octets, une adresse 8 octets
- Taille de l'index : $29\,142 * (20 + 8) = 815\,976$ octets

Beaucoup plus petit que le fichier ! Avantage principal sur la recherche par dichotomie.
Problème : maintenir l'ordre sur le fichier **et** sur l'index.

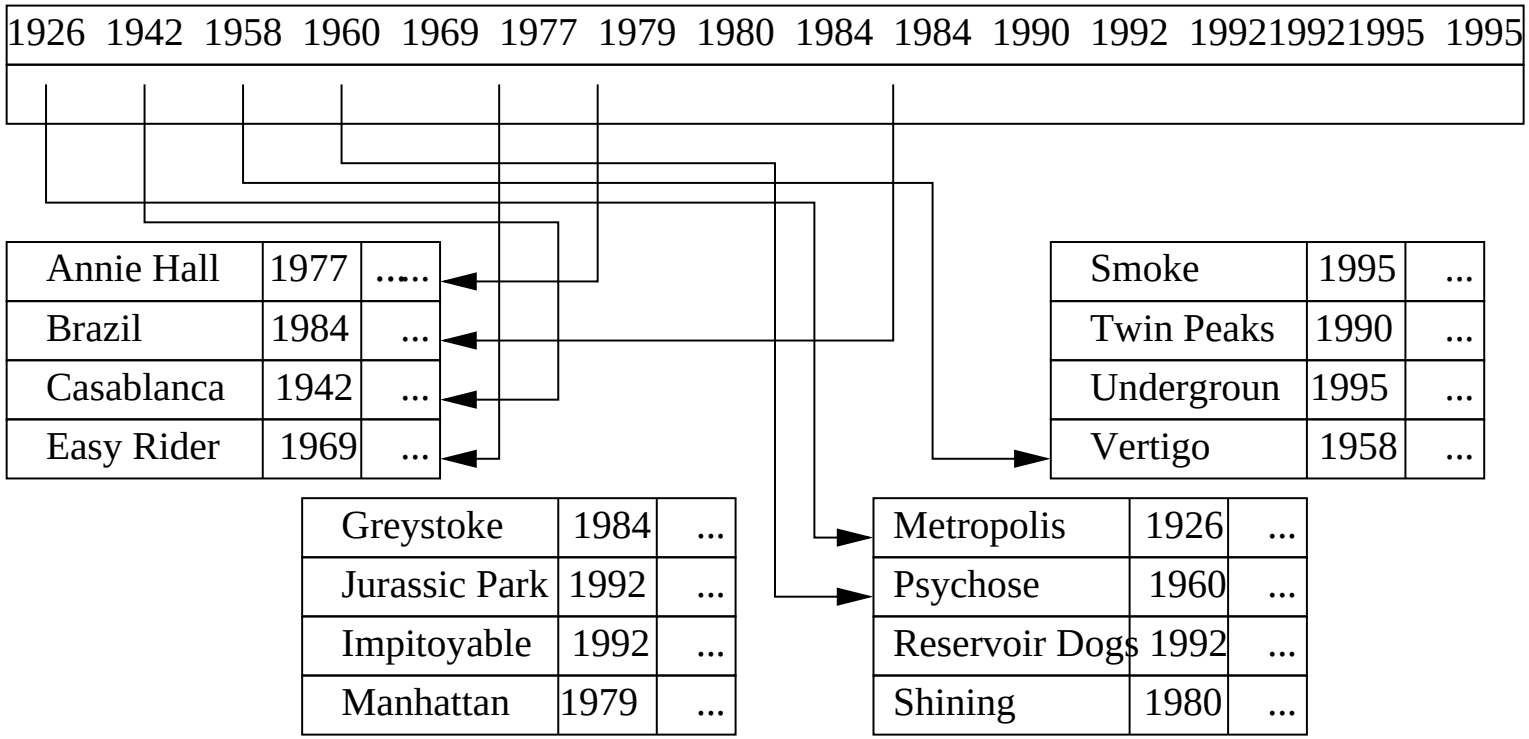
Index dense

Si on veut indexer un fichier non trié ?

- L'index est toujours un fichier
- Il est toujours trié sur la clé
- **Tous les enregistrements sont représentés**

L'index est dit **dense**.

Exemple


1926 1942 1958 1960 1969 1977 1979 1980 1984 1984 1990 1992 199219921995 1995
Annie Hall 1977 .....
Brazil 1984 ...
Casablanca 1942 ...
Easy Rider 1969 ...
Smoke 1995 ...
Twin Peaks 1990 ...
Undergroun 1995 ...
Vertigo 1958 ...
Greystoke 1984 ...
Jurassic Park 1992 ...
Impitoyable 1992 ...
Manhattan 1979 ...
Metropolis 1926 ...
Psychose 1960 ...
Reservoir Dogs 1992 ...
Shining 1980 ...

Exemple concret

Sur notre fichier de 120 Mo

- Une année = 4 octets, une adresse 8 octets
- Taille de l'index : $1\,000\,000 * (4 + 8) = 12$ Mo

Seulement dix fois plus petit que le fichier : la taille d'un index ne doit pas être négligée.

Opérations

Recherches :

- Par clé : comme sur un index non-dense
- Par intervalle (exemple [1950, 1979]) :
  - recherche, dans l'index de la borne inférieure
  - parcours séquentiel *dans l'index*
  - à chaque valeur : accès au fichier de données

Coût beaucoup plus élevé.

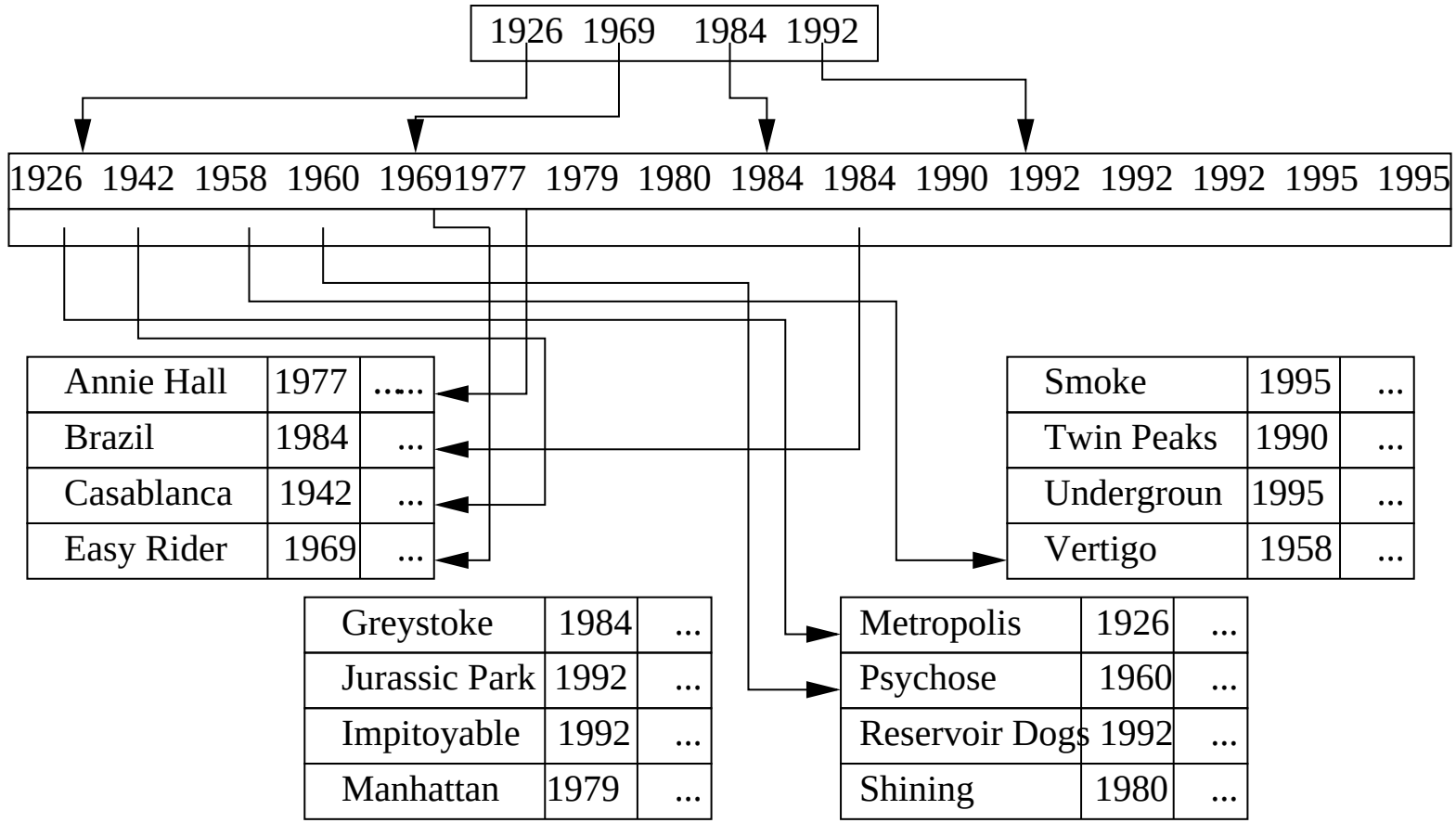
Index multi-niveaux

Si l'index est trop gros ? On l'indexe à son tour.

- **Essentiel** : l'index est trié, donc on peut l'indexer par un second niveau **non-dense**
- Sinon ça ne servirait à rien (pourquoi ?)
- Donc dès le second niveau on diminue drastiquement la taille.
- On peut continuer jusqu'à un niveau avec une seule page.

On obtient une **structure séquentielle indexée**

Exemple


1926 1969 1984 1992
1926 1942 1958 1960 19691977 1979 1980 1984 1984 1990 1992 1992 1992 1995 1995
Annie Hall 1977 .....
Brazil 1984 ...
Casablanca 1942 ...
Easy Rider 1969 ...
Smoke 1995 ...
Twin Peaks 1990 ...
Undergroun 1995 ...
Vertigo 1958 ...
Greystoke 1984 ...
Jurassic Park 1992 ...
Impitoyable 1992 ...
Manhattan 1979 ...
Metropolis 1926 ...
Psychose 1960 ...
Reservoir Dogs 1992 ...
Shining 1980 ...

Remarques/vocabulaire

- Index primaire, secondaire ?
  - ne veut pas dire grand chose
- Index *plaçant* ?
  - index qui détermine la position des données.
- Il ne peut y avoir qu'un index non-dense sur un fichier (pourquoi ?)
- Il peut y avoir autant d'index dense que l'on veut.

Arbre-B

Aboutissement des structures d'index basées sur **l'ordre** des données

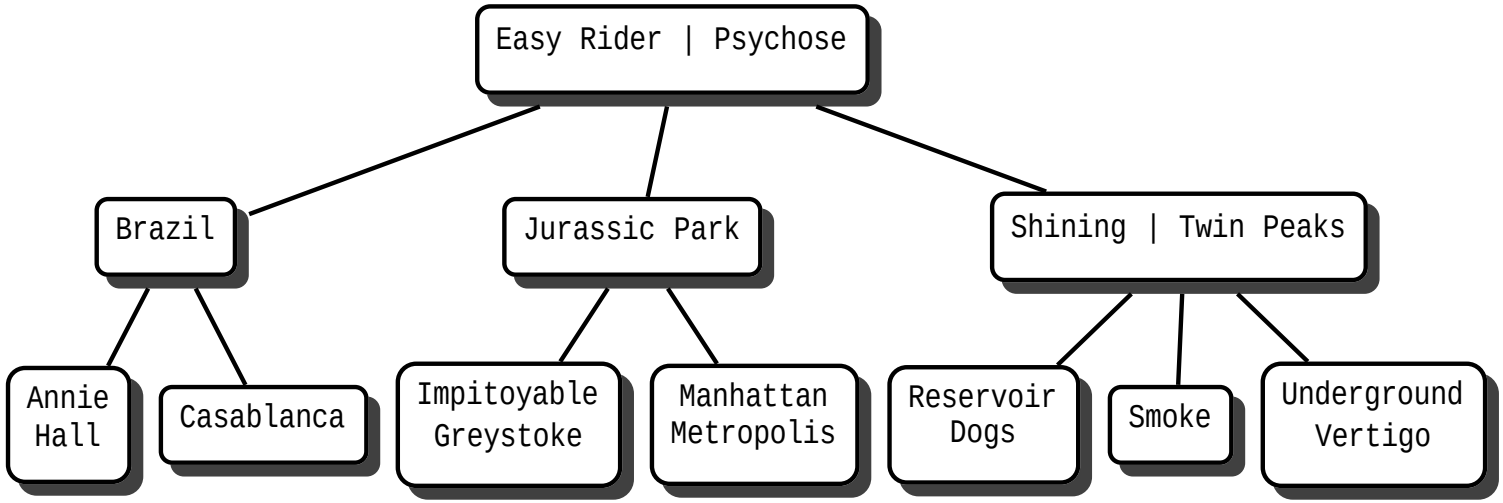
- c'est un arbre équilibré
- chaque nœud est un index local
- il se réorganise dynamiquement

Utilisé universellement !

