CS1103

Programación Orientado a Objetos II

Simulacro de la Práctica Calificada #3 (PC3)

2023 - 1

Profesor: Rubén Rivas

Nombre:

Heap – 4 puntos

Desarrollar un functor llamada ***nearest_spheres*** que utilizando un heap, reciba una colección de esferas (**sphere**) cuyo radio (expresado ms) es ***r*** y peso específico (kg/m^3) es ***sw*** y devolver la primeras ***n*** esferas cuyo peso es el más cercano al peso promedio de todas las esferas.

```
// Crear functor
nearest_spheres<int, double> ns;
// Agregar esferas
ns.add_sphere(10, 2);   // #1
ns.add_sphere(5, 1);    // #2
ns.add_sphere(8, 2);    // #3
ns.add_sphere(20, 1);   // #4
ns.add_sphere(15, 2);   // #5
// Obteniendo esferas en orden: #2, #3, #1
vector<spheres<int, double>> spheres = ns(3);
```

Hash Table – 4 puntos

Utilizando los conceptos de hash table, crear una template de funciones denominado ***remove_characters*** que permita recibir un texto ***t*** y que permita mantener **k** veces un carácter y remover el resto.

```
// Texto
string text = "esta es una prueba para remover subtextos";
// Ejecutar remover caracteres
auto result = remove_characters(text, 2, {'e', 'a', 'r'});
// Mostrando resultado
cout << result;   // "esta es una prub rmov subtxtos"
```

Árboles - 4 puntos

Dado un árbol binario de búsqueda, se denomina arboles sesgados (skew trees) a aquellos árboles donde todos sus nodos solo tienen un hijo excepto los nodos hojas que no tienen hijos. Al árbol desarrollado en clase agregarle el método **is_skew()** que permita determinar si el árbol es sesgado o no.

```
// Creando arbol
binary_search_tree<string, int> bst;
bst.insert({"A", 10});
bst.insert({"B", 40});
bst.insert({"C", 30});
bst.insert({"D", 38});
bst.insert({"E", 32});
// Verificando si es árbol sesgado
cout << boolalpha << bst.is_skew() << endl; // true

// Creando arbol
binary_search_tree<string, int> bst2;
bst2.insert({"A", 20});
bst2.insert({"B", 40});
bst2.insert({"C", 10});
bst2.insert({"D", 38});
bst2.insert({"E", 32});
// Verificando si es árbol sesgado
cout << boolalpha << bst2.is_skew() << endl; // false
```

Grafos - 4 puntos

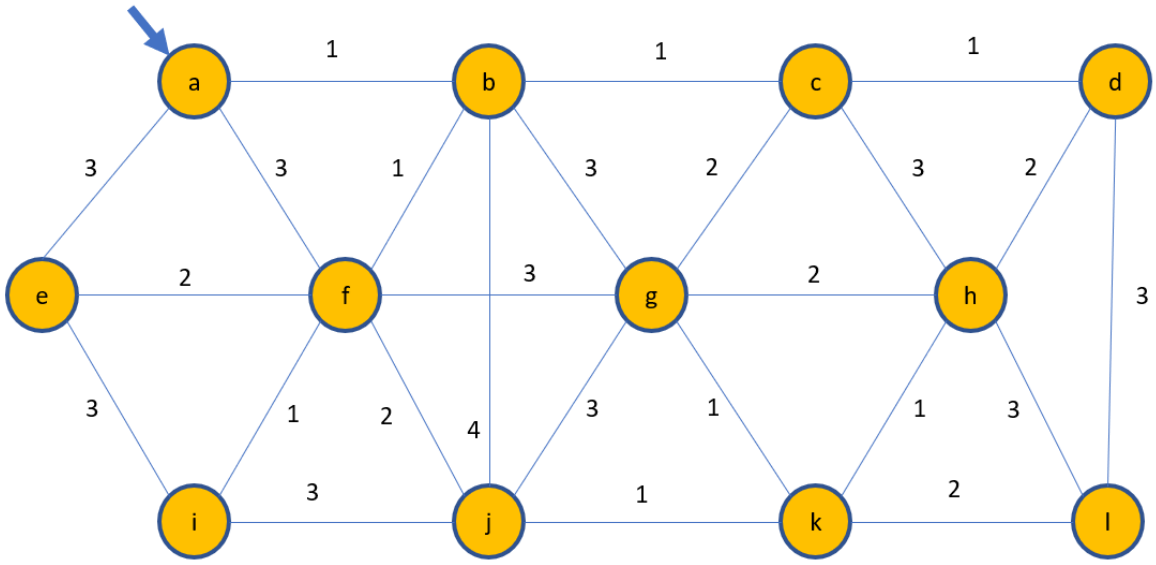
Desarrollar un grafo no dirigido y un método que determine si el grafo es conectado **is_connected()**

```
// Creando grafo
grafo<string, int, int> g;
// Agregando vertices
g.add_vertex({"A", 30});
g.add_vertex({"B", 40});
g.add_vertex({"C", 50});
g.add_vertex({"D", 60});
// Agregando aristas
g.add_edge("A", "C", 20);
g.add_edge("A", "B", 10);
g.add_edge("B", "D", 30);
// Verificar si esta conectado
cout << boolalpha << g.is_connected() << endl; // true.

// Creando grafo
grafo<string, int, int> g2;
// Agregando vertices
g2.add_vertex({"A", 30});
g2.add_vertex({"B", 40});
g2.add_vertex({"C", 50});
g2.add_vertex({"D", 60});
// Agregando aristas
g2.add_edge("A", "B", 20);
g2.add_edge("C", "D", 30);
// Verificar si esta conectado
cout << boolalpha << g2.is_connected() << endl; // false.
```

Grafos - 6 puntos

Utilizando el algoritmo de Prim e iniciando en el vértice **a** determinar el primer árbol expandido mínimo que se forme de 10 aristas.
Adicionalmente realizar el recorrido a lo largo (BFS) y el recorrido a profundidad (DFS) utilizando los algoritmos.
Sustentar claramente sus respuestas con el procedimiento de cálculo.



Barranco, 10 de julio del 2023