



Design and analysis of algorithms

Lecture 25& 26

Edyta Szymańska

edyta@cc.gatech.edu

Edit distance

Application: Spell Checkers

Edit distance

Application: Spell Checkers
occurrence ?

Edit distance

Application: Spell Checkers

ocurrance ?

“Perhaps you mean occurrence ?”

Edit distance

Application: Spell Checkers

ocurrance ?

“Perhaps you mean occurrence ?”

ocurrance and

occurrence are *close by*

Edit distance

Application: Spell Checkers

ocurrance ?

“Perhaps you mean occurrence ?”

ocurrance and

occurrence are *close by*

How to define “closeness” ?

minimum number of “edits”-insertions, deletions and substitutions of characters needed to transform string x into string y .

Edit distance

Application: Spell Checkers

ocurrance ?

“Perhaps you mean occurrence ?”

ocurrance and

occurrence are *close by*

How to define “closeness” ?

minimum number of “edits”-insertions, deletions and substitutions of characters needed to transform string x into string y .

Example 1: $d(\text{to}, \text{fro}) = 2$, $\text{to} \rightarrow \text{fo}$ (substitution) $\rightarrow \text{fro}$ (insertion)

Edit distance

Solution ? - dynamic programming ?

Edit distance

Solution ? - dynamic programming ?
What are the subproblems?

Edit distance

Solution ? - dynamic programming ?

What are the subproblems?

Example 2: $d(\textit{exponential}, \textit{polynomial}) = ?$

Edit distance

Solution ? - dynamic programming ?

What are the subproblems?

Example 2: $d(\text{exponential}, \text{polynomial}) = ?$

possible subproblem: the smallest number of edits needed to transform some prefix of x , say **expo** into some prefix of y , say **pol**.

Edit distance

Solution ? - dynamic programming ?

What are the subproblems?

Example 2: $d(\textit{exponential}, \textit{polynomial}) = ?$

possible subproblem: the smallest number of edits needed to transform some prefix of x , say **expo** into some prefix of y , say **pol**.

How to transform **expo** to **pol** ?

Edit distance

Solution ? - dynamic programming ?

What are the subproblems?

Example 2: $d(\text{exponential}, \text{polynomial}) = ?$

possible subproblem: the smallest number of edits needed to transform some prefix of x , say **expo** into some prefix of y , say **pol**.

How to transform **expo** to **pol** ?

- ⑥ left to right
- ⑥ right to left
- ⑥ middle-out ?

Edit distance

INPUT: two words $x : |x| = m$ and $y : |y| = n$

- ⑥ *insert* $y[j]$,
- ⑥ *substitute* $x[i] \rightarrow y[j]$

Edit distance

INPUT: two words $x : |x| = m$ and $y : |y| = n$

OUTPUT: $E(m, n)$, where $E(i, j)$ is the edit distance between prefixes $x[1 \dots i]$ and $y[1 \dots j]$

- ⑥ *insert* $y[j]$,
- ⑥ *substitute* $x[i] \rightarrow y[j]$

Edit distance

INPUT: two words $x : |x| = m$ and $y : |y| = n$

OUTPUT: $E(m, n)$, where $E(i, j)$ is the edit distance between prefixes $x[1 \dots i]$ and $y[1 \dots j]$

To figure out $E(i, j)$ check $x[i] = y[j]$.

If $x[i] \neq y[j]$ then we have one of the following operations:

- ⑥ *delete* $x[i]$,
- ⑥ *insert* $y[j]$,
- ⑥ *substitute* $x[i] \rightarrow y[j]$

Edit distance

INPUT: two words $x : |x| = m$ and $y : |y| = n$

OUTPUT: $E(m, n)$, where $E(i, j)$ is the edit distance between prefixes $x[1 \dots i]$ and $y[1 \dots j]$

To figure out $E(i, j)$ check $x[i] = y[j]$.

