ImportanteExamen.pdf

Anónimo

Estructuras de Datos no Lineales

2º Grado en Ingeniería Informática

Escuela Superior de Ingeniería
Universidad de Cádiz

FUNCIONES IMPORTANTES PARA EXAMEN

/***ABIN******/**

```
template <typename T>
int numeroNodos(const Abin<T>& A){
   return numeroNodosRec(A.raiz(), A);
}

template <typename T>
int numeroNodosRec(typename Abin<T>::nodo n , const Abin<T>& A){
   if ( n == Abin<T>::NODO_NULO) return 0;
   else  return 1 + numeroNodosRec(A.hizq(n),A) + numeroNodosRec(A.hder(n),A);
}


template <typename T>
int alturaArbol ( const Abin<T>& A){
   return alturaArbolRec ( A.raiz(),A);
}
// nodo tiene que pertenecer al arbol
template <typename T>
int alturaArbolRec(typename Abin<T>::nodo n , const Abin<T>& A){
   if ( n == Abin <T>::NODO_NULO) return -1;
   else return 1 + max( alturaArbolRec (A.hizq(n),A), alturaArbolRec( A.hder(n),A) );
}

template <typename T>
int profundidaNodo ( typename Abin<T>::nodo n , const Abin<T>& A){
   if (n == Abin <T>:: NODO_NULO) return -1;
   else return 1 + profundidadNodo ( A.padre(n),A);
}


template <typename T>
int desequilibrioArbin( const Abin <T>& A){
   return desequilibrioArbinRec (A.raiz(),A);
}

template <typename T>
int desequilibrioArbinRec (typename Abin<T>:: nodo n , const Abin <T>& A){
   if ( n == Abin<T>::NODO_NULO) return 0;
   else return max ( abs(alturaArbolRec(A.hizq(n),A)-alturaArbolRec(A.hder(n),A)
),max(desequilibrioArbinRec(A.hizq(n),A),desequilibrioArbinRec(A.hder(n),A)));
```

```
}


template <typename T>
bool psudoCompleto( const Abin<T>& A){
   return psudoCompletoRec( A.raiz(),A);
}

template <typename T>
bool pseudoCompletoRec( typename Abin<T>:: nodo n , const Abin <T>& A){

   if (n == Abin<T>::NODO_NULO) return true;
   else {
      if (alturaArbolRec(n , A) == 1 && ((A.hizq(n)==Abin<T>::NODO_NULO&&
A.hder(n)!=Abin<T>::NODO_NULO)||(A.hizq(n)!=Abin<T>::NODO_NULO&&
A.hder(n)==Abin<T>::NODO_NULO))
      {
         return false;
      }
      else {
         int alturaHizq, alturaDer;
         alturaHizq= alturaArbolRec(A.hizq(n),A);
         alturaDer = alturaArbolRec( A.hder(n),A);
         if ( alturaHizq > alturaDer) return pseudoCompletoRec(A.hizq(n),A);
         else if (alturaDer> alturaHizq) return pseudoCompletoRec(A.hder(n),A);
         else return pseudoCompletoRec(A.hizq(n),A) && pseudoCompletoRec(A.hder(n),A);
      }
   }
}


template <typename T>
int desequilibrio (const Abin<T>& A){
   return desequilibrioRec(A.raiz(),A);
}

template <typename T>
int desequilibrioRec ( typename Abin<T>::nodo n, const Abin<T>& A ){
   if ( n == Abin<T>::NODO_NULO) return 0;
   else max(abs(alturaRec(A.hizq(n),A)-
alturaRec(A.hder(n),A)),max(desequilibrioRec(A.hizq(n),A),desequilibrioRec(A.hder(n),A)));
}

template <typename T>
```

```
int alturaRec( typename Abin <T>:: nodo n , const Abin<T>& A){
   if (n == Abin<T>:: NODO_NULO) return -1;
   else return 1 + max( alturaRec( A.hizq(n),A), alturaRec (A.hder(n),A));
}

template <typename T>
bool pseudocompleto ( Abin<T>& A){
   return pseudoCompletoRec( A.raiz(), A);
}

template <typename T >
