Lecture 12: Introduction to Dynamic Programming

BT 3051 – Data Structures and Algorithms for Biology

Karthik Raman
Department of Biotechnology
Bhupat and Jyoti Mehta School of Biosciences
Indian Institute of Technology Madras

The Coin-Change Problem

The Coin-Change Problem

- To pay out a certain amount of money x, in a particular currency, what is the least number of coins necessary?

Mathematical Statement

- Given an amount of money M
- Denominations $c = c_1, c_2, \ldots, c_d$, in decreasing order
- Compute integers $i_1, i_2, \ldots, i_d$ such that

The Coin-Change Problem

- To pay out a certain amount of money x, in a particular currency, what is the least number of coins necessary?

Mathematical Statement

- Given an amount of money M
- Denominations $c = c_1, c_2, \ldots, c_d$, in decreasing order
- Compute integers $i_1, i_2, \ldots, i_d$ such that
  - $\sum_{k=1}^{d} c_k i_k = M$, and
  - $\sum_{k=1}^{d} i_k$ is minimized

The Coin-Change Problem

- To pay out a certain amount of money x, in a particular currency, what is the least number of coins necessary?

Mathematical Statement

- Given an amount of money M
- Denominations $c = c_1, c_2, \ldots, c_d$, in decreasing order
- Compute integers $i_1, i_2, \ldots, i_d$ such that
  - $\sum_{k=1}^{d} c_k i_k = M$, and
  - $\sum_{k=1}^{d} i_k$ is minimized

The Coin-Change Problem

- To pay out a certain amount of money x, in a particular currency, what is the least number of coins necessary?

Mathematical Statement

- Given an amount of money M
- Denominations $c = c_1, c_2, \ldots, c_d$, in decreasing order
- Compute integers $i_1, i_2, \ldots, i_d$ such that
  - $\sum_{k=1}^{d} c_k i_k = M$, and
  - $\sum_{k=1}^{d} i_k$ is minimised

The Coin-Change Problem

- To pay out a certain amount of money x, in a particular currency, what is the least number of coins necessary?

Mathematical Statement

- Given an amount of money M
- Denominations $c = c_1, c_2, \ldots, c_d$, in decreasing order
- Compute integers $i_1, i_2, \ldots, i_d$ such that
  - $\sum_{k=1}^{d} c_k i_k = M$, and
  - $\sum_{k=1}^{d} i_k$ is minimised

The Coin-Change Problem

- To pay out a certain amount of money x, in a particular currency, what is the least number of coins necessary?

Mathematical Statement

- Given an amount of money M
- Denominations $c = c_1, c_2, \ldots, c_d$, in decreasing order
- Compute integers $i_1, i_2, \ldots, i_d$ such that
  - $\sum_{k=1}^{d} c_k i_k = M$, and
  - $\sum_{k=1}^{d} i_k$ is minimised

The Coin-Change Problem

- A *greedy algorithm* may work!
  - But fails for certain denominations ...(e.g. ₹8 using ₹5, ₹4, ₹1)
- Let's try to break the problem down:
  - To pay out ₹9 (find $C(9)$), we will pick the minimum of
    - 1 [₹5] + $C(4)$, the best way to pay out ₹4
    - 1 [₹2] + $C(7)$, and
    - 1 [₹1] + $C(8)$

Recurrence Relation

$$minNumCoins(M) = \min \begin{cases} minNumCoins(M - c_1) + 1 \\ minNumCoins(M - c_2) + 1 \\ \dots \\ minNumCoins(M - c_d) + 1 \end{cases}$$

The Coin-Change Problem

- A *greedy algorithm* may work!
  - But fails for certain denominations ...(e.g. ₹8 using ₹5, ₹4, ₹1)
- Let's try to break the problem down:
  - To pay out ₹9 (find $C(9)$), we will pick the minimum of
    - 1 [₹5] + $C(4)$, the best way to pay out ₹4
    - 1 [₹2] + $C(7)$, and
    - 1 [₹1] + $C(8)$

Recurrence Relation

$$minNumCoins(M) = \min \begin{cases} minNumCoins(M - c_1) + 1 \\ minNumCoins(M - c_2) + 1 \\ \dots \\ minNumCoins(M - c_d) + 1 \end{cases}$$

