Notes de cours: Probabilité - Language C - Théorie des Graphes - Architecture des ordinateurs

BASSMA "Abby" ABISOUROUR

23/09/2024 - 24/09/2024

Lundi 23 Septembre 2024

M1.1.1 Algorithmique - A. Ettalbi (9h00 à 10h40)

Cours reporté.

M1.3.1 Probabilité I - I. Amrani (11h00 à 12h40)

Probabilités et Indépendance

Formule de probabilité conditionnelle

La probabilité de A sachant X est notée :

$$P(A|X) = \frac{P(A \cap X)}{P(X)} \quad (\text{si } P(X) \neq 0).$$

Indépendance de deux événements

Deux événements A et B sont dits indépendants si la probabilité de l'un n'est pas influencée par la réalisation de l'autre. Cela s'exprime ainsi :

$$P(A \cap B) = P(A) \times P(B).$$

Par exemple :

1. On jette une pièce deux fois : les résultats des deux jets sont indépendants.
2. On tire deux cartes sans remise : ici, les tirages sont dépendants car le second tirage dépend du premier.

Indépendance mutuelle

Si $A_1, A_2, \ldots, A_n$ sont des événements, ils sont dits *mutuellement indépendants* si pour tout ensemble d'événements parmi eux, on a :

$$P(A_1 \cap A_2 \cap \cdots \cap A_n) = P(A_1) \times P(A_2) \times \cdots \times P(A_n).$$

Cela signifie que chaque sous-ensemble d'événements est indépendant du reste.

Indépendance conditionnelle

Deux événements A et B sont conditionnellement indépendants sachant un événement C si :

$$P(A \cap B|C) = P(A|C) \times P(B|C).$$

Analyse combinatoire

L'analyse combinatoire est essentielle pour le dénombrement en probabilités.

Principe de multiplication

Si pour aller de A à B il y a 3 chemins, et de B à C il y a 4 chemins, alors il y a en tout $3 \times 4 = 12$ chemins pour aller de A à C.

Permutations

Une permutation d'ordre n consiste à organiser n objets dans un ordre particulier. Le nombre de permutations est donné par :

$$P_n = n! = n \times (n-1) \times \cdots \times 1.$$

Par exemple, pour $n = 3$ (avec a, b, et c),
il existe $3! = 6$ permutations : $(a, b, c), (a, c, b), (b, a, c), (b, c, a), (c, a, b), (c, b, a)$.

Arrangements

Un arrangement d'ordre p est une suite ordonnée de p objets choisis parmi n objets, sans répétition. Le nombre d'arrangements est :

$$A_{n,p} = \frac{n!}{(n-p)!}.$$

Exemple : Pour 3 éléments (a, b, c),
les arrangements d'ordre 2 sont $(a, b), (a, c), (b, a), (b, c), (c, a), (c, b)$.

Combinaisons

Les combinaisons sont des sélections de p objets parmi n objets, sans tenir compte de l'ordre. Le nombre de combinaisons est donné par :

$$C_{n,p} = \frac{n!}{p!(n-p)!}.$$

Exemple : Les combinaisons de 3 objets (a, b, c) par groupe de 2 sont $(a, b), (a, c), (b, c)$.

Permutations avec répétition

Si certains objets sont identiques, le nombre de permutations est ajusté :

$$P = \frac{n!}{k_1! \times k_2! \times \cdots \times k_r!}$$

où $k_1, k_2, \ldots, k_r$ sont les répétitions des objets.

M1.1.2 Langage C - M. Guermah (14h00 à 15h40)

Les ordinateurs comprennent plusieurs couches de langages de programmation permettant de communiquer avec la machine :

- **Langage machine** : C'est le plus bas niveau, directement interprété par l'ordinateur.
- **Langages de haut niveau** : Plus compréhensibles pour les humains, mais nécessitent d'être convertis en langage machine.

