Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556

ENHANCED HEAP FOR DIJKSTRA'S ALGORITHM

ANUBHAV KUMAR PRASAD
United Institute of Technology, Prayagraj, Uttar Pradesh, India. Email: anubhavkrprasad@gmail.com

ARUNI SINGH
KNIT Sultanpur, Uttar Pradesh, India. Email: arunisingh@knit.ac.in

Abstract

Success of min heap is because of its typical structure with parents containing values smaller than their children which makes them faster for sorting n elements in n$log_2 n$ time. This motivated Dijkstra to implement its 'single source shortest path' algorithm using min heap. However, one drawback of the min heap is its self-destructive nature that has been used in the algorithms of heapsort and priority queues. There is one variant of heap known as Fibonacci heap which is best suited for Dijkstra's algorithm. In the case of the heapsort algorithm, the elements are removed from the min heap one by one which destroy the min heap making it impossible to perform any operation on it which restricts Dijkstra's algorithm in detecting a negative weight cycle. The self-destructive nature of min heap inspired to introduce a novel structure named as Enhanced Heap to resolve the issue of self-destruction without any extra overhead.

Keywords: Min- Heap, Fibonacci Heap, Complete Binary Tree, Dijkstra's Algorithm, Negative Weight Cycle.

1. INTRODUCTION

Quest for a better data structure in terms of Time Complexity [6] for all operations like insertion, deletion, and search called for a new data structure known as heap. Heap [4] is a non-linear (tree) data structure [5] with two basic types: min-heap and max-heap. The former store's minimum value at the root while the latter store's maximum value at the root. It is already defined that insertion always takes place at the leaf and elements are removed from min-heap while sorting, i.e., the min-heap returns all elements while sorting making it an empty tree. The time complexity for insertion of a single element into a heap and deletion from a min-heap is always $log_2 n$, because min-heap rearranges elements such that heap property is never violated.

The application areas for min-heap include priority queues [9], order statistics [10], and even routing tables [11]. As discussed, the in-efficiency of min-heap to retain its elements during sorting makes it different from other sorting algorithms like merge-sort [12], quick-sort [13], or any other such sorting algorithms. While the other sorting algorithms can retain the elements, the min-heap cannot. This research introduces *Enhanced Heap (EH)* to eliminate the in-efficiency of min-heap such that elements are never removed. The copy of each element is maintained in EH such that for next time sorting we can easily get the same sequence without the need to add the elements again and rebuild the min-heap. The EH is also applicable to the Fibonacci Heap [7] which will be used for Dijkstra's algorithm in the latter section to show its usefulness for negative weight cycle.

Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556

This research work has been divided into the following sections: Section 1 contains the Introduction of this research work whereas Section 2 explains the literature related to the work. Priority queue and heap are explained in Section 3 including the Enhanced Heap followed by Section 4 which contains the main equations and formulas. Section 5 introduces our proposed algorithm and Section 6 abbreviates the Experimental work and analysis, whereas discussions on the work is explained in Section 7. Section 8 concludes the proposed work.

To find single source shortest path in a graph, two prime algorithms are implemented: Dijkstra's algorithm [1-3] and Bellman-Ford's algorithm [4]. The advantage of Dijkstra's algorithm over the Bellman-Ford is simple and greedy approach. The disadvantage is not able to detect negative weight cycle [15]. The disadvantage may not seem to be a severe one, as a negative weight cycle is just a general concept. The idea comes from theory of current flow in the circuit when the electrical circuit is represented in the form of graph, the nodes point the current flowing in and out of the junctions and the edges may represent some values of resistance as well as current flow but *Negative weight cycles* in this case opposes Kirchhoff's laws [8]. Hence, the *Negative weight cycles* could not be denied. Therefore, some structural changes are required to modify the min-heap. This modification in the min-heap is named as Enhanced Heap (EH) which introduces the advantage of retaining the deleted elements. This is not of the place to mention here that, priority queue and min-heap are different.

2. LITERATURE REVIEW

Dijkstra's Algorithm is a classical algorithm used to solve the 'single source shortest path' problem using a graph. The algorithm was originally proposed by Edsger Dijkstra [16] in 1959 is used in computer science, operations research, transportation engineering etc. Later the researchers have made significant progress to improve the efficiency of the algorithm, as well as application in different types of graph and optimization problems. Cormen *et al.* [17] provided a detailed description of Dijkstra's Algorithm and its implementation including its time complexity, data structure used, and optimization techniques. The author Sedgewick includes in his work Dijkstra's Algorithm [18] mentioning its working and application, another researcher named Eppstein provided a comprehensive overview of shortest paths and networks, covering topics such as Dijkstra's Algorithm, Bellman-Ford Algorithm, A* Algorithm [19] covering various applications of these algorithms. Garey and Johnson [20] provide a comprehensive guide to the theory of NP-completeness, including the shortest path problem and Dijkstra's Algorithm.

Cormen [21] in 1993 proposed a modification to the algorithm that reduces its time complexity. Eswaran and Tarjan [22] introduce the augmentation problem to obtain an efficient solution for it using Dijkstra's Algorithm. Goldberg and Tarjan [23] presented a new approach to the maximum flow problem, based on a modification of Dijkstra's Algorithm. Thorup [24] proposed a new data structure, called RAM priority queues to

Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556

improve the performance of Dijkstra's Algorithm. Frederickson [25] emphasized data structures that enable online updating of minimum spanning trees, with applications in various problems, including shortest paths. Demetrescu and Italiano [26] proposed a dynamic algorithm for computing shortest paths and reachability in planar graphs. Zwick [27] introduced a new algorithm for computing all pairs' shortest paths, based on bridging sets and rectangular matrix multiplication. Feder *et al.* presented an approximation algorithm for the longest path problem [28] and Cherkassky *et al.* represented the shortest-path algorithms to evaluate their performance in various graphs [29]. Klein and Ravi [30] proposed a nearly best-possible approximation algorithm for the node-capacitated generalized Steiner tree problem. Blum and Karger [31] described an algorithm for the shortest super-string problem, based on a modification of Dijkstra's Algorithm. Finally, Lawler *et al.* [32] provided a guide to the Traveling Salesman Problem, using a variation of Dijkstra's Algorithm.

