

Rapport de soutenance 2

Visualizer d'algorithme

Victor Tang B1
Sacha Layani B1
Olivier Bensemhoun B1
Jayson Vanmarcke B1

8 Avril 2024

Sommaire

1	Introduction	3
2	Description du projet	4
2.1	Idée du Projet	4
2.2	Description & Etat de l'art du projet	4
2.3	Outils de Visualisation d'Algorithmes Existant	4
2.4	Choix Technologiques et Architecture du projet	5
2.5	Implémentation des Algorithmes	5
3	Ressenti	6
3.1	Jayson Vanmarcke	6
3.2	Olivier Bensemhoun	6
3.3	Sacha Layani	6
3.4	Victor Tang	6
4	Organisation et répartition des tâches	7
4.1	Organisation du travail et répartition des tâches	7
4.2	Répartition du travail dans le temps	7
5	Explication détaillée des tâches	8
5.1	Algorithme Dijkstra's	8
5.2	Algorithme A* Search	9
5.3	ABR (Arbres Binaire de Recherche)	11
5.3.1	Insertion	11
5.3.2	Suppression	12
5.3.3	Parcours en profondeur	13
5.3.4	Parcours en largeur	13
5.4	BFS,DFS (graph)	14
5.4.1	BFS	14
5.4.2	DFS	15
5.5	Algorithme Floyd-Warshall	16
5.6	Algorithme de Ford-Bellman	17
5.7	Algorithme glouton de recherche Best-First	18
5.8	Algorithme Spanning Trees minimum	19
5.9	Insertion sort	20
5.10	Merge sort	21
5.11	Counting sort	22
5.12	Interface graphique	23
5.12.1	Mise a jour de qualite de vie	25
5.13	Site Internet	26
6	Conclusion	28

1 Introduction

Bonjour et bienvenue dans ce deuxième rapport de soutenance.

Nous allons vous présenter l'avancement de notre travail concernant notre projet qui s'intitule : Visualizer d'algorithme par rapport à la première soutenance.

Le but est de vous exposer clairement et simplement l'évolution de notre projet depuis notre dernière passation avec vous.

Pour rappel, notre visualizer va comparer « visuellement » différents algorithmes classiques tels que BFS, Dijkstra, A* (. . .).

Comme indiqué lors de la première soutenance, nous sommes quatre à constituer ce groupe qui s'intitule « Les Alchimistes ». Nous allons vous présenter ce projet et espérons que vous l'apprécierez ainsi que le travail commun déployé afin d'y parvenir et satisfaire vos attentes.

En ce qui concerne l'avancement du projet, nous sommes dans le délai demandé et sommes fières de ne constater aucun retard comme pour la première soutenance.

2 Description du projet

2.1 Idée du Projet

Pour commencer, nous avons pour première idée commune de réaliser un projet sur les blockchains. Le choix de ce thème fut très rapide. Toutefois, après concertation, cette idée a engendré de nombreux débats au sein du groupe car à l'unanimité, nous sommes arrivés au constat que les blockchains étaient un sujet très vaste voir trop vaste pour deux d'entre nous. Mais, nous décidons de garder cette idée qui fera partie des deux retenues pour les présenter à notre professeur.

Dans un second temps, nous avons cherché et étudié d'autres thèmes pour finalement s'être entendu sur un sujet particulièrement intéressant : le visualiser d'algorithmes. Lors de nos recherches, nous avons été attirés par un site qui mettait en relief le processus du vizualizer d'algorithmes et voulions d'un commun accord le présenter comme second sujet.

Finalement, notre professeur Mr Boulay, a donné son accord pour le vizualizer d'algorithmes qui avait captivé tous les membres de notre groupe lors de sa découverte durant la phase de recherches.

2.2 Description & Etat de l'art du projet

Dans notre groupe, nous réalisons presque chaque jour des projets informatiques, qu'ils soient personnels ou scolaires. Le problème est que nous appliquons soit des concepts vus en cours, soit des notions découvertes grâce à nos propres recherches. En principe, c'est une approche correcte, mais il manque un élément essentiel : un visualizer d'algorithmes. En effet, ce dernier serait utile pour comprendre parfaitement le fonctionnement de la théorie algorithmique.

Ainsi, l'objectif de ce projet sera de créer un visualizer d'algorithmes permettant d'observer l'exécution des algorithmes de base.

2.3 Outils de Visualisation d'Algorithmes Existant

Actuellement, plusieurs outils de visualisation d'algorithmes sont déjà disponibles, se déclinant sous diverses formes :

- Les IDE de JetBrains, tels que PyCharm pour Python, intègrent des fonctionnalités de débogage avancées. Ils permettent de suivre l'exécution du code étape par étape, d'inspecter les variables, et de détecter les erreurs efficacement.
- Les sites web, comme Python Tutor, offrent une visualisation en ligne des programmes Python, Java, JavaScript, C, ou encore le C++. Ces sites permettent de suivre l'exécution du code Python étape par étape.
- D'autres sites, tels que Pathfinding-Visualizer, permettent, comme son nom l'indique, de visualiser des algorithmes de recherche de chemins, comme l'algorithme A*. Ils offrent aux utilisateurs la possibilité de créer des grilles, d'ajouter des obstacles, et de visualiser comment l'algorithme trouve le chemin optimal.

2.4 Choix Technologiques et Architecture du projet

Afin de réaliser ce projet, nous avons besoin d'utiliser le langage de programmation Rust, qui est un langage de programmation novateur reprenant certains points C vus au S3 ou encore du C# vus au S2 et même du Ocaml abordé en cours lors du S1.

Pour la réalisation de ce projet, nous avons tout d'abord décidé qu'il fallait trouver une configuration qui soit valide aussi bien aujourd'hui que demain. Ainsi, lorsque nous établirons des liens entre les algorithmes et l'interface graphique, cela sera moins compliqué que s'il y avait plusieurs fichiers cargo.toml.

2.5 Implémentation des Algorithmes

Dans le cadre de ce projet, nous avons décidé d'implémenter des algorithmes relevant du domaine du *Path Finding, Recherche*, englobant des opérations de recherche, d'insertion et de suppression. La liste des algorithmes que nous avons choisi d'implémenter comprend :

- Algorithmes de Dijkstra
- Algorithme A* Search
- BFS (*Breadth-First Search*), DFS (*Depth-First Search*), insertion et suppression (arbre binaire)
- BFS, DFS (graph)
- Algorithme Floyd-Warshall
- Algorithme glouton de recherche Best-First
- Algorithme de Ford-Bellman
- Algorithme des arbres couvrants minimum (*Minimum Spanning Trees*)
- Insertion sort
- Merge sort
- Counting sort

Cette sélection d'algorithmes couvre une variété de domaines et de fonctionnalités, contribuant ainsi à la diversité et à la richesse de notre projet.

3 Ressenti

3.1 Jayson Vanmarcke

Depuis la première soutenance, je suis heureux de constater que le projet est de plus en plus performant. Nous avons retenu ce qui a été dit par le jury et l'avons mis en pratique pour une amélioration optimale de notre outil. Je constate que je suis toujours aussi passionné par le Rust, qui est un langage fascinant mais aussi facile à prendre en main. De plus, la création du site me conforte et m'encourage dans ma capacité à chercher des idées, créer et découvrir avec satisfaction le résultat obtenu. Par ailleurs, notre cohésion de groupe est très importante car elle est optimale, permettant ainsi de produire un projet de qualité en n'hésitant pas à remettre certaines de nos idées ou créations en question. Cette cohésion nous pousse sans cesse à donner le meilleur de nous même. Hâte de poursuivre ce projet qui me passionne, attise ma curiosité et qui évolue de jours en jours.

3.2 Olivier Bensemhoun

J'ai trouvé la réalisation de cette seconde partie particulièrement stimulante. Cette partie m'a permis de réfléchir et de comprendre comment fonctionnent ces différents algorithmes. De plus j'ai passer du temps a reflechir a un system de clock.

3.3 Sacha Layani

La réalisation d'un outil permettant de visualiser des algorithmes est un projet qui me tenait à cœur depuis un certain temps. J'avais peur de ne pas y arriver seul en raison d'un manque de connaissances ou d'une charge de travail trop importante. Cependant, lors du quatrième semestre, j'ai décidé de relever le défi en m'associant avec mon groupe de projet, les Alchimistes. Aujourd'hui, je suis fier de présenter notre travail lors de cette deuxième soutenance. Depuis notre première présentation, nous avons travaillé dur pour améliorer notre outil et le rendre plus performant. Nous avons également pris en compte les retours de notre jury et avons apporté les modifications nécessaires pour répondre à leurs attentes. Je tiens à remercier mes camarades pour leur investissement et leur soutien tout au long de ce projet. Nous avons su nous entraider et nous motiver pour avancer ensemble. Je suis convaincu que c'est grâce à cette cohésion que nous avons pu réaliser un outil de qualité. Je suis impatient de présenter notre travail final et j'espère que notre outil sera utile à d'autres personnes souhaitant visualiser des algorithmes de manière simple et efficace.

