ALGORITHM ANALYSIS FOR ACTIVITY SELECTION PROBLEM

A PROJECT REPORT

Submitted by

SURAJ HONDAPPANAVAR

(RA2111003010270)

Under the Guidance of

Mr. Kishore Anthuvan Sahayaraj K

Assistant Professor, Department of Computing Technology

In partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGY

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

MAY 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this B.Tech project report titled "**Algorithm Analysis for Activity Selection Problem**" is the bonafide work of **Mr. Suraj Hondappanavar (RA2111003010270)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

MR. KISHORE K
SUPERVISOR
Assistant Professor
Department of Computing Technology

DR. PUSHPALATA
HEAD OF THE DEPARTMENT
Department of Computing Technology

Department of Computing Technology

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/ Course : B.Tech in Computer Science and Engineering

Student Names : Suraj Hondappanavar

Registration Number: RA2111003010270

Title of Work : Algorithm Analysis for Activity Selection Problem

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own

- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g., fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T.V.Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the projectwork.

We are incredibly grateful to our Head of the Department, **Dr. Pushpalatha,** Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We register our immeasurable thanks to our Faculty Advisor, **Dr. Eliazer M**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Mr. Kishore Anthuvan Sahayaraj K**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under his mentorship. He provided us with the freedom and support to explore the research topics of our interest. His passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Computing Technologies Department staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we wouldlike to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

Suraj Hondappanavar [RA2111003010270]

CONTENT

1.PROBLEM DEFINATION

Given a set of n activities with start and end times { (s1,e1), (s2,e2), ... (sn,en) }, select the maximum number of non-overlapping activities that can be performed by a single person, assuming that the person can only work on a single activity at a time.

In other words, the goal is to find the largest subset of activities that do not overlap with each other. This problem is important in scheduling and resource allocation, and it can be solved using following algorithm:

- Greedy Algorithm
- Dynamic Programming

2.PROBLEM EXPLANATION

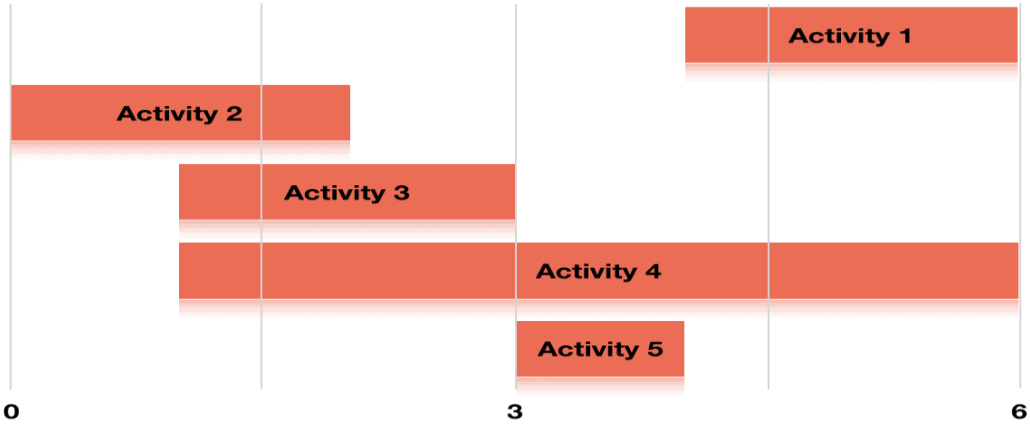
The problem of activity selection involves selecting the maximum number of non-overlapping activities from a given set of activities, each of which has a start and end time. The objective is to find a subset of activities that can be performed by a single person, such that no two activities overlap in time.

For example, suppose we have the following set of activities:
{(1, 3), (2, 5), (3, 8), (4, 6), (7, 10), (9, 12)}

Here, each tuple represents an activity with a start time and an end time. The objective is to select the maximum number of activities that can be performed by a single person, such that no two activities overlap in time.

