ESERCIZI MAX-HEAP E MIN-HEAP

Guida completa con soluzioni per esame

DEFINIZIONI FONDAMENTALI

Proprietà Max-Heap

- Albero binario quasi completo rappresentato come array A[1..n]
- Parent(i) = ⌊i/2⌋, Left(i) = 2i, Right(i) = 2i+1
- A[Parent(i)] ≥ A[i] per ogni i > 1
- A[1] contiene il massimo

Proprietà Min-Heap

- A[Parent(i)] ≤ A[i] per ogni i > 1
- A[1] contiene il minimo

Operazioni base

- MaxHeapify(A, i): O(log n) - sistema nodo i
- BuildMaxHeap(A): O(n) - costruisce heap da array
- HeapSort(A): O(n log n) - ordina usando heap
- ExtractMax(A): O(log n) - rimuove e restituisce massimo
- Insert(A, key): O(log n) - inserisce nuovo elemento

CATEGORIA 1: OPERAZIONI FONDAMENTALI

ES 1.1 - MaxHeapify

Traccia: Dato A[i] che viola proprietà max-heap, ripristinarla

Soluzione:

```
MaxHeapify(A, i)
l = Left(i)  // = 2*i
r = Right(i)  // = 2*i+1
if l ≤ A.heapsize and A[l] ≥ A[i]
    max = l
else
    max = i
```

```
if r ≤ A.heapsize and A[r] ≥ A[max]
    max = r
if max ≠ i
    A[i] ↔ A[max]  // scambio
    MaxHeapify(A, max)
```

Complessità: T(n) = T(2n/3) + Θ(1) = O(log n) = O(h)
Correttezza: Scende verso foglie spostando elemento piccolo giù

ES 1.2 - BuildMaxHeap

Traccia: Costruire max-heap da array non ordinato

Soluzione:

```
BuildMaxHeap(A)
A.heapsize = A.length
for i = ⌊A.length/2⌋ downto 1
    MaxHeapify(A, i)
```

Complessità: O(n) (non O(n log n)!)
Spiegazione: Somma pesata per livelli dà O(n)

- Foglie (n/2): 0 operazioni
- Livello h-1: n/4 nodi × O(1)
- Livello h-2: n/8 nodi × O(2)
- ...
- Radice: 1 nodo × O(log n)
- Totale = O(n)

Invariante: All'iterazione i, ogni nodo j > i è radice di max-heap

---

ES 1.3 - HeapSort

Traccia: Ordinare array usando heap

Soluzione:

```
HeapSort(A)
BuildMaxHeap(A)
for i = A.length downto 2
    A[1] ↔ A[i]
```

```
4.     A.heapsize = A.heapsize - 1
5.     MaxHeapify(A, 1)
```

Complessità: O(n log n)

- BuildMaxHeap: O(n)
- n-1 chiamate a MaxHeapify: O(n log n)

Invariante: A[1..i] è max-heap, A[i+1..n] ordinato, A[1..i] ≤ A[i+1..n]

ES 1.4 - ExtractMax

Traccia: Estrarre e rimuovere massimo da max-heap

Soluzione:

```
ExtractMax(A)
1. if A.heapsize < 1
2.     error "underflow"
3. max = A[1]
4. A[1] = A[A.heapsize]
5. A.heapsize = A.heapsize - 1
6. MaxHeapify(A, 1)
7. return max
```

Complessità: O(log n)

ES 1.5 - Insert (IncreaseKey + AddNewKey)

Traccia: Inserire nuovo elemento in max-heap

Soluzione:

```
IncreaseKey(A, i, key)
1. if key < A[i]
2.     error "nuova chiave minore"
3. A[i] = key
4. while i > 1 and A[Parent(i)] < A[i]
5.     A[i] ↔ A[Parent(i)]
6.     i = Parent(i)

AddNewKey(A, key)
1. A.heapsize = A.heapsize + 1
```

```
2. A[A.heapsize] = −∞
3. IncreaseKey(A, A.heapsize, key)
```

