ECS 122A: Algorithm Design and Analysis

Week 8 Discussion

Ji Wang

Fall 2020

Logistics

- Quiz 4 will be up this week.
- Regrades must be submitted on Gradescope within two lectures of the day the work was first returned to the whole class.
- No discussion next week, but TAs' office hours remain as usual up to and including Wednesday.

Outline

- Divide and Conquer VS Dynamic Programming
- Variant of LCS: Edit Distance[1]
- Graph Basics: Appendix B and Section 22.1

[1] by courtesy of Professor Z. Bai

Divide and Conquer VS Dynamic Programming

Dynamic Programming

- Subproblems usually **overlap**
- Use a lookup table and backtrace this table (memoization) in a bottom-up (iteration) manner

Divide and Conquer

- Subproblems are **disjoint**, mostly smaller instances of the **same** type
- Solve the subproblems recursively in a top-down (recursion) fashion

Divide and Conquer VS Dynamic Programming

Dynamic Programming

- Subproblems usually **overlap**
- Use a lookup table and backtrace this table (memoization) in a bottom-up (iteration) manner

Divide and Conquer

- Subproblems are **disjoint**, mostly smaller instances of the **same** type
- Solve the subproblems recursively in a top-down (recursion) fashion

A "Recipe" for Dynamic Programming

1. Characterize the structure of an optimal solution, and recursively define the value of an optimal solution. In other word, come up with a **formula**
2. Compute the value of an optimal solution in a **bottom-up** fashion, and make use of the computed information (**momoization**)

Edit distance

- An **alignment**, or matched up, of two strings is simply a way of writing the strings one above the other.

Edit distance

- An **alignment**, or matched up, of two strings is simply a way of writing the strings one above the other.

  example: alignments of "SNOWY" and "SUNNY":

s	-	n	o	w	y		-	s	n	o	w	-	y
s	u	n	n	-	y		s	u	n	-	-	n	y

  "-" indicates a "gap"

Edit distance

- An **alignment**, or matched up, of two strings is simply a way of writing the strings one above the other.

  example: alignments of "`SNOWY`*" and "*`SUNNY`*":*

  ```
  s - n o w y          - s n o w - y
  s u n n - y          s u n - - n y
  ```

  "–" indicates a "gap"

- The **cost** of an alignment is the number of columns in which the letters differ.

Edit distance

- An **alignment**, or matched up, of two strings is simply a way of writing the strings one above the other.

  example: alignments of "`SNOWY`*" and "*`SUNNY`*":*

  ```
  s - n o w y          - s n o w - y
  s u n n - y          s u n - - n y
  ```

  "–" indicates a "gap"

- The **cost** of an alignment is the number of columns in which the letters differ.

  example: alignments of "`SNOWY`*" and "*`SUNNY`*":*

  ```
  s - n o w y          - s n o w - y
  s u n n - y          s u n - - n y
   cost = 3               cost = 5
  ```

Edit distance

- An **alignment**, or matched up, of two strings is simply a way of writing the strings one above the other.

  example: alignments of "`SNOWY`*" and "*`SUNNY`*":*

  ```
  s - n o w y          - s n o w - y
  s u n n - y          s u n - - n y
  ```

  "–" indicates a "gap"

- The **cost** of an alignment is the number of columns in which the letters differ.

  example: alignments of "`SNOWY`*" and "*`SUNNY`*":*

  ```
  s - n o w y          - s n o w - y
  s u n n - y          s u n - - n y
   cost = 3              cost = 5
  ```

- **Edit distance** between two strings is the **minimum cost** of their alignment, i.e., *the best possible alignment*

Edit distance

- An **alignment**, or matched up, of two strings is simply a way of writing the strings one above the other.

  example: alignments of "`SNOWY`*" and "*`SUNNY`*":*

  ```
  s - n o w y        - s n o w - y
  s u n n - y        s u n - - n y
  ```

  "–" indicates a "gap"

- The **cost** of an alignment is the number of columns in which the letters differ.

  example: alignments of "`SNOWY`*" and "*`SUNNY`*":*

  ```
  s - n o w y        - s n o w - y
  s u n n - y        s u n - - n y
   cost = 3            cost = 5
  ```

- **Edit distance** between two strings is the **minimum cost** of their alignment, i.e., *the best possible alignment*
- Edit distance is the minimum number of *edits* – insertions, deletions and substitutions of characters – need to transform the first string into the second.