3.4 Victor Tang

Je trouve que le sujet est très passionnant. J'ai été responsable plusieurs fois pour le visuel d'une application. Faire l'application de A à Z est un vrai défi qui me donne envie de bien faire même si la documentation de GTK est faible et contient peu d'exemples. Pour ce qui est de la seconde soutenance, la difficulté majeure était de faire en sorte que l'affichage fonctionne correctement. Je n'ai pas de donnée chiffrée exacte mais j'ai probablement passé plus de 30 heures devant le morceau de code servant à afficher les images. Cependant, à force de travailler avec gtk, je me suis habitué et je peux affirmer aujourd'hui que je prends particulièrement du plaisir à implementer des fonctions.

4 Organisation et répartition des tâches

4.1 Organisation du travail et répartition des tâches

Tâches	Victor Tang	Jayson Van-marcke	Olivier Ben-shemoun	Sacha Layani
Algorithmes Dijkstra's		Responsable		
Algorithmes A* Search				Responsable
BFS,DFS, insertion et suppression (arbre binaire)				Responsable
BFS,DFS (graph)		Responsable		
Algorithme Floyd-Warshall				Responsable
Algorithme glouton de recherche Best-First			Responsable	
Algorithme de Ford-Bellman			Responsable	
Algorithme Spanning Trees minimum			Responsable	
Insertion sort	Responsable			
Merge sort			Responsable	
Counting sort		Responsable		
Interface graphique	Responsable			
Site Web		Responsable		Responsable

4.2 Répartition du travail dans le temps

Légende : += Ebauche, ++= Avancé, +++= Terminés

Tâches	Soutenance 1	Soutenance 2	Soutenance 3
Algorithmes Dijkstra's	+	++	+++
Algorithmes A* Search	+	++	+++
BFS,DFS, insertion et suppression (arbre binaire)	++	+++	+++
BFS,DFS (graph)	++	++	+++
Algorithme Floyd-Warshall	+	++	+++
Algorithme glouton de recherche Best-First	+	++	+++
Algorithme de Ford-Bellman	+	++	+++
Algorithme Spanning Trees minimum	+	++	+++
Insertion sort	++	+++	+++
Merge sort	++	+++	+++
Counting sort	+	++	+++
Interface graphique	+	++	+++
Site Web		++	+++

5 Explication détaillée des tâches

5.1 Algorithme Dijkstra's

L'algorithme de Dijkstra est un algorithme de recherche de chemin le plus court dans un graphe, c'est-à-dire un réseau de nœuds connectés par des arêtes ayant des poids ou des coûts associés. L'algorithme détermine le chemin le plus court entre un nœud de départ et tous les autres nœuds du graphe.

Cet algorithme est aussi surnommé algorithme glouton ou vorace car il utilise une approche gloutonne pour trouver le chemin optimal. Initialement, il attribue une distance infinie à tous les nœuds sauf au nœud de départ qui a une distance de 0. Puis, l'algorithme sélectionne le nœud non visité avec la distance la plus faible, appelé le « nœud courant » à chaque répétition. Il met à jour les distances des nœuds voisins en les comparant à la distance actuelle du nœud courant, plus le poids de l'arête les reliant. Si la nouvelle distance est plus petite, elle est mise à jour. L'algorithme répète ce processus jusqu'à ce que tous les nœuds aient été visités ou que la distance minimale vers le nœud d'arrivée soit trouvée.

L'algorithme de Dijkstra assure de trouver le chemin le plus court dans un graphe sans arêtes de poids négatif, mais peut ne fonctionner correctement si de tels poids sont présents. Son utilisation est très répandue dans les applications de routage, dans les réseaux de télécommunications, la planification de trajets dans les systèmes de transports et d'autres domaines où la recherche de chemins optimaux est incontournable.

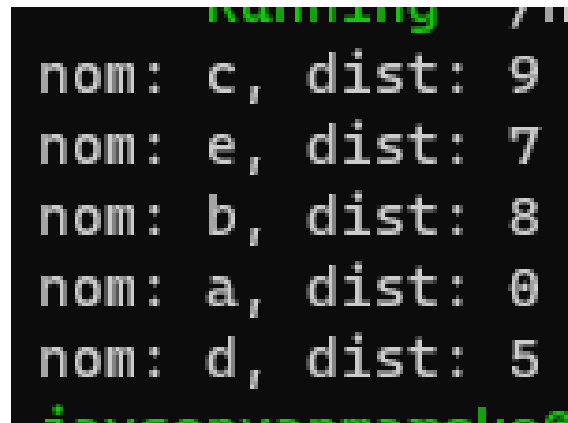
Afin de coder cet algorithme, il me fallait plusieurs informations. Tout d'abord, débutant le Rust, je n'ai aucune notion pour traiter et implémenter des graphs. Après de longues recherches et de compréhension, j'ai regardé le mimo (vidéo) sur l'algorithme de Dijkstra afin de comprendre son fonctionnement puis, j'ai étudié la fiche synthèse et la procédure donnée pour m'aider à coder cet algorithme. Une fois tous ces éléments en ma possession, je pouvais enfin coder mon algorithme. Ensuite, vient le codage de la fonction. Pour le codage, je me suis appuyé sur le mimo d'algorithmique S4 sur Dijkstra notamment du pseudo code pour écrire ma fonction. Cela m'a beaucoup aidé pour compléter la fonction.

L'initialisation de la fonction n'est pas très complexe : tout d'abord, elle prend en paramètre une source à partir d'un sommet. Ceci est mon point de départ pour trouver le plus court chemin. Ensuite, je mets en paramètre la liste d'adjacence du graph puis un vecteur comprenant les sommets.

Je commence à remplir la pile binaire avec les sommets qui ont la plus petite distance qui sont placés en premier. Puis, nous visitons les sommets qui ont les plus petites distances en premier. Par conséquent, comme nous l'avons mentionné dans le cahier des charges, l'algorithme Dijkstra se rapproche fortement de l'algorithme glouton (greedy en anglais). Pour réaliser cela, nous utilisons les méthodes pop, get et set, permettant de savoir quelles sont les plus petites distances ainsi que les coûts. Dès lors, on remarque que lorsque l'on modifie la distance d'un sommet, la pile binaire ne change pas, ce qui est très contrariant. Par conséquent, je décide de créer une nouvelle pile avec le bon ordre.

Après avoir réalisé la fonction, elle est accompagnée d'une fonction main permettant de tester le code et déduire le chemin le plus court en fonction des coûts pour accéder à cette distance.

A la fin du parcours, mon programme me renvoie les différents noms de sommets ainsi que sa distance par rapport à la racine, facilement identifiable puisque sa distance de lui à lui-même est de 0. Nous retrouvons ainsi le plus court chemin de la source (distance à 0) et du nœud correspondant.



```
nom: c, dist: 9
nom: e, dist: 7
nom: b, dist: 8
nom: a, dist: 0
nom: d, dist: 5
```

Concernant l'évolution de cette tâche, il n'y a pas de retard. Nous avons un algorithme fonctionnel et démontré sur le terminal lors de notre deuxième soutenance. Il ne reste plus qu'à la rendre fonctionnelle sur l'interface graphique pour l'ultime soutenance.

5.2 Algorithme A* Search

L'algorithme A* est utilisé pour trouver un chemin entre un nœud initial et un nœud final. Il repose sur une évaluation approximative de chaque nœud afin d'estimer le chemin optimal, puis explore les nœuds en suivant cette estimation. Cet algorithme est simple, ne nécessite pas de préparation préalable et limite l'utilisation de la mémoire.

L'algorithme A* est considéré comme le plus rapide pour trouver le chemin le plus court, surpassant l'algorithme de Dijkstra. En d'autres termes, intégrer cet algorithme dans notre projet serait une excellente manière de comparer le temps de traitement et le raisonnement entre l'algorithme A* et l'algorithme de Dijkstra.

Étant donné que l'algorithme de recherche A* n'était pas abordé au cours de ma scolarité, j'ai dû me renseigner sur son fonctionnement. J'ai commencé par explorer plusieurs sites internet universitaires pour comprendre la théorie derrière cet algorithme. Malheureusement, la plupart de ces sites semblaient être conçus pour des étudiants déjà familiarisés avec le sujet, ce qui m'a laissé dans le flou puisque je n'avais jamais été exposé à ce concept auparavant.

J'ai alors eu l'idée de me tourner vers des vidéos explicatives, qui ont souvent l'avantage d'être plus claires et détaillées. Cette démarche s'est avérée fructueuse, car une vidéo en particulier, que j'ai dû visionner à plusieurs reprises, m'a permis de comprendre le fonctionnement de cet algorithme.