Comparable aux séquentiel indexé, mais évite d'avoir à maintenir des fichiers triés.

Exemple d'un arbre B

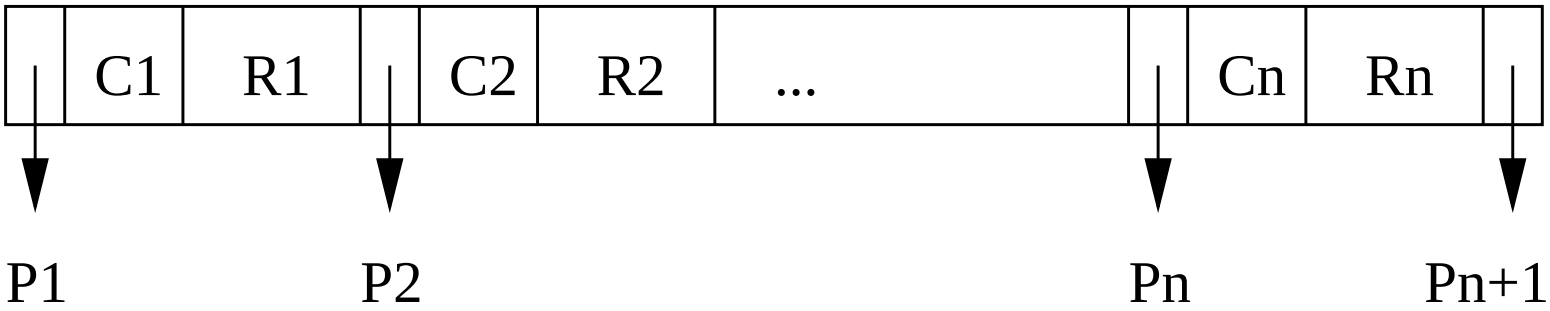
Un arbre, avec trois niveaux, une racine, les enregistrements répartis dans les nœuds.



Les recherches sont guidées par l'ordre dans chaque nœud.

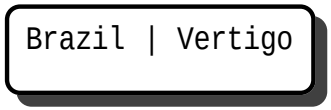
Nœud d'un arbre B

Un nœud est un index local, les enregistrements servant de clé, intercalés avec des pointeurs.



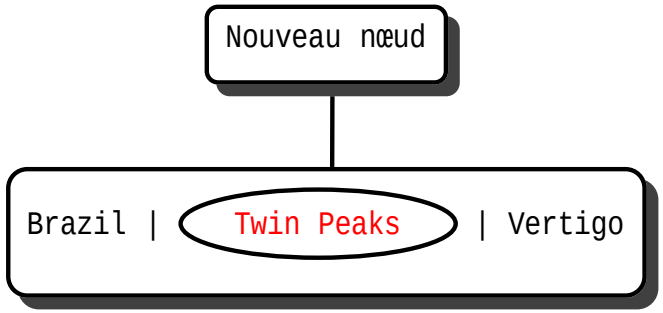
Le sous-arbre pointé par P_2 contient tous les enregistrements dont la clé est comprise entre C_1 et C_2.

Construction de l'arbre-B



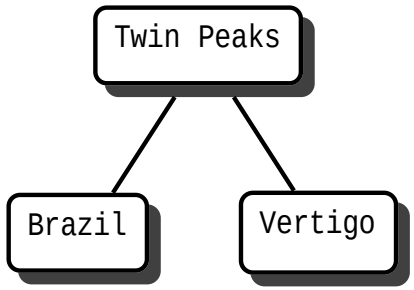
On suppose qu'on ne peut mettre que deux enregistrements par bloc. Voici le premier bloc de l'arbre

Construction de l'arbre-B



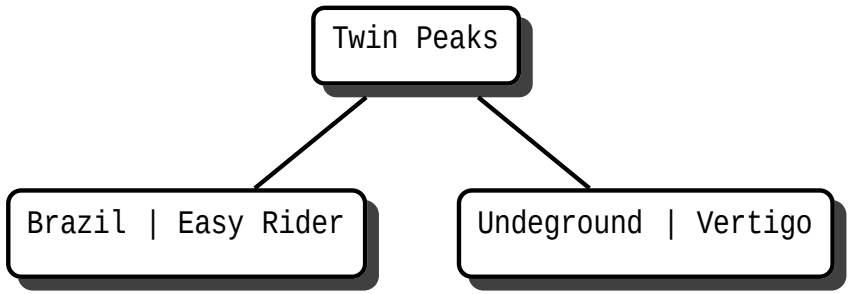
Ici le bloc drde. On prend l'ment du milieu pour le monter dans un nouveau nœud

Construction de l'arbre-B



Aprrtition des enregistrements.

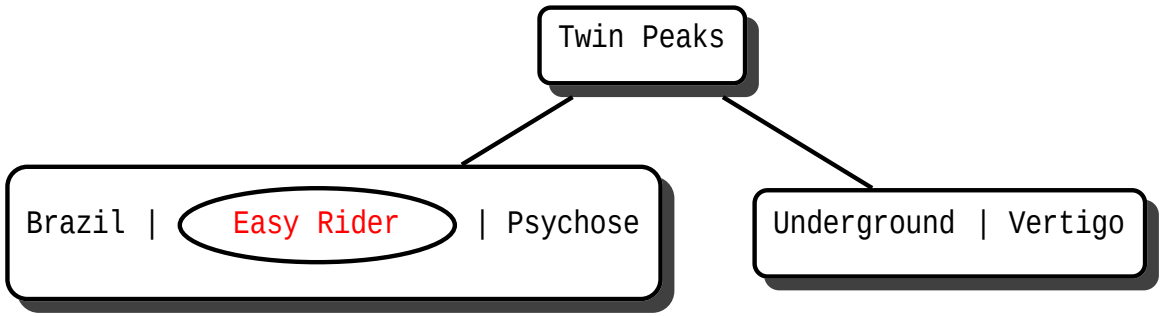
Construction de l'arbre-B



Aprnsertion de Underground et Easy Rider

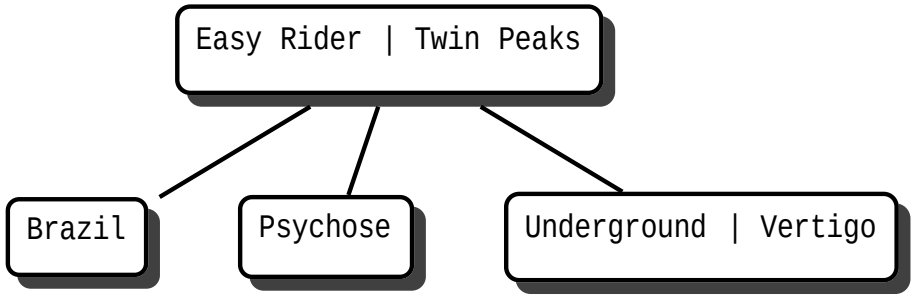
Maintenant il faut mettre Psychose

Construction de l'arbre-B



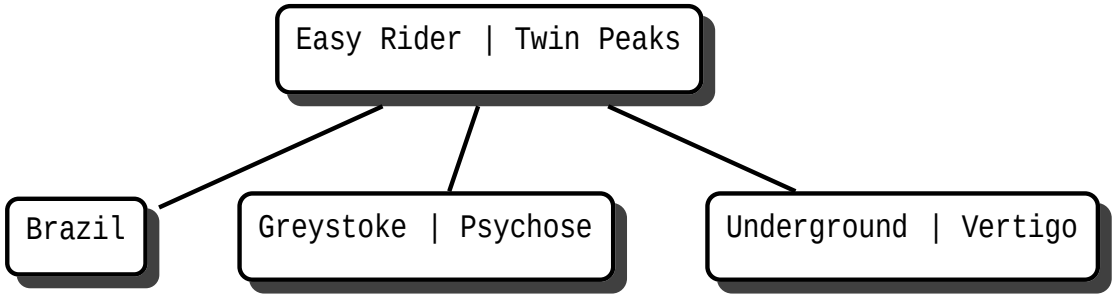
Insertion de Psychose : le nœud drde, et l'ment du milieu doit monter.

Construction de l'arbre-B



Apremonte Easy Rider, et rrtition des autres enregistrements

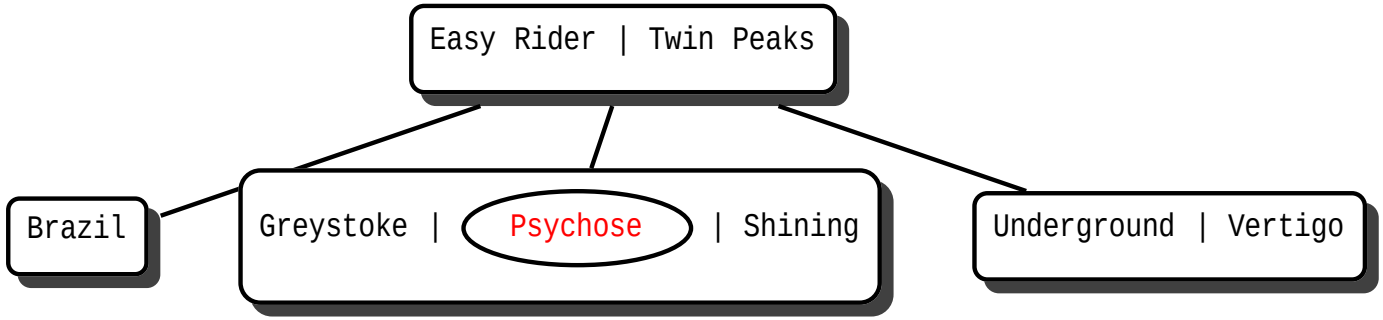
Construction de l'arbre-B



On ins Greystoke : arche.

