Cours ensae - structure de donnée

Nicolas Rousset

20 Octobre 2014

Plan

- Accès aux données
- Modèles relationnels
- Base de données usuelles
- Modèle NoSQL

Accès aux données

Facile ou difficile ?

Vous avez un dictionnaire Larousse à disposition, est-il facile de :

- Trouver la définition de ziggourat ?

Facile ou difficile ?

Vous avez un dictionnaire Larousse à disposition, est-il facile de :

- Trouver la définition de ziggourat ?
- Trouver le mot dont la définition est “Se dit d’une corolle gamopétale en forme d’entonnoir (fleur de liseron).” ?

Facile ou difficile ?

Vous avez un dictionnaire Larousse à disposition, est-il facile de :

- Trouver la définition de ziggourat ?
- Trouver le mot dont la définition est “Se dit d’une corolle gamopétale en forme d’entonnoir (fleur de liseron).” ?
- Pourquoi ?

Facile ou difficile ?

Vous avez un dictionnaire Larousse à disposition, est-il facile de :

- Trouver la définition de ziggourat ?
- Trouver le mot dont la définition est “Se dit d’une corolle gamopétale en forme d’entonnoir (fleur de liseron).” ?
- Pourquoi ?
- Comment faire un dictionnaire inversé ?

Structure de donnée - generalite

1. Les problèmes que nous allons aborder ici sont généraux, que l'on parle de base de données, de fichiers textes ou de données en memoire
2. Il n'y a pas de solution optimale universelle -> cela dépend de votre usage
3. Gagner en performance sur un type d'accès se traduit souvent par une perte de performance en insertion/suppression

Structure de donnée - complexité

Trois complexités à garder en tête :
- Accès séquentiel => O(1)
- Accès par recherche binaire / arbre binaire => O(log(n))
- Accès par lecture complète ou partielle => O(n)

Attention aussi à ce que la complexité ne fait pas tout . . .

Structure de donnée - complexité

Reprenons notre dictionnaire ...
- Accès séquentiel => vous avez le numéro de la page, vous y aller
- Accès par recherche binaire => vous cherchez le mot par rapport à l'ordre alphabétique
- Accès par lecture complète => vous lisez tout le dictionnaire à la recherche de votre définition

Et les tables de hachage dans tout cela ?

Les tables de hashage sont un type de conteneur spécial, qui permet un accès par index en O(1) quand tout va bien, O(n) quand tout va mal.
On ne les détaillera pas ici.
Elles sont très utilisées dans des structures à courte durée de vie, beaucoup moins sur des données persitantes.

Accès séquentiel

Un type d'accès très spécifique, avec une clé entière, qui peut se résumer en :
"Vous avez un tableau de 127 éléments, vous voulez accéder au 65ème"
"Vous avez un dictionnaire de 540 pages, vous voulez accéder à la 300ème"

Pensez-y, notamment pour les séries temporelles :
Si vous avez des données horaires annuelles, cela sera beaucoup plus rapide de faire la conversion heure $=>$ position de l'heure dans l'année qu'une recherche par date.

Recherche binaire

On cherche le nombre 13

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
               m |                 |
                 [8, 13, 21, 34, 55]
                 |      |  m
                 [8, 13]
                  m |   |
                    [13]
```

Recherche binaire (2)

Supposons la liste suivante triée par nom :
(Nous reviendrons sur les structures pandas)