Niveaux de langages de programmation

Les langages de programmation sont classés selon la manière dont ils sont exécutés :

- **Compilateur** : Traduit tout le programme en langage machine avant son exécution.
  - Exemple : Langage C, C++.
- **Interpréteur** : Lit et exécute le code ligne par ligne.
  - Exemple : Python, JavaScript, HTML (interprété par le navigateur).
- **Semi-compilé/Semi-interprété** : Certains langages comme Java sont compilés en bytecode, puis interprétés par une machine virtuelle.
  - Exemple : Java.

Paradigmes de programmation

- **Programmation procédurale** : Style utilisé en C, où le code est structuré en fonctions (ou procédures).
- **Programmation orientée objet** : Utilisé dans des langages comme Java ou C++, le code est organisé en objets (pas très utilisé en C pur).

Historique du langage C

Le langage C a été créé dans les années 70 par **Dennis Ritchie** chez Bell Labs. Son but initial était de fournir un langage plus simple et performant pour développer des systèmes d'exploitation comme **Unix**. Aujourd'hui, le C est encore largement utilisé pour ses performances.[1]

Le succès du langage C

- **Portabilité** : Les programmes en C peuvent être facilement exécutés sur de nombreux systèmes.
- **Efficacité** : C est très proche du matériel, ce qui le rend rapide.

[1] Idée de projet : Créer un petit compilateur en C

- **Flexibilité** : Il permet de programmer des systèmes bas niveau comme des applications complexes.

Structure d'un programme en C

Voici un exemple de structure d'un programme en C :

Listing 1: Exemple de programme en C

```c
#include <stdio.h>  // Directive du preprocesseur

int main() {        // Fonction principale
    printf("Hello, world!\n");
    return 0;
}
```

- **Directive au préprocesseur** : `#include` permet d'importer des bibliothèques.
- **Fonction `main`** : C'est le point d'entrée du programme.
- **Blocs d'instruction** : Délimités par des accolades {}, ils contiennent les instructions à exécuter.

Bonnes pratiques de programmation en C

- **Éviter les fonctions internes** : Il est déconseillé de définir des fonctions à l'intérieur d'autres fonctions.
- **Utilisation des constantes** : Ce sont des valeurs fixes que le programme ne peut pas modifier.

Les variables en C

Les variables permettent de stocker des valeurs. Elles doivent être déclarées avant leur utilisation.

Types de variables courants :

- **char** : Un seul caractère (1 octet).
- **int** : Un entier (2 ou 4 octets selon le système).
- **long int** : Un entier plus grand.
- **float** : Nombre à virgule flottante (4 octets).
- **double** : Nombre à virgule flottante avec double précision (8 octets).
- **long double** : Nombre à virgule avec une précision encore plus grande (généralement 10 ou 12 octets).

Voici un exemple de déclaration et d'utilisation de variables en C :

Listing 2: Exemple de variables en C

```c
#include <stdio.h>

int main() {
    int age = 25;               // Declaration d'un entier
    float taille = 1.75;        // Declaration d'un float
    char initiale = 'A';        // Declaration d'un char

    printf("Age : %d\n", age);
    printf("Taille : %.2f\n", taille);
    printf("Initiale : %c\n", initiale);

    return 0;
}
```

Mardi 24 Septembre 2024

M1.4.1 Théorie des Graphes - A. Bellabdaoui (9h00 à 10h40)

Introduction

Le cours commence par des rappels sur les **terminologies** pour s'assurer que tout le monde est sur la même page. Les démonstrations et exercices visent à mieux comprendre le développement des algorithmes pour résoudre les problèmes classiques des graphes.

Mises en garde du professeur :

- Attention à bien maîtriser la **terminologie** propre aux graphes.
- Les démonstrations ne sont pas là pour leur forme, elles sont importantes pour développer des **algorithmes**.
- Soyez attentifs à la **complexité** des algorithmes.

Démonstration : Une chaîne élémentaire est simple

L'objectif est de démontrer par contraposée qu'une chaîne élémentaire est simple. Il existe bien sûr plusieurs démonstrations, voici celle qu'on a abordé pendant le cours.