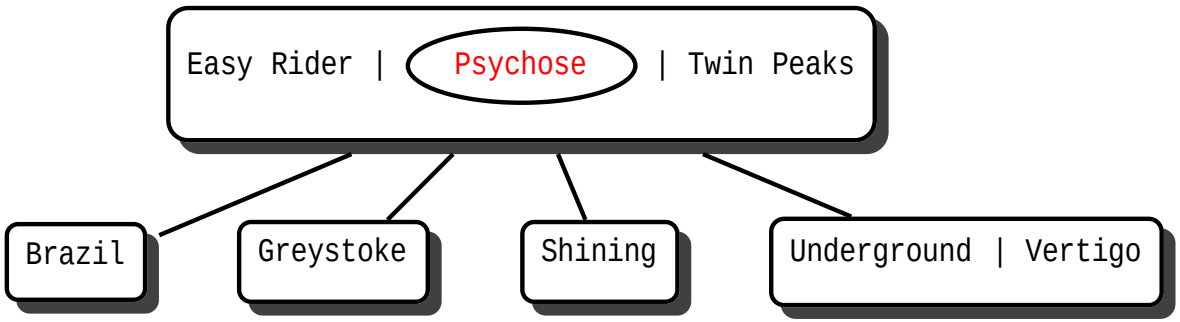
Maintenant il faut mettre Shining

Construction de l'arbre-B



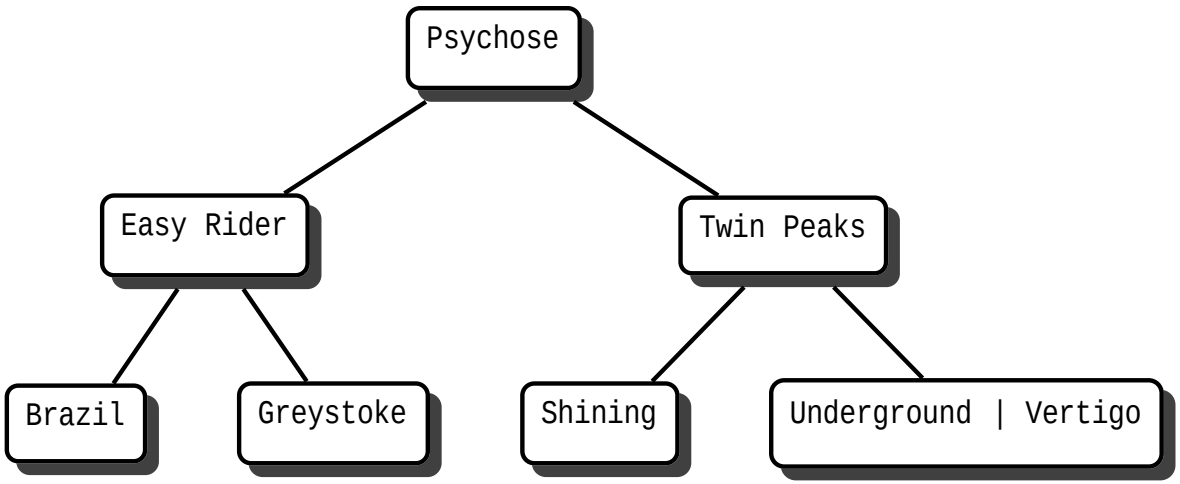
On ins Shining : rde

Construction de l'arbre-B



On a remontychose, mais maintenant c'est la racine qui drde !!

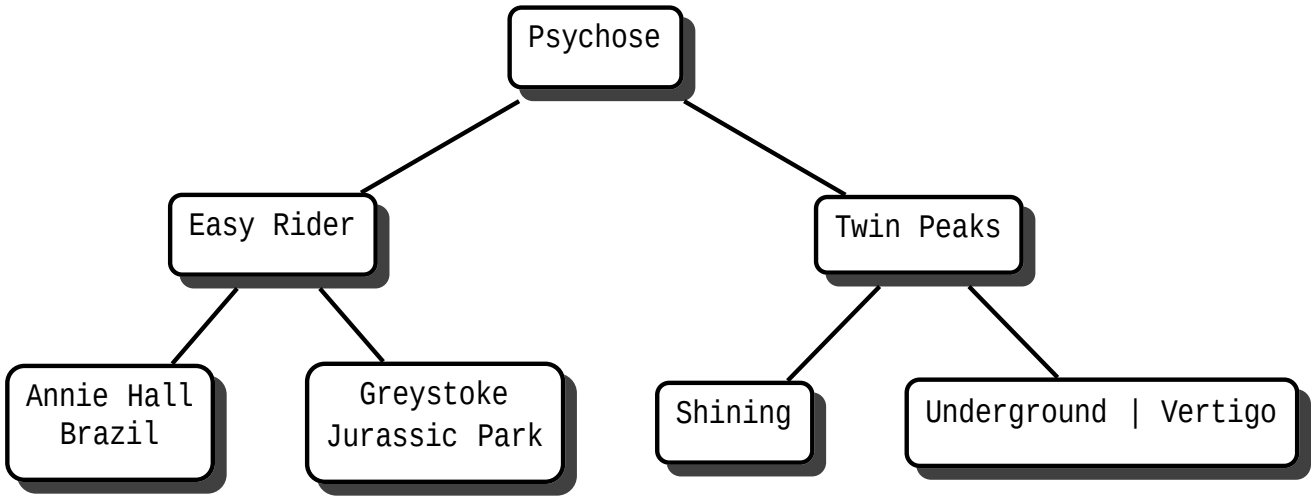
Construction de l'arbre-B



Alors on a crne nouvelle racine.

Il faut insr Annie Hall et Jurassic Park

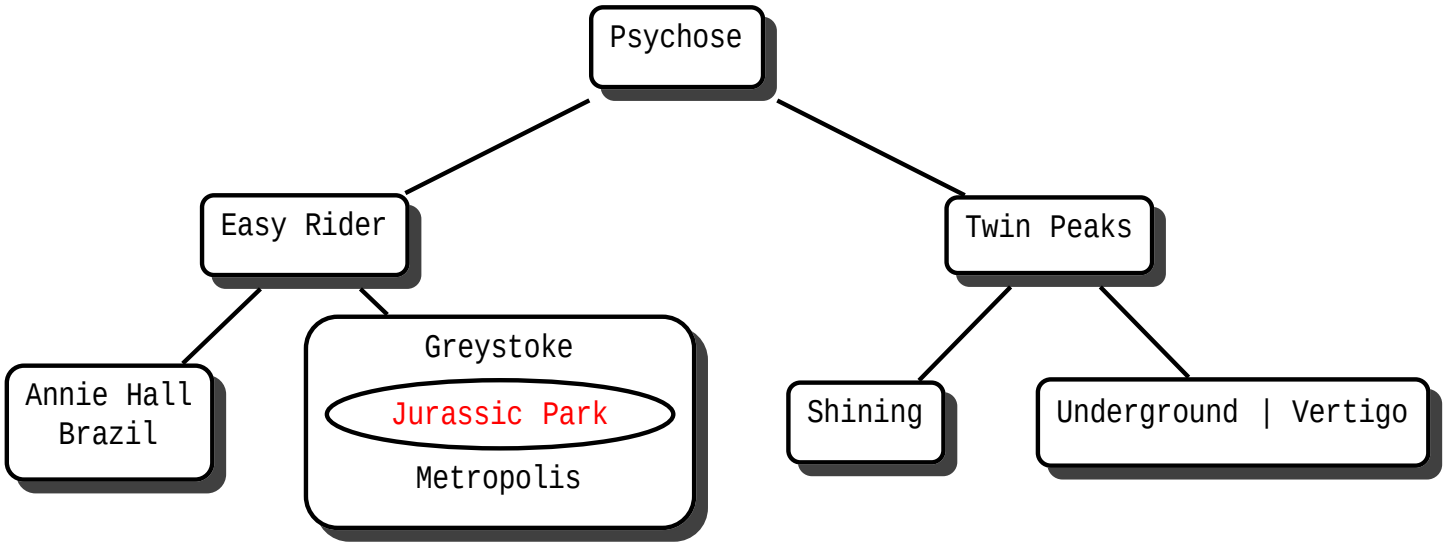
Construction de l'arbre-B



On a ins sans probl Annie Hall et Jurassic Park.

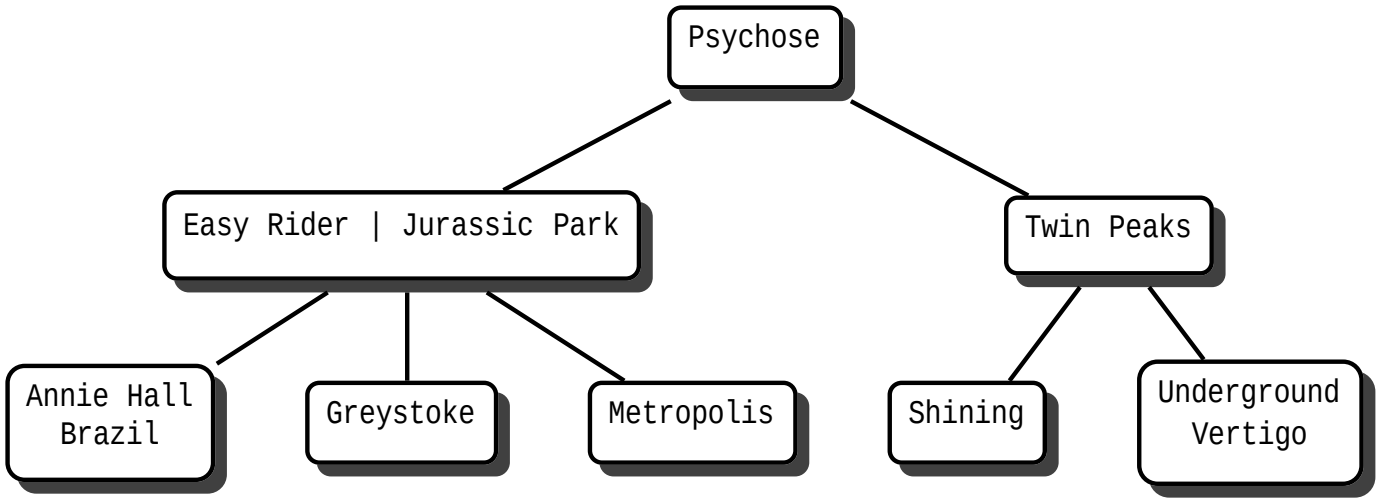
Maintenant il faut insr Metropolis

Construction de l'arbre-B



Ca drde encore

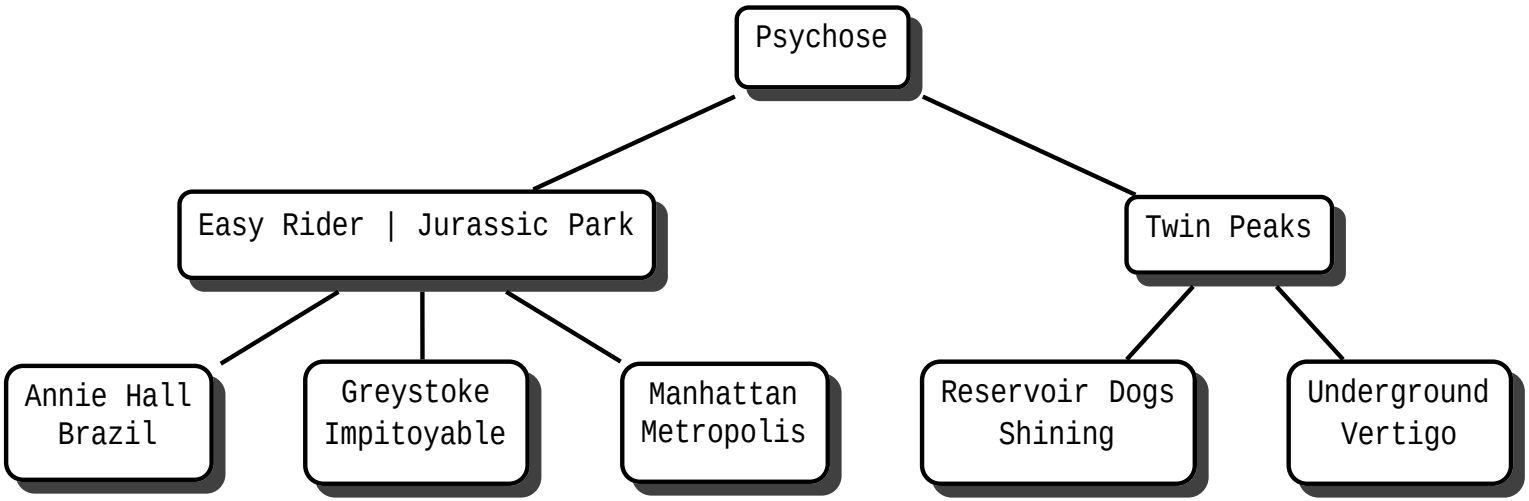
Construction de l'arbre-B



Aprganisation.