3. PRIORITY QUEUE VS HEAP

The queue data structure follows the First in First Out (FIFO) policy for its element. This policy is acceptable if the elements are not meant to be sorted in some sense. But when it comes to some type of arrangement, as with the priority consideration of the elements, normal queue operation needs to be modified such that it can fetch the elements in the required order. Consider the following case:

D(4)	A(5)	B(1)	C(10)

Fig 3.1: Priority Queue with priority value increasing from 1 to 10

Fig. 3.1 shows the priority queue with 4 elements with their associated priorities. Rear is pointing at C and front at D. Order for removal of elements must be C, A, D, and B, but queue operation would remove the elements D, A, B, and C which is incorrect. In this case, max heap can be employed with root always storing the highest priority value.

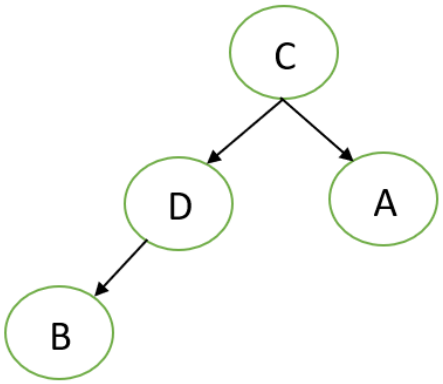


Fig 3.2: Max Heap representation for the Priority Queue in Fig. 3.1

Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556

Depending on the requirement, a min or max heap is better suited for this type of arrangement. As shown in fig. 3.2, the highest priority element is at the root with each parent's priority greater to its children. The elements can be sorted in $O(n log_2 n)$ time with "*n*" elements. This approach handles the condition efficiently, but heap rebuilding is required after every single removal from the heap.

3.1. Enhanced Heap

Drawback of Dijkstra's single source shortest path algorithm is because of the heap data structure that it uses for implementation. The negative weight cycle detection requires visiting the cycle one more time which is not possible as the elements/nodes are removed once visited. Enhanced heap can retain the removed elements/nodes and therefore the negative weight cycle can be visited once again to detect its presence. The enhanced heap has the following properties as enhancement to heap:

i. **Node structure:** Every node will contain three fields:
   a. The first and the second field will maintain the value of the element at that node during insertion.
   b. The third field will contain either 0 or 1 to indicate whether the element is removed from the first field or not. The value "1" is used to indicate that the element has been removed. This will be reversed when every node's first info gets removed.

ii. **Tree structure:** The enhanced heap structure will be a complete binary tree [14].

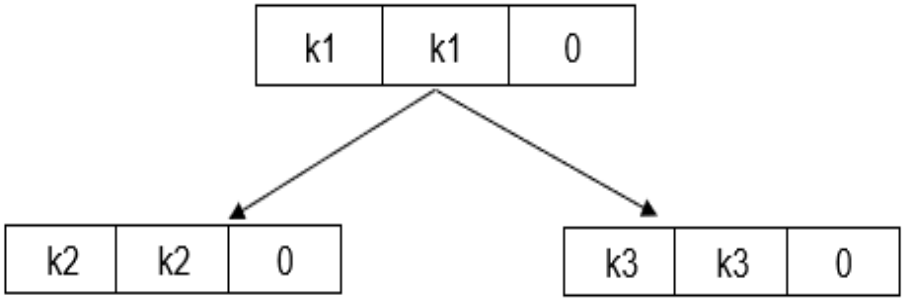
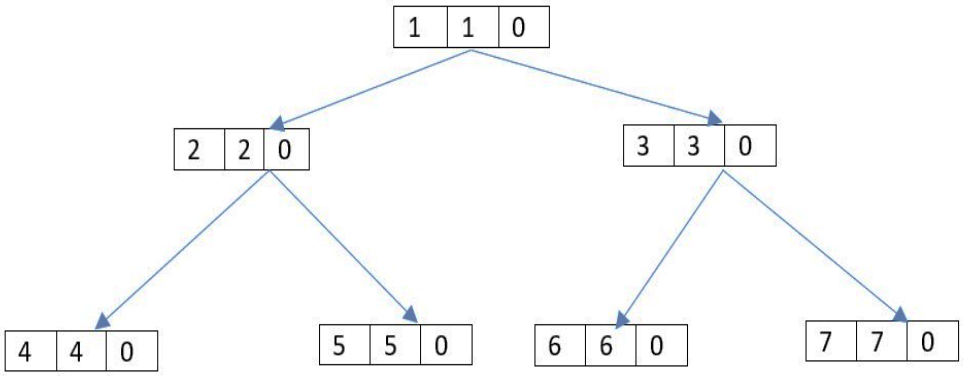


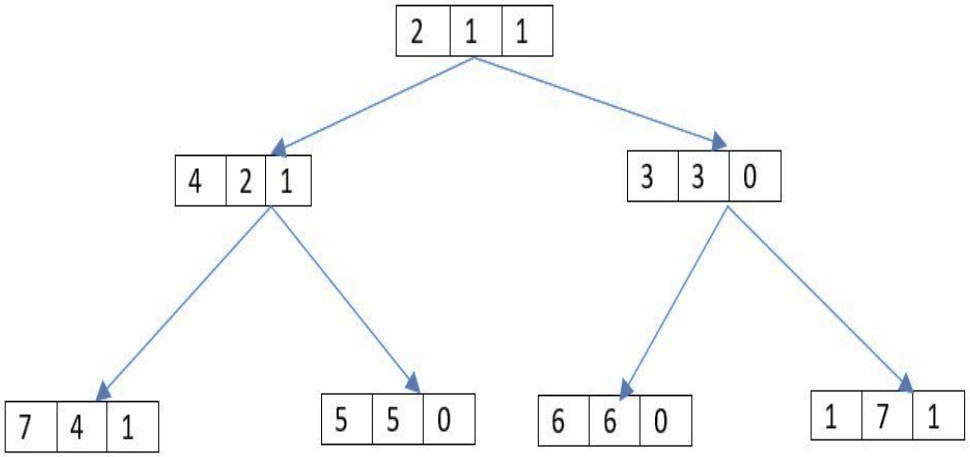
Fig 3.1.1: An Enhanced Heap with three nodes

Fig 3.1.1 contains EH with three nodes. This is a generalized enhanced heap, hence, no min/max condition is shown. The second info contains the first info element in all the tree nodes. The third info is showing value 0, as no element has been removed. Let us consider as an example, the following sequence of elements: 1, 2, 3, 4, 5, 6, and 7 to be inserted in EH.

Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556



Now, suppose we need to extract the smallest element from this heap. The info part will become 1, indicating the movement of the element.



To understand the sorting process, let us look at the array representation of the normal heap with elements 1, 2, 3, 4, 5, 6, and 7:

1	2	3	4	5	6	7

Fig 3.1.2: The Array representation of a Heap

For this normal heap, the successive order of removal of elements will provide the sorted list.

The same heap of fig. 4.2 can be modified to represent the modified heap as shown in fig. 4.3.

1	1	0	2	2	0	3	3	0	4	4	0	5	5	0	6	6	0	7	7	0

Fig 3.1.3: Modified Heap for Fig.3.1.2

Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556

4. FORMULAS USED

For normal heap with index starting from 1, with parent (*p*) index as i:

The address of the Left_child (*l*) = $2 * i$ …… (*1*)

The address of the Right_child (*r*) = $2 * i + 1$ …… (*2*)

For an enhanced heap with index starting from *1*, with parent (*p*) index as *i*:

The address of the Left_child (*l*) = $2 * i$ + 2…… (*3*)

The address of the Right_child (*r*) = $2 * i + 5$ …… (*4*)

From equations *3* and *4*, it is justifiable that EH is still following the properties of a heap, and the mathematical calculations for a normal heap are also applicable to EH with minor modifications.

5. ALGORITHM

5.1 Build_heap

```
void build_heap(int a[ ])
{
  for(int i = n/2-1 ; i >= 0; i-- )
  {
     heapify (a, i); // this line will be replaced with e_heapify for Enhanced Heap_Sort
  }
}
```

The time complexity of build_heap() is proven to be $O(n)$.

5.2 Heapify

```
void heapify(int heap[], int child)
  {
    parent = (child-1)/2
      if(heap[parent]                    <                    heap[child])
        {
          swap(heap[parent], heap[child])
          heapify(heap, parent)
        }
  }
```

The time complexity of heapify() is proven to be $O(log_2 n)$.

Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556

5.3 Heap_Sort

```
void heap_sort(int a[], int n)
{
   build_heap(a)
   for(int i = n-1; i >= 1; i--)
   {
      swap(a[i], a[0]);
      heapify(a, i+1);
   }
}
```

The time complexity of heap_sort() is $O(nlog_2 n)$.

5.4 Enhanced Heap_Sort

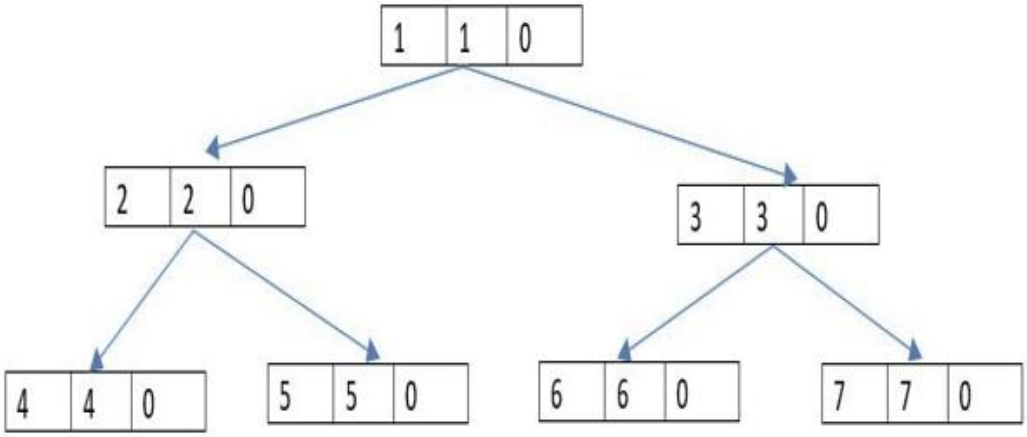
```
void eheap_sort(int a[], int n)
{
   build_heap(a);
   for(int i = 0; i <n; i++)
   {
      swap(a[n/3-3*(i+1)], a[0]);
      heapify(a, i+1);
   }
}
```

The working of eheap_sort() is similar to that of heap_sort(). The only difference is that eheap_sort() doesn't decrease the size of the heap, this is because EH retains the elements. Since elements are not removed, we need to take care of the swapping of the root element with the last legal element of the EH, which is handled by $\mathrm{a[n/3-3*(i+1)]}$. This makes sure that the legal element is selected each time.

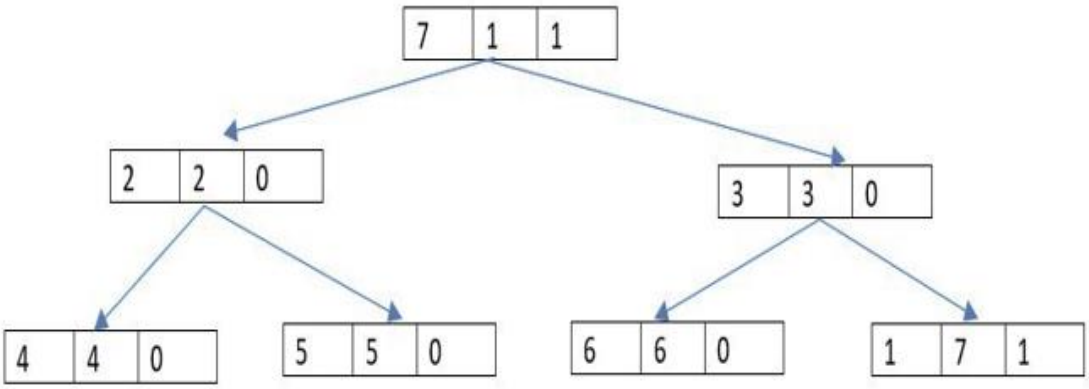
Since the process is similar to that of a normal heap, hence we have the same time complexity of O($nlog_2 n$).

Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556

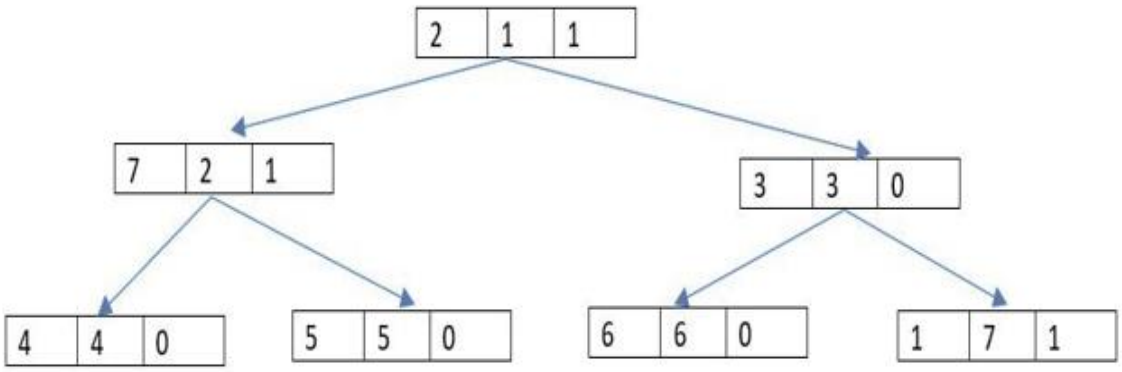
6. EXAMPLE



(a)

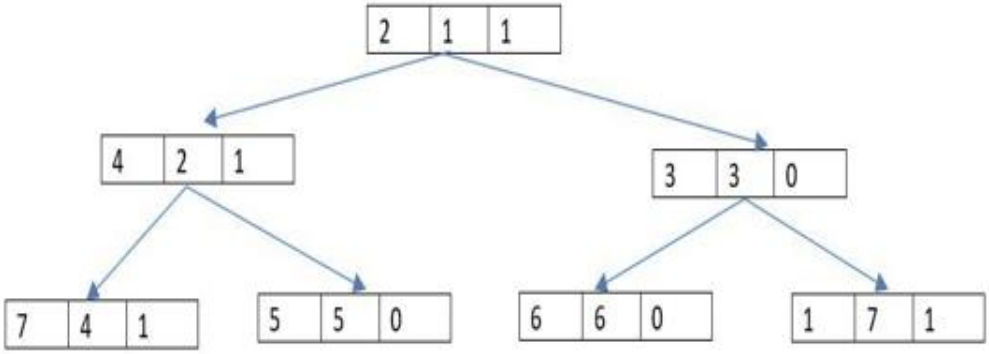


(b)

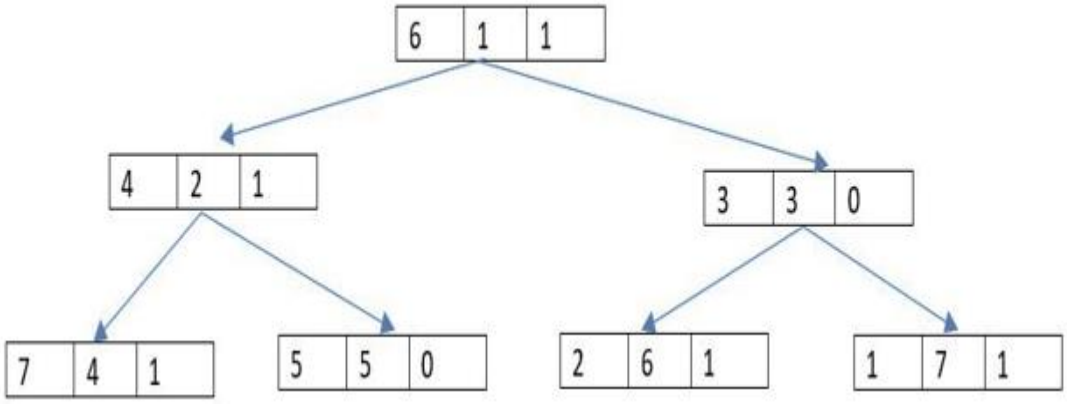


(c)

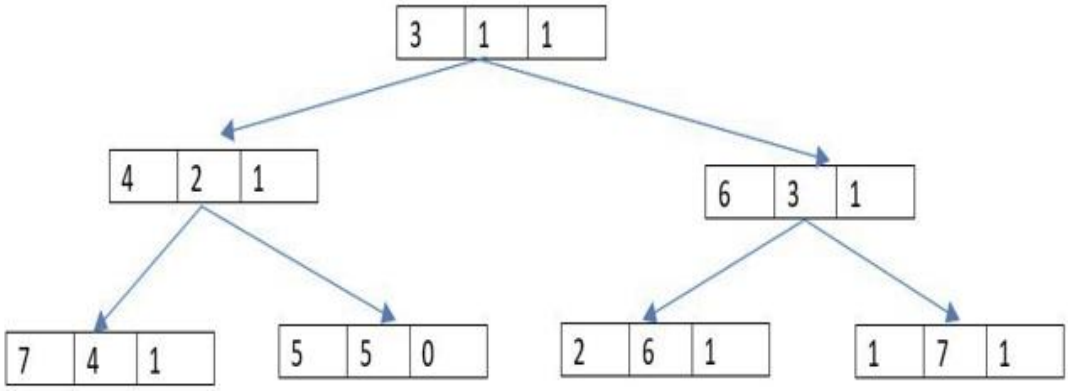
Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556



(d)

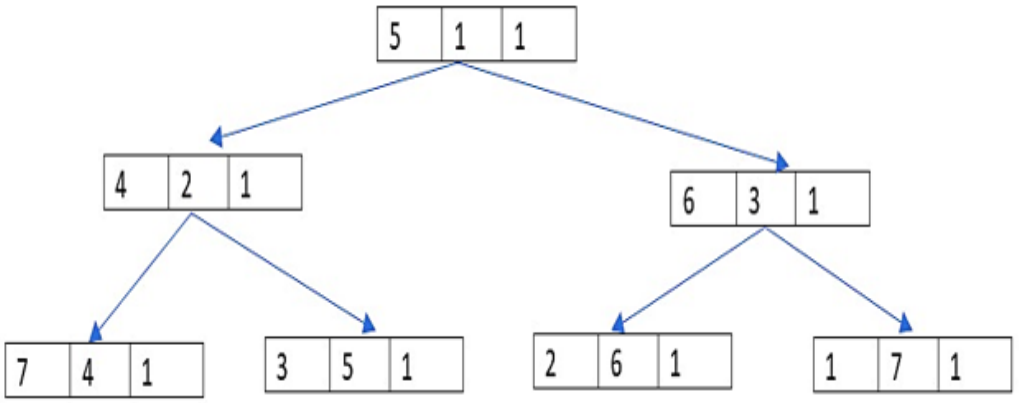


(e)

