BIOINFORMATICS

How do we compare biological sequences?

Marco Beccuti

Università degli Studi di Torino

Dipartimento di Informatica

March 2019

Outline

1. Introduction to Sequence Alignment
2. Hamming distance for similarity between sequences
3. Alignment Game and the Longest Common Subsequence;
4. The Manhattan Tourist Problem;
5. The Change Problem;
6. Dynamic programming and backtracking pointers;
7. From Manhattan to Alignment Graph;
8. From Global to Local Alignment;
9. Penalizing Insertions and Deletions in Sequence Alignment;
10. Space-Efficient Sequence Alignment;
11. Multiple Sequence Alignment.

Chapter 5 in ***Bioinformatics Algorithms: An active Learning Approach (Vol.1).***

Part 1

Introduction to Sequence Alignment

Introduction to Sequence Alignment

- Alignment of biological sequences is crucial operation in bioinformatics, and genetics research;
- The sequence alignment is required by a great number of applications:
  - *Genetic disease research;*
  - *Construction of phylogenetic trees;*
  - *Comparing functions between similar genes;*
  - *...*
- Its aims is to determine the similarity between different sequences:
  - sequences are aligned to get the *highest number* of matching characters;
  - *gaps* can be inserted into a sequence to shift the remaining characters into better matches;
  - a *scoring function* is used to rank different alignments so that biologically plausible alignments score higher;

Part 1

Hamming distance for similarity between sequences

Hamming distance for similarity between sequences

- It is a well-known metric to measure dissimilarity between two strings;
- It counts the minimum number of substitutions required to change one sequence into the other;
- it always aligns the i-th symbol of one sequence against the i-th symbol of the other;

```
G A G A C T C A T
X         X
T A G A C G C A T
```

What is the Hammming distance between the following sequences?

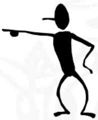
ATGCATGC

TGCATGCC

Hamming distance for similarity between sequences

- It is a well-known metric to measure dissimilarity between two strings;
- It counts the minimum number of substitutions required to change one sequence into the other;
- it always aligns the i-th symbol of one sequence against the i-th symbol of the other;

```
G A G A C T C A T
X         X
T A G A C G C A T
```



A Hamming distance of 2

What is the Hammming distance between the following sequences?

ATGCATGC
XXXXXXX↓
TGCATGCC

Hamming distance is equal to 7.

Hamming distance for similarity between sequences

Is Hamming distance enough to determine the similarity between biological sequences?

Hamming distance for similarity between sequences

Is Hamming distance enough to determine the similarity between biological sequences?

- Since biological sequences are subjected to insertions/deletions, it is often the case that the i-th symbol of one sequence corresponds to a symbol at a different position in the other sequence
- The goal is to find the most appropriate correspondence of symbols.

For instance considering the previous two sequences:

ATGCATGC
XXXXXXX↓
TGCATGCC

Seven matching positions can be found if we align them differently

ATGCATGC−

−TGCATGCC

Alignment Game
and
the Longest Common Subsequence

Alignment Game and the Longest Common Subsequence

- We can define a *good alignment* as one that matches as many symbols as possible;
- We introduce single-person game whose goal is to maximize the number of matched symbols in two strings;

Alignment Game

Possible actions and rewards:

- Remove the 1st symbol from each sequence and receive 1 point if the symbols match otherwise 0 points;
- Remove the 1st symbol from one of the sequences and receive 0 points.

Alignment Game and the Longest Common Subsequence

- An example of the Alignment Game

A T G T T A T A

A T C G T C C

Alignment Game

Possible actions and rewards:

- Remove the 1st symbol from each sequence and receive 1 point if the symbols match otherwise 0 points;
- Remove the 1st symbol from one of the sequences and receive 0 points.

Alignment Game and the Longest Common Subsequence

- An example of the Alignment Game

A T G T T A T A

A T C G T C C

+1

Alignment Game

Possible actions and rewards:

- Remove the 1st symbol from each sequence and receive 1 point if the symbols match otherwise 0 points;
- Remove the 1st symbol from one of the sequences and receive 0 points.

Alignment Game and the Longest Common Subsequence

- An example of the Alignment Game

A T G T T A T A

A T C G T C C

+1+1

Alignment Game

Possible actions and rewards:

- Remove the 1st symbol from each sequence and receive 1 point if the symbols match otherwise 0 points;
- Remove the 1st symbol from one of the sequences and receive 0 points.

Alignment Game and the Longest Common Subsequence

- An example of the Alignment Game

A T – G T T A T A

A T C G T C C

+1+1

Alignment Game

Possible actions and rewards:

- Remove the 1st symbol from each sequence and receive 1 point if the symbols match otherwise 0 points;
- Remove the 1st symbol from one of the sequences and receive 0 points.