```
## Liste Nom, Prenom, NumeroSecu, Age, Departement

[ ("Baudelaire", "Charles", 14256, 72, 75),
  ("Dupont",     "Jean",    18123, 33, 78),
  ("Guilloux",   "Anne",    29123, 23, 92),
  ("Legrand",    "Jacques", 16348, 51, 24),
  ("Martin",     "Anne",    27649, 38, 35) ]
```

Pouvez-vous utilisez une recherche binaire sur le critere “Nom” ?
Sur le critere NumeroSecu ?

Exemple pandas

Nom	Prenom	NumeroSecu	Age	Departement
Dupont	Jean	18123	33	78
Martin	Anne	27649	38	35
Legrand	Jacques	16348	51	24
Baudelaire	Charles	14256	72	75
Guilloux	Anne	29123	23	92

Exemple pandas (2)

In [19]: timeit_setup = “import pandas; df = pandas.DataFrame.from_csv(‘Data_exemple.csv’, sep=‘;’)”

In [20]: timeit.timeit(“df.ix[‘Martin’]”, setup = timeit_setup)
Out[20]: 271.23531663940526

In [21]: timeit.timeit(“df.loc[df[‘NumeroSecu’] == 16348]”, setup = timeit_setup)
Out[21]: 569.269467418345

Comment faire pour accéder rapidement par numéro de sécu ?

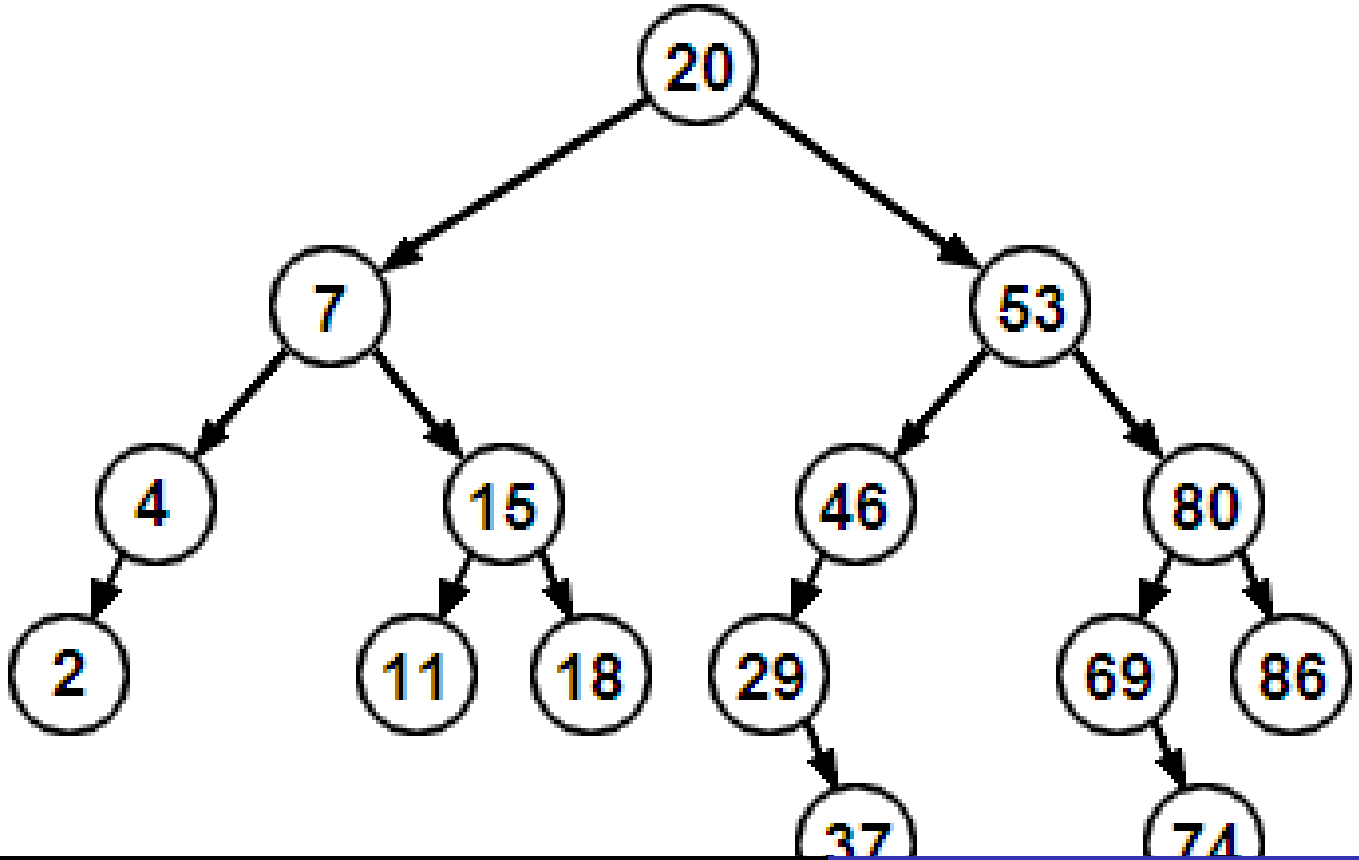
Comment faire pour accéder rapidement par numéro de sécu (2) ?

	Nom	Prenom	NumeroSecu	Age	Departement
1	Dupont	Jean	18123	33	78
2	Martin	Anne	27649	38	35
3	Legrand	Jacques	16348	51	24
4	Baudelaire	Charles	14256	72	75
5	Guilloux	Anne	29123	23	92

