NHẬP MÔN LẬP TRÌNH
(Introduction to Programming)

Chapter 7 – Algorithm with Python
(Lesson 3 – More Applications …)

Presenter: **Dr. Nguyen Dinh Long**

Email: dinhlonghcmut@gmail.com

Phone: +84 947 229 599

Google-site: https://sites.google.com/view/long-dinh-nguyen

Dec. 2022

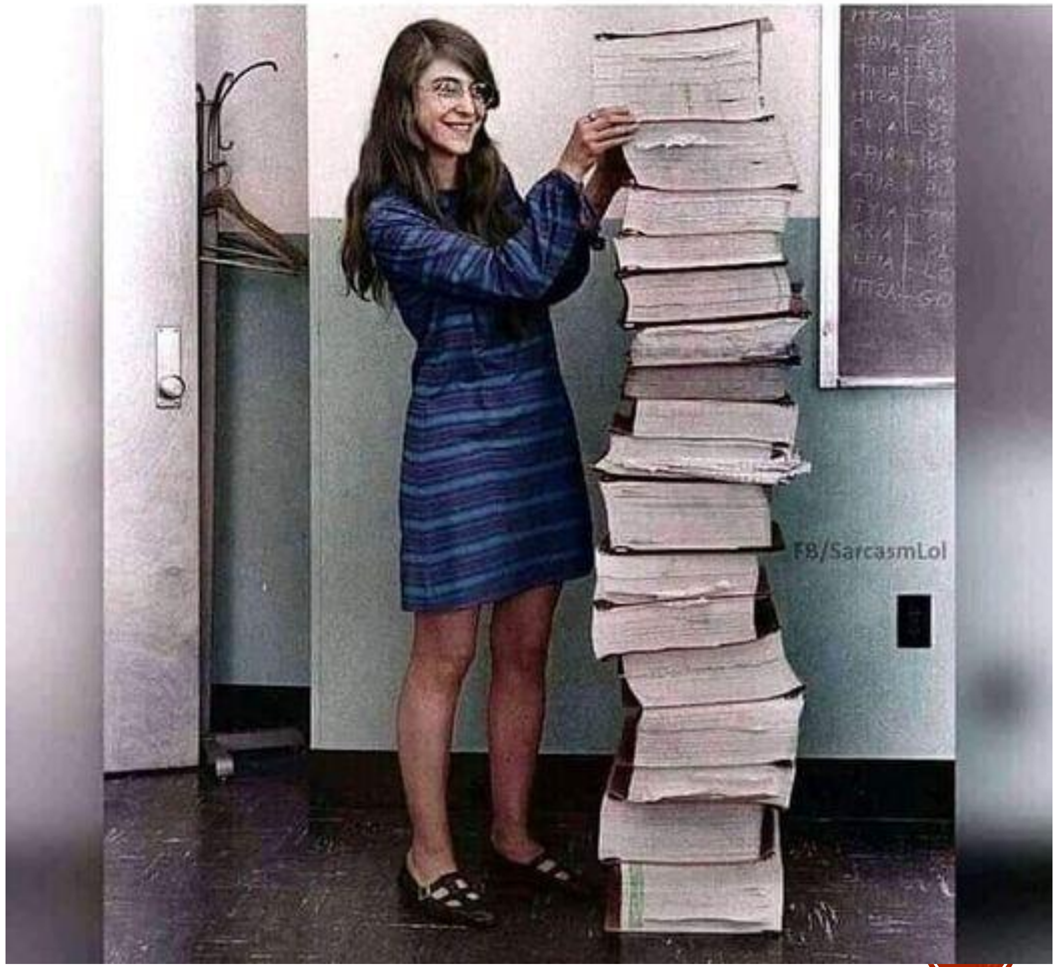
Programming

.

Hamilton was a self-taught programmer, working in the US in the 1960’s. Owing to the success of her previous work, Hamilton was the first programmer to be hired for the Apollo project. She became the Director of Software Engineering at the MIT Instrumentation lab. Her lab developed the on-board flight software for NASA’s Apollo space project, which took humankind to the moon.

The achievement was a monumental task at a time when computer technology was in its infancy: The astronauts had access to only 72 kilobytes of computer memory (a 256-gigabyte cell phone today carries almost a million times more storage space). Programmers had to use paper punch cards to feed information into room-sized computers with no screen interface.

Margaret Hamilton, NASA’s lead software engineer for the Apollo, stands next to the code she wrote by hand that took humanity to the moon in 1969.



Outline

References

Main:

- Maurizio Gabbrielli and Simone Martini, 2010. *Programming Languages: Principles and Paradigms*, Springer.
- Cao Hoàng Trụ, 2004. *Ngôn ngữ lập trình- Các nguyên lý và mô hình*, Nhà xuất bản Đại học Quốc gia Tp. Hồ Chí Minh