Alignment Game and the Longest Common Subsequence

- An example of the Alignment Game

A T – G T T A T A

A T C G T C C

+1+1 +1

Alignment Game

Possible actions and rewards:

- Remove the 1st symbol from each sequence and receive 1 point if the symbols match otherwise 0 points;
- Remove the 1st symbol from one of the sequences and receive 0 points.

Alignment Game and the Longest Common Subsequence

- An example of the Alignment Game

A	**T**	–	**G**	**T**	**T**	**A**	**T**	**A**
A	**T**	**C**	**G**	**T**	–	**C**	–	**C**

+1+1 +1+1 =4

Alignment Game

Possible actions and rewards:

- Remove the 1st symbol from each sequence and receive 1 point if the symbols match otherwise 0 points;
- Remove the 1st symbol from one of the sequences and receive 0 points.

Alignment Game and the Longest Common Subsequence

- several different strategies exists;
- each strategy provides a possible alignment of two sequences where:

An alignment of two sequences v and w is a *two row matrix* such that the first row contains the v symbols (in order) and the second row the w symbols (in order). *Space symbols* (i.e. $-$) can be added in both sequences.

$$\begin{array}{rcccccccc} v: & \mathbf{A} & \mathbf{T} & - & \mathbf{G} & \mathbf{T} & \mathbf{T} & \mathbf{A} & \mathbf{T} & \mathbf{A} \\ w: & \mathbf{A} & \mathbf{T} & \mathbf{C} & \mathbf{G} & \mathbf{T} & - & \mathbf{C} & - & \mathbf{C} \end{array}$$

- Columns containing the same letter are called **matches**;
- Columns containing different letters are called **mismatches**;
- Columns containing space symbols are called **in**del
  - a column containing a space symbol in the first row is called **insertion**
  - a column containing a space symbol in the second row is called **deletion**

Alignment Game and the Longest Common Subsequence

- Matches in the alignment define a *Common Subsequence*;
- An alignment of two sequences maximizing the number of matches corresponds to find the *Longest Common Subsequence*;
- How to efficiently solve the *Longest Common Subsequence problem*:

Longest Common Subsequence problem

Definition: find a longest common subsequence of two strings

Input: two strings

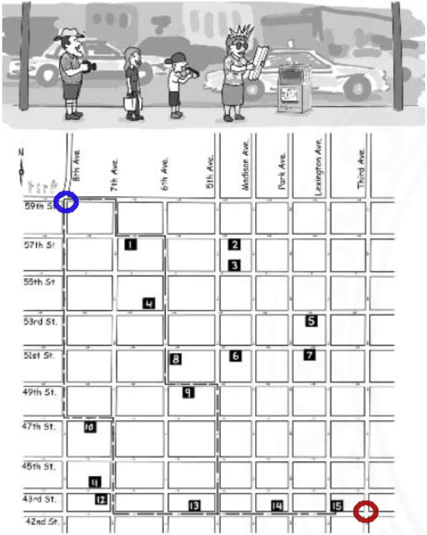
Output: a longest common subsequence of two strings

Part 1

The Manhattan Tourist Problem

The Manhattan Tourist Problem

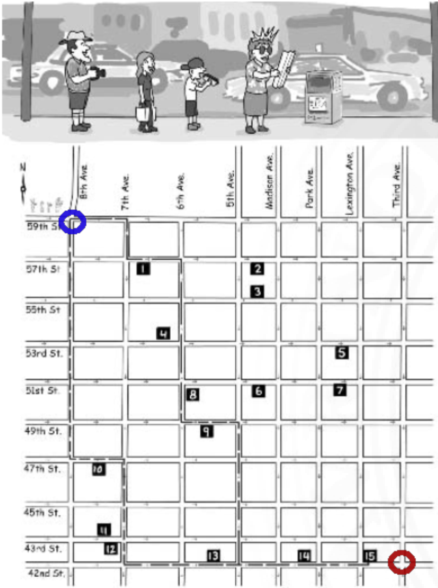
- Longest Common Subsequence problem can be connected to the well-known Manhattan Tourist Problem.



A sightseeing Tour of Manhattan

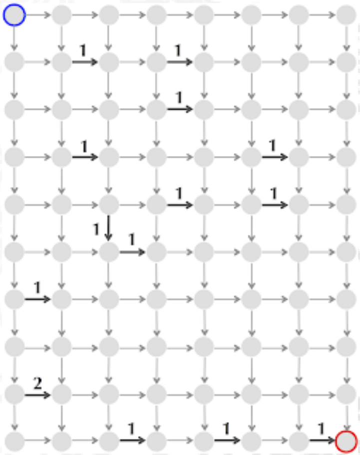
- Walk from the **source** to the **sink**;
- Only South (↓) or East (→) directions are allowed;
- Goal: to visit the maximum number of attractions (black box)

The Manhattan Tourist Problem



A directed graph encoding Manhattan

- Nodes are the street intersection and edges the streets;
- An edge is labeled with the number of attractions in that city block.