```
[("Baudelaire", 4), ("Dupont", 1), ("Guilloux", 5),
 ("Legrand", 3)   , ("Martin", 2)]
[(14256, 4), (16348, 3), (18123, 1),
 (27649, 2), (29123, 5)]
```

Notion d'index

Un index est une structure de donnée (souvent un arbre de recherche binaire) permettant un acces en log(n) pour un ou plusieurs champs



Notion d'index (2)

Un index peut avoir plusieurs niveaux de clé (si la première clé est dupliquée).
Par exemple pour des personnes, vous pouvez avoir un index uniquement sur les noms, ou un index sur les noms et prénoms.

Si vous avez un index sur les noms et que vous recherchez "Jacques Martin", vous allez devoir regarder toutes les personnes dont le nom est "Martin".
Si vous avez un index sur les noms et prénoms, vous ne regarderez que les personnes s'appelant "Jacques Martin".

Notion d'index (3)

L'ordre des clés est important, un index sur Nom, Prénom n'est pas la même chose qu'un index sur Prénom, Nom.

Un index sur Prénom, Nom peut être utilisé efficacement dans une recherche par Prénom.
Un index sur Nom, Prénom peut être utilisé efficacement dans une recherche par Nom.

Notion d'index (4) - exemple pandas

```
In [41]: df
Out[41]:
                          NumeroSecu  Age  Departement
Nom          Prenom
Dupont       Jean            18123   33           78
Martin       Anne            27649   38           35
Legrand      Jacques         16348   51           24
Baudelaire   Charles         14256   72           75
Guilloux     Anne            29123   23           92
```

En résumé

Il est nécessaire d'adapter votre structure de donnée à l'usage que vous voulez en faire, il n'y a pas de règle universelle.

Exemple 1

Vous gérez le fichier de la sécurité sociale, vous voulez faire des recherches par numéro de sécu, par nom, par département.

Exemple 2

Vous gérez un site internet, il y a beaucoup d'insertion concernant les données des visiteurs, et vous avez besoin de faire des statistiques sur ces entrées par heure ou par jour.

Pouquoi ne faut-il pas mettre d'index par département ?

Un index par département sur une base de donnée de la sécurité sociale n'est pas du tout discriminant (environ 100 valeurs possibles pour toute la population).

De plus une lecture complète (Full_scan) se fera par sequential read, 100 à 1000 fois plus rapide que les random read des accès par index.

Pour en revenir à notre dictionnaire . . .

- Est-ce une bonne idée de mettre un index sur un les définitions ?

Pour en revenir à notre dictionnaire . . .

- Est-ce une bonne idée de mettre un index sur un les définitions ?
- Par exemple “Se dit d’une corolle gamopétale en forme d’entonnoir (fleur de liseron).” ?

Pour en revenir à notre dictionnaire . . .

- Est-ce une bonne idée de mettre un index sur un les définitions ?
- Par exemple “Se dit d’une corolle gamopétale en forme d’entonnoir (fleur de liseron).” ?
- Probablement pas, éventuellement sur les mots/notions de l’index . . .

Modèles relationnels

Modèles relationnels - exemple

Nom	Prenom	Dep	DepName	Pref
Dupont	Jean	78	Yvelines	Versailles
Martin	Anne	92	Haut-de-Seine	Asnieres
Legrand	Jacques	78	Yvelines	Versailles
Baudelaire	Charles	75	Paris	Paris
Guilloux	Anne	92	Haut-de-Seine	Asnieres
Richard	Laurent	78	Yvelines	Versailles
Leclerc	Henri	75	Paris	Paris

Modèles relationnels - utilité numéro 1

- Eviter la duplication de donnée !
- Permettre d'accéder différemment aux données

Modèles relationnels - relation 1-n (sous-liste)

Id	Compagny_name	Country
1	Air France	France
2	Lufthansa	Germany

Id	Plane_immat	Company_id
1	F-AGDF	1
2	F-GDFT	1
3	D-JUKI	2
4	D-MLOK	2
5	D-PLOI	2

Modèles relationnels - relation n-n

```
    Compagny_name Country
Id
1   Air France    France
2   Lufthansa     Germany

    Flight_code
Id
1   AF6281
2   LH4567

    Company_id Flight_id
0   1          1
1   1          2
2   2          1
```

Manipulation de modeles relationnels avec pandas (1)

```
df.join( df_2 ) ## Jointure sur les index
df.join( df_2, on = "Nom" ) ## Sur la colonne nom
df.join( df_2, how = "left", rsuffix = "_" )
```

Manipulation de modeles relationnels avec pandas (2)

```
SELECT * FROM table1 JOIN table2 on Nom;
```

How :
- inner : intersection
- left / left outer : on garde les éléments de gauche
- right / right outer : on garde les éléments de droite
- outer : union

Manipulation de modeles relationnels avec pandas (3)