The Coin-Change Problem

- A *greedy algorithm* may work!
  - But fails for certain denominations ...(e.g. ₹8 using ₹5, ₹4, ₹1)
- Let's try to break the problem down:
  - To pay out ₹9 (find $C(9)$), we will pick the minimum of
    - 1 [₹5] + $C(4)$, the best way to pay out ₹4
    - 1 [₹2] + $C(7)$, and
    - 1 [₹1] + $C(8)$

Recurrence Relation

$$minNumCoins(M) = \min \begin{cases} minNumCoins(M - c_1) + 1 \\ minNumCoins(M - c_2) + 1 \\ \dots \\ minNumCoins(M - c_d) + 1 \end{cases}$$

The Coin-Change Problem

- A *greedy algorithm* may work!
  - But fails for certain denominations ...(e.g. ₹8 using ₹5, ₹4, ₹1)
- Let's try to break the problem down:
  - To pay out ₹9 (find $C(9)$), we will pick the minimum of
    - 1 [₹5] + $C(4)$, the best way to pay out ₹4
    - 1 [₹2] + $C(7)$, and
    - 1 [₹1] + $C(8)$

Recurrence Relation

$$minNumCoins(M) = \min \begin{cases} minNumCoins(M - c_1) + 1 \\ minNumCoins(M - c_2) + 1 \\ \dots \\ minNumCoins(M - c_d) + 1 \end{cases}$$

The Problem Tree

Edit Distance

Edit Distance/Levenshtein Distance

http://www.inf.usi.ch/carzaniga/edu/algo13s/index.html

- Given two strings s and t, find the smallest set of edit operations that transform s into t
  - edit operations: delete, insert, and modify a single character
- Has important applications

- D(Levenshtein, Levenstein) = 1 (1 deletion)
- D(money, monkey) = 1 (1 insertion)
- D(Shock, Spock) = 1 (1 substitution)

Edit Distance/Levenshtein Distance

http://www.inf.usi.ch/carzaniga/edu/algo13s/index.html

- Given two strings s and t, find the smallest set of edit operations that transform s into t
  - edit operations: delete, insert, and modify a single character
- Has important applications

- D(Levenshtein, Levenstein) = 1 (1 deletion)
- D(money, monkey) = 1 (1 insertion)
- D(Shock, Spock) = 1 (1 substitution)

Edit Distance/Levenshtein Distance

http://www.inf.usi.ch/carzaniga/edu/algo13s/index.html

- Given two strings s and t, find the smallest set of edit operations that transform s into t
  - edit operations: delete, insert, and modify a single character
- Has important applications

- D(Levenshtein, Levenstein) = 1 (1 deletion)
- D(money, monkey) = 1 (1 insertion)
- D(Shock, Spock) = 1 (1 substitution)

Edit Distance/Levenshtein Distance

http://www.inf.usi.ch/carzaniga/edu/algo13s/index.html

- Given two strings s and t, find the smallest set of edit operations that transform s into t
  - edit operations: delete, insert, and modify a single character
- Has important applications
  - Spell checking
  - Speech recognition
  - DNA analysis
  - Plagiarism detection

- D(Levenshtein, Levenstein) = 1 (1 deletion)
- D(money, monkey) = 1 (1 insertion)
- D(Shock, Spock) = 1 (1 substitution)

Edit Distance/Levenshtein Distance

http://www.inf.usi.ch/carzaniga/edu/algo13s/index.html

- Given two strings s and t, find the smallest set of edit operations that transform s into t
  - edit operations: delete, insert, and modify a single character
- Has important applications
  - Spell checking
  - Speech recognition
  - DNA analysis
  - Plagiarism detection

Example:

- D(Levenshtein, Levenstein) = 1 (1 deletion)
- D(money, monkey) = 1 (1 insertion)
- D(Shock, Spock) = 1 (1 substitution)

Edit Distance/Levenshtein Distance

http://www.inf.usi.ch/carzaniga/edu/algo13s/index.html

- Given two strings s and t, find the smallest set of edit operations that transform s into t
  - edit operations: delete, insert, and modify a single character
- Has important applications
  - Spell checking
  - Speech recognition
  - DNA analysis
  - Plagiarism detection

Example:

- D(Levenshtein, Levenstein) = 1 (1 deletion)
- D(money, monkey) = 1 (1 insertion)
- D(Shock, Spock) = 1 (1 substitution)