Tout d'abord, on commence par initialiser deux listes. La première est la liste des sommets à visiter (`open_liste`), et la deuxième est la liste des sommets déjà visités (`closed_list`). Ensuite, nous entamons le parcours du graphe. À chaque sommet, on calcule un "score" avec la formule suivante : $\text{score} = \text{coût déjà parcouru jusqu'à cet endroit} + \text{estimation du coût restant jusqu'à la destination}$. Une fois que les scores ont été calculés, on choisit le chemin le plus optimiste, celui ayant le coût le plus bas. Ainsi, notre algorithme continue à calculer et sélectionner les endroits jusqu'à ce qu'il atteigne sa destination ou qu'il n'y ait plus d'endroits à visiter.

En résumé, l'algorithme A* utilise une combinaison de la distance déjà parcourue et d'une estimation de la distance restante pour guider ses choix et trouver le meilleur chemin vers sa destination.

Voici un exemple :



Ce schéma représente un graphe. À présent, l'objectif est de le convertir en une liste d'adjacence.

Pour cela, nous utilisons la méthode suivante :

```
fn main() {  
    let _g: Graph = Graph::new( key: 0, adjlists: vec![vec![0, 1, 1, 0, 0],  
                                                    vec![1, 0, 1, 1, 0],  
                                                    vec![0, 0, 0, 1, 1],  
                                                    vec![0, 1, 0, 0, 1],  
                                                    vec![0, 0, 0, 1, 0]], order: 5);  
  
    let start: i32 = 0;  
    let end: i32 = 4;  
    let result: Vec<i32> = a_algorithm(_g, start, end);  
    println!("The shortest path from {} to {} is : {:?}", start, end, result);  
}
```

Voici le contenu de la fonction main(), qui est le point d'entrée de notre programme et qui exécutera le code avec l'exemple graphique.

En observant, nous constatons que le graphe a été converti en une liste d'adjacence.

The shortest path from 0 to 4 is : [0, 2, 4]

Le résultat que j'obtiens est bien le chemin le plus court, c'est à dire : 0 -> 2 -> 4.

Concernant l'avancement de cette partie, il n'y a aucun retard et sera implémentée pour l'ultime soutenance.

5.3 ABR (Arbres Binaire de Recherche)

Les arbres binaires de recherche sont une structure algorithmique étudiée au cours du semestre 2 à l'EPITA.

Ce qui a été le plus complexe n'a pas été les algorithmes en eux-mêmes, mais plutôt leur implémentation.

En effet, j'ai souhaité implémenter la même structure que celle étudiée en cours, à savoir :

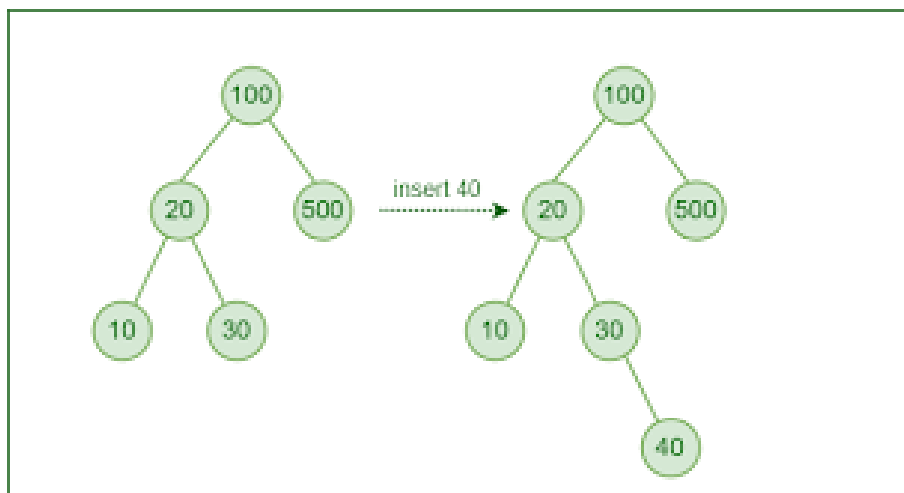
- Une clé (key) pour chaque nœud
- Un tableau contenant les sous-arbres gauches (left)
- Un tableau contenant les sous-arbres droits (right)

Il est important de noter que Btree.left & Btree.right correspondent aux sous-arbres de la clé visitée.

5.3.1 Insertion

L'objectif de l'insertion est d'ajouter un nœud à l'arbre binaire de recherche. Pour ce faire, on parcourt l'arbre de la manière suivante :

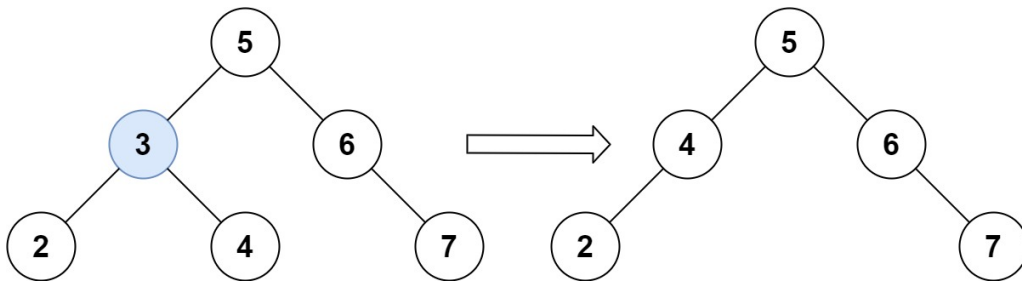
- Si le nœud que l'on souhaite insérer est inférieur ou égal au nœud actuel, on se dirige vers la partie gauche de l'arbre.
- Sinon, on se dirige vers la droite.
- Si l'un des sous-arbres que l'on souhaite accéder est vide, cela signifie que c'est à cet emplacement que l'on doit insérer le nouveau nœud.



5.3.2 Suppression

L'objectif de la suppression est de supprimer un nœud dans un arbre binaire de recherche. La différence entre l'insertion et la suppression réside dans le cas de l'élément que l'on souhaite supprimer :

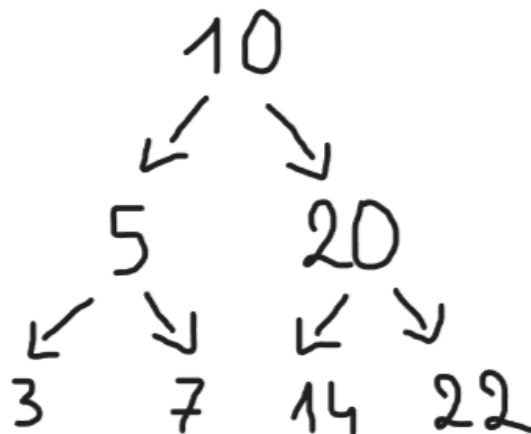
- Si c'est une feuille : on applique le même principe que pour l'insertion dans le sens de la recherche du nœud.
- Si ce n'est pas une feuille : on recherche le nœud. Une fois trouvé et lorsqu'on constate que ce n'est pas une feuille, on cherche le maximum du sous-arbre gauche et on l'échange avec l'élément que l'on souhaite supprimer. Ensuite, on le supprime comme une feuille.



Voici un exemple d'exécution :

```
Insert (node : 5) : 10 5
Insert (node : 20) : 10 5 20
Insert (node : 3) : 10 5 3 20
Insert (node : 7) : 10 5 3 7 20
Insert (node : 14) : 10 5 3 7 20 14
Insert (node : 22) : 10 5 3 7 20 14 22
BFS : 10 5 20 3 7 14 22
Delete (node : 7) : 10 5 3 20 14 22
Delete (node : 20) : 10 5 3 14 22
BFS : 10 5 14 3 22
```

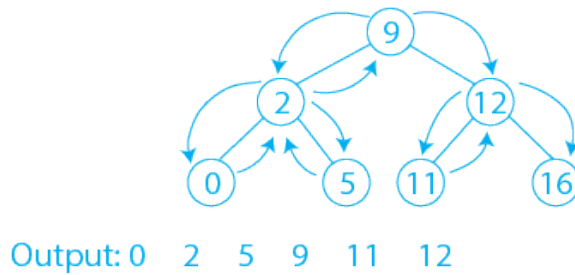
On constate bien qu'en suivant un parcours en profondeur (préfixe) du côté gauche principal (après l'insertion du nœud 22), l'arbre résultant est le suivant :



5.3.3 Parcours en profondeur

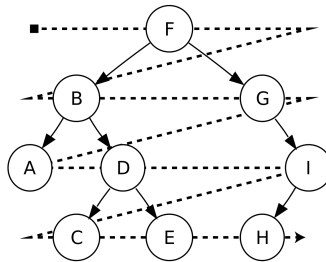
Pour afficher l'arbre après une insertion ou une suppression, nous avons actuellement la possibilité d'utiliser un parcours en largeur ou en profondeur.

Le parcours en profondeur consiste à lire chaque nœud en suivant une trajectoire du plus haut vers le plus bas, de gauche à droite. En d'autres termes, chaque nœud est noté dès sa première visite. Ce que je vous ai décrit correspond à ce que l'on appelle un parcours en profondeur de type préfixe. Il existe également d'autres types de parcours en profondeur : suffixe et infixe. Toutefois, dans un souci de simplicité pour l'utilisateur, nous souhaitons mettre en place uniquement la version préfixe.



