# Friends Pairing Problem

Lekhana Mitta
IIT2019204

Sanskar Patro
IIT2019205

Aamin Chaudhari
IIT2019206

***Abstract*—For a given set of friends n each single can remain single or can be paired up with some other friend . Each friend can be paired only once . We have to solve for finding the no. of ways in which friends can remain single or can be paired up using Dynamic Programming approach.Using Dynamic Programming approach,the algorithm achieves its best time complexity of O(n),thus reducing the time to a much significant extent .**

***Index Terms*—Recursion,Dynamic Programming**

## I. Introduction

This document describes the procedure followed to find the no. of ways in which given n no. of friends can remain single or can be paired using dynamic programming.

## II. Algorithm Design

### *A. Dynamic Programming*

Dynamic programming is mainly used in optimising solutions.Similarly like Divide and Conquer, Dynamic Programming also solves problems by combining solutions of sub-problems. Unlike Recursion it stores solutions of sub-problems dynamically for later use to avoid solving the sub-problems again and again.

We prefer solving a problem using Dynamic programming if we identify Overlapping Sub-problems and optimal substructure of its sub-problems.This approach reduces time complexity to linear from exponential by storing solutions of sub-problems to avoid solving sub-problems again and again.

### *B. Dynamic Programming Components*

This technique can be categorised into these steps:

Stages - Analyse and identify the structure of an optimal solution of the problem and its sub problems.Each sub-problem can be called as stage

States - Each stage has many states associated with it. A state of the problem usually represents a sub-solution, i.e. a partial solution or a solution based on a subset of the given input. And the states are built one by one, based on the previously built states.

Decision - At each stage, there can be multiple choices. From these choices , we can choose best decisions according to the problems.The decision taken at every stage should be best according to the problem , this is called as stage decision.

Optimal Policy - This determines the decision at each stage , a rule is called an optimal policy if it is globally optimal.

### *C. Steps to solve Dynamic Programming Problems*

This technique uses four steps :

Step-1 : Identify the problem as dynamic problem , by checking the properties i.e; Overlapping Sub-problems and optimal substructure.

Step-2 : Decide a state expression containing least number of parameters.

Step-3 : Formulate the relationship between the states.

Step-4 : Add tabulation or memoization to store the values of the sub-problems which have already been solved.

## III. Algorithmic Analysis

### *A. Algorithmic Steps for calculating Friends Pairing Problem. :*

To find the no. of ways given n friends are paired or remain single:

1) we input the no. of test cases and store it in variable t.
2) For every test case we use random number generation function to input the no. of friends for which we have to find the no. of ways they remain single or paired.
3) For every such input of t the rand() function returns a value which is stored in n, and passed as an argument to the function numOfCombos.
4) This function, we initialise a DP table locally.Here,we develop a bottom-up solution to find the no. of ways the friends remain single or paired up.
5) Iterate the value of i from 0 to n.If the value of i is less than or equal to 0,then store its value at index of i in DP table and return its value.
6) The above step is basically a base case if there are no friends or only 1 friend or 2 friends where no. of ways is possibly equal to no. of friends.
7) If the value of i is greater than 2 then there are two choices:
   (i) if i-th friend remains single , then we have to recur for i-1 friends which is basically finding the no. of ways i-1 friends are paired or remained single, it is given by value at index i-1 of DP table.
   (ii) The i-th friend pairs up with any of the i-1 friends then we solve it by finding number of ways a friend from i-1 friends is selected and no. of ways the remaining i-2

friends are paired up or remained single(i.e., the i-2 th index of DP table).

8) The equation for this explanation is DP[i] = DP[i-1]+DP[i-2]*(i-1).
9) And finally return the value at index i of DP table.
   - Similarly,it follows for every test case.This way can find the no. of ways in which n friends are paired up or remained single.

## IV. Experimental Analysis

Algorithmic Picturisation :

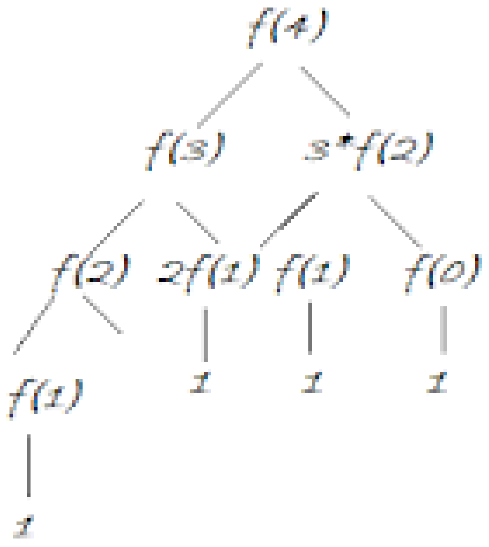


**Figure 1:** Example

## V. Pseudo Code

DP[n] stores the solutions of the sub-problems calculated . n is the variable which stores the no. of friends for every test case and DP[n-1] stores the result.

```
Main Function:
    int t
    t <- no. of test cases
    while t > 0
        int n
        n <- no. of friends
        print numOfCombos(n)
        print '\0'
numOfCombos Function:
    for i <- 0 to n
        initialise long long int DP[i] <- 0
    for i <- 0 to n
        if i <= 2
            DP[i] <- i
        else
            DP[i] <- DP[i-1]+DP[i-2]*(i-1)
    return DP[n]
```

## VI. Priori Analysis of friends pairing

This section explains Priori Analysis of finding the no. of ways where n friends are single or paired up.