Il faut insr Manhattan, Reservoir Dogs et Impitoyable.

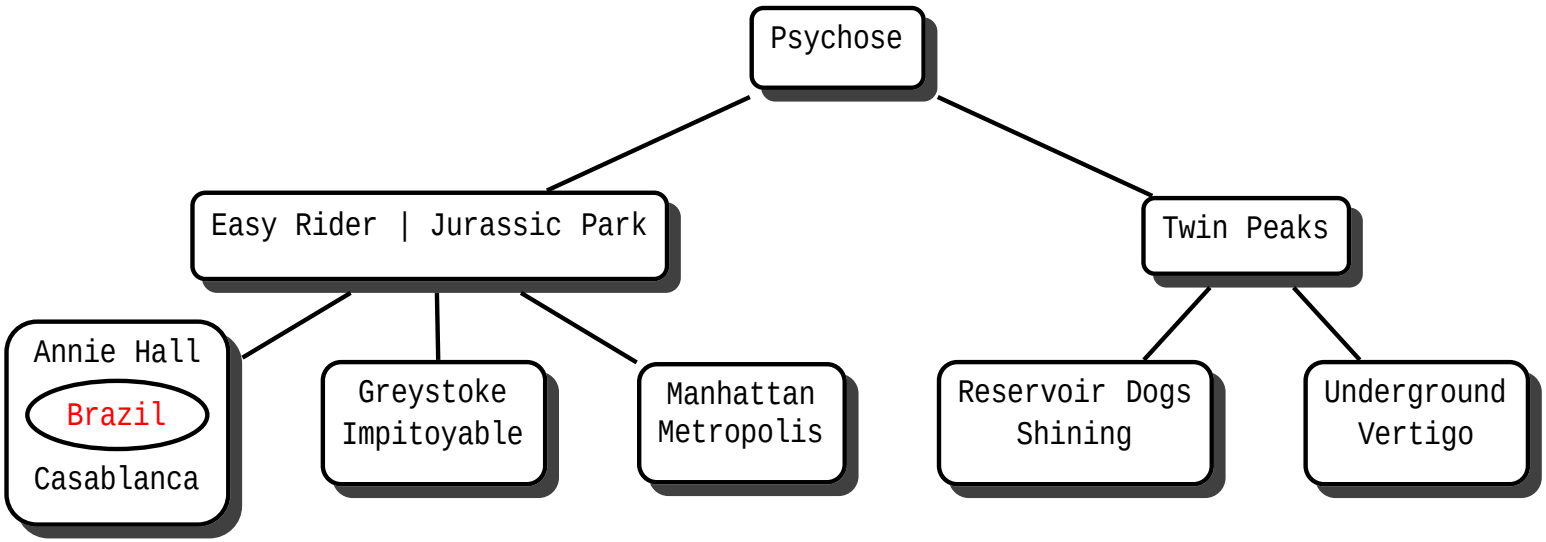
Construction de l'arbre-B



On a ins Manhattan, Reservoir Dogs et Impitoyable.

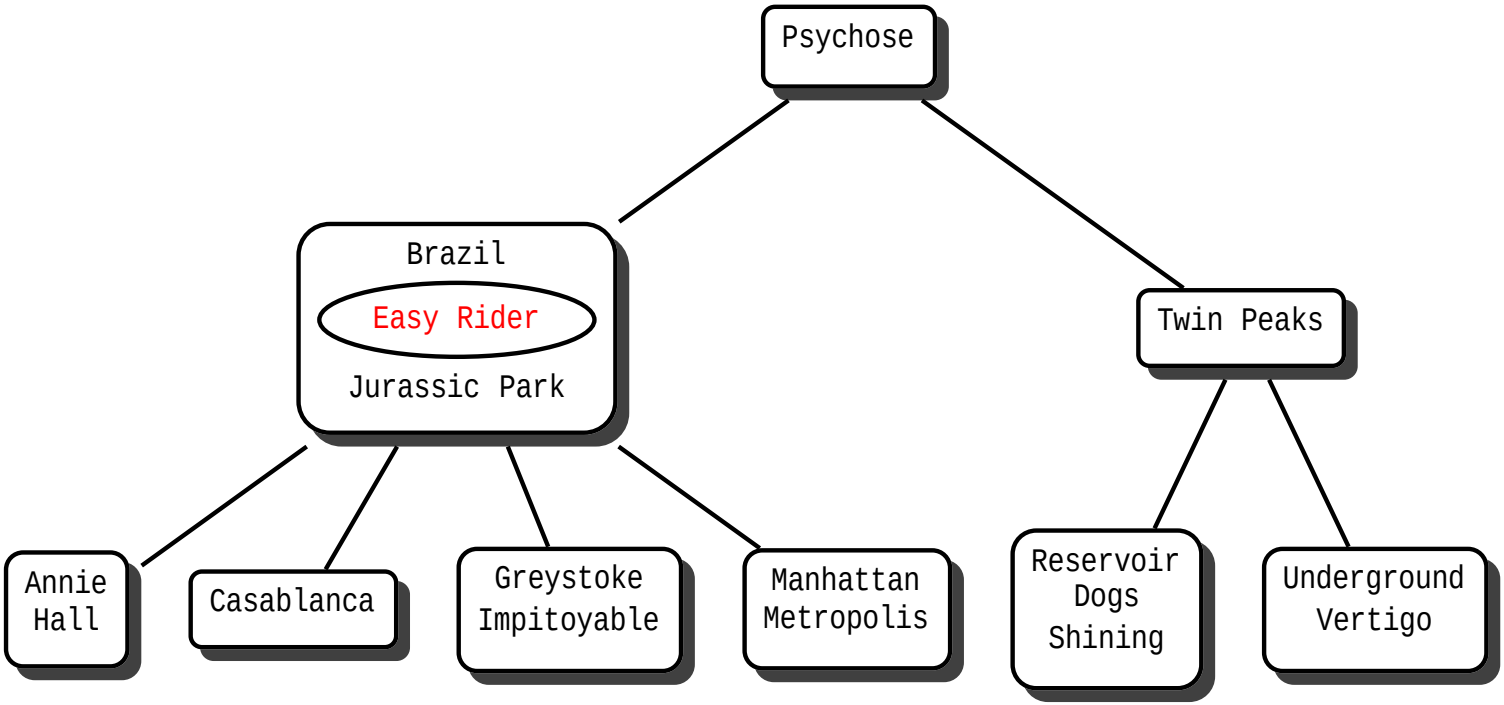
Au tour de Casablanca.

Construction de l'arbre-B



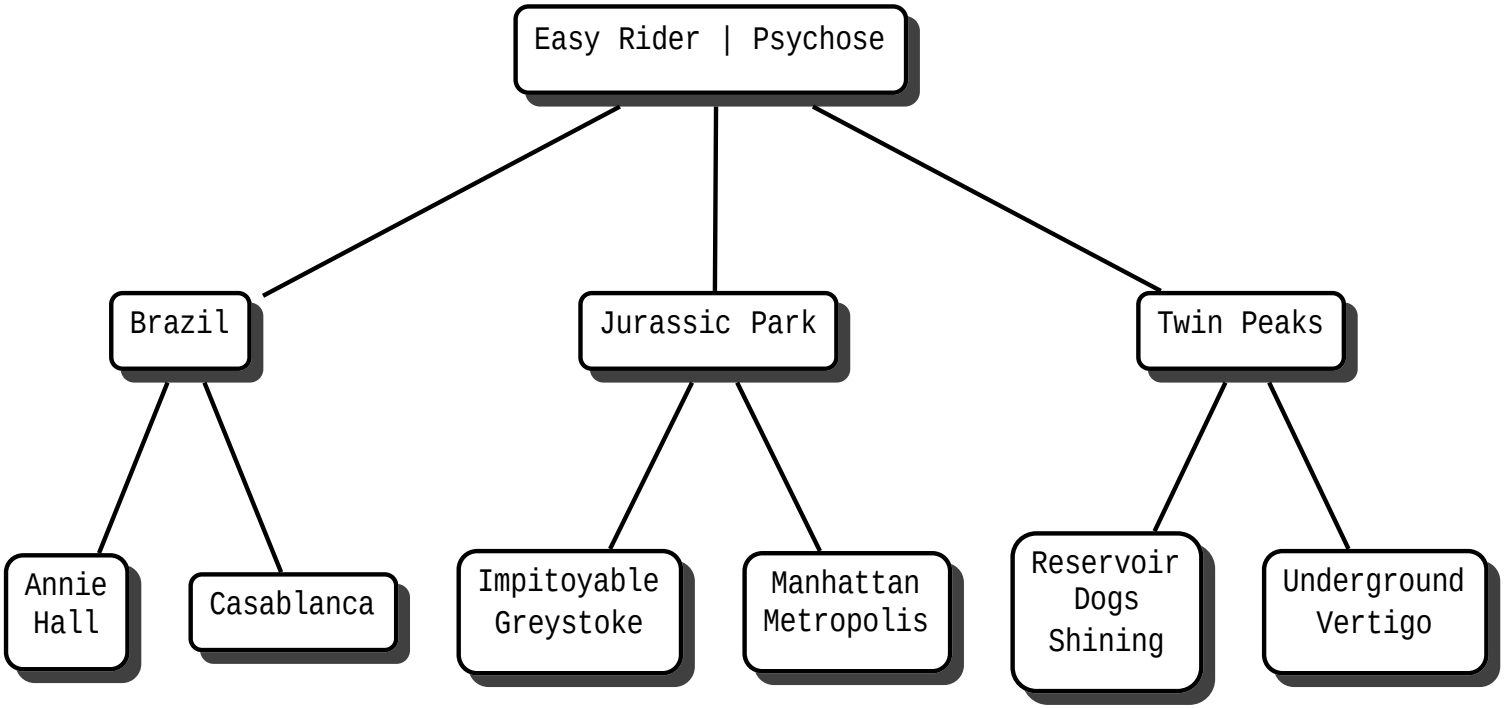
Et rde.