bool pseudocompletoRec ( typename Abin<T>:: nodo n , const Abin <T>& A){
   int alturaHizq, alturaHder;

   if ( n == Abin<T>:: NODO_NULO) return true;
   else {
      if (alturaRec(n,A)==1 && ((A.hizq(n)==Abin<T>::NODO_NULO && A.hder(n)!=
Abin<T>::NODO_NULO )|| (A.hizq(n)!=Abin<T>::NODO_NULO && A.hder(n)==
Abin<T>::NODO_NULO )) return false;
            else {
         alturaHizq= alturaRec(A.hizq(n),A);
         alturaHder= alturaRec(A.hder(n),A);
         if ( alturaHizq >alturaHder ) return pseudocompletoRec ( A.hizq(n),A);
         else if (alturaHder > alturaHizq ) return pseudocompletoRec (A.hder(n),A);
         else return pseudocompletoRec ( A.hizq(n),A) && pseudocompletoRec
(A.hder(n),A);
      }
   }
}


// contar nodos con tres nietos
template <typename T>
int numeroNodosTresNietos( const Abin<T>& A ){
   if ( alturaRec(A.raiz(),A) < 2 ) return 0;
   else {
      return numeroNodosTresNietos(A.raiz(),A);
   }
}

template <typename T>
int numeroNodosTreNietosRec ( typename Abin<T>::nodo n , const Abin<T>& A){
   int numHijos=0;
   if ( alturaRec(n,A) < 2) return 0;
```


```
   else {
      if (A.hizq(hizq(n)) != Abin<T>::NODO_NULO) numHijos++;
      if (A.hizq(hder(n)) != Abin<T>::NODO_NULO) numHijos++;
      if (A.hder(hizq(n)) != Abin<T>::NODO_NULO) numHijos++;
      if (A.hder(hder(n)) != Abin<T>::NODO_NULO) numHijos++;
      if ( numHijos == 3) return 1 + numeroNodosTresNietosRec( A.hizq(n),A) +
numeroNodosTreNietosRec (A.hder(n),A);
      else return 0 + numeroNodosTresNietosRec( A.hizq(n),A) +
numeroNodosTreNietosRec (A.hder(n),A);
   }
}

/****** AGEN ******/
// Altura maxima de un nodo
template <typename T>
int altura(typename Agen<T>::nodo n, const Agen<T>& A){
   int maximoActual= -1;
   if (n==Agen<T>::NODO_NULO){ return -1;}
   else {
      typename Agen<T>::nodo aux= A.hijoIzq(n);
      while (aux != Agen<T>::NODO_NULO){
         maximoActual= max( maximoActual ,altura(aux,A) );
         aux= A.hermDrcho(aux);
      }
      return 1 + maximoActual;
   }

}

//Altura minima de un nodo
template <typename T>
int alturaMin( typename Agen<T>::nodo n, const Agen<T>& A){
   if (n != Agen<T>::NODO_NULO) {
      typename Agen<T>::nodo aux = A.hijoIzqdo(n);
      int minimoActual = altura(aux, A);
      while (aux != Agen<T>::NODO_NULO) {
         minimoActual = min(minimoActual, altura(aux, A));
         aux = A.hermDrcho(aux);
      }
      return 1 + minimoActual;
   }
   else return -1;
}
```

/****** GRAFOS ********/**

```
vector<tCoste> Dijkstra ( const GrafoP<tCoste>& G, typename GrafoP<tCoste>::vertice
origen , vector<typename GrafoP<tCoste>::vertice> P);
vector<tCoste> DijkstraInverso ( const GrafoP<tCoste>& G, typename
GrafoP<tCoste>::vertice destino , vector<typename GrafoP<tCoste>::vertice> P);
matriz<tCoste> Floyd ( const GrafoP<tCoste>& G,  matriz<typename
GrafoP<tCoste>::vertice> P);
matriz<tCoste> FloydMaximo ( const GrafoP<tCoste>& G,  matriz<typename
GrafoP<tCoste>::vertice> P);
matriz<bool> Warshall ( const Grafo& G);
GrafoP<tCoste> kruskal ( const GrafoP<tCoste>& G);
GrafoP<tCoste> kruskalMax (const GrafoP<tCoste>& G);
GrafoP<tCoste> Prim (const GrafoP<tCoste>& G);
```