If $x[i] \neq y[j]$ then we have one of the following operations:

- ⑥ *delete* $x[i]$, $E(i, j) = E(i - 1, j) + 1$
- ⑥ *insert* $y[j]$, $E(i, j) = E(i, j - 1) + 1$
- ⑥ *substitute* $x[i] \rightarrow y[j]$ $E(i, j) = E(i - 1, j - 1) + 1$

Edit distance

INPUT: two words $x : |x| = m$ and $y : |y| = n$

OUTPUT: $E(m, n)$, where $E(i, j)$ is the edit distance between prefixes $x[1 \dots i]$ and $y[1 \dots j]$

To figure out $E(i, j)$ check $x[i] = y[j]$.

If $x[i] \neq y[j]$ then we have one of the following operations:

⑥ *delete* $x[i]$, $E(i, j) = E(i - 1, j) + 1$

⑥ *insert* $y[j]$, $E(i, j) = E(i, j - 1) + 1$

⑥ *substitute* $x[i] \rightarrow y[j]$ $E(i, j) = E(i - 1, j - 1) + 1$

Otherwise ($x[i] = y[j]$) we have $E(i, j) := E(i - 1, j - 1)$

Edit distance

INPUT: two words $x : |x| = m$ and $y : |y| = n$

OUTPUT: $E(m, n)$, where $E(i, j)$ is the edit distance between prefixes $x[1 \dots i]$ and $y[1 \dots j]$

To figure out $E(i, j)$ check $x[i] = y[j]$.

If $x[i] \neq y[j]$ then we have one of the following operations:

⑥ *delete* $x[i]$, $E(i, j) = E(i - 1, j) + 1$

⑥ *insert* $y[j]$, $E(i, j) = E(i, j - 1) + 1$

⑥ *substitute* $x[i] \rightarrow y[j]$ $E(i, j) = E(i - 1, j - 1) + 1$

Otherwise ($x[i] = y[j]$) we have $E(i, j) := E(i - 1, j - 1)$

Base case: $E(0, j) = j$ and $E(i, 0) = i$.

Edit distance

INPUT: two words $x : |x| = m$ and $y : |y| = n$

OUTPUT: $E(m, n)$, where $E(i, j)$ is the edit distance between prefixes $x[1 \dots i]$ and $y[1 \dots j]$

To figure out $E(i, j)$ check $x[i] = y[j]$.

If $x[i] \neq y[j]$ then we have one of the following operations:

⑥ *delete* $x[i]$, $E(i, j) = E(i - 1, j) + 1$

⑥ *insert* $y[j]$, $E(i, j) = E(i, j - 1) + 1$

⑥ *substitute* $x[i] \rightarrow y[j]$ $E(i, j) = E(i - 1, j - 1) + 1$

Otherwise ($x[i] = y[j]$) we have $E(i, j) := E(i - 1, j - 1)$

Base case: $E(0, j) = j$ and $E(i, 0) = i$. Let

$$\text{diff}(i, j) = \begin{cases} 1 & \text{if } x[i] \neq y[j] \\ 0 & \text{otherwise} \end{cases}$$

Algorithm for Edit distance

```
EDIT_DISTANCE( $x, y$ )
for  $i := 1$  to  $m$  do  $E(i, 0) := i$ 
for  $j := 1$  to  $n$  do  $E(0, j) := j$ 
for  $i := 1$  to  $m$  do
  for  $j := 1$  to  $n$  do
     $E(i, j) =$ 
     $\min\{E(i-1, j) + 1, E(i, j-1) + 1, E(i-1, j-1) + \text{diff}(i, j)\}$ 
return  $E(m, n)$ 
```

Algorithm for Edit distance

```
EDIT_DISTANCE( $x, y$ )
for  $i := 1$  to  $m$  do  $E(i, 0) := i$ 
for  $j := 1$  to  $n$  do  $E(0, j) := j$ 
for  $i := 1$  to  $m$  do
  for  $j := 1$  to  $n$  do
     $E(i, j) =$ 
     $\min\{E(i-1, j) + 1, E(i, j-1) + 1, E(i-1, j-1) + \text{diff}(i, j)\}$ 
return  $E(m, n)$ 
```

