CS211 ALGORITHMS & DATA STRUCTURES II

LAB 10

Dr. Phil Maguire

STRING SEARCHING ALGORITHMS

Pen and Paper Exercise

The goal is to find the pattern ABRACADABRA in the string ABRACABABRABRACABRACADABRA.

Show step-by-step how the Knuth-Morris-Pratt algorithm and how the Boyer-Moore algorithm would find the pattern. Find how many comparisons are carried out by each.

For the Knuth-Morris-Pratt algorithm you should first write out the Partial Match Table and then show clearly the progress of the algorithm through the string.

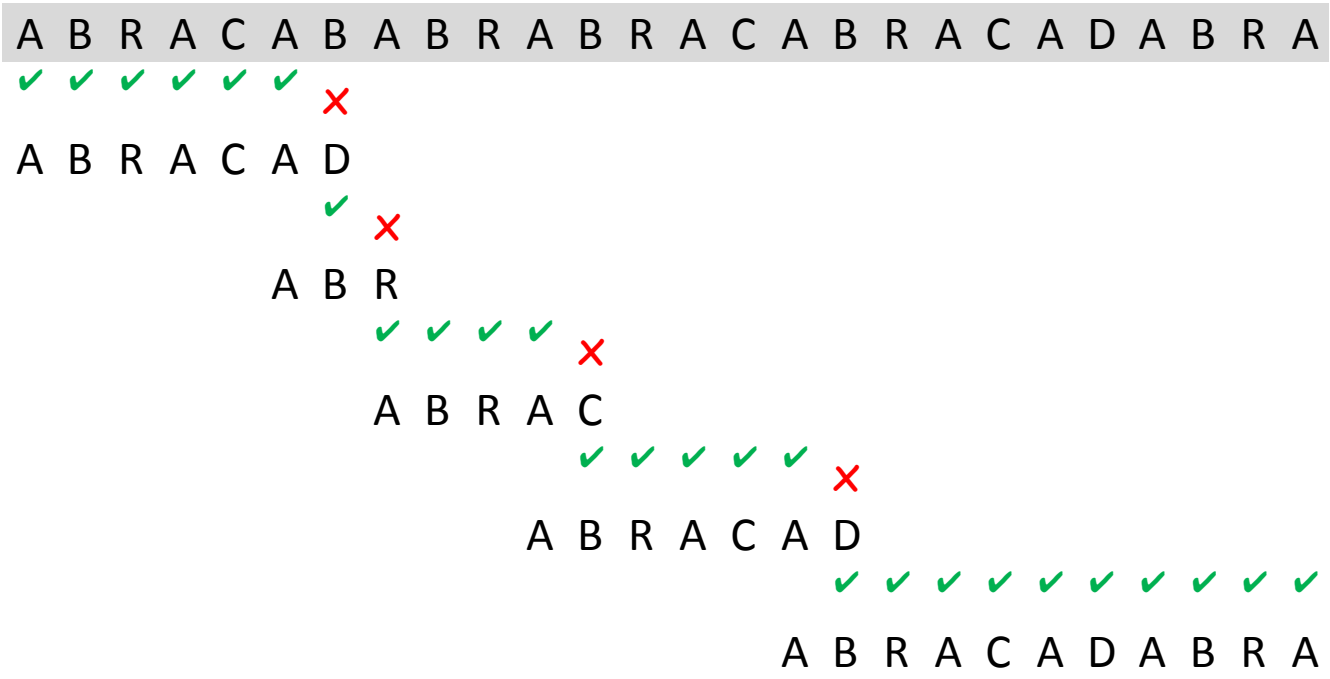
For the Boyer-Moore algorithm you should first write out the Bad Character Shift Table and the Good Suffix Shift Table and then show clearly the progress of the algorithm through the string.

1. Knuth-Morris-Pratt Algorithm

Partial Match Table

A	B	R	A	C	A	D	A	B	R	A
-1	0	0	0	1	0	1	0	1	2	3

Comparison Table - 30 comparisons in total



Programming Exercise

```
            while(visited<size){    //while not all vertices visited

                int minvalue=Integer.MAX_VALUE; //set to infinity as
default-there must be something shorter

                int minvertex=-1;   //a default which will crash if it
is not superseded

                for(int j=0;j<size;j++){    //find the next vertex to
visit

                    if(vertices[j].visited==false){       //if it has
not been visited

                        if(vertices[j].distance<minvalue){

                            minvalue=vertices[j].distance;

                            minvertex=j;    //we are choosing the
closest vertex to the starting position which has not been visited yet

                        }
```

```
                    }

                }

                vertices[minvertex].visited=true;   //we have now
visited it

                visited++;

                for(int j=0;j<size;j++){    //update distances from
this new point


if(array[j][minvertex]>0&&vertices[j].visited==false){  //if vertex is
connected and not visited


if(vertices[j].distance>vertices[minvertex].distance+array[j][minverte
x]){


vertices[j].distance=vertices[minvertex].distance+array[j][minvertex];
//update if a new shorter route to this vertex has been found

                            vertices[j].route=minvertex; //track the
path taken

                        }

                    }

                }

            }
```