### A. Time Complexity Analysis

We can observe that, we just used two for loops ,in which one for loop is used for initialising DP and other for storing solutions of DP.Hence,complexity is calculated as O(n) where n is no. of friends.

### B. Space Complexity

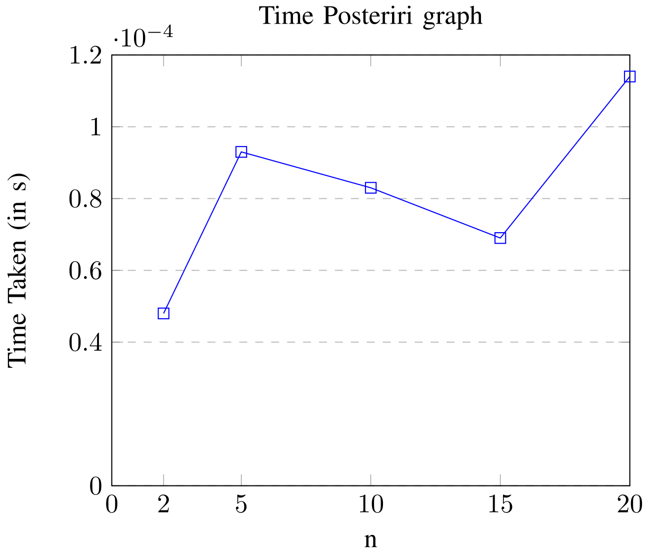
The space complexity of the Program is O(n) Because of the space allocation for resultant DP in each sub-problem.We solve every sub-problem till the end,that can be produced

## VII. Experimental Analysis

### A. Time Complexity

In the following table some cases are plotted

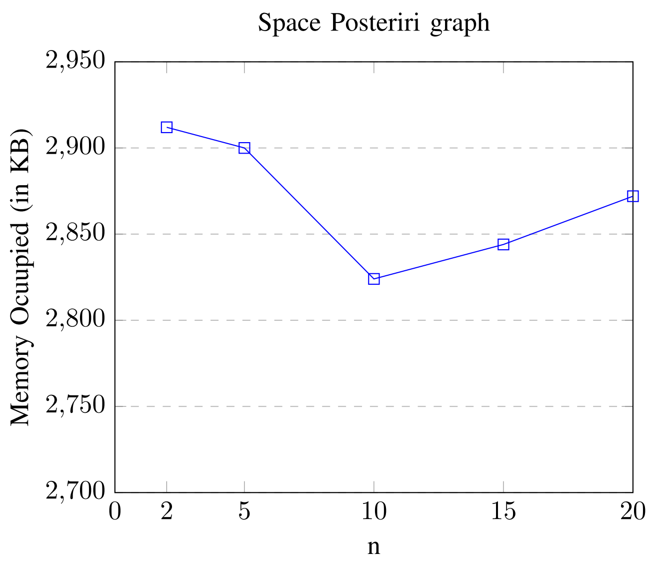
| n | Time Taken (in s) |
|---|---|
| 2 | 0.000048 |
| 5 | 0.000093 |
| 10 | 0.000083 |
| 15 | 0.000069 |
| 20 | 0.000114 |



### B. Space Complexity

In the following table some cases are plotted

| n | Space Occupied (in KB) |
|---|---|
| 2 | 2912 |
| 5 | 2900 |
| 10 | 2824 |
| 15 | 2844 |
| 20 | 2872 |



## VIII. Conclusion

We see that we are evaluating many sub-problem several times.It's easily understandable that for such small input there are so many repeated computations, then for higher inputs, how many overlapping there will be. Here comes the need for DP, where we can use tabulation to solve bigger sub problems using smaller ones.

## IX. Acknowledgment

We are very much grateful to our Course instructor Mr.Rahul kala and our mentor, Mr.Md Meraz, who have provided the great opportunity to do this wonderful work on the subject of Data Structure and Algorithm Analysis specifically on methodologies of Dynamic Programming.

## X. References

1.https://en.wikipedia.org/wiki/Dynamic-programming
2.https://www.geeksforgeeks.org/friends-pairing-problem/
3.https://www.includehelp.com/icp/friends-pairing-problem.aspx

## XI. Appendix

### *A. Project link on Github:*

https://github.com/LekhanaMitta/DAA-Assignment2