ALGORITHMS AND PROBLEM SOLVING LAB

**Made by-**

Yatharth Grover 9922103119

**INTRODUCTION**

Shyam is an uneducated milkman. Everyday he gets up early milk his cows, set off on his van to deliver milk to his customers. He has been working for 5 years now but he barely earns enough to earn a substantial amount of profit and sustain his family. He has good quality of cows and if he works efficiently he can earn a good profit.

**PROBLEM STATEMENT**

We need to help Shyam to milk the best quality of milk everyday without any wastage and also help in transport milk in minimum time and cost of transportation. And also help him provide change to his customers in minimum denomination.

UTILITY

Shyam has 'n' breeds cows of different milking capacity and quality. When Shyam receives an order he has to decide which cow to milk depending on the quantity and quality of milk as per the requirement of his customers. We need to provide him with such a solution to find which cows he needs o milk to meet the desired quantity and also provide his customers with the best quality of milk ensuring there is no wastage. After he has prepared all his orders he needs to transport the orders himself on his van. We need to provide Shyam with a root map such that he can reach to all his customers using the shortest route in minimum distance to save his expense on petrol. When he reaches his customers give him money and he needs to decide what denominations are required to return the change in minimum number coins.

Our aim is to help Shyam with his day to day activities so that he can earn maximum profit.



ALGORITHMS USED

1. **Perfect Subset Sum Problem(using DP)**

 declare mem[no\_of\_cows+1][sum+1]

 for i=0 to sum

  mem[0][i]= false

 for i= 0 to n

  mem[i][0]=true;

 for i= 1 to n

 for j =1 to n

 if arr[i] is greater than j

  then mem[i][j]= mem[i-1][j]

  else

if mem[i-1][j]=true

  then mem[i][j]=mem[i-1][j]

  else

  mem[i][j]=mem[i-1][j-arr[i]]

 memrecur:

 backtrack and store the perfect sum in myVector[]

1. **Weighted Average Method**

for i= 0 to arrr.size()

for j= 0 to arr[i].size()

sumw=sumw+arr[i][j]\*quant[i][j];

  sumww+=arr[i][j];

weighted\_average[i]=sumw/sumww

find the max in weighted average

1. **Coinage Problem(using DP)**

coin\_change(int deno[],int si,vector<vector<int> >& quan)

Mem[si+1][ret+1]

for i=0 to si+1

for j=0 to ret+1

mem[i][j]=0

for j 0 to ret+1

mem [0][j]=99999;

for i=1 to si+1

mem i,0=0;

for i=1 to si+1

for j=0 to ret+1

if deno[i-1]<=j

mem[ i][j]=min((1+mem[ i][j-deno[ i-1]],mem[ i-1][j]);

else

mem[ i][j]=mem[ i-1][j];

return mem [si][ret];

end

 backtrack and find the which denominations are required

1. **Travelling salesman problem**

First ask the user how many customer then input distance of each customer with repect to each other in a matrix form

void mincost(int city)

i,ncity;

completed[city]=1;

print city+1 "--->";

To find minimum distance and customer between calculated customer and remaining customers

ncity=least(city);

To retrun back to his farm

if(ncity==999)

ncity=0;

print ncity+1;

cost+=ary[city][ncity];

return

mincost(ncity);

int least(int c)

int i,nc=999;

int min=999,kmin;

for i 0 to V

To check it is calculating distance for other customers and should not travelled that node yet

if((ary[c][i]!=0)&&(completed[i]==0))

if(ary[c][i]+ary[i][c] < min)

min=ary[i][0]+ary[c][i];

kmin=ary[c][i];

