Algoritmos
y
Estructuras de Datos

Cursada 2011

Grafos

Estructura:

- Grafo
- Vértice
- Arista

Aplicaciones

- Mapas
- Materias de la facultad
- Redes de computadoras
- Muchos etc….

Grafos

Operaciones sobre grafo

- public void agregarVertice(Vertice<T> v);
- public void eliminarVertice(Vertice<T> v);
- public void conectar(Vertice<T> origen, Vertice<T> destino);
- public void conectar(Vertice<T> origen, Vertice<T> destino, int peso);
- public void desConectar(Vertice<T> origen, Vertice<T> destino);
- public boolean esAdyacente(Vertice<T> origen, Vertice<T> destino);
- public boolean esVacio();
- public ListaGenerica<Vertice<T>> listaDeVertices();
- public int peso(Vertice<T> origen, Vertice<T> destino);
- public ListaGenerica<Arista> listaDeAdyacentes(Vertice<T> v);
- public Vertice<T> vetice(int posicion);

Grafos

Operaciones sobre vértices

- public T dato();
- public int posicion();

Operaciones sobre aristas

- public Vertice<T> verticeDestino();
- public int peso();

Grafos: Implementación con Lista


```
public class GrafoImplListAdy<T> implements Grafo<T> {
    ListaGenerica<Vertice<T>> vertices = new ListaEnlazadaGenerica<Vertice<T>>();

     public void agregarVertice(Vertice<T> v) {
           //TODO: Comprobar que no exista el vértice
          ((VerticeImplListAdy<T>) v).setPosicion(vertices.tamanio());
           vertices.agregarAlFinal(v);
     }
     public void conectar(Vertice<T> origen, Vertice<T> destino, int peso) {
           ((VerticeImplListAdy<T>) origen).conectar(destino,peso);
      }
     public int peso(Vertice<T> origen, Vertice<T> destino) {
          return ((VerticeImplListAdy<T>) origen).peso(destino);
     }
           //…….resto de las operaciones
}
```

Grafos: Implementación con Lista


```
public class VerticeImplListAdy<T> implements Vertice<T> {
	private T dato;
	private int posicion;
	private ListaGenerica<Arista> adyacentes;
	public VerticeImplListAdy(T d) {
		dato = d;
		adyacentes = new ListaEnlazadaGenerica<Arista>();
	}
	public void conectar(Vertice<T> v, int peso) {
		 // TODO: Verificar que no exista
		Arista a = new AristaImpl(v, peso);
		adyacentes.agregar(a);
	 }
```

Grafos: Implementación con Lista


```
public int peso(Vertice<T> v) {
    Arista arista = obtenerArista(v);
    int ret = 0;
    if(arista != null){
          ret = arista.peso();
     }
   return ret;
}

private Arista obtenerArista(Vertice<T> v) {
      //Busca la arista cuyo destino es v
}

//……resto de las operaciones
}
```

Grafos: Implementación con Matriz


```
public class GrafoImplMatrizAdy<T> implements Grafo<T> {
    private int maxVertices;
    ListaGenerica<Vertice<T>> vertices;
    int[][] matrizAdy;

     public GrafoImplMatrizAdy(int maxVert){
          maxVertices = maxVert;
          vertices = new ListaEnlazadaGenerica<Vertice<T>>();
          matrizAdy = new int[maxVertices][maxVertices];
          for(int i=0; i<maxVertices; i++){
               for(int j=0; j<maxVertices; j++){
                    matrizAdy[i][j] = 0;
               }
          }
     }
```

Grafos: Implementación con Matriz


```
public void agregarVertice(Vertice<T> v) {
    //TODO: Verificar que no exista
    ((VerticeImplMatrizAdy<T>)v).setPosicion(vertices.tamanio());
    vertices.agregarAlFinal(v);
}
public void conectar(Vertice<T> origen, Vertice<T> destino, int peso) {
      matrizAdy[((VerticeImplMatrizAdy<T>)origen).
getPosicion()][((VerticeImplMatrizAdy<T>)destino).getPosicion()] = peso;
 }
 public int peso(Vertice<T> origen, Vertice<T> destino) {
      return matrizAdy[((VerticeImplMatrizAdy<T>)origen).
getPosicion()][((VerticeImplMatrizAdy<T>)destino).getPosicion()];
}
//……resto de las operaciones
}
```

```
public class VerticeImplMatrizAdy<T> implements Vertice<T> {
      private T dato;
      private int posicion;

      public VerticeImplMatrizAdy(T d) {
          dato = d;
       }
      //……resto de las operaciones
}
```