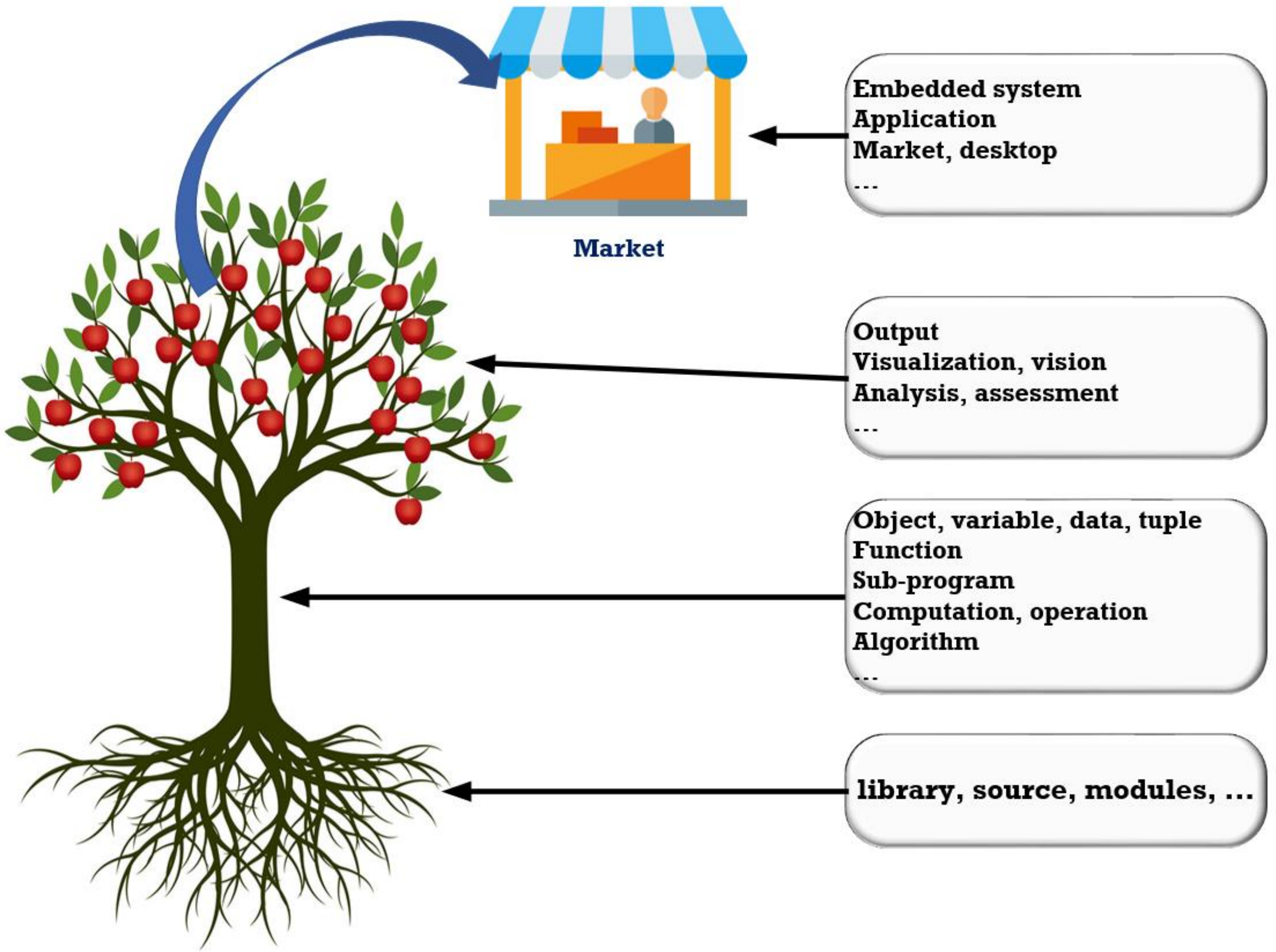
More:

- Wes McKinney, 2013. *Python for Data Analysis*, O'Reilly Media.
- Guido van Rossum, Fred L. Drake, Jr.,, 2012. *The Python Library Reference*, Release 3.2.3.

- Slides here are collected and modified from several sources in Universities and Internet.

Computer programs

- General structure:



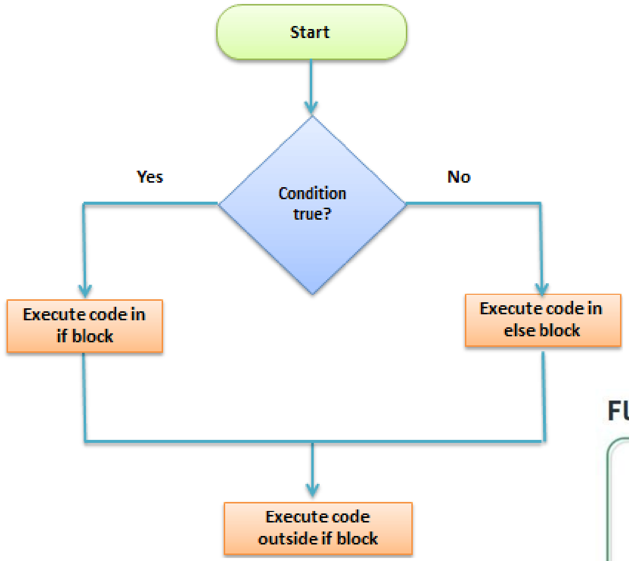
Content of Chapter 7 (Lesson 3)

