ANNEXE : Guide d'utilisation

1 Introduction

Ce guide d'utilisation est destiné aux utilisateurs finaux de *Graphy*. Il contient un guide de démarrage rapide qui explique comment déclarer un graphe, y ajouter des sommets et des arêtes/arcs et comment appliquer un algorithme dessus. Une fois que cela est fait, il est également possible de sauvegarder et charger le travail et l'exporter en format SVG. Ce document n'explique pas le détail des algorithmes, puisque ces informations sont facilement disponibles en ligne.

2 Démarrage rapide

La première chose à faire après le lancement de l'application est la création d'un nouvel onglet pour travailler. Ceci peut se faire via le menu `Graph->new` ou directement avec le raccourci clavier `Ctrl + N`. Il est à présent possible de taper des commandes.

2.1 Déclaration d'un graph

Un graph vide se déclare simplement avec `nomDeGraph = { };`. Il est également possible de déclarer un graphe directement avec des sommets et des arêtes ainsi : `g = {0,1,2,0-1,0-2};`. Ici, les numéros `0`, `1` et `2` correspondent aux sommets, alors que les numéros liés par deux tirets (`0--1` et `0--2`) correspondent à des arêtes entre les sommets respectifs.

2.2 Application d'un algorithme

Puisque les algorithmes renvoient un tableau ou un nouveau graph, il faut stocker ce résultat dans une variable avant de pouvoir l'afficher. Par exemple, `h = dijkstra(g, 1);` renvoie un tableau avec comme premier élément le graphe rendu par l'algorithme. Pour le dessiner ensuite, il faut assigner ce nouveau graphe à une nouvelle variable avec `l = h[0];`, puis le dessiner à l'aide de `draw(l);`. La liste des algorithmes disponibles et leur retour sont listés plus loin dans ce document.

3 Graphes

Il existe plusieurs manières de déclarer un graphe, du simple graphe vide : `g = { };`, à un graphe contenant plusieurs sommets : `g = {#3}`, ou encore un graphe avec des sommets et des arêtes : `g = {0,1,2,0--1,0--2};`. Il est également possible d'ajouter un sommet ou une arête avec l'opérateur `+=`, par exemple : `g += (3);` ou `g += (1--2);`.

3.1 Règles de déclaration et d'ajout

Deux règles s'appliquent lors de la déclaration d'un graphe ou de l'ajout d'éléments. Premièrement, pour l'ajout de sommets, il est impératif que tous les ID plus petits que celui du sommet courant soient utilisés. Par exemple, si l'on souhaite ajouter le sommet `3`, il faut que les sommets `0`, `1` et `2` soient présent dans le graphe. Il n'est pas non plus possible de sauter des sommets, en ajoutant par exemple le sommet `7` à un graphe qui n'en contient que 3. Deuxièmement, pour ajouter une arête à un graphe, il faut que les deux sommets des extrémités existent dans le graphe.

4 Sommets

4.1 Déclaration

Un sommet peut être déclaré avec un simple ID, mais peut également l'être avec un label, un poids, une capacité maximale et une capacité minimale. Un sommet simple peut être déclaré avec `v = (1);`. Si on souhaite ajouter plus d'informations, cela est possible en séparant chaque propriété avec le `:` deux-points, selon la grammaire suivante : `nomDeVariable = (ID:"label":poids:capacitéMax:capacitéMin);`, par exemple : `v = (0:"monPremierSommet":3:5:2);`. Si on souhaite ajouter uniquement certaines propriétés, il est possible de les spécifier en plaçant le bon nombre de `:` deux-points avant, par exemple pour préciser uniquement

la capacité maximale et minimale, il suffit de tapper : `v = (0:::5:2);`. Si, au contraire, on veut uniquement spécifier les première propriétés, il n'est pas nécessaire d'inclure les `:` deux-points après, exemple : `v = (0:"monPremierSommer");`. Si aucun label n'est spécifié, le numéro d'ID sera utilisé par défaut.