Edit distance

- An **alignment**, or matched up, of two strings is simply a way of writing the strings one above the other.

  example: alignments of "`SNOWY`*" and "*`SUNNY`*":*

  ```
  s - n o w y        - s n o w - y
  s u n n - y        s u n - - n y
  ```

  "–" indicates a "gap"

- The **cost** of an alignment is the number of columns in which the letters differ.

  example: alignments of "`SNOWY`*" and "*`SUNNY`*":*

  ```
  s - n o w y        - s n o w - y
  s u n n - y        s u n - - n y
   cost = 3            cost = 5
  ```

- **Edit distance** between two strings is the **minimum cost** of their alignment, i.e., *the best possible alignment*
- Edit distance is the minimum number of *edits* – insertions, deletions and substitutions of characters – need to transform the first string into the second. *e.g. a spell checker.*

Edit distance

- Given strings $x[1 \cdots m]$ and $y[1 \cdots n]$.

Edit distance

- Given strings $x[1 \cdots m]$ and $y[1 \cdots n]$. Define

$$e(m, n) = \text{the edit distance between } x \text{ and } y$$

Edit distance

- Given strings $x[1 \cdots m]$ and $y[1 \cdots n]$. Define

$$e(m, n) = \text{the edit distance between } x \text{ and } y$$

  Our objective is to compute $e(m, n)$ efficiently

Edit distance

- Given strings $x[1 \cdots m]$ and $y[1 \cdots n]$. Define

$$e(m, n) = \text{the edit distance between } x \text{ and } y$$

  Our objective is to compute $e(m, n)$ efficiently
- *Subproblem*:

Edit distance

- Given strings $x[1 \cdots m]$ and $y[1 \cdots n]$. Define

$$e(m, n) = \text{the edit distance between } x \text{ and } y$$

  Our objective is to compute $e(m, n)$ efficiently

- *Subproblem*:

$$\text{edit distance } e(i, j) \text{ between } x[1 \cdots i] \text{ and } y[1 \cdots j]$$

Edit distance

- Given strings $x[1 \cdots m]$ and $y[1 \cdots n]$. Define

$$e(m, n) = \text{the edit distance between } x \text{ and } y$$

  Our objective is to compute $e(m, n)$ efficiently
- *Subproblem*:

$$\text{edit distance } e(i, j) \text{ between } x[1 \cdots i] \text{ and } y[1 \cdots j]$$

- How to express $e(i, j)$ in terms of its subproblems, *recursively*?

Edit distance

- Given strings $x[1 \cdots m]$ and $y[1 \cdots n]$. Define

$$e(m, n) = \text{the edit distance between } x \text{ and } y$$

  Our objective is to compute $e(m, n)$ efficiently
- *Subproblem*:

$$\text{edit distance } e(i, j) \text{ between } x[1 \cdots i] \text{ and } y[1 \cdots j]$$

- How to express $e(i, j)$ in terms of its subproblems, *recursively*?
- **key observation:** the rightmost column of an alignment of $x[1 \cdots i]$ and $y[1 \cdots j]$ can only be one of the following three cases:

Case 1		Case 2		Case 3
$x[i]$	or	$-$	or	$x[i]$
$-$		$y[j]$		$y[j]$

Edit distance

- By the above key observation, then

$$e(i,j) = \min\{\underbrace{1 + e(i-1,j)}_{\text{case 1}}, \underbrace{1 + e(i,j-1)}_{\text{case 2}}, \underbrace{\mathrm{diff}(i,j) + e(i-1,j-1)}_{\text{case 3}}\}$$

where

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

- Question: how to find the corresponding optimal alignment?