The collection of subproblems forms a two dimensional table $E(i, j)$

Algorithm for Edit distance

```
EDIT_DISTANCE( $x, y$ )
for  $i := 1$  to  $m$  do  $E(i, 0) := i$ 
for  $j := 1$  to  $n$  do  $E(0, j) := j$ 
for  $i := 1$  to  $m$  do
  for  $j := 1$  to  $n$  do
     $E(i, j) =$ 
     $\min\{E(i-1, j) + 1, E(i, j-1) + 1, E(i-1, j-1) + \text{diff}(i, j)\}$ 
return  $E(m, n)$ 
```

The collection of subproblems forms a two dimensional table $E(i, j)$

Running time: $O(mn)$

Edit distance - solution

	j	P	O	L	Y	N	O	M	I	A	L
i	0	1	2	3	4	5	6	7	8	9	10
E	1	1	2	3	4	5	6	7	8	9	10
X	2	2	2	3	4	5	6	7	8	9	10
P	3	2	3	3	4	5	6	7	8	9	10
O	4	3	2	3	4	5	5	6	7	8	9
N	5	4	3	3	4	4	5	6	7	8	9
E	6	5	4	4	4	5	5	6	7	8	9
N	7	6	5	5	5	4	5	6	7	8	9
T	8	7	6	6	6	5	5	6	7	8	9
I	9	8	7	7	7	6	6	6	6	7	8
A	10	9	8	8	8	7	7	7	7	6	8
L	11	10	9	8	9	8	8	8	8	7	6

Edit distance - solution

	j	P	O	L	Y	N	O	M	I	A	L
i	0	1	2	3	4	5	6	7	8	9	10
E	1	1	2	3	4	5	6	7	8	9	10
X	2	2	2	3	4	5	6	7	8	9	10
P	3	2	3	3	4	5	6	7	8	9	10
O	4	3	2	3	4	5	5	6	7	8	9
N	5	4	3	3	4	4	5	6	7	8	9
E	6	5	4	4	4	5	5	6	7	8	9
N	7	6	5	5	5	4	5	6	7	8	9
T	8	7	6	6	6	5	5	6	7	8	9
I	9	8	7	7	7	6	6	6	6	7	8
A	10	9	8	8	8	7	7	7	7	6	8
L	11	10	9	8	9	8	8	8	8	7	6

Edit distance - solution

	j	P	O	L	Y	N	O	M	I	A	L
i	0	1	2	3	4	5	6	7	8	9	10
E	1	1	2	3	4	5	6	7	8	9	10
X	2	2	2	3	4	5	6	7	8	9	10
P	3	2	3	3	4	5	6	7	8	9	10
O	4	3	2	3	4	5	5	6	7	8	9
N	5	4	3	3	4	4	5	6	7	8	9
E	6	5	4	4	4	5	5	6	7	8	9
N	7	6	5	5	5	4	5	6	7	8	9
T	8	7	6	6	6	5	5	6	7	8	9
I	9	8	7	7	7	6	6	6	6	7	8
A	10	9	8	8	8	7	7	7	7	6	8
L	11	10	9	8	9	8	8	8	8	7	6

Edit distance - solution

	j	P	O	L	Y	N	O	M	I	A	L
i	0	1	2	3	4	5	6	7	8	9	10
E	1	1	2	3	4	5	6	7	8	9	10
X	2	2	2	3	4	5	6	7	8	9	10
P	3	2	3	3	4	5	6	7	8	9	10
O	4	3	2	3	4	5	5	6	7	8	9
N	5	4	3	3	4	4	5	6	7	8	9
E	6	5	4	4	4	5	5	6	7	8	9
N	7	6	5	5	5	4	5	6	7	8	9
T	8	7	6	6	6	5	5	6	7	8	9
I	9	8	7	7	7	6	6	6	6	7	8
A	10	9	8	8	8	7	7	7	7	6	8
L	11	10	9	8	9	8	8	8	8	7	6

