Algorithm Design Techniques

Greedy Algorithms
Dynamic Programming

Edit Distance

- Turn one word into another
- Three operations are possible: insert, delete, replace
- How do we determine the most efficient operation

- Example: **tire** into **admire**

Edit Distance

- Turn one word into another
- Three operations are possible: insert, delete, replace
- How do we determine the most efficient operation

- Example: **tire** into **admire**
  - we see that **ire** and **ire** match; don’t touch them

Edit Distance

- Turn one word into another
- Three operations are possible: insert, delete, ***replace***
- How do we determine the most efficient operation

- Example: **tire** into **admire**
  - we see that **ire** and **ire** match; don’t touch them
  - replace **t** with **a**: **aire**; one operation

Edit Distance

- Turn one word into another
- Three operations are possible: ***insert***, delete, replace
- How do we determine the most efficient operation

- Example: **tire** into **admire**
  - we see that **ire** and **ire** match; don’t touch them
  - replace **t** with **a**: **aire**; one operation
  - insert **d**: **adire**; one operation

Edit Distance

- Turn one word into another
- Three operations are possible: ***insert***, delete, replace
- How do we determine the most efficient operation

- Example: **tire** into **admire**
  - we see that **ire** and **ire** match; don’t touch them
  - replace **t** with **a**: **aire**; one operation
  - insert **d**: **adire**; one operation
  - insert **m**: **admire**; one operation

Edit Distance

- How do we program this to determine this in an efficient way? Dynamic Programming!
  - All possible sub-strings of **tire** into **admire**

		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1	2	3	4	5	6
i	2	2	2	3	3	4	5
r	3	3	3	3	4	3	4
e	4	4	4	4	4	4	3

Edit Distance

- Where did this table come from?

		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1	2	3	4	5	6
i	2	2	2	3	3	4	5
r	3	3	3	3	4	3	4
e	4	4	4	4	4	4	3

Edit Distance

- Where did this table come from?
  - Start with distance from blank to each of the sub-words (sub-problems)

		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1						
i	2						
r	3						
e	4						

Edit Distance

- Where did this table come from?
  - Start with distance from blank to blank, 0
  - Then go to its neighbors, and compute the distance

blank to a: 1

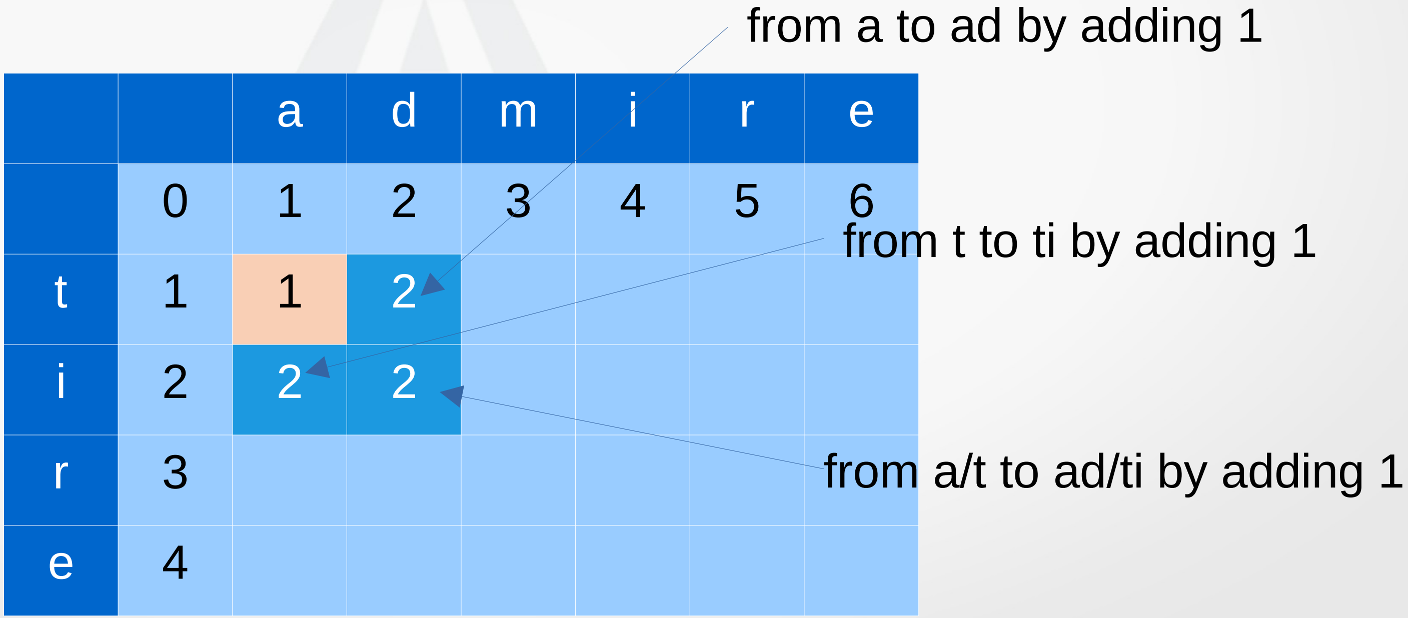
		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1					
i	2						
r	3						
e	4						

blank to a (or t): 1

t to blank: 1

Edit Distance

- Where did this table come from?
  - Just repeatedly work through cells with neighbors
  - Let’s look at neighbors of t/a



