**Analyse**

**Déterminer le chemin le plus court entre deux cases d’une grille**

***Licence RGI - Groupe ERP CISCO :***

**Maël RHUIN**

**Sommaire**

**Approche**

***Sujet*** : Déterminer le chemin le plus court entre deux cases d’une grille d’entiers positifs.

***Réflexion autour de l’algorithme de Dijkstra :***

L’[algorithme de Dijkstra](https://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra) est une méthode qui permet de trouver le plus court chemin entre deux sommets d’un graphe pondéré par des réels positifs. Il a été inventé par le mathématicien et informaticien néerlandais [Edsger Dijkstra](https://fr.wikipedia.org/wiki/Edsger_Dijkstra) en 1956. L’idée principale de l’[algorithme](https://www.maths-cours.fr/methode/algorithme-de-dijkstra-etape-par-etape) est de maintenir un ensemble de sommets dont les distances minimales à la source sont connues, et d’ajouter progressivement le sommet le plus proche de la source à cet ensemble. Ci-après l’algorithme de Dijkstra en pseudo code :

// Entrée : un graphe G = (V, E) pondéré par des réels positifs et un sommet source s

// Sortie : un tableau dist qui contient les distances minimales de s à tous les autres sommets

// Initialisation

Pour chaque sommet v de V

dist[v] = +infini // Distance infinie à l'origine

visité[v] = faux // Sommet non visité

Fin pour

dist[s] = 0 // Distance nulle à la source

// Boucle principale

Tant qu'il existe un sommet non visité

u = le sommet non visité ayant la plus petite distance // Choix glouton

visité[u] = vrai // Marquer u comme visité

Pour chaque voisin v de u

si dist[u] + poids(u, v) < dist[v] // Relâchement des arêtes

dist[v] = dist[u] + poids(u, v) // Mise à jour de la distance de v

Fin si

Fin pour

Fin tant que

Retourner dist

**Cahier des charges**

La réalisation de cette analyse donnant lieu à un programme informatique est conditionnée par un cahier des charges :

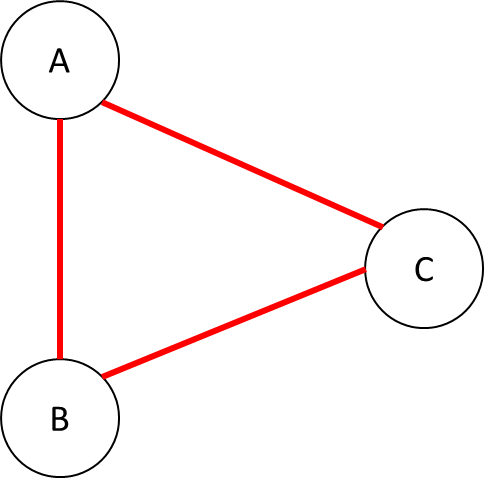
**Analyse de Niveau 0**

***Sujet*** : Déterminer le chemin le plus court entre deux cases d’une grille d’entiers positifs.

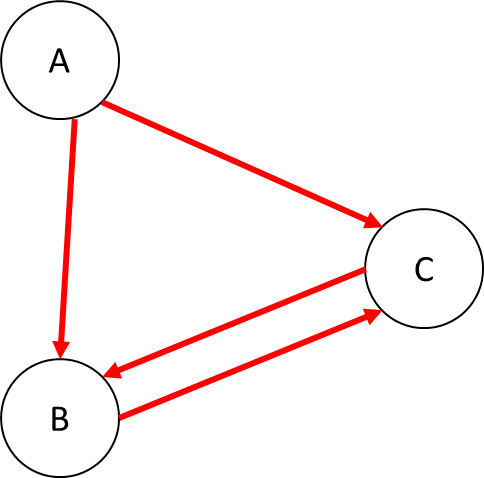
Après avoir trouvé une approche à la résolution de notre sujet. Il nous faut l’appliquer. Pour résumé l’algorithme de Dijkstra requiert un graphe orienté et pondéré (chaque arc possède un poids qui doit être ≥ 0.