The underlying dag

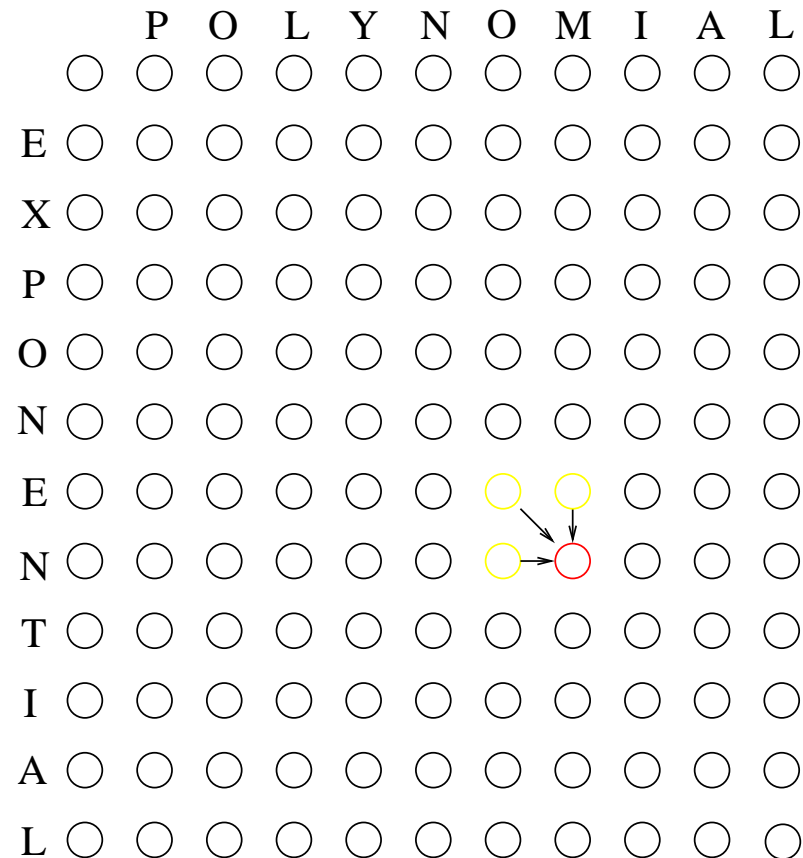
Every dynamic programming algorithm has an underlying dag structure

The underlying dag

Nodes: positions (i, j) , Edges (precedence constraints):
 $(i - 1, j) \rightarrow (i, j)$, $(i, j - 1) \rightarrow (i, j)$, $(i - 1, j - 1) \rightarrow (i, j)$

The underlying dag

Nodes: positions (i, j) , Edges (precedence constraints):
 $(i - 1, j) \rightarrow (i, j)$, $(i, j - 1) \rightarrow (i, j)$, $(i - 1, j - 1) \rightarrow (i, j)$



The underlying dag

Nodes: positions (i, j) , Edges (precedence constraints):
 $(i - 1, j) \rightarrow (i, j)$, $(i, j - 1) \rightarrow (i, j)$, $(i - 1, j - 1) \rightarrow (i, j)$
In fact, we can reduce the Edit Distance Problem to the shortest paths in the dag with weights on edges. How ?

The underlying dag

Nodes: positions (i, j) , Edges (precedence constraints):
 $(i - 1, j) \rightarrow (i, j)$, $(i, j - 1) \rightarrow (i, j)$, $(i - 1, j - 1) \rightarrow (i, j)$
In fact, we can reduce the Edit Distance Problem to the shortest paths in the dag with weights on edges. How ?

The underlying dag

Nodes: positions (i, j) , Edges (precedence constraints):
 $(i - 1, j) \rightarrow (i, j)$, $(i, j - 1) \rightarrow (i, j)$, $(i - 1, j - 1) \rightarrow (i, j)$
In fact, we can reduce the Edit Distance Problem to the shortest paths in the dag with weights on edges. How ?

