Esquema-resumido-curso.pdf

Anónimo

Estructuras de Datos y Algoritmos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Cosas que ahí que usar, sus implementaciones y sus métodos

- **Clase Grafo (Cuando el grafo es no dirigido aristas/2) hereda de Adyacente**
  - Lo implementan GrafoDirigido y GrafoNoDirigido
  - numVertices();
  - numAristas();
  - existeArista(int i, int j);
  - pesoArista(int i, int j);
  - insertarArista(int i, int j); o insertarArista(int i, int j, double p);
  - adyacentesDe(inti); ->ListaConPI
  - toString(){…}

- **Clase Adyacente**
  - getDestino();
  - getPeso();
  - toString();

- **ListaConPI**
  - ListaConPI<E> l = new LEGListaConPI<E>();
  - insertar(E e);
  - siguiente();
  - recuperar();
  - eliminar();
  - inicio();
  - fin();
  - esVacia();
  - talla();
  - esFin();

- **Pila (LIFO: Last In First Out)**
  - Pila<E> p = new ArrayPila<E>();
  - esVacia();
  - desapilar();
  - tope();

- **Cola (FIFO: First In First Out)**
  - Cola<E> p = new ArrayCola<E>();
  - encolar();
  - desencolar();
  - primero();
  - esVacia();

- **MonticuloBinario implementa ColaPrioridad**
  - MonticuloBinario<E extends Comparable<E>> m = new ColaPrioridad<E>();
  - Insertar(E e);
  - esVacia();
  - recuperarMin();
  - eliminarMin();
  - duplicarArray();

Montículo binario(MB)

- **Propiedad Estructural**
  - un Montículo Binario es un Árbol Binario(AB) **Completo**
  - se empieza por la pos 1 del array, NO por la 0

- **Propiedad de ordenación**
  - En el que usamos (Min-Heap) el dato que se sitúa en cada nodo es siempre menor o igual que los situados en sus hijos
  - Esto implica que en la raíz siempre estará el elemento mas pequeño

- **Propiedades por ser un AB completo**
  - La altura de un MB ES log2 talla
  - El coste de la búsqueda es, peor caso: O(log talla)

- **Propiedades por estar en un array flamisimo que empieza por 1**
  - elArray[1] es el nodo raíz
  - siendo elArray[i] el i-ésimo nodo:
  - su hijo izq. es elArray[2i], si 2i <= talla (si existe vamos)
  - su hijo der. es elArray[2i + 1], si 2i + 1 <= talla (si existe vamos)
  - su Padre es elArray [i/2], excepto para i = 2 (gran cerebro momento)

- **Propiedades por ser un AB Parcialmente Ordenado**
  - Cualquier subárbol o hoja del heap es un heap
  - Cada camino del Heap esta ordenado ascendientemente (dudo que esto sirva pal examen)

- **Método hundir** *(T5 p. 22)*

```
protected void hundir(int pos) {
    posActual = pos;
    E aHundir = elArray[posActual];
    int hijo = posActual * 2; boolean esHeap = false;
    while (hijo <= talla && !esHeap) {
        if (hijo < talla
            && elArray[hijo + 1].compareTo(elArray[hijo]) < 0) {
            hijo++;
        }
        if (elArray[hijo].compareTo(aHundir) < 0) {
            elArray[posActual] = elArray[hijo];
            posActual = hijo;  hijo = posActual * 2;
        }
        else { esHeap = true; }
    }
    elArray[posActual] = aHundir;
}
```

- **Reflotar** (T5 p. 19 dentro del método insertar)

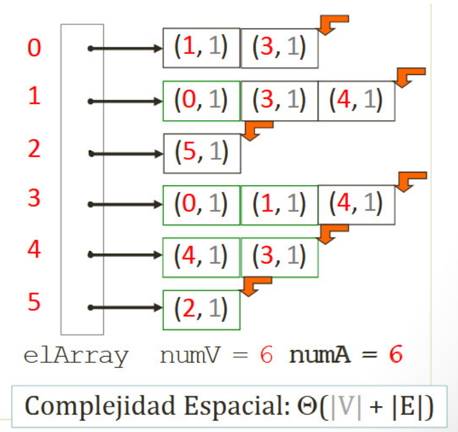
```
while (posIns > 1
       && e.compareTo(elArray[posIns / 2]) < 0) {

      elArray[posIns] = elArray[posIns / 2];
      posIns = posIns / 2;
}
```

