Auxiliar 11 - Grafos

Profesores: Nelson Baloian
Jeremy Barbay
Patricio Poblete

Auxiliares: Gabriel Flores, Sven Reisenegger
Juglar Díaz, Gabriel Norambuena
Cristóbal Muñoz

P1. Grafos

Un grafo G es un par ordenado $G = (V, E)$ en donde V representa un conjunto de nodos o vértices y E representa un conjunto de pares ordenados donde cada tupla relaciona dos vértices distintos mediante una arista.

Se pide que programe una clase *Grafo*, con un constructor `Grafo(int numNodos)`, el método `agregarArista(int origen, int destino)`, que agrega una nueva arista entre dos nodos y el método `hayArista(int origen, int destino)`, que revisa si existe una arista entre dos nodos. Para ello se le pide que lo implemente usando las siguientes variantes:

(a) Utilizando una matriz de adyacencia.

(b) Utilizando una lista de adyacencia.

(c) Utilizando una lista con las aristas del grafo.

P2. Recorrido en grafos

Utilizando la implementación de la parte anterior se pide que programe el algoritmo de búsqueda por anchura visto en clases (Breadth First Search). Su método debería tomar un nodo inicial, un nodo final, y el Grafo, y retornar la distancia entre el nodo inicial y el final. Considere que moverse a través de una arista tiene coste 1.

P3. Union find

Una estructura de datos de conjuntos disjuntos es, como el nombre lo indica, una estructura que mantiene un conjunto de elementos particionados en conjuntos disjuntos. Union-find es un algoritmo que realiza dos operaciones sobre esta estructura: *unir* y *buscar*. Existen varias aplicaciones para esta estructura como por ejemplo, encontrar componentes conexas de un grafo o encontrar el *árbol recubridor mínimo*.

(a) Se pide que programe una clase Union-find, para ello se pide que implemente los siguientes métodos:

i. Un constructor `UnionFind(int n)` que tome la cantidad de nodos y cree un árbol para cada nodo.

ii. Un método `public int Buscar(int nodo)` que avanza por los padres hasta encontrar la raíz del árbol.

iii. Un método `public void Union(int x, int y)`, que busca las raíces de los árboles de `x` e `y`, y los une.

(b) La implementación anterior no es óptima pues los árboles que se van generando pueden quedar muy desbalanceados, para mejorar esto se pide que realice las siguientes mejoras a la estructura:

- Realizar una *Union por Rank*, que consiste en unir el árbol más pequeño a la raíz del árbol más grande. Para esto necesita modificar el método del constructor y el método de unión.
- Realizar *Compresión de caminos*, que consiste en que al buscar la raíz de un nodo, todos los nodos que estén en el camino pasen a tener la raíz directamente como padre.