Knapsack

INPUT: n items of weights w_1, \dots, w_n and values v_1, v_2, \dots, v_n , a knapsack of fixed weight W

Knapsack

INPUT: n items of weights w_1, \dots, w_n and values v_1, v_2, \dots, v_n , a knapsack of fixed weight W

OUTPUT: find the most valuable subset S of items that fit into the given knapsack, i.e. S maximizes $\sum_{i \in S} v_i$ subject to the

restriction $\sum_{i \in S} w_i \leq W$.

Knapsack

INPUT: n items of weights w_1, \dots, w_n and values v_1, v_2, \dots, v_n , a knapsack of fixed weight W

OUTPUT: find the most valuable subset S of items that fit into the given knapsack, i.e. S maximizes $\sum_{i \in S} v_i$ subject to the

restriction $\sum_{i \in S} w_i \leq W$.

The problem has more parameters than our previous examples: longest increasing subsequence, edit distance

Knapsack

Example: $W = 10$,

item	weight	value
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$9

Knapsack

Example: $W = 10$,

item	weight	value
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$9

Version I: unlimited quantities of each item are available

Knapsack

Example: $W = 10$,

item	weight	value
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$9

Version I: unlimited quantities of each item are available
optimal solution: $1 \times \$30 + 2 \times \$9 = \$48$

Knapsack

Example: $W = 10$,

item	weight	value
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$9

Version I: unlimited quantities of each item are available

optimal solution: $1 \times \$30 + 2 \times \$9 = \$48$

Version II: only one of each item is available

Knapsack

Example: $W = 10$,

item	weight	value
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$9

Version I: unlimited quantities of each item are available

optimal solution: $1 \times \$30 + 2 \times \$9 = \$48$

Version II: only one of each item is available

optimal solution: $\$30 + \$16 = \$46$

Knapsack

Example: $W = 10$,

item	weight	value
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$9

Version I: unlimited quantities of each item are available

optimal solution: $1 \times \$30 + 2 \times \$9 = \$48$

Version II: only one of each item is available

optimal solution: $\$30 + \$16 = \$46$

Neither version of this problem is likely to have a polynomial time algorithm.

Knapsack (with repetitions)

Can a greedy method find an optimal solution for this problem?

Knapsack (with repetitions)

Subproblems ?

- ⑥ smaller knapsack capacities: $w \leq W$
- ⑥ fewer items: $1, 2, \dots, j, j \leq n$

Knapsack (with repetitions)

Subproblems ?

- ⑥ smaller knapsack capacities: $w \leq W$
- ⑥ fewer items: $1, 2, \dots, j, j \leq n$

Define

$K(w)$ = maximum value achievable with a knapsack of capacity w

Knapsack (with repetitions)

Subproblems ?

- ⑥ smaller knapsack capacities: $w \leq W$
- ⑥ fewer items: $1, 2, \dots, j, j \leq n$

Define

$K(w)$ = maximum value achievable with a knapsack of capacity w

How to express $K(w)$ in terms of smaller subproblems ?

Knapsack (with repetitions)

Subproblems ?

- ⊗ smaller knapsack capacities: $w \leq W$
- ⊗ fewer items: $1, 2, \dots, j, j \leq n$

Define

$K(w)$ = maximum value achievable with a knapsack of capacity w

How to express $K(w)$ in terms of smaller subproblems ?

If the optimal solution to $K(w)$ includes item i , then removing it from knapsack should yield an optimal solution to $K(w - w_i)$.

Knapsack (with repetitions)

Subproblems ?

- ⦿ smaller knapsack capacities: $w \leq W$
- ⦿ fewer items: $1, 2, \dots, j, j \leq n$

Define

$K(w)$ = maximum value achievable with a knapsack of capacity w

How to express $K(w)$ in terms of smaller subproblems ?

If the optimal solution to $K(w)$ includes item i , then removing it from knapsack should yield an optimal solution to $K(w - w_i)$.