Edit distance

- The answers to all the subproblems $e(i, j)$ form a two-dimensional table, and the final answer (our objective) is at $e(m, n)$.

Edit distance

- The answers to all the subproblems $e(i, j)$ form a two-dimensional table, and the final answer (our objective) is at $e(m, n)$.
- Initialization:

$$
\begin{aligned}
e(0,0) &= 0; \\
e(i,0) &= i \text{ for } i = 1, \ldots, m \\
e(0,j) &= j \text{ for } j = 1, \ldots, n
\end{aligned}
$$

Edit distance

- The answers to all the subproblems $e(i, j)$ form a two-dimensional table, and the final answer (our objective) is at $e(m, n)$.
- Initialization:

$$e(0,0) = 0;$$
$$e(i,0) = i \text{ for } i = 1, \ldots, m$$
$$e(0,j) = j \text{ for } j = 1, \ldots, n$$

- Pseudocode

Edit distance

- The answers to all the subproblems $e(i, j)$ form a two-dimensional table, and the final answer (our objective) is at $e(m, n)$.
- Initialization:

$$e(0,0) = 0;$$
$$e(i,0) = i \text{ for } i = 1, \dots, m$$
$$e(0,j) = j \text{ for } j = 1, \dots, n$$

- Pseudocode
- Example 1. $x =$ 'snowy', $y =$ 'sunny'

Edit distance

- The answers to all the subproblems $e(i, j)$ form a two-dimensional table, and the final answer (our objective) is at $e(m, n)$.
- Initialization:

$$e(0,0) = 0;$$
$$e(i,0) = i \text{ for } i = 1, \ldots, m$$
$$e(0,j) = j \text{ for } j = 1, \ldots, n$$

- Pseudocode
- Example 1. x = 'snowy', y = 'sunny'

		s	u	n	n	y
	0	1	2	3	4	5
s	1	0	1	2	3	4
n	2	1	1	1	2	3
o	3	2	2	2	2	3
w	4	3	3	3	3	3
y	5	4	4	4	4	3

Edit distance

- The answers to all the subproblems $e(i, j)$ form a two-dimensional table, and the final answer (our objective) is at $e(m, n)$.
- Initialization:

$$\begin{aligned} e(0,0) &= 0; \\ e(i,0) &= i \text{ for } i = 1, \ldots, m \\ e(0,j) &= j \text{ for } j = 1, \ldots, n \end{aligned}$$

- Pseudocode
- Example 1. $x =$ 'snowy', $y =$ 'sunny'

		s	u	n	n	y
	0	1	2	3	4	5
s	1	0	1	2	3	4
n	2	1	1	1	2	3
o	3	2	2	2	2	3
w	4	3	3	3	3	3
y	5	4	4	4	4	3

Therefore, the edit distance between x and $y = e(5, 5) = 3$.

Edit distance

Example 2. $x =$ 'heroically', $y =$ 'scholarly'

Edit distance

Example 2. $x =$ 'heroically', $y =$ 'scholarly'

		s	c	h	o	l	a	r	l	y
	0	1	2	3	4	5	6	7	8	9
h	1	1	2	2	3	4	5	6	7	8
e	2	2	2	3	3	4	5	6	7	8
r	3	3	3	3	4	4	5	5	6	7
o	4	4	4	4	3	4	5	6	6	7
i	5	5	5	5	4	4	5	6	7	7
c	6	6	5	6	5	5	5	6	7	8
a	7	7	6	6	6	6	5	6	7	8
l	8	8	7	7	7	6	6	6	6	7
l	9	9	8	8	8	7	7	7	6	7
y	10	10	9	9	9	8	8	8	7	6

Edit distance

Example 2. $x =$ 'heroically', $y =$ 'scholarly'

		s	c	h	o	l	a	r	l	y
	0	1	2	3	4	5	6	7	8	9
h	1	1	2	2	3	4	5	6	7	8
e	2	2	2	3	3	4	5	6	7	8
r	3	3	3	3	4	4	5	5	6	7
o	4	4	4	4	3	4	5	6	6	7
i	5	5	5	5	4	4	5	6	7	7
c	6	6	5	6	5	5	5	6	7	8
a	7	7	6	6	6	6	5	6	7	8
l	8	8	7	7	7	6	6	6	6	7
l	9	9	8	8	8	7	7	7	6	7
y	10	10	9	9	9	8	8	8	7	6

