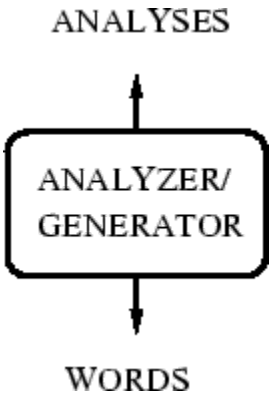
EXP-1: Word Analysis

Aim:A word can be simple or complex. For example, the word 'cat' is simple because one cannot further decompose the word into smaller part. On the other hand, the word 'cats' is complex, because the word is made up of two parts: root 'cat' and plural suffix '-s'



Theory: Analysis of a word into root and affix(es) is called as Morphological analysis of a word. It is mandatory to identify root of a word for any natural language processing task. A root word can have various forms. For example, the word 'play' in English has the following forms: 'play', 'plays', 'played' and 'playing'.

Thus we understand that the morphological richness of one language might vary from one language to another. Indian languages are generally morphologically rich languages and therefore morphological analysis of words becomes a very significant task for Indian languages.

Types of Morphology

Morphology is of two types,

1. Inflectional morphology

Deals with word forms of a root, where there is no change in lexical category. For example, 'played' is an inflection of the root word 'play'. Here, both 'played' and 'play' are verbs.

2. Derivational morphology

Deals with word forms of a root, where there is a change in the lexical category. For example, the word form 'happiness' is a derivation of the word 'happy'. Here, 'happiness' is a derived noun form of the adjective 'happy'.

Morphological Features:

All words will have their lexical category attested during morphological analysis. A noun and pronoun can take suffixes of the following features: gender, number, person, case For example, morphological analysis of a few words is given below:

Language	input:word	output:analysis
English	boy	rt=boy, cat=n, gen=m, num=sg
English	boys	rt=boy, cat=n, gen=m, num=pl

A verb can take suffixes of the following features: tense, aspect, modality, gender, number, person

Language	input:word	output:analysis
English	toys	rt=toy, cat=n, num=pl, per=3

'rt' stands for root. 'cat' stands for lexical category. Thev value of lexicat category can be noun, verb, adjective, pronoun, adverb. 'gen' stands for gender. The value of gender can be masculine or feminine. 'num' stands for number. The value of number can be singular (sg) or plural (pl). 'per' stands for person. The value of person can be 1, 2 or 3

The value of tense can be present, past or future. This feature is applicable for verbs. The value of aspect can be perfect (pft), continuous (cont) or habitual (hab). This feature is not applicable for verbs.

'case' can be direct or oblique. This feature is applicable for nouns. A case is an oblique case when a postposition occurs after noun. If no postposition can occur after noun, then the case is a direct case. This is applicable for hindi but not english as it doesn't have any postpositions.

Procedure:

STEP 1: Select the language.

OUTPUT: Drop down for selecting words will appear.

STEP 2: Select the word.

OUTPUT: Drop down for selecting features will appear.