Trying all possibilities for i we get:

$$K(w) = \max_{i:w_i \leq w} \{K(w - w_i) + v_i\}$$

Knapsack (with repetitions)

KNAPSACK(n, w, v, W)

$K(0) := 0$

for $w := 1$ to W do

$K(w) := \max_{i:w_i \leq w} \{K(w - w_i) + v_i\}$

return $K(W)$

Knapsack (with repetitions)

KNAPSACK(n, w, v, W)

$K(0) := 0$

for $w := 1$ to W do

$K(w) := \max_{i:w_i \leq w} \{K(w - w_i) + v_i\}$

return $K(W)$

Running time: $O(nW)$, i.e. pseudopolynomial in n , efficient only for small values of W .

Knapsack (with repetitions)

KNAPSACK(n, w, v, W)

$K(0) := 0$

for $w := 1$ to W do

$K(w) := \max_{i:w_i \leq w} \{K(w - w_i) + v_i\}$

return $K(W)$

Running time: $O(nW)$, i.e. pseudopolynomial in n , efficient only for small values of W .

Example:

weight(w)		1	2	3	4	5	6	7	8	9	10
(j)	0	0	9	14	18	23	30	32	39	44	48

Knapsack (with repetitions)

Underlying dag:

vertices: integers from 0 to W

edges: $(w, w') \in E$ if $w' = w + w_i$, for some $i \leq n$

Knapsack (with repetitions)

Underlying dag:

vertices: integers from 0 to W

edges: $(w, w') \in E$ if $w' = w + w_i$, for some $i \leq n$

Optimal Solution: the shortest path in the dag.

Knapsack (with repetitions)

Underlying dag:

vertices: integers from 0 to W

edges: $(w, w') \in E$ if $w' = w + w_i$, for some $i \leq n$

Optimal Solution: the shortest path in the dag.

0-1 Knapsack

The subproblems need to carry information about items used.

0-1 Knapsack

The subproblems need to carry information about items used.

$K(j, w)$ = maximal value achievable with knapsack of capacity w and items $1, 2, \dots, j$. We need $K(n, W)$

0-1 Knapsack

The subproblems need to carry information about items used.

$K(j, w)$ = maximal value achievable with knapsack of capacity w and items $1, 2, \dots, j$. We need $K(n, W)$

$K(j, w)$ in terms of smaller subproblems:

0-1 Knapsack

The subproblems need to carry information about items used.

$K(j, w)$ = maximal value achievable with knapsack of capacity w and items $1, 2, \dots, j$. We need $K(n, W)$

$K(j, w)$ in terms of smaller subproblems:

$$K(j, w) = \max\{K(j - 1, w - w_j) + v_j, K(j - 1, w)\}$$

0-1 Knapsack

The subproblems need to carry information about items used.

$K(j, w)$ = maximal value achievable with knapsack of capacity w and items $1, 2, \dots, j$. We need $K(n, W)$

$K(j, w)$ in terms of smaller subproblems:

$$K(j, w) = \max\{K(j - 1, w - w_j) + v_j, K(j - 1, w)\}$$

0-1KNAPSACK(n, w, v, W)

for $j := 0$ to n do $K(j, 0) := 0$

for $w := 0$ to W do $K(0, w) := 0$

for $j := 1$ to n do

 for $w := 1$ to W do

 if $w_j > w$ then $K(j, w) := K(j - 1, w)$

 else $K(j, w) := \max\{K(j - 1, w - w_j) + v_j, K(j - 1, w)\}$

return $K(n, W)$

0-1 Knapsack

Example: $W = 10$,

weight(w)		1	2	3	4	5	6	7	8	9	10
(j)	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Knapsack and scheduling problem

A scheduling problem: we have a single machine that can process jobs and we have a set of n requests: $\{1, 2, \dots, n\}$. We are only able to use this resource in the time interval from 0 to W , for some number W . Each request corresponds to a job that requires time t_i to process. If our goal is to process jobs so as to keep the machine as busy as possible up to the "cut-off" W , which jobs should we include ?