**Rehnoor Aulakh**

**102317137**

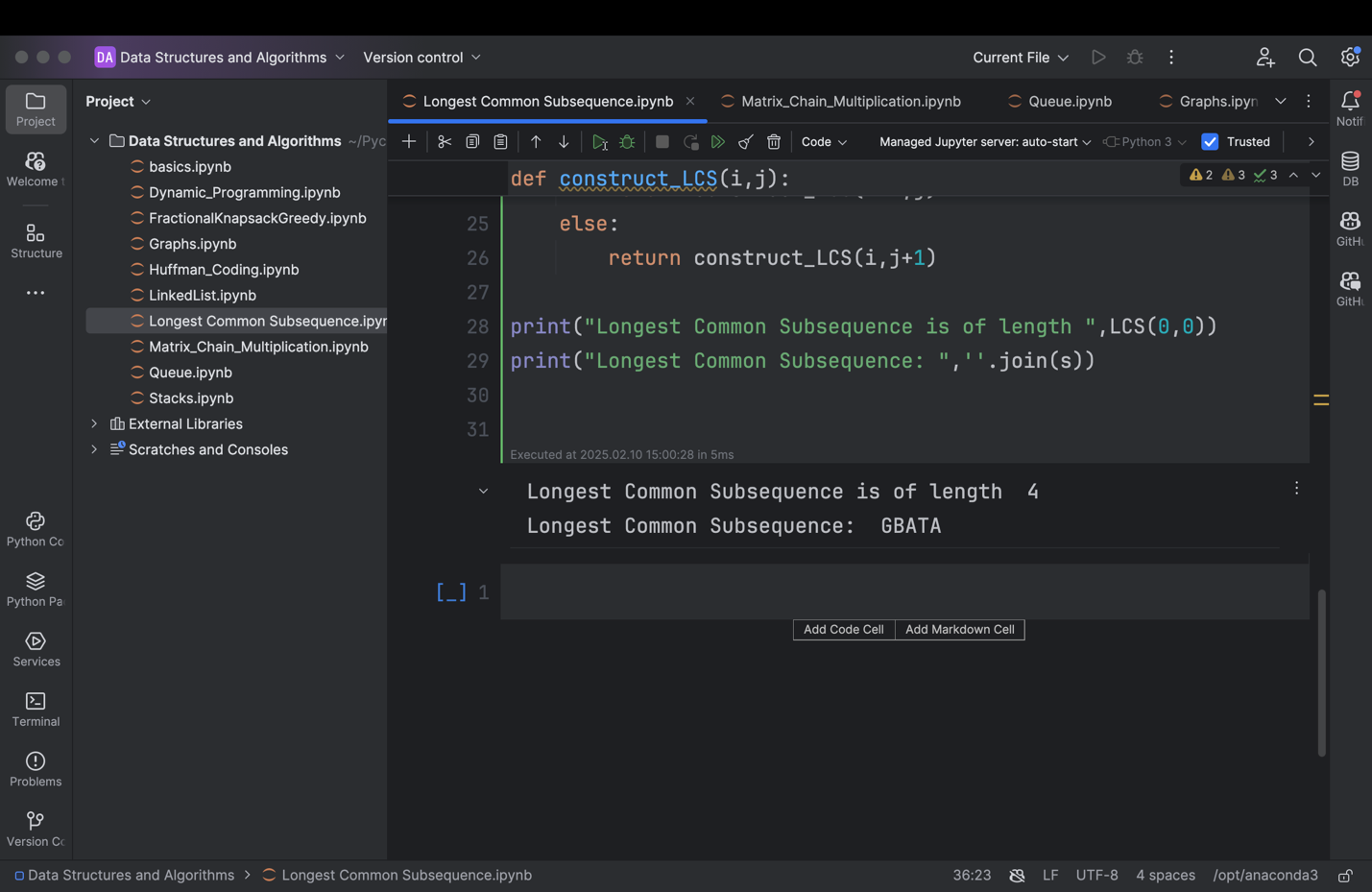
**2Q16**

**Assignment 3**

**Q1 Longest Common Subsequence**

A="AGGTAB"  
B="GXTXAYB"  
memo={}  
s=[]  
  
