Estructures de dades i algorismes

Tema 5
Cua de Prioritat i Monticle Binari.
Ordenació segons un Monticle (Heap Sort)

Curs 2018-2019

Objectius

- Presentar una implementació eficient del model Cua de Prioritat
- Representació contigua (implícita) de les dades mitjançant un array
- Aquesta implementació rep el nom de Monticle Binari o Binary Heap
- Disseny del mètode genèric d'ordenació Heap Sort sobre la base d'aquesta implementació

Introducció

Introducció

La Cua de Prioritat és un model per a una col·lecció de dades en el qual les operacions característiques són aquelles que permeten accedir a la dada de major prioritat:

```
public interface ColaPrioridad<E extends Comparable<E>> {
  // Afegir x a la cua
  void inserir(E x);
  // SII !esBuida(): torna la dada amb major prioritat
  E recuperarMin();
  // SII !esBuida(): torna i elimina la dada amb major prioritat
  E eliminarMin();
  // Torna true si la cua esta buida
  boolean esBuida();
}
```

Monticle Binari

Monticle Binari: Propietats

Un heap es defineix per següents propietats:

- Propietat estructural: un heap és un arbre binari quasi-complet
  - Els arbres binaris quasi-complets permeten una representació implícita sobre array
  - La seua altura és, com a màxim, $\lfloor log_2 N \rfloor$
  - S'assegura llavors un cost logarítmic en el pitjor cas si els algorismes impliquen l'exploració d'una branca sencera
- Propietat d'ordre: En un **minHeap**, la dada d'un node és sempre menor o igual que el dels seus fills, mentre que en un **maxHeap** és sempre major o igual.

Tot subarbre d'un Heap és també un Heap

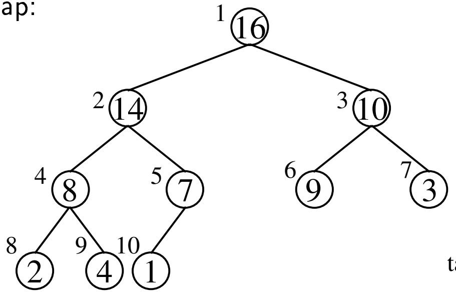
Monticle Binari: Representació

- S'emmagatzema en un array segons el seu recorregut per nivells
- El node arrel se situa en la posició 1 (la 0 es deixa lliure, la qual cosa facilita el càlcul dels fills d'un node)
- $A[1]$ és la arrel i donat un node $A[i]$,
  - si $2i \leq n$: $A[2i]$ fill esquerre
  - si $2i+1 \leq n$: $A[2i+1]$ fill dret
  - si $i \neq 1$: $A[\lfloor i/2]$ pare
- Propietat d'ordre maxHeap: $\boxed{A[pare(i)] \geq A[i], \quad 2 \leq i \leq n}$
- Propietat d'ordre minHeap: $\boxed{A[pare(i)] \leq A[i], \quad 2 \leq i \leq n}$
- Un Heap de n elements té una altura $\Theta(\log n)$

Monticle Binari: Representació

- Exemple maxHeap:

pare(i)=i/2
fill_esq(i)=2i
fill_dret(i)=2i+1



talla=10

	1	2	3	4	5	6	7	8	9	10	11	...	
A	16	14	10	8	7	9	3	2	4	1			

2. Monticle Binari: Exercici

Suposant que no hi han elements repetits i que estem parlant d'un maxHeap,

a) On estarà el màxim?

b) On estarà el mínim?

c) Qualsevol element d'una fulla serà menor que el dels nodes interns?

d) És un vector ordenat de forma decreixent un maxheap?

e) És la seqüència $\{23, 17, 14, 6, 13, 10, 1, 5, 7, 12\}$ un maxHeap?

La classe MonticuloBinario (MinHeap)

Monticle Binari

```
public class MonticuloBinario<E extends Comparable<E>> implements
    ColaPrioridad<E> {
  // Atributos
  protected static final int CAPACIDAD_INICIAL = 50;
  protected E elArray[];
  protected int talla;

  // Constructor de un heap minimal vacio
  @SuppressWarnings(``unchecked'')
  public MonticuloBinario() {
    talla = 0;
    elArray = (E[]) new Comparable[CAPACIDAD_INICIAL];
  }
```

Monticle Binari:inserir

- Pas 1: s'insereix el nou element en la primera posició disponible del vector: elArray[talla + 1]
- Pas 2: es reflota sobre els seus antecessors fins que no viole la propietat d'ordre

```
public void inserir(E x) {
  // hi ha espai a l'array per a la nova dada?
  if (talla == elArray.length - 1) duplicarArray();
  // hueco es la posicio on inserirem x
  int hueco = ++talla;
  // reflotem fins que no viole la propietat d'ordre
  while (hueco > 1 && x.compareTo(elArray[hueco/2]) < 0) {
    elArray[hueco] = elArray[hueco/2];
    hueco = hueco/2;
  }
  // ja tenim la posicio de la nova dada -> inserim
  elArray[hueco] = x;
}
```

Monticle Binari:inserir

- El cost és O(log2N) si l'element afegit és el nou mínim
- El cas més favorable és quan l'element a inserir és major que el seu pare (requereix per tant una única comparació)
- S'ha demostrat que, en mitjana, es requereixen 2.6 comparacions per a dur a terme una inserció (cost constant)
- Exercici: fer una traça d'inserir a partir d'un monticle buit els següents valors: 6, 4, 15, 2, 10, 11, 8, 1, 13, 7, 9, 12, 5, 3, 14

Monticle Binari:eliminar

- Pas 1: el mínim està en el node arrel. La dada del node arrel se substitueix llavors per l'últim element del Heap
- Pas 2: la nova arrel s'enfonsa a través dels seus fills fins a no violar la propietat d'ordre.

