Edit Distance

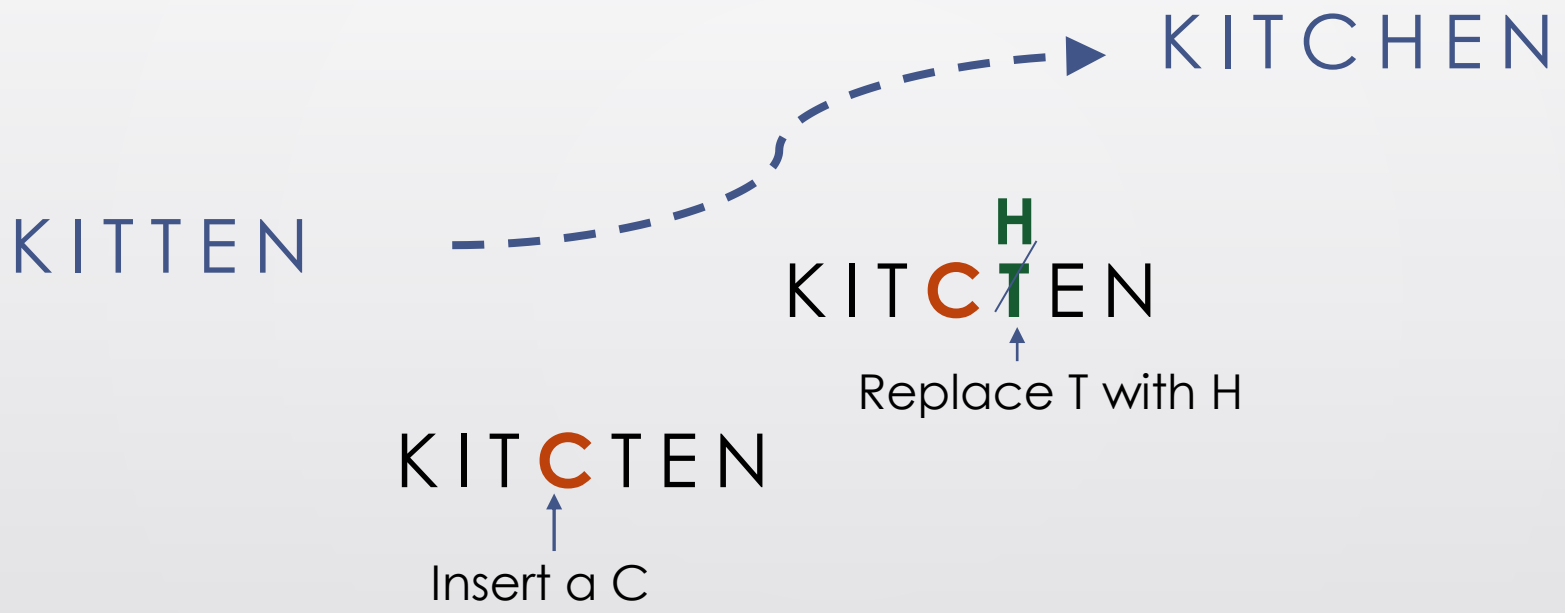
CS6370:NLP Course

Jan-May 2021

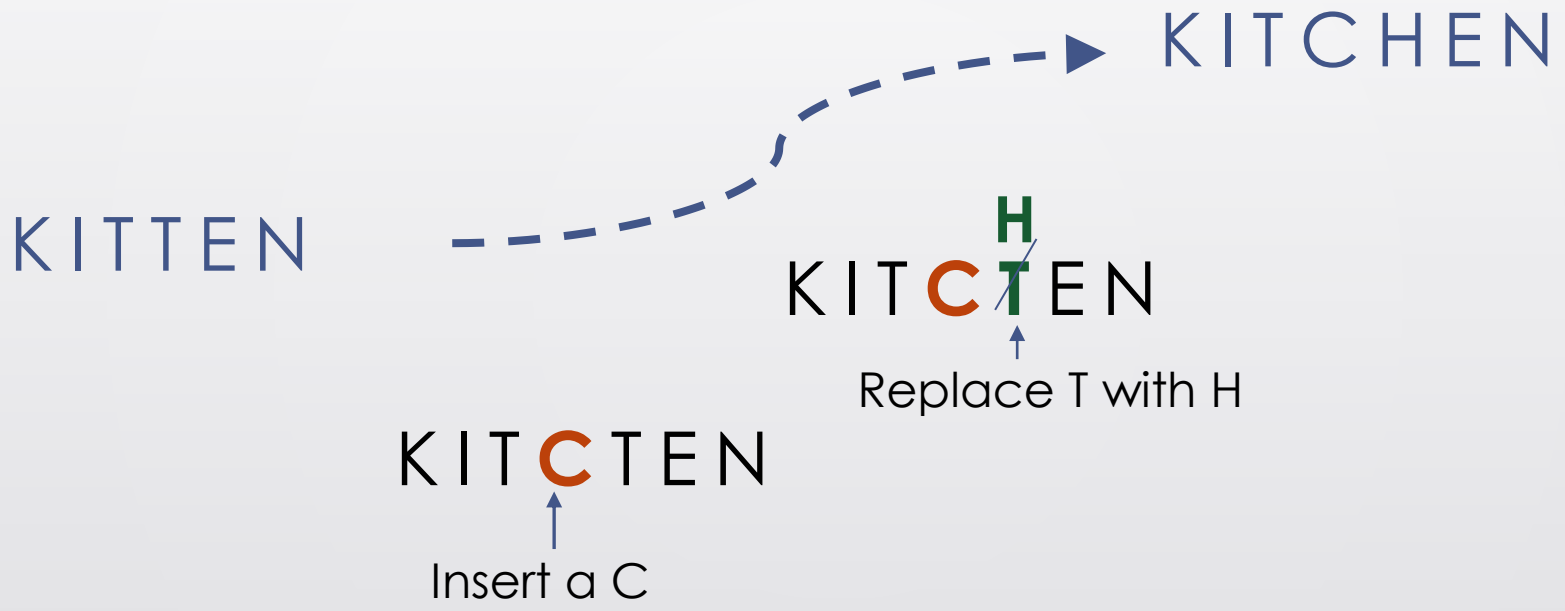
What is Edit distance?

What is Edit distance?

What is Edit distance?