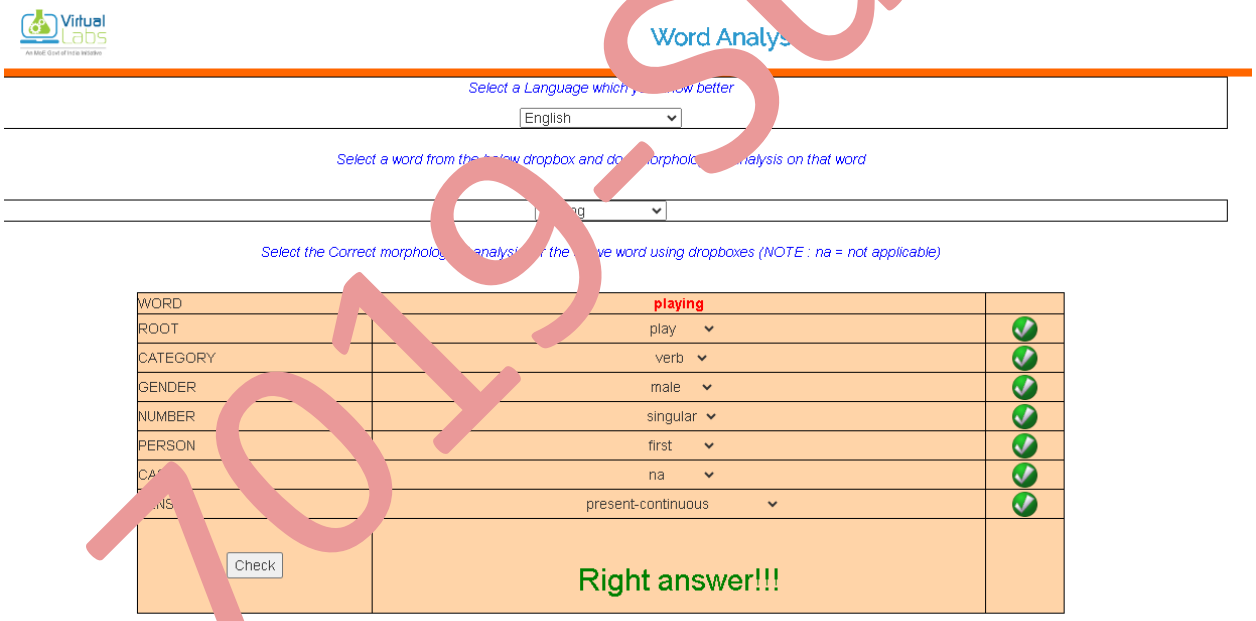
STEP 3: Select the features.

STEP 4: Click "Check" button to check your answer.

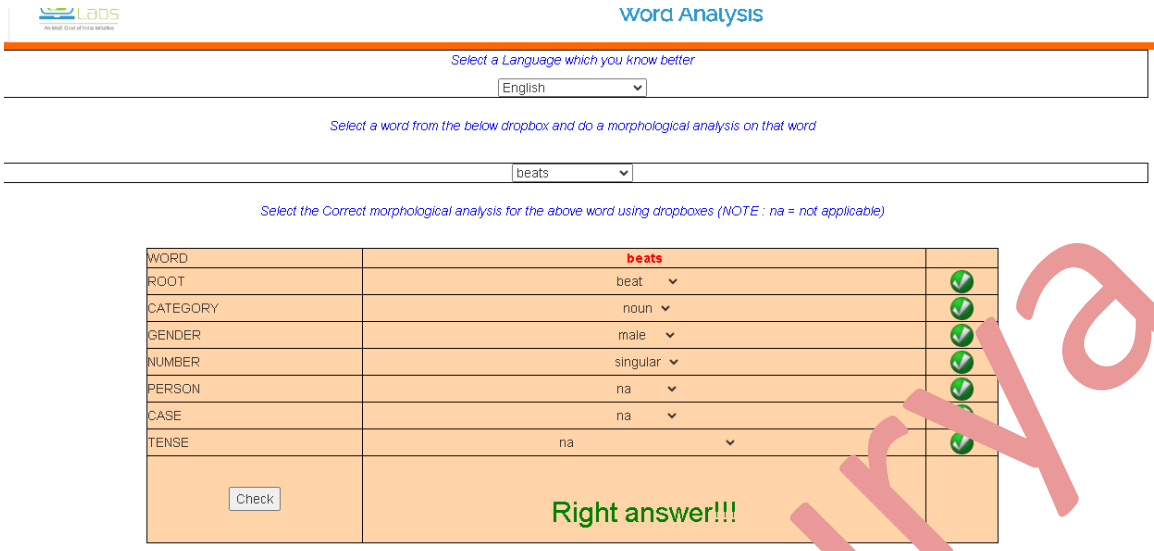
OUTPUT: Right features are marked by tick and wrong features are marked by cross

Simulation:

Inflectional Morphology:



Derivational morphology:



EXP-2: Word Generation

Aim: A word can be simple or complex. For example, the word 'cat' is simple because one cannot further decompose the word into smaller part. On the other hand, the word 'cats' is complex, because the word is made up of two parts: root 'cat' and plural suffix '-s'

Theory:
Given the root and suffix information, a word can be generated. For example,

Language	input:analysis	output:word
English	rt=boy, cat=n, num=pl	boys
English	rt=play, cat=v, num=sg, per=3, tense=pr	plays

Morphological analysis and generation: Inverse processes
Analysis may involve non-determinism, since more than one analysis is possible.
Generation is a deterministic process. In case a language allows spelling variation, then till that extent, generation would also involve non-determinism.

