Algorithmique avancée - Le Nombre de Catalan

Capiaux Bastien L3A 21002589

06/12/2023

En mathématiques, les nombres de Catalan C(n) sont une suite de nombres entiers utilisés dans divers problèmes en combinatoire. Ils permettent notamment de dénombrer le nombre d'arbres binaires possédant exactement n nœud internes, ou encore le nombre de manières de parenthéser correctement un mot de longueur n avec autant de parenthèses ouvrantes que fermantes, et une parenthèse ouvrante arrive toujours avant la parenthèse qui la referme.

Contents

1 Énoncé

Implémenter plusieurs algorithmes permettant de calculerles nombres C(n), et les comparer en temps d'exécution. Écrire un programme qui affiche toutes les manières de bien parenthéser un mot de longueur n, discuter de son efficacité.

2 A quoi servent les fichiers ?

2.1 Makefile et readme

Le Makefile permet d'executer des commandes simple afin de nous faciliter la vie. Grâce à "make clean" nous pouvons nettoyer le projet des fichiers temporaire et binaire. Grâce à "make dist" le fichier est directement compressé. Dans le make file vous pourrez retrouver une ligne de commande pour nettoyer votre terminal, compiler le projet et l'exécuter sur la même ligne.

2.1.1 La compilation

```
$ make
```

2.1.2 L'exécution

```
$ ./prog
```

2.2 main.c

main.c est le fichier qui contient le main et qui va utiliser nos autres fichiers compris dans "sources" et "headers".

2.3 Fichiers - sources

2.3.1 catalan.c

catalan.c regroupe toutes les fonctions que le nombre de Catalan a besoin pour fonctionner.

2.3.2 parenthese.c

parenthese.c regroupe toutes les fonctions pour que l'on puisse parenthésé un mot de longueur n.

2.4 Fichiers - headers

2.4.1 catalan.h

Permet d'utiliser toutes les fonctions de "catalan.c" dans un autre fichiers tel que ici notre "main.c".

2.4.2 parenthese.h

Permet d'utiliser toutes les fonctions de "parenthese.c" dans un autre fichiers tel que ici notre "main.c".

3 Contenus des fichiers

3.1 Catalan

3.1.1 Header

```c
struct Catalan_Func
{
    unsigned long long int (*recursive)(int);
    unsigned long long int (*iterative)(int);
    unsigned long long int (*coefficient_binomial)(int);
    void (*tests)(unsigned long long int (*)(int));
};
extern const struct Catalan_Func Catalan;
```

Structure permettant de créer un objet Catalan permettant de lancer les fonctions qui lui sont attribuées.

"extern const struct Catalan_Func Catalan;" Sert a initialisé la structure ailleurs que dans le fichier header.

```c
unsigned long long int binomial_coefficient(int N);
unsigned long long int binomial_coefficient_calcul(int N, int k);
unsigned long long int iterative(int N);
unsigned long long int recursive(int N);

float c_execution_time(unsigned long long int (*function_p)(int), int);
void c_execution_time_test(unsigned long long int (*function_p)(int));
void c_tests(unsigned long long int (*function_p)(int));
char *c_function_name(void *function_p);
```

Répertoire des fonctions présente dans "catalan.c".

3.1.2 Source

```c
#include "../headers/catalan.h"

const struct Catalan_Func Catalan = {
    .recursive = recursive,
    .iterative = iterative,
    .coefficient_binomial = binomial_coefficient,
    .tests = c_tests
};
```

Initialisation de "extern const struct Catalan_Func Catalan;"

sources > C catalan.c > iterative(int)

```c
unsigned long long int recursive(int N) {
    if (N <= 1)
        return 1;
    unsigned long long int somme = 0;
    int index;
    for (index = 0; index < N; index += 1)
        somme += recursive(index) * recursive(N - index - 1);
    return somme;
}
```

Fonction permettant de calculer le nombre de catalan de manière récursive.