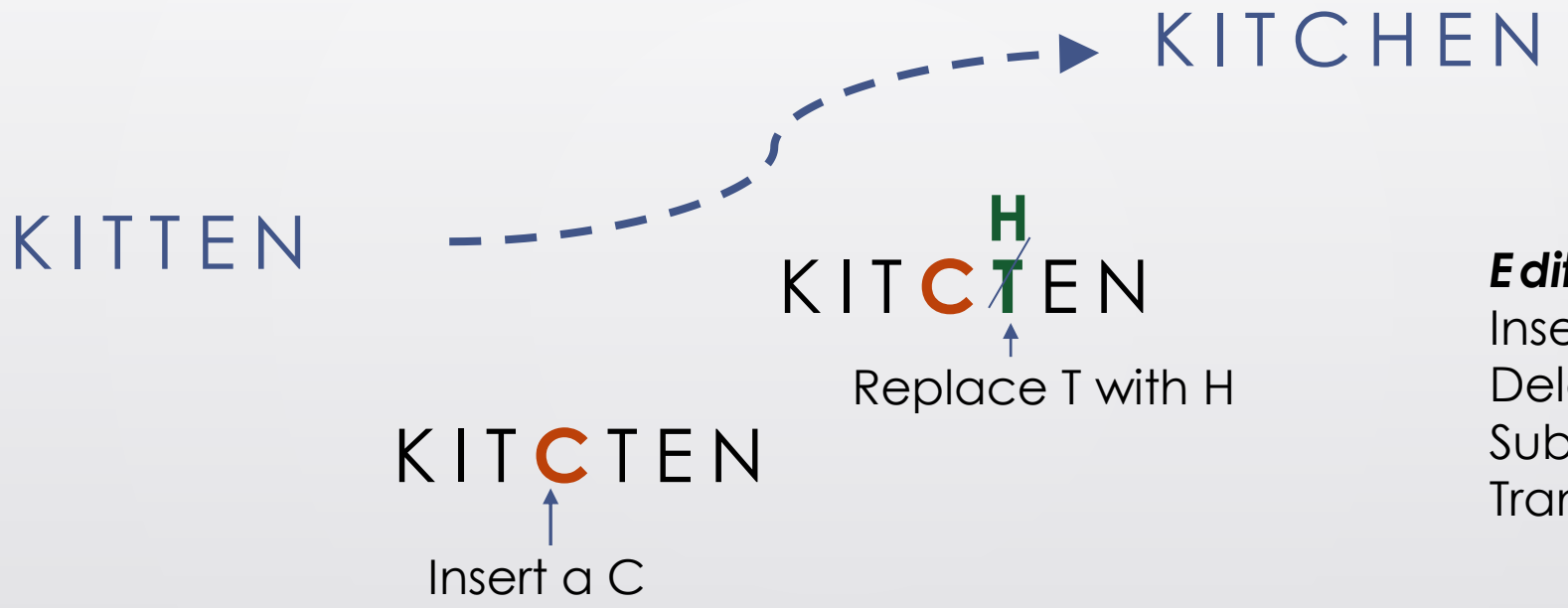


What is Edit distance?



Minimum number of edit operations needed to transform one word into another

What is Edit distance?



Edit operations
Insertion
Deletion
Substitution/Replace
Transposition (ab <-> ba)

Minimum number of edit operations needed to transform one word into another

Where is it used?

- Approximate String Matching
  - Spell Checker
    - Which is the closest candidate to ‘appel’ ?
    - Candidates: apple ; april ; append

Where is it used?

- Approximate String Matching
  - Spell Checker
    - Which is the closest candidate to ‘appel’ ?
    - Candidates: apple ; april ; append
  - Computational Biology
    - Matching of nucleotide sequences

An Example Worked out (Manual)

- **K I T T E N → K I T C H E N**
  - INSERT **C** (K I T C T E N)
  - REPLACE **T** WITH **H** (K I T C ~~T~~ H E N)
  - REPLACE **T** WITH **C** (K I T ~~T~~ C E N)
  - INSERT **H** (K I T C H E N)
  - DELETE **T** (K I T ~~T~~ E N)
  - INSERT **C** (K I T C E N)
  - INSERT **H** (K I T C H E N)

How to do it computationally?

- Enumerate all possible of ways of editing ‘KITTEN’ to ‘KITCHEN’
- Choose the one with minimum edit distance
- Computational complexity?

What is the complexity?

Naïve Recursive Algorithm

- K I T T E N → K I T C H E N

Given: two strings of length m and n

IF the first characters of the two strings are the same, then ignore them and compute the edit distance of the remaining substrings of length m-1 and n-1

Edit Distance (**K** I T T E N , **K** I T C H E N)

= 0 + Edit Distance (**I** T T E N , **I** T C H E N)

= 0 + 0 + Edit Distance (**T** T E N , **T** C H E N)

= 0 + 0 + 0 + Edit Distance(T E N , C H E N)

What is the complexity?

Naïve Recursive Algorithm

- K I T T E N → K I T C H E N

Given: two strings of length m and n

IF the first characters of the two strings are the same, then ignore them and compute the minimum edit distance of the remaining substrings of length m-1 and n-1

Edit Distance (**K** I T T E N , **K** I T C H E N)

= 0 + Edit Distance (**I** T T E N , **I** T C H E N)

= 0 + 0 + Edit Distance (**T** T E N , **T** C H E N)

= 0 + 0 + 0 + Edit Distance(T E N , C H E N)

What is the complexity?

Naïve Recursive Algorithm

T E N → C H E N

Given: two strings of length m and n

IF the first characters of the two don't match, then edit the first string using the three edit operations : insertion, deletion and substitution and compute the minimum edit distance corresponding to all three operations . The minimum of these three distances is returned as the edit distance of the given strings.