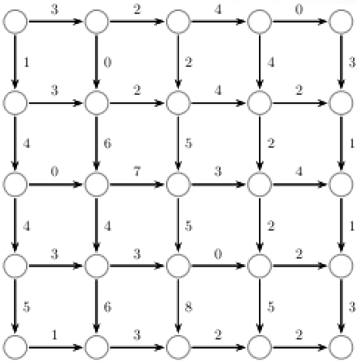
The Manhattan Tourist Problem

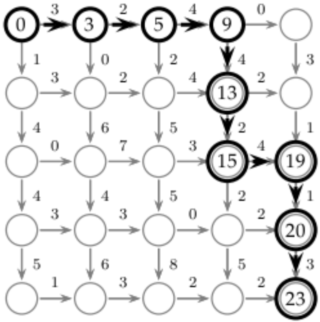
The Manhattan Tourist Problem

Definition: Find a path visiting most attractions (namely ***longest path***) in a rectangular city

Input: A weighted $n \times m$ rectangular grid with $n + 1$ rows and $m + 1$ columns

Output: A longest path from source (0,0) to sink (m,n) in the grid.

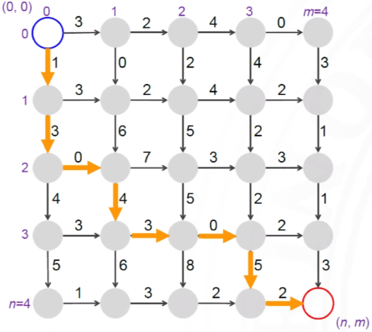




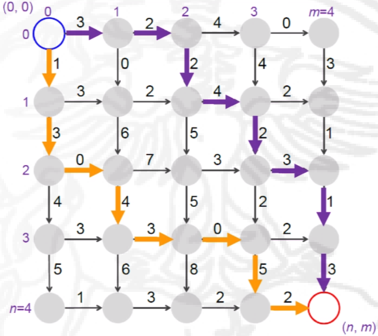
A path in the graph

The Manhattan Tourist Problem

- To solve this problem applying **brute force** is impractical ⇒ the number of possible paths is too huge.



1+3+0+4+3+0+5+2=18

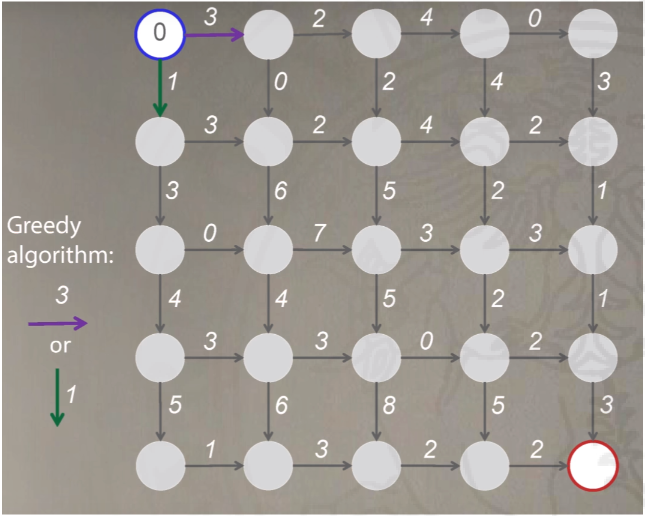


3+2+2+4+2+3+1+3=20

- A **greedy approach** could be exploited: *make the choice that looks best at the moment.*

