TD n°14 – Recherche exhaustive

Exercice 1 *Le mot de passe*

Le mot de passe de Jean est formé de 4 caractères qui sont soit des lettres minuscules, soit des lettres majuscules, soit des chiffres. (Vous aurez besoin de la table ASCII, regardez sur internet)

On vous donne une fonction `bool est_correct(char* mdp)` qui indique si le mot de passe proposé est le mot de passe de Jean.

Écrire une fonction `char* bruteforce()` qui trouve le mot de passe de Jean par la force brute.

Exercice 2 *Un diner presque parfait*

Alice a invité $n \in \mathbb{N}$ personnes à son anniversaire, mais ces personnes ne s'entendent pas toutes ensemble.

Elle a réservé un restaurant avec une grande table ronde, et elle voudrait répartir les convives (elle y compris) de sorte à ce que deux personnes qui se détestent ne soient pas côte à côte.

Les convives seront numérotés de 0 à N (Alice est le numéro 0). On vous donne une fonction `deteste : int->int->bool` qui indique si la personne a déteste la personne b (la détestation est réciproque).

Le plan de table sera représenté par une liste de taille $N+1$. Il faut se souvenir que la tête est voisine du dernier élément puisque la table est ronde.

1. Écrire une fonction `est_plan : 'a list -> bool` qui prend en entrée un plan de table t et teste si ce plan de table ne contient pas deux voisins qui se détestent.

Pour énumérer tous les plans de table possible, on peut procéder récursivement.

Par exemple si on veut un plan de table avec 0, 1 et 2 :

- On fixe d'abord le premier convive à 0. On énumère alors récursivement les plans de table possibles avec les convives 1 et 2 et on rajoute 0 en tête. Cela donne les plans de table $[0;1;2]$ et $[0;2;1]$.
- Ensuite on fixe le premier convive à 1. On énumère alors récursivement les plans de table possible avec les convives 0 et 2. Cela donne les plans de table $[1;0;2]$ et $[1;2;0]$.
- Enfin on fixe le premier convive à 2. On énumère alors récursivement les plans de table possible avec les convives 0 et 1. Cela donne les plans de table $[2;0;1]$ et $[2;1;0]$.

2. Écrire une fonction `plan_de_table_naif : int -> 'a list` qui prend en entrée N et renvoie un plan de table qui fonctionne en suivant une méthode par force brute.

Exercice 3 *Gestion de troupeau extraterrestre*

Sur une planète longue et fine (qu'on assimilera au segment $[|0, N|]$) vit un fermier extraterrestre. Il possède n vaches, réparties sur la planète à des coordonnées entières $v_i \in [|0, N|]$ et n piquets, plantés à des coordonnées entières $p_i \in [|0, N|]$. Les vaches sont peu actives et restent en permanence à leur position.

Le fermier veut attacher ses vaches à des piquets, de sorte à minimiser la longueur de corde nécessaire. Chaque piquet ne peut être attaché qu'à une seule vache (et réciproquement).

1. Écrire une fonction `solution_exhaustive : int array -> int array -> int array` qui prend en entrée le tableau des positions des vaches, le tableau des positions des piquets et renvoie un tableau de taille n, qui indique à la case i à quel piquet la vache i est attachée dans une solution optimale.
   Cette fonction effectuera une recherche exhaustive, il faut bien réfléchir à quoi ressemble l'espace des solutions pour pouvoir le générer.
2. Réfléchir à une méthode par balayage qui permet de résoudre le problème en $O(nlog(n))$.

Exercice 4 *Nombre d'occurences d'un mot dans un texte*

Proposer une méthode pour déterminer le nombre d'occurences d'un mot m dans un texte t, en C.

Il est recommandé de réfléchir à votre méthode par morceaux, en réfléchissant aux fonctions intermédiaires dont vous pourriez avoir besoin. On pourra, en bonus, proposer des améliorations à la méthode par force brute.