Construction de l'arbre-B



On a remontun cran. Pas encore

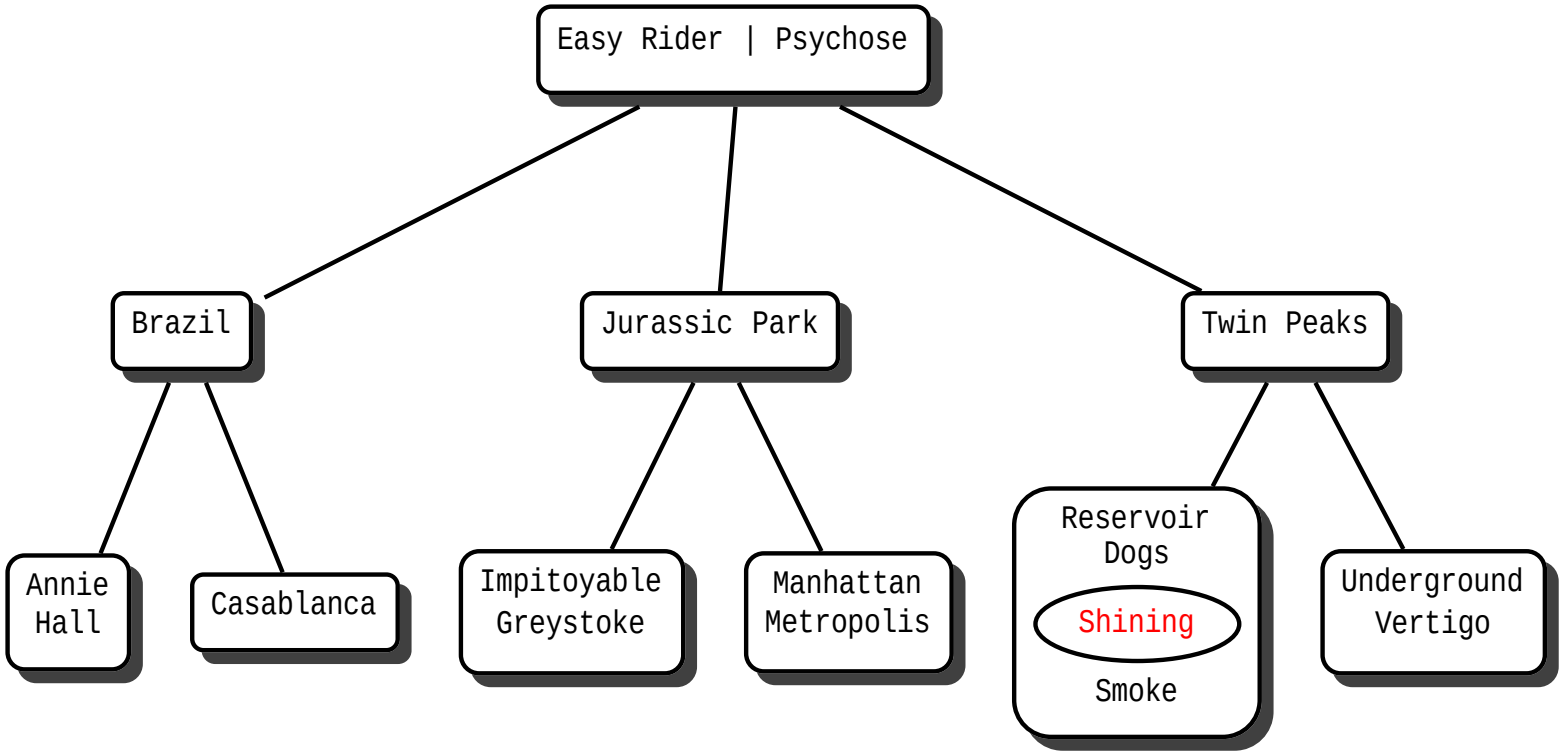
Construction de l'arbre-B



Finalement on a ins dans la racine.

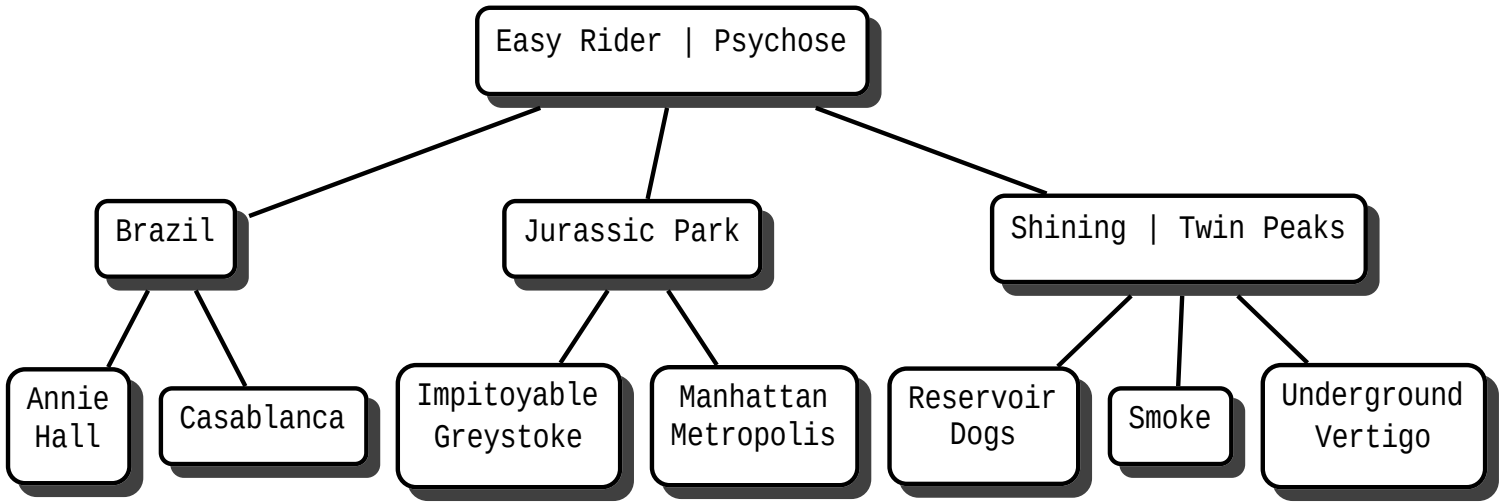
Il reste Smoke

Construction de l'arbre-B



Ca drde toujours

Construction de l'arbre-B



Et voil rltat

L'arbre B+

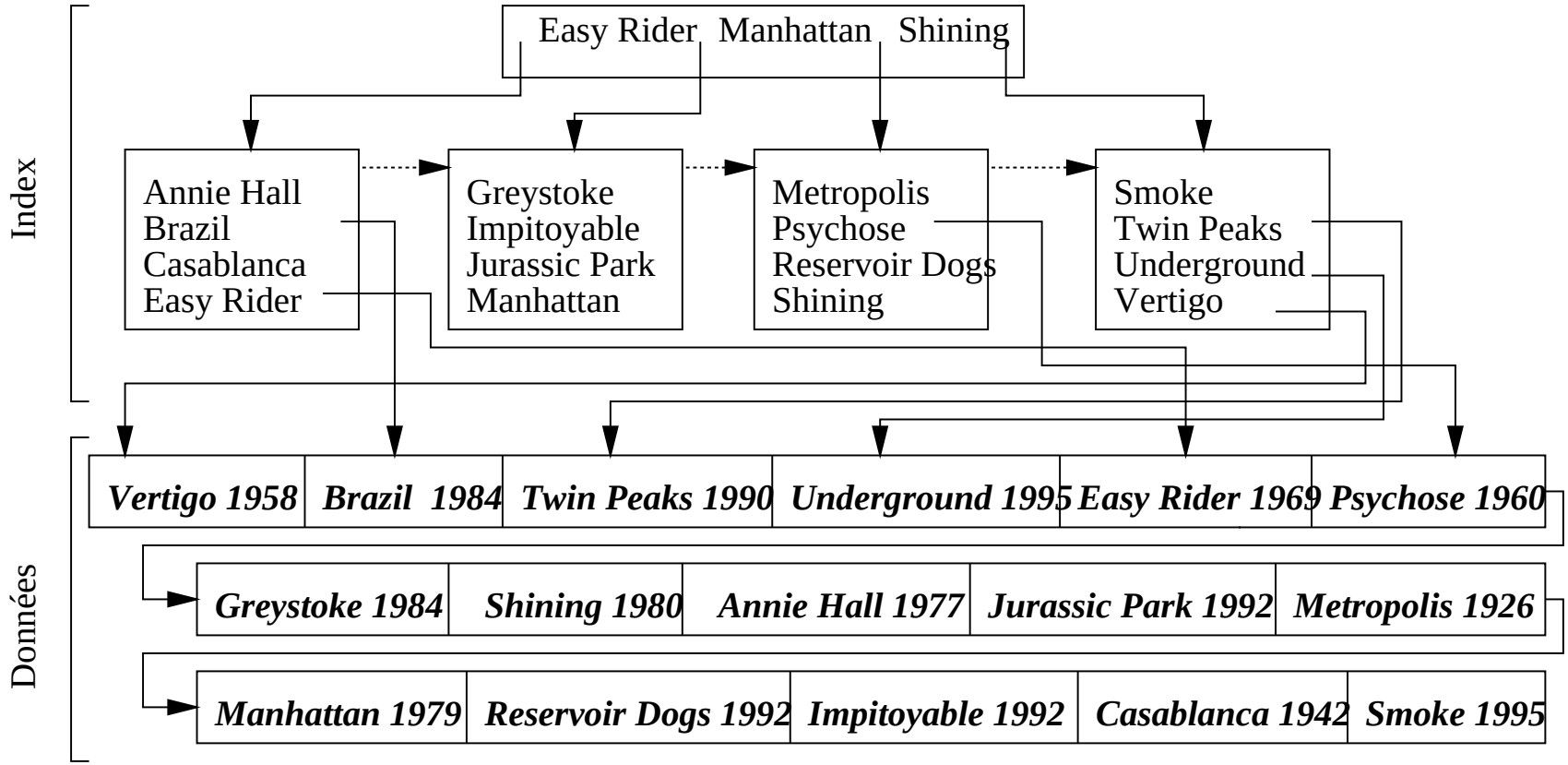
Inconvénient de l'arbre B

- il est plaçant (un seul par fichier)
- Les enregistrements sont déplacés pendant la construction (pb d'adressage)

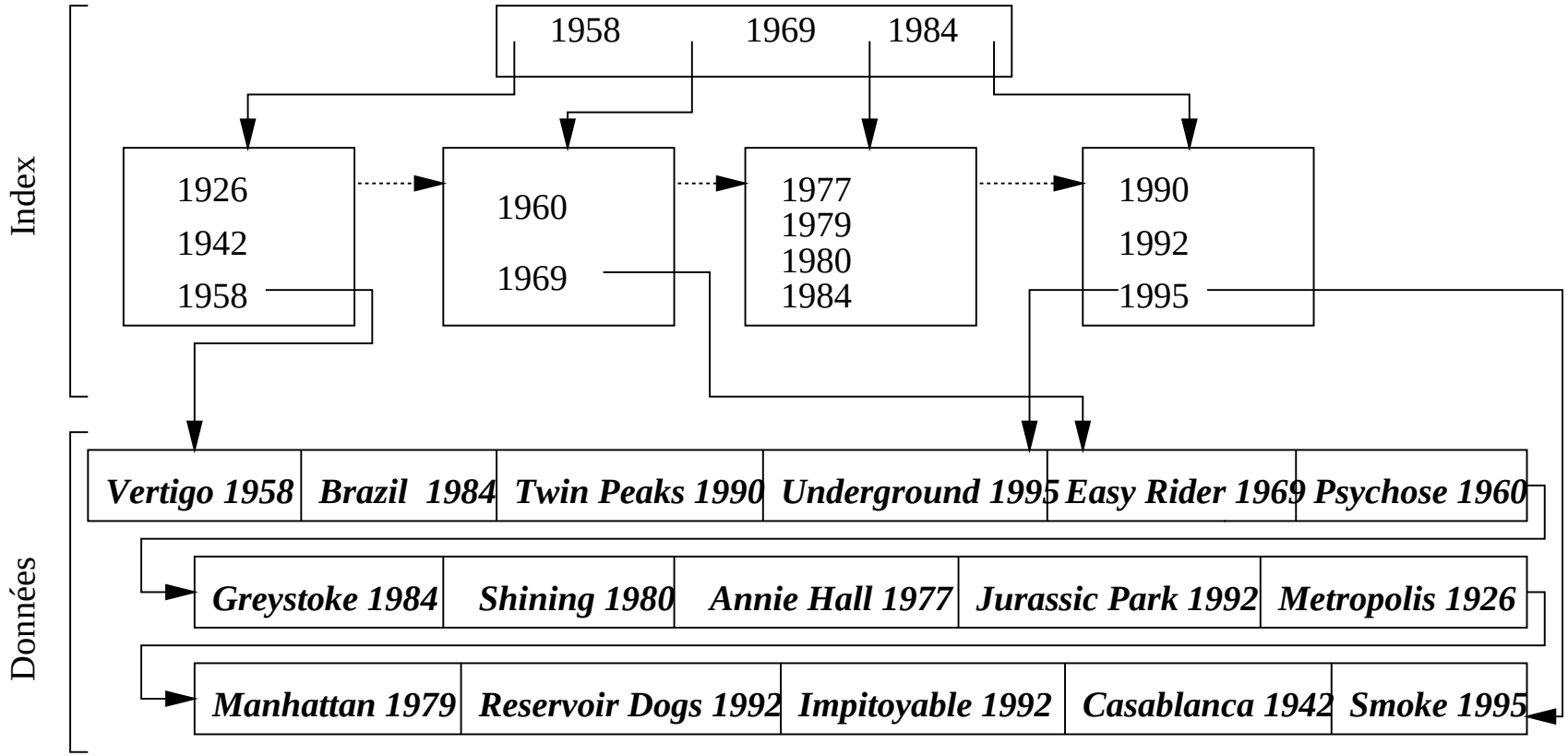
L'arbre B+ est une variante non plaçante,

- Construit uniquement sur les clés
- Toutes les clés sont conservées dans les feuilles
- Dans les feuilles, à chaque clé est associée l'adresse de l'enregistrement

Arbre B+ : exemple 1



Arbre B+ : exemple 2



Exemple concret

Fichier de 120 Mo.