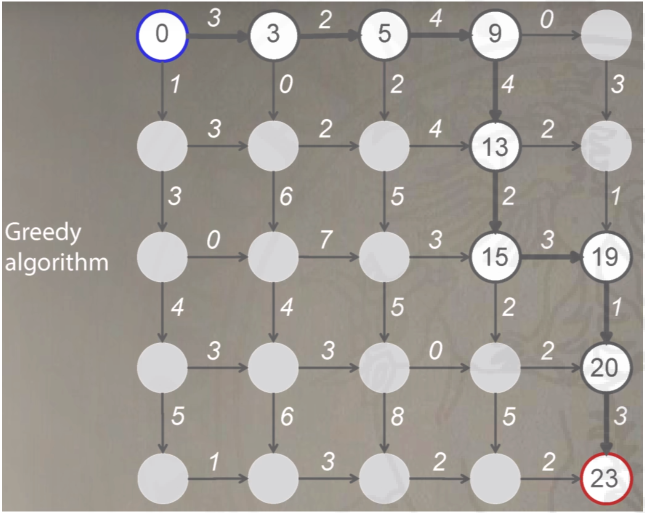
The Manhattan Tourist Problem

- Greedy approach



The Manhattan Tourist Problem

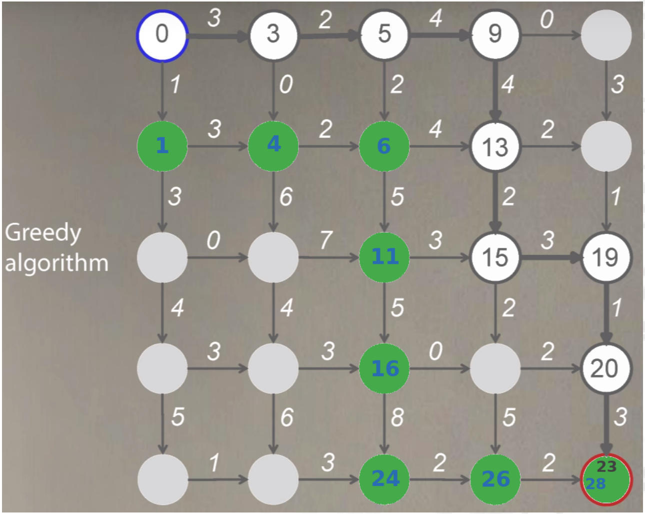
- Greedy approach



Is it the longest path from source to sink?

The Manhattan Tourist Problem

- Greedy approach does not guarantee to find the longest path from source to sink



The Manhattan Tourist Problem

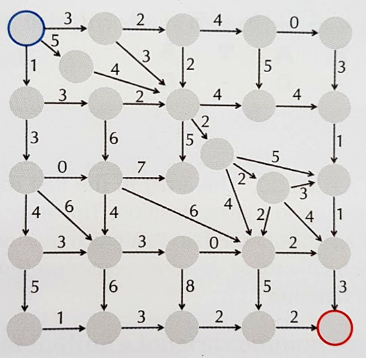
It can be easily generalized for any weighted **Directed Acyclic Graph (DAG)**.

The Manhattan Tourist Problem

Definition: Find a longest path in a weighted DAG

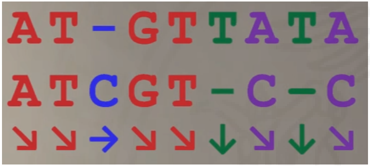
Input: A weighted DAG with a source and a sink

Output: A longest path from source to sink in the DAG.



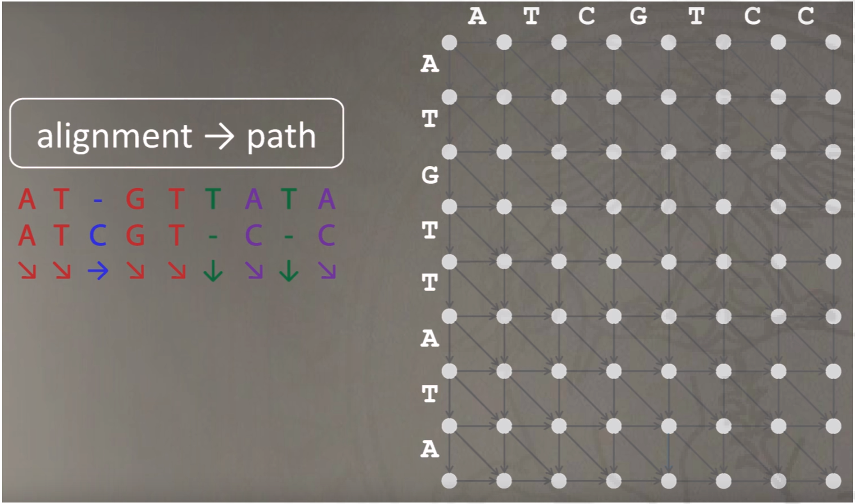
The Manhattan Tourist Problem

What is the connection between the Longest Path Problem and the Alignment Game?



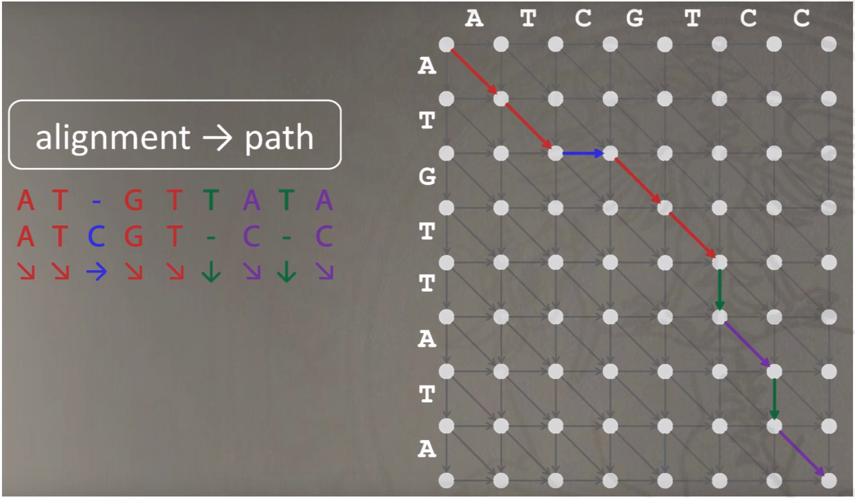
The Manhattan Tourist Problem

What is the connection between the Longest Path Problem and the Alignment Game?



