S20190010034_A6

C Bhavesh Kumar

November 2020

1 Coin Denominations

1. Pseudo code

```
def coin_denominations():
    declare dp[C+1]
    for i=1 to c:
        for j=0 to n:
            if a[j]<=i:
                dp[i]<-min(dp[i],dp[i-a[j]]+1)
            endif

    while c>0:
        for j=0 to n:
            if c-a[j]>=0 and dp[c]==dp[c-a[j]]+1:
                print(a[j])
                c-=a[j]
                break
            endif

```

2. Runtime analysis
   The time complexity of the above algorithm is $O(nc)$,where n is the number of coin denominations and c is the amount to be paid.
   The above algorithm uses dynamic programming.
   The given problem can be solved using complete search using recursion,but the time complexity is exponential.
   Also, the recursion calculates sub-problems multiple times.Since the subproblems are overlapping, dynamic programming can be used to overcome this situation and hence time complexity $O(nc)$ can be achieved.

3. Correctness
   The given algorithm selects one choice of the available choices from the complete search.We do this with help of memorisation to reduce time complexity.
   To do this we use the bottom up approach of the dynamic programming to approach this problem.
   To find the answer of the query c, we find the subproblems of c first and then relate them with c and find out the minimum possiblity case.

we iterate from 1 to c and find out the best case scenario for dividing the denomination to get our result for each number in 1 to c.since c is dependent on its subproblems, which are already calculated, producing accurate result.
Hence our algorithm not only works with good time complexity but also is proven to be correct.
This is because memorisation works well with overlapping subproblems and produces accurate result.

2 Travel trip planner

1. Pseudo-code

```
def travel_planner():
  declare penalty[n]
  declare parent[n]
    for i=0 to n-1:
       penalty[i]=(20-a[i])*(20-a[i]);
       parent[i]=-1
    for i=0 to n-1:
       for j=i+1 to n-1:
          if penalty[j]>penalty[i]+power((20-(a[j]-a[i])),2);
            penalty[j]=penalty[i]+power((20-(a[j]-a[i])),2)
            parent[j]=i
          endif

```

   Penalty is the dp array which contains the minimum penalty to reach i'th hotel and parent is the array used to find the path taken to reach the last hotel or the destination.
   This can be done using the following pseudo code.

```
def path_calculator(parent,i):
    if parent[i]==-1:
        print(i+1)
        return;
    endif
    path_calculator(parent,parent[i])
    print(i+1)

```

2. Runtime Analysis
   The runtime complexity of the algorithm is $O(n^2)$. where n is the number of hotels present in the path.
   The algorithm uses a nested for loop to calculate the penalty array.
   Hence the time complexity of O(n^2).

3. Correctness
   We use bottom up approach of dp to solve this problem.
   We initially calculate the penalty thinking every hotel is reached directly from source.

After that we iterate from starting hotel to ending hotel and for each hotel in the iteration we check whether any other hotel to the right of this hotel can be reached from this hotel as a stepping stone to decrease its penalty. We do this for all the hotels and hence end up with minimum penalty for the last hotel.
This is similar to dijkstra algorithm's relaxing concept where we do the similar thing and concept used is also dynamic programming.
Also since the penalties cannot be negative and distances from source to hotels are in increasing order the algorithm does not fail.
Hence the bottom up approach of dynamic programming where we calculate the subproblems first and memorise them works and produces a correct answer for the given problem.
Since all the possibilities are covered and minimum is selected in the penalty array each iteration the given algorithm works perfectly fine and produces accurate result.
Hence the above algorithm is correct.