sources > C catalan.c > iterative(int)

```c
unsigned long long int iterative(int n) {
    unsigned long long int *nombresCatalan = malloc((sizeof *nombresCatalan) * (n + 1));
    unsigned long long int resultat;
    int indexcourant, indexPP;
    nombresCatalan[0] = nombresCatalan[1] = 1;
    for (indexcourant = 2; indexcourant <= n; indexcourant += 1) {
        nombresCatalan[indexcourant] = 0;
        for (indexPP = 0; indexPP < indexcourant; indexPP += 1) {
            nombresCatalan[indexcourant] += nombresCatalan[indexPP] * nombresCatalan
            [indexcourant - indexPP - 1];
        }
    }
    resultat = nombresCatalan[n];
    free(nombresCatalan);
    return resultat;
}
```

Fonction permettant de calculer le nombre de catalan de manière itérative.

sources > C catalan.c > binomial_coefficient(int)

```c
unsigned long long int binomial_coefficient(int N) {
    return binomial_coefficient_calcul(N * 2, N) / (N + 1);
}
unsigned long long int binomial_coefficient_calcul(int N, int k) {
    unsigned long long int result = 1;
    int temp, index;
    temp = N - k;
    if (k > temp) k = temp;
    for (index = 0; index < k; index += 1)
    {
        result *= (N - index);
        result /= (index + 1);
    }
    return result;
}
```

Fonction permettant de calculer le nombre de catalan a l'aide du coefficient binomiale.

sources > C catalan.c > ...

```c
float c_execution_time(unsigned long long int (*function_p)(int), int N) {
    long start = clock();
    function_p(N);
    return (float)(clock() - start) / CLOCKS_PER_SEC;
}
void c_execution_time_test(unsigned long long int (*function_p)(int)) {
    float execut_time;
    float overall_execution_time = 0.f;
    float arret_time = 0.5f;
    unsigned long long int index;
    for (index = 0; index <= 1000; index += 1) {
        execut_time = c_execution_time(function_p, index);
        if (execut_time > arret_time) {
            printf("( %llu ) : \033[33m%f\033[39m supérieur à \033[33m%f\033[39ms -> arret du
            test.\n", index, execut_time, arret_time);
            printf("%s | overall_execution_time: \033[33m%f\033[39ms.\n",c_function_name
            (function_p),  overall_execution_time);
            return;
        }
        overall_execution_time += execut_time;
    }
    printf("%s | overall_execution_time: \033[33m%f\033[39ms.\n",c_function_name
    (function_p),  overall_execution_time);
}

void c_tests(unsigned long long int (*function_p)(int)) {
    printf("\nExecution time test:\n");
    c_execution_time_test(function_p);
}

char *c_function_name(void *function_p) {
    if (function_p == binomial_coefficient)
        return "Catalan -> binomial_coefficient";
    else if (function_p == iterative)
        return "Catalan -> iterative";
    else if (function_p == recursive)
        return "Catalan -> recursive";
    else
        return "unknown_function";
}

```

Série de fonctions de test pour Catalan:

c_test Permet de lancer les tests.

c_fonction_name renvoie le nom de la fonction demander.

c_execution_time donne le temps d'exécution d'une fonction.

c_execution_time_test permet de faire le test plusieurs fois sur une même fonction avec différentes valeurs et renvoie le temps totale d'exécutions.

3.2 Parenthèse

3.2.1 Header

```c
struct Parenthese_Func {
    void (*recursive)(int);
    // void (*iterative)(int);
    void (*tests)(void (*)(int));
};
extern const struct Parenthese_Func Parenthese;
```

Structure permettant de créer un objet Parenthese permettant de lancer les fonctions qui lui sont attribuées.