5.3.4 Parcours en largeur

Le parcours en largeur est une méthode qui consiste à explorer chaque niveau de l'arbre de manière progressive, de gauche à droite. Pour mieux comprendre cette approche, imaginez-vous lire un texte en français : vous le parcourez de gauche à droite, ligne par ligne. De la même manière, lors du parcours en largeur, nous notons chaque nœud de gauche à droite, niveau par niveau, comme si nous parcourions l'arbre de manière horizontale et ordonnée.



Pour cette deuxième soutenance, nous avons implémenté les algorithmes sur les arbres binaires de recherche dans notre interface graphique. Nous avons ainsi pu réaliser notre objectif initial, à savoir visualiser étape par étape l'exécution de ces algorithmes. Grâce à cette nouvelle fonctionnalité, il est maintenant possible de suivre l'exécution des algorithmes de manière détaillée, ce qui facilite la compréhension de leur fonctionnement. Nous avons également veillé à ce que l'interface graphique soit intuitive et facile à utiliser, afin que les utilisateurs puissent se concentrer sur l'essentiel : la visualisation des algorithmes.

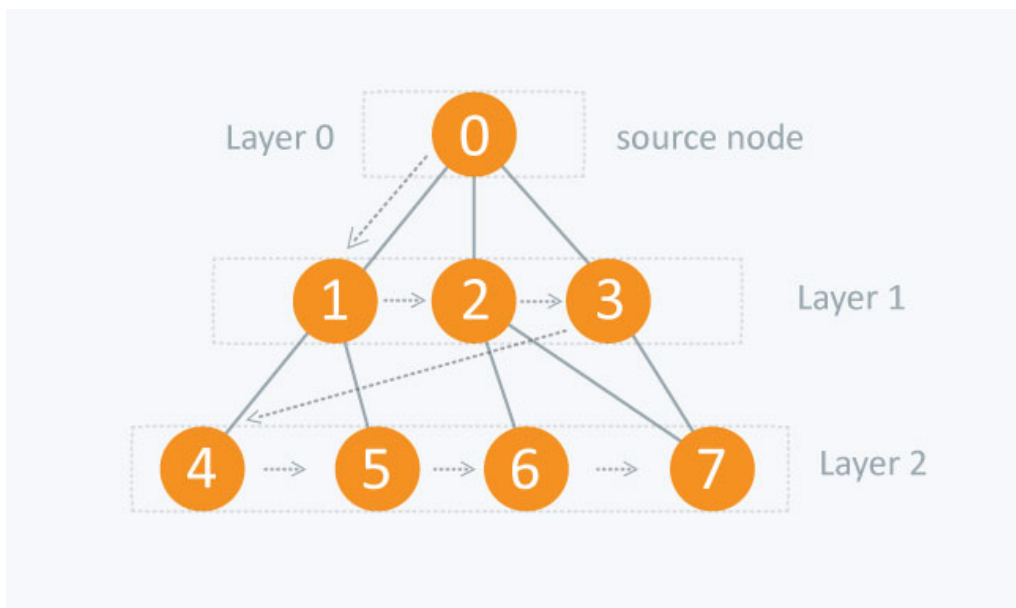
5.4 BFS,DFS (graph)

5.4.1 BFS

Le BFS est utilisé pour explorer de manière rapide, un graphe dans la largeur, niveau par niveau. Chaque itération augmentera la distance par rapport au nœud de départ. Son objectif est de déterminer rapidement s'il existe un chemin entre deux sommets. Il peut aussi définir le nombre de niveaux séparant deux sommets. Cet algorithme va ainsi explorer, niveau par niveau dans tous les chemins simultanément et non se concentrer chemin par chemin comme le DFS. Son fonctionnement se base sur une structure de données nommée « la queue ».

Afin de faire le parcours BFS, il fallait au préalable, se renseigner sur les méthodes disponibles. Fort heureusement, nous pouvons utiliser les queues comme en python grâce à la méthode suivante : `VecDeque` issue de la bibliothèque `std : collections`. Ensuite, il faut initialiser un graph pour utiliser le parcours. Le graph est une structure, comprenant des sommets et des arrêtes. J'ai utilisé une liste d'adjacence. Puis, j'ai une fonction `addeges` me permettant de tester ma fonction. Vient le codage du BFS.

Cette fonction prend deux paramètres : un graph et une source qui est la racine de notre graph. Comme en python, nous commençons par initialiser une queue et un vecteur. Dès lors, nous ajoutons dans notre queue la racine. A partir du moment où la queue n'est pas vide, on ajoute le sommet dans le vec final et on parcourt le graph en regardant si il possède des fils afin de les insérer (de gauche à droite) dans la queue. Et finalement, on reboucle jusqu'à parcourir le graph en entier. Dans l'ultime étape, nous retournons la liste de vecteurs, correspondant aux sommets parcourus en largeur.

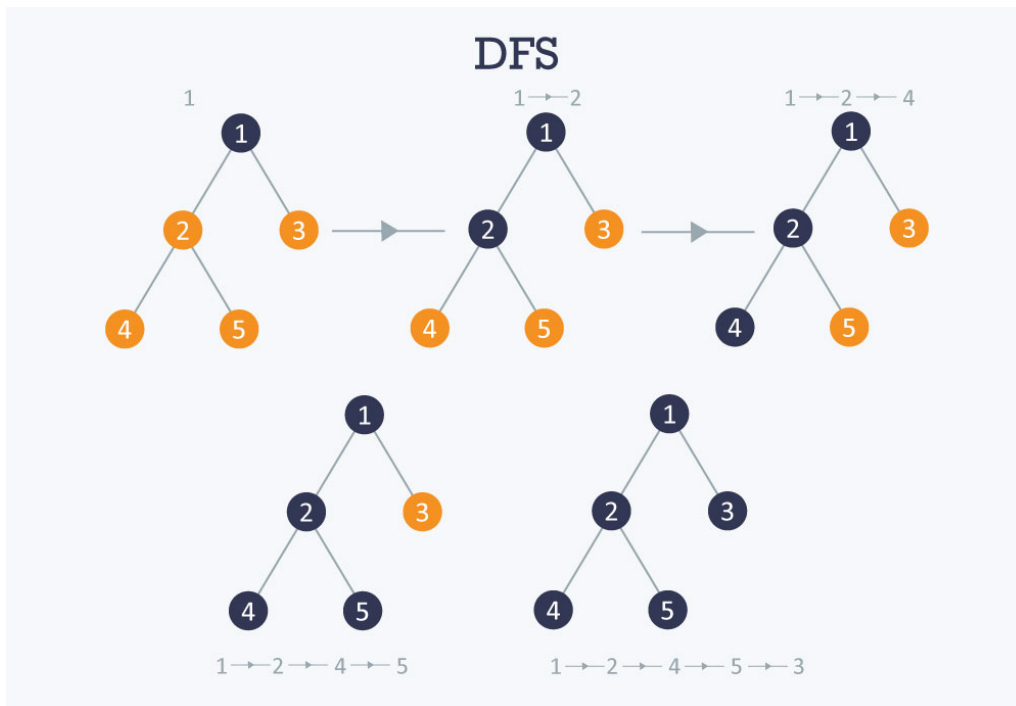


5.4.2 DFS

Le principe du DFS est d'établir du retour sur trace (backtracking) ou de faire des recherches complètes. Pour chaque chemin, il va explorer méticuleusement le chemin actuel. Puis, il va passer au chemin suivant pour être sûr d'avoir exploré toutes les possibilités de chaque chemin. Son fonctionnement se base sur une structure de données : la stack.

Afin de faire le parcours DFS, nous avons du nous renseigner sur les méthodes disponibles. Comme ce dernier n'a pas besoin de queue, son implémentation est plus simple. Il a fallu initialiser un graph pour utiliser le parcours. Le graph est une structure, comprenant des sommets et des arrêtes. J'ai utilisé une liste d'adjacence. Ensuite, j'ai une fonction addeges me permettant de tester ma fonction. Vient le codage du DFS.

Le DFS prend en paramètre un graph, une source et un Hashset permettant d'ajouter les sommets. Comme en python, on commence par insérer la source dans le Hashset. Puis, on initialise un vec contenant notre sommet source. Par la suite, nous regardons les fils possédés par notre racine source et dans le cas ou le Hashset ne contient pas ses fils, nous l'ajoutons dans le vec via une récursivité du fils et du Hashset. Pour finir, je retourne le vec, contenant tous les sommets du graph, parcouru en profondeur.



Concernant l'évolution de cette tâche, il n'y a pas de retard et lors de l'ultime soutenance, ces 2 algorithmes seront présents dans l'interface graphique.

5.5 Algorithme Floyd-Warshall

L'algorithme de Floyd-Warshall est un algorithme que nous avons étudié dans le cadre du cours d'algorithmes du semestre 4, portant sur les graphes.

Cet algorithme est utilisé pour déterminer les distances des plus courts chemins entre tous les sommets d'un graphe orienté."

