Aim: Removal of Left Factoring for the given context free grammar (CFG).

Input: S->iEtS | iEtSeS | a | b
Output: S->iEtSS' | a | b

S'-> eS | ε

Theory:

A grammar is said to be left factored when it is of the form –

A -> αβ1 | αβ2 | αβ3 | …… | αβn | γ i.e the productions start with the same terminal (or set of terminals). On seeing the input α we cannot immediately tell which production to choose to expand A.

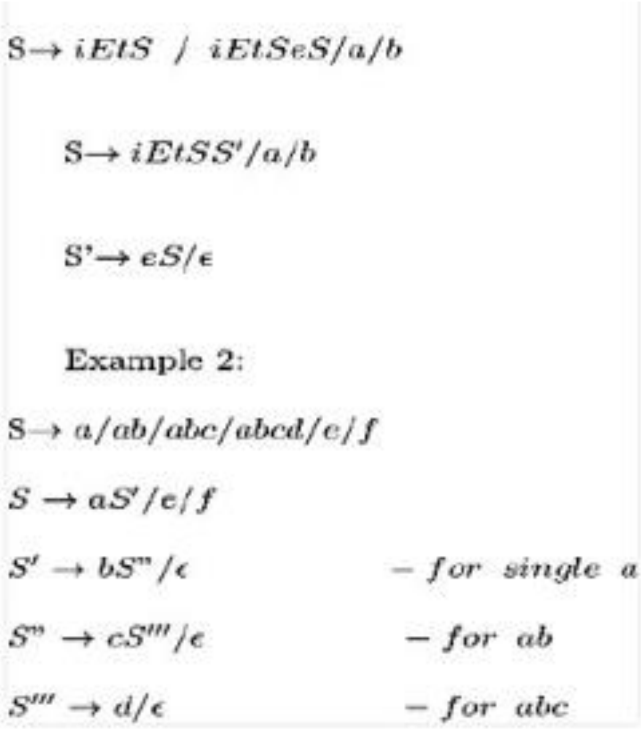
Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive or top down parsing. When the choice between two alternative A-productions is not clear, we may be able to rewrite the productions to defer the decision until enough of the input has been seen to make the right choice.

For the grammar A -> αβ1 | αβ2 | αβ3 | …… | αβn | γ

The equivalent left factored grammar will be –

A -> αA’ | γ

A’ -> β1 | β2 | β3 | …… | βn



$S \rightarrow iEtS \ / \ iEtSeS/a/b$

$S \rightarrow iEtSS'/a/b$

$S' \rightarrow eS/\epsilon$

Example 2:

$S \rightarrow a/ab/abc/abcd/e/f$

$S \rightarrow aS'/e/f$

$S' \rightarrow bS''/\epsilon$ $- for \ single \ a$

$S'' \rightarrow cS'''/\epsilon$ $- for \ ab$

$S''' \rightarrow d/\epsilon$ $- for \ abc$

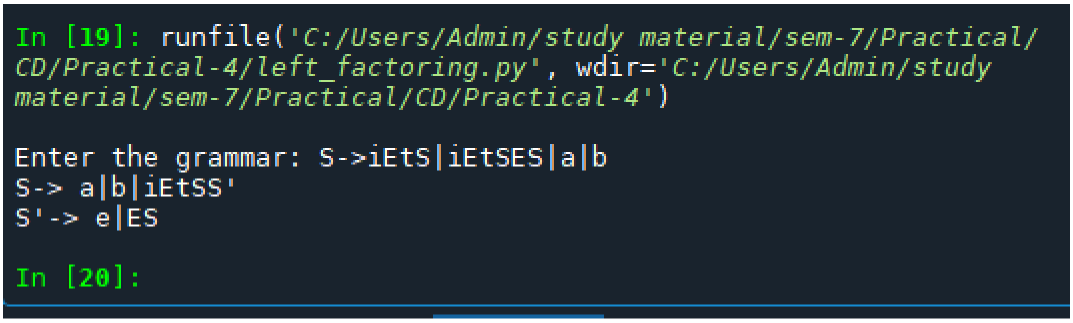
Source Code:

```
def minimum_matched_string(a,
b,len_a,len_b):
   if len_a == 0 or len_b == 0:
     return 0;
   elif a[len_a-1] == b[len_b-1]:
      return 1 +
minimum_matched_string(a, b, len_a-
1, len_b-1);
   else:
     return
max(minimum_matched_string(a, b,
len_a, len_b-1),
minimum_matched_string(a, b, len_a-
1, len_b));


length=1000
grammer=input('Enter the grammar: ')
lhs=grammer[0]
f=grammer[3:]
rhs=f.split('|')
```

```
grammar={lhs:rhs}

for key in grammar.copy():
    item=grammar[key]
    for i in range(0, len(item)-1):
        for j in range(i, len(item)):
            if i!=j:
                # print(item[i], item[j],
len(item[i]), len(item[j]))

l=minimum_matched_string(item[i],ite
m[j],len(item[i]),len(item[j]))
                if l>0:
                    if length > l:
                        length = l

eq2=[]
if length>0:

common_val=grammar[key][0][:length]
     for i in range(len(grammar[key])):
         if common_val in grammar[key][i]:

grammar[key][i]=grammar[key][i][lengt
h:]
             eq2.append(grammar[key][i])
             grammar[key][i]=''

eq1=grammar[key]
for i in range(len(eq1)):
    if '' in eq1:
        eq1.remove('')


eq1.append(common_val+"S'")

for i in eq2:
    if i=='':
        ind=eq2.index(i)
        eq2[ind]='e'
```

```
fi='S-> '
fii='''S'-> '''

for i in eq1:
    fi+=i+'|'

for i in eq2:
    fii+=i+'|'

fi=fi[:-1]
fii=fii[:-1]

print(fi)
print(fii)
```

Output:



```
In [19]: runfile('C:/Users/Admin/study material/sem-7/Practical/
CD/Practical-4/left_factoring.py', wdir='C:/Users/Admin/study
material/sem-7/Practical/CD/Practical-4')

Enter the grammar: S->iEtS|iEtSES|a|b
S-> a|b|iEtSS'
S'-> e|ES

In [20]:
```