***Qu'est-ce qu'un graphe ?***

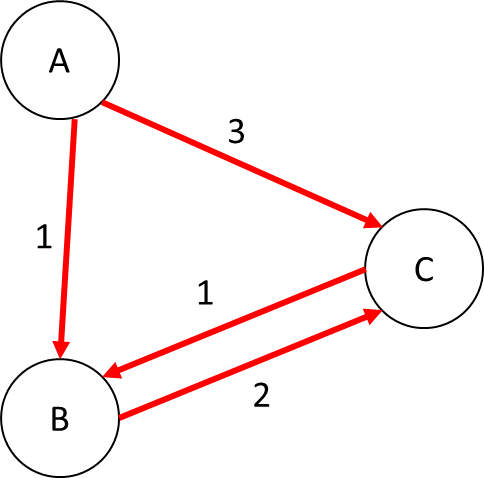
Un [graphe](https://fr.wikipedia.org/wiki/Graphe_(math%C3%A9matiques_discr%C3%A8tes)) en mathématiques est une structure composée d’objets appelés sommets et de relations entre eux appelées arêtes (ou arcs).

À gauche un graphe composé de trois sommets A, B, C relié par des arcs en rouge.

Un graphe ***orienté*** est un graphe où les arcs ont une direction représentée par une flèche.

 À gauche un graphe composé de trois sommets A, B, C relié par des arcs orientés en rouge.

Un graphe ***pondéré*** est un graphe où chaque arc porte un nombre appelé **poids**. Les poids peuvent représenter des coûts, des longueurs ou des capacités selon le problème.

 À gauche un graphe composé de trois sommets A, B, C relié par des arcs orientés et pondérés en rouge.

Il faut donc d’abord transformer notre matrice d’entiers en un graphe orienté et pondéré pour pouvoir lui appliquer Dijkstra.

***Comment faire ?***

Il nous faut découper le problème en plusieurs outils de résolution. Pour former notre graphe il nous faut des sommets, des arcs et des poids. Pour cela on décompose en objets :

Notre ***grille*** (ou ***plateau***) possède une retraduction en un graphe :

Un ***graphe*** est composé de sommets :

Un ***sommet*** possède un nom unique et des arcs orientés :

Un ***arc*** possède un sens (d’un sommet A vers B) et un poids positif.

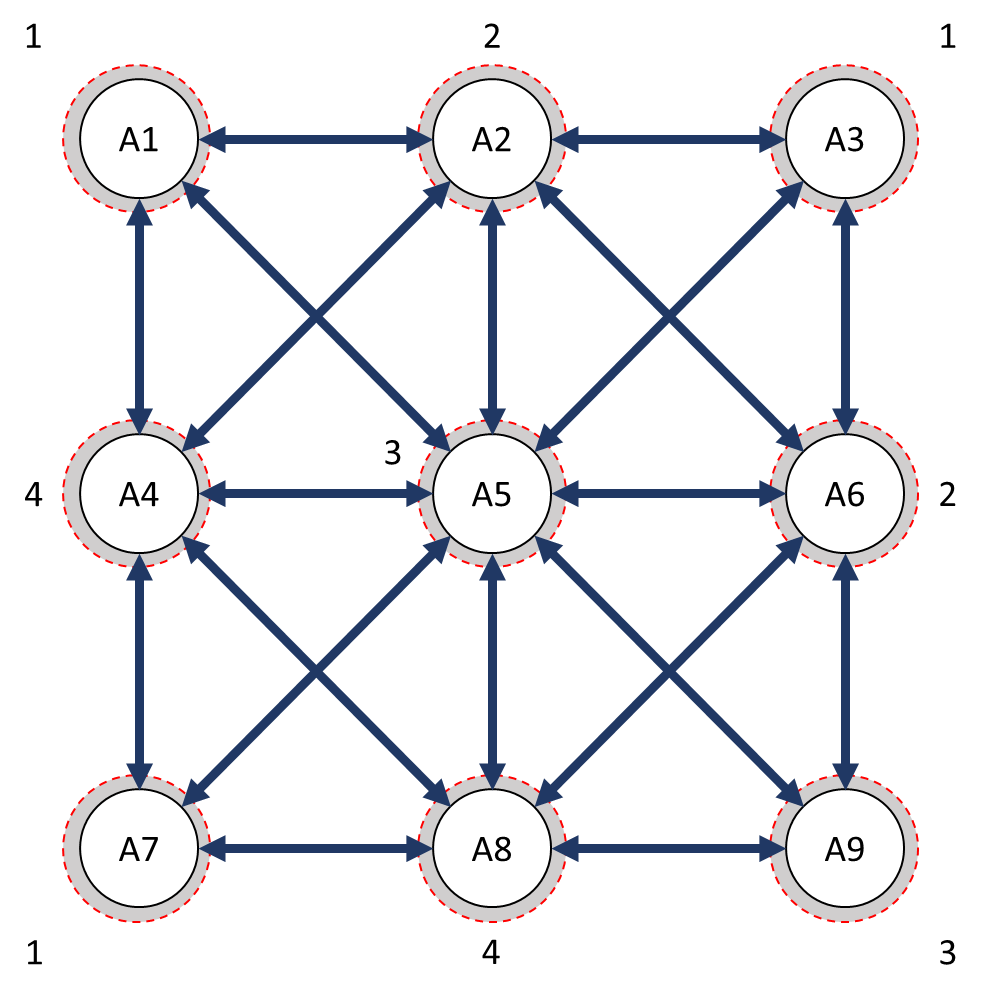
Soit une matrice d’entiers compris entre un minimum ***a*** et un maximum ***b*** de taille donnée par deux entiers ***l*** (largeur)et ***h*** (hauteur) :