Edit Distance (T E N , C H E N)
(m, n)

= min (cost of inserting 'C' in first string + Edit Distance(T E N , H E N)
(m, n-1)

cost of deleting 'T' from first string + Edit Distance(E N , C H E N)
(m-1, n)

cost of substituting 'T' by 'C' in first string + Edit Distance(E N , H E N)
(m-1, n-1)

What is the complexity?

Naïve Recursive Algorithm

a p p l e → b o s s

Given: two strings of length m and n

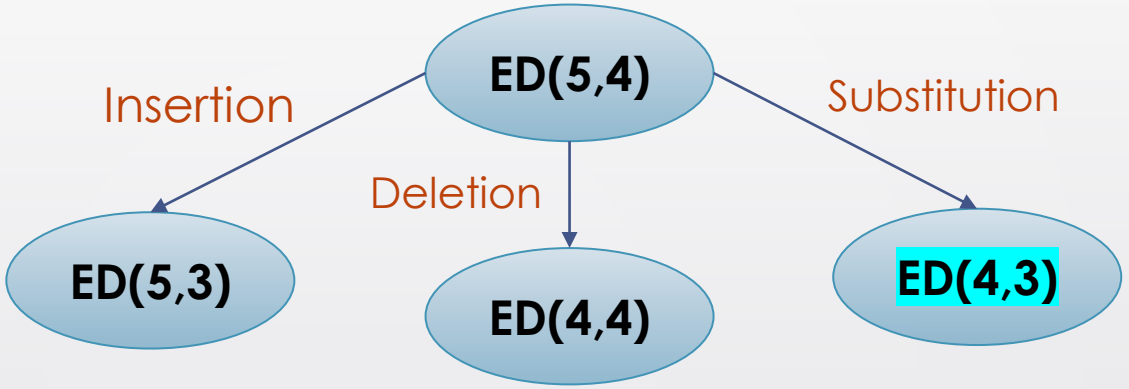
IF the first characters of the two don't match, then edit the first string using the three edit operations : insertion, deletion and substitution and compute the minimum edit distance corresponding to all three operations . The minimum of these three distances is returned as the edit distance of the given strings.

