GRADO EN INTELIGENCIA ARTIFICIAL (MOSTOLES)

2361 - ESTRUCTURAS DE DATOS II - TARDE A - 2Q

Evaluación > Prueba Ordinaria

Comenzado el	viernes, 24 de mayo de 2024, 19:05
Estado	Finalizado
Finalizado en	viernes, 24 de mayo de 2024, 20:50
Tiempo empleado	1 hora 45 minutos
Calificación	**5,50** de 8,50 (**64,71**%)

Pregunta 1

Finalizado

Se puntúa 0,75 sobre 1,50

[1,5 puntos] Responde al **apartado f) de la pregunta larga** del enunciado en PDF.

```
void ampliar_horas(std :: list<std :: string > temas, std :: li
     std :: vector<std :: string> temasV;
     std :: vector<int> horasV;
     for(auto& it : temas){
          temasV.push_back(it);
     }
     for( auto& it: horas){
          horasV.push_back(it);
     }
     for(auto i = 0; i < temasV.size(); ++i){
          grafo[temasV[i]].getNode() -> h_invertidas += horasV[
     }

}
```

Comentario:
Si vas a usar std::vector, por qué no recibes std::vector directamente (es una lista también).

El acceso al grafo no es correcto.

SOLUCIÓN:

Hay otras alternativas al bucle presentado.

```
        void addInvested(std::vector<typename T::ID> nodes, std::vector<int>
hours) {
                for(auto nit = nodes.begin(),
                    auto hit = hours.begin();
                        nit != nodes.end()
                        && hit != hours.end();
                        ++nit, ++hit) {

                        auto& t = getNode(*nit);
                        t.invertidas += *hit;
                }
        }
```

Pregunta 2

Finalizado

Se puntúa 0,25 sobre 0,25

[0,25 puntos] Responde al **apartado a) de la pregunta larga** del enunciado en PDF.

```
struct asignaturas{
    std :: string nombre;
    int h_completar;
    int h_invertidas;
    std :: string fecha;

    using ID = std :: string;

    ID getId(){return nombre;}

    auto operator<=>(const asignaturas& rhs) const = default;
}

template <typename T>
class asignaturasGrafo{
public:
     T getNode(ID nodeId){return reverse[nodeId]}
private:
     std :: unordered_map<typename T :: ID, unordered_set<typer
     std :: unordered_map<typename T :: ID, T> reverse;
}
```

Comentario:

SOLUCIÓN:

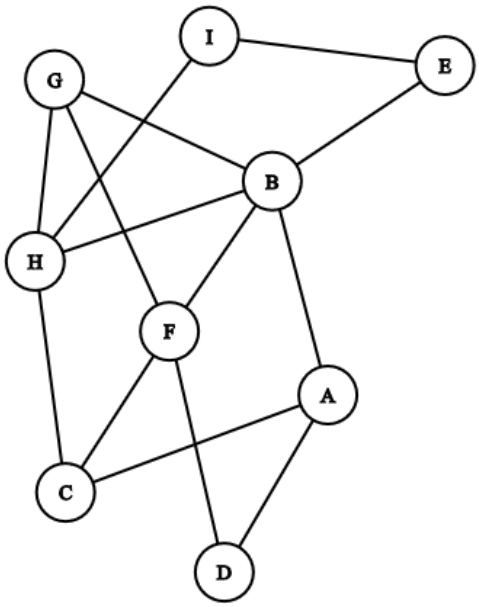
```
struct {
   std::string nombre;
   std::string fecha;
   int horasEstimadas;
   int horasInvertidas;

   using ID = std::string;

   ID getID() {return nombre;}
}
template <typename T>
class grafo {
...
private:
    std::unordered_map<typename T::ID, std::unordered_set<typename T::ID>>
nodes;
    std::unordered_map<typename T::ID, T> reverse;
};
```

Pregunta 3

Correcta

Se puntúa 1,00 sobre 1,00

[1 punto] Dado el grafo de la figura, indica los recorridos en profundidad y en anchura **visto en clase**, partiendo del **nodo H**.



