

Tugas Tantangan IF2211 Strategi Algoritma
Penyelesaian *Traveling Salesman Problem* (TSP) Menggunakan *Dynamic Programming* dengan Bahasa Ruby



Disusun oleh :

Erdianti Wiga Putri Andini (13522053)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

DAFTAR ISI.....	2
DESKRIPSI TUGAS.....	3
SOURCE CODE DAN PENJELASAN PROGRAM.....	4
HASIL OUTPUT PROGRAM.....	7
1. Test Case 1 (tes1.txt).....	7
2. Test Case 2 (tes2.txt).....	7
3. Test Case 3 (tes3.txt).....	8
LAMPIRAN.....	9

DESKRIPSI TUGAS

Program yang dibuat adalah program penyelesaian persoalan Traveling Salesman Problem (TSP) dengan menggunakan *Dynamic Programming*. Tugas ini ditujukan sebagai tantangan yang menjadi bonus poin untuk tugas kecil sebelumnya. Pada tugas ini, diberi beberapa opsi bahasa yaitu Rust, Perl, Swift, atau Ruby. Saya menggunakan bahasa Ruby untuk mengimplementasikan program ini.

SOURCE CODE DAN PENJELASAN PROGRAM

File main.rb

```
require 'set'
require 'matrix'

def readFile(path)
  file = File.open(path)
  lines = file.readlines.map(&:chomp)
  n = lines.size # Ukuran matriks
  adjMatrix = Array.new(n) { Array.new(n, Float::INFINITY) }

  lines.each_with_index do |line, i|
    val = line.split.map { |x| x == "infinity" ? Float::INFINITY : x.to_f } #
    Ubah infinity ke Float::INFINITY
    adjMatrix[i] = val
  end

  [n, adjMatrix]
end

def TSP(i, s, adjMatrix, memo, startIdx)
  if s.empty?
    return adjMatrix[i][startIdx] # Pastikan balik ke titik awal
  end
  return memo[[i, s]] if memo.key?([i, s])

  minCost = Float::INFINITY
  s.each do |j|
    next if adjMatrix[i][j] == Float::INFINITY

    s_ = s.dup
    s_.delete(j)
    cost = adjMatrix[i][j] + TSP(j, s_, adjMatrix, memo, startIdx)
    minCost = [minCost, cost].min
  end

  memo[[i, s]] = minCost
  minCost
end

def getPath(i, s, adjMatrix, memo, startIdx)
  if s.empty?
    return [startIdx] # Pastikan balik ke titik awal
  end

  minCost = Float::INFINITY
  minPath = []

  s.each do |j|
    next if adjMatrix[i][j] == Float::INFINITY

    s_ = s.dup
```

```

    s_.delete(j)
    cost = adjMatrix[i][j] + TSP(j, s_, adjMatrix, memo, startIdx)
    if cost < minCost
      minCost = cost
      minPath = [j, *getPath(j, s_, adjMatrix, memo, startIdx)]
    end
  end

  minPath
end

def getRoute(start, n, adjMatrix, memo)
  s = Set.new(1...n) - [start]
  path = getPath(start, s, adjMatrix, memo, start)
  [start, *path]
end

title = <<ART
|_|/|_||. \ /|_||. | | | | | | | | | | \
| | \ \ \ \ \ / \ \ \ \ \ | | | | | | | | | | /
|_| [|_|/|_|] [|_|/|_|] [|_|/|_|] [|_|/|_|] [|_|/|_|]
                                     13522053

ART

puts title

puts "Input the file that want to be loaded (e.g., tes.txt):"
fileName = gets.chomp
file = File.join("../", "test", fileName)

n, adjMatrix = readFile(file)

puts "Enter the starting city index (1 to #{n}):"
startIdx = gets.to_i - 1

memo = {}
minCost = TSP(startIdx, Set.new((0...n).to_a - [startIdx]), adjMatrix, memo,
startIdx)
route = getRoute(startIdx, n, adjMatrix, memo)
route = route.map { |x| x + 1 }

puts "Most optimal TSP route is [ #{route.join(' - ')} ] with cost #{minCost}"

```

Implementasi *dynamic programming* dalam program ini terlihat dalam penggunaan memoization, yang merupakan teknik utama dalam *dynamic programming* untuk menghindari perhitungan ulang sub-*problem* yang sama. Teknik ini dilakukan dengan menyimpan hasil dari operasi yang membutuhkan biaya komputasi tinggi dalam sebuah dictionary memo, di mana

setiap *key* adalah kombinasi dari node saat ini dan subset dari node yang belum dikunjungi. Nilai yang disimpan adalah biaya minimum untuk menjelajahi semua node dalam subset tersebut mulai dari node saat ini dan kembali ke titik awal.