Objective: The objective of the experiment is to learn about morphological features of a word by analysing it.

Procedure:
STEP 1: Select the language.

OUTPUT: Drop down for selecting words will appear.

STEP 2: Select the word.

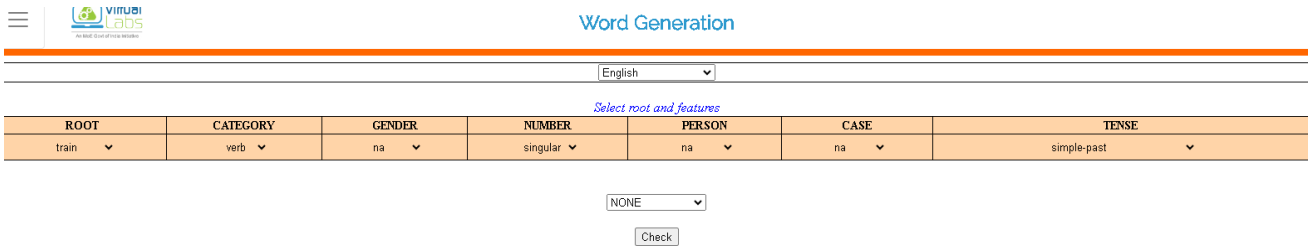
OUTPUT: Drop down for selecting features will appear.

STEP 3: Select the features.

STEP 4: Click "Check" button to check your answer.

OUTPUT: Right features are marked by tick and wrong features are marked by cross.

Simulation:



EXP-3: Morphology

Aim: Morphology is the study of the way words are built up from smaller meaning bearing units i.e., morphemes. A morpheme is the smallest meaningful linguistic unit.

Objective: The Objective of the experiment is understanding the morphology of a word by the use of Add-Delete table.

Theory:

Morph Analyser

Definition

Morphemes are considered as smallest meaningful units of language. These morphemes can either be a root word(play) or affix(-ed). Combination of these morphemes is called morphological process. So, word "played" is made out of 2 morphemes "play" and "-ed". Thus finding all parts of a word(morphemes) and thus describing properties of a word is called "Morphological Analysis". For example, "played" has information verb "play" and "past tense", so given word is past tense form of verb "play".

Analysis of a word :

बच्चों (bachchoM) = बच्चा(bachchaa)(root) + ओं(oM)(suffix) (ओं=3 plural oblique) A linguistic paradigm is the complete set of variants of a given lexeme. These variants can be classified according to shared inflectional categories (eg: number, case etc) and arranged into tables.

Paradigm for बच्चा

Case/num	Singular	Plural
Direct	बच्चा(bachchaa)	बच्चे(bachche)
oblique	बच्चे(bachche)	बच्चों (bachchoM)

Algorithm to get बच्चों(bachchoM) from बच्चा(bachchaa)

1. Take Root बच्च(bachch)आ(aa)
2. Delete आ(aa)
3. output बच्च(bachch)

4. Add ओं(oM) to output
5. Return बच्चों (bachchoM)
   Therefore आ is deleted and ओं is added to get बच्चों

Add-Delete table for बच्चा

Delete	Add	Number	Case	Variants
आ(aa)	आ(aa)	sing	dr	बच्चा(bachchaa)
आ(aa)	ए(e)	Plu	dr	बच्चे(bachche)
आ(aa)	ए(e)	Sing	ob	बच्चे(bachche)
आ(aa)	ओं(oM)	Plu	ob	बच्चों(bachcho

Paradigm Class

Words in the same paradigm class behave sim[illegible]ly, for [illegible]ample लड़क is in the same paradigm class as बच्च, so लड़का would behave similarly as बच्चा as [illegible]ey share the same paradigm class.