		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1	2				
i	2	2	2				
r	3						
e	4						

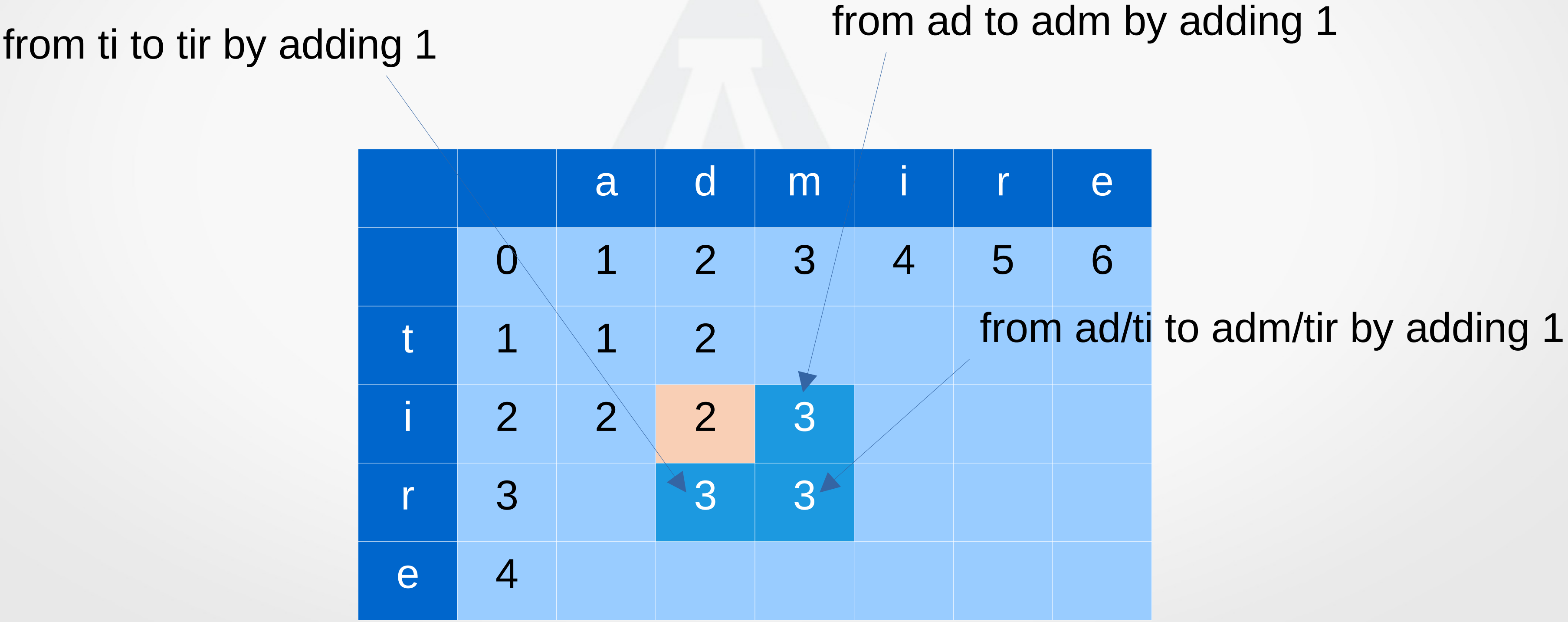
from a to ad by adding 1

from t to ti by adding 1

from a/t to ad/ti by adding 1

Edit Distance

- Where did this table come from?



from ti to tir by adding 1

from ad to adm by adding 1

from ad/ti to adm/tir by adding 1

		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1	2				
i	2	2	2	3			
r	3		3	3			
e	4						

Edit Distance

- Where did this table come from?
  - Implied no-operation (matching)
  - The sub-problem of i to i, r to r, and e to e have shorter distance, therefore we keep them
  - Alternatively, the sub-problem of ire to ire is a shorter distance, therefore we keep it

		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1	2				
i	2	2	2	3	3		
r	3		3	3		3	
e	4						3

match, no operation

Edit Distance

- Final result

		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1	2	3	4	5	6
i	2	2	2	3	3	4	5
r	3	3	3	3	4	3	4
e	4	4	4	4	4	4	3

Edit Distance – Full Code

```java
public static int minDistance(String word1, String word2) {
    int len1 = word1.length();
    int len2 = word2.length();
    int[][] dp = new int[len1 + 1][len2 + 1];

    // it takes at least this many operations
    for (int i = 0; i <= len1; i++) { dp[i][0] = i; }
    for (int j = 0; j <= len2; j++) { dp[0][j] = j; }

    for (int i = 0; i < len1; i++) {
        for (int j = 0; j < len2; j++) {
            if (word1.charAt(i) == word2.charAt(j)) {
                //no op needed to transform, same number
                dp[i + 1][j + 1] = dp[i][j];
            } else {
                // Select the best from the three operations
                int replace = dp[i][j] + 1;
                int delete= dp[i][j + 1] + 1;
                int insert = dp[i + 1][j] + 1;

                int min = replace > insert ? insert : replace;
                min = delete > min ? min : delete;
                dp[i + 1][j + 1] = min;
            }
        }
    }
}
```

