Essentiel d'informatique

2023/2024

Victor Sarrazin

Bienvenue dans l'essentiel d'informatique de mes cours de prépa. Ce document a pour objectif de contenir l'intégralité des cours d'informatique afin de les condenser et de les adapter.

Bonne lecture...

Sommaire

Introduction aux langages :

I Introduction au C 3
II Introduction au OCaml 5

Structures de données :

I Structures de données 6
II Piles, files, dictionnaires 6
III Arbres 6
IV Graphes 6

Informatique théorique :

I Bases 7
II Récursion 7
III Stratégies algorithmiques 7
IV SQL 7

Introduction

I Introduction au C

I.1 Variables

Pour définir une variable en C on a la syntaxe suivante : `type nom`

```c
int mango = 0;
```

Il est possible de définir plusieurs variables en même temps :

```c
int banana = apple = 12;
```

I.2 Opérateurs

On a les opérations arithmétiques suivantes :

Opération	En C
Addition	`a + b`
Soustraction	`a - b`
Multiplication	`a * b`
Division	`a / b`
Modulo	`a % b`

On peut utiliser `+=`, `-=`, `*=`, `/=` et `%=` pour faire des opérations arithmétiques et des assignations

De plus on peut utiliser `++` et `--` pour incrémenter/décrémenter

Les comparaisons se font avec `>`, `>=`, `<=`, `<` et `==`.

On a des opérateurs binaires `&&` (et logique), `||` (ou logique) et `!` (négation de l'expression suivante)

> **Attention :**
> Le `&&` est prioritaire sur le `||`

I.3 Structures de contrôle

Pour exécuter de manière conditionnelle, on utilise `if (cond) {...} else if (...) {} ... {} else {}`

Ainsi le code suivant est valide :

```c
if (x == 1) {
  // Do code
} else if (x > 12) {
  // Do code bis
} else {
  // Do code ter
}
```

> **Attention :**
> En C un 0 est considéré comme `false` et toute autre valeur numérique `true`

Pour faire une boucle on peut utiliser un `while (cond) {}` qui exécute le code tant que la condition est valide

On peut utiliser `do {} while (cond)` qui exécute une fois puis tant que la condition est vérifiée

Il est aussi possible d'utiliser `for (...) {}`, de la manière suivante :

```c
// De 0 à n - 1
for (int i = 0; i < n; i++) {

}

// De 0 à n - 1 tant que cond
for (int i = 0; i < n && cond; i++) {

}
```

A noter qu'en C il est possible de modifier la valeur de `i` et donc de sortir plus tôt de la boucle

Il est possible de sortir d'une boucle avec `break`, ou de passer à l'itération suivante avec `continue`

I.4 Fonctions

Pour définir une fonction on écrit :

```c
int my_func(int a, int b) {
  // Do code
  return 1;
}
```

Si on ne prend pas d'arguments on écrit `int my_func(void) {}` et si on ne veut rien renvoyer on utilise `void my_func(...) {}`

Ainsi pour appeller une fonction on fait :

```c
int resp = my_func(12, 14);
```

On peut déclarer une fonction avant de donner son code mais juste sa signature avec :

```c
int my_func(int);
```

I.5 Tableaux en C

Le type d'un tableau en C est `type[]` ou `* type`

Pour initialiser un tableau on a les manières suivantes :

```c
int[4] test = {0, 1, 2, 3}; // Initialise un tableau de taille 4 avec 0,1,2,3
int[] test = {0, 1, 2, 3}; // Initialise un tableau avec 0,1,2,3 (avec 4 éléments)
int[4] test = {0}; // Initialise un tableau de taille 4 avec 0,0,0,0
```

Il n'est pas obligé de donner la taille d'un tableau elle sera déterminée au moment de l'exécution

> **Attention :**
> Si on dépasse du tableau C ne prévient pas mais s'autorise à faire n'importe quoi

Pour affecter dans une case de tableau on fait :

```c
test[1] = test[2] // On met dans la case 1 la valeur de la case 2
```

II Introduction au OCaml

Structures de données

I Structures de données

II Piles, files, dictionnaires

III Arbres

IV Graphes

IV.1 Recherche de plus cours chemin

IV.1.a Graphes avec poids négatifs

Dans un graphe on dit que l'arc $u \triangleright v$ est en **tension** si $\delta(v) > \delta(u) + w(u, v)$

L'approche de Ford est donc d'éliminer les arcs en tension

Tant qu'il existe des arcs en tension, on traite tous les arcs de E et on traite ceux en tension, on a donc une complexité $O(n \times p)$

Informatique théorique

I Bases

I.1 Fonctions

On dit qu'une fonction a des **effets de bord** si son exécution a des conséquences sur d'autres choses que ses variables locales

Une fonction est **déterministe** si le résultat est toujours le même avec les mêmes arguments