Démonstration (par contraposée)

Soit L une chaîne de x à y, définie par :

$$L = xu_1u_2 \dots u_py.$$

Supposons que L **n'est pas simple**. Cela signifie qu'il existe des sommets u_i et u_j avec $i < j$ tels que $u_i = u_j$. Deux cas se présentent alors :

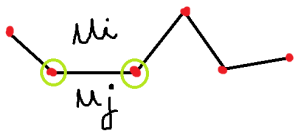

Figure 1: Les deux sommets (entourés en vert) sont revisités.

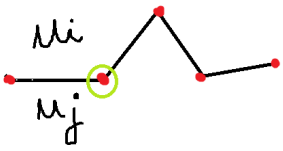

Figure 2: Le sommet (entouré en vert) est revisité

1. Si u_i et u_j **ne sont pas adjacents**, alors on peut représenter ce cas avec un schéma montrant qu'au moins deux sommets sont revisités.
2. Si u_i et u_j **sont adjacents**, le parcours forme un cycle, et on peut montrer qu'au moins un sommet est revisité.

Dans les deux cas, cela montre que L n'est pas élémentaire. Par contraposée, si L est élémentaire, alors elle est simple.

□

Lemme de Kœnig et Algorithme

Soit L une chaîne allant de x à y.

- Si L est élémentaire, alors c'est fini.
- Sinon, L est non élémentaire.

Dans ce cas, il existe un parcours fermé que l'on peut éliminer pour obtenir une chaîne élémentaire.[2]

Algorithme pour éliminer les parcours fermés

Voici une idée d'algorithme :

- Parcourir L à partir de x de manière élémentaire.
- Dès qu'un sommet z est revisité, éliminer le parcours C entre la première et la seconde apparition de z.
- Répéter cette opération jusqu'à ce que tous les cycles fermés soient éliminés.

Le résultat final est un chemin élémentaire (par construction). Ce processus permet de démontrer le **lemme de Kœnig**.

[2]L'algorithme derrière la construction élémentaire de la chaîne m'étais pas très intuitif, j'ai essayé d'être le plus fidèle aux explications du Prof. L'idée, c'est en gros de la construire au fur et à mesure qu'on parcours la chaîne L. La complexité de cette algorithme est la taille de cette chaîne L.

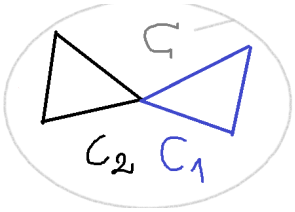

Figure 3: Un cycle C décomposé en deux cycles élémentaires: C_1 et C_2

Définition des cycles

- Un **cycle** est un parcours qui commence et finit au même sommet, sans revisiter aucun sommet interne.
- Un cycle est dit **élémentaire** s'il ne contient pas de sous-cycles.

Décomposition des cycles

Tout cycle est une réunion disjointe de cycles élémentaires. Cet algorithme permet de décomposer un cycle en plusieurs cycles élémentaires.[3]

Lemme des poignées de main (et un Corollaire)

Le **lemme des poignées de main** est un résultat classique en théorie des graphes qui peut se démontrer simplement. Voici une idée pour la démonstration. On rappelle que le Lemme stipule que *la somme des degrés des sommets d'un graphe est égal à 2 fois son nombre d'arêtes.)*

Démonstration :

- Chaque sommet dans un graphe a un degré[4] pair ou impair.
- La somme des degrés dans un graphe est égale à deux fois le nombre d'arêtes, ce qui implique que cette somme est toujours paire.
- Par conséquent, le nombre de sommets de degré impair est toujours pair.

Ce résultat est souvent utilisé pour démontrer des propriétés de parcours dans les graphes.

Conclusion

Ce cours m'a personellement beaucoup plu cat il a introduit plusieurs concepts et démonstrations importants pour comprendre la théorie des graphes. (Et moi qui adore les maths xD). L'accent a été mis sur la terminologie et les algorithmes, sans oublier la complexité.

