Design and Analysis of Computer Algorithms

11059005 蕭耕宏

Assignment 3

Due: May 12, 2025 (before class)

Please use A4-sized papers to write your answers for the homework sets.

1. Let H be an n-node min-heap that contains n distinct items. Given any k ≤ n, explain how to select the kth smallest item in H in O(k log k) time.

   Solution:

   Maintain a pripriority queue. First insert the root of H into the priority queue. Then, do the following k times:

   1. extract the minimum item from the priority queue and store it as the kth smallest item. (log k)
   2. insert the left child and right child of the extracted item into the priority queue if they exist. (2 log k)
   3. If the priority queue is empty, break the loop.
   4. If the kth smallest item is found, break the loop.
   5. Return the kth smallest item.

   Since extracting the minimum item from the priority queue takes O(log k) time and inserting an item into the priority queue takes O(log k) time, the total time complexity is O(k log k).

2. A d-heap is like a binary heap except that nodes have d children instead of just two. This will reduce the height of the underlying tree having n elements to $O(\log_d n)$. Please explain how to perform the following operations, like what we did in binary heaps and give the time complexity of each operation.

   (a) INSERT

   (b) MINIMUM

   (c) EXTRACT-MIN

   (d) DECREASE-KEY

   Besides, suppose we use an array as the fundamental data structure for d-heap. That is, we store a complete d-ary tree into an array. Please show how to MAKEHEAP in O(n) time on a set of n elements.

   Solution:

   - INSERT(S, x, k)
     1. increase the heap size by 1
     2. append (x, k) to the end of the array
     3. compare the key of x with its parent, if x < parent, swap x with its parent
     4. repeat step 3 until x is in the correct position

     time complexity is $O(\log_d n)$

   - MINIMUM(S)
     1. returns the root, $\Theta(1)$

     time complexity is $\Theta(1)$

   - EXTRACT-MIN(S)

1. output root
2. copy last element in array to the root
3. decrease the array size by 1
4. compare the root with its children, if root > child, swap root with the smallest child
5. repeat step 4 until root is in the correct position

each level requires $O(d)$ comparisons, and the height of the tree is $O(\log_d n)$, so the time complexity is $O(d \log_d n)$.

- DECREASE_KEY(S, x, k)
1. find x in the array
2. decrease the value of x to k
3. compare x with its parent, if x < parent, swap x with its parent
4. repeat step 3 until x is in the correct position

time complexity is $O(\log_d n)$

- MAKEHEAP(A) use bottom up approach

1. for i = n/2 to 1
2. for j = 1 to d
3. compare A[i] with A[j]
4. if A[i] > A[j], swap A[i] with A[j]
5. repeat step 3 and 4 until A[i] is in the correct position
6. return A

time complexity analysis:
- each level have d^i nodes
- inserting a node takes $\lfloor \log_d n \rfloor - i$ in the worst case
- each level takes $(\lfloor \log_d n \rfloor - i) \times d^i$ for insertion

T(n) = $\sum_{i=0}^{i=\lfloor \log_d n \rfloor} (\lfloor \log_d n \rfloor - i) \times d^i$

let $j = \lfloor \log_d n \rfloor - i$, then $i = \lfloor \log_d n \rfloor - j$

T(n) = $\sum_{j=0}^{j=\lfloor \log_d n \rfloor} j \times d^{\lfloor \log_d n \rfloor - j}$

T(n) = $d^{\lfloor \log_d n \rfloor} \times \sum_{j=0}^{j=\lfloor \log_d n \rfloor} j \times d^{-j}$

since $\sum_{j=1}^{\infty} j \times x^j = \frac{x}{(1-x)^2}$, we can use this to find the sum of the series.

T(n) = $d^{\lfloor \log_d n \rfloor} \times \frac{d^{-1}}{(1-d^{-1})^2}$

T(n) = O(n)

3. A sequence of n operations is performed on a data structure. The ith operation costs i if i is an exact power of 2, and 1 otherwise. Please determine the amortized cost per operation using

(a) aggregate analysis

(b) accounting method of analysis; and

(c) potential method of analysis.

Solution:

(a) Aggregate analysis:

- T(n) = $\left(\sum_{i=0}^{\lfloor \lg n \rfloor} 2^i\right) + n - \lfloor \lg n \rfloor - 1 = 2^{\lfloor \lg n \rfloor + 1} - 1 + n - \lfloor \lg n \rfloor + 1 \leq 2^{\lfloor \lg n \rfloor + 1} + n - \lfloor \lg n \rfloor \leq 2n + n - \lg n \leq 3n = \mathrm{O}(n)$
- The amortized cost per operation is 3

(b) Accounting method of analysis:

Operation Type	Actual Cost	Amortized Cost
power of 2 operation	1	3
not power of 2 operation	2^i	2

c_i: the actual cost of the i-th operation

$\hat{c_i}$: the amortized cost of the i-th operation

To show that for each operation, $\sum_{i=1}^{n} \hat{c_i} \geq \sum_{i=1}^{n} c_i$,

- for i-th operation where i is a power of 2, there will be enough credit stored by $2^{\lg i - 1} + 1, 2^{\lg i - 1} + 2, ..., i - 1$

for example, if i = 8,

total actual cost 1 + 2 + 1 + 4 + 1 + 1 + 1 + 8 = 19

amortized cost = 2 + 3 + 2 + 3 + 2 + 2 + 2 + 3 = 19

- for i-th operation where i is not a power of 2, 2 credit will be stored.

The amortized cost per operation is 3 for both cases to work.

(c) Potential method of analysis:

let c_i be the actual cost of the i-th operation, and $\hat{c_i}$ be the amortized cost of the i-th operation.

let $\Phi(i)$ be the potential of the i-th operation, and $\Phi(0) = 0$

$\hat{c_i} = c_i + \Phi(i) - \Phi(i-1)$

set $\hat{c_i} = 3$, and $\Phi(i) = \sum_{j=1}^{i} 3 - c_j$

$\Phi(i) - \Phi(i-1) = 3 - c_i$

To argue that $\Phi(i) \geq 0$, we need to show that $\sum_{j=1}^{i} c_j \leq 3i$.

when i = 2^k, $\sum_{j=1}^{i} c_j = \sum_{j=1}^{k} 2^j + 2^k - k \leq \sum_{j=1}^{k-1} +2 \times 2^k \leq 3 \times 2^k$

this holds for all i, so we can conclude that $\Phi(i) \geq 0$.

Thus, the amortized cost per operation is 3.