Grafos

- **Representaciones.**

Lista de Adyacencias

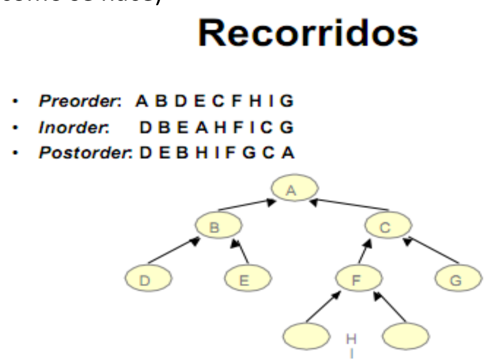


Matriz de Adyacencias

	V1	V2	V3	V4	V5
V1	0	1	0	0	0
V2	1	0	1	0	1
V3	0	0	0	0	1
V4	1	0	0	0	0
V5	1	0	0	1	0

Complejidad Espacial: $\Theta(|V|^2)$

- **Recorridos** (nunca supe como se hace)



- **DFS (T6 p.22 para ctrl. + c) ->** Hasta el fondo

```
protected int[] visitados; // atributo "auxiliar"
protected int ordenVisita; // atributo "auxiliar"
public int[] toArrayDFS() {
    int[] res = new int[numVertices()]; ordenVisita = 0;
    visitados = new int[numVertices()];
    for (int v = 0; v < numVertices(); v++) {
        if (visitados[v] == 0) { toArrayDFS(v, res); }
    }
    return res;
}
protected void toArrayDFS(int v, int[] res) {
    visitados[v] = 1;
    res[ordenVisita] = v; ordenVisita++;
    ListaConPI<Adyacente> l = adyacentesDe(v);
    for (l.inicio(); !l.esFin(); l.siguiente()) {
        int w = l.recuperar().getDestino();
        if (visitados[w] == 0) { toArrayDFS(w, res); }
    }
}
```

- **BFS (T6 p. 32 para ctrl + c) ->** A to lo ancho que dé

```
protected int[] visitados; protected int ordenVisita; // atributos "auxiliares"
protected Cola<Integer> // atributo "auxiliar"
public int[] toArrayBFS() {
    int[] res = new int[numVertices()]; visitados = new int[numVertices()];
    ordenVisita = 0; q = new ArrayCola<Integer>();
    for (int v = 0; v < numVertices(); v++) { if (visitados[v] == 0) { toArrayBFS(v, res); } }
    return res;
}
protected void toArrayBFS(int v, int[] res) {
    visitados[v] = 1; res[ordenVisita++] = v; q.encolar(new Integer(v));
    while (!q.esVacia()) {
        int u = q.desencolar().intValue();
        ListaConPI<Adyacente> l = adyacentesDe(u);
        for (l.inicio(); !l.esFin(); l.siguiente()) {
            int w = l.recuperar().getDestino();
            if (visitados[w] == 0) {
                visitados[w] = 1; res[ordenVisita] = w; ordenVisita++;
                q.encolar(new Integer(w));
            }
        }
    }
}
```

- **Dijkstra, Kruskal y Spanning Tree** (apenas se diferenciarlos no te voy a mentir mi pana)
  - Dijkstra: https://www.youtube.com/watch?v=eLFEIxDEphA
  - Kruskal: https://www.youtube.com/watch?v=YHzllcQpEdA
  - El Spanning Tree DFS o BFS son igual que hacer un DFS o BFS pero desde el vértice 0 se debe llegar todos los vértices, si no se puede devuelve null. (Código en la solución del ejercicio 10 del T6 p. 38)

- **Orden Topológico**
  - Es el inverso del DFS Post-Orden del grafo (un poco movida no te voy a mentir)
  - DFS Post-Orden: https://www.youtube.com/watch?v=aj-HunGHI-Y
  - Código Orden Topológico T6 p. 62 y p. 66 con test de aciclicidad

- **Test Aciclicidad**
  - Código T6 p. 65

ForestUFSet(EZ M8)

