Department of Computer Science and Engineering

Lab Manual

Program: B.Tech CSE (AIDS)	
Course Name: Design and Analysis of Algorithm Lab **Subject Code:** 231ADUCL32 **Credits:** 1	**Year & Sem:** SE-II **Practical:** 02 hours/week **Total Contact Hours:**PR-30hrs
Faculty: Mrs. Kausar Fakir Mrs. Kalyani Salvi	**Academic Year:** 2024-25 (Odd)

Institutional Vision, Mission and Quality Policy

Our Vision

To foster and permeate higher and quality education with value added engineering, technology programs, providing all facilities in terms of technology and platforms for all round development with societal awareness and nurture the youth with international competencies and exemplary level of employability even under highly competitive environment so that they are innovative adaptable and capable of handling problems faced by our country and world at large.

RAIT's firm belief in new form of engineering education that lays equal stress on academics and leadership building extracurricular skills has been a major contribution to the success of RAIT as one of the most reputed institution of higher learning. The challenges faced by our country and world in the 21 Century needs a whole new range of thought and action leaders, which a conventional educational system in engineering disciplines are ill equipped to produce. Our reputation in providing good engineering education with additional life skills ensure that high grade and highly motivated students join us. Our laboratories and practical sessions reflect the latest that is being followed in the Industry. The project works and summer projects make our students adept at handling the real life problems and be Industry ready. Our students are well placed in the Industry and their performance makes reputed companies visit us with renewed demands and vigour.

Our Mission

The Institution is committed to mobilize the resources and equip itself with men and materials of excellence thereby ensuring that the Institution becomes pivotal center of service to Industry, academia, and society with the latest technology. RAIT engages different platforms such as technology enhancing Student Technical Societies, Cultural platforms, Sports excellence centers, Entrepreneurial Development Center and Societal Interaction Cell. To develop the college to become an autonomous Institution & deemed university at the earliest with facilities for advanced research and development programs on par with international standards. To invite international and reputed national Institutions and Universities to collaborate with our institution on the issues of common interest of teaching and learning sophistication.

RAIT's Mission is to produce engineering and technology professionals who are innovative and inspiring thought leaders, adept at solving problems faced by our nation and world by providing quality education.

The Institute is working closely with all stake holders like industry, academia to foster knowledge generation, acquisition, dissemination using best available resources to address the great challenges being faced by our country and World. RAIT is fully dedicated to provide its students skills that make them leaders and solution providers and are Industry ready when they graduate from the Institution.

We at RAIT assure our main stakeholders of students 100% quality for the programmes we deliver. This quality assurance stems from the teaching and learning processes we have at work at our campus and the teachers who are handpicked from reputed institutions IIT/NIT/MU, etc. and they inspire the students to be innovative in thinking and practical in approach. We have installed internal procedures to better skills set of instructors by sending them to training courses, workshops, seminars and conferences. We have also a full-fledged course curriculum and deliveries planned in advance for a structured semester long programme. We have well developed feedback system employers, alumni, students and parents from to fine tune Learning and Teaching processes. These tools help us to ensure same quality of teaching independent of any individual instructor. Each classroom is equipped with Internet and other digital learning resources.

The effective learning process in the campus comprises a clean and stimulating classroom environment and availability of lecture notes and digital resources prepared by instructor from the comfort of home. In addition student is provided with good number of assignments that would trigger his thinking process. The testing process involves an objective test paper that would gauge the understanding of concepts by the students. The quality assurance process also ensures that the learning process is effective. The summer internships and project work based training ensure learning process to include practical and industry relevant aspects. Various technical events, seminars and conferences make the student learning complete.

Departmental Vision, Mission

Vision

To impart higher and quality education in computer science with value added engineering and technology programs to prepare technically sound, ethically strong engineers with social awareness. To extend the facilities, to meet the fast changing requirements and nurture the youths with international competencies and exemplary level of employability and research under highly competitive environments.

Mission

To mobilize the resources and equip the institution with men and materials of excellence to provide knowledge and develop technologies in the thrust areas of computer science and Engineering. To provide the diverse platforms of sports, technical, co curricular and extracurricular activities for the overall development of student with ethical attitude. To prepare the students to sustain the impact of computer education for social needs encompassing industry, educational institutions and public service. To collaborate with IITs, reputed universities and industries for the technical and overall upliftment of students for continuing learning and entrepreneurship.

Departmental Program Outcomes (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences..
PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

PSO1: To build competencies towards problem solving with an ability to understand, identify, analyze and design the problem, implement and validate the solution including both hardware and software.

PSO2: To build appreciation and knowledge acquiring of current computer techniques with an ability to use skills and tools necessary for computing practice.

PSO3: To be able to match the industry requirements in the area of computer science and engineering. To equip skills to adopt and imbibe new technologies.

Index

List of Experiments

Sr. No.	Experiments Name
1.	Implementation of Selection Sort.
2.	Implementation of Binary Search.
3.	Implementation of Merge Sort.
4.	Implementation of Knapsack Problem.
5.	Implementation of Single Source Shortest Path Algorithm (Dijkstra).
6.	**Implementation of Huffman Coding Algorithm.**
7.	Implementation of Single Source Shortest Path Algorithm (Bellman-Ford).
8.	Implementation of All Pair Shortest Path Algorithm (Floyd Warshall).
9.	Implementation of Longest Common Subsequence Problem.
10.	Implementation of 8 Queen's Problem.
11.	Implementation of Naïve String Matching Algorithm.

