Lecture 9: Dynamic Programming III

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

October 5, 2023

Overview

- Edit Distance
- Graphs & DP on Trees
- Acknowledgements

Edit Distance

- **Input:** two strings $A := a_1 a_2 \cdots a_m$ and $B := b_1 b_2 \cdots b_n$, where $a_i, b_j \in \Sigma$
- **Output:** minimum number of edits to string A (*add, delete, change*) to transform it into string B
- **Model:** word RAM
- Example:

SNOWY and SUNNY

- approach 1 (3 edits)

S – N O W Y
S U N N – Y

- approach 2 (4 edits)

– S N O W Y
S U N N – Y

Edit Distance

- **Input:** two strings $A := a_1 a_2 \cdots a_m$ and $B := b_1 b_2 \cdots b_n$, where $a_i, b_j \in \Sigma$
- **Output:** minimum number of edits to string A (*add, delete, change*) to transform it into string B
- Looks a bit hard. Can we DP it?

Edit Distance

- **Input:** two strings $A := a_1 a_2 \cdots a_m$ and $B := b_1 b_2 \cdots b_n$, where $a_i, b_j \in \Sigma$
- **Output:** minimum number of edits to string A (*add, delete, change*) to transform it into string B

- Subproblems: let $A_i := a_1 \cdots a_i$ and $B_j := b_1 \cdots b_j$, and let $D(i, j)$ be edit distance between A_i, B_j. Base case: $D(0, 0) = 0$.

Edit Distance

- **Input:** two strings $A := a_1 a_2 \cdots a_m$ and $B := b_1 b_2 \cdots b_n$, where $a_i, b_j \in \Sigma$
- **Output:** minimum number of edits to string A (*add, delete, change*) to transform it into string B

- Subproblems: let $A_i := a_1 \cdots a_i$ and $B_j := b_1 \cdots b_j$, and let $D(i,j)$ be edit distance between A_i, B_j. Base case: $D(0,0) = 0$.
- Cases (based on allowed operations)
  1. **Add:** add b_j to string A_i.
     Total cost: $Sol_1 := 1 + D(i, j-1)$
  2. **Delete:** delete a_i from A_i.
     Total cost $Sol_2 := 1 + D(i-1, j)$
  3. **Change/Match:** can change $a_i \mapsto b_j$. (if $a_i = b_j$ we simply match them)

$$\text{Total cost: } Sol_3 := \begin{cases} 1 + D(i-1, j-1), & \text{if } a_i \neq b_j \\ D(i-1, j-1), & \text{if } a_i = b_j \end{cases}$$

Edit Distance - Recurrence

- Thus, recurrence given by

$$D(i,j) = \min\{Sol_1, Sol_2, Sol_3\}$$

Edit Distance - Recurrence

- Thus, recurrence given by

$$D(i,j) = \min\{Sol_1, Sol_2, Sol_3\}$$

- **Correctness:** proof by induction.
  1. True for base case, i.e. $D(0,0) = 0$.
  2. If all subcases are correct, then recurrence tells us all possible ways to handle the last symbols of the strings (thus one must lead to the optimum distance).

Edit Distance - Recurrence

- Thus, recurrence given by

$$D(i,j) = \min\{Sol_1, Sol_2, Sol_3\}$$

- **Correctness:** proof by induction.
  1. True for base case, i.e. $D(0,0) = 0$.
  2. If all subcases are correct, then recurrence tells us all possible ways to handle the last symbols of the strings (thus one must lead to the optimum distance).
- **Runtime:**
  1. # Subproblems: $O(mn)$
  2. Time per subproblem (given previous subproblems computed): $O(1)$
  3. **Runtime:** $O(mn)$

Edit Distance - Recurrence

- Thus, recurrence given by

$$D(i,j) = \min\{Sol_1, Sol_2, Sol_3\}$$

- **Correctness:** proof by induction.
  1. True for base case, i.e. $D(0,0) = 0$.
  2. If all subcases are correct, then recurrence tells us all possible ways to handle the last symbols of the strings (thus one must lead to the optimum distance).
- **Runtime:**
  1. # Subproblems: $O(mn)$
  2. Time per subproblem (given previous subproblems computed): $O(1)$
  3. **Runtime:** $O(mn)$
- Computing the table (*bottom up*): want to go from $(0,0)$ to (m,n). Can compute in increasing row order, from left to right.

- Edit Distance
- Graphs & DP on Trees
- Acknowledgements

Wait graphs already?!

Graphs - Definition

Trees

Maximum Independent Set on Trees

- **Input:** A tree $T([n], E)$
- **Output:** A maximum independent set in T

Maximum Independent Set on Trees

- **Input:** A tree $T([n], E)$
- **Output:** A maximum independent set in T
- Idea: pick vertex to be a root. Traverse the tree downwards as follows:
  1. Given vertex v, if we include it in our independent set, then don't include its children (look at the grandchildren)
  2. If don't include v, then pick *all* its children

Maximum Independent Set on Trees

- **Input:** A tree $T([n], E)$
- **Output:** A maximum independent set in T
- Idea: pick vertex to be a root. Traverse the tree downwards as follows:
  1. Given vertex v, if we include it in our independent set, then don't include its children (look at the grandchildren)
  2. If don't include v, then pick *all* its children
- Recurrence:

$$MIS(v) = \max \left\{ 1 + \sum_{w \text{ grandchild of } v} MIS(w), \sum_{u \text{ child of } v} MIS(u) \right\}$$

Maximum Independent Set on Trees

- **Input:** A tree $T([n], E)$
- **Output:** A maximum independent set in T
- Idea: pick vertex to be a root. Traverse the tree downwards as follows:
  1. Given vertex v, if we include it in our independent set, then don't include its children (look at the grandchildren)
  2. If don't include v, then pick *all* its children
- Recurrence:

$$MIS(v) = \max \left\{ 1 + \sum_{w \text{ grandchild of } v} MIS(w), \sum_{u \text{ child of } v} MIS(u) \right\}$$

- **Running time:**
  1. $\#$ subproblems: $O(n)$ ($\#$ vertices)
  2. time per subproblem (once we have subproblems): $O(|E|) = O(n)$
  3. Total runtime: $O(n^2)$

Acknowledgement

- Based on [DPV 2006]

References I

Cormen, Thomas and Leiserson, Charles and Rivest, Ronald and Stein, Clifford. (2009)

Introduction to Algorithms, third edition.

MIT Press

Dasgupta, Sanjay and Papadimitriou, Christos and Vazirani, Umesh (2006)

Algorithms

Kleinberg, Jon and Tardos, Eva (2006)

Algorithm Design.

Addison Wesley