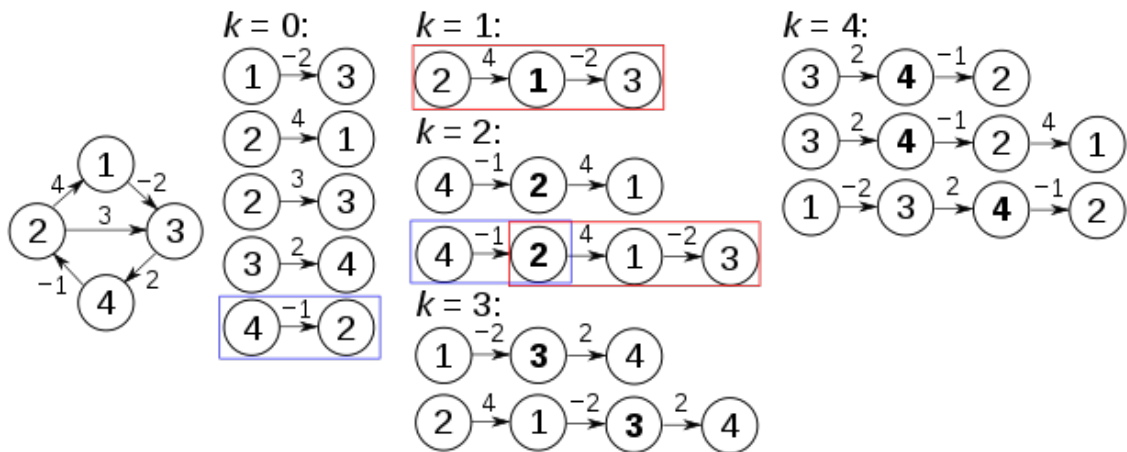
Maintenant, je vais vous expliquer brièvement le fonctionnement de l'algorithme de Floyd-Warshall :

L'algorithme de Floyd-Warshall est utilisé pour trouver les chemins les plus courts entre tous les paires de sommets dans un graphe pondéré, même s'il contient des arêtes de poids négatif, mais sans cycle de poids négatif.

Son fonctionnement est basé sur la notion de programmation dynamique. Il maintient une matrice des distances minimales entre chaque paire de sommets. Initialement, cette matrice est remplie avec les poids des arêtes du graphe. Ensuite, l'algorithme explore toutes les paires de sommets (i, j) et vérifie si passer par un sommet intermédiaire k réduit la distance entre i et j . Si c'est le cas, la distance minimale entre i et j est mise à jour en utilisant la distance minimale entre i et k ajoutée à la distance minimale entre k et j . Cette mise à jour est répétée jusqu'à ce que toutes les paires de sommets aient été explorées et que la matrice des distances minimales soit stabilisée.

En fin de compte, la matrice résultante contient les distances les plus courtes entre tous les paires de sommets du graphe.

Aucun retard n'est à déployé sur cette partie.



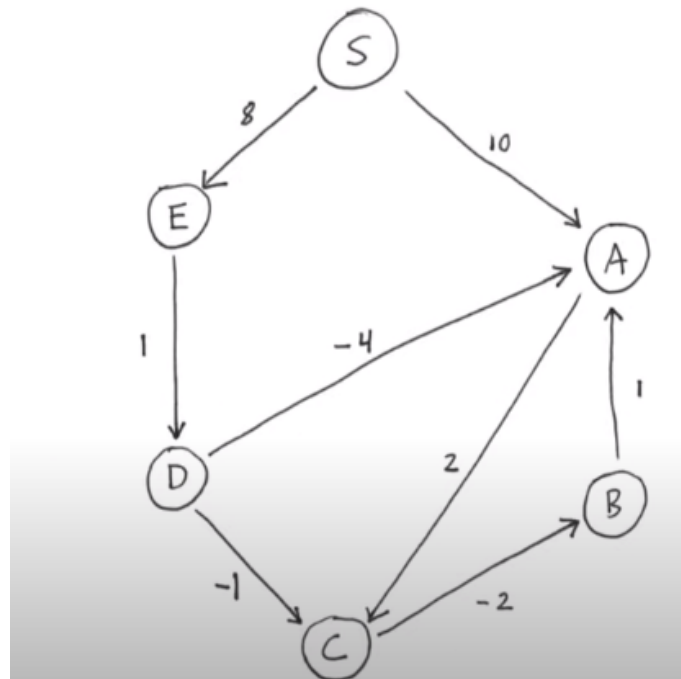
5.6 Algorithme de Ford-Bellman

L'algorithme de Bellman-Ford est utilisé pour calculer les chemins de coût minimal dans les graphes. Il est nommé d'après Richard E. Bellman et Lester Ford, Jr. L'algorithme fonctionne en initialisant le coût de tous les sommets à l'infini, à l'exception du sommet source, qui est initialisé à 0.

Ensuite, il effectue $V-1$ itérations, où V est le nombre de sommets dans le graph. Dans chaque itération, l'algorithme examine chaque arête (u, v) et met à jour le coût du sommet v si le coût du chemin de u à v via cette arête est inférieur au coût actuel du chemin de u à v . Si une mise à jour se produit pendant la dernière itération, cela indique qu'il existe un chemin négatif dans le graph, ce qui est impossible dans un graph sans cycle négatif.

new Pour l'algorithme de Ford-Belleman, j'ai passé du temps pour cree un system de clock. Ceci m'a permis de realiser un system de clock pour de l'algorithme et d'en réaliser une seconde version. Pour le moment, mon programme prend le graph et un histogramme pour les coûts. Il rajoute les coûts dans le second graph pour simuler son utilisation.

Nous allons le réaliser pour tous les arcs en faisant un parcours DFS qui a pour objectif de parcourir le graph. J'ai réalisé des modifications dans l'algorithme pour l'optimiser en arrêtant le code si aucune valeur dans l'histogramme n'a été changée ce qui permet de faire moins $V-1$ parcours. J'ai ensuite testé mon code avec le graph que vous pouvez retrouver ci-dessous.

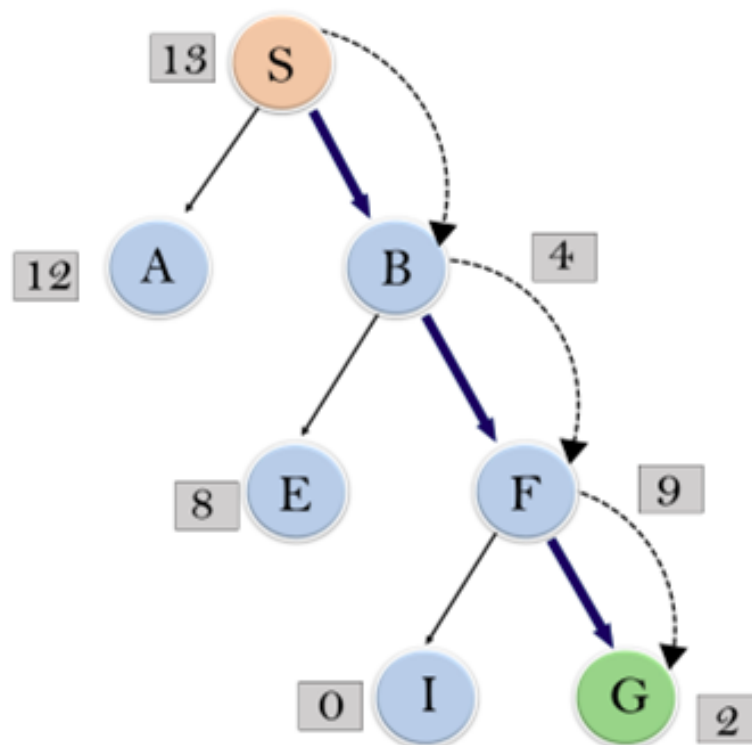


Pour conclure, j'ai bien avancé sur ma partie. Par conséquent, il n'y a aucun retard. Les différentes recherches que j'ai utilisées m'ont permis de réaliser une première version qui permet de suivre avec un peu d'avance le cahier des charges. De plus, comme vous avez pu remarquer dans le descriptif de cet algorithme, j'ai opté pour le choix de le modifier et de faire des optimisations.

5.7 Algorithme glouton de recherche Best-First

Pour l'algorithme glouton de recherche Best-First, j'ai passé du temps à comprendre et réfléchir à un système de clock. Cependant, pour ce sujet, j'ai rencontré plus de difficultés à trouver des exemples car l'algorithme de recherche best first est majoritairement utilisé dans d'autres cas tel que le tri.

Ces recherches m'ont permis de comprendre le fonctionnement de l'algorithme et d'en réaliser une seconde version très simplifiée au niveau de son fonctionnement. Pour le réaliser, j'ai calculé la distance optimale entre le point a et b et j'ai essayé d'établir la ligne droite. En cas de mur, le programme suit les formes dans les différents sens pour qu'il puisse contourner l'obstacle et atteindre b.