Complessità: O(log n)
Meccanismo: Inserisce in fondo e fa "salire" verso radice

CATEGORIA 2: MIN-HEAP

ES 2.1 - MinHeapify

Soluzione:

```
MinHeapify(A, i)
1. l = 2*i
2. r = 2*i+1
3. min = i
4. if l ≤ A.heapsize and A[l] < A[min]
5.     min = l
6. if r ≤ A.heapsize and A[r] < A[min]
7.     min = r
8. if min ≠ i
9.     A[i] ↔ A[min]
10.    MinHeapify(A, min)
```

ES 2.2 - ExtractMin

Soluzione:

```
ExtractMin(A)
1. if A.heapsize < 1
2.     error "underflow"
3. min = A[1]
4. A[1] = A[A.heapsize]
5. A.heapsize = A.heapsize − 1
6. MinHeapify(A, 1)
7. return min
```

CATEGORIA 3: OPERAZIONI SPECIALI

ES 3.1 - Secondo minimo in Min-Heap

Traccia: Trovare secondo elemento più piccolo in min-heap

Soluzione:

```
sndmin(A)
if A.heapsize > 2
    return min(A[2], A[3])
else if A.heapsize = 2
    return A[2]
else
    error "meno di 2 elementi"
```

Complessità: O(1)
Spiegazione: Secondo minimo è sempre figlio della radice

ES 3.2 - Verificare se array è Max-Heap

Traccia: Dato array A[1..n], verificare se soddisfa proprietà max-heap

Soluzione ricorsiva:

```
IsMaxHeap(A, i, n)
l = 2*i
r = 2*i+1
if l ≤ n
    leftOk = A[i] ≥ A[l] and IsMaxHeap(A, l, n)
else
    leftOk = true
if r ≤ n
    rightOk = A[i] ≥ A[r] and IsMaxHeap(A, r, n)
else
    rightOk = true
return leftOk and rightOk
```

Complessità: T(n) = 2T(n/2) + c = O(n)

Soluzione iterativa:

```
IsMaxHeap(A, n)
i = 2
while i ≤ n and A[i] ≤ A[i/2]
    i = i + 1
return i = n + 1
```

ES 3.3 - Fusione di due Max-Heap (strutture linked)

Traccia: Fondere H1 (completo) e H2 (quasi completo) stessa altezza in O(log n)

Prerequisito:

```
numToParent(H, k)  // Trova parent di nodo in posizione k
if k = 1
    return nil
else
    B = getBitVector(k)  // rappresentazione binaria
    i = 1
    while B[i] = 0
        i = i + 1
    x = H.root
    for j = i+1 to B.length-1
        if B[j] = 0
            x = x.left
        else
            x = x.right
    return x
```

Soluzione Fusion:

```
Fusion(H1, H2)
p = numToParent(H2, H2.size)
if p = nil
    x = H2.root
    H2.root = nil
else
    if H2.size mod 2 = 0
        x = p.left
        p.left = nil
    else
        x = p.right
        p.right = nil
H2.size = H2.size - 1
// Crea nuovo heap con x come radice
H.root = x
H.size = H1.size + H2.size + 1
x.left = H1.root
x.right = H2.root
MaxHeapify(H, x)
return H
```

Complessità: O(log n)
Meccanismo: Usa ultima foglia H2 come nuova radice, poi ripristina proprietà

ES 3.4 - Intersezione di Min-Heap

Traccia: Dati A1, A2 min-heap capacità n, restituire A min-heap con intersezione

Soluzione:

```
intersect(A1, A2, n)
A = new array[1..n]
k = 0
while A1.heapsize > 0 and A2.heapsize > 0
    v1 = ExtractMin(A1)
    v2 = ExtractMin(A2)
    if v1 = v2
        k = k + 1
        A[k] = v1
    else if v1 < v2
        // Scarta v1, rimetti v2
        Insert(A2, v2)
    else
        // Scarta v2, rimetti v1
        Insert(A1, v1)
A.heapsize = k
return A
```

