DESIGN & ANALYSIS OF ALGORITHMS (4)

techworldthink • March 09, 2022

11 Write the Linear Search Algorithm and analyse the best, worst and average case complexities of the algorithm.

Linear search is also called as sequential search algorithm. It is the simplest searching algorithm. In Linear search, we simply traverse the list completely and match each element of the list with the item whose location is to be found. If the match is found, then the location of the item is returned; otherwise, the algorithm returns NULL.

It is widely used to search an element from the unordered list, i.e., the list in which items are not sorted. The worst-case time complexity of linear search is O(n).

The steps used in the implementation of Linear Search are listed as follows -

- First, we have to traverse the array elements using a for loop.
- In each iteration of for loop, compare the search element with the current array element, and -If the element matches, then return the index of the corresponding array element.
- If the element does not match, then move to the next element.
- If there is no match or the search element is not present in the given array, return -1.

```
procedure linear_search (list, value)

   for each item in the list
      if match item == value
         return the item's location
      end if
   end for

end procedure
```

- Best Case Complexity - In Linear search, best case occurs when the element we are finding is at the first position of the array. The best-case time complexity of linear search is O(1).
- Average Case Complexity - The average case time complexity of linear search is O(n).
- Worst Case Complexity - In Linear search, the worst case occurs when the element we are looking is present at the end of the array. The worst-case in linear search could be when the target element is not present in the given array, and we have to traverse the entire array. The worst-case time complexity of linear search is O(n).

The time complexity of linear search is O(n) because every element in the array is compared only once.

12 Explain the Merge Sort algorithm and give its worst-case analysis.

O(n*logn)

Merge sort is similar to the quick sort algorithm as it uses the divide and conquer approach to sort the elements. It is one of the most popular and efficient sorting algorithm. It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves. We have to define the merge() function to perform the merging.

The sub-lists are divided again and again into halves until the list cannot be divided further. Then we combine the pair of one element lists into two-element lists, sorting them in the process. The sorted two-element pairs is merged into the four-element lists, and so on until we get the sorted list.

MERGE_SORT(arr, beg, end)

if beg < end

set mid = (beg + end)/2

MERGE_SORT(arr, beg, mid)

MERGE_SORT(arr, mid + 1, end)

MERGE (arr, beg, mid, end)

end of if

END MERGE_SORT

The important part of the merge sort is the MERGE function. This function performs the merging of two sorted sub-arrays that are A[beg…mid] and A[mid+1…end], to build one sorted array A[beg…end]. So, the inputs of the MERGE function are A[], beg, mid, and end.

> *MERGE_SORT(arr, beg, mid) ---> T(n/2)*

> *MERGE_SORT(arr, mid + 1, end) ---> T(n/2)*

> *MERGE (arr, beg, mid, end) n*

Tn = g(n) , n=1

Tn = (a.T(n/b)) + f(n) , n>1

here = a,b=2

f(n) = n

log a base of b ?

log 2 base of 2 is 1

n^k? where k = 1

n^k = n

so, n*logn will be the worst case complexity

O(n.logn)

13 Write Kruskal's algorithm to compute the minimum cost spanning tree.

There are two methods to find Minimum Spanning Tree

1. Kruskal's Algorithm
2. Prim's Algorithm

Kruskal's Algorithm:

An algorithm to construct a Minimum Spanning Tree for a connected weighted graph. It is a Greedy Algorithm. The Greedy Choice is to put the smallest weight edge that does not because a cycle in the MST constructed so far.

If the graph is not linked, then it finds a Minimum Spanning Tree.

Steps for finding MST using Kruskal's Algorithm:

- Step 1: Sort all edges in increasing order of their edge weights.
- Step 2: Pick the smallest edge.
- Step 3: Check if the new edge creates a cycle or loop in a spanning tree.
- Step 4: If it doesn't form the cycle, then include that edge in MST. Otherwise, discard it.
- Step 5: Repeat from step 2 until it includes |V| - 1 edges in MST.

Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A *minimum spanning tree (MST)* or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

How many edges does a minimum spanning tree has?

1. A minimum spanning tree has (V – 1) edges where V is the number of vertices in the given graph.

What Is Union Find Algorithm?

Union Find is an algorithm that keeps track of elements that are split into one or over one disjoint set. It has two primary operations: Find and Union. The Find operation returns the set of elements to which the given element (argument) belongs, whereas the Union operation merges two disjoint sets.

You need to divide the provided graph G(V, E) into three separate sets while building the Minimum Spanning Tree using Kruskal's approach. The first contains edge weight values, the second has a tree hierarchy for distinct nodes, and the third includes the rank of all nodes. By using Union and Find operations, it joins the distinct nodes which are treated as different trees themselves to formulate a minimum spanning tree.

14 Explain the dynamic programming algorithm for the Travelling Salesman problem.

Problem Statement

A traveler needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once. What is the shortest possible route that he visits each city exactly once and returns to the origin city?

Solution

Travelling salesman problem is the most notorious computational problem. We can use brute-force approach to evaluate every possible tour and select the best one. For n number of vertices in a graph, there are $(n - 1)!$ number of possibilities.

Instead of brute-force using dynamic programming approach, the solution can be obtained in lesser time, though there is no polynomial time algorithm.

Let us consider a graph $G = (V, E)$, where V is a set of cities and E is a set of weighted edges. An edge $e(u, v)$ represents that vertices u and v are connected. Distance between vertex u and v is $d(u, v)$, which should be non-negative.

Suppose we have started at city *1* and after visiting some cities now we are in city *j*. Hence, this is a partial tour. We certainly need to know *j*, since this will determine which cities are most convenient to visit next. We also need to know all the cities visited so far, so that we don't repeat any of them. Hence, this is an appropriate sub-problem.

For a subset of cities $S \in \{1, 2, 3, \ldots, n\}$ that includes *1*, and $j \in S$, let $C(S, j)$ be the length of the shortest path visiting each node in S exactly once, starting at *1* and ending at *j*.

When $|S| > 1$, we define $C(S, 1) = \propto$ since the path cannot start and end at 1.

Now, let express C(S, j) in terms of smaller sub-problems. We need to start at *1* and end at j. We should select the next city in such a way that

$C(S,j) = \min C(S-\{j\},i) + d(i,j) \text{ where } i \in S \text{ and } i \neq j \; c(S,j)$

$= \min C(s-\{j\},i) + d(i,j) \text{ where } i \in S \text{ and } i \neq j$

$C(S,j) = \min C(S-\{j\},i) + d(i,j) \text{ where } i \in S \text{ and } i \neq j \; c(S,j) = \min C(s-\{j\},i) + d(i,j) \text{ where } i \in S \text{ and } i \neq j$

```
Algorithm: Traveling-Salesman-Problem
C ({1}, 1) = 0
for s = 2 to n do
   for all subsets S ∈ {1, 2, 3, … , n} of size s and containing 1
      C (S, 1) = ∞
   for all j ∈ S and j ≠ 1
      C (S, j) = min {C (S – {j}, i) + d(i, j) for i ∈ S and i ≠ j}
Return minj C ({1, 2, 3, …, n}, j) + d(j, i)
```

Analysis

There are at the most (2^n).n sub-problems and each one takes linear time to solve.Therefore, the total running time is O(2^n . n^2)