Therefore, the edit distance between x and $y = e(10, 9) = 6$

Edit distance

Example 2. x = 'heroically', y = 'scholarly'

		s	c	h	o	l	a	r	l	y
	0	1	2	3	4	5	6	7	8	9
h	1	1	2	2	3	4	5	6	7	8
e	2	2	2	3	3	4	5	6	7	8
r	3	3	3	3	4	4	5	5	6	7
o	4	4	4	4	3	4	5	6	6	7
i	5	5	5	5	4	4	5	6	7	7
c	6	6	5	6	5	5	5	6	7	8
a	7	7	6	6	6	6	5	6	7	8
l	8	8	7	7	7	6	6	6	6	7
l	9	9	8	8	8	7	7	7	6	7
y	10	10	9	9	9	8	8	8	7	6

Therefore, the edit distance between x and $y = e(10, 9) = 6$

Note: LCS(x, y) = *5*

Edit Distance: Suboptimality Proof

Given that O is optimal for $e(x_m, y_n)$.

Proof.

Assume that O' is not an optimal solution to $e(x_{m-1}, y_n)$.

$\implies \exists$ an optimal solution A to $e(x_{m-1}, y_n)$ such that $|A| < |O'|$.

$\implies |A| + 1 < |O'| + 1$

$\implies |A| + 1 < |O|$

This contradicts with the fact that O is optimal.

$\implies O'$ is optimal for aligning x_{m-1} and y_n. $\square$

Graph Basics: Notation

Undirected Graph and Directed Graph

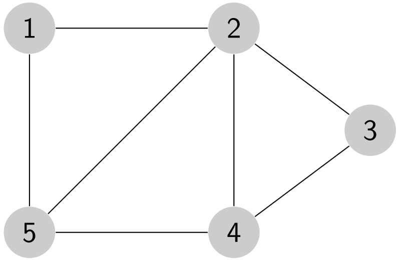


Figure: An undirected graph G_1 with 5 vertices and 7 edges

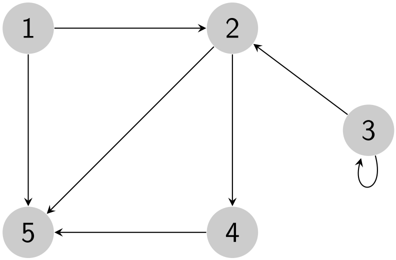


Figure: A directed graph G_2 with 5 vertices and 7 edges.

Self-loops—edges from a vertex to itself—are possible in a directed graph, but are forbidden in an undirected graph.

Graph Basics: Notation

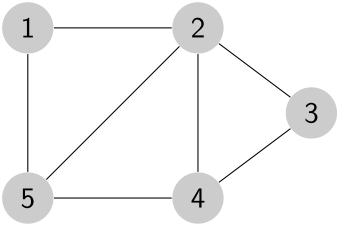


Figure: An undirected graph G_1 with 5 vertices and 7 edges

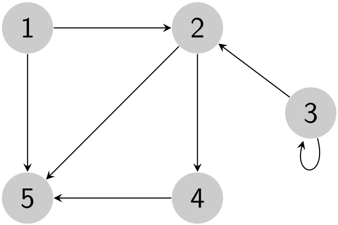


Figure: A directed graph G_2 with 5 vertices and 7 edges.

1. If (u, v) is an edge in an undirected graph G, we say that (u, v) is **incident** on vertices u and v.
2. If (u, v) is an edge in a directed graph G, we say that (u, v) is incident from or leaves vertex u and is incident to or enters vertex v.