1. How to Design an Algorithm
2. A Story of Algorithm
3. More Applications and Practices ...

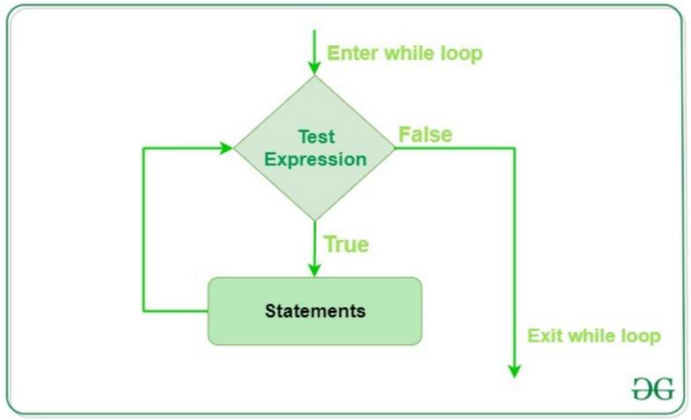
Structure of Computer programs

❑ **Python Programming Analysis:**

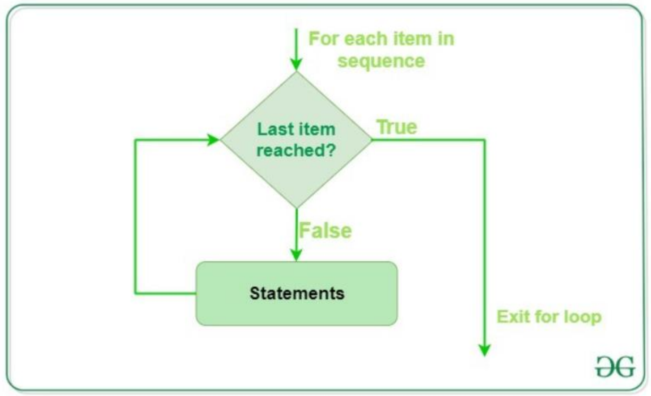
- Objects
- Types: boolean, integer, float, string, complex
- Variables: global variables, local variables
- Methods, Calculation, Computations
- Classes, functions
- Sequences: list, tuples, set, dictionary
- Arrays: 1D array (vector), 2D array (matrix), n-D array ...



Flowchart of While Loop :