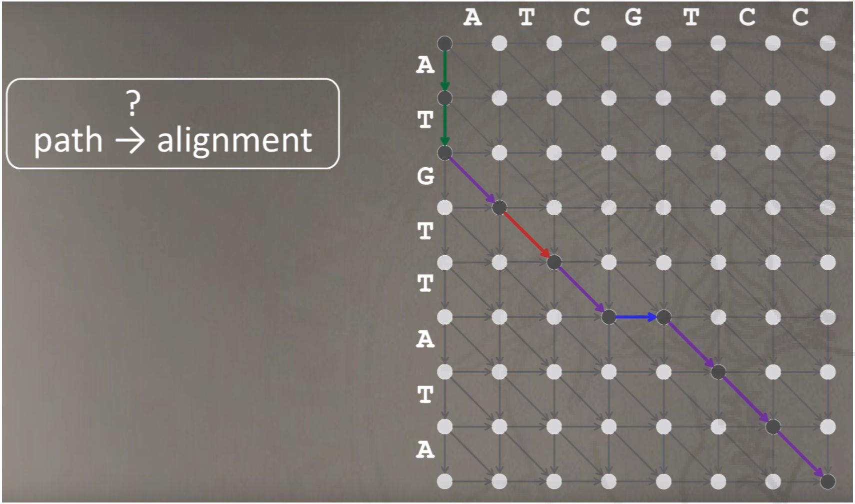
The Manhattan Tourist Problem

What is the connection between the Longest Path Problem and the Alignment Game?



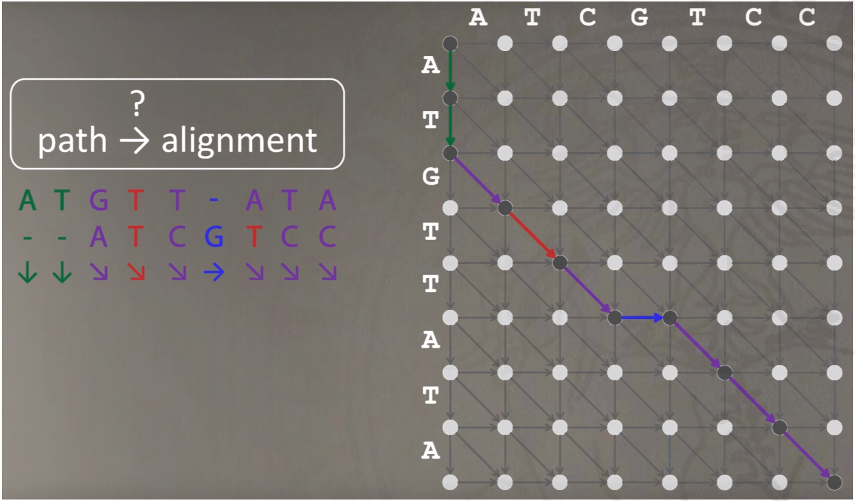
The Manhattan Tourist Problem

A path in the DAG can be always converted in an alignment



The Manhattan Tourist Problem

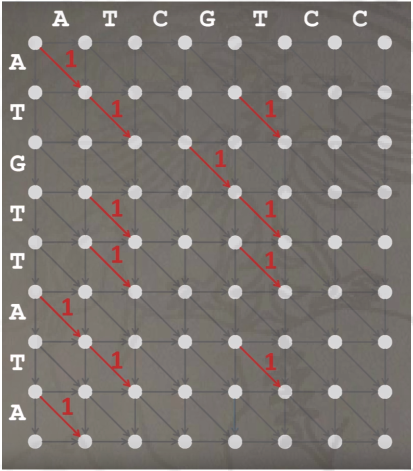
How to build "Manhattan" for the Alignment Game?



The Manhattan Tourist Problem

How to build a “Manhattan” for the Alignment Game?

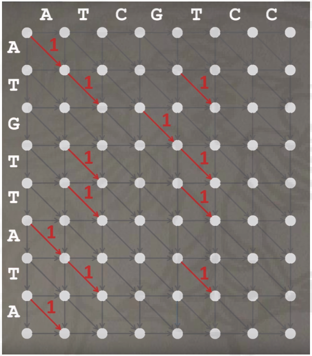
- **Diagonal red edges** correspond to matching symbols and have score **1**;
- All other edges have score 0;



The Manhattan Tourist Problem

How to build a “Manhattan” for the Alignment Game?

