Algoritmos y Estructuras de Datos II - 14 de mayo de 2012

Parcial 2: Estructuras de Datos

Docentes: Daniel Fridlender y Silvia Pelozo

Alumno: .....

1. Se define el TAD grafo, de los grafos dirigidos cuyos nodos son números naturales. El tipo abstracto tiene 2 constructores: uno para crear el grafo vacío (sin nodos ni aristas) y otro para agregar una arista entre dos nodos a un grafo preexistente. Este último constructor tiene por argumento dos nodos u y v y un grafo G y agrega a G una arista que va de u a v, es decir, agrega la arista $u \to v$. Es posible que alguno de los nodos u y v, o ambos, ya estén en G, pero también es posible que no estén aún en G (pero obviamente pasan a estar como resultado de agregar la arista $u \to v$).

TAD grafo
constructores
vacío : grafo
agregar_arista : natural × natural × grafo → grafo
operaciones
es_vacío : grafo → booleano
está_arista : natural × natural × grafo → booleano
cantidad_aristas : grafo → natural
está_nodo : natural × grafo → booleano
cantidad_nodos : grafo → natural
elim_arista : natural × natural × grafo → grafo
elim_nodo : natural × grafo → grafo
ecuaciones
agregar_arista(u,v,agregar_arista(u,v,g)) = agregar_arista(u,v,g)
agregar_arista(u,v,agregar_arista(u',v',g)) = agregar_arista(u',v',agregar_arista(u,v,g))

es_vacío(vacío) = verdadero
es_vacío(agregar_arista(u,v,g)) = falso

está_arista(u,v,vacío) = falso
está_arista(u,v,agregar_arista(u,v,g)) = verdadero
u ≠ u' ∨ v ≠ v' ⟹ está_arista(u,v,agregar_arista(u',v',g)) = está_arista(u,v,g)

cantidad_aristas(vacío) = 0
está_arista(u,v,g) ⟹ cantidad_aristas(agregar_arista(u,v,g)) = cantidad_aristas(g)
¬está_arista(u,v,g) ⟹ cantidad_aristas(agregar_arista(u,v,g)) = 1 + cantidad_aristas(g)

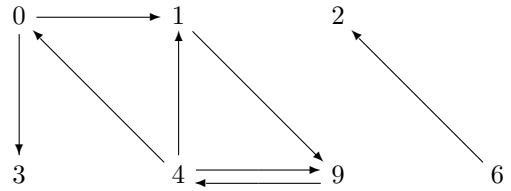
está_nodo(u,vacío) = falso
está_nodo(u,agregar_arista(u,v,g)) = verdadero
está_nodo(v,agregar_arista(u,v,g)) = verdadero
u ≠ u' ∧ u ≠ v' ⟹ está_nodo(u,agregar_arista(u',v',g)) = está_nodo(u,g)

cantidad_nodos(vacío) = 0
está_nodo(u,g) ∧ está_nodo(v,g) ⟹ cantidad_nodos(agregar_arista(u,v,g)) = cantidad_nodos(g)
¬está_nodo(u,g) ⟹ cantidad_nodos(agregar_arista(u,u,g)) = 1 + cantidad_nodos(g)
¬está_nodo(u,g)∧¬está_nodo(v,g)∧u ≠ v ⟹ cantidad_nodos(agregar_arista(u,v,g)) = 2 + cantidad_nodos(g)
está_nodo(u,g) ∧ ¬está_nodo(v,g) ⟹ cantidad_nodos(agregar_arista(u,v,g)) = 1 + cantidad_nodos(g)
¬está_nodo(u,g) ∧ está_nodo(v,g) ⟹ cantidad_nodos(agregar_arista(u,v,g)) = 1 + cantidad_nodos(g)

elim_arista(u,v,vacío) = vacío
elim_arista(u,v,agregar_arista(u,v,g)) = elim_arista(u,v,g)
u ≠ u' ∨ v ≠ v' ⟹ elim_arista(u,v,agregar_arista(u',v',g)) = agregar_arista(u',v',elim_arista(u,v,g))

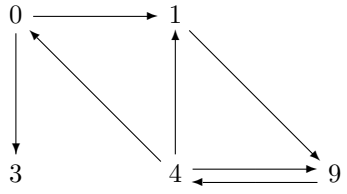
elim_nodo(u,vacío) = vacío
elim_nodo(u,agregar_arista(u,v,g)) = elim_nodo(u,g)
elim_nodo(v,agregar_arista(u,v,g)) = elim_nodo(v,g)
u ≠ u' ∧ u ≠ v' ⟹ elim_nodo(u,agregar_arista(u',v',g)) = agregar_arista(u',v',elim_nodo(u,g))

Así, el grafo

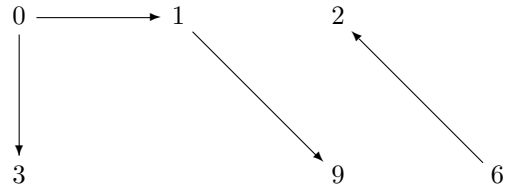


se puede construir como g = a_a(9,4,a_a(6,2,a_a(4,9,a_a(0,3,a_a(4,0,a_a(1,9,a_a(0,1,a_a(4,1,vacía)))))))), donde a_a es usado para abreviar el constructor agregar_arista. Observar que también se puede construir de otras maneras, ya que el orden en que se agregan aristas no importa; además podría agregarse la misma arista varias veces, por ejemplo agregar_arista(4,0,g) da el mismo grafo porque dicha arista ya está.