def LCS(i,j):  
 if (i,j) in memo:  
 return memo[(i,j)]  
 if i>=len(A) or j>=len(B):  
 return 0  
 elif A[i]==B[j]:  
 s.append(A[i])  
 memo[(i,j)]=1+LCS(i+1,j+1)  
 else:  
 memo[(i,j)]=max(LCS(i+1,j),LCS(i,j+1))  
 return memo[(i,j)]  
  
def construct\_LCS(i,j):  
 if i>=len(A) or j>=len(B):  
 return ""  
 if A[i]==B[j]:  
 return A[i]+construct\_LCS(i+1,j+1)  
 if (i+1,j) in memo and memo[(i+1,j)]>=memo.get((i,j+1),0):  
 return construct\_LCS(i+1,j)  
 else:  
 return construct\_LCS(i,j+1)  
  
print("Longest Common Subsequence is of length ",LCS(0,0))  
print("Longest Common Subsequence: ",''.join(s))

**OUTPUT**

****

**Q2. Matrix Chain Multiplication**

#include<iostream>

using namespace std;

void printMatrix(int cost[][4], int dim)

{

cout<<endl<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

for(int i=0;i<dim;i++)

{

for(int j=0;j<dim;j++)

{

cout<<cost[i][j]<<" ";

}

cout<<endl;

}

cout<<endl<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

}

void printOptimalParenthesis(int s[][4], int i, int j)

{

if(i==j)

{

cout<<"M"<<i;

}

else

{

cout<<"(";

printOptimalParenthesis(s,i,s[i][j]);

printOptimalParenthesis(s,s[i][j]+1,j);

cout<<")";

}

}

int main()

{

int n=4;

//setting the dimensions for the four matrices

int p[]={2,1,3,4};

//initializing the m matrix and s matrix with zeros

int cost[4][4]={0};

int s[4][4]={0};

int j=0;

int min=0;

int q=0;

// d loop runs for the difference, initially 1 and then goes upto j-1

for(int d=1;d<n;d++)

{

for(int i=1;i<n-d;i++)

{

j=i+d;

min=INT\_MAX;

for(int k=i;k<j;k++)

{

q=cost[i][k]+cost[k+1][j]+(p[i-1]\*p[k]\*p[j]);

if(q<min)

{

min=q;

s[i][j]=k;

}

}

cost[i][j]=min;

}

}

printMatrix(cost,n);

printMatrix(s,n);