```
import pandas
pandas.merge( df_1, df_2, left_on = "Id",
    right_on = "Flight_id", how = "outer" )
```

Manipulation de modeles relationnels avec pandas (4)

```
import pandas
df_1.groupby( "Departement" )["Age"].means()
```

Base de données usuelles

Base de données usuelles

La majeure partie des bases de données actuelles (PostGres, MySql, Oracle, SQL Server, db2, MariaDB, etc . . .) sont des bases de données relationnelles & transactionnelles “classiques”.
Elles assurent la cohérence des données selon les principes définies par l’acronyme ACID.
Ces principes impliquent des mécanismes assez coûteux, notamment les “undo-segment”.

Notion de transaction

Une transaction est un ensemble de requête qui amène la base d'un état cohérent vers un autre état cohérent.

Par exemple pour une base de donnée comptable, cela pourrait être :

```
INSERT into DEBIT values( ... );
INSERT into CREDIT values( ... );

INSERT into DEPARTEMENT( 78, 'Yvelines', 'Versailles' );
INSERT into PERSON( 'Martin', 'Jacques', 32, 13245, 78 );
```

Principes ACID

- Atomicité
- Cohérence
- Isolation
- Durabilité

Ces principes concernent à la fois la couche applicative (software) et physique (hardware).

Principes ACID

Atomicité

Une transaction doit être exécutée totalement ou pas du tout.

- Cohérence
- Isolation
- Durabilité

Principes ACID

- Atomicité

Cohérence

Une transaction doit amener la base d'un état respectant toutes les règles de la base vers un autre état respectant ces mêmes règles.

- Isolation
- Durabilité

Principes ACID

- Atomicité
- Cohérence

Isolation

Les transactions doivent être isolées les unes des autres, c'est à dire que les résultats d'une exécution parallèle doivent correspondre à ceux d'une exécution séquentielle.

- Durabilité

Principes ACID

- Atomicité
- Cohérence
- Isolation

Durabilité

Les données en base doivent être stockées de façon permanente, de façon à supporter une panne de courant.

Problème des undo segment . . .

La propriété d'isolation peut être très coûteuse avec des grosses bases de données, lorsque le volume d'information et les temps d'exécution deviennent importants.
Pourquoi ?

```
SELECT * FROM DOCUMENT WHERE AGENCY_CODE = 'FR0076'
  and issue_date = '2014-10-19';
UPDATE DOCUMENT SET CANCELLED = 1 WHERE ID = 2001567345;
```

Problème des undo segment (2)...

```
SELECT * FROM DOCUMENT WHERE AGENCY_CODE = 'FR0076'
  and issue_date = '2014-10-19';
UPDATE DOCUMENT SET CANCELLED = 1 WHERE ID = 2001567345;
```

Le principe d'isolation signifie que la première query doit renvoyer un résultat identique que la deuxième soit faite ou non.
Comment cela est-il possible ? En fait on va devoir se souvenir de l'état de la base de donnée au moment où la première query a été lancée.
C'est le principe des undo segments, qui peut être très coûteux.

Limites des bases de données classiques : ACID VS Big data

- Le principe ACID implique plusieurs systèmes qui peuvent être extrêmement coûteux.
- Dans un monde BigData, avons-nous vraiment besoin de données parfaites ?

Modèle NoSQL

Limites des bases de données classiques

- Elles reposent sur des principes de cohérence des données et de garantie de leur qualité qui peuvent être plus coûteux que bénéfique.
- Bien entendu cela dépend du contexte, si vous êtes une banque, vous avez intérêt à ce que votre base de donnée soit cohérente. . .
- Elles sont basées sur des modèles relationnels qui peuvent être assez coûteux, notamment sur des données un peu trop hétérogènes.

Structure relationnel VS acces hierarchique

- Les structures relationnels classiques ne permettent pas de stocker des données éthérogènes dans une seule table . . .
- Ce qui supposent de les répartir sur plusieurs tables, et donc de faire plusieurs accès.
- Avec une structure hiérarchique, une fois que vous avez accédé à votre enregistrement, vous pouvez accéder à toutes les données

Données structurées

- Il n'est pas toujours évident de stocker des valeurs dont le label est complètement libre ...
- Dans ce cas là on les stocke sous forme de clé/valeur
- Mais que faire si on a une liste ?

MongoDB & JSON

```
{
  "Name" : "Dupont",
  "Age"  : 25,
  "Hobbies" : ["Guitar", "Swimming", "Horse-riding"],
  "Statut" : "V"
}
```