When none of the characters match,

Worst case complexity = $O(3^m)$

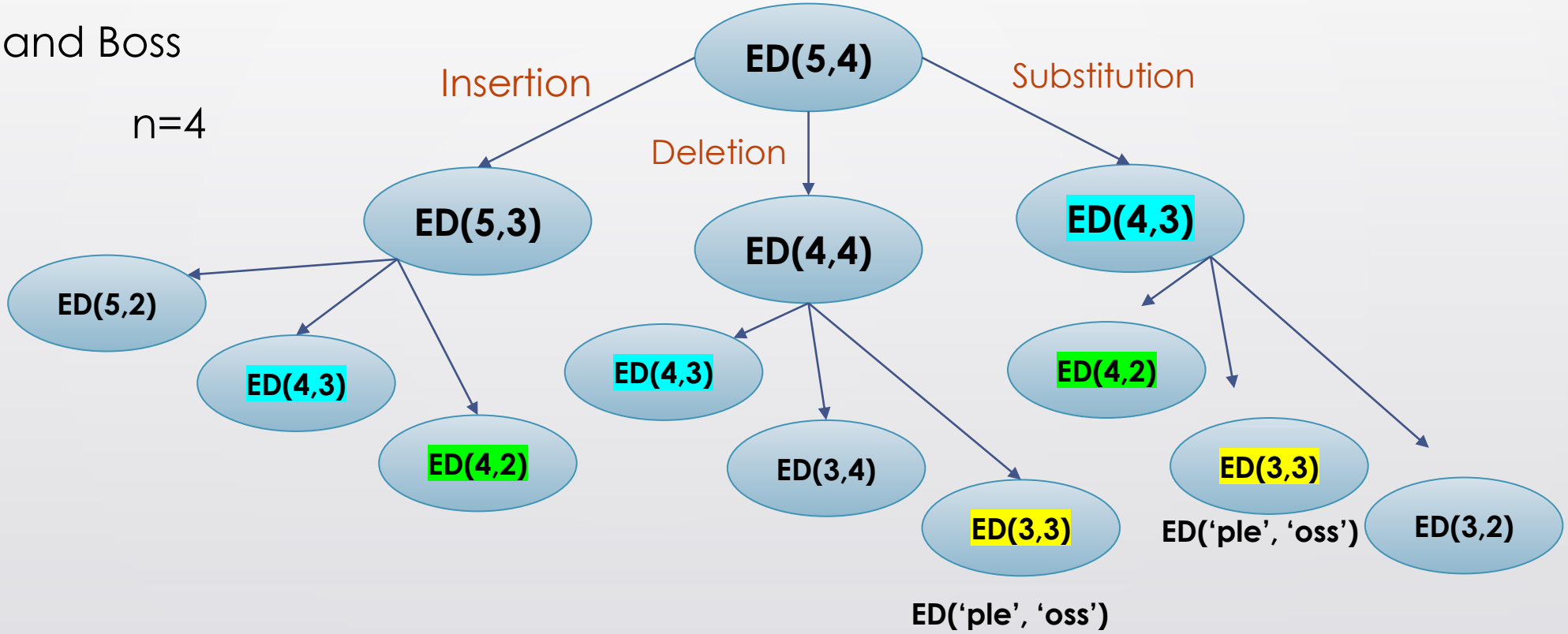
Worst Case Complexity

- Apple and Boss
- m=5 n=4



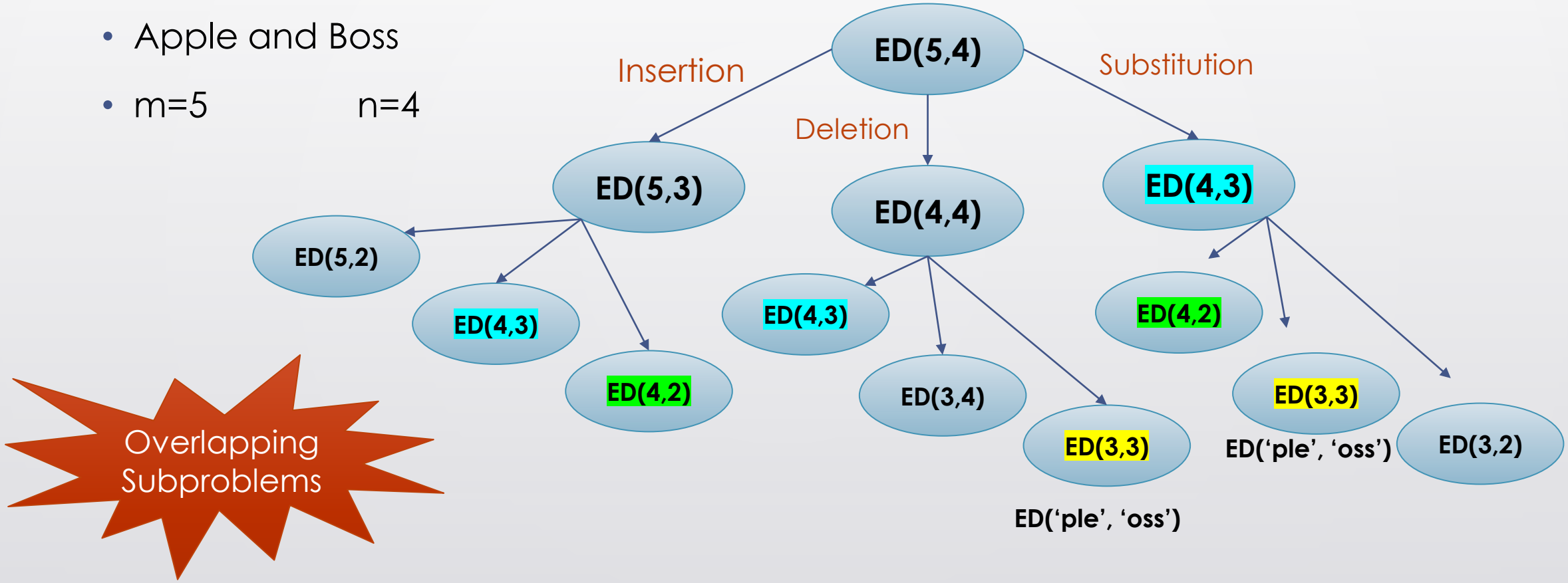
Worst Case Complexity

- Apple and Boss
- m=5 n=4



Worst Case Complexity

- Apple and Boss
- m=5 n=4



Can we do away with repeated computation of sub-problems?