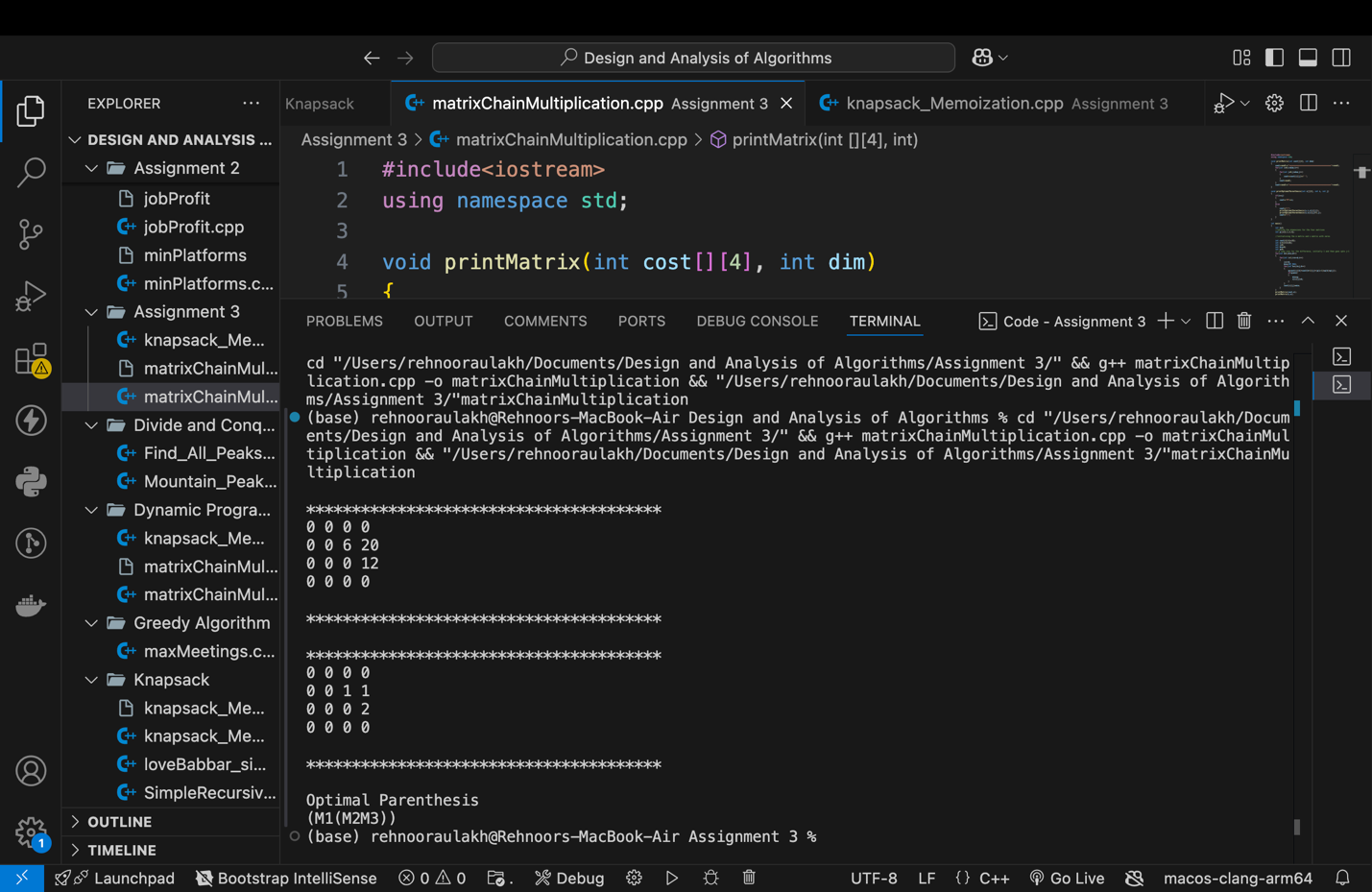
//Now printing the resultant parenthesis using inorder traversal on s matrix

cout<<endl<<"Optimal Parenthesis"<<endl;

printOptimalParenthesis(s,1,n-1);

cout<<endl;

}



Q3 0/1 Knapsack Problem

#include<iostream>

#include<vector>

using namespace std;

int solveMem(vector<int> weight, vector<int> value, int index, int capacity, vector<vector<int> >&dp)

{

//Base Case: for one element only

if(index==0)

{

if(weight[index]<=capacity)

{

return value[0];

}

else

{

return 0;

}

}

if(dp[index][capacity]!=-1)

{

return dp[index][capacity];

}

int include=0;

if(weight[index]<=capacity)

{

include=value[index]+solveMem(weight,value,index-1,capacity-weight[index],dp);

}

int exclude=solveMem(weight,value,index-1,capacity,dp);

dp[index][capacity]=max(exclude,include);

return dp[index][capacity];

}

int knapsack(vector<int> weight, vector<int> value, int n, int maxWeight)

{

// return solve(weight,value,n-1,maxWeight);

vector<vector<int> >dp(n,vector<int>(maxWeight+1,-1));

return solveMem(weight,value,n-1,maxWeight,dp);

}

int main() {

vector<int> profit = {5, 3, 8, 6};

vector<int> weight = {2, 3, 4, 5};