Observar que los nodos 7 y 8 no están en el grafo, no hay ninguna necesidad de usar los nodos en el orden natural. Por ello, se tiene está_nodo(7,g) = falso pero está_nodo(9,g) = verdadero. Asimismo está_arista(2,6,g) = false pero está_arista(6,2,g) = verdadero. También cantidad_aristas(g) = 8 y cantidad_nodos(g) = 7. Además, elim_arista(2,6,g) = g pues la arista 2 $\longrightarrow$ 6 no está en el grafo g, mientras que elim_arista(6,2,g) da el grafo



mientras que elim_nodo(4,g) da el grafo



a) Implementar el TAD grafo con una matriz de adyacencia. Es decir, se asume que los grafos usarán solamente naturales entre 0 y N para los nodos, y se define

type graph = **array**[0..N,0..N] **of bool**

Si g tiene tipo graph, $g[u, v]$ = **true** sii el grafo que implementa g tiene una arista de u a v. El grafo g del ejemplo de arriba sería

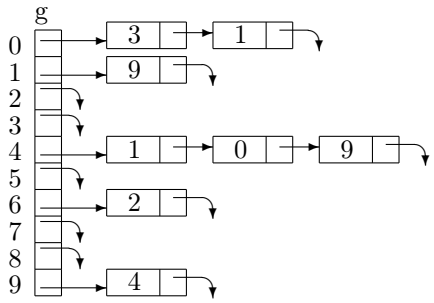
g	0	1	2	3	4	5	6	7	8	9
0		t		t						
1										t
2										
3										
4	t	t								t
5										
6			t							
7										
8										
9					t					

donde t es **true**, y los casilleros dejados en blanco, **false**. Con esta representación, implementar todas las operaciones del TAD: procedimientos empty (vacío), add_edge (agregar_arista), erase_edge (elim_arista) y erase_node (elim_nodo), y las funciones is_empty (es_vacío), edge_is_in (está_arista), node_is_in (está_nodo), number_of_edges (cantidad_aristas) y number_of_nodes (cantidad_nodos).

b) Implementar el TAD grafo con un arreglo de listas de adyacencia. Es decir, se asume que los grafos usarán solamente naturales entre 0 y N para los nodos, y se define

```
type graph = array[0..N] of ady_list
type ady_list = pointer to node
type node = tuple
              target: nat
              next: pointer to node
            end
```

Si g tiene tipo graph, v pertenece a la lista que comienza en $g[u]$ sii el grafo que g implementa tiene una arista de u a v. El grafo g del ejemplo de arriba sería

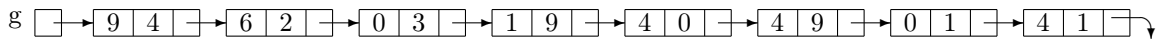


Con esta representación, implementar los procedimientos empty, add_edge y erase_edge. Se recomienda mantener cada una de las listas de nodos sin repeticiones, para lo cual puede convenir utilizar la función edge_is_in en el procedimiento add_edge. Implementar las funciones is_empty, edge_is_in y number_of_edges. Explicar las dificultades de implementar el procedimiento erase_node y las funciones node_is_in y number_of_nodes. Explicar intuitivamente cómo las implementaría.

c) Implementar el TAD grafo con una lista de aristas. Es decir, se define

type graph = **pointer to** edge
type edge = **tuple**
source: **nat**
target: **nat**
next: **pointer to** edge
end

Si g tiene tipo graph, el grafo que implementa tiene una arista de u a v sii hay una arista en la lista g que tiene el campo source igual a u y target igual a v. El grafo g del ejemplo de arriba sería



Con esta representación, implementar los procedimientos empty, add_edge y erase_edge. También se recomienda mantener la lista sin repeticiones, para lo cual vale la misma sugerencia que en la representación anterior. Implementar las funciones is_empty, edge_is_in, number_of_edges y node_is_in. Explicar las dificultades de implementar el procedimiento erase_node y la función number_of_nodes. Explicar intuitivamente cómo las implementaría

2. Una empresa de servicio técnico está diseñando un sistema para la asignación de tareas a sus empleados.

Cuando un empleado finalice una tarea se le asignará la siguiente tarea según el orden en que las mismas fueron ingresando al repositorio de tareas pendientes.

Además, si un empleado no pudo completar una tarea por algún motivo (falta de tiempo, desconocimiento, etc.) puede devolverla al repositorio, pero esta vez al principio del mismo, de forma de que la tarea se asigne al siguiente empleado que se desocupe.

Especifique un tipo abstracto de datos (constructores, operaciones y ecuaciones) para representar la cola o repositorio de tareas, que permita almacenar las tareas en orden de ingreso a la cola, y cuente (como mínimo) con operaciones para:

- crear el repositorio vacío
- agregar al final de la cola una tarea a asignar eventualmente a un empleado
- determinar si hay alguna tarea pendiente
- determinar la primera tarea a asignar al próximo empleado que se desocupe
- eliminar la primera tarea a asignar
- agregar una tarea que no pudo finalizarse al principio de la cola

Implementar este TAD utilizando arreglos circulares.