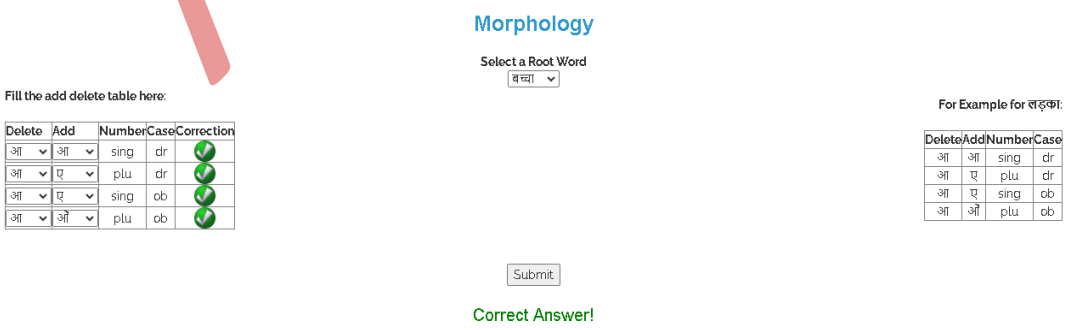
Procedure:

STEP 1: Select a word root

STEP 2: Fill the add-delet[illegible] tab[illegible]and sub[illegible]it.

STEP 3: If wrong [illegible]see th[illegible]correct a[illegible]wer or repeat STEP1.

Simulati[illegible]:

