**ALGORITHMES POUR LA RESOLUTION DE PROBLEMES**

**EXERCICE 1**

**Représentation du problème :** Retourner l’index d’une valeur dans une liste triée. Si la valeur existe, sinon retourner -1 et ce le plus rapidement possible.

Par exemple soit la liste [1,2,5,8,16,23,50,128]. Si on souhaite trouver l’index de 23 dans la liste, nous devons avoir 5 comme retour et si on souhaite trouver l’index de 14, nous devons avoir -1.

**Solution :** L’algorithme prends en paramètre une liste de nombres et une valeur. Elle récupère la valeur la plus au milieu de la liste et la compare avec la valeur recherchée. Si elles sont égales, alors on retourne l’indice de ce milieu. Dans le cas contraire, si la valeur recherché est plus grande que celle du milieu la nouvelle liste sera entre le milieu et la fin de la liste sinon elle sera entre le début de la liste et le milieu.

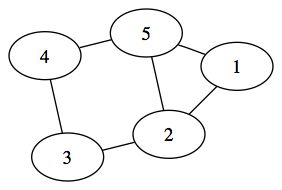
**Résultat** **:** l’indice de l’élément 23 dans cette liste [1,2,5,8,16,23,50,128] est 5 et 14 n’existe pas dans la liste.

**Conclusion :** L’algorithme de recherche binaire divise l’espace de recherche en 2 à chaque itération, assurant une complexité de O (log n).

**Dépôt git :**

**EXERCICE 2**

**Représentation du problème :** Explorer un graphique et trouver le chemin le plus court entre un emplacement E1 et un emplacement E2. Soit le graphe suivant :



Le point de départ étant 1, le chemin le plus cours pour arriver en 4 doit être 1,5,4.

**Solution :** Le BFS parcours un graphe en fonction de la valeur initiale passée en paramètre et retourne une liste ordonnée d’éléments en fonction du parcours en longueur et le DFS parcours le graphe et ressort avec une liste en fonction du parcours en profondeur.

**Résultat** **:** Dans le graphe précèdent avec point de départ 1, Le BFS est 1,5,2,4,3 ; Le BFS est 1,5,4,2,3. Le chemin le plus court est 1,5,4.

**Conclusion :** Les algorithmes de recherche en profondeur (DFS) et de recherche en largeur (BFS) explorent un graphe pour trouver des nœuds, avec BFS garantissant le chemin le plus court dans un graphe non pondéré. En termes de complexité, DFS et BFS ont tous deux une complexité temporelle de O (V + E), où V est le nombre de nœuds et E le nombre d'arêtes.

**Dépôt git :**

**EXERCICE 3**

**Représentation du problème :** Maximiser la valeur totale des éléments à mettre dans le sac à dos sans dépasser la capacité totale de poids. Soit les articles (A1 :{10,4}, A2 :{15,5}, A3 :{40,6}, A4 :{30,3}) , la valeur maximale devra être 30 pour un poids maximal de 6.

**Solution :** L’algorithme de sac à dos crée un tableau dp de dimension (n+1)\*(w+1) avec n le nombre d’éléments que nous avons et w le poids maximal du sac à dos. I rempli le tableau en itérant sur chaque élément et chaque capacité de poids afin d’obtenir la valeur maximale sur la dernière cellule et les éléments ajoutés dans le sac.

**Résultat** **:** Pour les articles (A1 :{10,4}, A2 :{15,5}, A3 :{40,6}, A4 :{30,3}) avec le poids maximal du sac 6, la valeur maximale est 30 et l’article ajouté dans le sac est A4(valeur : 30, poids : 3)

**Conclusion :** L’algorithme du sac à dos utilise la programmation dynamique pour maximiser la valeur totale des articles pouvant être inclus dans le sac à dos, tout en respectant une contrainte de poids. L’algorithme itère à travers chaque article, et pour chaque élément, il parcourt les poids possibles de 0 à w d’où la complexité O (n\*w).

**Dépôt git :**

**EXERCICE 4**

**Représentation du problème :** Fusionner les intervalles qui se chevauchent dans une liste d’intervalles. Soit la série d’intervalles suivant : [1,3] et [2,4] l’intervalle fusionné devra être [1,4].

**Solution :** L’algorithme de fusion des intervalles fusionne les intervalles se chevauchant en commençant par trier les intervalles par leur début. Il parcourt ensuite les intervalles fusionnant ceux qui se chevauchent et ajoutant les non chevauchants à une liste. Et retourne un tableau d’intervalles fusionnés sans chevauchement.

**Résultat** **:** Pour les intervalles [1,3], [2,4],[5,7],[6,8],[9,10] les intervalles fusionnés sont [1,4] et [5,10]

**Conclusion :** L'algorithme de fusion d'intervalles offre une solution efficace pour simplifier des ensembles d'intervalles en éliminant les chevauchements, avec une complexité temporelle de O(nlogn) en raison du tri, suivie d'une itération O(n) pour la fusion. Cela permet d'obtenir une représentation concise et claire des intervalles.

**Dépôt git :**

**EXERCICE 5**

**Représentation du problème :** Trouver le sous tableau contigu avec le maximum. Soit un tableau [-2,-3,-1,-5], le sous tableau contigu devra être [-1] et la somme maximum devra être -1 car la somme devra être réinitialisée dans le cas où elle est négative

**Solution :** L’algorithme de Kadane prends en paramètre un tableau et calcule la somme maximale des éléments le comportant. Pendant le parcours, si la somme devient négative elle est réinitialisée. Le résultat étant le sous tableau ayant le sous tableau contigu avec la somme maximum.

**Résultat** **:** Pour le tableau [-2,-3,-1,-5] le sous tableau contigu est [-1] et la somme maximale est -1 ; Pour le tableau [5,4,-1,7,-8] le sous tableau contigu est [5,4,-1,7] la somme maximale est 15

**Conclusion :** L'algorithme de Kadane permet de trouver efficacement le sous-tableau contigu avec la somme maximale en une seule passe sur le tableau, offrant une complexité temporelle de O(n).

**Dépôt git :**