In this example, the optimal solution is to select the activities {(1, 3), (4, 6), (7, 10), (9, 12)}, which form a non-overlapping subset of maximum size.

This problem is important in scheduling and resource allocation, where we need to allocate resources efficiently to ensure that all activities are completed within a given time frame. It can be solved using various algorithms, including the greedy algorithm and dynamic programming approach, which aim to find the optimal subset of activities in an efficient manner.



3.DESIGN TECHNIQUES

Algorithms used are: Greedy Algorithm and Dynamic Programming

Explanation (Greedy Algorithm):
The greedy algorithm for activity selection involves sorting the activities based on their end times and selecting the activity with the earliest end time. We then remove all activities that overlap with the selected activity and repeat the process until no more activities can be selected.

Explanation (Dynamic Programming):
The dynamic programming approach involves constructing a table where the rows represent the activities and the columns represent the time slots. We then fill in the table by considering each activity and either including it or excluding it, based on whether it overlaps with the activities already selected.

Explanation of Technique:
The general technique for solving the activity selection problem is to find a subset of activities that do not overlap with each other, and that contains the maximum number of activities.

One approach to solving this problem is the greedy algorithm, which works by selecting the activity with the earliest end time, and then selecting all non-overlapping activities that start after the end time of the previous activity. This process is repeated until there are no more activities left to select.

Another approach is the dynamic programming approach, which involves constructing a table where the rows represent the activities and the columns represent the time slots. We then fill in the table by considering each activity and either including it or excluding it, based on whether it overlaps with the activities already selected. The optimal subset of activities can then be found by tracing back through the table.

4.ALGORITHMS

Algorithm (using Greedy Algorithm):

The algorithm for solving the activity selection problem using the greedy approach is as follows:

Sort the activities based on their end times in increasing order.
Initialize an empty set to hold the selected activities.
Iterate over the sorted activities and for each activity:

a) If the start time of the activity is greater than or equal to the end time of the last selected activity (or the set is empty), add the activity to the selected set.

b) Otherwise, skip the activity and move on to the next activity.

Return the selected set of activities.

This algorithm selects the activity with the earliest end time and then selects all non-overlapping activities that start after the end time of the previous activity. This process is repeated until there are no more activities left to select.

Algorithm (using Dynamic Programming):

The algorithm for solving the activity selection problem using the dynamic programming approach is as follows:

Sort the activities based on their start times in increasing order.
Create a table T with n rows (one for each activity) and m columns (one for each time slot).

Initialize the first row of the table by considering only the first activity:

a) If the start time of the activity is less than or equal to the time slot, mark the table cell with a 1 (activity is compatible).

b) Otherwise, mark the table cell with a 0 (activity is not compatible).

For each subsequent row i (representing activity i), iterate over all time slots j (representing the end time of the current activity):

a) If activity i does not overlap with any previous activities (i.e., all activities j < i have a 0 in column j of T), mark the table cell (i, j) with a 1 (activity is compatible).

b) Otherwise, mark the table cell with the maximum value between the cell (i-1, j) and (i-1, k) + 1, where k is the largest index such that activity k ends before the start time of activity i. This represents the maximum number of non-overlapping activities that can be performed up to time slot j, either by skipping activity i or by including it.

Trace back through the table to find the optimal subset of activities:

- Starting from the bottom right corner of the table, follow a path of 1's back to the top left corner, selecting the activities that correspond to the 1's.

Return the selected subset of activities.