Une fonction est dite **pure** lorsqu'elle est déterministe et sans effets de bord

I.2 Algorithmes de tri

II Récursion

III Stratégies algorithmiques

IV SQL

IV.1 Généralités

En SQL on stocke des entités avec des attributs et à chaque attribut on lui associe un type

On peut définir des relations entre les différentes entités

On stocke ces entités dans des tables : dans chaque table on stocke une entité

Il est possible de garder une case vide en plaçant un `NULL` dans la case

IV.2 Requêtes

Pour récupérer des données (projections) dans une table on a :

```sql
# Seulement les colonnes spécifiées
SELECT col1, ..., coln FROM table

# Toutes les colonnes
SELECT * FROM table

# Toutes les colonnes mais sans doublon
SELECT DISTINCT * FROM table
```

Ainsi on récupère toutes les lignes de la table avec ces projections

On peut aussi faire une sélection sur un critère :

```sql
SELECT * FROM table WHERE bool
```

Les opérations booléennes sont les suivantes :

- `col > a`/`col < a`/`col = a` pour faire des comparaisons
- `col IN (a, b, c)` pour savoir si la cellule est dans un ensemble de valeur
- `col IS NULL`/`IS NOT NULL` pour savoir si la cellule est nulle ou non
- `col LIKE '% Text %'` pour regarder si `Text` est dans la chaine de caractère de la cellule

On peut combiner les critères avec `AND`/`OR`/`NOT`

Il est possible de sélectionner un attribut non projeté

Pour ordonner les résultats on ordonne en utilisant

```sql
# Triés par valeur croissante
SELECT * FROM table ORDER BY col

# Triés par valeur décroissante
SELECT * FROM table ORDER BY col DESC
```

Pour limiter le nombre de valeurs on utilise

```sql
# On prend au maximum 3 éléments
SELECT * FROM table LIMIT 3

# On prend au maximum 3 éléments mais sans les 2 premiers
SELECT * FROM table LIMIT 3 OFFSET 2
```

IV.3 Fonctions

On peut compter le nombre d'entités qui vont être renvoyées

```sql
# Nombre d'éléments dans la table
SELECT COUNT(*) FROM table
```

On peut compter sur une colonne spécifique avec `COUNT(col1, ..., col2)`, les cases ne sont pas comptées si `NULL`,

Il est aussi possible de compter le nombre de valeur distinctes pour une colonne :

```sql
SELECT COUNT(DISTINCT col) FROM table
```

On peut utiliser `MAX`, `MIN`, `SUM` et `AVG` pour avoir du préprocessing, il est aussi possible d'avoir la moyenne en faisant `SUM(col)/COUNT(*)`

> **Attention :**
> On ne peut mélanger une colonne et une fonction dans la projection

Il est possible de grouper les valeurs

```sql
# Renvoie des groupes des valeurs de col
SELECT col FROM table GROUP BY col
```

> **Attention :**
> Il n'est pas possible d'utiliser `GROUP BY` sur des colonnes non groupées

Par contre les fonctions agissent sur chaque groupe, ainsi il est possible d'écrire

```sql
# Renvoie des groupes des valeurs de col avec le nombre d'occurence de cette valeur dans la table
SELECT col, COUNT(*) FROM table GROUP BY col
```

Pour sélectionner des groupes on peut utiliser :

```sql
# Renvoie des groupes des valeurs de col si la valeur minimale du groupe dans la colonne col2 est supérieure à x avec la valeur minimale de col2 de ce groupe dans la table
SELECT col1, MIN(col2) FROM table GROUP BY col1 HAVING MIN(col2) > x
```

Les opérations sont executées dans cet ordre :

- `WHERE`
- `GROUP BY`
- `HAVING`
- `ORDER BY`
- `LIMIT/OFFSET`
- `SELECT` à la fin bien qu'on le mette en tête de la requête

Ainsi une clause valide est

```sql
SELECT * WHERE cond GROUP BY col HAVING cond2 ORDER BY col2 LIMIT 3 OFFSET 2
```

IV.4 Sous requêtes

Il est possible d'écrire une sous requête :

```sql
# Ici on sélectionne seulement les éléments donc la valeur col est supérieure à la valeur moyenne de col
SELECT * FROM table WHERE col > (SELECT AVG(col) FROM table)
```

Il est donc aussi possible d'utiliser cette syntaxe avec des `IN`

```sql
# Ici on va sélectionner seulement les lignes dont la valeur de col correspond à la condition cond
SELECT * FROM table WHERE col IN (SELECT DISTINCT col FROM * WHERE cond)
```

Le `col AS nameBis` permet de renommer une colonne

Si on reçoit un tableau, on peut sélectionner dans les réponses

```sql
# Ainsi on renvoie la moyenne d'une colonne col2 telle que ses éléments vérifient la condition
SELECT AVG(resp.colName) FROM (SELECT col1, col2 AS colName FROM table WHERE cond) AS resp
```