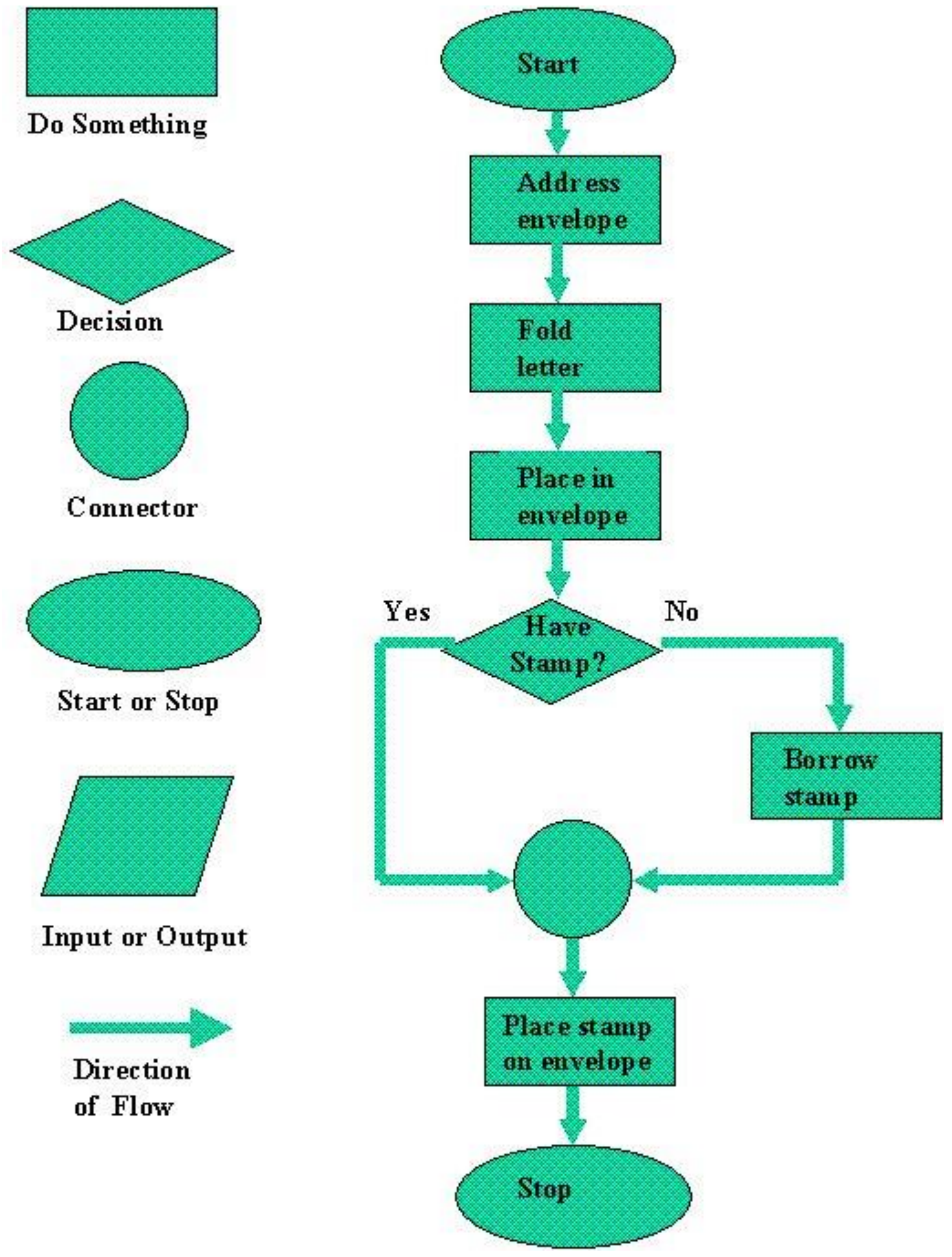


Flowchart of for loop



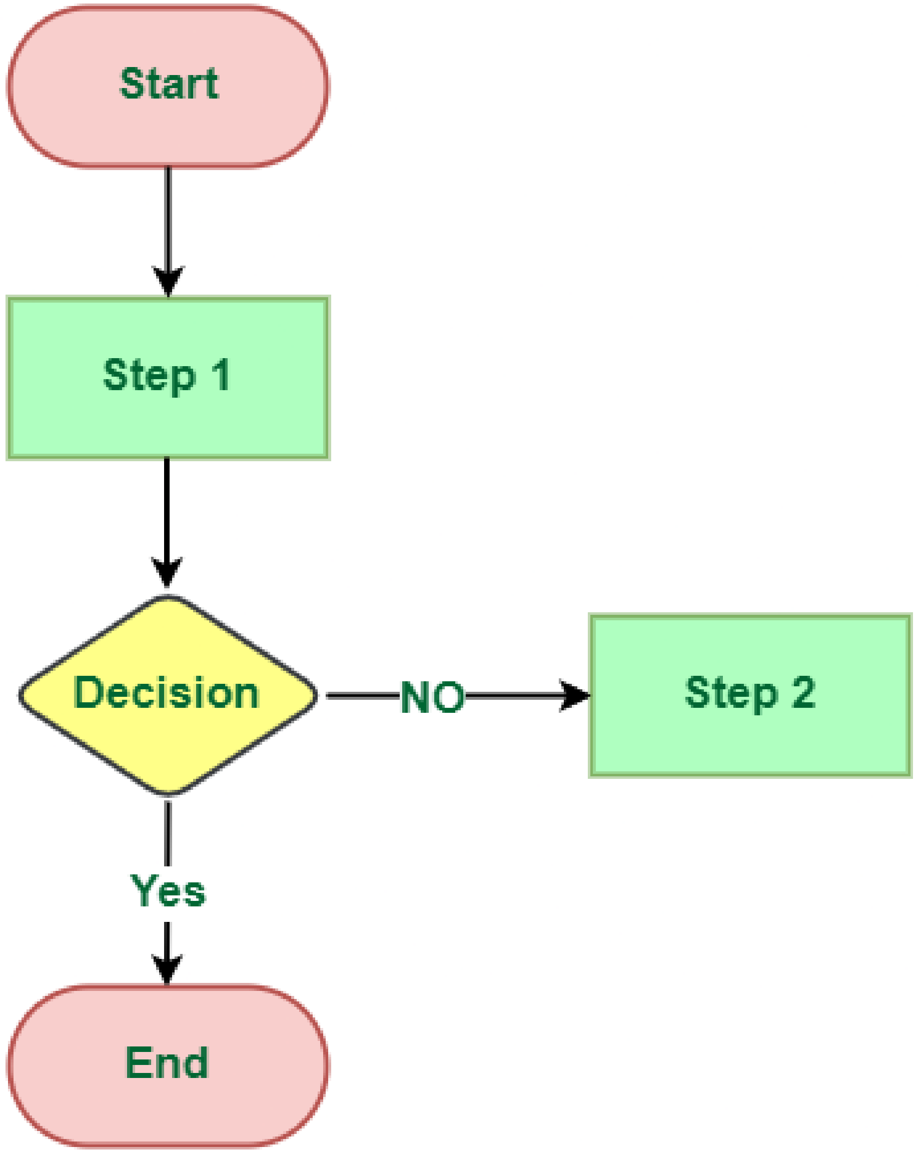
Structure of Computer programs

- **Computer programming:**
  - Modeling
  - Data Reading-Writing-Updating-Deleting
  - Flow charts, diagram, graph/figure ...