Complessità: O(n log n)
Correttezza: Elementi estratti in ordine crescente → array risultante già min-heap

---

CATEGORIA 4: SELECT CON HEAP

ES 4.1 - Trovare k-esimo elemento più piccolo

Traccia: Dato array A[1..n], trovare k-esimo elemento più piccolo

Soluzione:

```
Select(A, k)
BuildMaxHeap(A, k)  // heap con primi k elementi
for i = k+1 to n
    if A[i] < A[1]
        A[i] ↔ A[1]
        MaxHeapify(A, 1)
return A[1]
```

Complessità: O(k + n log k) = O(n log k)

Invariante:

- A[1..k] è max-heap
- A[k+1..i-1] ≥ A[1..k]

Correttezza: Al termine A[1..k] contiene k elementi più piccoli, A[1] è il k-esimo

---

CATEGORIA 5: IMPOSSIBILITÀ ALGORITMI LINEARI

ES 5.1 - Impossibilità elenco ordinato da Max-Heap in O(n)

Domanda: Esiste algoritmo O(n) per elencare elementi max-heap in ordine decrescente?

Risposta: NO

Dimostrazione:

1. Se esistesse algoritmo O(n) per elencare max-heap ordinato
2. E posso costruire max-heap in O(n) (BuildMaxHeap)
3. Allora potrei ordinare in O(n)
4. Ma abbiamo Ω(n log n) come limite inferiore per ordinamento
5. **Contraddizione** → algoritmo non può esistere

CONFRONTO MAX-HEAP vs MIN-HEAP

Aspetto	Max-Heap	Min-Heap
Radice	Massimo	Minimo
Proprietà	A[Parent(i)] ≥ A[i]	A[Parent(i)] ≤ A[i]
Extract	ExtractMax	ExtractMin
Heapify	MaxHeapify	MinHeapify
Uso tipico	HeapSort, code priorità max	Algoritmi greedy, Dijkstra

---

HEAP vs BST

Aspetto	Heap	BST
Struttura	Array (quasi completo)	Puntatori
Ordine	Parziale (parent-child)	Totale (InOrder)
Max/Min	O(1)	O(h)

Aspetto	Heap	BST
Search	O(n)	O(h)
Insert	O(log n)	O(h)
Delete	O(log n)	O(h)

TEMPLATE RISOLUZIONE ESERCIZI HEAP

Step 1: Identificare operazione

- Ripristino proprietà → MaxHeapify/MinHeapify
- Costruzione heap → BuildMaxHeap
- Estrazione → ExtractMax/Min
- Inserimento → IncreaseKey + AddNewKey

Step 2: Complessità tipiche

- Singolo Heapify: O(log n) = O(h)
- BuildHeap: O(n)
- HeapSort: O(n log n)
- Operazioni su heap costruito: O(log n)

Step 3: Invarianti chiave

- MaxHeapify: figli di i sono radici di max-heap
- BuildMaxHeap: nodi > i sono radici di max-heap
- HeapSort: A[1..i] heap, A[i+1..n] ordinato

CHECKLIST ESAME

✓ Parent(i) = ⌊i/2⌋, Left(i) = 2i, Right(i) = 2i+1
✓ BuildMaxHeap costa O(n), NON O(n log n)
✓ HeapSort costa O(n log n)
✓ ExtractMax/Min e Insert costano O(log n)
✓ Secondo min in min-heap: O(1) guardando figli radice
✓ Fusione heap linked: O(log n) usando ultima foglia
✓ Impossibile ordinare heap in O(n)

ERRORI COMUNI DA EVITARE

❌ Confondere O(n) di BuildHeap con O(n log n)
❌ Dimenticare A.heapsize ≠ A.length durante operazioni
❌ Non decrementare heapsize dopo ExtractMax
❌ Cercare elemento in heap come in BST (non è O(log n)!)
❌ Pensare che secondo minimo richieda più di O(1)
❌ Usare MaxHeapify su nodo che non ha sottoalberi heap