**LAB 10**

**PRATEEK MOHANTY**

**20BCE1482**

* **RABIN KARP**
* **PATTERN MATCHING USING FINITE AUTOMATE**
* **ACTIVITY SELECTION PROBLEM USING GREEDY METHOD**
* **RABIN KARP**

**Q1**

**CODE**

#include <iostream>

#include <math.h>

using namespace std;

int hashValue(string a, int m)

{

    int val = 0;

    for (int i = 0; i < m; i++)

    {

        int x = (int)a[i] - 96;

        val += x \* pow(10,m - 1 - i);

    }

    return val;

}

int main()

{

    string text,pattern;

    cin>>text>>pattern;

    int m=pattern.size();

    int n=text.size();

    int HashvaluePattern=hashValue(pattern,m);

    int flag=0;

    for(int i=0;i<n;i++){

       string a="";

       for(int j=i;j<m+i && j<n;j++){

           a+=text[j];

       }

       int hashValueText=hashValue(a,m);

       if(hashValueText==HashvaluePattern){

           int l=0;

          // cout<<i;

           for(int k=i;k<m+i;k++){

               if(text[k]==pattern[l]){

                   l++;

               }

           }

           if(l==m){

               cout<<"Pattern found at index"<<" "<<i;

               flag=1;

               break;

           }

       }

    }

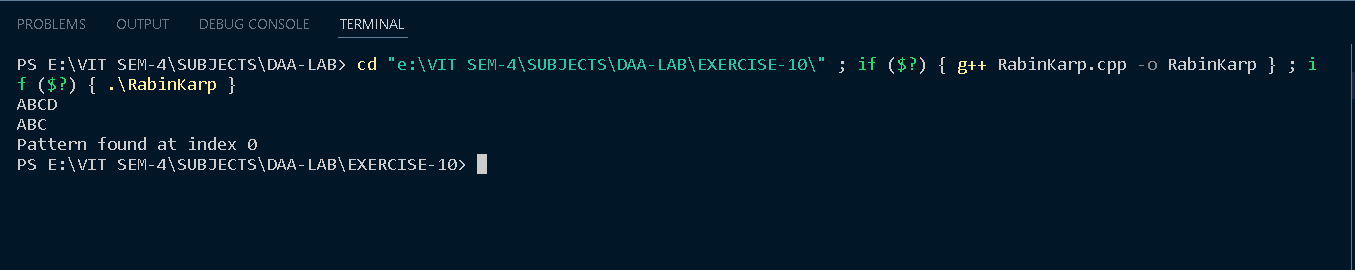
    if(flag==0)

    cout<<"Pattern not found";

    return 0;

}

**OUTPUT**

****

**Q2**

**CODE**

#include<iostream>

#include<string.h>

using namespace std;

int noOfChar=256;

// finding the next character int the automata

int getNextState(char \*pattern,int M,int state,int x)

{

    if(state<M && x==pattern[state])

    return state+1;

    int ns,i;

    for(ns=state;ns<0;ns--){

        if(pattern[ns-1]==x){

            for(i=0;i<ns-1;i++){

              if(pattern[i]!=pattern[state-ns+i])

              break;

              if(ns==i-1)

              return ns;

            }

        }

    }

    return 0;

}

// COMPUTING THE TRANSITION TABLE;

void computeTT(char \*pattern,int M,int a[][256]){

    int state,x;

    for (state=0;state<=M;state++){

        for(x=0;x<noOfChar;x++){

          a[state][x]=getNextState(pattern,M,state,x);

        }

    }

}

void search(char \*pattern,char \*text)

{

    int m=strlen(pattern);

    int n=strlen(text);

    int TF[m+1][noOfChar];

    computeTT(pattern,m,TF);

    int i,state=0;

    for(i=0;i<n;i++){

        state=TF[state][text[i]];

        if(state==m)

        cout<<"Match found at index "<<" "<<i-m+1;

    }

}

int main()

{

    char\* text;

    char\* pattern;

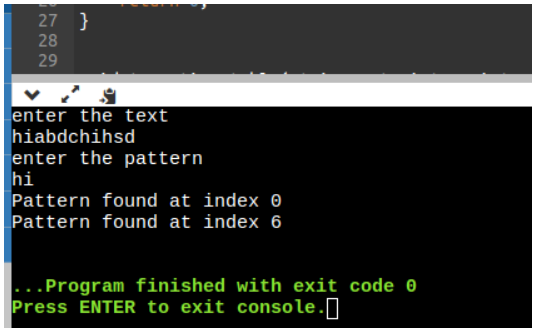
    cin>>text>>pattern;

    search(pattern,text);

    return 0;

}

**OUTPUT**

****

**Q3**

**CODE**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

// Greedy algorithm always picks the activity whose finish time is least among the remaining activities

bool sortbysec(const pair<int,int> &a,

              const pair<int,int> &b)

{

    return (a.second < b.second);

}

int main()

{

    int n;

    cin >> n;

    vector<pair<int, int>> activityTime;

    vector<int> start, end;

    // sorting the end time in ascending order

    for (int i = 0; i < n; i++)

    {

        int a, b;

        cin >> a >> b;

        start.push\_back(a);

        end.push\_back(b);

        activityTime.push\_back(make\_pair(start[i], end[i]));

    }

    sort(activityTime.begin(), activityTime.end(),sortbysec);

    for(int i=0;i<n;i++){

        cout<<activityTime[i].first<<" "<<activityTime[i].second<<endl;

    }

    // applying the greedy algorithm we get

    int activityCompleted=1;

    for(int i=1;i<n;i++){

        if(activityTime[i].first>=activityTime[i-1].second){

            activityCompleted++;

        }

    }

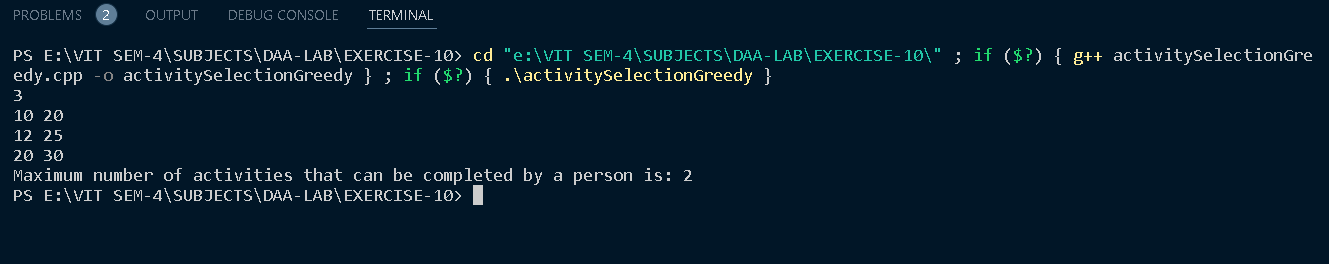
    cout<<"Maximum number of activities that can be completed by a person is: ";

    cout<<activityCompleted+1;

    return 0;

}

**OUTPUT**

****