Analysis of Algorithms

BLG 335E

Project 2 Report

Çağla Mıdıklı 150200011
midikli20@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 11.12.2023

1. Implementation

1.0.1. Structures

I defined a structure named ’data’ to hold two types of data obtained from reading a file together. This structure is holding cities and their populations. To accommodate these structure entities, I have defined an array. However, due to its uncertain length, I have designated it as a vector.

1.0.2. Libraries

<iostream> : This library is used for receiving data from the terminal or printing something to the terminal.
<string> : when reading data from the document, we can read the words as whole entities.
<fstream> : I used it because this library can opening, closing, and reading files in C++.
<sstream> : It enables us to split the strings read from a file under certain conditions.
<vector> : I used it to prevent unnecessary memory occupation, i am storing the array as a vector.
<chrono> : I used it for calculating the duration between the start and end of a process using the chrono library.
<cmath> : I used it for can using the ceil function to round up a fractional number obtained from a division to the nearest integer.

1.0.3. Myswap

It is swapping two elements of a vector or array with each other by declaring a temporary variable.

1.0.4. max heapify

In this function, firstly, the indices of the left and right children of the variable sent as an index parameter are calculated. If at least one of the left or right children has an index greater than the current index, the function swaps the element with the largest child. This function ensures the ordered arrangement of items along that line by recursively calling itself.

1.0.5. dary max heapify

The function, given an index as a parameter, identifies the indices of its children. If at least one of the children has a population larger than the population at the given index, it swaps the element with the child that has the largest population.This function ensures the ordered arrangement of items along that line by recursively calling itself.

1.0.6. dary build max heap

In the dary max heapify function, the sorting operation obtained for a single branch is applied to all branches.Thus, a max d-ary heap is created.

1.0.7. dary extract max

It returns the root value(the largest value) and simultaneously removes it from the array.To remove it from the array, it exchanges the root value with the last element of the array and then deletes the last element using the popback function.

1.0.8. heap extract max

It returns the root value(the largest value) and removes it from the array.To remove it from the array, it exchanges the root value with the last element of the array and then deletes the last element using the popback function.

1.0.9. heap increase key

Firstly I call buildmaxheap.Then I modify the population at a chosen index and reorganizes the max heap property. An exchange takes place if the new value at the designated index is higher than the value of its parent. Until the index value is no longer less than the value of its parent, this process is repeated.

1.0.10. dary increase key

Firstly I call darybuildmaxheap. And i change the population at a selected index and rearranges the max heap property. If the new value at the specified index is greater than its parent’s value, an exchange is performed. This process repeats until the index value is no longer smaller than its parent’s value. Time complexity is O(log(n) / log(d)).

1.0.11. max heap insert

This function for to add a new element to our max heap. It places the new element at index size + 1 and then calls the increase key function to ensure that the index value is adjusted accordingly.

1.0.12. dary insert element

I can add a new element to my d-ary max heap. The increase key function is called after the new element is positioned at index size + 1 to make sure the index value is changed appropriately. Time complexity is O(log(n) / log(d)).

1.0.13. heap maximum

It returns the root value(the largest value).

1.0.14. dary calculate height

The function begins with a height of 1 and, provided k is higher than or equal to power, enters a loop. Power is updated in each cycle by multiplying it by d and then deducting this amount from k. The cycle keeps going until k is smaller than power. Following every update, it increases the height and prints the power and k values as of that moment. If k is still greater than 0 at the conclusion of the loop, the height is raised once again.

1.0.15. build max heap

In the max heapify function, the sorting operation obtained for a single branch is applied to all branches.Thus, a max heap is created.

1.0.16. heapsort

The vector is initially organized to satisfy the max heap property calling buildmaxheap. Subsequently, starting from the last element, it is exchanged with the first element in each step, and then max heapify is called to ensure ordering. Time complexity is O(nlogn) and is stable.

