TD 1
Application des fondamentaux

Mathieu Noizet

Février 2025

1 Premiers pas

Dans cet exercice, vous devez programmer différentes fonctions afin de vous familiariser avec la syntaxe du langage.

1. *estBissextile* : Implémentez une fonction *estBissextile* qui prend en entrée une année sous forme d'entier et renvoie *true* si l'année est bissextile, sinon *false*.
2. *estPremier* : Écrire une fonction *estPremier* qui, étant donné un entier positif n, retourne *true* si n est un nombre premier, sinon *false*.
3. *listeNombrePremier* : Écrire une fonction *premiersNombresPpremiers* qui, étant donné un entier positif n, retourne un tableau contenant les n premiers nombres premiers.
4. *initTableau* : Écrire une fonction *genererTableauAleatoire* qui prend en entrée un entier n représentant la taille du tableau et renvoie un tableau de taille n initialisé avec des valeurs aléatoires.*Astuce : utilisez le package math/rand, notamment la fonction Intn*
5. *trie à bulle* : Écrire une fonction *triBulles* qui prend en entrée un tableau d'entiers et modifie ce même tableau afin qu'il soit trié par ordre croissant à l'aide du tri à bulles.
6. *trie par sélection* : Écrire une fonction *triSelection* qui prend en entrée un tableau d'entiers et modifie ce même tableau afin qu'il soit trié par ordre croissant à l'aide du tri par sélection.
7. *recherche dichotomie* : Écrire une fonction *rechercheDichotomique* qui prend en entrée un tableau d'entiers trié ainsi qu'un entier x, et retourne *true* si x est présent dans le tableau, sinon *false* avec l'indice associé .
8. *Groupement par taille* :Écrire une fonction *organiserParTaille* qui prend en entrée une liste de noms et les regroupe dans une *slice* en fonction de leur longueur.

2 Jeu de la vie

Le Jeu de la Vie est un automate cellulaire inventé par *John Conway*. Il se déroule sur une grille de cellules, où chaque cellule peut être vivante (1) ou morte (0). À chaque tour, l'état des cellules est mis à jour en fonction des règles suivantes :

- Une cellule vivante reste en vie si elle a 2 ou 3 voisins vivants, sinon elle meurt.
- Une cellule morte devient vivante si elle a exactement 3 voisins vivants.

Une cellule possède 8 voisins (haut, bas, gauche, droite, et diagonales).

1. Écrire une fonction *initGrille(n, m)* qui renvoie une grille de taille $n \times m$ remplie aléatoirement avec des cellules vivantes et mortes. *Astuces : utilisez le package math/rand vu précédemment pour initialiser la grille avec des états aléatoires.*

2. Écrire une fonction *compterVoisins(grille, i, j)* qui retourne le nombre de cellules vivantes autour de la cellule (i, j).

3. Implémentez une fonction *update* qui prend en entrée une grille de cellules et renvoie une nouvelle grille mise à jour en appliquant les règles du Jeu de la Vie.

4. Écrire une procédure *afficherGrille(grille)* qui affiche la grille dans le terminal. 0 sera affiché " u2588" et 1 sera affiché " ".

5. Écrire une boucle permettant d'afficher l'évolution du jeu au fil du temps. Afin d'effacer l'affichage du terminal, vous pouvez utilisez les packages *os* et *os/exec* avec les instructions avec les instructions suivantes :

```
c := exec.Command("clear")
c.Stdout = os.Stdout
c.Run()
```

   De plus, pour mettre des temps de pause entre chaque affichage, vous pouvez utilisez le package *time* et la fonction *Sleep(int)*.

3 Les vecteurs

Le but de cet exercice est de proposer une *structure* facilitant l'utilisation de vecteurs 2D. Pour ce faire, vous proposerez une structure *vec2i* qui représentera un vecteur 2D d'entiers.

Afin de faciliter l'utilisation des vecteurs, vous devrez implémenter l'ensemble des méthodes permettant les interactions suivantes :

1. L'initialisation d'un vecteur
2. L'addition de deux vecteurs
3. La soustraction de deux vecteurs
4. La multiplication de deux vecteurs
5. Le calcul de la norme d'un vecteur
6. La normalisation d'un vecteur
7. Le calcul du produit scalaire de deux vecteurs
8. Le calcul du produit vectorielle de deux vecteurs

4 La liste chaînée

Une liste chaînée est une structure de données composée d'un ensemble de *nœuds*. Chaque *nœud* contient :

- Un élément de donnée.

  Un pointeur vers le *nœud* suivant de la liste.

- La liste chaînée est dynamique, ce qui signifie que l'on peut ajouter ou supprimer des éléments facilement, sans avoir besoin de redimensionner un tableau.

4.1 Structure de la Liste Chaînée

Commencez par implémenter la structure *Node* qui contient deux champs :

- *Data* : pour stocker l'élément (de type *int* pour simplification).
- *Next* : un pointeur vers le *nœud* suivant de type **Node*.