- une entrée = 28 octets. Donc $\lfloor \frac{4096}{28} \rfloor = 146$ entrées par bloc
- $\lceil \frac{1000000}{146} \rceil = 6\,850$ blocs pour le premier niveau de l'arbre-B+.
- $\lfloor \frac{6\,850}{146} \rfloor = 47$ blocs pour le second niveau
- un bloc avec 47 entrées pour le troisième niveau.

Recherche par clé

```
SELECT * FROM Film WHERE titre = 'Impitoyable'
```

- on lit la racine de l'arbre : *Impitoyable* étant situé dans l'ordre lexicographique entre *Easy Rider* et *Manhattan*, on doit suivre le chaînage situé entre ces deux titres ;
- on lit le bloc feuille dans lequel on trouve le titre *Impitoyable* associé à l'adresse de l'enregistrement dans le fichier des données ;
- il reste à lire l'enregistrement.

Recherche par intervalle

```
SELECT * FROM Film WHERE annee BETWEEN 1960 AND 1975
```

- On fait une recherche par clé pour l'année 60
- on parcourt les feuilles de l'arbre en suivant le chaînage, jusqu'à l'année 1975
- à chaque fois ont lit l'enregistrement

Attention, les accès aux fichiers peuvent coûter très cher.

Recherche par préfixe

Exemple :

```
SELECT * FROM Film WHERE titre LIKE 'M%'
```

- Revient à une recherche par intervalle.
  ```
  SELECT * FROM Film WHERE titre BETWEEN 'MAAAAAA...'  AND
  'MZZZZZZ...'
  ```

Contre-exemple :

```
SELECT * FROM Film WHERE titre LIKE '%e'
```

- Ici index inutilisable

Capacité d'un arbre B

- avec un niveau d'index (la racine seulement) on peut donc référencer 146 films ;
- avec deux niveaux on indexe 146 blocs de 146 films chacun, soit $146^2 = 21\,316$ films ;
- avec trois niveaux on indexe $146^3 = 3\,112\,136$ films ;
- enfin avec quatre niveaux on index plus de 450 millions de films.

L'efficacité d'un arbre-B+ dépend de la taille de la clé : plus elle est petite, plus l'index sera petit et efficace.

Efficacité de l'arbre B+

Il est (presque) parfait !

- On a très rarement besoin de plus de trois niveaux
- Le coût d'une recherche par clé est le nombre de niveaux, plus 1.
- Supporte les recherches par clé, par intervalle, par préfixe
- Dynamique

On peut juste lui reprocher d'occuper de la place.

Le hachage

Un concurrent de l'arbre-B+

- **Meilleur** pour les recherches par clé
- **N'occupe aucune place**

Mais

- se réorganise difficilement
- ne supporte pas les recherches par intervalle

Principe du hachage

On *calcule* la position d'un enregistrement d'après la clé.

- un *fonction de hachage h* associe des valeurs de clé à des adresses de bloc
- *h* doit répartir uniformément les enregistrements dans les *n* blocs alloués à la structure
- recherche d'un enregistrement => calcul de l'endroit où il se trouve.

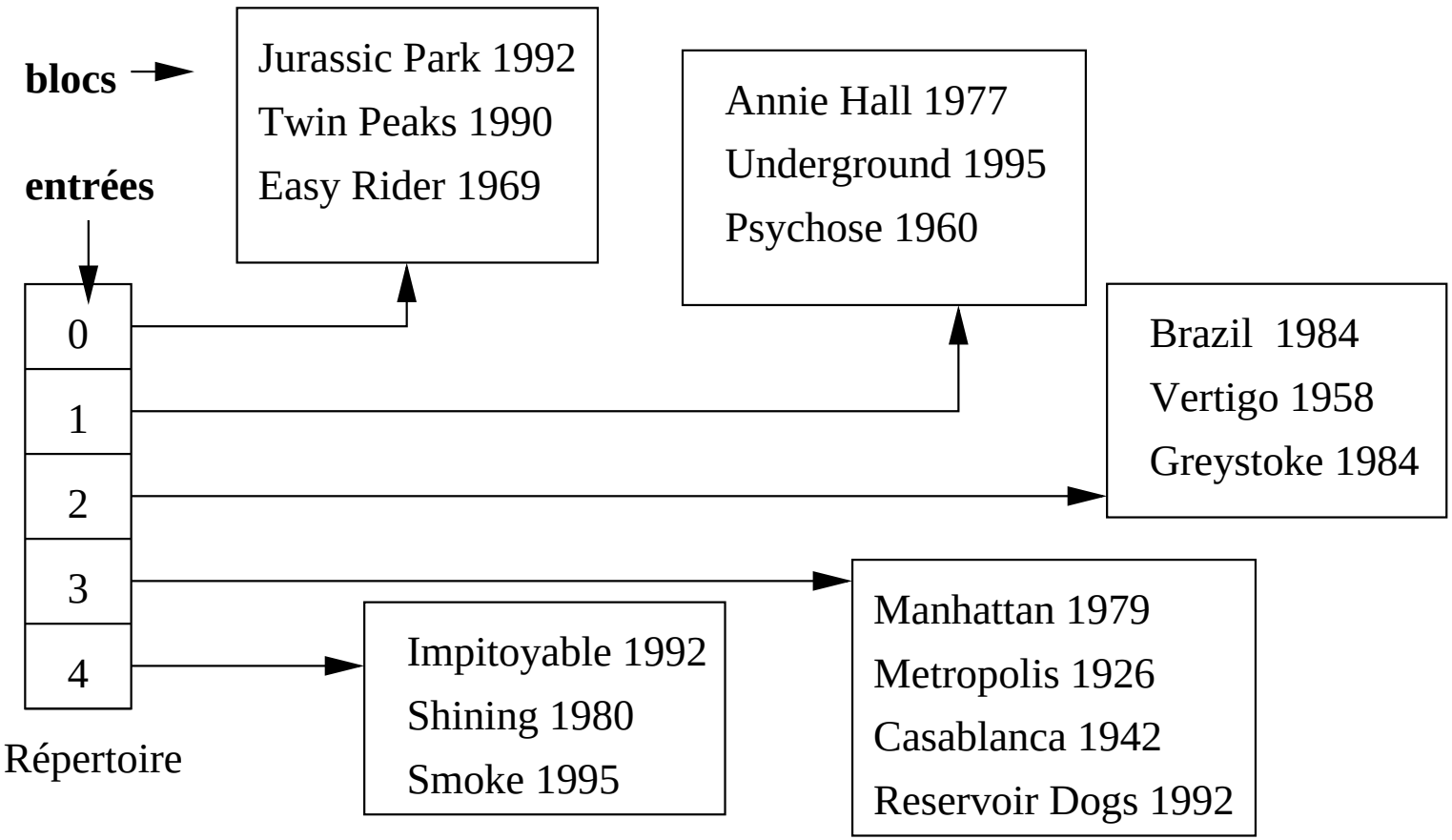
Simple et efficace !

Exemple

On veut créer une structure de hachage pour nos 16 films (hyp. : 4 enregistrements par page)

- on alloue 5 pages (pour garder une marge de manœuvre)
- un répertoire à 5 entrées (0 à 4) pointe vers les pages
- On définit la fonction $h(titre) = rang(titre[0]) \, mod \, 5$

Le résultat


blocs
entrées
Jurassic Park 1992
Twin Peaks 1990
Easy Rider 1969
Annie Hall 1977
Underground 1995
Psychose 1960
Brazil 1984
Vertigo 1958
Greystoke 1984
0
1
2
3
4
Impitoyable 1992
Shining 1980
Smoke 1995
Manhattan 1979
Metropolis 1926
Casablanca 1942
Reservoir Dogs 1992
Répertoire

Recherches

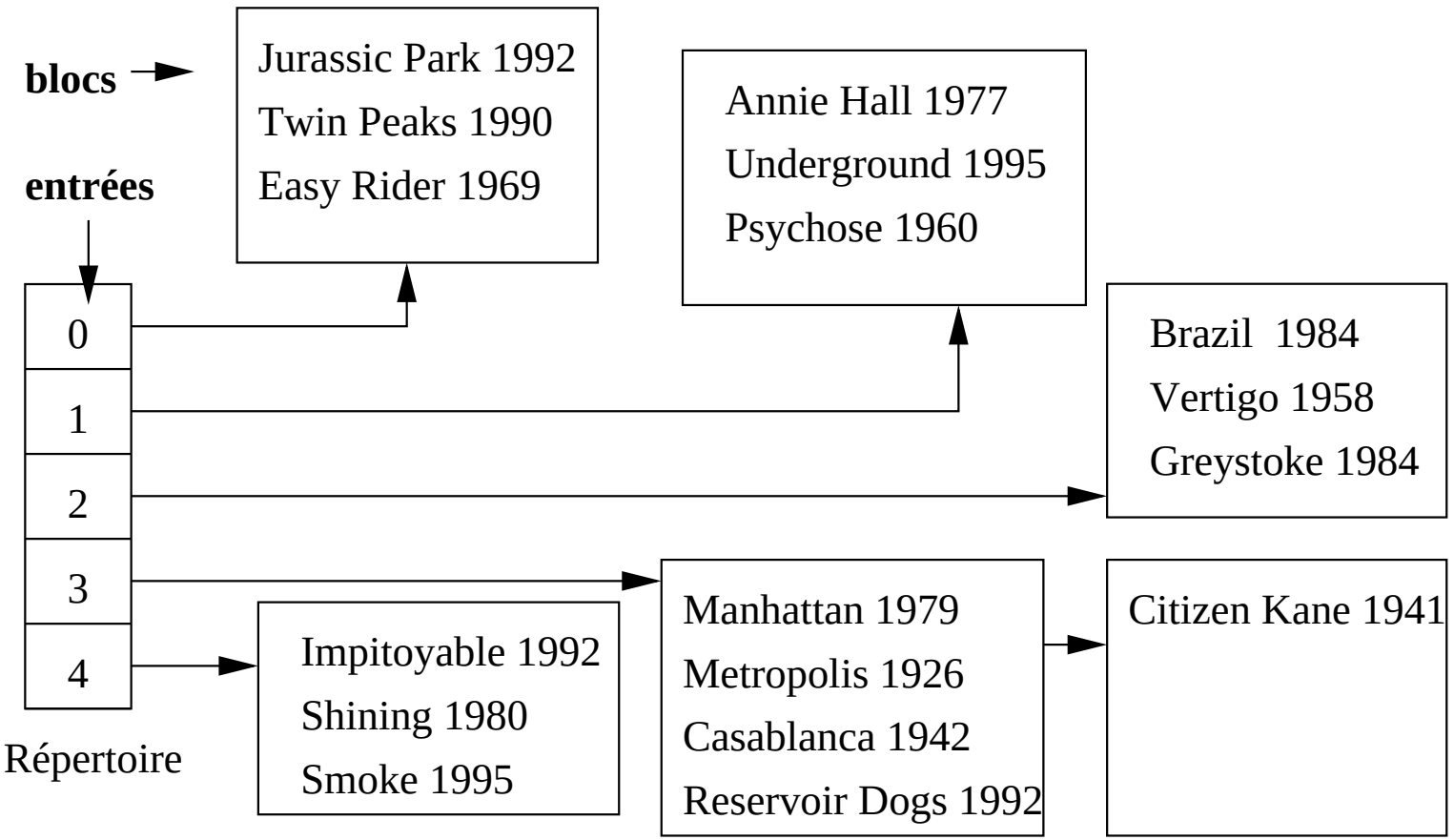
- Par clé , Oui:
  ```
  SELECT * FROM Film WHERE titre = 'Impitoyable'
  ```
- Par préfixe ? Ici, oui.
  ```
  SELECT * FROM Film WHERE titre LIKE 'M%'
  ```
- Par intervalle : non !
  ```
  SELECT * FROM Film WHERE titre BETWEEN 'Annie Hall' AND
  'Easy Rider'
  ```

Mises à jour : ça se gâte

La structure simple décrite précédemment n'est pas **dynamique**

- On ne peut pas changer un enregistrement de place
- Donc il faut créer un chaînage de pages quand une page déborde
- Et donc les performances se dégradent...