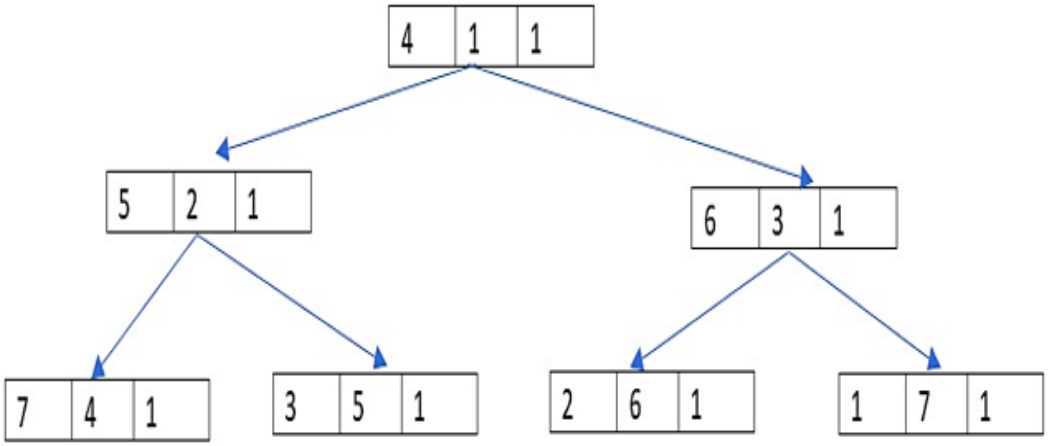


(f)

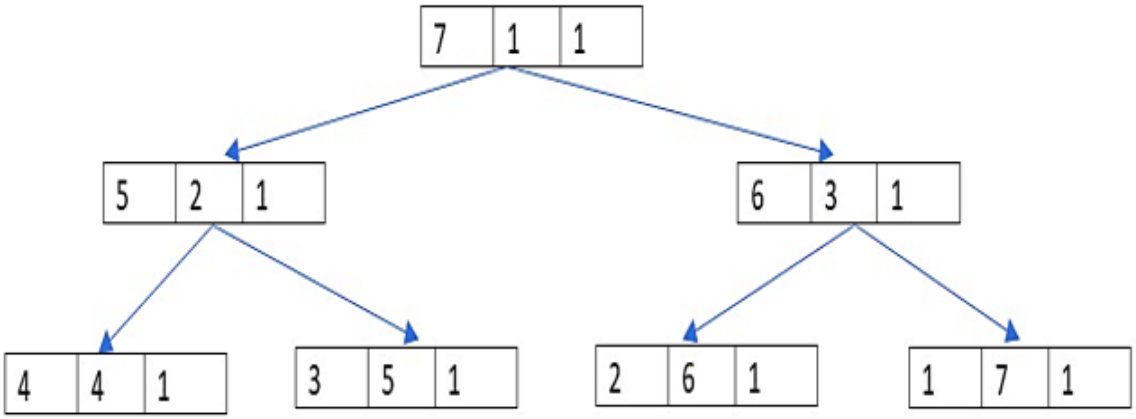
Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556



(g)

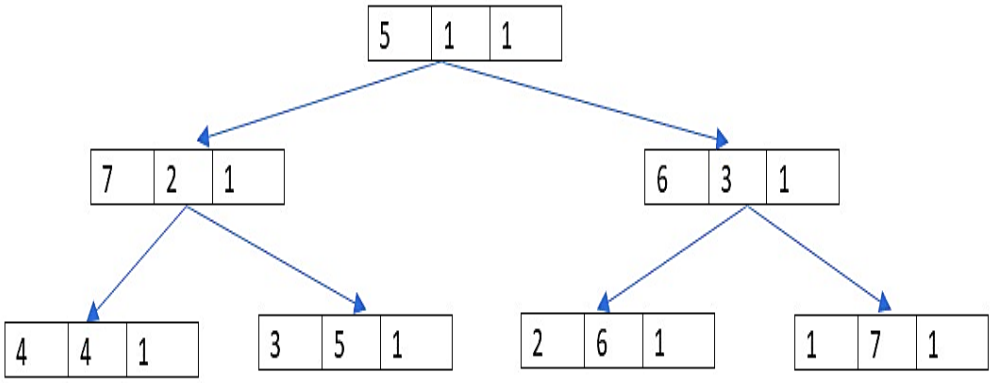


(h)

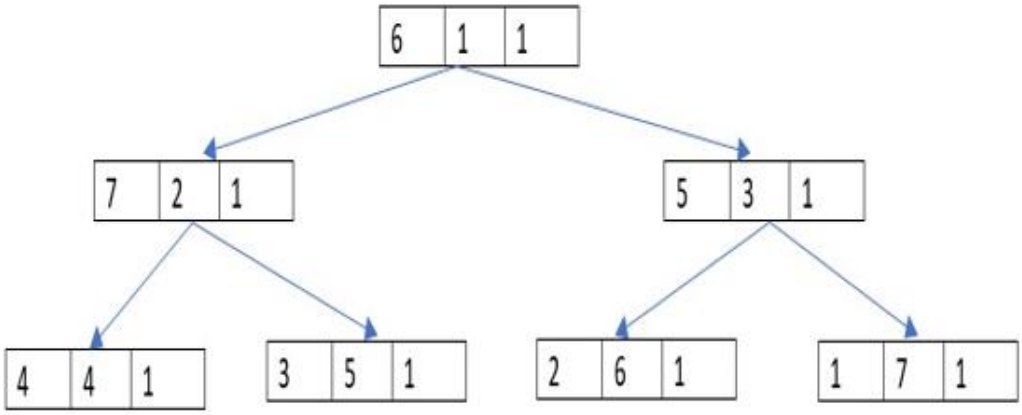


(i)