Course Objective, Course Outcome & Experiment Plan

Course Objective :

1	To introduce the methods of designing and analysing algorithms
2	Design and implement efficient algorithms for a specified application
3	Strengthen the ability to identify and apply the suitable algorithm for the given real-world Problem
4	Analyze worst-case running time of algorithms and understand fundamental algorithmic problems

Course Outcomes :

CO1	Analyze the running time and space complexity of algorithms.
CO2	**Implement and analyze the complexity of algorithms using divide and conquer approach and compare its complexity with other approach for solving specific problem.**
CO3	**Implement and Analyze the greedy strategy for solving optimization problems and compare its complexity with other approach for solving specific problem.**
CO4	**Implement, analyze and apply dynamic programming strategy for solving the optimization problem and compare its complexity with other approach for solving specific problem.**
CO5	**Implement and analyze complexity of the backtracking and branch and bound approach and compare its complexity with other approach to solve specific problem.**
CO6	**Implement and analyze complexity of different string-matching algorithms.**

Experiment Plan :

Module No.	Week No.	Experiments Name	Course Outcome	Weightage
1	W1	Implementation of Selection Sort.	CO1	10
2	W2	Implementation of Binary Search.	CO2	05
3	W3	Implementation of Merge Sort.	CO2	05
4	W4	Implementation of Knapsack Problem.	CO3	03
5	W5	Implementation of Single Source Shortest Path Algorithm (Dijkstra).	CO3	03
6	W6	**Implementation of Huffman Coding Algorithm.**	CO3	04
7	W7	Implementation of Single Source Shortest Path Algorithm (Bellman-Ford).	CO4	03
8	W8	Implementation of All Pair Shortest Path Algorithm (Floyd Warshall).	CO4	03
9	W9	Implementation of Longest Common Subsequence Problem.	CO4	04
10	W10	Implementation of 8 Queen's Problem.	CO5	10
11	W11	Implementation of Naïve String Matching Algorithm.	CO6	10

Mapping of Course outcomes with Program outcomes:

Subject Weight	Course Outcomes		Contribution to Program outcomes PO'S											
			1	2	3	4	5	6	7	8	9	10	11	12
PRACTICAL 100%	CO1	Analyze the running time and space complexity of algorithms.		2	2	1	1		2					2
	CO2	Implement and analyze the complexity of algorithms using divide and conquer approach.		2	2		1		1		1		1	2
	CO3	Implement and analyze the complexity of algorithms using greedy strategy.		2	2		1		2		1		1	1
	CO4	plement and analyze the complexity of algorithms using dynamic programming strategy		2	2		1		1		1		1	2
	CO5	plement and analyze the complexity of algorithms using backtracking approach.		2	2	1	1		1		1		1	1
	CO6	plement and analyze the complexity of algorithms using string-matching techniques.	3	2	2	2			1					

CO Weightage

Course Outcomes	Weightage (in %)
CO1: Analyze the running time and space complexity of algorithms.	01 Exp - 09%
CO2: Analyze the running time and space complexity of algorithms.	02 Exp – 19%
CO3: Implement and analyze the complexity of algorithms using divide and conquer approach.	03 Exp - 27%
04: Implement and analyze the complexity of algorithms using greedy strategy.	03 Exp - 27%
05: Implement and analyze the complexity of algorithms using dynamic programming strategy	01 Exp – 09%
06: Implement and analyze the complexity of algorithms using backtracking approach.	01 Exp - 09%

Study and Evaluation Scheme

Course Code	Course Name	Teaching Scheme			Credits Assigned			
231ADUCL32	**Design and Analysis of Algorithm Lab**	Theory	Practical	Tutorial	Theory	TW/ Pract	Tutorial	Total
		--	02	--	--	01	--	01

Course Code	Course Name	Examination Scheme		
CEL403	**Design and Analysis of Algorithm Lab**	Term Work	Practical & Oral	Total
		25	25	50

Term Work:

The Term work Marks are based on the weekly experimental performance of the students, Oral performance and regularity in the lab. Students are expected to be prepared for the lab ahead of time by referring the manual and perform the experiment under the guidance and discussion. Next week the experiment write-up to be corrected along with oral examination.

Practical/Experiments:

Oral & Practical Exam will be based on the experiments implemented in the Laboratory.

Design and Analysis of Algorithm Lab

Experiment No.: 1

Selection Sort

Experiment No. 1

1. **Aim:** To implement Selection Sort.

2. **Objectives:**
   - To provide mathematical approach for Analysis of Algorithms.
   - To calculate time complexity and space complexity.
   - To solve problems using various strategies.

3. **Outcomes:**
   - Students will be able to analyse the complexities of various problems in different domains.

4. **Hardware / Software Required:** Turbo C

5. **Theory:**

This sorting is called "Selection Sort" because it works by repeatedly element. It works as follows: first find the smallest in the array and exchange it with the element in the first position, then find the second smallest element and exchange it with the element in the second position, and continue in this way until the entire array is sorted.

Selection sort is among the simplest of sorting techniques and it work very well for small files. Furthermore, despite its evident "naïve approach "Selection sort has a quite important application because each item is actually moved at most once, Section sort is a method of choice for sorting files with very large objects (records) and small keys

The worst case occurs if the array is already sorted in descending order. Nonetheless, the time require by selection sort algorithm is not very sensitive to the original order of the array to be sorted: the test "if $A[j]$ < min x" is executed exactly the same number of times in every case. The variation in time is only due to the number of times the "then" part (i.e., min $j \leftarrow j$; min $x \leftarrow A[j]$ of this test are executed.

The Selection sort spends most of its time trying to find the minimum element in the "unsorted" part of the array. It clearly shows the similarity between Selection sort and Bubble sort. Bubble sort "selects" the maximum remaining elements at each stage, but wastes some effort imparting some order to "unsorted" part of the array. Selection sort is quadratic in both the worst and the average case, and requires no extra memory.