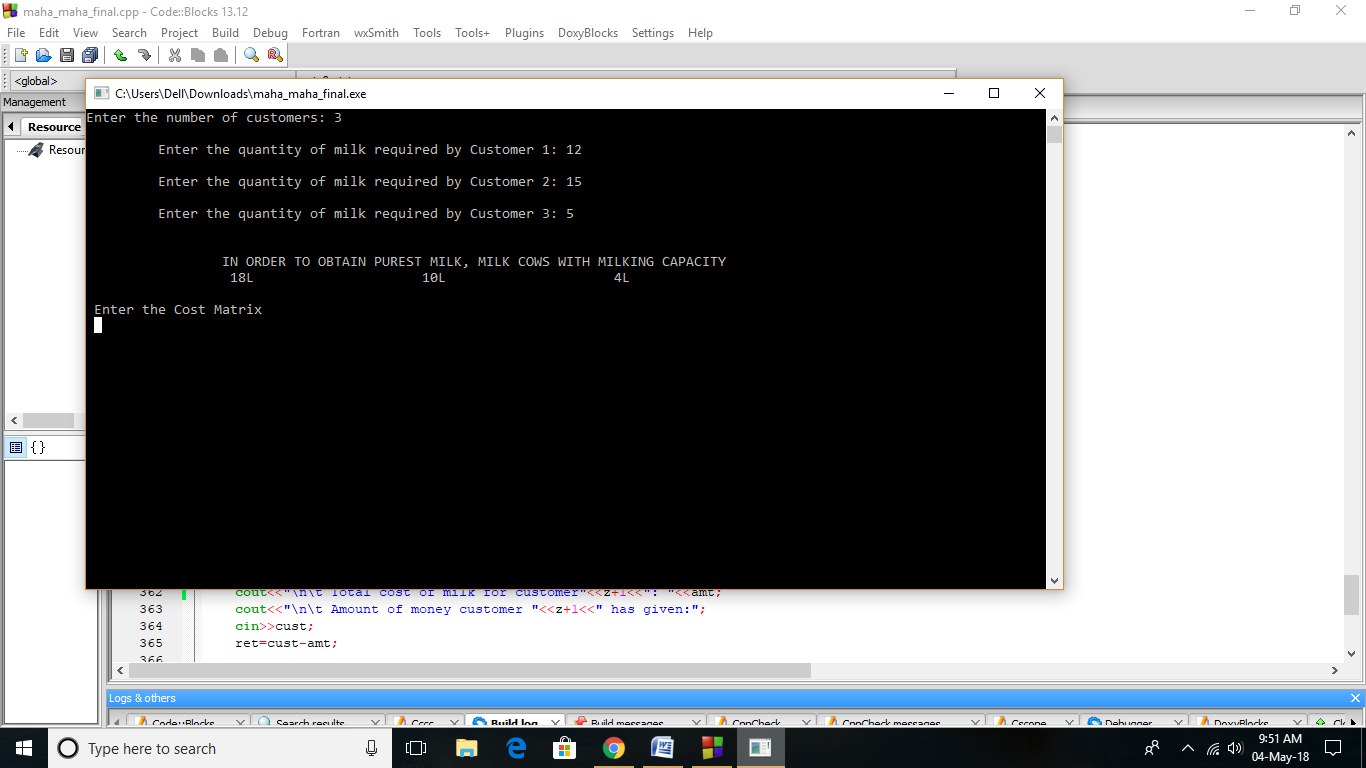
Returning index of minimum path

nc=i;

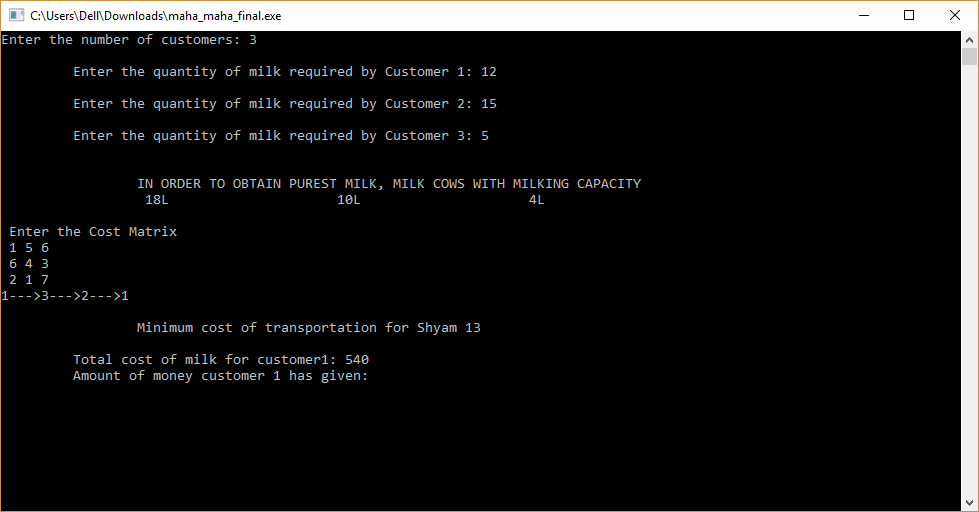
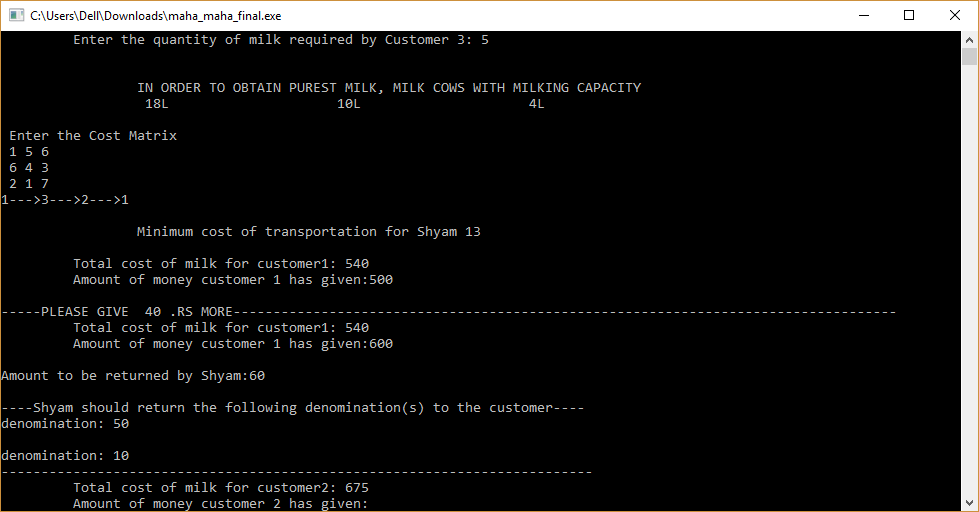
if(min!=999)