- **Union(x, y)**

```
/** PRECONDICIÓN: claseI != claseJ
 *                AND claseI = find(i) AND claseJ = find(j)
 *  Une las clases de equivalencia -subconjuntos disjuntos
 *  de identificadores claseI y claseJ
 */
public void union(int claseI, int claseJ) {
    if (elArray[claseI] == elArray[claseJ]) { // Mismo Rango
        elArray[claseI] = claseJ; // Colgar claseI de claseJ
        elArray[claseJ]--; // Incrementar Rango de claseJ
    }
    else { // Distinto Rango --> NO se incrementa su Rango
        if (elArray[claseI] < elArray[claseJ]) { // Negativos!!
            elArray[claseJ] = claseI; // Colgar claseJ de claseI
        }
        else {
            elArray[claseI] = claseJ; // Colgar claseI de claseJ
        }
    }
}
```

  - Si x < y (x es mas alto que y == mas negativo/menor) -> se cuelga y de x
  - Si x > y (x es menos alto que y) -> se cuelga x de y
  - Si x == y (misma altura) -> se cuelga x de y

- **Find(x)**

```
/** Devuelve el identificador de la clase de
 *  equivalencia a la que pertence el elemento i */
public int find(int i) {
    if (elArray[i] < 0) { return i; }
    return elArray[i] = find(elArray[i]);
}
```

  - Consideraciones:
    - En el array se actualizan los valores de los hijos solo, los de los padres no porque es asin y punto.
    - El find afecta también a los elementos que hay arriba y debajo de x, es decir si x tiene y arriba y z abajo, se ejecutara find(x) pero también find(y) (por la recursión) y find(z)

Costes

- Omega Ω -> Mínimo
- Theta Θ -> Promedio
- Omicron O-> Máximo

Representación (tipo)	coste promedio de `insertar`	coste promedio de `recuperarMin`	coste promedio de `eliminarMin`
Lista Enlazada Ordenada (Lineal)	lineal con `x`	constante	constante
Árbol Binario de Búsqueda (Jerárquica Enlazada)	logaritmo de `x`, (pero `O(x)`)	logaritmo de `x` (pero `O(x)`)	logaritmo de `x` (pero `O(x)`)
Montículo Binario (Jerárquica Contigua)	constante (pero `O(log x)`)	constante	logaritmo de `x`

- Santo grial de los costes alabado sea forocoches

Teoremas de coste:

Teorema 1: $f(x) = a \cdot f(x - c) + b$, con $b \geq 1$

- si $a=1$, $f(x) \in \Theta(x)$;
- si $a>1$, $f(x) \in \Theta(a^{x/c})$;

Teorema 3: $f(x) = a \cdot f(x/c) + b$, con $b \geq 1$

- si $a=1$, $f(x) \in \Theta(\log_c x)$;
- si $a>1$, $f(x) \in \Theta(x^{\log_c a})$;

Teorema 2: $f(x) = a \cdot f(x - c) + b \cdot x + d$, con b y $d \geq 1$

- si $a=1$, $f(x) \in \Theta(x^2)$;
- si $a>1$, $f(x) \in \Theta(a^{x/c})$;

Teorema 4: $f(x) = a \cdot f(x/c) + b \cdot x + d$, con b y $d \geq 1$

- si $a<c$, $f(x) \in \Theta(x)$;
- si $a=c$, $f(x) \in \Theta(x \cdot \log_c x)$;
- si $a>c$, $f(x) \in \Theta(x^{\log_c a})$;

Teoremas maestros:

Teorema para recurrencia divisora: la solución a la ecuación $T(x) = a \cdot T(x/b) + \Theta(x^k)$, con $a \geq 1$ y $b>1$ es:

- $T(x) \in O(x^{\log_b a})$ si $a>b^k$;
- $T(x) \in O(x^k \cdot \log x)$ si $a=b^k$;
- $T(x) \in O(x^k)$ si $a<b^k$;

Teorema para recurrencia sustractora: la solución a la ecuación $T(x) = a \cdot T(x-c) + \Theta(x^k)$ es:

- $T(x) \in \Theta(x^k)$ si $a<1$;
- $T(x) \in \Theta(x^{k+1})$ si $a=1$;
- $T(x) \in \Theta(a^{x/c})$ si $a>1$;