6. Algorithm:

SELECTION SORT

1. Start.
2. Define an array 'A' of size 'n'.
3. Read array elements from the user.
4. Define first element of the array as 'min' i.e. min=A [0].
5. for i =0 to n-2

   5.1 min =A[i]

   5.2 loc = i //loc variable keeps the track of minimum element

   5.3 //Inner Loop

   for j =i+1 to n-1

   //find number of comparisons and swapping for reference.

   5.3.1 if (A[j]<min) then

   5.3.1.1 min=A[j]

   5.3.1.2 loc=j

   5.4 if (loc! =i) then

   5.5 swap(A[i], A[loc])
6. Display sorted array.
7. Stop.

7. Conclusion and Discussion:

In selection sort the first pass makes (n-1) comparisons, the second pass makes (n-2) and so on. Therefore, time complexity becomes O (n^2).

8. Viva Questions:

- What is the basic operation in selection sort algorithm?
- What is the complexity of selection sort?
- Which one is best sorting algorithm?

9. References:

- Ellis horowitz ,sartajSahni , s. Rajsekaran. "Fundamentals of computer algorithms" University Press.
- T.H.coreman , C.E. Leiserson,R.L. Rivest, and C. Stein, "Introduction to algorithms", 2nd edition , PHI publication 2005.
- http://www.tutorialspoint.com/data_structures_algorithms/selection_sort_algorithm.htm
- http://interactivepython.org/runestone/static/pythonds/SortSearch/TheSelectionSort.html
- http://www.programiz.com/article/selection-sort-algorithm-programming

Design and Analysis of Algorithm Lab

Experiment No. : 2

Binary Search Technique

Experiment No. 2

1. **Aim :** Write a program to implement binary search technique.

2. **Objectives :**
   - To provide mathematical approach for Analysis of Algorithms.
   - To calculate time complexity and space complexity.
   - To solve problems using various strategies.

3. **Outcome :**
   - Students will be able to prove the correctness and analyze the running time of the basic algorithms for those classic problems in various domains using divide and conquer strategy.

4. **Hardware / Software Required :** Turbo C

5. **Theory :**

A binary search algorithm is a technique for finding a particular value in a linear array, by ruling out half of the data at each step, widely but not exclusively used in computer science.

A binary search finds the median, makes a comparison to determine whether the desired value comes before or after it, and then searches the remaining half in the same manner. Another explanation would be: Search a sorted array by repeatedly dividing the search interval in half Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half Otherwise, narrow it to the upper half.

6. **Algorithm :**

1. Read array elements in increasing order.
2. Read the element to be searched, say 'KEY '.
3. int BinarySearch(int array[], int start_index, int end_index, int KEY)

```
{
   if (end_index >= start_index)
   {
      int middle = start_index + (end_index - start_index )/2;
      if (array[middle] == KEY)
         return middle;
      if (array[middle] > KEY)
         return BinarySearch(array, start_index, middle-1, KEY);
```

```
            return BinarySearch(array, middle+1, end_index, KEY);
        }
        return -1;
    }
```

7. Conclusion and Discussion :

The method is called is binary search because at each step, we reduce the length of the table to be searched by half and the table is divided into two equal parts. Binary Search can be accomplished in logarithmic time in the worst case, i.e., $T(n) = \theta(\log n)$. This version of the binary search takes logarithmic time in the best case.

8. Viva Questions :

- Which strategy binary search makes use of?
- What is the best case and worst complexity of binary search?
- Draw a tree representation of binary search

9. References :

- http://interactivepython.org/runestone/static/pythonds/SortSearch/TheBinarySearch.html
- https://www.khanacademy.org/computing/computer-science/algorithms/binary-search/a/implementing-binary-search-of-an-array
- http://www.tutorialspoint.com/data_structures_algorithms/binary_search_algorithm.htm

Design and Analysis of Algorithm Lab

Experiment No.: 3

Merge Sort

Experiment No. 3

1. **Aim :** Write a program to implement Merge Sort**.**

2. **Objectives :**
   - To provide mathematical approach for Analysis of Algorithms.
   - To calculate time complexity and space complexity.
   - To solve problems using various strategies.

3. **Outcome :**
   - Students will be able to prove the correctness and analyze the running time of the basic algorithms for those classic problems in various domains using divide and conquer strategy.

4. **Hardware / Software Required :** Turbo C

5. **Theory :**

Merge sort algorithms are based on a divide and conquer strategy. First, the sequence to be sorted is decomposed into two halves (Divide). Each half is sorted independently (Conquer). Then the two sorted halves are merged to a sorted sequence (Combine).

Its worst-case running time has a lower order of growth than insertion sort. Since we are dealing with subproblems, we state each subproblem as sorting a subarray $A[p .. r]$. Initially, $p = 1$ and $r = n$, but these values change as we recurse through sub problems.

To sort $A[p .. r]$:

1. Divide Step

If a given array A has zero or one element, simply return; it is already sorted. Otherwise, split $A[p .. r]$ into two sub arrays $A[p.. q]$ and $A[q + 1 .. r]$, each containing about half of the elements of $A[p .. r]$. That is, q is the halfway point of $A[p .. r]$.

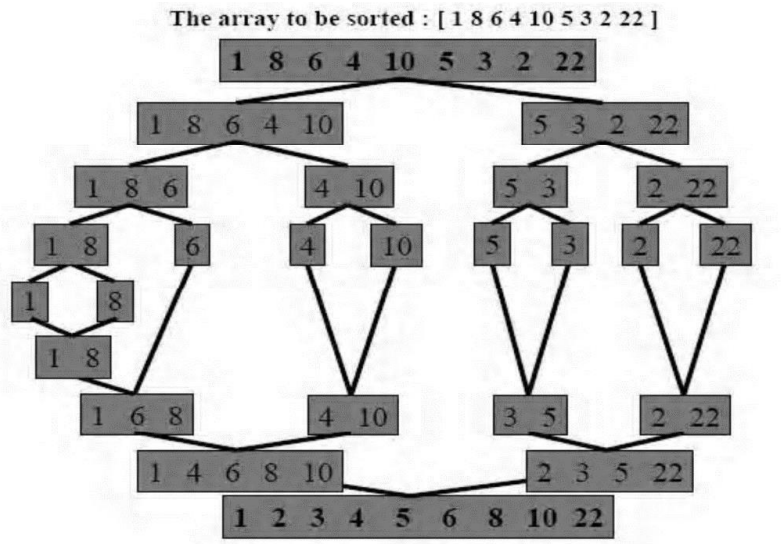
2. Conquer Step

Conquer by recursively sorting the two sub arrays $A[p .. q]$ and $A[q + 1 .. r]$.