4.2 Ajout à un graphe

Une fois qu'un graphe a été créé, il est possible de lui ajouter des sommets, avec l'opérateur `+=`, par exemple `g += v;`. Pour ajouter plusieurs sommets, il faut utiliser l'opérateur de tableaux `[]` crochets, par exemple `g += [v,w,x];`. Il est également possible d'ajouter des sommets temporaires avec l'opérateur `()` parenthèse : `g += (1);`. On peut en outre mélanger les notations dans au sein de l'opérateur `[]` crochets, par exemple `g += [(1), (2:"monSecondSommet"), w];`.

4.3 Modification d'un sommet dans un graphe

Le propriétés d'un sommet peuvent être modifiées en utilisant l'opérateur d'ajout `+=`. Pour cela, il suffit de spécifier l'ID d'un sommet existant, puis ses propriétés. Par exemple, `g = { };`, suivi de `g += [(0::5), (1::5)];` et enfin `g += (1::3);` aura pour effet de changer le poids du sommet avec l'ID `1` de `5` à `3`.

4.4 Supprimer un sommet d'un graphe

La suppression des sommets est faite de la même manière que l'ajout, mais avec l'opérateur `-=`. Comme avec l'opérateur d'ajout, il est possible de supprimer un seul sommet ou d'en supprimer plusieurs avec la notation de tableaux `[ ]`. Les sommets doivent être identifiés par leur ID, par exemple : `g -= [(1), (2)];`. Une fois qu'un sommet est supprimé toutes les arêtes correspondantes seront également supprimées du graphe. Remarque : si on essaye de supprimer un sommet inexistant, rien ne se passe.

5 Arêtes

5.1 Déclaration

Tout comme avec les sommets, une arête peut être déclarée comme une simple connexion entre deux sommets, par exemple : `e = (0-1);`. Il peut également inclure d'autres informations, comme le label, le poids, la capacité maximale et la capacité minimale selon la même grammaire que les sommets, à savoir : `nomDeVariable = (IDpremierSommet-IDsecondSommet:"label":poids:capacitéMaximale:capacitéMinimale);`.

5.2 ID de l'arête

Chaque arête a un ID local a la connexion. Une connexion est une relation symbolique entre deux sommets, cela signifie qu'il peut y avoir plusieurs arêtes avec le même ID dans un graphe. Cet ID local est utilisé pour différentier les arêtes lors de la suppression dans une connexion contenant plusieurs arêtes.

5.3 Ajout à un graphe et modification

L'ajout et la modification d'une arête dans un graphe se font de la même manière qu'avec les sommets.

5.4 Suppression d'une arête

La suppression se fait avec l'opérateur `-=` comme avec les sommets. Une différence est qu'il peut exister plusieurs arête entre deux sommets. Ceux-ci peuvent donc être différentiés avec l'opérateur d'accès `[ ]`, par exemple : `g -= (0-1)[0];` supprime l'arête avec l'ID local `0`.

6 Arcs

Il est également possible d'ajouter des arcs à un graphe. Ceux-ci sont représentés avec les symboles `->` et `<-`. Ajouter un arc à un graphe transforme tous les arcs présents en deux arcs de sens opposé. Exécuter `dg = {#3, 0--1};` et `dg += (1->2);` créera donc le même graphe que la commande `dg = {#3, 0->1, 1->0, 1->2};`. Toutes les règles mentionnées pour les arêtes s'appliquent également pour les arcs.

7 Fonctions

7.1 Mode de passage des paramètres

Tous les paramètres dans *Graphy* sont passés par valeur. Cela veut dire que modifier un sommet après l'avoir ajouté à un graphe n'aura aucune influence sur ce dernier. Par exemple en tapant : `g = { };`, puis `v = (0::5);`, ensuite `g += v;` et enfin `v = (0::3);`, le sommet avec l'ID `0` aura un poids de `5` et non pas de `3`.

7.2 Appel de fonctions

Toutes les transformations sont faites en passant un graphe à une fonction. Dès qu'une fonction a été appelée, le résultat peut être assigné à un graphe existant, au même graphe, ou encore à un nouveau graphe. La notation est semblable à celle de beaucoup de langages de programmation : `resultat = dijkstra(g, 1);`. Remarque : plusieurs fonctions retournent un booléen pour savoir si l'opération a pu être effectuée ou pas.