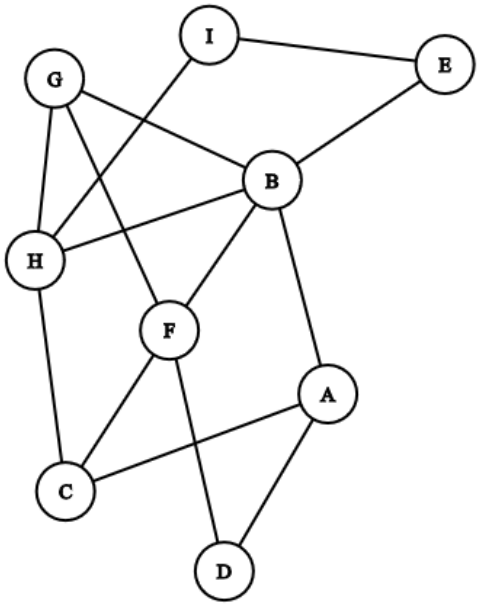
Recorrido en Profundidad: H ✔ I ✔ E ✔ B ✔ G ✔ F ✔ D ✔ A ✔ C ✔

Recorrido en Anchura: H ✔ B ✔ C ✔ G ✔ I ✔ A ✔ E ✔ F ✔ D ✔

G I A D C B E H F

Respuesta correcta

La respuesta correcta es:
[1 punto] Dado el grafo de la figura, indica los recorridos en profundidad y en anchura **visto en clase**, partiendo del **nodo H**.



Recorrido en Profundidad: [H][I][E][B][G][F][D][A][C]

Recorrido en Anchura: [H][B][C][G][I][A][E][F][D]

Pregunta 4

Finalizado

Se puntúa 0,50 sobre 1,50

[1,5 puntos] Responde al **apartado e) de la pregunta larga** del enunciado en PDF.

```
int relacion(T tema){
    int hi = tema->h_invertidas;
    int he = tema->h_estimadas;
    for(auto& temas : grafo[tema.getId()]){
         hi += temas.getNode -> h_invertidas;
         he += temas.getNode->h_estimadas

    }
    if(he - hi < 0){std :: cout << "Se han invertido mas horas
    else{std :: cout <<"Se ha estudiado mas rapido de lo espera
    return he - hi;
}
```

Comentario:
Tema no es un puntero ni shared_ptr, las → deberían ser .

Falta el parámetro de la función getNode.

Un tema puede depender de otros temas no directamente.

SOLUCIÓN:

```
	void hours(typename T::ID node) {
		int e, i;
		dependenthours(node, e, i);
		auto t = getNode(node);
		e += t.estimadas;
		i += t.invertidas;
		if (i > e * 2) std::cout << "Tema dificil.\n";
		else if (e > i * 2) std::cout << "Tema fácil.\n";
		else std::cout << "Tema normal.\n";
	}

	void dependenthours(typename T::ID node, int& e, int& i) {
		for(const auto& d: nodes[node]) {
			dependenthours(d, e, i);
			auto t = getNode(d);
			e += t.estimadas;
			i += t.invertidas;
		}
	}
```

Pregunta 5

Finalizado

Se puntúa 0,50 sobre 0,50

[0,5 puntos] Responde al **apartado d) de la pregunta larga** del enunciado en PDF.

```
std :: list<T> dependenDe (T tema){
    std :: list<T> l;
    for(auto& temas : grafo[tema.getId()]){
         l.push(temas.getNode());
    }
    return l;
}
```

Comentario:

SOLUCIÓN:

```
   const std::unordered_set<typename T::ID>& getEdges(typename T::ID node)
const {
       return nodes[node];
   }
```

Pregunta 6

Finalizado

Se puntúa 0,75 sobre 1,50

[1,5 puntos] Responde a la **pregunta corta** del enunciado en PDF.

```
bool equilibrado(Node<T>* node ){
     if(node){
          int NHI = NH(node->LeftChild);
          int NHD = NH(node->RightChild);
          if(NHI < 2*NHD && NHI > NHD/2){
              equilibrado(node->LeftChild, b);
              equilibrado(node->RightChild, b);
          }else{return false}
     }
     return true;
}

int NH(Node<T>* node, int hojas){
     if node{
          if(esHoja(node)){
              hojas += 1;
          }
          if(node-> LeftChild){
              return NH(node->LeftChild, hojas);
          }
          if(node-> RightChild){
               return NH(node->RightChild, hojas);
          }
          return hojas;
     }
}
```

Comentario:
La condición de equilibrado no comprueba correctamente los subárboles.