3. Combine Step

Combine the elements back in $A[p .. r]$ by merging the two sorted sub arrays $A[p .. q]$ and $A[q + 1 .. r]$ into a sorted sequence. To accomplish this step, we will define a procedure MERGE (A, p, q, r).

Example



6. Algorithm :

Algorithm MergeSort(int A[0..n-1,low,high) :

```
if (low<high) then
    {
        mid ← (low+high)/2
        MergeSort (A,low,mid)
        MergeSort (A,mid+1, high)
        Combine(A,low,mid,high)
    }
```

Algorithm Combine(int A[0…n-1,low,high) :

1. k ← low
2. i ← low
3. j ← mid+1
4. while(i<= mid && j<=high) do

```
    {
        if (A[i]<=A[j] then
            {
                temp[k] ← A[i]
                i←i+1
                k←k+1
            }
```

```
        else
            {
                temp[k] ← A[j]
                j←j+1
                k←k+1
            }
     }
```

5. while (i<=mid) do

```
    {
        temp[k] ← A[i]
        i←i+1
        k←k+1
     }
```

6. while (j<=high) do

```
    {
        temp[k] ← A[j]
        j←j+1
       k←k+1
    }
```

7. **Conclusion and Discussion :**

Thus in merge sort no best/worst cases. Log N divisions before a size of 1 is reached. Each step requires 0(N) for merging. Hence O(N log N). Auxiliary storage as large as the original is required

8. **Viva Questions :**
   - What is the space complexity of merge sort?
   - What strategy merges sort uses for sorting?

9. **References :**

   - http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/mergeSort.htm
   - https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/overview-of-merge-sort

Design and Analysis of Algorithm Lab

Experiment No.: 04

Knapsack Problem

Experiment No. 04

1. **Aim :** Write a program to implement Knapsack Problem

2. **Objectives :**
- To provide mathematical approach for Analysis of Algorithms.
- To calculate time complexity and space complexity.
- To solve problems using various strategies.

3. **Outcome :**
- Students will be able to create and apply the efficient algorithms for the effective problem solving with the help of different strategies like greedy method.

4. Hardware / Software Required : Turbo C

5. Theory :

Problem Definition :

We have given an empty knapsack of capacity 'W' and we are given 'n' different objects from i=1,2……n. Each object 'i' has some positive weight 'w_i' and some profit value is associated with each object which is denoted as 'p_i'.

We want to fill the knapsack with total capacity 'W' such that profit earned is maximum. When we solve this problem main goal is :

1. Choose only those objects that give maximum profit.
2. The total weight of selected objects should be <= W

Most problems have n input and require us to obtain a subset that satisfies some constraints is called as a feasible solution. We are required to find a feasible solution that optimize (minimize or maximizes) a given objective function. The feasible solution that does this is called an optimal solution.

Example :

n = 3
profit ->Pi = (25,24,15)
Weight (in kg) -> Wi = (18,15,10) where i = 1,2,3
and knapsack capacity = 20

Feasible solutions :

1. As knapsack capacity is 20 , we first put W1(i.e 18 kg) into kanpsack , hence gained profit is 25 ; but now we can put only 2 kg , in knapsack ,so we will put 2 kgs from W2 , hence profit from this 2kg of w2 is 3.2. Thus , total profit is 25+ 3.2 = 28.

2. Now , we will put W3 (i.e 10 KG) into knapsack , hence gained profit is 15 , but

we can put only 10 kg in knapsack , so we will put 10 kg form w2 , hence profit from this 10 kg of w2 is 16. Thus, total profit is 15+ 16 = 31

3. Now , we will put W2 (i.e 15 KG) into knapsack , hence gained profit is 24 , but we can put only 5 kg in knapsack , so we will put 5 kg form w3 , hence profit from this 5 kg of w3 is 7.5.
Thus, total profit is 24+ 7.5 = 31.5. Hence, optimal solution for given problem is 3rd solution, which is giving total profit as 31.5.

The knapsack problems are often solved using dynamic programming, though no polynomial-time algorithm is known. Both the knapsack problem(s) and the subset sum problem are NP-hard.

6. Algorithm :

1. Let 'W' be the maximum weight of the knapsack
2. Let 'w_i' and 'p_i' be the weight and profit of individual items i.e. for i=1……n
3. Calculate p_i / $w_i$ratio and arrange that in decreasing order.
4. initially weight=0 and profit = 0
5. for i=1 to n
   {
   add item in knapsack till weight <= W
   profit = profit + p_i
   }
6. Stop

7. Conclusion and Discussion :

The knapsack algorithm takes $\theta(nw)$ times, broken up as follows: $\theta(nw)$ times to fill the c-table, which has $(n+1).(w+1)$ entries, each requiring $\theta(1)$ time to compute. $O(n)$ time to trace the solution, because the tracing process starts in row n of the table and moves up 1 row at each step.

8. Viva Questions :

- Type of knapsack problem?
- Which strategy provides optimal solution Greedy or Dynamic programming?