Edit Distance/Levenshtein Distance

http://www.inf.usi.ch/carzaniga/edu/algo13s/index.html

- Given two strings s and t, find the smallest set of edit operations that transform s into t
  - edit operations: delete, insert, and modify a single character
- Has important applications
  - Spell checking
  - Speech recognition
  - DNA analysis
  - Plagiarism detection

Example:

- D(Levenshtein, Levenstein) = 1 (1 deletion)
- D(money, monkey) = 1 (1 insertion)
- D(Shock, Spock) = 1 (1 substitution)

Edit Distance/Levenshtein Distance

http://www.inf.usi.ch/carzaniga/edu/algo13s/index.html

- Given two strings s and t, find the smallest set of edit operations that transform s into t
  - edit operations: delete, insert, and modify a single character
- Has important applications
  - Spell checking
  - Speech recognition
  - DNA analysis
  - Plagiarism detection

Example:

- D(Levenshtein, Levenstein) = 1 (1 deletion)
- D(money, monkey) = 1 (1 insertion)
- D(Shock, Spock) = 1 (1 substitution)

Edit Distance

Algorithm

- Align the two strings s and t, possibly inserting "gaps" between letters
  - gap in the source $\Rightarrow$ *insertion*
  - gap in the destination $\Rightarrow$ *deletion*
  - two different characters in the same position $\Rightarrow$ *modification*

Edit Distance

Algorithm

- Align the two strings s and t, possibly inserting "gaps" between letters
  - gap in the source $\Rightarrow$ *insertion*
  - gap in the destination $\Rightarrow$ *deletion*
  - two different characters in the same position $\Rightarrow$ *modification*

Edit Distance

Algorithm

- Align the two strings s and t, possibly inserting "gaps" between letters
  - gap in the source $\Rightarrow$ *insertion*
  - gap in the destination $\Rightarrow$ *deletion*
  - two different characters in the same position $\Rightarrow$ *modification*

Edit Distance

Algorithm

- Align the two strings s and t, possibly inserting "gaps" between letters
  - gap in the source $\Rightarrow$ *insertion*
  - gap in the destination $\Rightarrow$ *deletion*
  - two different characters in the same position $\Rightarrow$ *modification*

Edit Distance

Algorithm

- Align the two strings s and t, possibly inserting "gaps" between letters
  - gap in the source $\Rightarrow$ *insertion*
  - gap in the destination $\Rightarrow$ *deletion*
  - two different characters in the same position $\Rightarrow$ *modification*

Edit Distance

Algorithm

- Align the two strings s and t, possibly inserting "gaps" between letters
  - gap in the source $\Rightarrow$ *insertion*
  - gap in the destination $\Rightarrow$ *deletion*
  - two different characters in the same position $\Rightarrow$ *modification*

Obviously, several alignments are possible: the alignment with the least number of insertions, deletions, and modifications defines the *edit distance*

Edit Distance

How to solve?

- Think of the coin-change problem again ...
- How to decompose into sub-problems?
- Let $E(i, j)$ be the smallest set of changes that turn the first i characters of s into the first j characters of t
- Now, the last column of the alignment of $E(i, j)$ can have either
  - a gap for s (i.e., insertion)
  - a gap for t (i.e., deletion)
  - no gaps (i.e., modification if $s[i] \neq t[j]$)

Recurrence Relation

$$E(i,j) = \min \begin{cases} E(i-1,j)+1 \\ E(i,j-1)+1 \\ E(i-1,j-1)+\delta(s[i],t[j]) \end{cases}$$

Edit Distance

How to solve?

- Think of the coin-change problem again ...
- How to decompose into sub-problems?
- Let $E(i, j)$ be the smallest set of changes that turn the first i characters of s into the first j characters of t
- Now, the last column of the alignment of $E(i, j)$ can have either
  - a gap for s (i.e., insertion)
  - a gap for t (i.e., deletion)
  - no gaps (i.e., modification if $s[i] \neq t[j]$)

Recurrence Relation

$$E(i,j) = \min \begin{cases} E(i-1,j)+1 \\ E(i,j-1)+1 \\ E(i-1,j-1)+\delta(s[i],t[j]) \end{cases}$$

Edit Distance

How to solve?