Recorres el árbol demasiadas veces.

SOLUCIÓN:

```
int LeaftBalancedNode(Tree t, Node* n) {
    if (n == nullptr) return 0;

    auto l = t.getLeftChild(n);
    auto r = t.getRightChild(n);
    auto nhl = LeaftBalancedNode(t, l);
    auto nhr = LeaftBalancedNode(t, r);

    if (nhl == -1 || nhr == -1) return -1;  // unbalanced subtree
    if (nhl == 0 && nhr == 0) return 1;  // leaf Node
    if (nhl > nhr * 2 || nhr > nhl * 2) return -1;

    return nhl + nhr;
}
bool LeaftBalanced(Tree t) {
    return LeaftBalancedNode(Tree t, t.getRoot()) != -1;
}
```

Pregunta 7

Finalizado

Se puntúa 0,00 sobre 0,25

[0,25 puntos] Responde al **apartado c) de la pregunta larga** del enunciado en PDF.

```
void dependencia(T tema1, T tema2){
    grafo[tema1.getId()][tema2.getId()];
}
```

Comentario:
Lo tienes que igualar a algo.

SOLUCIÓN:

```
void setEdge(typename T::ID sourceNode, typename T::ID targetNode) {
    nodes[sourceNode].insert(targetNode);
}
```

Pregunta 8

Correcta

Se puntúa 1,50 sobre 1,50

[1,5 puntos] Dada la secuencia de elementos [6, 5, 11, 3, 4, 12, 9, 7, 10, 8], se pide construir dos árboles binarios de búsqueda, un AVL y un Rojo Negro. Para cada tipo de árbol, contesta con el recorrido pedido:

Preorden AVL: 6 ✔ 4 ✔ 3 ✔ 5 ✔ 9 ✔ 7 ✔ 8 ✔ 11 ✔ 10 ✔ 12 ✔

Preorden RN: 9 ✔ 6 ✔ 4 ✔ 3 ✔ 5 ✔ 7 ✔ 8 ✔ 11 ✔ 10 ✔ 12 ✔

3 6 11 10 14 16 15 8 1 4 9 2 12 5 13 7

Respuesta correcta

La respuesta correcta es:
[1,5 puntos] Dada la secuencia de elementos [6, 5, 11, 3, 4, 12, 9, 7, 10, 8], se pide construir dos árboles binarios de búsqueda, un AVL y un Rojo Negro. Para cada tipo de árbol, contesta con el recorrido pedido:

Preorden AVL: [6] [4] [3] [5] [9] [7] [8] [11] [10] [12]

Preorden RN: [9] [6] [4] [3] [5] [7] [8] [11] [10] [12]

Pregunta 9

Finalizado

Se puntúa 0,25 sobre 0,50

[0,5 puntos] Responde al **apartado b) de la pregunta larga** del enunciado en PDF.

```
void insertar(std :: string nombre, std :: string fecha, int h_
    bool estanTemas = true;
    for(auto& it : temas){
        if(!grafo.contains(it.getId())){
             estanTemas = false;
             std :: cout << "No esta en el grafo"
        }
     }
     if(estanTemas){
         T node;
         node-> nombre = nombre;
         node->fecha = fecha;
         node->h_estimadas = h_estimadas;
         node->h_invertidas = 0;
         grafo[node.getId()] = {};
         reverse[node.getId()] = node;
    }
}
```

Comentario:
node no es un puntero, deberías usar . en lugar de →

Compruebas si existen los temas, pero no creas las dependencias.

SOLUCIÓN:

```
    void addNode(std::string name, int estimated, std::vector<std::string>
dependencies) {
        Tema t{name, estimated, 0};
        nodes[name] = {}
        reverse[name] = t;

        for (const auto& d: dependencies) {
            if (nodes.contains(d))
                nodes[name].insert(d);
            else
                std::cout << "No se puede insertar la dependencia: " << d <<
"\n";
        }
    }
```

Actividad previa

Prueba Ordinaria - Extra

Ir a...

Siguiente actividad

Prueba Ordinaria - Modelo A