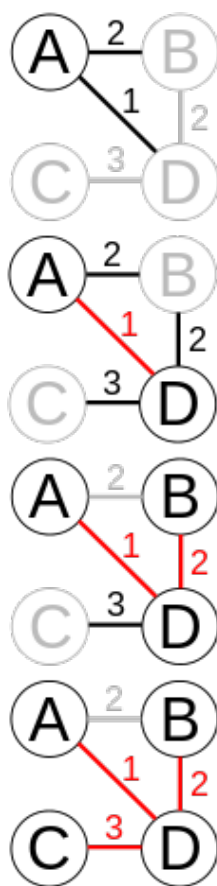
Pour conclure, j'ai bien avancé sur ma partie et les différentes recherches que j'ai utilisées m'ont permis de réaliser une première version qui permet de suivre le cahier des charges. Il n'y a donc aucun retard.

5.8 Algorithme Spanning Trees minimum

Pour l'algorithme de prim, j'ai réalisé de nombreuses recherches pour me permettre de comprendre le fonctionnement de cet algorithme. Suite aux différentes vidéos que j'ai visionnées, j'ai pu vous établir cette description. L'algorithme commence sur une arête du graph et l'inclut dans l'arbre sous- arbre que j'ai créé.

Puis, à chaque étape, il ajoute une arête supplémentaire au minimum de poids qui relie un sommet de l'arbre à un sommet non inclus dans ce sous graph. Le processus se poursuit jusqu'à ce que tous les sommets soient inclus dans l'arbre. Après avoir parcouru de nombreux sites, j'ai effectué une seconde version du programme qui a pour objectif de vérifier ma compréhension de cet algorithme et de passer du temps pour comprendre comment transformer l'algorithme pour qu'il soit fonctionnel en exécutant le code en plusieurs fois pour nous permettre d'avoir une animation.

Pour cette nouvelle version du programme, J'ai réalisé un système de clock. Ce système utilise un ensemble de fonctions permettant d'appeler une fonction réalisant une itération par une itération.

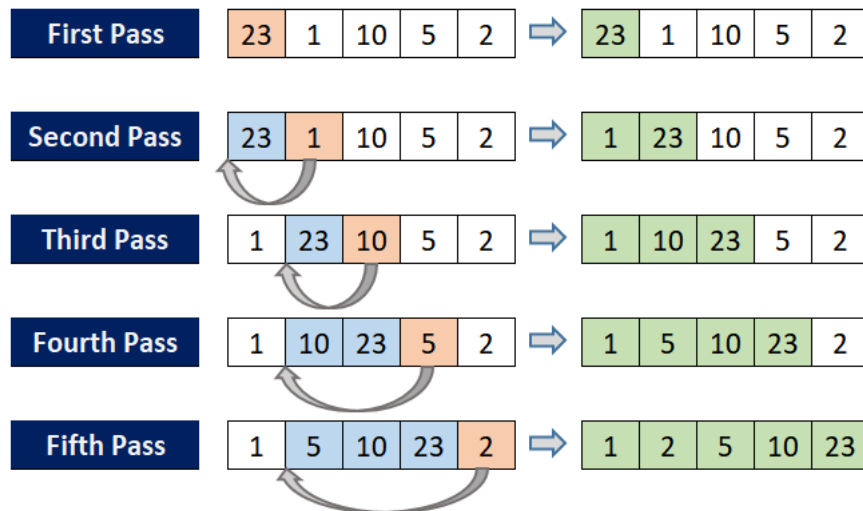


Pour conclure, j'ai réalisé la partie qui m'a été demandé dans le cahier des charges et j'ai passé du temps à comprendre le fonctionnement de cet algorithme. Il n'y a donc aucun retard.

5.9 Insertion sort

Le tri par insertion est l'un des premiers algorithmes de tri vu en classe. C'est l'un des tris le plus basique et souvent on le fait sans s'en rendre compte. Il est utilisé pour trier les tableaux et est efficace pour trier les tableaux de petite taille ou des tableaux presque triés. Le meilleur cas a une complexité de $O(n)$, sa complexité moyenne est de $O(n^2)$ et le pire cas est de $O(n^2)$, avec le tableau trié à l'envers.

Son principe est très simple. On parcourt tout le tableau, et durant ce parcours, si on tombe sur une valeur qui est mal placée dans le tableau, on décale cette valeur à gauche jusqu'à ce que le tableau soit trié entre la première valeur et là où était la valeur.

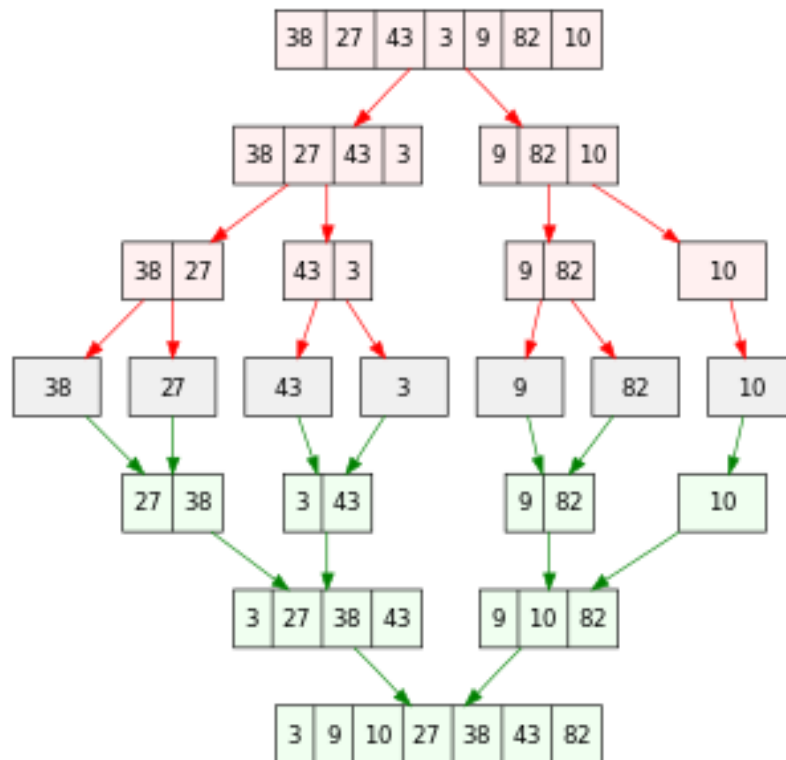


Cet algorithme est terminée et est déjà implémenté dans l'interface.

5.10 Merge sort

Pour l'algorithme du merge sort, j'ai effectué une seconde version qui se base sur le cours du S2. Cependant, pour le faire fonctionner avec des animations, j'ai découpé mon programme en différents sous programmes qui peuvent être appelés avec une liste et une liste de pointeurs. La version que j'ai créé utilise une liste de pointeurs comme marquer de différentes sous listes, après une autre fonction utiliser ses et la liste pour fusionner la sous-liste 2 pars deux pour réaliser le merge sort.

Cependant j'ai trouvé un certain nombre de problèmes pour réaliser une modification en place donc j'utilise une sous liste. L'une des solutions qui pourrait rendre mon algorithme plus efficace serait avec l'utilisation de liste chaîné, mais qui ne sera pas le cas pour ce projet S4.

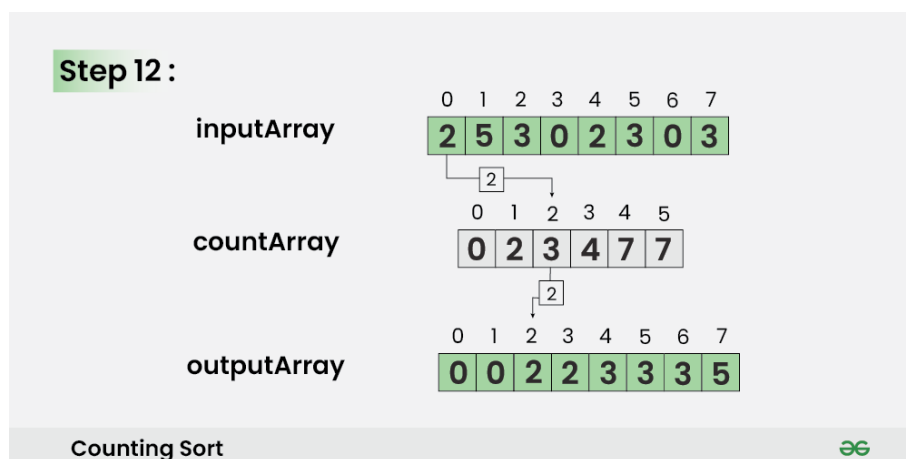


Pour conclure, cet algorithme est opérationnel et prêt à l'emploi sur l'interface graphique.

5.11 Counting sort

Le tri par dénombrement (counting sort) est l'un des algorithmes de tri le plus rapide, même s'il a quelques restrictions et défauts. Le tri s'exécute en un temps linéaire, mais uniquement sur des nombres entiers non signé (u64 par défaut). La particularité du tri est qu'il est la base d'autres algorithmes de tri en temps linéaires, permettant de s'adapter aux besoins en temps et en mémoire.