"extern const struct Parenthese_Func Parenthese;" Sert a initialisé la structure ailleurs que dans le fichier header.

```c
// struct Parenthese_struct;
// typedef struct Parenthese_struct{
//     char *str;
//     int left;
//     int right;
// }Parenthese_struct;

// extern Parenthese_struct *queue;


void parenthese_rec(int);
void generateParenthesesRec(char [], int, int, int );
// void generateParenthesesIte(int);

float p_execution_time(void (*function_p)(int), int N);
void p_execution_time_test(void (*function_p)(int));
void p_tests(void (*function_p)(int));
char *p_function_name(void *function_p);
```

Répertoire des fonctions présente dans "parenthese.c".

3.2.2 Source

```c
const struct Parenthese_Func Parenthese = {
    .recursive = parenthese_rec,
    // .iterative = generateParenthesesIte,
    .tests = p_tests
};
```

Définition de Parenthese pour appeler nos fonctions dans le "main.c".

```c
void generateParenthesesRec(char current[], int left, int right, const int N) {
    if (left + right == 2 * N) {
        printf("%s\n", current);
        return;
    }
    if (left < N) {
        current[left + right] = '(';
        generateParenthesesRec(current, left + 1, right, N);
    }
    if (right < left) {
        current[left + right] = ')';
        generateParenthesesRec(current, left, right + 1, N);
    }
}
void parenthese_rec(const int N) {
    char current[2 * N + 1];  // +1 for the null terminator
    current[2 * N] = '\0';    // null terminator
    generateParenthesesRec(current, 0, 0, N);
}
```

Permet d'affiché le bon parenthèsage d'un mot de longueur n avec C(n) print.
(Se qui est très long a afficher)

```c
float p_execution_time(void(*function_p)(int), int N) {
    long start = clock();
    function_p(N);
    return (float)(clock() - start) / CLOCKS_PER_SEC;
}
void p_execution_time_test(void(*function_p)(int)) {
    float execut_time;
    float overall_execution_time = 0.f;
    float arret_time = 2.0f;
    unsigned long long int index;
    for (index = 1; index <= 50; index += 1)
    {
        execut_time = p_execution_time(function_p, index);
        printf("%s ( %llu ) : \033[33m%f\033[39ms\n\n", p_function_name(function_p), index, execut_time);
        if (execut_time > arret_time)
        {
            printf("( %llu ) : \033[33m%f\033[39m supérieur à \033[33m%f\033[39ms -> arret du test.\n", index,
            execut_time, arret_time);
            printf("%s | overall_execution_time: \033[33m%f\033[39ms.\n",p_function_name(function_p),
            overall_execution_time);
            return;
        }
        overall_execution_time += execut_time;
    }
    printf("%s | overall_execution_time: \033[33m%f\033[39ms.\n",p_function_name(function_p),
    overall_execution_time);
}

void p_tests(void(*function_p)(int)) {
    printf("\nExecution time test:\n");
    p_execution_time_test(function_p);
}

char *p_function_name(void *function_p) {
    if (function_p == parenthese_rec)
        return "Parenthese -> recursive";
    // else if (function_p == generateParenthesesIte)
    //  return "Parenthese -> iterative";
    else
        return "unknown_function";
}
```

p_test Permet de lancer les tests.

p_fonction_name renvoie le nom de la fonction demander.

p_execution_time donne le temps d'exécution d'une fonction.

p_execution_time_test permet de faire le test plusieurs fois sur une même fonction avec différentes valeurs et renvoie le temps totale d'exécutions.

3.3 Main

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>

#include "headers/catalan.h"
#include "headers/parenthese.h"

int main() {
    Parenthese.tests(Parenthese.recursive);
    printf("\n");
    Catalan.tests(Catalan.recursive);
    printf("\n");
    Catalan.tests(Catalan.iterative);
    printf("\n");
    Catalan.tests(Catalan.coefficient_binomial);
    return 0;
}
```

Le fichier "main.c" Prends le const struct de Catalan et Parenthese et applique la fonction de test sur ces fonctions.