- How to realize this computationally?

Dynamic Programming

What is it?

- Dynamic Programming (DP) is an algorithmic technique for solving optimization problems
- Mainly an optimization over plain recursion
- Applicable when the following properties hold:
  - Optimal substructure
  - Overlapping Subproblems

Where is it useful?

- Optimal Substructure
  - optimal solution to the problem contains within it optimal solutions to subproblems.

- Overlapping Subproblems
  - When a recursive algorithm revisits the same problem over and over again, the problem has overlapping subproblems.

- Solve each subproblem once and then store the solution in a table and when needed, perform lookup operation.

An Example : Fibonacci number calculation

- Fib (n) = Fib (n-1) + Fib (n-2)

An Example : Fibonacci number calculation

- Fib (n) = Fib (n-1) + Fib (n-2)

Plain Recursive solution

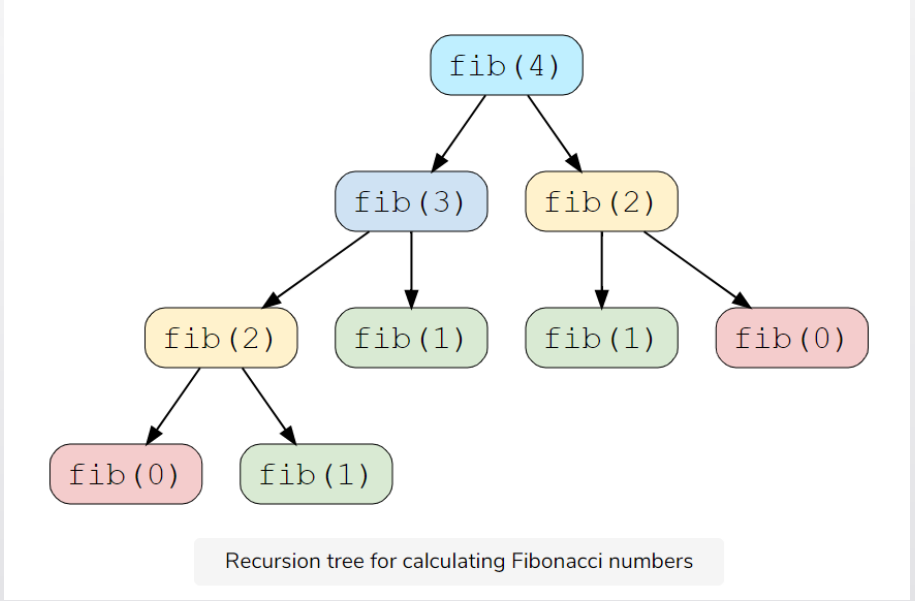
```
int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}
```

An Example : Fibonacci number calculation

- Fib (n) = Fib (n-1) + Fib (n-2)

Plain Recursive solution

```
int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}
```



Recursion tree for calculating Fibonacci numbers

Src: https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2G1pAq0OO0

DP solution

```
int fib(int n) {
   if (n==0) return 0;
   int dp[] = new int[n+1];          //table to store solutions to subproblems

   //base cases
   dp[0] = 0;     dp[1] = 1;

   for(int i=2; i<=n; i++)
            dp[i] = dp[i-1] + dp[i-2];
   return dp[n];
}
```

Back to Edit distance

Can we try a DP solution to Edit distance?