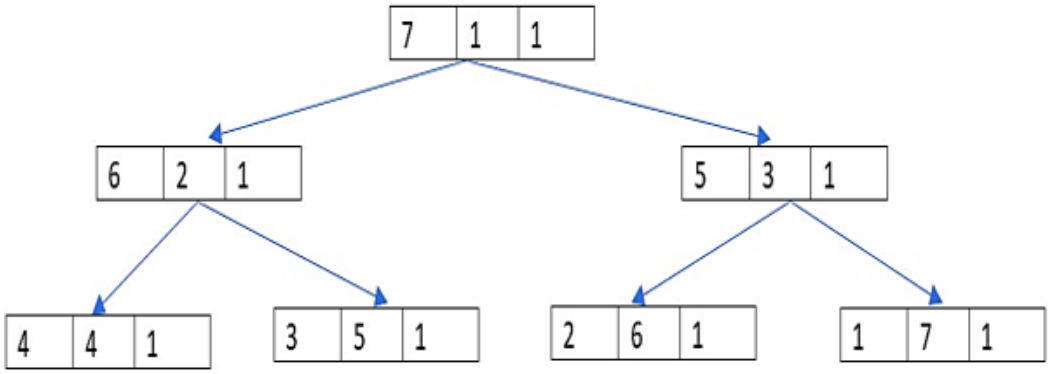
Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556



(j)



(k)



(l)

Fig 6.1: Sorting of elements using the Modified Heap (a)-(l)

Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556

In the example shown above, no element is getting deleted completely because a copy of it in the node.

This ensures that next time sorting, insertion, or deletion can be done on the same heap.

Another point of interest is that the loop will take care of the swapping mechanism as shown below:

```
for(int i = 0; i < size; i++)
{
   swap(root, size – 3*(i + 1));
}
```

The swap function will swap the root element from the specific info part with the last legal element from the last info part.

Since the loop will terminate on the last element, the last element will also be the sorted sequence's last element.

This way we can achieve the functionality of the Heap using the Enhanced Heap and can reuse it too.

7. DIJKSTRA'S ALGORITHM AND ENHANCED HEAP

Single source shortest paths can be calculated using either Bellman-Ford or Dijkstra's algorithms. The advantage which the latter one enjoys is its simple process. The process is a Greedy one and is faster than the former if the sorted sequence of edges is provided.

The priority heap is used for this sorted sequence, which in turn uses min heap as a calculative measure. As the heap size grows, more and more nodes are visited, and finally, it calculates the shortest distances from the source to every other node. The problem however is with heap implementation.

This is because to detect the negative weight cycle, the algo must use the same heap to check for any change in the path's distance calculated. If any difference is found show the presence of it. As heap gets destroyed, we need to again repeat the process for the new path distance calculated.

The above-mentioned problem can be checked if heap is replaced with EH which is discussed with the help of example in next sub-section. All the applications where Dijkstra's Algorithm is applicable like map applications and telephone networks can use the EH to perform better.

Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556

7.1 Example

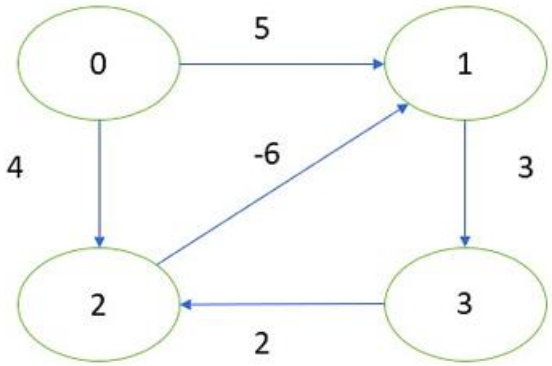


Fig 7.1.1: Graph with negative weight cycle

Consider the above negative weight cycle (2-1-3-2) graph of Fig. 7.1.1. with starting point/source vertex as 0. We can apply the special structure of the EH in the Fibonacci Heap to detect this negative weight cycle:

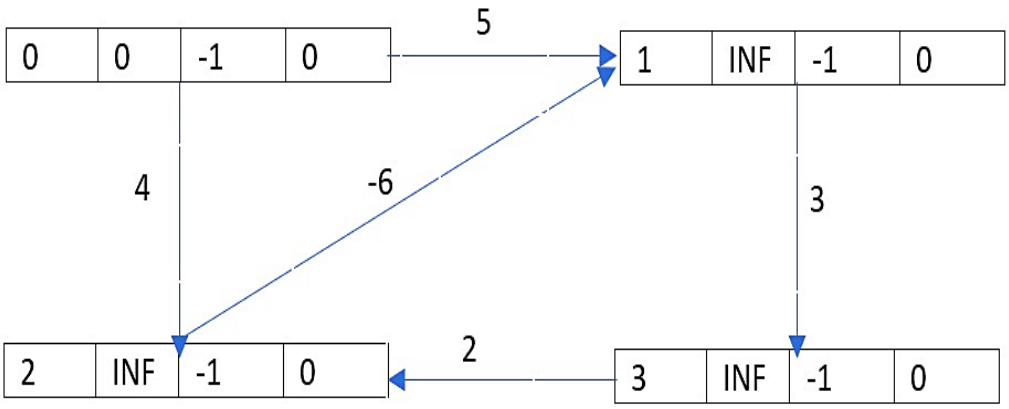


Fig 7.1.2. Graph of Fig.7.1.1.: Using the Enhanced Heap concept with each node representing a name, distance, parent, and used/unused information

The graph is shown in Fig. 7.1.2. depicts the same graph as in Fig.7.1.1. but with following information; (a) name of the vertex, (b) distance of a vertex from the source, (c) parent of the node, and (d) whether that node has been visited or not (0-unvisited, 1-unvisited). The information part (d) is the most important, as it will make sure that the vertex is still present and will not be reused. The steps involved in solving the above graph using Fibonacci Heap and the corresponding MH implementations are shown in fig. 7.1.3 to 7.1.6, with each root parent having two pieces of information; node number and the distance from the source.

Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556

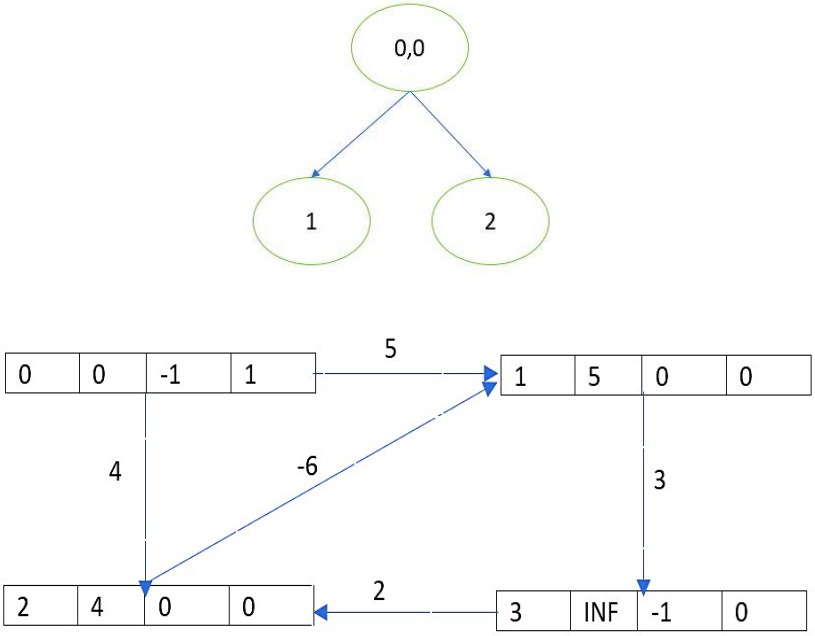


Fig 7.1.3: Exploring the source vertex, 0, and marking it as visited in the corresponding EH implementation

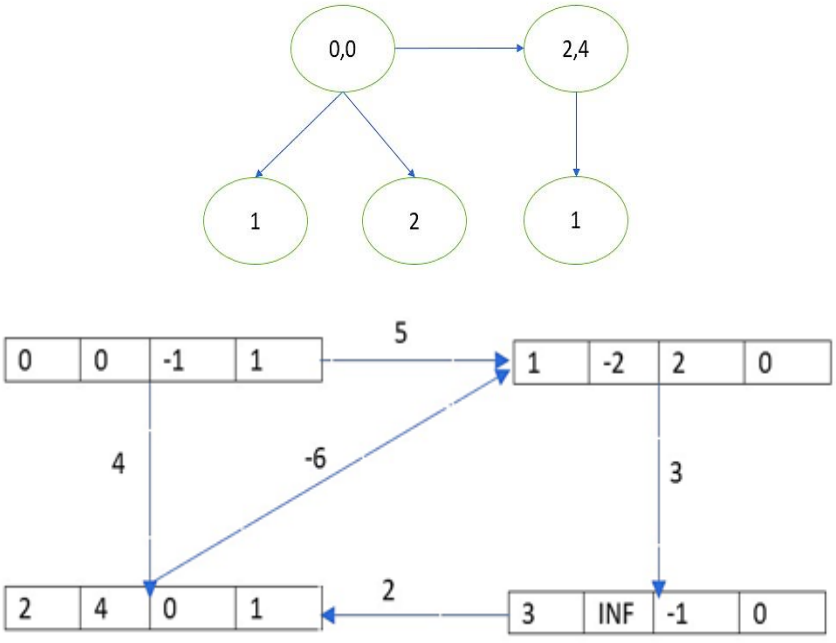


Fig.7.1.4: Exploring the vertex, 2, and marking it as visited in the corresponding EH implementation

Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556

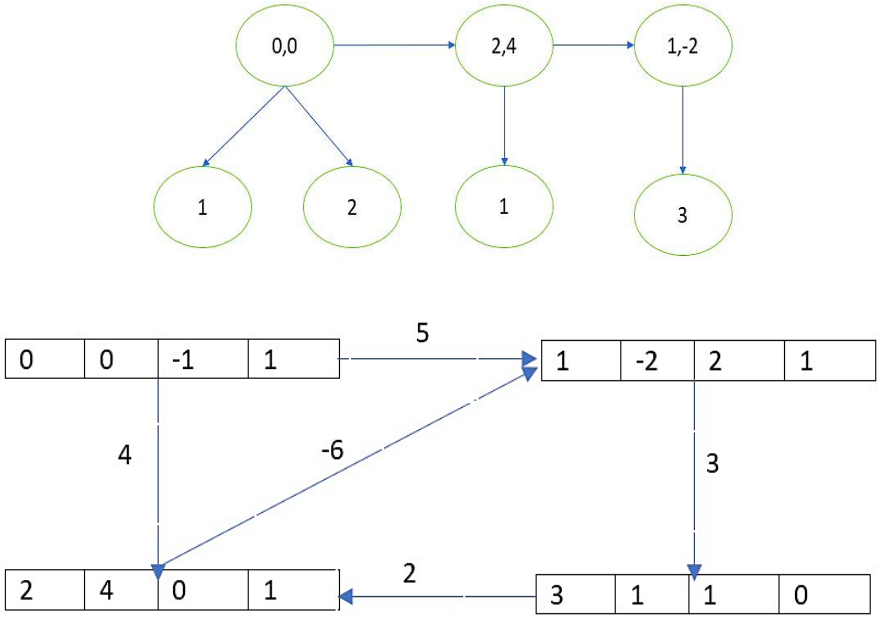


Fig 7.1.5: Exploring the source vertex, 1, and marking it as visited in the corresponding EH implementation

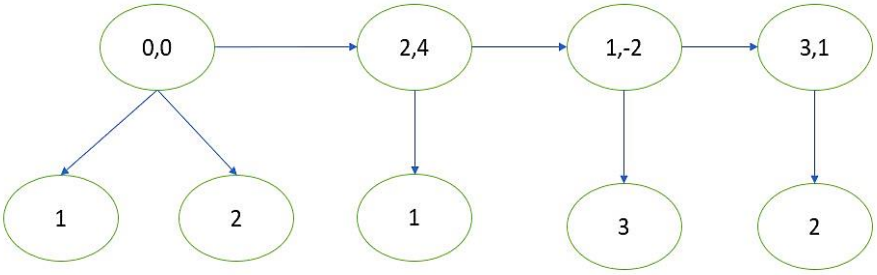


Fig 7.1.6: Exploring the source vertex, 3, and marking it as visited in the corresponding EH implementation

The process stops at this stage, this is because vertex 3 is trying to relax a vertex that is already visited, this indicates the presence of a negative weight cycle. Hence, EH can detect the negative weight cycle in this way.