Edit Distance

- Let’s take the table for a spin
  - how to turn a **blank** into **a** (1 operation)

		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1	2	3	4	5	6
i	2	2	2	3	3	4	5
r	3	3	3	3	4	3	4
e	4	4	4	4	4	4	3

Edit Distance

- Let’s take the table for a spin
  - how to turn a **blank** into **ad** (2 operations)

		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1	2	3	4	5	6
i	2	2	2	3	3	4	5
r	3	3	3	3	4	3	4
e	4	4	4	4	4	4	3

Edit Distance

- Let’s take the table for a spin
  - how to turn **t** into **a** (1 operation)

		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1	2	3	4	5	6
i	2	2	2	3	3	4	5
r	3	3	3	3	4	3	4
e	4	4	4	4	4	4	3

Edit Distance

- Let’s take the table for a spin
  - how to turn **ti** into **ad** (2 operations)

		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1	2	3	4	5	6
i	2	2	2	3	3	4	5
r	3	3	3	3	4	3	4
e	4	4	4	4	4	4	3

Edit Distance

- Let’s take the table for a spin
  - how to turn **tire** into **admi** (4 operations)

		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1	2	3	4	5	6
i	2	2	2	3	3	4	5
r	3	3	3	3	4	3	4
e	4	4	4	4	4	4	3

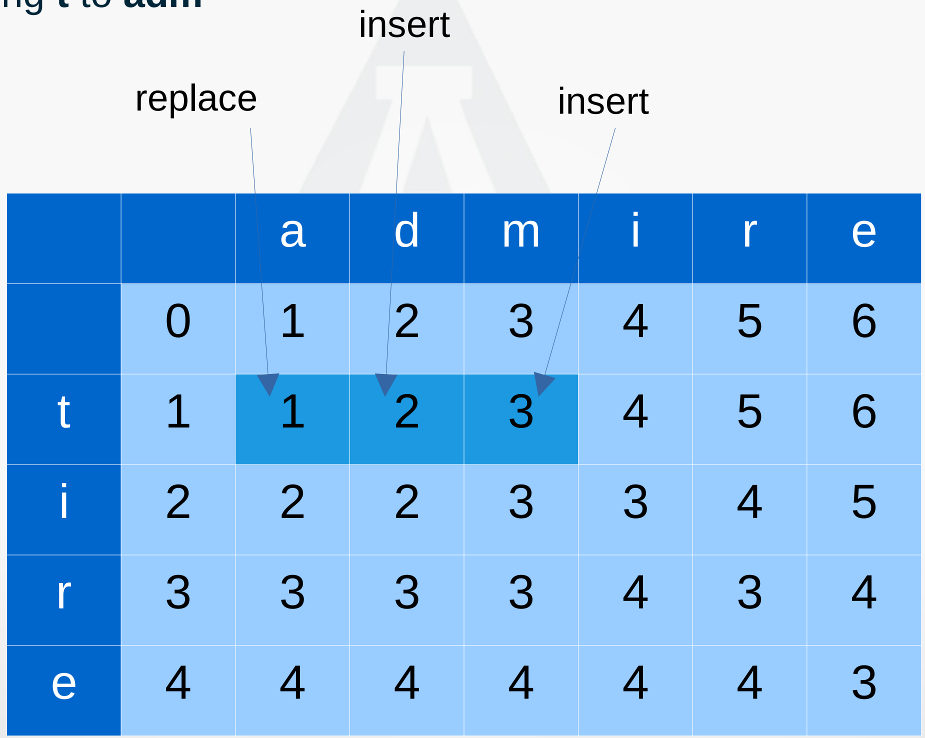
Edit Distance

- Finally, let’s find the edit distance from **tire** to **admire**
  - (notice that **ire** and **ire** match)
  - That means we focus on converting **t** to **adm**

		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1	2	3	4	5	6
i	2	2	2	3	3	4	5
r	3	3	3	3	4	3	4
e	4	4	4	4	4	4	3

Edit Distance

- How do we program this to determine this in an efficient way? Dynamic Programming!
  - Focus on converting **t** to **adm**
    - 3 operations



		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1	2	3	4	5	6
i	2	2	2	3	3	4	5
r	3	3	3	3	4	3	4
e	4	4	4	4	4	4	3

Edit Distance

- How do we program this to determine this in an efficient way? Dynamic Programming!
  - Focus on converting **t** to **adm**
    - 3 operations
    - Then no-ops all the way from t/adm to tire/admire

		a	d	m	i	r	e
	0	1	2	3	4	5	6
t	1	1	2	3	4	5	6
i	2	2	2	3	3	4	5
r	3	3	3	3	4	3	4
e	4	4	4	4	4	4	3

match