Implémentez une structure *LinkedList* qui contient un pointeur vers le premier *nœud* de la liste (tête de la liste) :

4.2 Opérations essentielles d'une liste chaînée

Vous devrez implémenter plusieurs fonctions pour manipuler la liste chaînée. Voici les opérations à réaliser :

1. Ajouter d'élément : Implémentez une fonction *Append(data int)* qui ajoute un élément à la fin de la liste. Si la liste est vide, il faut initialiser la tête de la liste avec ce premier *nœud*.
2. Afficher tous les éléments : Implémentez une fonction *Print()* qui parcourt la liste et affiche les données de chaque *nœud*.
3. Supprimer un élément (par valeur) : Implémentez une fonction *Delete(data int)* qui supprime le premier *nœud* qui contient cette donnée. Si l'élément est en tête de liste, ajustez la tête de la liste.
4. Insérer un élément à une position donnée : Implémentez une fonction *InsertAtPosition(data int, position int)* qui insère un élément à une position donnée dans la liste (en commençant par 0 pour le premier élément). Si la position est au-delà de la fin de la liste, l'élément doit être ajouté à la fin.

Pour vous assurez du bon fonctionnement de votre structure, ainsi que de chacune des méthodes, vous réaliserez un petit script de test utilisant l'ensemble des méthodes précédentes.

4.3 Questions facultatives

1. Quelle est la différence entre une liste chaînée simple et une liste chaînée double ?
2. Que se passe-t-il si vous essayez de supprimer un élément qui n'existe pas dans la liste ?
3. Quelle est la complexité en temps des différentes opérations (ajout, suppression, recherche) sur une liste chaînée ?

5 Regex

Pour cet exercice, nous utiliserons le package *regexp*. Pour avoir des informations sur ce package, il est conseillé d'utiliser la commande : *go doc regexp*

5.1 Numéro de téléphone

Écrire une fonction qui vérifie qu'un numéro soit au bon format :

- commençant par 0 ou +33 puis suivi de 9 chiffres,
- pouvant être séparés par des espaces, points ou tirets tous non consécutifs

5.2 Adresse mail

Écrire une fonction qui vérifie qu'une adresse mail soit au bon format :

- un seul "@"
- finis par .fr, .org ou .com (pour faciliter)
- caractère autorisé seulement : lettres, chiffres, points non-consécutifs ni en première ou dernière postion et "„

6 Les livres

1. Définir une structure *Livre* contenant des champs tels que *ID* (*entier*), *Titre* (string), Auteur (string) et *Description* (*string*).
2. Instanciez un livre puis affichez les détails de cette instance
3. Ajouter une fonction *NouveauLivre* qui prend en paramètre les détails d'un livre *(ID, Titre, Auteur, Description)* et renvoie une nouvelle instance de la structure Livre.
4. Implémenter une fonction *AfficherDetails* qui prend un livre en paramètre et affiche ses détails.
5. Ajouter une nouvelle structure *Bibliotheque* qui contient une liste de Livres. Implémenter des fonctions pour ajouter un livre à la bibliothèque, afficher la liste des livres et rechercher un livre par son ID.

7 Interface de tableau dynamique

Cet exercice vise à implémenter deux types de tableaux dynamiques :

- Un tableau qui double sa taille pour un ajout lorsque la capacité est dépassée.
- Un tableau qui augmente sa capacité d'une unité à chaque fois.

Vous devrez donc écrire les codes suivants :

1. Définir une interface *TableauDynamique* avec les méthodes Ajouter(valeur interface) et *Obtenir(index int) interface.*
2. Définir la structure *TableauDoublement* qui implémentent l'interface *TableauDynamique.* Cette structure double la taille du tableau dans Ajouter chaque fois que la capacité est dépassée.
3. De même, définir *TableauAgrandissementUnitaire* qui implémentent l'interface *TableauDynamique.* Cette structure augmente la capacité du tableau d'une unité à chaque fois que la capacité est dépassée.

8 Nuances de "Hello world !"

1. Écrire un programme qui affiche *Hello world !* dans la console.
2. Modifier le programme précédent pour qu'il prenne un nom en argument de ligne de commande et imprime un message de bonjour personnalisé. Par exemple, *go run bonjour.go -nom John* devrait afficher *"Hello John !"*. Utilisez le package *flag* pour parcourir les arguments en ligne de commande et formatez les chaînes avec *fmt.*
3. Modifier à nouveau le programme pour qu'il demande au utilisateur son nom au lieu de le prendre comme argument. Utilisez le package *bufio* pour lire l'entrée utilisateur depuis l'entrée standard *os.Stdin.*

4. Écrire une fonction qui prend un code linguistique (par exemple, *"fr"* pour français) et renvoie un message d'accueil approprié dans cette langue. Utilisez des tableaux associatifs (*maps*) ou des dictionnaires comme structure de données. Testez soigneusement votre fonction avec diverses entrées, y compris des entrées invalides telles qu'une chaîne vide ou des codes non valides, etc.

5. Créez une autre version du *"Hello world !"* où vous accueillerez les utilisateurs en fonction de l'heure actuelle (matin/après-midi/soir/nuit) ainsi que les informations sur la date. Vous pouvez utiliser *time* en Go pour cela. Par exemple : *"Nous sommes le vendredi 10 avril 2020. Bonne après-midi !"*.