7.3 Fonctions standards

- `toString (a : T) : String` retourne un String décrivant a.
- `save (g : Graph, file : String) : Boolean` sauvegarde le graphe en paramètre dans le répertoire courant.
- `load (file : String) : Graph` charge un graphe spécifié du répertoire courant.
- `type (a : T) : String` retourne le type de a.
- `er (V : Integer, p : float) : Graph` génère un graphe aléatoire avec `T` sommets et une probabilité `p` d'avoir une arête, selon le modèle de Erdős–Rényi.

7.4 Fonctions de l'interface utilisateur

- `draw (g : Graph) : Boolean` dessine `g` dans une nouvelle fenêtre.
- `exportAsSvg (g : Graph) : Boolean` exporte le graphe en format SVG dans le répertoire courant.
- `exportAsSvg (g : Graph) : Boolean` exporte le graphe en format SVG dans le répertoire courant avec le nom passé en paramètre.

7.5 Algorithmes

- `bellmanFord (g : Graph, from : Integer) : Array` applique l'algorithme de Bellman-Ford au graphe `g` depuis le sommet `from`. Le tableau (`Array`) retourné contient le graphe résultant à l'index `0` et la matrice des distances à l'index `1`.
- `bfs (g : Graph, from : Integer) : Array` applique l'algorithme de parcours en largeur depuis le sommet spécifié. Le tableau de retour contient le graphe résultant à l'index `0` et la matrice des distances à l'index `1`.
- `cc (g : Graph) : Array` applique l'algorithme des composantes connexes au graphe depuis le sommet spécifié et retourne un tableau associant chaque sommet avec une composante, par exemple : `[1,1,2,2,3]` indique que les sommets `0` et `1` sont dans la même composante, ainsi que les sommets `2` et `3`, alors que le sommet `4` est seul dans une composante.
- `detectCycle (g : Graph) : Graph` retourne le cycle d'un graphe s'il y en a un.
- `dfs (g : Graph, from : Integer) : Array` applique l'algorithme de parcours en profondeur et retourne un tableau avec le graphe résultant à l'index `0` et la matrice des distances à l'index `1`.
- `dijkstra (g : Graph, from : Integer) : Array` applique l'algorithmes de Dijkstra et retourne un tableau contenant le graphe résultant à l'index `0` et la matrice des distances à l'index `1`.
- `kruskal (g : Graph) : Graph` applique l'algorithme de Kruskal à `g` et retourne le graphe résultant.
- `tarjan (g : Graph) : Array` applique l'algorithme de Tarjan au graphe `g` (qui doit être orienté) et retourne un tableau, contenant le graphe résultant à l'index `0` et un tableau contenant les composantes connexes à l'index `1` (voir `cc` pour un exemple).
- `topologicalSort (g : Graph) : Array` applique l'algorithme de tri topologique au graphe `g` (qui doit être orienté et acyclique) et retourne un tableau contenant l'ordre de visite des sommets.
- `isConnected (g : Graph) : Boolean` indique si le graphe est connexe.
- `isDirected (g : Graph) : Boolean` indique si le graphe est orienté

— `isEmpty (g : Graph) : Boolean` indique si le graphe est vide (sans arête ou arc).

— `isNegativeWeighted (g : Graph) : Boolean` indique si le graphe contient des arêtes ou arcs avec une pondération négative.

— `isNull (g : Graph) : Boolean` indique si le graphe est nul (sans sommets, ni arêtes).

— `isPlanar (g : Graph) : Boolean` indique si le graphe est planaire.

— `isSimple (g : Graph) : Boolean` indique si le graphe est simple (sans arête/arcs doublés).

— `isStronglyConnected (g : Graph) : Boolean` indique si le graphe est fortement connexe (pour les graphes orientés uniquement).

— `isWeighted (g : Graph) : Boolean` indique si le graphe contient des arêtes ou arcs pondérés.