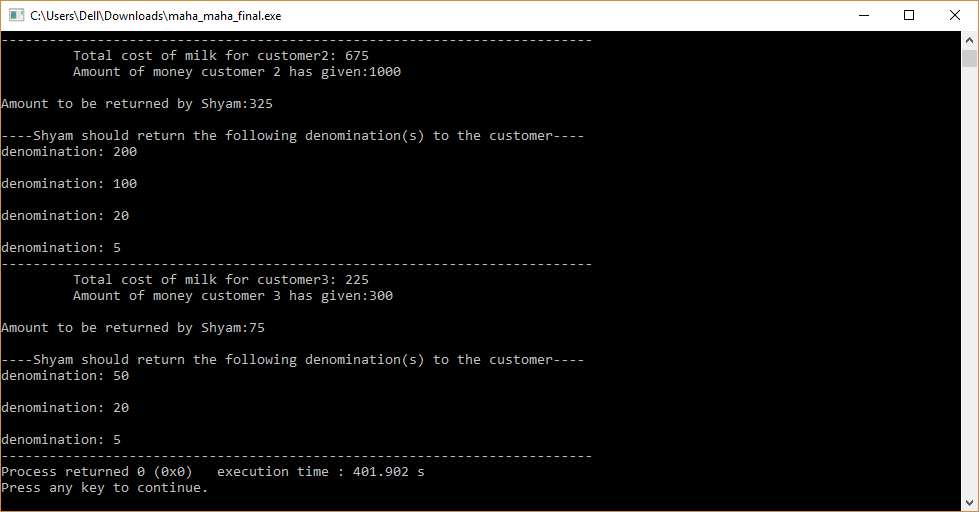
cost+=kmin;

return nc;

OUTPUT







COMPARISION

(With other techniques)

1. **Perfect Subset Sum Problem(using DP)**

One way to find subsets that sum to K is to consider all possible subsets. A [power set](http://en.wikipedia.org/wiki/Power_set) contains all those subsets generated from a given set. The size of such a power set is 2N. Also Backtracking can be used to make a systematic consideration of the elements to be selected. Using exhaustive search we consider all subsets irrespective of whether they satisfy given constraints or not. During recursion, there may exist a case where same sub-problems are solved multiple times thus increases memory and time.

1. **Travelling Salesman Problem(Hamiltonian Cycle with minimum weights)**

The naïve solution is to consider city 1 as the starting and ending point. Then Generate all (n-1)! [Permutations](https://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/)of cities. Calculate cost of every permutation and keep track of minimum cost permutation. Return the permutation with minimum cost.(Time Complexity: O(n!))

To calculate using Dynamic Programming, we need to have some recursive relation in terms of sub-problems.

1. **Coinage Problem(using DP)**

To find the minimum number of coins and/or notes needed to make the change we can also use greedy approach to solve it but greedy may not always give the optimal answer.

REFERENCES/BIBLIOGRAPHY

Introduction to Algorithm by Thomas S.