9. References :

- http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Dynamic/knapsackdyn.htm
- http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Greedy/knapscakFrac.htm
- http://www.geeksforgeeks.org/dynamic-programming-set-10-0-1-knapsack-problem/

Design and Analysis of Algorithm Lab

Experiment No.: 05

Dijikstra’s Algorithm

Experiment No. 05

1. **Aim :** Write a program to find single source shortest path using Dijkstra's Algorithm

2. **Objectives :**
   - To provide mathematical approach for Analysis of Algorithms.
   - To calculate time complexity and space complexity.
   - To solve problems using various strategies.

3. **Outcome :**
   - Students will be able to create and apply the efficient algorithms for the effective problem solving with the help of different strategies like greedy method.

4. **Hardware / Software Required :** Turbo C

5. **Theory :**

Dijkstra's shortest path algorithm,a greedy algorithm that efficiently finds shortest paths in a graph. Dijkstra's algorithm solves the single-source shortest-path problem when all edges have non-negative weights. It is a greedy algorithm and similar to Prim's algorithm. Algorithm starts at the source vertex, s, it grows a tree, T, that ultimately spans all vertices reachable from S. Vertices are added to T in order of distance i.e., first S, then the vertex closest to S, then the next closest, and so on. Following implementation assumes that graph G is represented by adjacency lists.

6. **Algorithm :**

1. Let 'adj' be the adjacency matrix of graph 'G' having 'v' vertices numbered from 1 to v and having 'e' edges.
2. Let distance, path and visited be arrays of 'v' elements each.
3. Array 'distance' is initialized to ∞, while 'path' array and 'visited' array is initialized to 0.
4. Let current = source vertex.
5. visited[current]=1
6. T = 0
7. Let number of vertices already added to the trace be given as nv and let nv=1.
8. Repeat steps 9 to 13 while nv ≠ v.
9. for i= 1 to v

   if (adj[current][i] ≠ 0)

   if (visited[i] ≠ 1)

```
            if ( distance[i] >adj[current][i])
                {
                    distance[i] = adj[current][i] + T
                    path[i]=current
                }
```

10. min = ∞ (in program min=32767)

```
        for i= 1 to v
            if ( visited[i] ≠ 1)
              if ( distance[i] < min)
                {
                  min = distance[i]
                  current = i
                }
```

11. visited[current]=1
12. nv = nv+1
13. T = distance[current]
14. Let dest be the destination vertex

    Shortest distance from the source vertex to the destination vertex = distance[dest]

```
        y = dest
do
          {
            x = path[y]
            Display vertex i is connected to vertex x.
          }
```

15. while y ≠ sorce vertex repeat steps 2 to 15 for each vertex as the sorce vertex.
16. Stop.

7. Conclusion and Discussion :

An upper bound of the running time of Dijkstra's algorithm on a graph with edges E and vertices V can be expressed as a function of |E| and |V| using the Big-O notation. For sparse graphs, that is, graphs with fewer than |V|2 edges, Dijkstra's algorithm can be implemented more efficiently by storing the graph in the form of adjacency lists and using a binary heap, pairing heap, or Fibonacci heap as a priority

queue to implement the Extract-Min function efficiently. With a binary heap, the algorithm requires O((|E|+|V|) log |V|) time (which is dominated by O(|E| log |V|) assuming every vertex is connected, that is, $|E| \geq |V| - 1$), and the Fibonacci heap improves this to O(| E | + | V | log | V |).

8. Viva Questions :

- Does dijikstra is capable of handling negative weight edges?
- Which strategy dijikstra make use of, to find the shortest path?
- Given a directed weighted graph. You are also given the shortest path from a source vertex 's' to a destination vertex 't'. If weight of every edge is increased by 10 units, does the shortest path remain same in the modified graph?

9. References :

- http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/dijkstraAlgor.htm
- http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf
- http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/

Design and Analysis of Algorithm Lab

Experiment No. : 06

Huffman Coding

Experiment No. 06

1. **Aim :** Write a program to Implement Huffman Coding Algorithm.

2. **Objectives :**
   - To provide mathematical approach for Analysis of Algorithms.
   - To calculate time complexity and space complexity.
   - To solve problems using various strategies.

3. **Outcome :**
   - Students will be able to create and apply the efficient algorithms for the effective problem solving with the help of different strategies like greedy method.

4. **Hardware / Software Required :** Turbo C

5. **Theory :**

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters; lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.
The variable-length codes assigned to input characters are Prefix Codes means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not prefixed of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bit stream. Huffman's greedy algorithm uses a table giving how often each character occurs (i.e., its frequency) to build up an optimal way of representing each character as a binary string

It is important for an encoding scheme to be unambiguous. Since variable-length encodings are susceptible to ambiguity, care must be taken to generate a scheme where ambiguity is avoided. Huffman coding uses a greedy algorithm to build a prefix tree that optimizes the encoding scheme so that the most frequently used symbols have the shortest encoding. The prefix tree describing the encoding ensures that the code for any particular symbol is never a prefix of the bit string representing any other symbol. To determine the binary assignment for a symbol, make the leaves of the tree correspond to the symbols, and the assignment will be the path it takes to get from the root of the tree to that leaf.

The Huffman Coding algorithm takes in information about the frequencies or probabilities of a particular symbol occurring. It begins to build the prefix tree from the bottom up, starting with the two least probable symbols in the list. It takes those symbols and forms a sub tree containing them, and then removes the individual symbols from the list. The algorithm sums the probabilities of elements in a sub tree and adds the sub tree and its probability to the list. Next, the algorithm searches the list and selects the two

symbols or sub trees with the smallest probabilities. It uses those to make a new sub tree, removes the original sub trees/symbols from the list, and then adds the new sub tree and its combined probability to the list. This repeats until there is one tree and all elements have been added.