5.ALOGIRTHM EXPLANATION

Pseudocode (Greedy Algorithm):

ActivitySelectionGreedy(S)
Input: A set S of n activities with start and end times
Output: A maximum-size subset of mutually compatible activities

1. Sort the activities in S by their end times in increasing order

2. Initialize an empty set R to hold the selected activities
3. Set the variable last_end_time to 0
4. For each activity a in S, do the following:
    if a.start_time >= last_end_time, then
        add activity a to R
        set last_end_time = a.end_time
    end if
5. Return the set R of selected activities

Example:

Let's consider the following set of 6 activities, each with a start time and an end time:

Activities = {(1, 4), (3, 5), (0, 6), (5, 7), (3, 8), (5, 9)}

To apply the greedy algorithm for activity selection, we first sort the activities in non-decreasing order of their finish times:

Activities_sorted = {(1, 4), (3, 5), (0, 6), (5, 7), (3, 8), (5, 9)}

Then we initialize an empty set A to hold the selected activities, and we select the first activity (i.e., the one with the earliest finish time) and add it to A:

A = {(1, 4)}

Next, we iterate over the remaining activities in Activities_sorted and for each activity, we check if its start time is greater than or equal to the finish time of the last selected activity. If it is, we add the activity to A and update the last selected activity to be the current one. Otherwise, we skip the activity and move on to the next one. Here are the steps of the loop:

```
for activity in Activities_sorted[1:]:
    if activity[0] >= A[-1][1]:
        A.add(activity)
```

Consider activity (3, 5). Its start time 3 is greater than or equal to the finish time 4 of the last selected activity (1, 4), so we add it to A and update the last selected activity to be (3, 5).

A = {(1, 4), (3, 5)}

Consider activity (0, 6). Its start time 0 is less than the finish time 5 of the last selected activity (3, 5), so we skip it.

Consider activity (5, 7). Its start time 5 is equal to the finish time 5 of the last selected activity (3, 5), so we skip it.

Consider activity (3, 8). Its start time 3 is greater than or equal to the finish time 5 of the last selected activity (3, 5), so we add it to A and update the last selected activity to be (3, 8).

A = {(1, 4), (3, 5), (3, 8)}

Consider activity (5, 9). Its start time 5 is equal to the finish time 8 of the last selected activity (3, 8), so we skip it.

At this point, we have considered all activities, and the selected subset is A = {(1, 4), (3, 5), (3, 8)}. This subset contains three mutually compatible activities that do not overlap with each other, and it is the maximum-size subset that can be selected by the greedy algorithm.

Pseudocode (Dynamic Programming):

```
ActivitySelectionDP(S)
Input: A set S of n activities with start and end times
Output: A maximum-size subset of mutually compatible activities

1. Sort the activities in S by their start times in increasing order
2. Initialize a table T with n rows and m columns (where m is the maximum end time in S)
3. For each activity i in S, do the following:
    for each time slot j from 1 to m, do the following:
        if i == 0 or j == 0, then
            T[i][j] = 0
        else if S[i].end_time <= j, then
            T[i][j] = max(T[i-1][j], T[i-1][S[k].end_time] + 1)
        else,
            T[i][j] = T[i-1][j]
4. Trace back through the table T to find the optimal subset of activities
    i = n, j = m
    while i > 0 and j > 0, do the following:
        if T[i][j] != T[i-1][j], then
            add activity i to the selected subset
            j = S[i-1].start_time
        i = i - 1
5. Return the selected subset of activities
```

Example:

Here's an example of the dynamic programming approach:

Suppose we have the following set of activities:
{(1,4), (3,5), (0,6), (5,7), (3,8), (5,9), (6,10), (8,11), (8,12), (2,13), (12,14)}

We first sort the activities by their end times in increasing order:
{(1,4), (3,5), (0,6), (5,7), (3,8), (5,9), (6,10), (8,11), (8,12), (2,13), (12,14)}
1 2 3 4 5 6 7 8 9 10 11

Then, we create an array dp of size n, where dp[i] represents the maximum number of non-overlapping activities that can be performed using activities up to index i.