- We saw that the problem of edit distance has both the following properties
  - Overlapping subproblems
  - Optimal substructure

Recursive Formula for DP

- Distance b/n two strings P and Q of length M and N is denoted by D(M,N)

```
For each   i = 1…M
      For each   j = 1…N
                          { D(i-1,j) + 1
            D(i,j)= min   { D(i,j-1) + 1
                          { D(i-1,j-1) +    1; { if X(i) ≠ Y(j)
                                            0; { if X(i) = Y(j)
```

[3]Image taken from Coursera NLP lecture(Slightly Modified)

Recursive Formula for DP

- Distance b/n two strings P and Q of length M and N is denoted by D(M,N)

Cost of deletion

```
For each   i = 1…M
      For each   j = 1…N
                          { D(i-1,j) + 1
          D(i,j)= min     { D(i,j-1) + 1
                          { D(i-1,j-1) +    1; { if X(i) ≠ Y(j)
                                            0; { if X(i) = Y(j)
```

[3]Image taken from Coursera NLP lecture(Slightly Modified)

Recursive Formula for DP

- Distance b/n two strings P and Q of length M and N is denoted by D(M,N)

```
For each   i = 1…M
      For each   j = 1…N
                          { D(i-1,j) + 1
          D(i,j)= min     { D(i,j-1) + 1
                          { D(i-1,j-1) +    1; if X(i) ≠ Y(j)
                                            0; if X(i) = Y(j)
```

Cost of insertion

[3]Image taken from Coursera NLP lecture(Slightly Modified)

Recursive Formula for DP

- Distance b/n two strings P and Q of length M and N is denoted by D(M,N)

```
For each   i = 1…M
      For each   j = 1…N
                          { D(i-1,j) + 1
            D(i,j)= min   { D(i,j-1) + 1
                          { D(i-1,j-1) +   { 1; if X(i) ≠ Y(j)
                                           { 0; if X(i) = Y(j)
```

Cost of substitution

[3] Image taken from Coursera NLP lecture(Slightly Modified)

Recursive Formula for DP

- Base Conditions

$$D(i,0) = i$$
$$D(0,j) = j$$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 1; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

Costs of Edits

- Let's assume that the cost of all edit operations are the same and equal to 1
- Sometimes, cost may be different for different edit operations

Note:

- When only Insertion, Deletion and Substitution are considered, it is called Levenshtein distance
- When transposition is also considered, it is called Damerau-Levenshtein distance

Distance between two strings PARK and SPAKE

- m =4
- n = 5

			1	2	3	4
		_	P	A	R	K
	_	0	1	2	3	4
1	S	1				
2	P	2				
3	A	3				
4	K	4				
5	E	5				

Base Cases

Distance between two strings PARK and SPAKE

- m =4
- n = 5

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 1; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

		j	1	2	3	4
			P	A	R	K
i		0	1	2	3	4
1	S	1				
2	P	2				
3	A	3				
4	K	4				
5	E	5				

Distance between two strings PARK and SPAKE

- m =4
- n = 5

			1	2	3	4
			P	A	R	K
		0	1	2	3	4
1	S	1	1	2	3	4
2	P	2	1	2	3	4
3	A	3	2	1	2	3
4	K	4	3	2	2	2
5	E	5	4	3	3	3

Blue numbers help to keep track of the path via which the minimum edit distance 3 is achieved

Algorithm

```
function MIN-EDIT-DISTANCE(target, source) returns min-distance

  n ← LENGTH(target)
  m ← LENGTH(source)
  Create a distance matrix distance[n+1,m+1]
  distance[0,0] ← 0
  for each column i from 0 to n do
    for each row j from 0 to m do
      distance[i,j] ← MIN( distance[i−1,j] + ins-cost(target_j),
                           distance[i−1,j−1] + subst-cost(source_j, target_i),
                           distance[i,j−1] + ins-cost(source_j))
```

Ref: NLP Textbook of Jurafsky and Martin

- Runtime is O(mn). A polynomial time algorithm.

For Practice

- Compute the edit distance table for KITTEN and KITCHEN

References

- https://www.geeksforgeeks.org/edit-distance-dp-5/
- Slides of Dileep Kumar Pattipati, taken from CS6370:2018