Python Programming

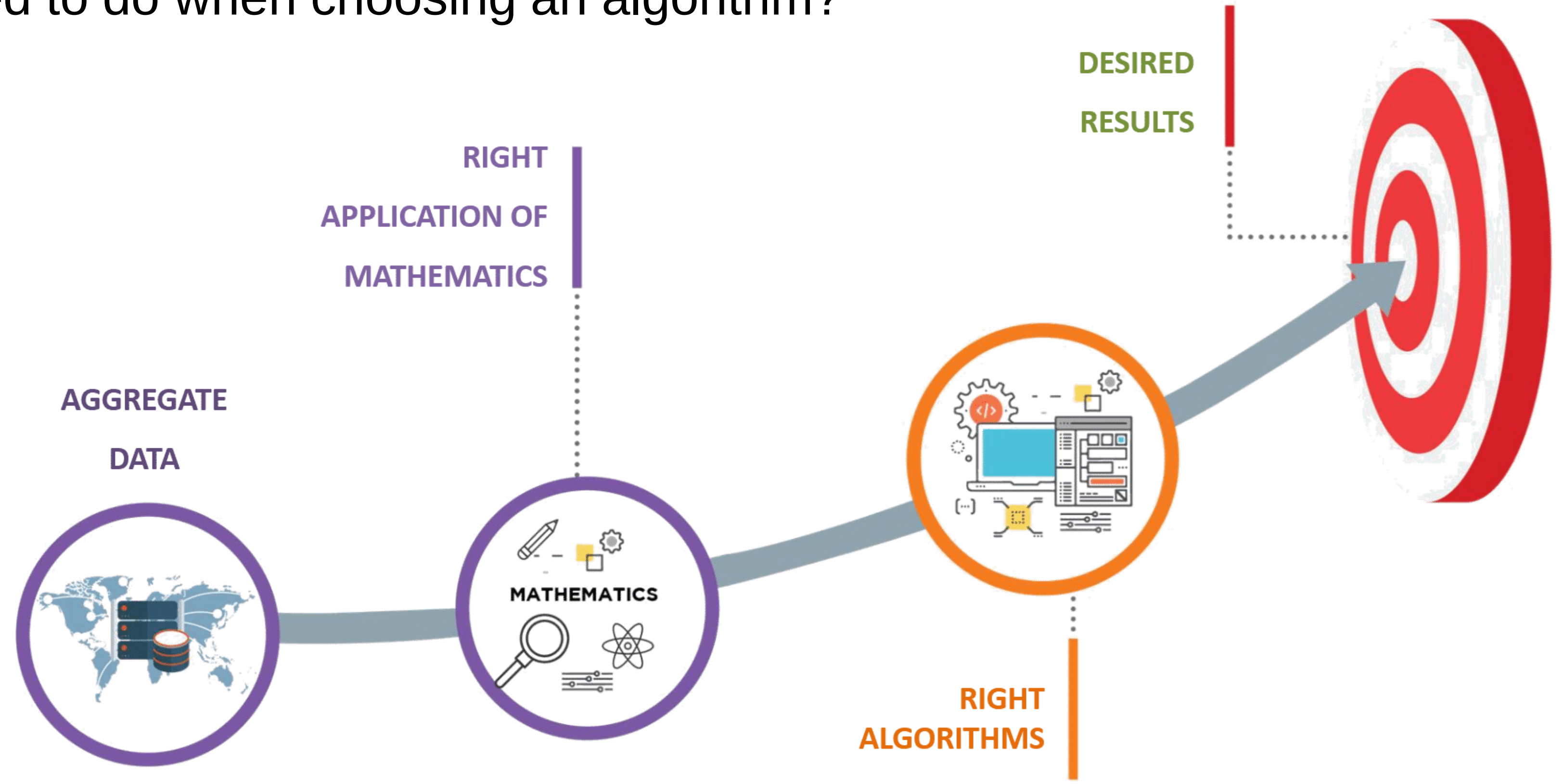
Algorithms



Python Algorithms

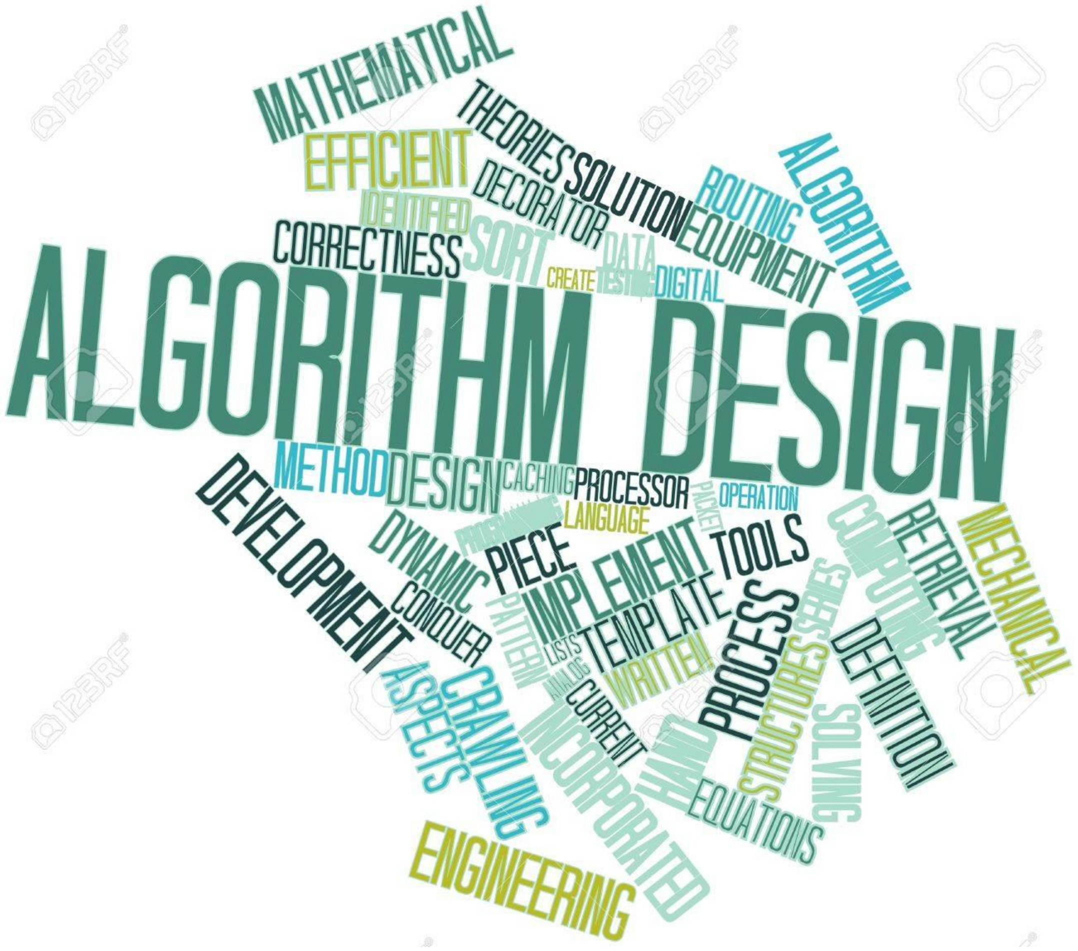
❑ Thinking in Algorithms:

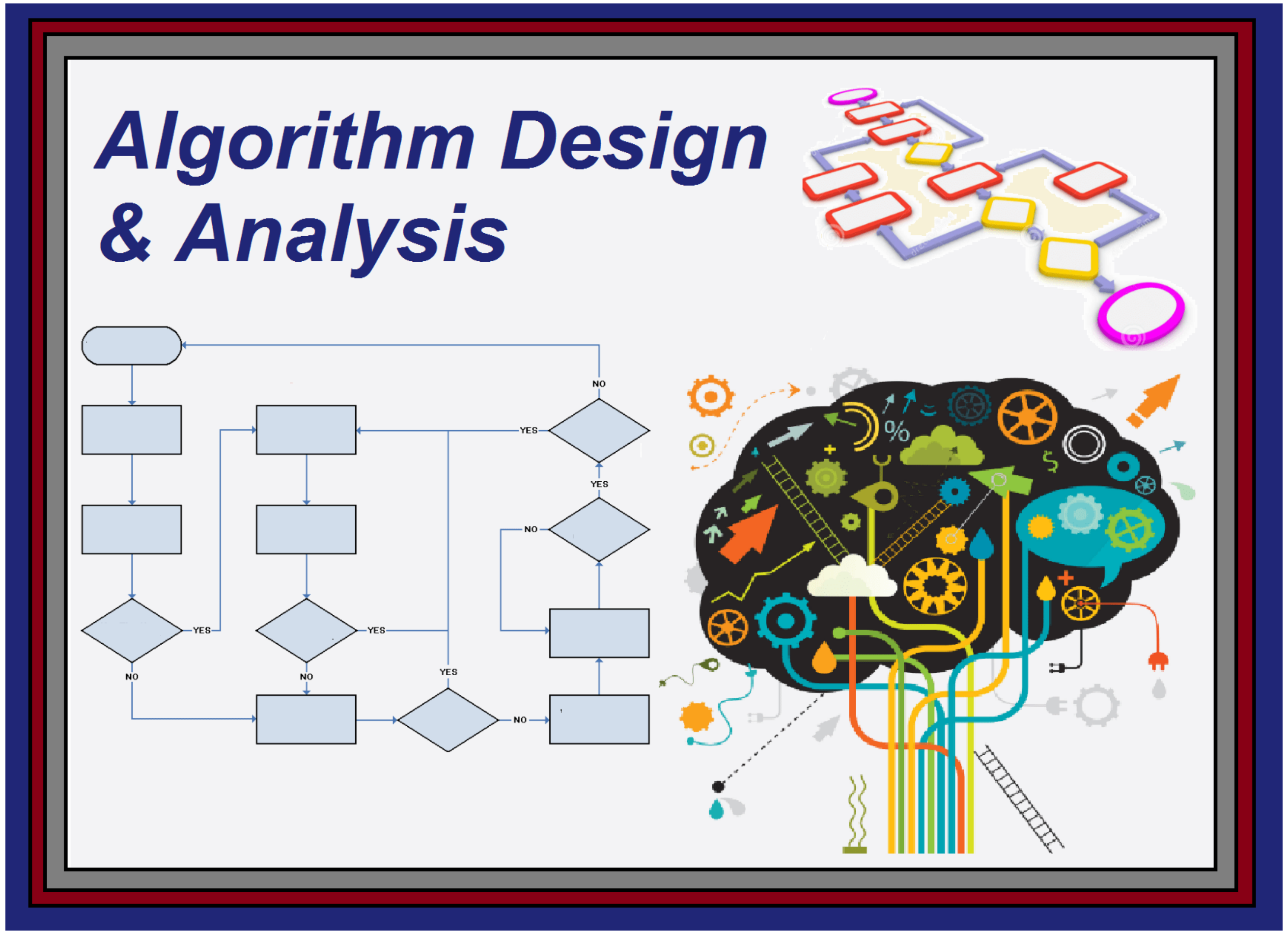
- Data structures have been tightly tied to algorithms since the dawn of computing.
- A number of general approaches used by algorithms to solve problems.
- What do you need to do when choosing an algorithm?