6. Algorithm :

1. n = |C|
2. Q = C
3. for i = 1 to n - 1
4. allocate a new node ´
5. z. left = x = EXTRACT-MIN(Q)
6. z. right = y = EXTRACT-MIN(Q)
7. z. freq = x.freq + y.freq
8. INSERT(Q,z)
9. return EXTRACT-MIN(Q)

7. Conclusion and Discussion:

Huffman coding is a technique used to compress files for transmission. Time complexity is O(n log n) where n is the number of unique characters.

8. Viva Questions:

- Explain Huffman coding and its usage?
- What is the time complexity of Huffman Coding?

9. References:

- http://www.personal.kent.edu/~rmuhamma/Algorithms/ Huffman Coding.htm
- http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf
- https://brilliant.org/wiki/huffman-encoding/
- https://www.geeksforgeeks.org/greedy-algorithms-set-3-huffman-coding/

Design and Analysis of Algorithm Lab

Experiment No. : 07

Bellman-Ford

Experiment No. 07

1. **Aim :** Write a program to implement a Single Source Shortest Path Algorithm (Bellman-Ford).
2. **Objectives :**
   - To provide mathematical approach for Analysis of Algorithms.
   - To calculate time complexity and space complexity.
   - To solve problems using various strategies.
3. **Outcome :**
   - Students will be able to apply dynamic programming strategy to solve different problems effectively.
4. **Hardware / Software Required :** Turbo C
5. **Theory :**

The **Bellman-Ford algorithm** is a graph search algorithm that finds the shortest path between a given source vertex and all other vertices in the graph. This algorithm can be used on both weighted and unweighted graphs. Like Dijikstra's shortest path algorithm, the Bellman-Ford algorithm is guaranteed to find the shortest path in a graph. Though it is slower than Dijikstra's algorithm, Bellman-Ford is capable of handling graphs that contain negative edge weights, so it is more versatile. Going around the negative cycle an infinite number of times would continue to decrease the cost of the path (even though the path length is increasing). Because of this, Bellman-Ford can also detect negative cycles which are a useful feature.

Bellman-Ford algorithm returns a Boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source. If there is such a cycle, the algorithm indicates that no solution exists. If there is no such cycle, the algorithm produces the shortest paths and their weights. The algorithm relaxes edges, progressively decreasing an estimated on the weight of a shortest path from the source to each vertex until it achieves the actual shortest-path weight. The algorithm returns TRUE if and only if the graph contains no negative-weight cycles that are reachable from the source.

6. **Algorithm :**

BELLMAN-FORD (G,w,s)

1. INITIALIZE-SINGLE-SOURCE (G, s)
2. for i = 1 to |V| - 1
3. for each edge (u, v) ∈ G
4. RELAX (u, v, w)

5. for each edge (u, v) ∈ G
6. if v.d > u.d + w(u, v)
7. return FALSE
8. return TRUE

INITIALIZE-SINGLE-SOURCE (G, s)

1. for each vertex v ∈ G.V
2. v.d = ∞
3. v.pi = NIL
4. s.d = 0

RELAX (u, v, w)

1. if v.d > u.d + w(u, v)
2. v.d = u.d + w(u, v)
3. v.pi = u

7. Conclusion and Discussion :

Bellman-Ford is used to find single source shortest path, simpler than Dijkstra. It work for Graphs with negative weight edges and suites well for distributed systems. But time complexity of Bellman-Ford is O(VE), which is more than Dijkstra.

8. Viva Questions :

- What is meant by Shortest Path?
- What is time and space complexity of algorithm?
- Compare Dijkstra's and Bellman-Ford algorithm.

9. References :

- http://www.personal.kent.edu/~rmuhamma/Algorithms/Single Source Shortest Path Algorithm /Bellman-Ford.htm
- http://quiz.geeksforgeeks.org/Single Source Shortest Path Algorithm /Bellman-Ford
- http://interactivepython.org/runestone/static/pythonds/ Single Source Shortest Path Algorithm /Bellman-Ford.html
- https://www.khanacademy.org/computing/computer-science/algorithms/Single Source Shortest Path Algorithm /Bellman-Ford.

Design and Analysis of Algorithm Lab

Experiment No. : 08

Floyd Warshall

Experiment No. 08

1. **Aim :** Write a program to implement a All Pair Shortest Path Algorithm (Floyd Warshall).

2. **Objectives :**
   - To provide mathematical approach for Analysis of Algorithms.
   - To calculate time complexity and space complexity.
   - To solve problems using various strategies.

3. **Outcome :**
   - Students will be able to apply dynamic programming strategy to solve different problems effectively.

4. **Hardware / Software Required :** Turbo C

5. **Theory :**

The Floyd-Warshall algorithm is a shortest path algorithm for graphs. Like the Bellman-Ford algorithm or the Dijkstra's algorithm, it computes the shortest path in a graph. They only compute the shortest path from a single source and Floyd-Warshall, on the other hand, computes the shortest distances between every pair of vertices in the input graph.

The Floyd-Warshall algorithm is an example of dynamic programming. It breaks the problem down into smaller sub problems, and then combines the answers to those sub problems to solve the big, initial problem. The idea is this: either the quickest path from A to C is the quickest path found so far from A to C, or it's the quickest path from A to B plus the quickest path from B to C.

Floyd-Warshall is extremely useful in networking, similar to solutions to the shortest path problem. However, it is more effective at managing multiple stops on the route because it can calculate the shortest paths between all relevant nodes. In fact, one run of Floyd-Warshall can give you all the information you need to know about a static network to optimize most types of paths. It is also useful in computing matrix inversions.