- Think of the coin-change problem again ...
- How to decompose into sub-problems?
- Let $E(i, j)$ be the smallest set of changes that turn the first i characters of s into the first j characters of t
- Now, the last column of the alignment of $E(i, j)$ can have either
  - a gap for s (i.e., insertion)
  - a gap for t (i.e., deletion)
  - no gaps (i.e., modification if $s[i] \neq t[j]$)

Recurrence Relation

$$E(i,j) = \min \begin{cases} E(i-1,j)+1 \\ E(i,j-1)+1 \\ E(i-1,j-1)+\delta(s[i],t[j]) \end{cases}$$

Edit Distance

How to solve?

- Think of the coin-change problem again ...
- How to decompose into sub-problems?
- Let $E(i, j)$ be the smallest set of changes that turn the first i characters of s into the first j characters of t
- Now, the last column of the alignment of $E(i, j)$ can have either
  - a gap for s (i.e., insertion)
  - a gap for t (i.e., deletion)
  - no gaps (i.e., modification iff $s[i] \neq t[j]$)

Recurrence Relation

$$E(i,j) = \min \begin{cases} E(i-1,j)+1 \\ E(i,j-1)+1 \\ E(i-1,j-1)+\delta(s[i],t[j]) \end{cases}$$

Edit Distance

How to solve?

- Think of the coin-change problem again ...
- How to decompose into sub-problems?
- Let $E(i, j)$ be the smallest set of changes that turn the first i characters of s into the first j characters of t
- Now, the last column of the alignment of $E(i, j)$ can have either
  - a gap for s (i.e., insertion)
  - a gap for t (i.e., deletion)
  - no gaps (i.e., modification iff $s[i] \neq t[j]$)

Recurrence Relation

$$E(i,j) = \min \begin{cases} E(i-1,j)+1 \\ E(i,j-1)+1 \\ E(i-1,j-1)+\delta(s[i],t[j]) \end{cases}$$

Edit Distance

How to solve?

- Think of the coin-change problem again ...
- How to decompose into sub-problems?
- Let $E(i, j)$ be the smallest set of changes that turn the first i characters of s into the first j characters of t
- Now, the last column of the alignment of $E(i, j)$ can have either
  - a gap for s (i.e., insertion)
  - a gap for t (i.e., deletion)
  - no gaps (i.e., modification iff $s[i] \neq t[j]$)

Recurrence Relation

$$E(i,j) = \min \begin{cases} E(i-1,j)+1 \\ E(i,j-1)+1 \\ E(i-1,j-1)+\delta(s[i],t[j]) \end{cases}$$

Edit Distance

How to solve?

- Think of the coin-change problem again ...
- How to decompose into sub-problems?
- Let $E(i, j)$ be the smallest set of changes that turn the first i characters of s into the first j characters of t
- Now, the last column of the alignment of $E(i, j)$ can have either
  - a gap for s (i.e., insertion)
  - a gap for t (i.e., deletion)
  - no gaps (i.e., modification iff $s[i] \neq t[j]$)

Recurrence Relation

$$E(i,j) = \min \begin{cases} E(i-1,j)+1 \\ E(i,j-1)+1 \\ E(i-1,j-1)+\delta(s[i],t[j]) \end{cases}$$

Edit Distance

How to solve?

- Think of the coin-change problem again ...
- How to decompose into sub-problems?
- Let $E(i, j)$ be the smallest set of changes that turn the first i characters of s into the first j characters of t
- Now, the last column of the alignment of $E(i, j)$ can have either
  - a gap for s (i.e., insertion)
  - a gap for t (i.e., deletion)
  - no gaps (i.e., modification iff $s[i] \neq t[j]$)

Recurrence Relation

$$E(i,j) = \min \begin{cases} E(i-1,j)+1 \\ E(i,j-1)+1 \\ E(i-1,j-1)+\delta(s[i],t[j]) \end{cases}$$

Edit Distance

```
def EditDistRec(s,t):
  delta={True:0,False:1} #substitution cost

  #BASE CASES
  if len(s)==0:
    return len(t) #whole of t is an insertion
  elif len(t)==0:
    return len(s) #whole of s has been deleted

  #RECURSIVE CALL
  return min(
     EditDistRec(s[:-1],t[:-1]) + delta[s[-1]==t[-1]],
     EditDistRec(s     ,t[:-1]) + 1,
     EditDistRec(s[:-1],t     ) + 1)
```