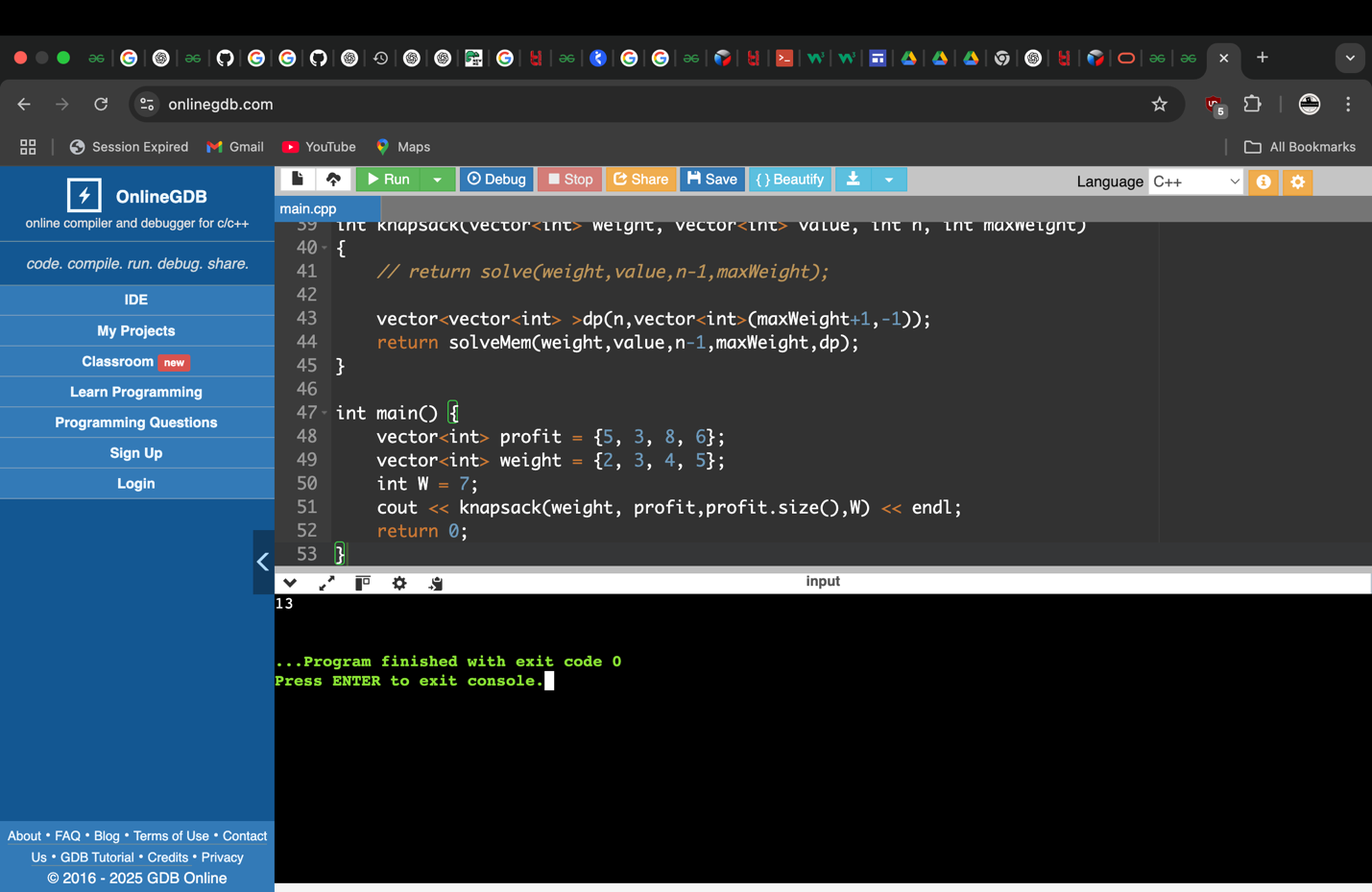
int W = 7;

cout << knapsack(weight, profit,profit.size(),W) << endl;

return 0;

}

OUTPUT



Q4 Maximum Square length

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int maxSquare(vector<vector<int> >&mat)

{

//getting the number of rows and columns

int n=mat.size();

int m=mat[0].size();

//creating a 2d matrix for tabulation

vector<vector<int>>dp(n+1,vector<int>(m+1,0));

int ans=0;

//fill the dp

for(int i=n-1;i>=0;i--)

{

for(int j=m-1;j>=0;j--)

{

if(mat[i][j]==0)

{

dp[i][j]=0;

continue;

}

dp[i][j]=1+min({dp[i][j+1],dp[i+1][j],dp[i+1][j+1]});

ans=max(ans,dp[i][j]);

}

}

return ans;

}

int main()

{

vector<vector<int> >mat={{0, 1, 1, 0, 1},

{1, 1, 0, 1, 0},

{0, 1, 1, 1, 0},

{1, 1, 1, 1, 0},

{1, 1, 1, 1, 1},

{0, 0, 0, 0, 0}};

cout<<maxSquare(mat)<<endl;

}

OUTPUT