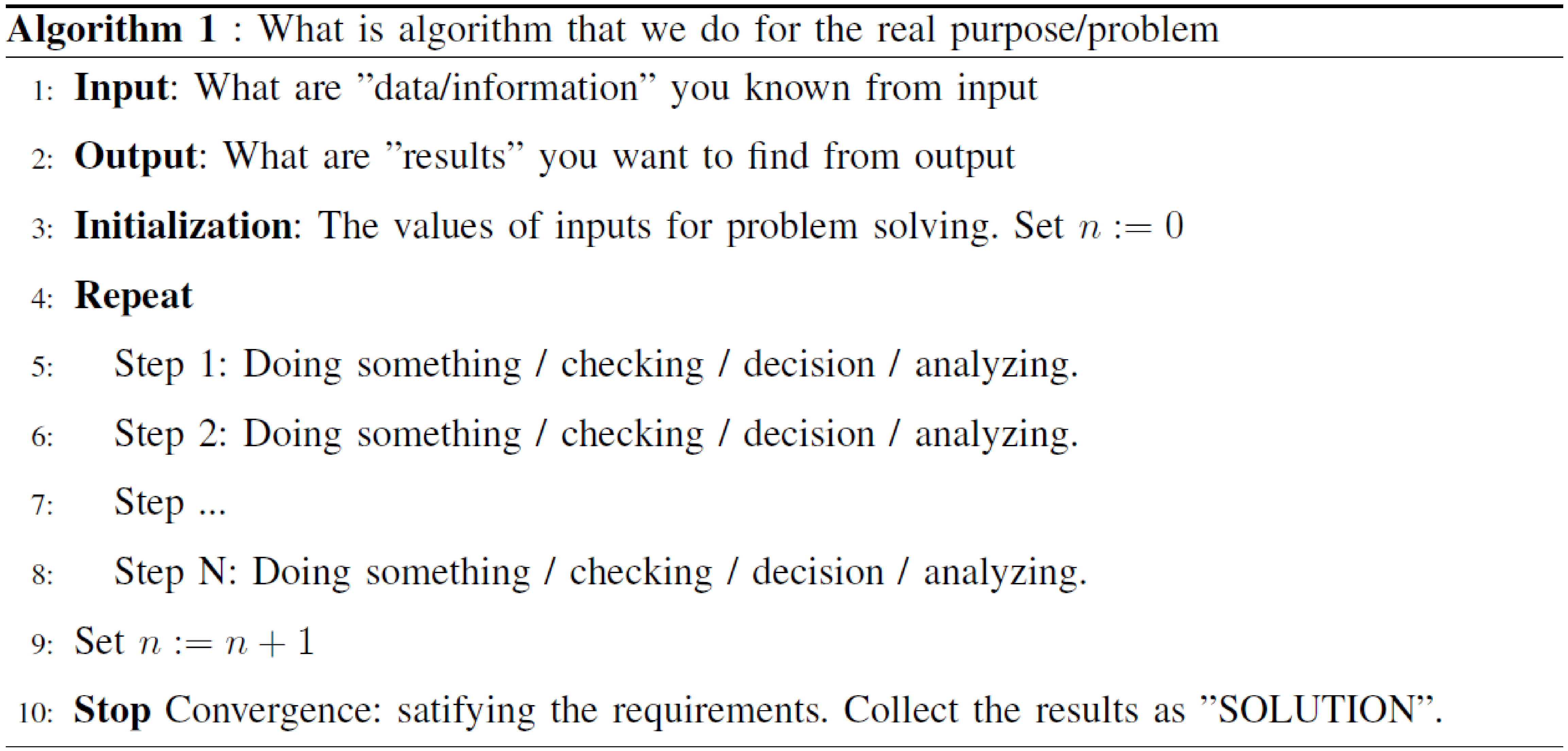
Python Algorithms

Design of Algorithm in Python Programming





Python Algorithms



Algorithm 1 : What is algorithm that we do for the real purpose/problem

```
Input: What are "data/information" you known from input
Output: What are "results" you want to find from output
Initialization: The values of inputs for problem solving. Set $n := 0$
Repeat
    Step 1: Doing something / checking / decision / analyzing.
    Step 2: Doing something / checking / decision / analyzing.
    Step ...
    Step N: Doing something / checking / decision / analyzing.
Set $n := n + 1$
Stop Convergence: satifying the requirements. Collect the results as "SOLUTION".
```

A story of Alogrithm

Travelling Salesman Problem

https://en.wikipedia.org/wiki/Travelling_salesman_problem

Python Algorithms

❑ **Travelling Salesman Problem (TSP)**

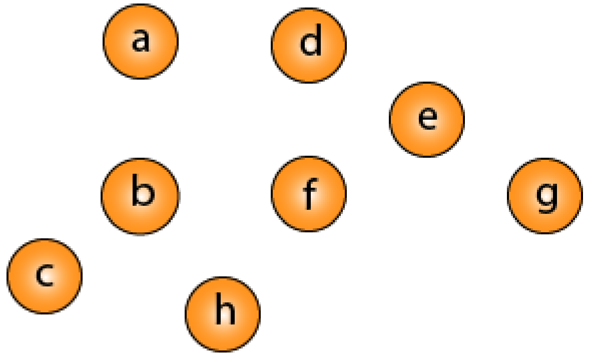
The travelling salesman problem (the traveling salesperson problem or TSP) asks the following question:

"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

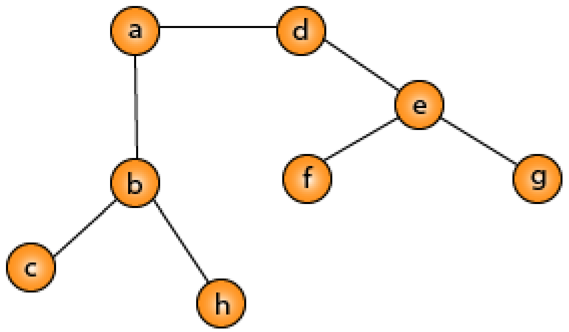
- The problem was first formulated in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods.
- Even though the problem is computationally difficult, many heuristics and exact algorithms are known, so that some instances with tens of thousands of cities can be solved completely and even problems with millions of cities can be approximated within a small fraction of 1%.
- It is an NP-hard problem in combinatorial optimization, important in theoretical computer science and operations research.

Python Algorithms

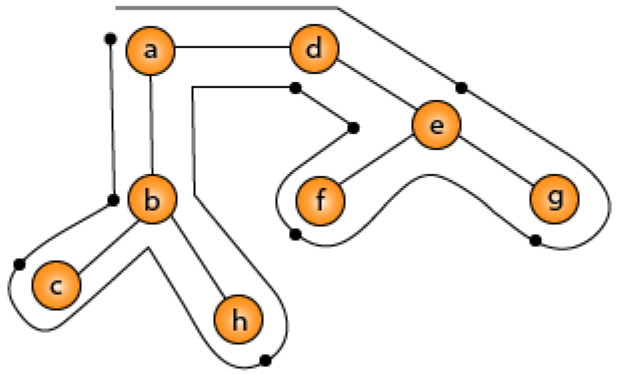
❑ Travelling Salesman Problem (TSP)