6. **Algorithm :**

1. Create a |V| x |V| matrix, M, that will describe the distances between vertices
2. For each cell (i, j) in M

```
if i == j
    M[i][j] = 0
if (i, j) is an edge in E:
```

```
            M[i][j] = weight(i, j)
        else
            M[i][j] = infinity
    3. for k from 1 to |V|
        for i from 1 to |V|
          for j from 1 to |V|
              if M[i][j] > M[i][k] + M[k][j]
                 M[i][j] = M[i][k] + M[k][j]
```

7. Conclusion and Discussion :

The algorithm consists of three loops over all nodes, and the most inner loop contains only operations of a constant complexity. Hence the asymptotic complexity of the whole Floyd-Warshall algorithm is $O(N^3)$, where N is number of nodes of the graph.

8. Viva Questions :

- What is Time Complexity of Floyd Warshall?
- What is different All Pair Shortest Path Algorithm?
- Diffrentiate: Dijkstra's and Floyd–Warshall algorithm.

9. References :

- http://www.personal.kent.edu/~rmuhamma/All Pair Shortest Path Algorithm/ Floyd Warshall
- http://quiz.geeksforgeeks.org/ All Pair Shortest Path Algorithm/ Floyd Warshall
- https://www.khanacademy.org/computing/computer-science/All Pair Shortest Path Algorithm/ Floyd Warshall

Design and Analysis of Algorithm Lab

Experiment No.: 09

Longest Common Subsequence

Experiment No. 09

1. **Aim :** Write a program to find Longest Common Subsequence from the given two sequences.

2. **Objectives :**
   - To provide mathematical approach for Analysis of Algorithms.
   - To calculate time complexity and space complexity.
   - To solve problems using various strategies.

3. **Outcome :**
   - Students will be able to apply dynamic programming strategy to solve different problems effectively.

4. **Hardware / Software Required :** Turbo C

5. **Theory :**

A Longest Common Subsequence is a common subsequence of minimal length. In the longest common subsequence problem we are given two sequences X=<x1, x2…xm> and Y=<y1,y2…yn> and wish to find a maximum-length common subsequence of x and y.

The optimal substructure of the LCS problem gives the recursive formula,

0	if i=0 or j=0
c[i,j]= c[i-1,j-1]+1	if i,j>0 & xi=yj
max(c [i, j-1], c[i-1,j])	if i,j>0 &xi≠yj

Computing the length of an LCS :

Procedure LCS-LENGTH takes two sequences X=<x1, x2… xm> and Y=<y1, y2… yn> as inputs. It stores the c[i,j] values in a table c[0…m,0…n] whose entries are computed in row-major order i.e. the first row of c is filled in form left to right, then second row and so on. It also maintains the table b [1…m, 1…n] to simplify construction of an optimal solution. Initially, b[i,j] points to the table entry corresponding to the optimal sub problem solution. The procedure returns the b and c tables; c[m,n] contains the length of an LCS of X and Y.

Constructing an LCS :

The b table returned by LCS_LENGTH can be used to quickly construct an LCS of X=<x1, x2… xm> and Y=<y1, y2… yn>. We begin at b[m, n] and trace through the table following the arrows. Whenever we encounter a “ ” in entry b[i, j] it implies that xi=yj is an element of the LCS. The elements of the LCS are encountered in reverse order by this method.

6. **Algorithm :**

LCS - Length (X , Y)

1. m = length [X]
2. n = length (Y)
3. for i= 1 to m
4. do c [i , 0]= 0
5. for j= 1 to n
6. do c [0 , j] = 0
7. for i = 1 to m
8. do for j = 1 to n
9. do if $x_i = y_i$
10. then c [i , j] = c [i – 1 , j – 1] + 1
11. b [i , j] = “ “
12. else if c [i – 1 , j] >= c [i , j – 1]
13. then c [i , j] = c [i – 1 , j]
14. b [i , j] = “ “
15. else c [i , j] = c [i , j - 1]
16. b [i , j] = “ “
17. return c and b

PRINT-LCS (b, x, i, j)

1. if i = 0 or j = 0
2. then return
3. if b [i , j] = ”
4. then PRINT – LCS (b , x , i – 1 , j - 1)
5. print x_i
6. else if b [i , j] = ”
7. then PRINT – LCS (b , x , i – 1 , j)
8. Else PRINT – LCS (b , x , i , j – 1)

7. Conclusion and Discussion :

In compilation of program, in software design or in system design text processing is a vital activity. While processing the text, string matching is an important activity which is needed most of the time. Least common sub sequences is one of the pattern matching algorithm. This algorithm is more efficient than other algorithm and hence implemented successfully..

8. Viva Questions :

.

- Find the LCS for LCS for input Sequences “ABCDGH” and “AEDFHR”?
- Find the LCS for LCS for input Sequences “AGGTAB” and “GXTXAYB”?
- Complexity of LCS using Dynamic programming?
- Complexity of LCS using naïve recursive approach?

9. References :

- https://www.hackerrank.com/challenges/dynamic-programming-classics-the-longest-common-subsequence
- http://wordaligned.org/articles/longest-common-subsequence
- http://www.geeksforgeeks.org/dynamic-programming-set-4-longest-common-subsequence/

Design and Analysis of Algorithm Lab

Experiment No.: 10

8 Queen’s Problem

Experiment No. 10

1. **Aim :** Write a program to implement 8 Queen's Problem.

2. **Objectives :**
   - To provide mathematical approach for Analysis of Algorithms.
   - To calculate time complexity and space complexity.
   - To solve problems using various strategies.

3. **Outcome :**
   - Students will be able to create and apply backtracking, branch and bound and string matching techniques to deal with some hard problems.

4. **Hardware / Software Required :** Turbo C

5. **Theory:**

The eight queens puzzle is the problem of putting eight chess queens on an 8×8 chessboard such that none of them is able to capture any other using the standard chess queen's moves. The queens must be placed in such a way that no two queens would be able to attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general n queens puzzle of placing n queens on an n×n chessboard, where solutions exist only for $n = 1$ or $n \geq 4$.