***Exemple :***

***a = 1 et b = 4 ; l = 3 et h = 3***

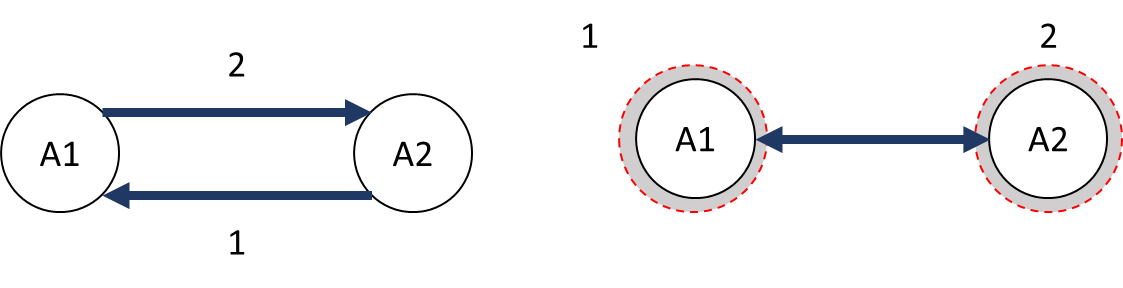
On affecte à chaque ***case*** de cette matrice un nouveau ***sommet*** de manière à obtenir une liste de sommets :

On créé pour chaque ***voisin*** (haut, bas, gauche, droite et diagonales) de chaque ***sommet*** un ***arc*** de ***poids*** correspondant à la ***case*** de notre matrice pour obtenir le graphe correspondant :

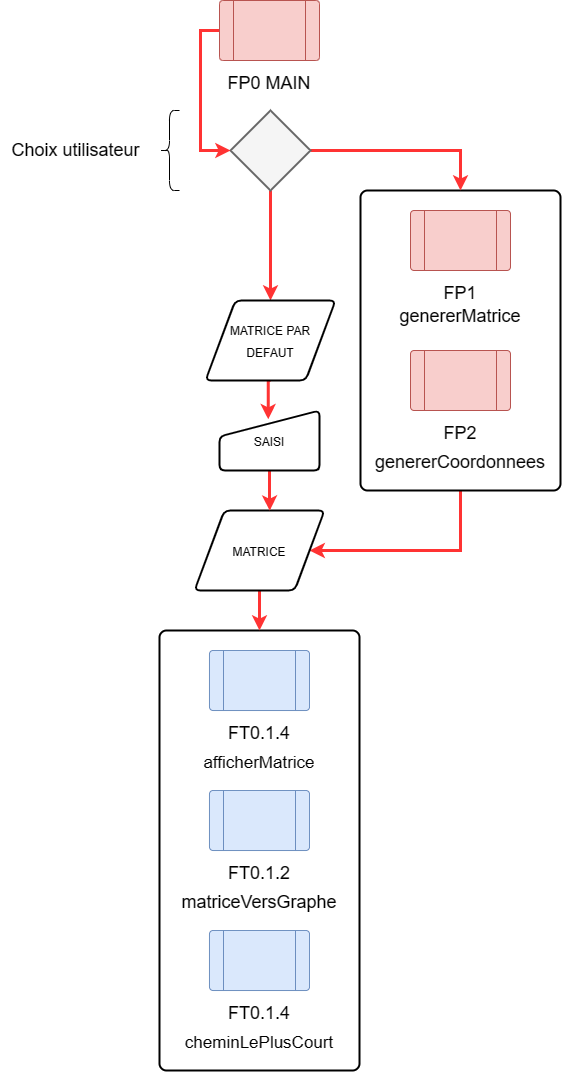


Chaque ***sommet*** est relié par des arcs bidirectionnels (en bleu) à ses voisins. Chaque ***arc*** entrant au sommet possède le ***poids*** (cercle rouge pointillé et sa valeur externe) de la ***case*** de la matrice correspondant au sommet.