8. CONCLUSION AND DISCUSSION

The presented research demonstrated the effectiveness of reusable heap named as Enhanced Heap in this work. The concept of reusable heap is extended to Dijkstra's well known algorithm for calculating single source shortest path problem. The EH is capable of handling negative cycle for a given network of routes which removes the drawback of

Xi'an Shiyou Daxue Xuebao (Ziran Kexue Ban)/
Journal of Xi'an Shiyou University, Natural Sciences Edition
ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556

Dijkstra's algorithm discussed earlier in the work. The implementation of EH with Fibonacci heap is presented in Section 7. It has been verified in Section 5 that the time complexity for Dijkstra's algorithm is same with EH as was with min-heap. The application area of EH can be extended to other problem domains where reusable heaps are required. The drawback of heap to retain its structure when sorting process is called makes it inefficient and this is where EH is handy to implement.

References

1) Bellman, R. J. N. J. "Dynamic programming princeton university press princeton." *New Jersey Google Scholar* (1957).

2) Dijkstra, Edsger W. "A note on two problems in connexion with graphs." *Numerische mathematik* 1.1 (1959): 269-271.

3) Kruskal, Joseph B. "On the shortest spanning subtree of a graph and the traveling salesman problem." *Proceedings of the American Mathematical society* 7.1 (1956): 48-50.

4) Hoare, C. A. R. "Algorithm 63, 64 and 65." *Comm. ACM* 4.7 (1961): 321-322.

5) Fyfe, Colin, and Roland Baddeley. "Non-linear data structure extraction using simple Hebbian networks." *Biological cybernetics* 72.6 (1995): 533-541.

6) Woeginger, G. J. (2004). Space and time complexity of exact algorithms: Some open problems. *Proc. IWPEC, LNCS*, *3162*, 281-290.

7) Zhang, W., Jiang, C., & Ma, Y. (2012, October). An improved Dijkstra algorithm based on pairing heap. In *2012 Fifth International Symposium on Computational Intelligence and Design* (Vol. 2, pp. 419-422). IEEE.

8) Quintela, F. R., Redondo, R. C., Melchor, N. R., & Redondo, M. (2009). A general approach to Kirchhoff's Laws. *IEEE Transactions on Education*, *52*(2), 273-278.

9) Hunt, G. C., Michael, M. M., Parthasarathy, S., & Scott, M. L. (1996). An efficient algorithm for concurrent priority queue heaps. *Information Processing Letters*, *60*(3), 151-157.

10) Mahmoud, H. M., Modarres, R., & Smythe, R. T. (1995). Analysis of quickselect: An algorithm for order statistics. *RAIRO-Theoretical Informatics and Applications*, *29*(4), 255-276.

11) Sklower, K. (1991, January). A tree-based packet routing table for Berkeley unix. In *USENIX Winter* (Vol. 1991, pp. 93-99).

12) Cole, R. (1988). Parallel merge sort. *SIAM Journal on Computing*, *17*(4), 770-785.

13) Sedgewick, R. (1978). Implementing quicksort programs. *Communications of the ACM*, *21*(10), 847-857.

14) Choudum, S. A., & Nandini, R. U. (2004). Complete binary trees in folded and enhanced cubes. *Networks: An International Journal*, *43*(4), 266-272.

15) Huang, X. (2006, June). Negative-Weight Cycle Algorithms. In *FCS* (pp. 109-115).

16) Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1(1), 269-271.

17) Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). MIT Press.

ISSN: 1673-064X
E-Publication: Online Open Access
Vol: 68 Issue 02 | 2025
DOI: 10.5281/zenodo.14915556

18) Sedgewick, R. (2011). Algorithms (4th ed.). Addison-Wesley.

19) Eppstein, D. (2007). Shortest paths and networks. In Handbook of Discrete and Computational Geometry (pp. 355-384). CRC Press.

20) Garey, M. R., & Johnson, D. S. (1979). Computers and intractability: A guide to the theory of NP-completeness. W. H. Freeman.

21) Cormen, T. H. (1993). The optimization of Dijkstra's shortest path algorithm. Information Processing Letters, 47(4), 207-210.

22) Eswaran, K. P., & Tarjan, R. E. (1976). Augmentation problems. SIAM Journal on Computing, 5(4), 653-665.

23) Goldberg, A. V., & Tarjan, R. E. (1987). A new approach to the maximum flow problem. Journal of the ACM (JACM), 34(4), 921-940.

24) Thorup, M. (2004). On RAM priority queues. Journal of the ACM (JACM), 51(3), 315-328.

25) Frederickson, G. N. (1987). Data structures for on-line updating of minimum spanning trees, with applications. SIAM Journal on Computing, 16(6), 1031-1046.

26) Demetrescu, C., & Italiano, G. F. (2002). Fully dynamic reachability and shortest paths in planar graphs. SIAM Journal on Computing, 31(1), 160-183.

27) Zwick, U. (2002). All pairs shortest paths using bridging sets and rectangular matrix multiplication. Journal of the ACM (JACM), 49(3), 289-317.

28) Feder, T., Motwani, R., & Olston, C. (1994). Approximating the longest path in a graph. Algorithmica, 11(4), 369-382.

29) Cherkassky, B. V., Goldberg, A. V., & Radzik, T. (1996). Shortest paths algorithms: theory and experimental evaluation. Mathematical Programming, 73(2), 129-174.

30) Klein, P. N., & Ravi, R. (1995). A nearly best-possible approximation algorithm for node-capacitated generalized Steiner trees. Journal of Algorithms, 19(1), 104-115.

31) Blum, A., & Karger, D. (1998). On the shortest superstring problem. SIAM Journal on Computing, 27(5), 1117-1133.

32) Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1993). The Traveling Salesman Problem: A Guide.