Informatique

TP

Algorithmique- Arithmétique, Retour sur trace

Ipesup

Problème des N reines

Le problème des N reines consiste à placer N reines sur un échiquier $N \times N$ de telle sorte qu'aucune reine ne menace une autre. On représente une solution par un tableau `q` de taille N, où `q[i]` désigne la colonne de la reine placée sur la ligne i.

1. Montrer qu'une solution est valide si et seulement si pour tous $i \neq j$:

$$\texttt{q[i]} \neq \texttt{q[j]} \quad \text{et} \quad |\texttt{q[i]} - \texttt{q[j]}| \neq |i - j|.$$

2. Implémenter en C une fonction `bool check_queen(int q[], int N, int row)` qui vérifie que la reine de la ligne `row` n'est en conflit avec aucune reine des lignes précédentes.
3. Écrire la fonction récursive de retour sur trace `int solve_queens(int q[], int N, int row)` qui renvoie le nombre total de solutions pour un échiquier $N \times N$.
4. Donner le nombre exact de solutions pour $N = 8$. Que se passe-t-il pour $N = 2$ et $N = 3$?
5. **(Optimisation)** Proposer une structure de données permettant de vérifier en $O(1)$ l'absence de conflit et modifier `check_queen` en conséquence.

Chiffrement RSA et arithmétique modulaire

Rappel : clé publique (n, e) avec $n = pq$, $\phi(n) = (p-1)(q-1)$, $\gcd(e, \phi(n)) = 1$; clé privée $d = e^{-1} \pmod{\phi(n)}$; chiffrement $c = m^e \bmod n$, déchiffrement $m = c^d \bmod n$.

1. Soit $p = 61$, $q = 53$. Calculer n et $\phi(n)$.
2. On choisit $e = 17$. Vérifier que $\gcd(e, \phi(n)) = 1$ en déroulant l'algorithme d'Euclide à la main.
3. À l'aide de l'algorithme d'Euclide étendu, calculer $d = e^{-1} \pmod{\phi(n)}$.
4. Chiffrer le message $m = 65$ avec la clé publique (n, e).
5. Déchiffrer le chiffré obtenu et vérifier qu'on retrouve $m = 65$.
6. Expliquer pourquoi connaître $\phi(n)$ permettrait de retrouver p et q à partir de n.
7. **(Complexité)** Donner la complexité (en nombre de bits de n) du calcul de d et du déchiffrement d'un message.

Crible d'Ératosthène et théorie des nombres

1. En utilisant $\displaystyle\sum_{p \leq N} \frac{1}{p} \sim \ln \ln N$, montrer que la complexité du crible est $O(N \ln \ln N)$.
2. Donner la complexité spatiale. Décrire le principe du *crible segmenté* qui réduit la mémoire à $O(\sqrt{N})$.
3. Deux entiers p et $p+2$ sont des *premiers jumeaux* si tous deux sont premiers. Modifier le crible pour lister tous les couples de premiers jumeaux inférieurs à N.
4. **(Indicatrice d'Euler)** Soit $\varphi(n)$ le nombre d'entiers de $\{1, \ldots, n\}$ premiers

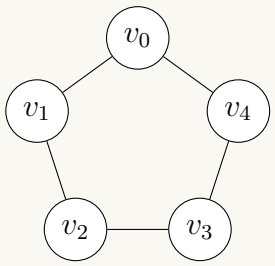
avec n. Montrer que $\varphi(p) = p - 1$ pour p premier. Expliquer comment calculer $\varphi(n)$ pour tout $n \le N$ en $O(N \ln \ln N)$ par une variante du crible.

5. Calculer le nombre d'entiers de $\{1, \ldots, 100\}$ inversibles modulo 100 et les lister.

Coloration de graphe par retour sur trace

Soit $G = (V, E)$ un graphe non orienté à n sommets. On souhaite le colorier avec au plus k couleurs (deux sommets adjacents de couleurs distinctes). Représentation : matrice d'adjacence `adj[n][n]`, coloration `color[n]` (initialisé à -1).

1. Montrer que le cycle C_5 nécessite au moins 3 couleurs.



2. Écrire en C `bool is_valid(int color[], int adj[][MAXN], int n, int v)` qui vérifie l'absence de conflit pour le sommet v.

3. Écrire la fonction de retour sur trace
`bool solve_coloring(int color[], int adj[][MAXN], int n, int k, int v)`.

4. **(Analyse)** En notant Δ le degré maximal, montrer que $k = \Delta + 1$ couleurs suffisent toujours. En déduire une borne sur la taille de l'arbre de recherche.

5. Expliquer pourquoi le retour sur trace reste pratique pour des instances de petite taille, malgré la NP-complétude du problème.

Théorème chinois des restes

Le théorème chinois des restes (TCR) affirme que si $m_1, \ldots, m_r$ sont deux à deux premiers entre eux, le système

$$x \equiv a_i \pmod{m_i}, \quad i = 1, \ldots, r$$

possède une unique solution modulo $M = m_1 \cdots m_r$.

1. Résoudre et vérifier :

$$x \equiv 2 \pmod 3, \quad x \equiv 3 \pmod 5, \quad x \equiv 2 \pmod 7.$$

2. Montrer que pour deux équations la solution est

$$x \equiv a_1 M_1 (M_1^{-1} \bmod m_1) + a_2 M_2 (M_2^{-1} \bmod m_2) \pmod M, \quad M_i = M/m_i.$$

3. Implémenter en OCaml `crt : (int * int) list -> int * int` qui prend une liste de paires (a_i, m_i) et renvoie (x, M) en utilisant `extended_gcd`.

4. Expliquer comment le TCR permet de réduire le calcul de $A \times B \bmod N$ (grand N) à des multiplications sur de petits entiers.

5. Montrer que tout $x \in \{0, \ldots, M-1\}$ est représenté de façon unique par $(x \bmod m_1, \ldots, x \bmod m_r)$ et que addition et multiplication s'effectuent composante par composante.

Génération de labyrinthes par retour sur trace

Un *labyrinthe parfait* sur une grille $n \times m$ est un labyrinthe où il existe un chemin unique entre deux cases quelconques.

Algorithme (DFS aléatoire) :

— Partir d'une case de départ, la marquer visitée.
— Choisir aléatoirement un voisin non visité, abattre le mur intermédiaire.
— Si aucun voisin non visité, revenir en arrière (*backtrack*).
— Terminer quand toutes les cases sont visitées.

Murs représentés par `h_wall[n][m]` (horizontal) et `v_wall[n][m]` (vertical).

1. Montrer que l'algorithme génère un arbre couvrant de la grille, ce qui garantit l'existence d'un chemin unique entre toute paire de cases.
2. Implémenter en C
   `void generate(bool visited[][], bool h_wall[][], bool v_wall[][], int i, int j, int n, int m)`
   en mélangeant aléatoirement les quatre directions avant de les explorer.
3. Écrire `bool solve_maze` qui trouve par retour sur trace un chemin de $(0, 0)$ à $(n-1, m-1)$ et le renvoie comme liste de cases.
4. **(Complexité)** Donner la complexité en temps et en espace. Quel risque pose un appel récursif sur une grille 1000×1000 ?
5. **(Variante itérative)** Réécrire `generate` avec une pile explicite. Quel avantage cela apporte-t-il ?