Le principe de cet algorithme est le suivant : on parcourt notre tableau puis on compte le nombre de fois où chaque élément doit apparaître. Dès que l'on obtient le tableau T ($T[i]$ correspond au nombre où il apparaît dans le tableau), nous avons la possibilité de le parcourir dans le sens croissant ou décroissant. Cela dépend du tri effectué. Il est ainsi possible de mettre dans le tableau trié $T[i]$ fois l'élément i , allant du minimum jusqu'au maximum du tableau.



Cette fonction prend en paramètre un tableau mutable car on va modifier son contenu pour le trier dans l'ordre croissant et une valeur : l qui sera la valeur maximale de notre tableau. J'ai préféré mettre en paramètre cette valeur maximale plutôt que la calculée car cela facilite les tests et les modifications à apporter si besoin pour l'amélioration du code. Le tableau est modifié en place pour éviter de recréer un tableau supplémentaire où l'on insère nos valeurs finales. Ma première version de l'algorithme était comme cela mais j'ai voulu essayer en place. Cela n'a pas été simple car il y avait des changements de calculs mais j'ai réussi à le faire.

Je commence par créer un tableau (vec!) de type `usize`, de longueur valeur maximale du tableau + 1 contenant que des 0. Ce vec va permettre de stocker toutes les occurrences des différents éléments présents dans notre tableau d'origine. Vient une première boucle : celle-ci va parcourir notre tableau et stocker le nombre de chaque élément du tableau d'origine avec leur indice correspondant. Puis, une fois ces éléments en mains, grâce aux index proposés, de parcourir notre tableau d'occurrences et en fonction des résultats de notre première boucle, il est possible de trier notre tableau (si le 1er élément à 5 comme nb d'occurrence et le 2e 2, le 2e sera placé ainsi avant le 1er et ainsi de suite).

Après avoir codé ma fonction, il fallait naturellement la tester. J'ai donc fait une fonction main basique permettant de tester mon code. J'ai initialisé deux tableaux de longueurs et valeurs différentes puis j'ai testé.

Le counting sort à un défaut majeur : il n'est possible de travailler qu'avec des nombres entiers positifs et qu'en temps linéaire. Par ailleurs, il est inefficace si la page de valeur à trier est très large. Toutefois, il fait partie des plus rapides algorithmes de tri basé sur des comparaisons. Quand nous donnons un tableau connu, nous avons la possibilité de se servir de cet algorithme pour avoir un temps d'exécution rapide et efficace. On pourrait modifier légèrement le fonctionnement de ce tri pour avoir une meilleur complexité en mémoire, en utilisant le Radix sort (tri par base), autre algorithme linéaire.

```
arr1 = [4, 3, 12, 1, 5, 5, 3, 9]
Application counting_sort: arr1 = [1, 3, 3, 4, 5, 5, 9, 12]
arr1 = [14, 5, 13, 13, 0, 512, 42, 6, 0, 1, 12, 1000, 80, 17, 7]
Application counting_sort: arr1 = [0, 0, 1, 5, 6, 7, 12, 13, 13, 14, 17, 42, 80, 512, 1000]
```

Je vous informe que cet algorithme est totalement fonctionnel sur le terminal.

5.12 Interface graphique

L'interface graphique sera implémentée avec gtk-rs, qui est une bibliothèque disponible sur crates.io. J'ai déjà rencontré plusieurs problèmes concernant gtk : la documentation propose peu d'exemples, est incomplète, c'est à dire, qu'elle ne montre pas toutes les fonctions qui peuvent être utilisées sur un struct. De plus, à cause d'un manque de connaissance sur les struct comme Arc ou Mutex, il fut très compliqué d'avoir une variable qui est "globale" donc accessible partout, surtout avec les fonctions de gtk, où j'ai encore des problèmes d'emprunts. L'interface se décompose en plusieurs parties

La partie haute c'est à dire la barre de Menu et la partie basse qui représente celle qui contient la partie la plus importante.

Pour la partie haute :

La barre du Menu est composée de 3 éléments : File, Edit et Help.

- File : ouvrir un fichier qui contient un tableau, un arbre binaire ou un graph.
- Edit : enregistrer un tableau/arbre/graph sous forme d'image ou en fichier txt ou dot.
- Help : un tutoriel pour utiliser l'application et les crédits.

Pour la partie basse :

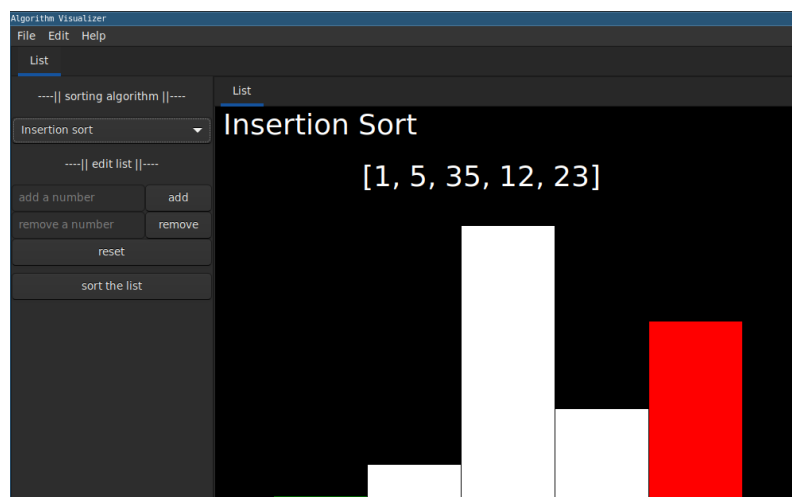
Elle contiendra 3 onglets ou chaque onglet permettra de faire des opérations sur les tableaux, arbres et graphes.

- List : onglet pour faire des opérations sur des tableaux.
 - Add : ajouter une valeur en fin de tableau.
 - Remove : retirer le premier élément rencontré dans le tableau.
 - Reset : enlever tous les éléments du tableau.
 - Sort the list : trie la list en fonction de l'algorithme choisi dans le menu déroulant.
 - Refresh : affiche l'état actuelle de la list
- Tree : onglet pour faire des opérations sur des arbres binaires.
 - Add : ajouter une valeur dans l'arbre.
 - Remove : retirer une valeur dans l'arbre.
 - Reset : enleve tout les noeud de l'arbre.
 - Search : rechercher la valeur entrée avec l'algorithme choisi
 - Refresh : supprime toutes les pages et affiche l'état actuelle de l'arbre
- Graph : onglet pour faire des recherches/opérations sur les graphes.
 - Add node : ajouter un sommet dans le graphe.
 - Add edge : ajouter un arc dans le graphe.
 - Remove node : retirer un sommet du graphe.
 - Remove edge : retirer un arc du graphe.
 - Search : rechercher la valeur entrée depuis un sommet donnée par l'utilisateur.

En ce qui concerne l'onglet Graph, son implémentation n'a pas été prévue lors des deux premières soutenances. Elle a été planifiée lors de l'ultime soutenance et sera réalisée à l'aide de GraphViz qui est une bibliothèque de Crates.io permettant de générer des images de graphs et d'arbres depuis leurs formats .dot.

Concernant les petits dysfonctionnements constatés lors de la première soutenance, nous avons le plaisir de vous informer que l'onglet List fonctionne désormais dans son intégralité tout comme l'onglet Tree. Cependant, nous vous confirmons que la sauvegarde de la list en tant qu'image est une petite implémentation qui sera réalisée pour l'ultime soutenance.

Le résultat est celui ci :

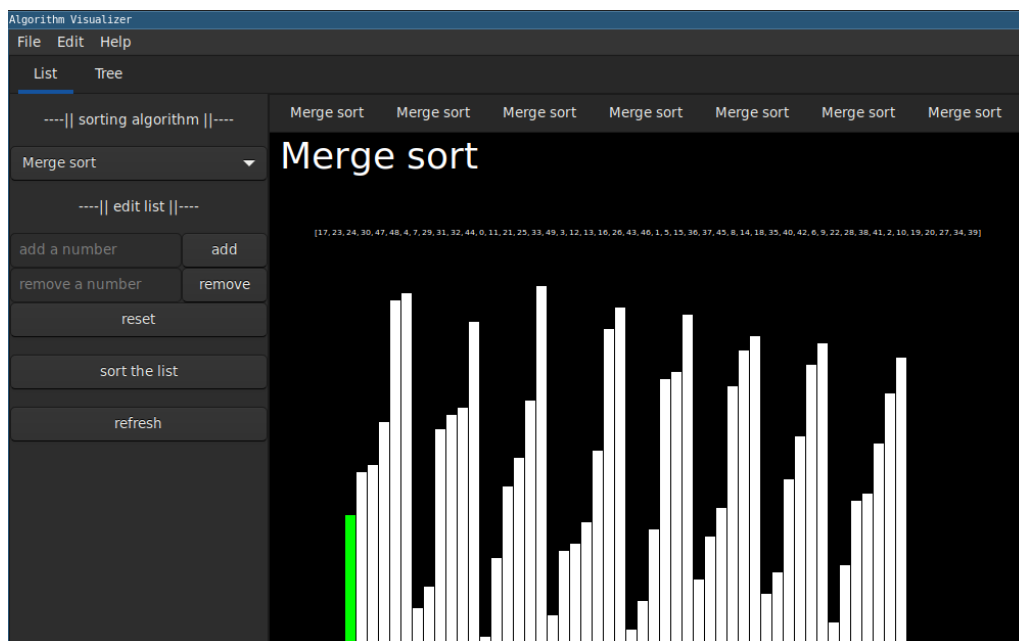


5.12.1 Mise a jour de qualite de vie

Depuis la première soutenance, une multitude de petites implémentations ont été faites pour une meilleure expérience lors de l'utilisation de l'interface.