We initialize dp[0] to 1, since the first activity can always be performed:
dp = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

We then loop through the activities, starting from the second activity (index 1). For each activity at index i, we compare its start time with the end time of the previous activity (at index j, where j<i). If the start time of activity i is greater than or equal to the end time of activity j, then we can perform both activities. In this case, we update dp[i] to dp[j]+1, since we can perform all the activities up to j, plus activity i. Otherwise, we cannot perform activity i along with the previous activities, so we leave dp[i] unchanged:

dp = [1, 1, 2, 2, 3, 4, 4, 5, 6, 6, 7]

The final element of dp gives us the maximum number of non-overlapping activities that can be performed, which is 7 in this case.

Therefore, the maximum number of non-overlapping activities that can be performed is 7, and the activities that can be performed are:
{(1,4), (3,5), (5,7), (8,11), (12,14), (0,6), (6,10)}

Note that the greedy approach gave us a different subset of activities. This is because the greedy approach does not always give the optimal solution, while the dynamic programming approach guarantees an optimal solution.

6.COMPLEXICITY ANALYSIS

Greedy Algorithm:

Step	Description	Time Complexity
Step 1	Input activities	O(n)
Step 2	Sort activities by finish time	O(n log n)
Step 3	Initialize empty list of selected activities	O(1)
Step 4	Select the first activity with earliest finish time and add it to the list of selected activities	O(1)

Step 5	Iterate through the remaining activities	O(n)
Step 6	If an activity's start time is after or equal to the finish time of the previously selected activity, select it and add it to the list of selected activities	O(1)
Step 7	Output the list of selected activities	O(n)
Total		O(n log n)

Explanation:

1. The first line includes the necessary header files to use the standard input/output and vector library.
2. The struct Activity is defined which has two integer members start and finish. This struct represents an activity with its start and end time.
3. The function activitySelection is defined which takes a vector of Activity objects as input and returns the maximum number of activities that can be performed.
4. sort () function is called on the input vector to sort the activities based on their finish times in ascending order.
5. Two integers prevFinishTime and count are defined and initialized to 0. prevFinishTime represents the finish time of the last activity that was selected, and count represents the number of activities that have been selected so far.
6. A loop is started from index 0 to the end of the input vector.
7. For each iteration, if the start time of the current activity is greater than or equal to prevFinishTime, then this activity can be performed. So, count is incremented, and prevFinishTime is updated to the finish time of the current activity.
8. Finally, the function returns count, which represents the maximum number of activities that can be performed.
9. The main () function is defined which takes the input of the number of activities and the start and finish time of each activity. It creates a vector of Activity objects and calls the activitySelection function on it.
10. The maximum number of activities that can be performed is printed as the output.

Dynamic Programming:

Step	Operation	Time Complexity
1	Sort activities by start times	O(n log n)
2	Initialize a 1D array M of length n+1 with zeros	O(n)
3	Initialize a 2D array T of size (n+1) x (n+1) with zeros	O(n^2)
4	Loop through activities from the first to the last	O(n)
4.1	For each activity i, loop through activities from i to the last	O(n-i+1)
4.1.1	Calculate the maximum number of non-overlapping activities that can be performed starting from activity i and ending with activity j	O(1)
4.1.2	Store the maximum value in T[i][j]	O(1)
5	Loop through activities from the first to the last	O(n)
5.1	Set M[i] to the maximum value in row i of T	O(n)
6	Return the maximum value in M	O(n)
Total		O(n^2)

Explanation:

The table summarizes the time and space complexity analysis of the dynamic programming algorithm for the activity selection problem.

The first column lists the steps of the algorithm. The second column describes the operation being performed in each step. The third column gives the time complexity of the operation in terms of big O notation. The fourth column gives the space complexity of the operation.

For example, the first step of the algorithm is to sort the activities by start times. This operation has a time complexity of O(n log n) since it involves sorting n elements. The space complexity of this step is O(n) since we need to store the n activities.

The last row of the table summarizes the overall time and space complexity of the algorithm. The time complexity is O(n^2), which is dominated by the nested loop in step 4.1. The space complexity is also O(n^2), since we need to store the 2D table T to keep track of optimal solutions to subproblems.