Fungsi TSP menerapkan pendekatan rekursif untuk menghitung biaya minimum dengan terlebih dahulu memeriksa memo untuk melihat apakah hasil sudah diketahui, sehingga mengurangi kebutuhan untuk perhitungan yang berulang. Dalam basis rekursi, jika subset kosong, artinya semua node telah dikunjungi dan langsung mengembalikan biaya dari node saat ini ke titik awal. Dalam kasus rekursif, fungsi mengevaluasi setiap kemungkinan node selanjutnya yang bisa dijangkau, menghitung biaya untuk setiap transisi, dan menambahkannya dengan biaya rekursif dari node selanjutnya untuk mengunjungi sisa node, kemudian memilih biaya minimum dari semua pilihan ini.

Fungsi `getPath` kemudian digunakan untuk mencari jalur yang sesuai dengan biaya minimum yang telah dihitung, menggunakan pendekatan yang sama dan memanfaatkan memo untuk efisiensi. Ini memastikan bahwa setiap *sub-problem* hanya dihitung sekali dan hasilnya digunakan kembali sebanyak yang dibutuhkan.

HASIL OUTPUT PROGRAM

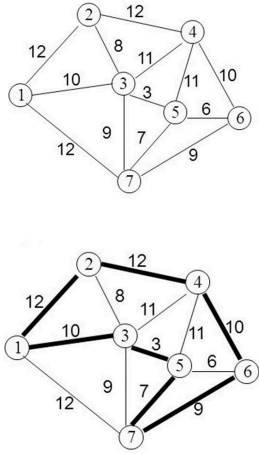
1. Test Case 1 (tes1.txt)

Test Case	Output
<pre>tes1.txt 0 10 15 20 5 0 9 10 6 13 0 12 8 8 9 0</pre>	<pre>PS D:\ITB\Tugas Stima\Tantangan\TugasTantangan_13522053\src> ruby main.rb 13522053 Input the file that want to be loaded (e.g., tes.txt): tes1.txt Enter the starting city index (1 to 4): 2 Most optimal TSP route is [2 - 3 - 4 - 2] with cost 35.0</pre>

2. Test Case 2 (tes2.txt)

Test Case	Output
<pre>tes2.txt 0 20 30 10 11 15 0 16 4 2 3 5 0 2 4 19 6 18 0 3 16 4 7 16 0</pre>	<pre>PS D:\ITB\Tugas Stima\Tantangan\TugasTantangan_13522053\src> ruby main.rb 13522053 Input the file that want to be loaded (e.g., tes.txt): tes2.txt Enter the starting city index (1 to 5): 4 Most optimal TSP route is [4 - 2 - 5 - 3 - 4] with cost 28.0</pre>

3. Test Case 3 (tes3.txt)

Test Case	Output
 <pre>tes3.txt 0 12 10 infinity infinity infinity 12 12 0 8 12 infinity infinity infinity 10 8 0 11 3 infinity 9 infinity 12 11 0 11 10 infinity infinity infinity 3 11 0 6 7 infinity infinity infinity 10 6 0 9 12 infinity 9 infinity 7 9 0</pre>	<pre>PS D:\ITB\Tugas Stima\Tantangan\TugasTantangan_13522053\src> ruby main.rb 13522053 Input the file that want to be loaded (e.g., tes.txt): tes3.txt Enter the starting city index (1 to 7): 1 Most optimal TSP route is [1 - 2 - 4 - 6 - 7 - 5 - 3 - 1] with cost 63.0</pre>

LAMPIRAN

Pranala ke repository:

https://github.com/wigaandini/TugasTantangan_13522053