- **Diagonal red edges** correspond to matching symbols and have score **1**;
- All other edges have score 0;



highest scoring alignment
=
longest path in a properly built Manhattan

Part 1

The Change Problem

The Change Problem

- To speed-up the search of longest path in a properly built Manhattan **dynamic programming** can be used;
- We introduce dynamic programming through the **Change Problem**

The Change Problem

Definition: Find the minimum number of coins needed to make change

Input: An integer money and an array of positive integers $\langle coin_1, \ldots, coin_n \rangle$

Output: The minimum number of coins $\langle coin_1, \ldots, coin_n \rangle$ that changes money.

The Change Problem

Changing Money with a Greedy approach

```
GreedyChange(money)
  change ← empty collection of coins
  while money > 0
    coin ← largest denomination that does not exceed money
    add coin to change
    money ← money − coin
  return change
```

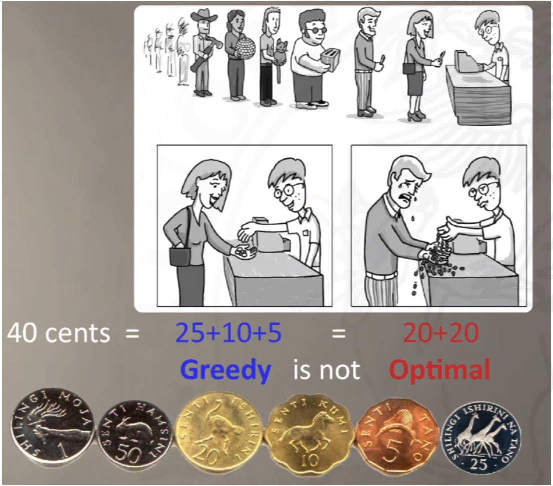
The Change Problem

Changing Money with a Greedy approach in Tanzania

The Change Problem

Changing Money with a Greedy approach in Tanzania

GreadyChange Fails



The Change Problem

Changing Money with a recursive approach

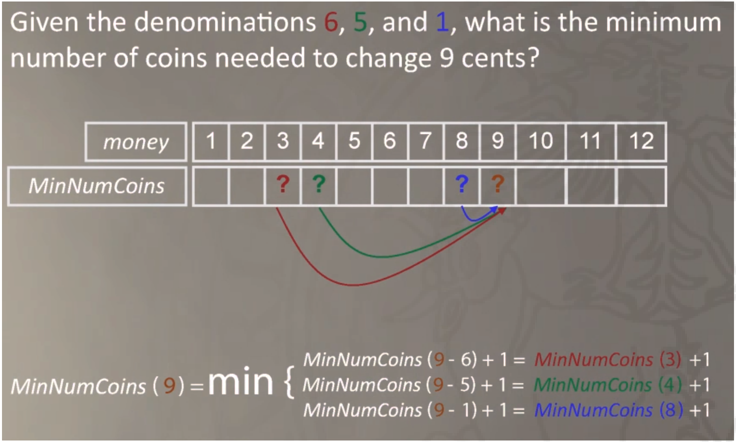
Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?

money	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins									?			

MinNumCoins (9) = ?

The Change Problem

Changing Money with a recursive approach



The Change Problem

Changing Money with a recursive approach

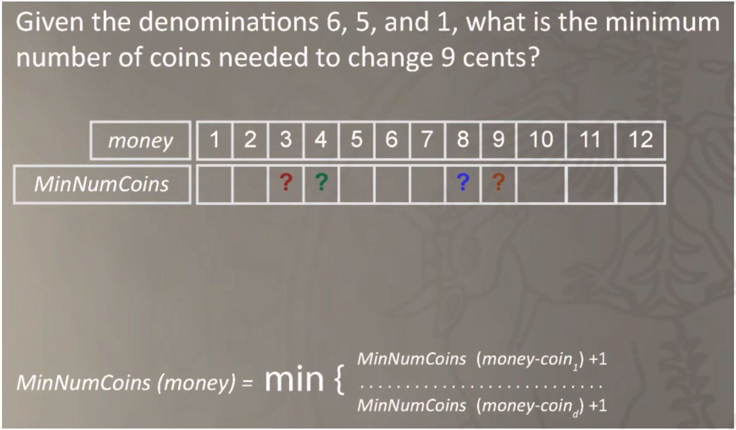
Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?

money	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins			?	?				?	?			

$$MinNumCoins(money) = \min \begin{cases} MinNumCoins(money - 6) + 1 \\ MinNumCoins(money - 5) + 1 \\ MinNumCoins(money - 1) + 1 \end{cases}$$