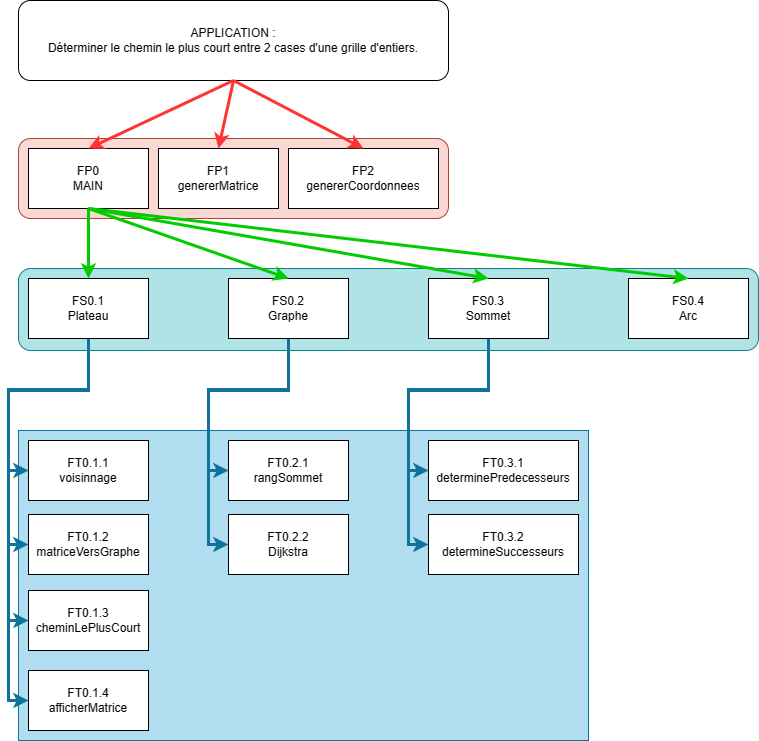
Le graphe maintenant créé, il ne nous reste plus qu’à appliquer l’algorithme de Dijkstra entre deux sommets choisit, afin d’en déterminer le chemin le plus court.

***(Note : par soucis de lisibilité du graphe des flèches bidirectionnelles ont été utilisées pour remplacer dans chaque cas deux flèches de sens opposés. Cependant le poids n’est pas obligatoirement le même dans les deux sens !)***

**Logigramme fonctionnel**



**Arbre hiérarchique**



**Analyse de niveau 1**

|  |  |
| --- | --- |
| FP0  MAIN |  |
| **Valeur Ajoutée**  Cette fonction principale permet de lancer le programme. Elle permet de choisir entre un plateau de jeu aléatoire ou un plateau de jeu prédéfini. On y utilise l'algo de Dijkstra pour trouver le chemin le plus court entre deux points. | |
| **INPUT**  …………… | **OUTPUT**  …………… |
| **Logigramme Fonctionnel**  MAIN | |
| **Service Fonctionnel**  Début du programme  Créer une matrice de 3x3 avec des valeurs entières prédéfinies.  Demander à l'utilisateur de choisir entre deux options en affichant un menu :  1.Trouver le chemin le plus court pour la matrice prédéfinie.  2.Trouver le chemin le plus court pour une matrice aléatoire générée par le programme.  Si l'utilisateur choisit l'option 1 :  Demander à l'utilisateur de saisir deux couples de coordonnées : une pour la case de départ et une pour la case d'arrivée.  Créer un objet Plateau à partir de la matrice.  (FT0.1.2) Transformer la matrice en graphe.  (FT0.1.4) Afficher la matrice.  (FT0.1.3) Trouver le chemin le plus court entre les coordonnées de départ et d'arrivée.  FinSi  Si l'utilisateur choisit l'option 2 :  Générer aléatoirement une matrice de dimensions comprises entre 3 et 32 avec des valeurs entières comprises entre 1 et la largeur.  Générer aléatoirement deux couples de coordonnées.  Créer un objet Plateau à partir de la matrice aléatoire.  (FT0.1.2) Transformer la matrice en graphe.  (FT0.1.4) Afficher la matrice.  (FT0.1.3) Trouver le chemin le plus court entre les coordonnées de départ et d'arrivée.  FinSi  Si l'utilisateur ne choisit aucune des deux options, afficher "Choix invalide".  Fin du programme | |