Exemple : insertion de Citizen Kane



Hachage dynamique

Objectif : réorganiser la table de hachage en fonction des insertions et suppressions.

- le nombre d'entrées dans le répertoire est une puissance de 2
- la fonction h donne toujours un entier sur 4 octets (32 bits)

Idée de base : on utilise les n premiers bits du résultat de la fonction, avec $n < 32$

Exemple : le hachage des 16 films

titre	h(titre)
Vertigo	01110010
Brazil	10100101
Twin Peaks	11001011
Underground	01001001
Easy Rider	00100110
Psychose	01110011
Greystoke	10111001
Shining	11010011

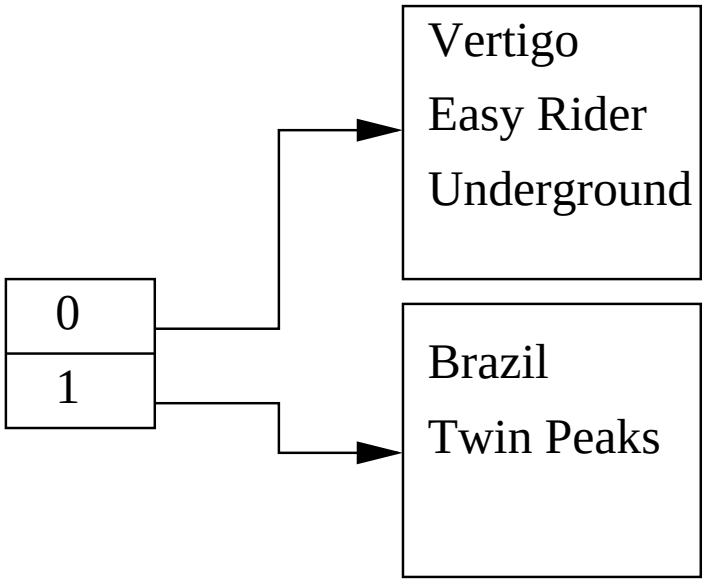
Construction de la table

On départ on utilise seulement le premier bit de la fonction

- Deux valeurs possibles : 0 et 1
- Donc deux entrées, et deux blocs
- L'affectation d'un enregistrement dépend du premier bit de sa fonction de hachage

=> pour l'instant on reste dans un cadre classique

Avec 5 films



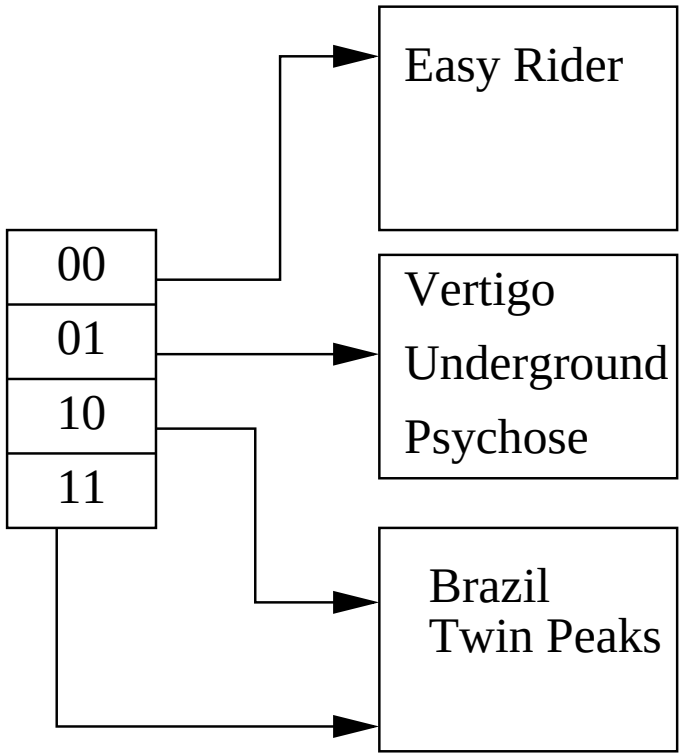
Insertions

Supposons 3 films par bloc. L'insertion de Psychose (valeur 01110011) entraîne le débordement du premier bloc.

- On double la taille du répertoire
- On alloue un nouveau bloc pour l'entrée 01
- Les entrées 10 et 11 pointent sur le *même* bloc.

On agrandit seulement ce qui est nécessaire.

Illustration



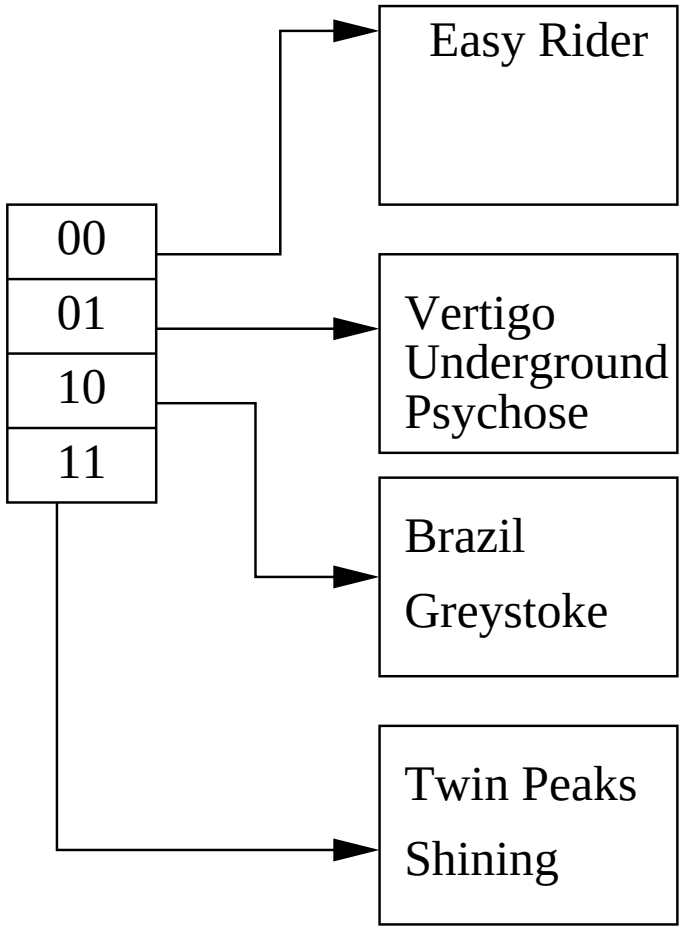
Insertions suivantes

Plusieurs cas

- on insère dans un bloc plein, mais plusieurs entrées pointent dessus
  - on alloue un nouveau bloc, et on répartit les pointeurs
- on insère dans un bloc plein, associé à une entrée
  - on double à nouveau le nombre d'entrées

  Problème : le répertoire peut devenir très grand.

Greystoke 10111001 et Shining 11010011



Comparaison hachage / arbre-B

La hachage, intéressant quand :

- Le jeux de données est figé
- Les recherches se font par clé

Ce sont des situations relativement courantes : le hachage n'occupe alors pas de place.
Sinon l'arbre-B est meilleur.

Le problème

Comment indexer une table sur un attribut qui ne prend qu'un petit nombre de valeurs ?

- Avec un arbre B : pas très bon car chaque valeur est peu sélective
- Avec un hachage : pas très bon non plus car il y a beaucoup de collisions.

Or situation fréquente, notamment dans les entrepôts de données

Exemple : codification des films

rang	titre	genre	...
1	Vertigo	Suspense	...
2	Brazil	Science-Fiction	...
3	Twin Peaks	Fantastique	...
4	Underground	Drame	...
5	Easy Rider	Drame	...
6	Psychose	Drame	...
7	Greystoke	Aventures	...
8	Shining	Fantastique	...
...	...	...	...

Principes de l'index bitmap

Soit un attribut A, prenant n valeurs possibles $[v_1, \ldots v_n]$

- On crée n tableaux de bit, un pour chaque valeur v_i
- Ce tableau contient un bit pour chaque enregistrement e
- Le bit d'un enregistrement e est à 1 si $e.A = v_i$, à 0 sinon

Exemple

	1	2	3	4	5	6	7	8
Drame	0	0	0	1	1	1	0	0
Science-Fiction	0	1	0	0	0	0	0	0
Comédie	0	0	0	0	0	0	0	0

	9	10	11	12	13	14	15	16
Drame	0	0	0	0	0	0	1	0
Science-Fiction	0	1	1	0	0	0	0	0
Comédie	1	0	0	1	0	0	0	1

Recherche

Soit une requête comme :
`SELECT * FROM Film WHERE genre='Drame'`

- On prend le tableau pour la valeur `Drame`
- On garde toutes les cellules à 1
- On accède aux enregistrements par l'adresse

=> très efficace si n, le nombre de valeurs, est petit.

Autre exemple

```
SELECT COUNT(*) FROM Film WHERE genre IN ('Drame', 'Comédie')
```

- On compte le nombre de 1 dans le tableau `Drame`
- On compte le nombre de 1 dans le tableau `Comédie`
- On fait la somme et c'est fini

=> typique des requêtes *DataWarehouse*

Indexation de données géométriques

Certaines applications stockent des données géométriques (ex, cadastre, réseaux routiers, etc.). Requêtes courantes :

1. pointé,
2. fenêtrage,
3. jointure spatiale.

Comment optimiser ces opérations avec des index ?

Obstacles

Principaux obstacles à l'utilisation d'un index standard :

1. On ne peut pas utiliser un arbre-B
   car pas d'ordre sur les points ou les polygones.
2. Opérations coûteuses
   Algorithmes géométriques, souvent de complexité polynomiale, difficiles à implanter.

$\Rightarrow$ des structures spécialisées pour les données géométriques

Principes

Pour limiter le coût des opérations, on utilise une approximation rectangulaire des données géométriques : le rectangle englobant.