The Change Problem

Changing Money with a recursive approach



The Change Problem

Recursive Change algorithm

```
RecursiveChange(money, coins)
  if money = 0
    return 0
  MinNumCoins ← infinity
  for i ← 1 to |coins|
    if money ≥ coin_i
      NumCoins ← RecursiveChange(money-coin_i, coins)
      if numCoins + 1 < MinNumCoins
        MinNumCoins ← numCoins + 1
  return MinNumCoins
```

The Change Problem

Recursive Change algorithm

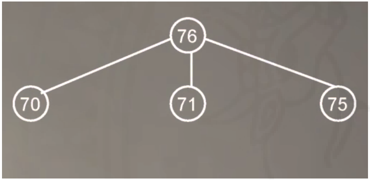
```
RecursiveChange(money, coins)
  if money = 0
    return 0
  MinNumCoins ← infinity
  for i ← 1 to |coins|
    if money ≥ coin_i
      NumCoins ← RecursiveChange(money-coin_i, coins)
      if numCoins + 1 < MinNumCoins
        MinNumCoins ← numCoins + 1
  return MinNumCoins
```

- it is **correct**: it finds the minimum number of coins that changes the money, but ..
- it **very expensive** in time and memory!!!

The Change Problem

- To show how fast is the recursive change we consider the recursive tree for changing 76 cents;
- We assume 6, 5 and 1 as possible coin values.

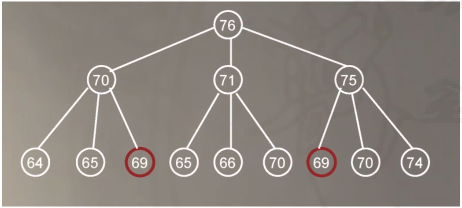
Recursive Tree



The Change Problem

- To show how fast is the recursive change we consider the recursive tree for changing 76 cents;
- We assume 6, 5 and 1 as possible coin values.

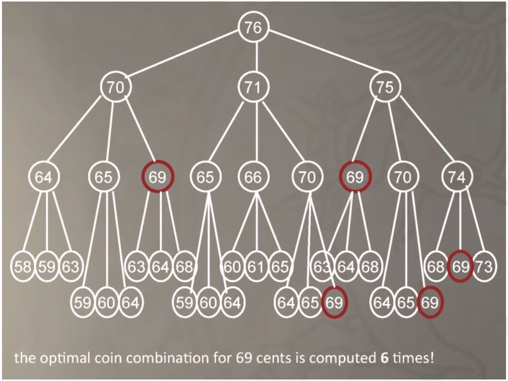
Recursive Tree



The Change Problem

- To show how fast is the recursive change we consider the recursive tree for changing 76 cents;
- We assume 6, 5 and 1 as possible coin values.

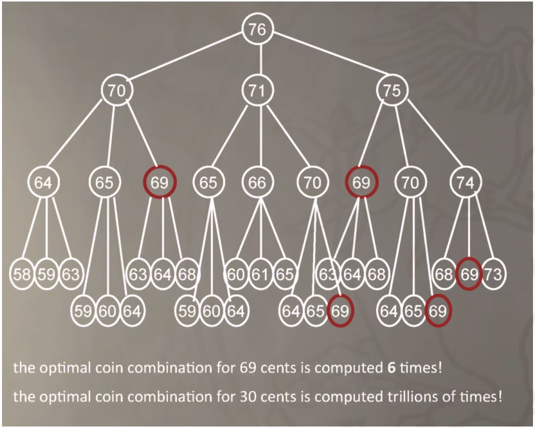
Recursive Tree



The Change Problem

- To show how fast is the recursive change we consider the recursive tree for changing 76 cents;
- We assume 6, 5 and 1 as possible coin values.

Recursive Tree



The Change Problem

Changing Money with dynamic programming

Richard Bellman
(August 26, 1920 - March 19, 1984)

- He was an American applied mathematician;
- He developed dynamic programming in 1953.
- He was awarded the IEEE Medal of Honor in 1979:
  contributions to decision processes and control system theory, particularly the creation and application of dynamic programming

The Change Problem

Changing Money with dynamic programming

- we compute all the values of ***MinNumCoins(money$-$coin$_i$)*** before computing ***MinNumCoins(money)***;
- instead of time consuming reversely calls we simply look up the values previously computed to generate the new one.

