Dynamic Programming

Introduction

The problem at hand is to design a renting plan for a football club that receives requests from other clubs to rent players from their youth team. The plan should cover a period of 'n' years, during which the club can promote a limited number of players, denoted by 'p', at no cost. If the number of players requested for a given year, denoted by 'demand[i]', exceeds 'p', the club can choose to hire additional trainers at a cost of 'c' to promote the remaining players. Furthermore, if the club has any unrented players each year, they will need to pay a 'salary' for each of those players, denoted by 'salary[i]'. To solve this problem, the program uses dynamic programming principles.

Algorithm

The program has a recursive function called DP. The function takes several parameters, including the period, the current year, the number of promotable players, the number of players to hire each year, the cost of hiring players, the cost of reserving players, the minimum cost of a subproblem, and the reserved number of players on the team. The algorithm first calculates the difference between the player demand and the number of players currently on the team, subtracting the number of players to be hired each year. If the difference is negative, it recursively calculates the minimum cost for each possible number of players to hire for the next year. If the difference is non-negative, it calculates the cost of hiring and adds it to the total cost.

Time Complexity and Space Complexity

The algorithm forks if the demand is lower than p plus the previously hired player. This situation creates sub-problems, which the program follows each path and calculates the cost of each one. Then, it chooses the minimum cost. If there is no demand lower than p plus the hired player, there is no fork on the path, and that is the best-case scenario with a time complexity of O(n). However, if there are forks, the time complexity is higher according to the inputs. In a case where every demand creates p forks, which is the maximum number of forks, for n numbers of years, the time complexity is O(p^n). The space complexity of the algorithm depends on the number of forks. In the best-case scenario, where there is no fork, the space complexity is constant. The algorithm does not create extra variables or allocate memory; it only runs a for loop and changes already created variables. In the worst-case scenario, where every year has a fork, the space time complexity is exponential. Because for every fork, the program calls itself and allocates new memory.