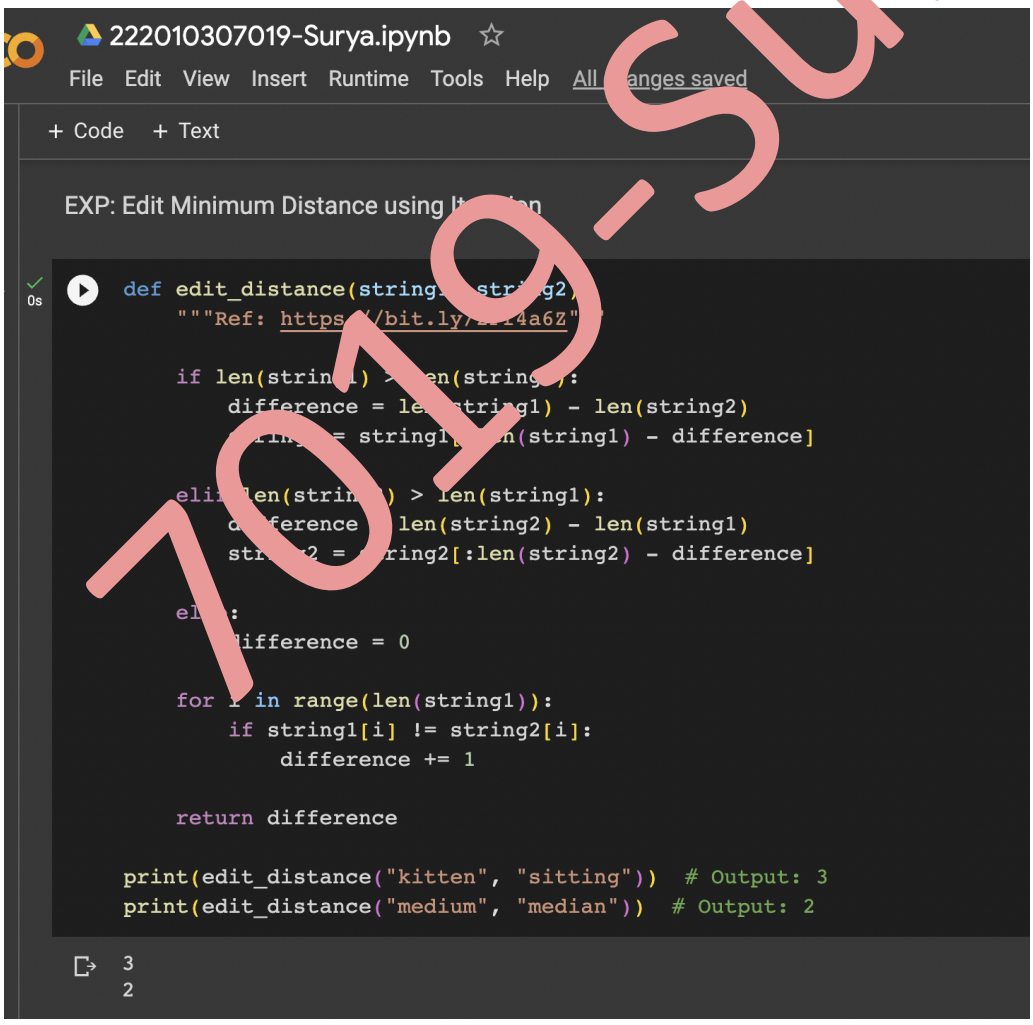


EXP: Minimum Edit Distance

In computational linguistics and computer science, edit distance is a way of quantifying the dissimilarity between two strings by counting the minimum number of operations required to transform one string into the other. Edit distances are commonly used in natural language processing for spelling correction and in bioinformatics to measure the similarity of DNA sequences.

The edit distance between two strings refers to the minimum number of character insertions, deletions, and substitutions required to change one string to the other. For example, the edit distance between "kitten" and "sitting" is three: substitute the "k" for "s", substitute the "e" for "i", and append a "g".

Program: Compute edit distance between two strings.



Program: Compute edit distance between two strings using Recursion.

```
# Below are the costs of different operations.
ins_cost = 1
del_cost = 1
sub_cost = 2

# Below function will take the two sequences and will return the distance between them.
def edit_distance_recurse(seq1, seq2, operations=[]):
    """Returns the Edit Distance between the provided two sequences."""

    if len(seq2) == 0:
        operations = operations + ([f"Delete `{seq1}` from sequence1."] i  len(  1) else [])
        return len(seq1), operations

    if len(seq1) == 0:
        operations = operations + ([f"Insert `{seq2}` into sequen e1."]  f len(seq2) else [])
        return len(seq2), operations

    if seq1[0] == seq2[0]:
        operations = operations + [f"Make no change for charac r `{seq1[0]} ."]
        return edit_distance_recurse(seq1[1:], seq2[1:],  erati s)

    # calculate cost if insertion was made
    ins_operations = operations + [f"Insert `  q2[0]  in se  nce1."]
    insertion, ins_operations = edit_distan  recurse(s  1, s q2[1:], ins_operations)

    # calculate cost if deletion was done
    del_operations = operations + [f"Delete `{seq1[0]}  from sequence1."]
    deletion, del_operations = edit_distance_recurse( q1[1:], seq2, del_operations)

    # calculate cost if substit  s d e
    sub_operations = operation  + [f"R lace `{seq1[0]}` in sequence1 with `{seq2[0]}`."]
    substitution, sub_operatio  = edi  istance_recurse(seq1[1:], seq2[1:], sub_operations)

    # calculate minimum  st
    min_cost = min(inse  + ins_cost  eletion + del_cost, substitution + sub_cost)
```

```
    if min_cost == (substitution + sub_cost):
        return min_cost, sub_operations
    elif min_cost == deletion + del_cost:
        return min_cost, del_operations
    else:
        return min_cost, ins_operations

seq1 = "numpy"
seq2 = "numexpr"
score, operations = edit_distance_recurse(seq1, seq2)

print(f"Edit Distance between `{seq1}` & `{seq2}` is: {score}")
print("Operations performed are:")
for operation in operations:
    print(operation)
```

```
Edit Distance between `numpy` & `numexpr` is: 4
Operations performed are:
Make no change for character `n`.
Make no change for character `u`.
Make no change for character `m`.
Insert `e` in sequence1.
Insert `x` in sequence1.
Make no change for character `p`.
Replace `y` in sequence1 with `r`.
```


Explanation:

1. The first code snippet calculates the edit distance using iteration by comparing characters of the two strings and counting the differences.
2. The second code snippet uses recursion to calculate the edit distance along with operations performed for insertion, deletion, and substitution.

Results:

Iterative Edit Distance:

- Edit distance between "kitten" and "sitting" is 3.
- Edit distance between "medium" and "median" is 2.

Recursive Edit Distance:

- Edit distance between "numpy" and "numexpr" is 4.
- Operations performed include insertions, deletions, and substitutions.