IV.5 Combiner les tables

Il est possible de combiner des tables

```sql
# Sélectionne dans le produit cartésien des deux tables
SELECT * FROM table1, table2
```

Mais en faisant ça on va avoir plein de lignes qui n'ont pas de sens, ainsi si on veut garder seulement les lignes qui nous intéressent

```sql
# Sélectionne dans le produit cartésien des deux tables seulement les éléments donc la col1 de la table 1 est le même que celui de la col 2 de la table 2
SELECT * FROM table1, table2 WHERE table1.col1 = table2.col2
```

Mais pour éviter ça on peut aussi de manière équivalente écrire :

```sql
# On sélectionne les éléments de la table1 en ajoutant la table2 si la condition est vérifiée, le ON est donc un WHERE
SELECT * FROM table1 JOIN table2 ON table1.col1 = table2.col2
```

Le produit cartésien n'est donc qu'une manière de jointure

On peut aussi utiliser le `LEFT JOIN` qui permet de garder un élément de la première table même si il n'a pas d'équivalent dans la seconde table

```sql
# On sélectionne les éléments de la table1 en concaténant les éléments dont la condition est vérifiée, et rien si il n'y a pas d'équivalent
SELECT * FROM table1 LEFT JOIN table2 ON table1.col1 = table2.col2
```

On peut faire l'union de deux requêtes

```sql
# On a les éléments qui vérifient la cond1 ou cond2
SELECT * FROM table WHERE cond1 UNION SELECT * FROM table WHERE cond2
```

> **Attention :**
> Pour utiliser l'union il faut juste que les types sont compatibles mais pas les noms de colonne

On peut aussi faire l'intersection de deux requêtes

```sql
# On a les éléments qui vérifient la cond1 et cond2
SELECT * FROM table WHERE cond1 INTERSECT SELECT * FROM table WHERE cond2
```

On peut faire des différences ensemblistes avec `MINUS` ou `EXCEPT`

IV.6 Créer une BDD

Pour créer une base de données on utilisera

```sql
CREATE TABLE IF NOT EXISTS table (
  col1 TYPE1,
  col2 TYPE2,
  col3 TYPE3
)
```

Si on veut limiter le nombre de caractères, on peut le préciser entre parenthèses, par exemple `VARCHAR(6)` pour avoir des chaînes d'au plus 6 caractères

On peut définir une **clé primaire** qui ne peut avoir 2 fois la même valeur, on indiquera `PRIMARY KEY` après le type :

```sql
CREATE TABLE IF NOT EXISTS table (
  col1 TYPE1 PRIMARY KEY,
  ...
)
```

Les autres attibuts seront dépendant de la clé primaire : si on connaît la clé primaire on peut connaître les autres valeurs associées à la liste

Si on a une clé primaire dans un GROUP BY autorise à projeter sur tous les éléments (pas comme précédemment)

Il y a au plus une clé primaire par table, et une valeur `NULL` ne peut être une valeur pour cette case

On peut définir un clé étrangère qui vont être des liens entre les différentes tables

```sql
CREATE TABLE IF NOT EXISTS table (
  ...,
  FOREIGN KEY (col) REFERENCES table(col)
)
```

Il est aussi possible de modifier une table en utilisant `ALTER TABLE`

Pour insérer dans une table on utilise :

```sql
INSERT INTO table (col1, col2, col3) VALUES (value1, value2, value3)
```

On peut modifier un élement :

```sql
UPDATE table SET col1 = value WHERE cond
```

On peut aussi supprimer un élement :

```sql
DELETE FROM table WHERE cond
```

IV.7 Type entités

Les types entités sont liées par des types associations

A faire (Cardinalité)

On précise les cardinalités :

- $1,1$ en liaison avec une et une seule entité
- $1,n$ en liaison avec au moins une autre entité
- $0,1$ en liaison avec au plus une autre entité
- $0,n$ en liaison avec un nombre quelconque d'entités

Table des matières

I Introduction au C 3
I.1 Variables 3
I.2 Opérateurs 3
I.3 Structures de contrôle 3
I.4 Fonctions 4
I.5 Tableaux en C 5
II Introduction au OCaml 5
I Structures de données 6
II Piles, files, dictionnaires 6
III Arbres 6
IV Graphes 6
IV.1 Recherche de plus cours chemin 6
IV.1.a Graphes avec poids négatifs 6
I Bases 7
I.1 Fonctions 7
I.2 Algorithmes de tri 7
II Récursion 7
III Stratégies algorithmiques 7
IV SQL 7
IV.1 Généralités 7
IV.2 Requêtes 7
IV.3 Fonctions 8
IV.4 Sous requêtes 9
IV.5 Combiner les tables 10
IV.6 Créer une BDD 11
IV.7 Type entités 11
Table des matières 13