1.0.17. d i k

This function takes three strings and an integer array as parameters. It performs operations based on the first characters of these strings. In each case, it checks the first character of the string and performs specific operations depending on whether it is ’d’, ’i’, or ’k’. If the character is ’d’, the corresponding numerical value is assigned to the first element of the array. If it is ’i’, the value is assigned to the second element. If it is ’k’, the remaining part of the string is parsed to extract the city name (city) and population (pop), and these values are assigned to the third element of the array. The stoi function is used for converting strings to integers in all cases. After these operations, the processed information represented by the string city is returned by the function.

1.1. Max Heap Implementation

1.1.1. MAX-HEAPIFY Procedure

The indices of the left and right children of the variable supplied as an index argument are calculated first in this function. If at least one of the left or right children has an index greater than the current index, the element with the largest child is swapped. By repeatedly invoking itself, this method ensures the orderly arrangement of things along that line.

1.1.2. BUILD-MAX-HEAP Procedure

The sorting process acquired for a single branch is applied to all branches in the max heapify function.

1.1.3. HEAPSORT Procedure

Firstly, The max heap property is satisfied by calling buildmaxheap function. The last element is then swapped out for the first element in each step, starting with the last one, then max heapify is called to guarantee ordering. Time complexity is O(nlogn) and is stable.

1.2. Priority Queue Operations

1.3. Implementation of d-ary Heap Operations

Time complexity is O(log(n) / log(d)).

1.3.1. Height Calculation

The function starts at height 0 and enters a loop if k is greater than or equal to power. Every cycle, power is updated by multiplying it by d and subtracting that result from k. This loop continues until k is less than power. It raises the height and prints the current power and k values after each update. The height is increased once more if, at the end of the loop, k is still bigger than 0.

1.3.2. EXTRACT-MAX Implementation

It returns the root value (the largest value) while also removing it from the array.It exchanges the root value with the last member of the array and then deletes the last element using the popback function to remove it from the array. Time complexity is O(log(n) / log(d)).

1.3.3. INSERT Implementation

In order to add a new element to my d-ary max heap, I built this function. After the new element is positioned at index size + 1, the increase key function is run to make sure the index value is changed appropriately. There is an O(log(n) / log(d)) time complexity.

1.3.4. INCREASE-KEY Implementation

It allows us to modify the population at a chosen index and An exchange takes place if the new value at the selected index is higher than the value of its parent. Until the index value no longer goes below the value of its parent, this action is repeated. There is O(log(n) / log(d)) time complexity.

	Population1	**Population2**	**Population3**	**Population4**
Last Element	305828990	5955532820	2383328206	10127404
Random Element	9814818	8054297	9411106	9675454
Median of 3	8928915	8585322	9488677	10588553
Heapsort	10956919	1788590002	1704970081	1642521568

Table 1.1: Comparison of different pivoting strategies on input data.

1.4. Results

1.4.1. Compare Tıme

The worst-case time complexity of QuickSort is O(n*n) when the last element is always chosen as the pivot, especially when the data is sorted. In this case, HeapSort with a consistent O(nlogn) time complexity outperformed QuickSort in the table, except for the population of 4 (as it was not sorted). However, in other pivot scenarios, QuickSort performed faster because its time complexity was O(nlogn).

1.4.2. Compare the number of comparisons

Heapsort performs O(nlogn) comparisons in all scenarios, while Quicksort makes O(nlogn) comparisons in the best case, and O(n*n) comparisons in the worst case. Consequently, overall, Heapsort is more efficient in terms of comparisons.

1.4.3. Strengths and Weaknesses of Heapsort and Quicksort

Quicksort has the advantage of being well-suited for sorting smaller-sized arrays. However, its drawback lies in the fact that when a good pivot is not chosen, its complexity becomes O(n *n), resulting in longer processing times.

Heapsort, on the other hand, boasts a strong feature with a consistent O(nlogn) complexity, regardless of the case. Its weakness, however, is that it may take longer to sort smaller-sized arrays compared to quicksort.