The Change Problem

Changing Money with dynamic programming

What is the minimum number of coins needed to change 9 cents for denominations 6, 5, and 1?

money	0	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins	0												

The Change Problem

Changing Money with dynamic programming

What is the minimum number of coins needed to change 9 cents for denominations 6, 5, and 1?

money	0	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins	0	1											

The Change Problem

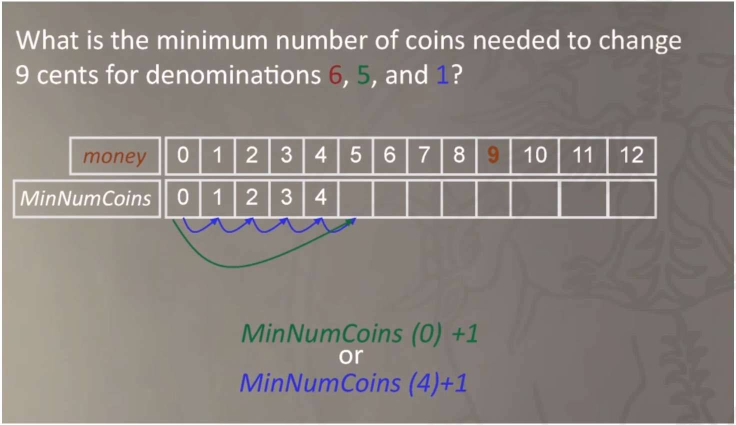
Changing Money with dynamic programming

What is the minimum number of coins needed to change 9 cents for denominations 6, 5, and 1?

money	0	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins	0	1	2	3	4								

The Change Problem

Changing Money with dynamic programming



The Change Problem

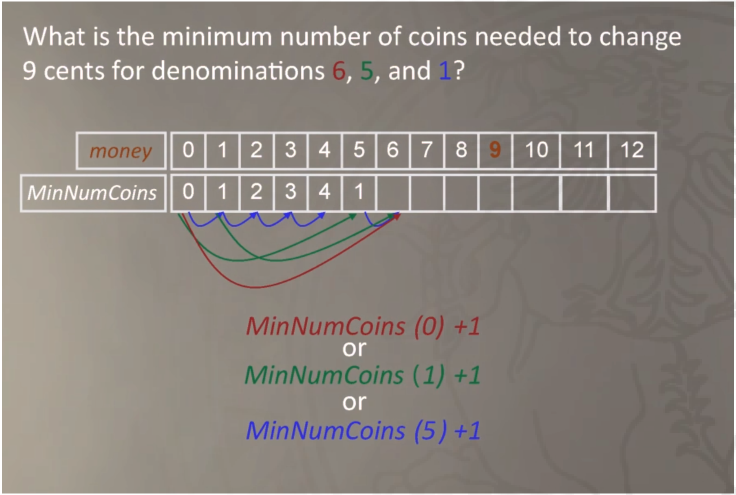
Changing Money with dynamic programming

What is the minimum number of coins needed to change 9 cents for denominations 6, 5, and 1?

money	0	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins	0	1	2	3	4	1							

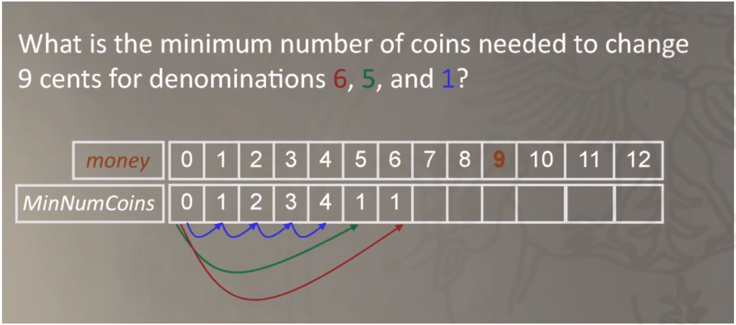
The Change Problem

Changing Money with dynamic programming



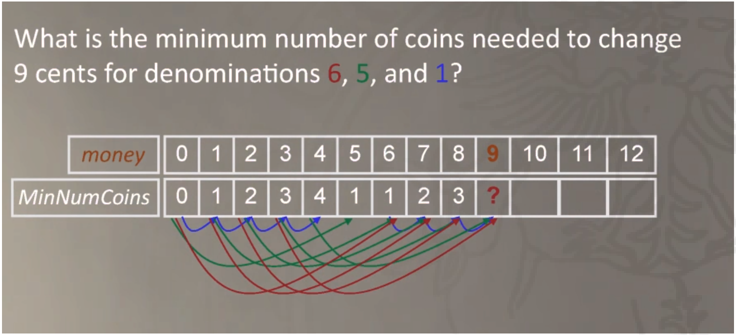
The Change Problem

Changing Money with dynamic programming



The Change Problem

Changing Money with dynamic programming



The Change Problem

Changing Money with dynamic programming

```
DPChange(money, coins)
  MinNumCoins(0) ← 0
  for m ← 1 to money
    MinNumCoins(m) ← infinity
    for i ← 1 to |coins|
      if m ≥ coin_i
        if MinNumCoins(m – coin_i) + 1 < MinNumCoins(m)
          MinNumCoins(m) ← MinNumCoins(m – coin_i) + 1
  return MinNumCoins(money)
```