1 ASSIGNMENT NO: A5

Author: Ameeth Kanawaday
Roll No: 4430

2 Problem Definition

Int code generation for sample language using LEX and YACC.

3 Learning Objectives:

1. To understand Divide and Conquer strategy.
2. To use Divide and Conquer for implementing binary search.

4 S/W and H/W requirements:

1. Open source 64 bit OS.
2. Gedit text editor.
3. Scala

5 Theory

Divide and Conquer:

In computer science, divide and conquer is an algorithm design paradigm based on multi-branched recursion. A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

This divide and conquer technique is the basis of efficient algorithms for all kinds of problems, such as sorting (e.g., quicksort, merge sort), multiplying large numbers (e.g. the Karatsuba algorithm), finding the closest pair of points, syntactic analysis (e.g., top-down parsers), and computing the discrete Fourier transform (FFTs).

Understanding and designing Divide and Conquer algorithms is a complex skill that requires a good understanding of the nature of the underlying problem to be solved. As when proving a theorem by induction, it is often necessary to replace the original problem with a more general or complicated problem in order to initialize the recursion, and there is no systematic method for finding the proper generalization. These Divide and Conquer complications are seen when optimizing the calculation of a Fibonacci number with efficient double recursion.

The correctness of a divide and conquer algorithm is usually proved by mathematical induction, and its computational cost is often determined by solving recurrence relations.

Binary Search algorithm
Given an array A of n elements with values or records A0...An-1, sorted such that A0 ¡=...¡= An-1, and target value T, the following subroutine uses binary search to find the index of T in A.

1. Set L to 0 and R to n-1.
2. If L ¿ R, the search terminates as unsuccessful.
3. Set m (the position of the middle element) to the floor of (L+R)/2.
4. If Am ¡ T, set L to m+1 and go to step 3.
5. If Am ¿ T, set R to m-1 and go to step 3.
6. Now Am = T, the search is done; return m.

6 Related Mathematics

Let S be the solution perspective of the given problem.
The set S is defined as:
$S = \{ s, e, X, Y, F, DD, NDD | \varnothing_s \}$
Where,
s= Start point
e= End point
F= Set of main functions
DD= set of deterministic data
NDD= set of non deterministic data

X= Input Set.
X = source program code in high level language.

Y = $\{intermediatecodeforthesamplecode\}$

s = sample language ready into read buffer.
e = Intermediate code generated.

F = $\{f_{read}, f_{lex}, f_{parse}, f_{ret}\}$

f_{read} :function to read the sample code.

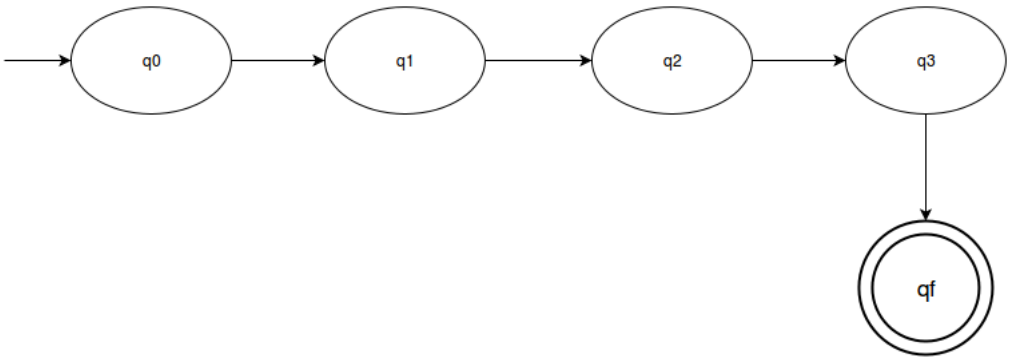
f_{lex} :function to generate tokens for the given codes.

f_{parse} :function to generate parse tree using the generated tokens.

f_{ret} :function generate intermediate code using the syntax tree.

DD = $\{samplecode, tokenmatchingrules, grammarrules\}$
NDD = ϕ

7 State Diagram



q0 = read input language state
q1 = token generation state
q2 = parsing state
q3 = display intermediate code state
qf = final state

8 Program

```
////a5.l

ALPHA [A-Za-z]
DIGIT [0-9]
%%

{ALPHA}({ALPHA}|{DIGIT})* return ID;
{DIGIT}+ {yylval=atoi(yytext); return NUM;}
[\n\t] yyterminate();
. return yytext[0];
%%

////a5.y

%token ID NUM
%right '='
%left '+' '-'
%left '*' '/'
%left UMINUS
%%

S : ID{push();} '='{push();} E{codegen_assign();}
   ;
```

```
E : E '+'{push();} T{codegen();}
   | E '-'{push();} T{codegen();}
   | T
   ;
T : T '*'{push();} F{codegen();}
   | T '/'{push();} F{codegen();}
   | F
   ;
F : '(' E ')'
   | '-'{push();} F{codegen_umin();} %prec UMINUS
   | ID{push();}
   | NUM{push();}
   ;
%%

#include "lex.yy.c"
#include<ctype.h>
char st[100][10];
int top=0;
char i_[2]="0";
char temp[2]="t";

main()
 {
 printf("Enter the expression : ");
 yyparse();
 }

push()
{
  strcpy(st[++top],yytext);
 }

codegen()
 {
 strcpy(temp,"t");
 strcat(temp,i_);
  printf("%s = %s %s %s\n",temp,st[top-2],st[top-1],st[top]);
  top-=2;
 strcpy(st[top],temp);
 i_[0]++;
 }

codegen_umin()
 {
 strcpy(temp,"t");
 strcat(temp,i_);
 printf("%s = -%s\n",temp,st[top]);
 top--;
 strcpy(st[top],temp);
```

```
 i_[0]++;
 }

codegen_assign()
 {
 printf("%s = %s\n",st[top-2],st[top]);
 top-=2;
 }


OUTPUT:
ameeth@ubuntu-16.0.4:~/CL1$ lex a5.l
ameeth@ubuntu-16.0.4:~/CL1$ yacc a5.y
ameeth@ubuntu-16.0.4:~/CL1$ gcc y.tab.c -ll -ly
ameeth@ubuntu-16.0.4:~/CL1$ ./a.out
Enter the expression : a=(u+9)/(5-c)
t0 = u + 9
t1 = 5 - c
t2 = t0 / t1
a = t2
ameeth@ubuntu-16.0.4:~/CL1$
```

9 Conclusion

Thus we have implemented the Naive Bayes classifier in python using sklearn.