The N queen's problem solved using the recursive approach, where in the placement of the queen is decided by every recursive call to the function that decides queen's placement at every level . The main care has to be taken to place the queen is that :

1. No queen can be placed in the immediate vicinity of the queen at preceding level.
2. No queens should be on same diagonal.

The problem can be quite computationally expensive as there are 4,426,165,368 or 64! / (56!8!)) possible arrangements of eight queens on the board, but only 92 solutions. For example, just by applying a simple rule that constrains each queen to a single column (or row), it is possible to reduce the number of possibilities to just 16,777,216 (8^8) possible combinations, which is computationally manageable for n=8, but would be intractable for problems of n=1,000,000. This heuristic solves n queens for any n, $n \geq 4$ or $n = 1$:

Steps :

1. Divide n by 12. Remember the remainder (n is 8 for the eight queens puzzle).
2. Write a list of the even numbers from 2 to n in order.
3. If the remainder is 3 or 9, move 2 to the end of the list.

4. Append the odd numbers from 1 to n in order, but, if the remainder is 8, switch pairs (i.e. 3, 1, 7, 5, 11, 9, …).
5. If the remainder is 2, switch the places of 1 and 3, then move 5 to the end of the list.
6. If the remainder is 3 or 9, move 1 and 3 to the end of the list.
7. Place the first-column queen in the row with the first number in the list, place the second-column queen in the row with the second number in the list, etc.

For n = 8 this results in the solution shown above. A few more examples follow.

- 14 queens (remainder 2): 2, 4, 6, 8, 10, 12, 14, 3, 1, 7, 9, 11, 13, 5.
- 15 queens (remainder 3): 4, 6, 8, 10, 12, 14, 2, 5, 7, 9, 11, 13, 15, 1, 3.
- 20 queens (remainder 8): 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 3, 1, 7, 5, 11, 9, 15, 13, 19, 17.

6. Algorithm :

Algorithm Queen (n)
// Input : Total number of Queen's n

```
for column =1 to n do
 {
if(place(row,column)) then
        {
board[row]=column // no conflict so place queen
if(row==n) then  //dead end
print_board(n) // printing the board configuration
else // try next queen with next position
Queen(row+1,n);
         }
 }
```

Algorithm place (row, column)
// This algorithm is for placing the queen at appropriate position
// Input : row and column of the chessboard

```
fori□ 1 to row-1 do
{
   // checking for column and diadonal conflicts
```

```
if ( board[i] = column ) then
return 0;
else if ( abs(board[i] – column) = abs(i-row)) then
return 0;
 }
 // no conflicts hence Queen can be placed
return 1;
```

7. Conclusion and Discussion :

The backtracking technique, the eight queen's problem is to place the eight queen's checks against any other queen. The backtracking is applied till optimal solution is reached**.**

8. Viva Questions:

- Which strategy N-queen problem uses to find solution?
- How many solutions exist for 8-queen problem?
- Draw the 4X4 board and place a 4-queens using backtracking step by step.

9. References:

- http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/
- http://algorithms.tutorialhorizon.com/backtracking-n-queens-problem/
- https://developers.google.com/optimization/puzzles/queens

Design and Analysis of Algorithm Lab

Experiment No.: 11

Naïve String Matching Algorithm

Experiment No. 11

1. **Aim:** Write a program to implement Naïve String Matching Algorithm**.**

2. **Objectives:**

- To analyze strategies for solving problems not solvable in polynomial time.
- To improve the logical ability

3. **Outcomes:**
   - Students will be able to create and apply backtracking, branch and bound and string matching techniques to deal with some hard problems.

4. **Hardware / Software Required :** Turbo C

5. **Theory:**

In the Naive String matching algorithm, we always slide the pattern by 1. When all characters of pattern are different, we can slide the pattern by more than 1. When a mismatch occurs after j matches, we know that the first character of pattern will not match the j matched characters because all characters of pattern are different. So we can always slide the pattern by j without missing any valid shifts. The naïve approach simply test all the possible placement of Pattern P[1 . . m] relative to text T[1 . . n].

The naïve string-matching procedure can be interpreted graphically as a sliding a pattern P[1 . . m] over the text T[1 . . n] and noting for which shift all of the characters in the pattern match the corresponding characters in the text.

6. **Algorithm** :

NAÏVE_STRING_MATCHER (T, P)
1. n ← length [T]
2. m ← length [P]
3. for s ← 0 to n - m do
4. if P[1 . . m] = T[s +1 . . s + m]
5. then return valid shift s

NAÏVE_STRING_MATCHER (T, P)
1. n ← length [T]
2. m ← length [P]
3. for s ← 0 to n-m do
4. j ← 1
5. while j ≤ m and T[s + j] = P[j] do

6. j ← j +1
7. If j > m then
8. return valid shift s
9. return no valid shift exist // i.e., there is no substring of T matching P.

7. Conclusion and Discussion :

The running time of the algorithm is O((n - m +1)m), which is clearly O(nm). Hence, in the worst case, when the length of the pattern, m are roughly equal, this algorithm runs in the quadratic time. One worst case is that text, T, has n number of A's and the pattern, P, has (m -1) number of A's followed by a single B.

8. Viva Questions :

- What do you mean by string?
- What is time complexity of naive string matching algorithm?
- What are different types of string matching algorithm?

9. References:

- https://www.geeksforgeeks.org/pattern-searching-set-4-a-naive-string-matching-algo-question
- http://www.idconline.com/technical_references/pdfs/information_technology/String_Matching_Algorithms.pdf
- http://www.student.montefiore.ulg.ac.be/~s091678/files/OHJ2906_Project.pdf