Monticle Binari:eliminar

El mètode enfonsar (heapify) enfonsa un node a través del heap fins que no es viole la propietat d'ordre:

```
private void enfonsar(int hueco) {
  E aux = elArray[hueco];
  int fill = hueco * 2;
  boolean esHeap = false;
  while (fill <= talla && !esHeap) {
    if (fill != talla &&
        elArray[fill+1].compareTo(elArray[fill]) < 0)
        fill++;                // triem el menor dels fills
    if (elArray[fill].compareTo(aux) < 0) {  // Afonem
      elArray[hueco] = elArray[fill];
      hueco = fill;
      fill = hueco*2;
    } else esHeap = true; // s'acompleix la prop. d'ordre
  }
  elArray[hueco] = aux;
}
```

Monticle Binari:eliminar

```
// SII !esBuida(): torna I elimina la dada de menor prioritat
public E eliminarMin() {
  E elMinim = recuperarMin();
  // Substituim l'arrel per l'lutim element
  elArray[1] = elArray[talla--];
  //S'afona la nova arrel fins que no viole la propietat d'ordre
  enfonsar(1);
  return elMinim;
}

// SII !esBuida(): torna la dada de menor prioritat
public E recuperarMin() {
  return elArray[1];
}
```

Monticle Binari: Exercicis

- Exercici: fer una traça de eliminarMin sobre el Monticle Binari [0; 1; 4; 8; 2; 5; 6; 9; 15; 7; 12; 13]
- Exercici: escriure un mètode en la classe MonticuloBinario que obtinga el seu element màxim realitzant el mínim nombre de comparacions
- Exercici: dissenya una funció, eliminarMax, que elimine el màxim en un Monticle Binari Minimal

Monticle Binari: heapify

```
private void arreglarMonticulo() {
  for (int i = talla / 2; i > 0; i--)
    enfonsar(i);
}
```

- Restableix la propietat d'ordre a partir d'un Arbre Binari Quasi-Complet per a obtenir un Monticle Binari
- Es basa a enfonsar els nodes en ordre invers al recorregut per nivells
- Té una complexitat lineal Les fulles tenen una altura 0 I l'arrel $\lfloor log2N \rfloor$
- El cost d'afonar un node d'altura h es O(h)
- El cost de arreglar un monticle es O(n)

Monticle Binari: HeapSort

- El cost del HeapSort és O(N*log2N)
    - QuickSort té un cost O(N2) en el pitjor dels casos
    - MergeSort requereix un vector auxiliar
- Este algorisme d'ordenació es basa en les propietats dels Heaps
    - Primer pas: emmagatzemar tots els elements del vector a ordenar en un monticle (heap)
    - Segon pas: extraure l'element arrel del monticle (el mínim) en successives iteracions, obtenint el conjunt ordenat

Monticle Binari: HeapSort

La forma més eficient d'inserir els elements d'un vector en un Heap és mitjançant el mètode arreglarMonticulo:

```
@SuppressWarnings(``unchecked'')  // Constructor a partir d'un
    vector
public MonticuloBinario(E v[]) {
  talla = v.length;   // Copiem les dades del vector
  elArray = (E[]) new Comparable[talla+1];
  System.arraycopy(v, 0, elArray, 1, talla);
  arreglarMonticulo();    // Arreglem la propiedad d'ordre
}
```

- El cost d'este constructor és O(N), sent N la talla del vector

Monticle Binari: HeapSort

```
public class Ordenacio {
  public static <E extends Comparable<E>> void heapSort(E v[]) {
    // Creamos el heap a partir del vector
    MonticuloBinario<E> heap = new MonticuloBinario<E>(v);
    // Vamos extrayendo los datos del heap de forma ordenada
    for (int i = 0; i < v.length; i++)
      v[i] = heap.eliminarMin();
  }
}
```

- Cost HeapSort = cost constructor + N * cost d'eliminarMin
  $T_{heapSort}(N) \in O(N) + N * O(log2N) = O(N * log2N)$
- HeapSort pot modificar-se fàcilment per a ordenar sols els k primers elements del vector amb cost $O(N + k * log2N)$

Exercicis

- Fer una traça del mètode arreglarMonticulo sobre l'arbre binari complet [7, 3, 5, 9, 1, 8]
- Dissenya un mètode que comprove si una dada x donada està en un Monticle Binari Minimal i estudia el seu cost
- Dissenya un mètode que elimine la dada de la posició k d'un Monticle Binari Minimal i estudia el seu cost
- Implementar l'algorisme d'ordenació HeapSort de manera que no requerisca una estructura de dades addicional (com un Monticle Binari o un vector auxiliar)

Bibliografia

- Data structures, algorithms, and applications in Java, Sahni (capítulo 13)
- M.A. Weiss. "Estructuras de Datos en Java", Adisson-Wesley, 2000 (Apartados 1 – 5 del capítulo 20)
- Data Structures and Algorithms in Java (4th edition), Goodrich y Tamassia (capítulo 8)
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. "Introduction to Algorithms" (segunda edición). The MIT Press, 2007 (Capítulo 6)
- G. Brassard y P. Bratley . "Fundamentos de Algoritmia", Prentice Hall, 2001