Vous trouverez ci-dessous la liste des implémentations faites et leurs descriptions :

- Menu
 - Menu déroulant : utilisation des menus déroulant pour éviter de surcharger l'interface de boutons.
- List
 - Universel : l'affichage fonctionne aussi avec les nombres négatifs et zéro
 - Affichage adaptatif : l'affichage de l'état de la liste s'adapte en fonction de la longueur de celle-ci et change la police afin de l'afficher complètement
 - Refresh : ajout d'un bouton refresh pour enlever toutes les pages et affiche la page actuelle
 - Pages : les pages ont le nom de l'opération faite
 - Etat actuelle : affiche la dernière la page après chaque opération
- Tree
 - DFS : ajout des 3 parcours en profondeur (préfixe, infixe, suffixe)
 - Positon : affiche les anciennes positions et l'actuelle position dans l'arbre
 - Supression : affiche les etape de supression apres la recherche
 - Pages : les pages ont le nom de l'operation faite
 - Etat actuel : affiche la dernière la page après chaque opération



5.13 Site Internet

Dans le cadre de notre deuxième projet, nous avons développé deux sites web distincts. Le premier est dédié à la présentation du projet en lui-même, tandis que le second est consacré à notre groupe, les Alchimistes. Ce dernier a été créé au cours du deuxième semestre de l'année précédente, avec Olivier. Nous avons donc pensé qu'il serait intéressant de mettre en place un site web présentant les différents projets que nous avons réalisés.

Pour ce faire, nous avons utilisé les langages de programmation suivants :

- HTML / CSS
- JavaScript
- Tailwind CSS

Nous aurions souhaité intégrer le langage PHP dans sa version 8.0, qui corrige de nombreux défauts des versions précédentes. Nous aurions également aimé utiliser le framework Laravel, ce qui nous aurait permis de gérer une petite base de données pour les messages et les contacts. Cependant, nous avons rencontré un obstacle majeur : GitHub ne permet pas de stocker et d'exécuter du code PHP.

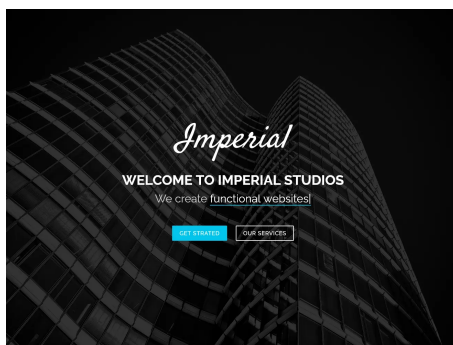
Ce site internet sera la vitrine de notre projet. Il est donc indispensable qu'il possède un beau design et qu'il soit simple et fluide au niveau de la navigation.

J'ai apprécié être en charge de la réalisation de ce nouveau site internet pour le projet du S4, car j'ai aussi effectué la création de site pour les projets du S2 et S3.

Pour réaliser le site de ce projet, j'ai pris la décision d'utiliser un Template : Bootstrap afin d'avoir une base sur laquelle travailler. Ce Template Imperial est un modèle d'une page Bootstrap moderne et créative, idéal pour les agences de création, les studios, les agences de design numérique ou d'autres entreprises similaires.

L'entête est livré avec une partie héros en plein écran. Imperial est également livré avec un menu mobile moderne hors toile pour une meilleure navigation et une meilleure expérience utilisateur. Bien évidemment, j'ai changé plusieurs lignes de codes afin d'obtenir un résultat optimal.

J'ai pris la décision d'utiliser le même Template que pour la création du site du S3 car selon moi, il est particulièrement riche en ressources avec de beaux designs en corrélation avec notre projet.



Commençons par parler de la conception graphique du site. La charte graphique doit être entièrement définie. Pour ce faire, nous avons prédéfini les logos, les couleurs et la typographie que nous appliquerons sur un style graphique accessible mais moderne.

Le Logo est symbolisé par le nom de notre groupe. Il est rouge avec quelques effets en blanc car le rouge reflète la force, l'énergie et la passion et surtout cette couleur attire l'attention.



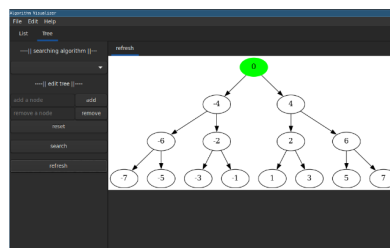
Ensuite, parlons des spécifications techniques. Le site est développé en partie sur une seule page appelée « one page » ou seule la partie présentation du projet renvoie pour chaque thème vers une autre page complète et détaillée.

Ce type de site est totalement adapté car il y a peu de contenu et il présente l'avantage de mieux contrôler la navigation des internautes en proposant une expérience utilisateur adaptée. Cela permet l'amélioration de la conversion du site web favorisant ainsi le référencement.

Laissez-moi vous montrer quelques captures écrans de notre site internet :

À PROPOS DE L'ALGORITHME VISUALIZER

Le visualizer d'algorithmes sera une application permettant de comparer visuellement différents types d'algorithmes tels que les algorithmes de parcours. Le but de cette application sera de comprendre leurs fonctionnements, leurs caractéristiques mais aussi, lequel est le plus efficace en termes de temps et d'optimisation.

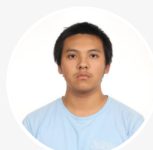


Comment comparer visuellement des algorithmes?

Afin de comparer visuellement ces différents algorithmes, nous aurons une interface graphique permettant de comparer ces algorithmes, différents paramètres à notre disposition pour s'intéresser aux différents parcours. Cette comparaison va vous permettre de vous donner une idée sur le fonctionnement mais aussi sur la complexité des différents algorithmes que l'on vous présente.

L'ÉQUIPE

Présentation des membres du groupe



Victor Tang
Lawless Inc
Twitter Facebook LinkedIn

“ Je trouve que le sujet est très passionnant. J'ai été responsable plusieurs fois pour le visuel d'une application. Faire l'application de A à Z est un vrai défi qui me donne envie de bien faire même si la documentation de GTK est faible et contient peu d'exemples. ”

RÉSUMÉ

Notre outil d'Algorithme Visualizer est une application puissante qui fait appel à des algorithmes de pointe pour comprendre, analyser rapidement n'importe quel algorithme tels que les algorithmes de tri, les algorithmes de plus courts chemins ou encore, des algorithmes sur les arbres binaires.



PRÉSENTATION

Ce site présente notre projet Algorithme Visualizer 2024. Les éléments fonctionnels du programme sont : les différents algorithmes proposés tels que les arbres binaires, graph et tri, puis notre interface graphique.



UTILITAIRES

Nous disposons également de petits utilitaires créés pour simplifier divers opérations répétitives (formatage du code, compilation de l'ensemble du projet, ...).



PLANNING

Le projet s'est déroulé en respectant les délais demandés et les choses se présentent de façon à ce que le projet soit terminé pour l'échéance requise. Chacun a travaillé dur pour proposer un travail de qualité.

Le deuxième site web que nous avons développé est un site de présentation du projet. Il s'agit d'un site épuré, mettant en avant les éléments les plus importants pour comprendre comment utiliser notre application, ainsi que les personnes qui ont contribué à sa création.

6 Conclusion

Pour cette seconde soutenance, nous sommes satisfaits de la continuité et de l'amélioration de certaines fonctionnalités de notre projet. Nous estimons que tout se déroule conformément à nos attentes. Pour rappel, les fonctionnalités principales du projet ont été implémentées avec succès et le prototype est fonctionnel. La force de notre groupe nous permet de surmonter les difficultés rencontrées et facilite notre créativité. De même, nous constatons que la communication est essentielle et indispensable à la bonne réalisation de ce projet.

Nous sommes satisfaits de la création des deux sites car ils sont le reflet de notre projet. Nous les trouvons « stylés » et espérons que vous aussi, vous partagerez notre ressenti. La fluidité de l'avancement du projet est rassurante et renforce notre confiance sur notre capacité à créer et à travailler ensemble. Désormais, nous sommes vraiment impatients de vous présenter notre projet dans sa version finale lors de l'ultime soutenance.