Graph Basics: Notation

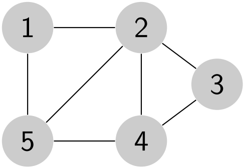


Figure: An undirected graph G_1 with 5 vertices and 7 edges

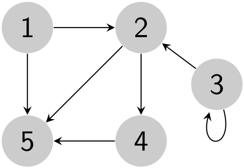


Figure: A directed graph G_2 with 5 vertices and 7 edges.

1. The **degree** of a vertex in an undirected graph is the number of edges incident on it. $\sum_{u \in V} \text{degree}(u) = 2|E|$.
2. In a directed graph, the **out-degree** of a vertex is the number of edges leaving it, and the **in-degree** of a vertex is the number of edges entering it. The degree of a vertex in a directed graph is its in-degree plus its out-degree. The whole graph has $\sum_{u \in V} \text{out-degree}(u) = \sum_{u \in V} \text{in-degree}(u) = |E|$.
3. A vertex whose degree is 0 is **isolated**.

Graph Basics: Notation

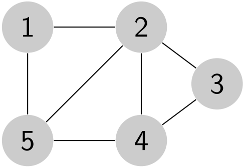


Figure: An undirected graph G_1 with 5 vertices and 7 edges

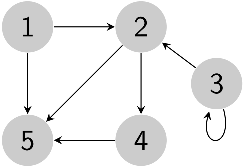


Figure: A directed graph G_2 with 5 vertices and 7 edges.

1. A **path** of length k from a vertex u to a vertex u' in a graph G is a sequence $\langle v_0, v_1, \cdots, v_k \rangle$ of vertices such that $u = v_0, u' = v_k$, and $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \cdots, k-1$.
2. If there is a path p from u to u', we say that u' is **reachable** from u via p.
3. In a directed graph, a path $\langle v_0, v_1, \cdots, v_k \rangle$ forms a **cycle** if $v_0 = v_k$ and the path contains at least one edge. Any graph with no cycles is acyclic. If the graph is directed, we call it dag ("directed acyclic graph").

Graph Basics: Representation

Adjacency List and Adjacency Matrix

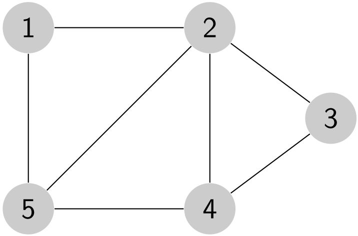


Figure: Graph G_1

Graph Basics: Representation

Adjacency List and Adjacency Matrix

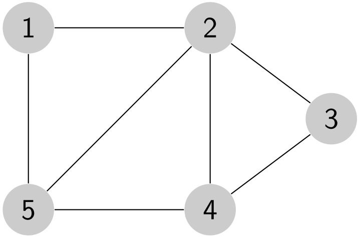
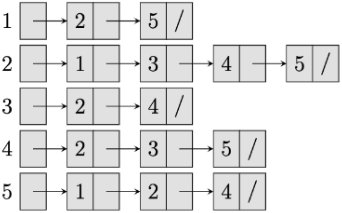


Figure: Graph G_1



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Figure: Adjacency list and adjacency matrix representation of G_1

Graph Basics: Representation

How about G_2? It's your turn now!

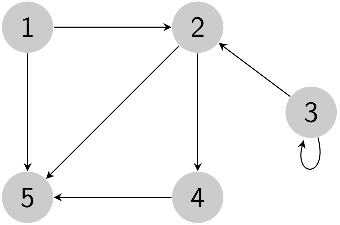


Figure: Graph G_2

Graph Basics: Representation

How about G_2? It's your turn now!

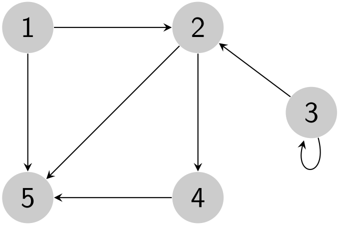


Figure: Graph G_2

1. Adjacency list is of size $\Theta(|V| + |E|)$ while adjacency matrix needs $|V| \times |V|$ space.
2. If G is undirected, its adjacency matrix A is **symmetric**. Namely, $A^T = A$. Further, the main diagonal entries of A are all zeros.