PROJET RAD

La spécification algébrique

VOEDZO ESSIVI MARIE-JOSEE
DE BADAR BADAROU Hosniath
Mohamed Lemine Abdellahi

Spécification algébrique des graphes

G : Type Graphe indiquant qu'on obtient un graphe en sortie ;
E : Type Numérique indiquant qu'on obtient un numérique en sortie ici le poids ;
B : Type Booléen indiquant qu'on obtient un booléen en sortie {vrai ou faux};
N : Type nœud indiquant qu'on obtient un nœud en sortie ;
A : Type arc indiquant qu'on obtient un arc en sortie ;

Observateurs	Opérateur interne
• Are_connected: G X N X N ➔B. • Get_degree: G X N ➔ E • Get_lightest : G ➔ A • Get_weight : GX A➔E • Num_edge : G ➔ E	• Graph : ➔ G • Add_edge: G X N X N X E ➔G • Add_node: G X N ➔ G • Del_edge: G X A ➔ G • Del_node: G X N ➔G

Précondition

- Add_edge (N1, N2, E) ssi N1 et N2 existe dans G
- Add_node (N1) ssi N1 n'existe pas encore dans G et ssi G existe
- Del_edge (A) ssi A existe dans G
- Del_node(N) ssi N existe dans G
- Are_connected (N1, N2) ssi N1 et N2 existe dans G
- Get_degree (N) ssi N existe dans G
- Get_weight (A) ssi A existe dans G
- Get_lightest G

Axiome

Num_edge

- Num_edge (Graph ()) == 0
- Num_edge (Add_edge (G, N1, N2, E)) == {Num_edge (G) +1 si (N1, N2) n'existait pas dans G Num_edge (G) s'il existait}
- Num_edge (Add_node (N1)) == Num_edge (G)
- Num_edge (Del_edge (A)) == si l'arc en question était présente dans G Num_edge (G) -1 et sinon Num_edge (G) ;
- Num_edge (Del_node (N)) == si le nœud en question n'existe pas dans G : Num_edge (G) sinon Num_edge (G) – Get_degree (N) ;

Get_degree

- Get_degree (Graph (), N) == 0
- Get_degree (Add_edge (G, N1, N2, E), N) == {Num_edge (G) +1 si (N1, N2) n'existait pas dans G sinon Num_edge (G) s'il existait}

- Get_degree (Add_node (N1), N2) == si N1=N2 alors 0 sinon Get_degree (G, N2)
- Get_degree (Del_edge (A)) == si l'arc en question était présente dans G Num_edge (G) -1 sinon Num_edge (G) ;
- Get_degree (Del_node (N), N') == si le nœud en question n'existe pas dans G : Num_edge (G) sinon Num_edge (G) – Get_degree (N) ;

Get_lightest

- Get_lightest (Add_edge (G, N1, N2, poids)) =Get_lightest (G) si poids > Get_lightest (G) sinon =A {N1, N2, poids) : arc de nœud 1 et 2 avec un poids E
- Get_lightest (Add_node (G, n)) = Get_lightest (G)) puisqu'aucun arc n'est relié au nouveau nœud, l'arc de poids le plus faible ne changera donc pas
- Get_lightest (Graph ()) = violation car un graphe vide ne peut avoir d'arc de poids plus faible

- Get_lightest (Del_edge (G, n1, n2)) =Get_lightest (Del_edge (G, n1, n2))) si A {n1, n2, poids} = Get_lightest (G) sinon = Get_lightest (G) c'est-à-dire si l'arc supprimé était le plus faible alors on recherche le suivant sinon il reste le même

- Get_lightest (Del_node (G, n)) si ce nœud était relié à l'arc le plus faible, celui-ci changera à la suite de la suppression sinon il reste le même

Get_weight

- Get_weight (Add_edge (G, n1, n2, poids), n1', n2') = poids si (n1=n1' et n2=n2') ou (n1=n2' et n2=n1') sinon retourne Get_weight (G, n1', n2')
- Get_weight (Add_node (G, n), n1', n2') = violation si (n=n1' et n=n2') sinon Get_weight (G, n1', n2')
- Get_weight (Del_edge (G, n1, n2), n1', n2') si (n1=n1' et n2=n2') ou (n1=n2' et n2=n1') retourne une violation car on ne peut retourner le poids d'un arc qui n'existe pas Sinon retourne Get_weight (G, n1, n2)
- Get_weight (Del_node (G, n), n1, n2) violation si (n=n1' et n=n2') car l'un des nœuds étant supprimé, il ne peut exister un arc Sinon Get_weight (G, n1', n2')
- Get_weight (Graph (), n1, n2) = violation car le graphe est vide

Are_connected.

- Are_connected (Add_edge (G, n1, n2, poids), n1',n2') = vrai si (n1=n1' et n2=n2') ou (n1=n2' et n2=n1') sinon retourne Are_connected (G,n1',n2')

- Are_connected (Add_node (G, n), n1', n2') = Faux si (n=n1' et n=n2') car tout nouveau nœud ajouté n'est à priori relié à aucun autre nœud sinon Are_connected (G, n1', n2')

- Are_connected (Del_edge (G, n1, n2), n1', n2') si (n1=n1' et n2=n2') ou (n1=n2' et n2=n1') retourne Faux car l'arc entre ces nœuds a été supprimé Sinon retourne Are_connected (G, n1, n2)
- Are_connected (Del_node (G, n), n1, n2) Faux si (n=n1 ou n=n2) car l'un des nœuds n'est plus dans le graphe sinon Are_connected (G, n1', n2')

- Are_connected (Graph (), n1, n2) = violation car le graphe est vide

QUESTION 7 : ALTERNATIVE

```
La structure efficace pour stocker les arcs serait un HashMap et on prendra
comme clé :
 * le poids de l'arc
 * L'arc a plusieurs valeurs sommet1,sommet2,et le poids mais celui au quelle
on fait le plus appel est le poids
 * donc il est préférable de l'utiliser
```