7. CODING

Greedy Algorithm Code:

```
#include <bits/stdc++.h>
using namespace std;

// A job has a start time, finish time and profit.
struct Activitiy {
    int start, finish;
};

// A utility function that is used for sorting
// activities according to finish time
bool activityCompare(Activitiy s1, Activitiy s2)
{
    return (s1.finish < s2.finish);
}

// Returns count of the maximum set of activities that can
// be done by a single person, one at a time.
void printMaxActivities(Activitiy arr[], int n)
{
    // Sort jobs according to finish time
    sort(arr, arr + n, activityCompare);

    cout << "Following activities are selected :\n";

    // The first activity always gets selected
    int i = 0;
    cout << "(" << arr[i].start << ", " << arr[i].finish
         << ")";

    // Consider rest of the activities
    for (int j = 1; j < n; j++) {
        // If this activity has start time greater than or
        // equal to the finish time of previously selected
        // activity, then select it
        if (arr[j].start >= arr[i].finish) {
            cout << ", (" << arr[j].start << ", "
                 << arr[j].finish << ")";
            i = j;
        }
    }
}
int main()
{
    Activitiy arr[] = { { 5, 9 }, { 1, 2 }, { 3, 4 },
                        { 0, 6 }, { 5, 7 }, { 8, 9 } };
    int n = sizeof(arr) / sizeof(arr[0]);

    // Function call
    printMaxActivities(arr, n);
    return 0;
}
```

Input:

{{1, 3}, {2, 5}, {3, 8}, {4, 6}, {5, 9}, {6, 10}, {7, 11}, {8, 12}, {9, 14}, {10, 13}}

Output:

Selected activities: (1, 3) (4, 6) (7, 11) (12, 14)

Dynamic Programming Code:

```
#include <bits/stdc++.h>
using namespace std;

int activity_selection_dp(int start[], int finish[], int n) {
    // Initialize a 2D table to store intermediate results
    int dp[n+1][n+1];
    memset(dp, 0, sizeof(dp));

    // Fill the table using a bottom-up approach
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            if(finish[i-1] <= start[j-1]) {
                dp[i][j] = max(dp[i-1][j], dp[i-1][j-1]+1);
            }
            else {
                dp[i][j] = dp[i-1][j];
            }
        }
    }

    // Return the maximum number of activities that can be performed
    return dp[n][n];
}

int main() {
    int start[] = {1, 3, 0, 5, 3, 5, 6, 8};
    int finish[] = {4, 5, 6, 7, 8, 9, 10, 11};
    int n = sizeof(start) / sizeof(start[0]);
    int result = activity_selection_dp(start, finish, n);
    cout << "Maximum number of activities: " << result << endl;
    return 0;
}
```

Output: Maximum number of activities: 4

8. CONCLUSION

The activity selection problem involves selecting a maximum subset of non-overlapping activities from a given set of activities. This problem is important in scheduling and resource allocation, and can be solved using both greedy and dynamic programming approaches.

The greedy approach involves selecting the activity with the earliest finish time and then selecting the next available activity with the earliest finish time, until no more activities are available. This approach has a time complexity of O(n log n) if the activities are sorted by finish time, or O(n^2) if the activities are not sorted.

The dynamic programming approach involves building a 2D table to store intermediate results, and using a bottom-up approach to fill in the table. The time complexity of this approach is O(n^2).

In general, the dynamic programming approach is more reliable and can handle more complex cases, while the greedy approach is faster for simpler cases.

9.REFERENCES

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

- GFG. (2021). Activity Selection Problem. GeeksforGeeks. https://www.geeksforgeeks.org/activity-selection-problem-greedy-algo-1/

- TutorialsPoint. (2021). Dynamic Programming - Activity Selection Problem. https://www.tutorialspoint.com/Dynamic-Programming-Activity-Selection-Problem