[3] C'est la même idée que l'algorithme précédant. On peut schématiser

[4] Le degré d'un sommet x d'un graphe G, noté $d(x)$, est le nombre d'arêtes incidentes à x

M1.2.1 Architecture des ordinateurs - Z. Alaoui (11h00 à 12h40)

Ce cours traite (bien évidemment) des architectures des ordinateurs, avec un focus sur les architectures **x86** et **x86_64**, qui sont couramment utilisées dans les processeurs modernes.

Architecture x86 et x86_64

L'architecture **x86** fait référence à une famille de jeux d'instructions pour les processeurs, originellement développée par Intel avec le processeur 8086. Les processeurs **x86_64** sont une extension de cette architecture qui permet la gestion de plus grandes quantités de mémoire (64 bits au lieu de 32 bits).

Composantes des instructions : Opcode et Opérande

Les instructions exécutées par un processeur sont divisées en deux parties principales :

- **Opcode** (Operation Code) : représente l'action que le processeur doit effectuer (comme additionner, soustraire, charger un registre, etc.). *Exemple: l'opération de l'addition sur les entiers*
- **Opérande** : les données sur lesquelles l'opération doit être effectuée (par exemple, les valeurs ou les adresses de mémoire). *Exemple: les entiers 2 et 3 qu'on souhaitent additionner*

Cycle d'exécution des instructions

Le CPU exécute les instructions en suivant un cycle appelé le **cycle d'exécution**. Ce cycle se compose de trois étapes principales :

1. **Recherche** : le CPU récupère l'instruction en mémoire.
2. **Décodage** : le CPU interprète l'instruction (quelles opérations et quelles données sont concernées).
3. **Exécution** : le CPU exécute l'instruction en manipulant les données.

Diagramme de bloc du CPU

Voici un résumé des composants principaux d'un processeur :

- **Unité de contrôle** : coordonne les différentes opérations du processeur.
- **ALU (Unité arithmétique et logique)** : effectue les calculs et les opérations logiques.
- **Registres** : petites mémoires rapides qui stockent les données temporaires pendant l'exécution des instructions.[5]

[5] On a abordé pas mal d'exemples de registres: PC: Program Counter, MAR: Memory Adress Register, MBR: Memory Buffer Register (également appelé MDR: Memory Data Register), IR: Instruction Register, AC: Accumulateu, Pile (LiFo - Last In First Out)... On a parlé de leur rôle et comment ils rentrent en interaction pour exécuter les instructions.

Traitement des instructions : Micro-opérations

Lorsqu'une instruction complexe est décodée, elle est souvent décomposée en **micro-opérations**, qui sont des étapes plus simples que le processeur exécute pour accomplir la tâche complète.

Exercice potentiel

Un exercice classique pourrait consister à lister les étapes du cheminement d'une instruction dans le processeur, en décrivant comment chaque composant est impliqué.

Architectures RISC et CISC

Deux grandes familles d'architectures existent :

- **RISC (Reduced Instruction Set Computer)** : une architecture où les instructions sont simples et s'exécutent en un seul cycle. Elle est plus efficace pour certaines applications.
- **CISC (Complex Instruction Set Computer)** : une architecture où les instructions sont plus complexes, mais peuvent réaliser plusieurs actions en une seule instruction. Les architectures **x86** et **x86_64** sont basées sur CISC.

Architectures Harvard vs Von Neumann

Architecture Harvard

Dans l'architecture **Harvard**, la mémoire pour les instructions (programme) est séparée de la mémoire pour les données. Cela permet au processeur de lire des instructions et des données en parallèle, ce qui augmente l'efficacité.

Architecture Von Neumann

L'architecture **Von Neumann**, en revanche, utilise une mémoire commune pour les instructions et les données. Cela simplifie la conception, mais peut ralentir les performances car les instructions et les données doivent être récupérées séquentiellement.

Exemple de processeur : Intel 8086

Le processeur **Intel 8086** est l'un des premiers processeurs basés sur l'architecture x86. Il a introduit un jeu d'instructions CISC, et bien qu'il soit ancien, il a servi de base à de nombreuses générations de processeurs modernes.

TD M1.3.1 Probabilité - I. Amrani (14h00 à 15h40)

Correction des exercices du support PDF envoyé par le prof.