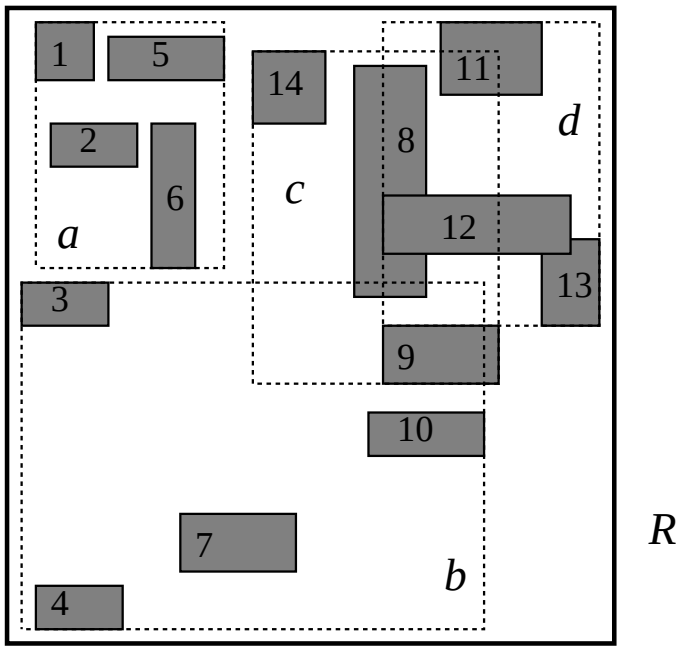
1. l'index est construit sur l'ensemble des rectangles ;
2. On effectue un *filtrage* basé sur les rectangles pour les pointés, fenêtrages et jointures.
3. Quand on accède aux objets eux-mêmes, on effectue une seconde opération sur la vraie géométrie (étape de *raffinement*)

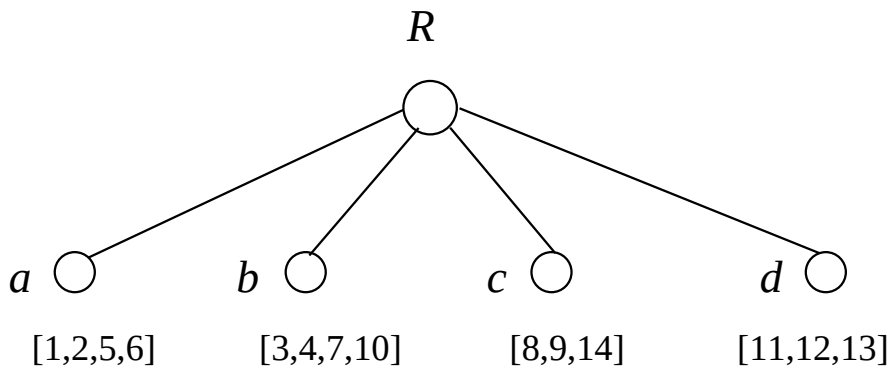
L'arbre R, définition

Assez proche de l'arbre-B dans ses principes :

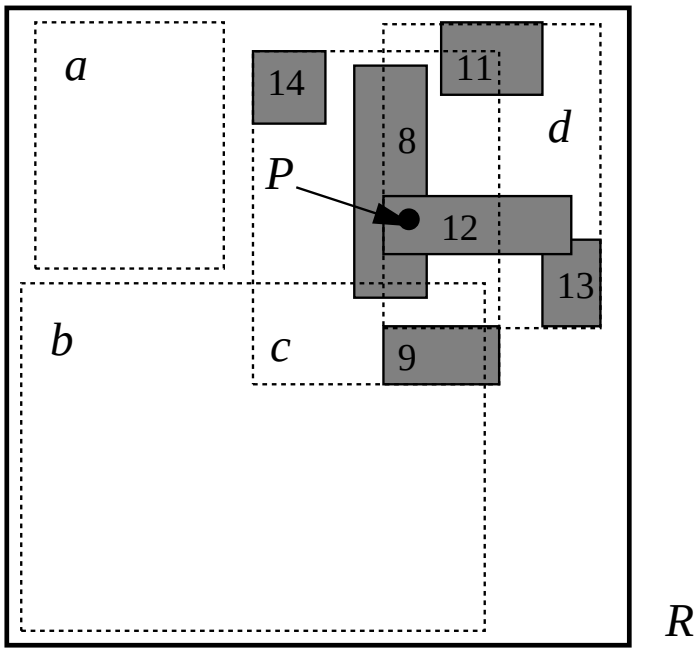
1. c'est un arbre équilibré ;
2. chaque entrée est une paire $[adresse, rectangle]$;
3. tous les nœuds sont occupés au moins à 50 % ;
4. le rectangle d'une entrée $[adresse, rectangle]$ englobe *tous* les rectangles du nœud pointé par *adresse* ;

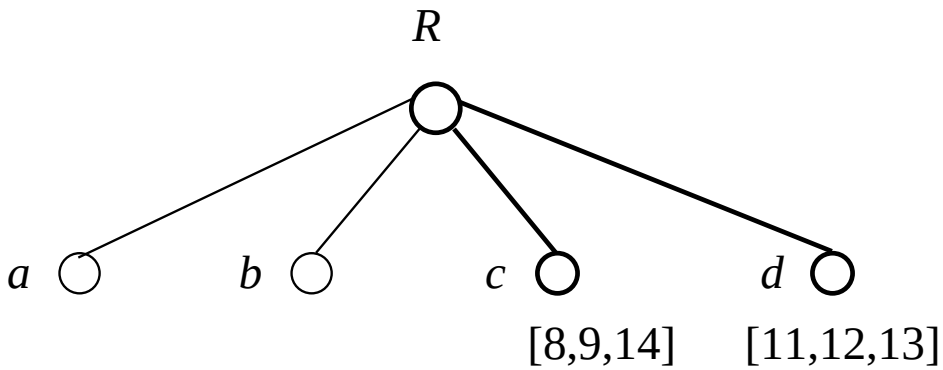
L'arbre R, exemple



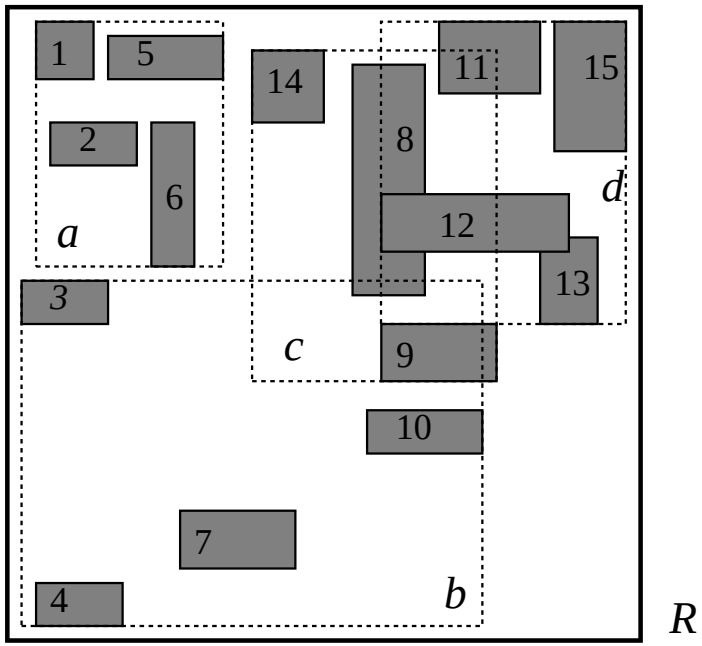


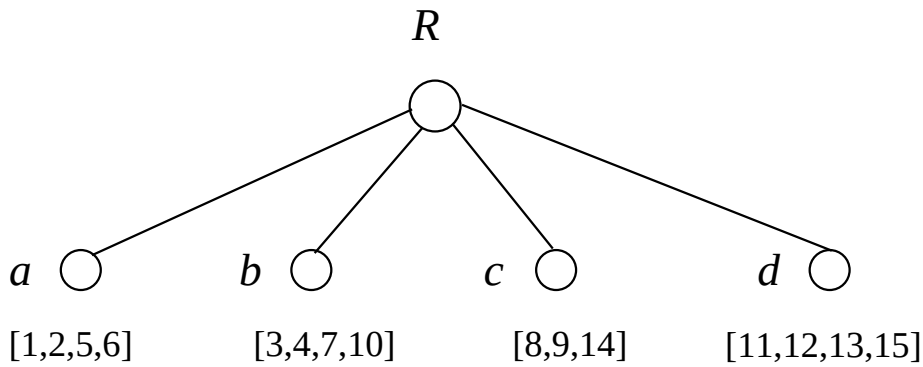
L'arbre R, pointé



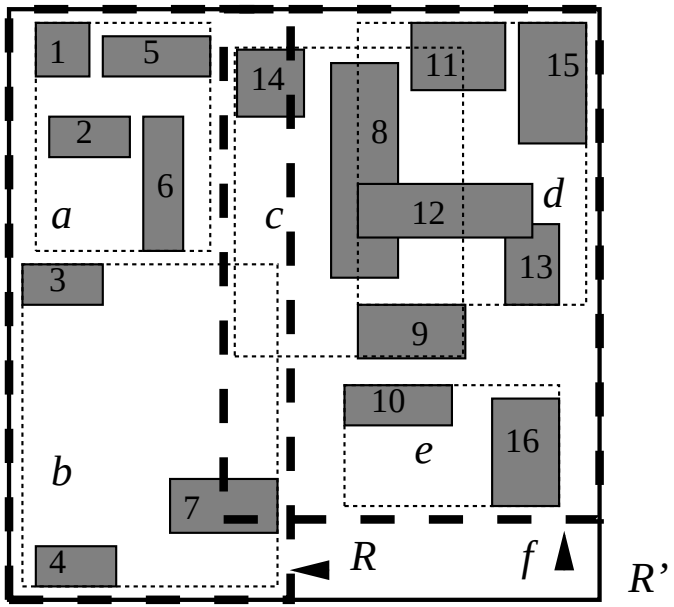


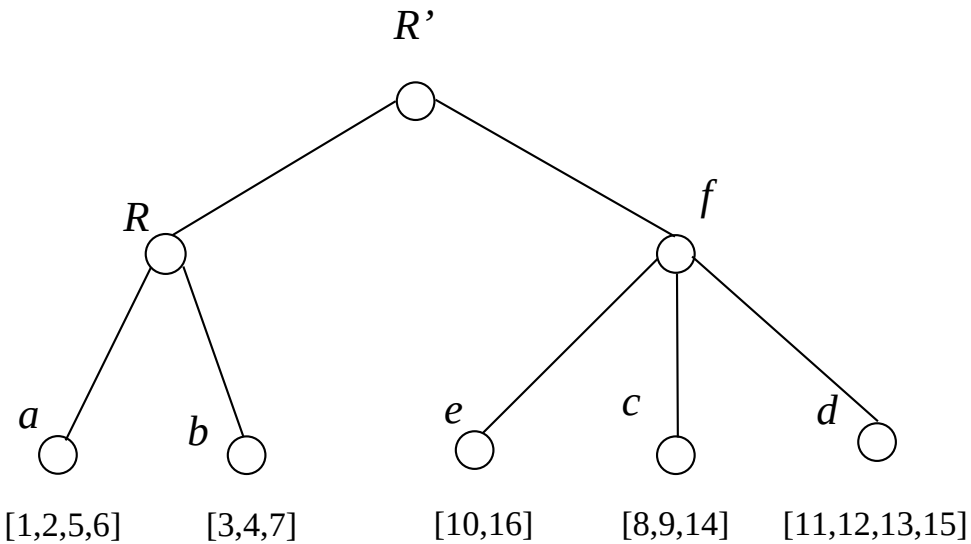
L'arbre R, insertion (1)





L'arbre R, insertion (2)





Les choix d'Oracle

Oracle propose à peu près *toutes* les structures d'index vues précédemment.

- Par défaut l'index est un arbre B+
- Il est possible d'organiser une table en arbre B (plaçant)
- Le hachage (non dynamique) existe aussi
- Les index bitmap
- Les arbres R

Arbres B et B+

- Dès qu'on utilise une commande `PRIMARY KEY`, Oracle crée un arbre B+ sur la clé primaire
  - l'arbre est stocké dans un *segment d'index*
- On peut organiser la table en arbre B avec l'option `ORGANIZATION INDEX`
  - plus efficace car évite les accès par adresse
  - moins valable pour les enregistrements de grande taille

Hachage

Structure appelée `Hash Cluster`. Utilisée en deux étapes

- On crée la structure avec tous ses paramètres
- On affecte une ou plusieurs tables à la structure

Attention, le hachage dans Oracle n'est pas dynamique

Création d'un `Hash Cluster`

```
CREATE CLUSTER HachFilms (id INT)
SIZE 500 HASHKEYS 500;
```

- La clé de hachage est de type `INTEGER` ; Oracle fournit automatiquement une fonction avec de bonnes propriétés
- Nombre de valeurs de la fonction donné par `HASHKEYS`
- Taille de chaque entrée estimée par `SIZE`

Donc ici 8 entrées par bloc

Affectation à un *Hash Cluster*

On indique la structure d'affectation dans la commande `CREATE TABLE`

```
CREATE TABLE Film (idFilm INT,
...  )
CLUSTER HachFilms (idFilm)
```

- Assez délicat à paramétrer
- Demande un contrôle régulier par un DBA