|  |  |
| --- | --- |
| FP1 genererMatrice |  |
| **Valeur Ajoutée**  Cette fonction permet de générer une matrice pour le plateau de jeu. | |
| **INPUT**  Largeur, hauteur  min valeur, max valeur | **OUTPUT**  matrice de taille hauteur x largeur avec des valeurs comprises entre min et max |
| **Logigramme Fonctionnel** | |
| **Service Fonctionnel**  Début de fonction genererMatrice(paramètres largeur, hauteur, min, max)  Déclarer une matrice de taille hauteur x largeur.    Pour chaque chaque ligne de la matrice :  Pour chaque chaque colonne de la matrice :  Affecter à chaque case de la matrice une valeur aléatoire comprise entre min et max.  Fin  Fin    Retourner la matrice générée.  Fin de fonction | |

|  |  |
| --- | --- |
| FP2 genererCoordonnees |  |
| **Valeur Ajoutée**  Cette fonction permet de générer des coordonnées aléatoires de départ et d'arrivée dans le plateau de jeu. | |
| **INPUT**  largeur, hauteur | **OUTPUT**  tableau de coordonnées comprises dans une taille donnée. |
| **Logigramme Fonctionnel** | |
| **Service Fonctionnel**  //FP2 genererCoordonnees  Début de fonction genererCoordonnees(paramètres largeur, hauteur)  Déclarer un tableau de deux entiers qui représente les coordonnées.    Affecter à la première case du tableau une valeur aléatoire comprise entre 0 et hauteur-1.  Affecter à la deuxième case du tableau une valeur aléatoire comprise entre 0 et largeur-1.    Retourner le tableau de coordonnées généré.  Fin de fonction | |

**Analyse de niveau 2**

|  |  |
| --- | --- |
| FS0.1  Plateau |  |
| **Valeur Ajoutée**  Cette classe représente le plateau de jeu. | |
| **INPUT**  …………… | **OUTPUT**  …………… |
| **Logigramme Fonctionnel**  Plateau | |
| **Service Fonctionnel**  //FS0.1  Attributs :  matrice d'entiers : matrice  objet graphe : gaphe  Début constructeur de Plateau (Paramètre matrice)  Attribut matrice = Paramètre matrice  Attribut graphe = Nouvel Objet Graphe.  Fin du constructeur | |

|  |  |
| --- | --- |
| FS0.2  Graphe |  |
| **Valeur Ajoutée**  Cette classe représente un graphe. Un graphe est caractérisé par une liste de sommets.  (Note : le graphe est celui représentant le plateau de jeu, il est pondéré de poids positifs et orienté) | |
| **INPUT**  …………… | **OUTPUT**  …………… |
| **Logigramme Fonctionnel**  Graphe | |
| **Service Fonctionnel**  //FS0.2  liste d'objets Sommet : sommets  Début constructeur de Graphe  Attribut sommet = Nouvelle liste vide  Fin du constructeur | |

|  |  |
| --- | --- |
| FS0.3  Sommet |  |
| **Valeur Ajoutée**  Cette classe représente un sommet du graphe. Un sommet est caractérisé par son nom, ses prédécesseurs, ses successeurs et ses arcs. | |
| **INPUT**  …………… | **OUTPUT**  …………… |
| **Logigramme Fonctionnel**  Graphe | |
| **Service Fonctionnel**  //FS0.3  Attributs :  Une chaine de caractères : nom  Liste de Sommet : predecesseurs  Liste de Sommet : successeurs  Liste d'Arcs : arcs  Constructeur de Sommet (paramètre chaine de caractères nom)  Attribut nom = paramètre nom  Attribut predecesseurs = Nouvelle liste vide  Attribut successeurs = Nouvelle liste vide  Attribut arcs = nouvelle liste vide  Fin de constructeur de Sommet  Constructeur de Sommet (paramètre Sommet sommet)  Attribut nom = paramètre sommet->nom  Attribut predecesseurs = sommet->predecesseurs  Attribut successeurs = sommet->successeurs  Attribut arcs = sommet->arcs  Fin de constructeur de Sommet | |

|  |  |
| --- | --- |
| FS0.4  Arc |  |
| **Valeur Ajoutée**  Cette classe représente un arc du graphe. Un arc est caractérisé par son sommet de départ, son sommet d'arrivée et son poids. | |
| **INPUT**  …………… | **OUTPUT**  …………… |
| **Logigramme Fonctionnel**  Arc | |
| **Service Fonctionnel**  //FS0.4  Attributs :  objet sommet : depart  objet sommet : arrive  entier : poids  Constructeur de Arc(paramètres Sommet depart, Sommet arrive, entier poids)  Attribut depart = paramètre depart  Attribut arrive = paramètre arrive  Attribut poids = paramètre poids  Fin de constructeur | |