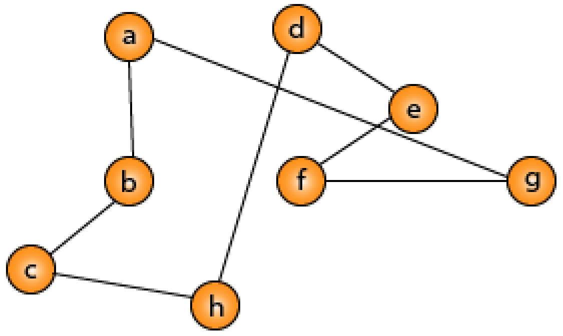
(1) A given set of points



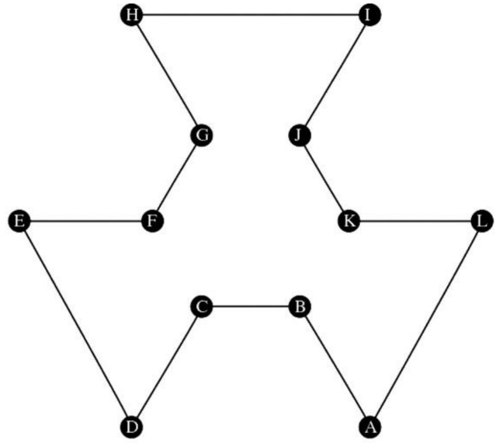
(2) MST T



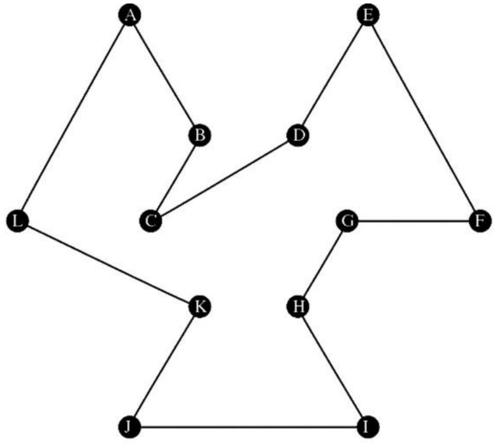
(3) Full tree walk on T.



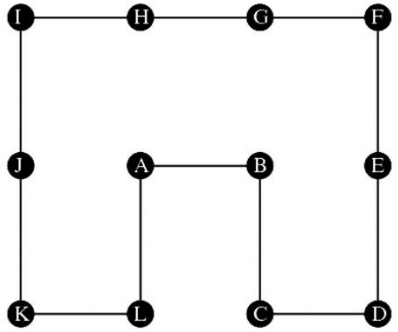
(4) A preorder sequence gives a tour H.



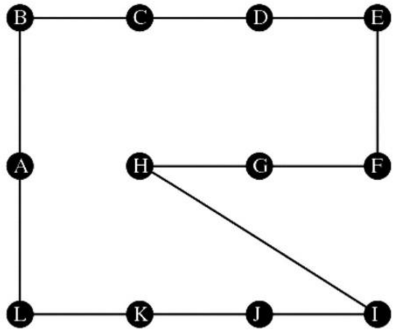
Cost: 54.706521



Cost: 58.974686



Cost: 39.666667



Cost: 41.525684

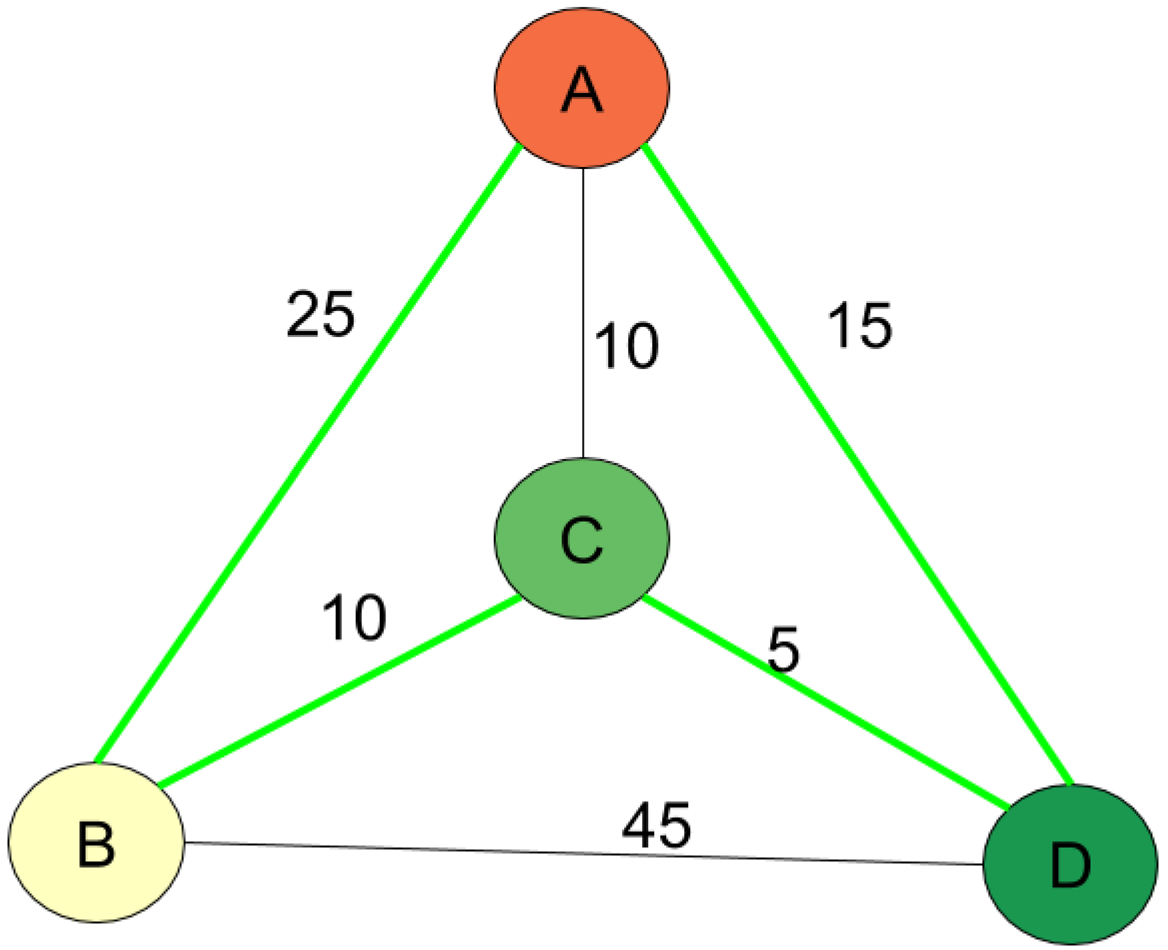
Python Algorithms

- **Travelling Salesman Problem (TSP)**

3 cities …

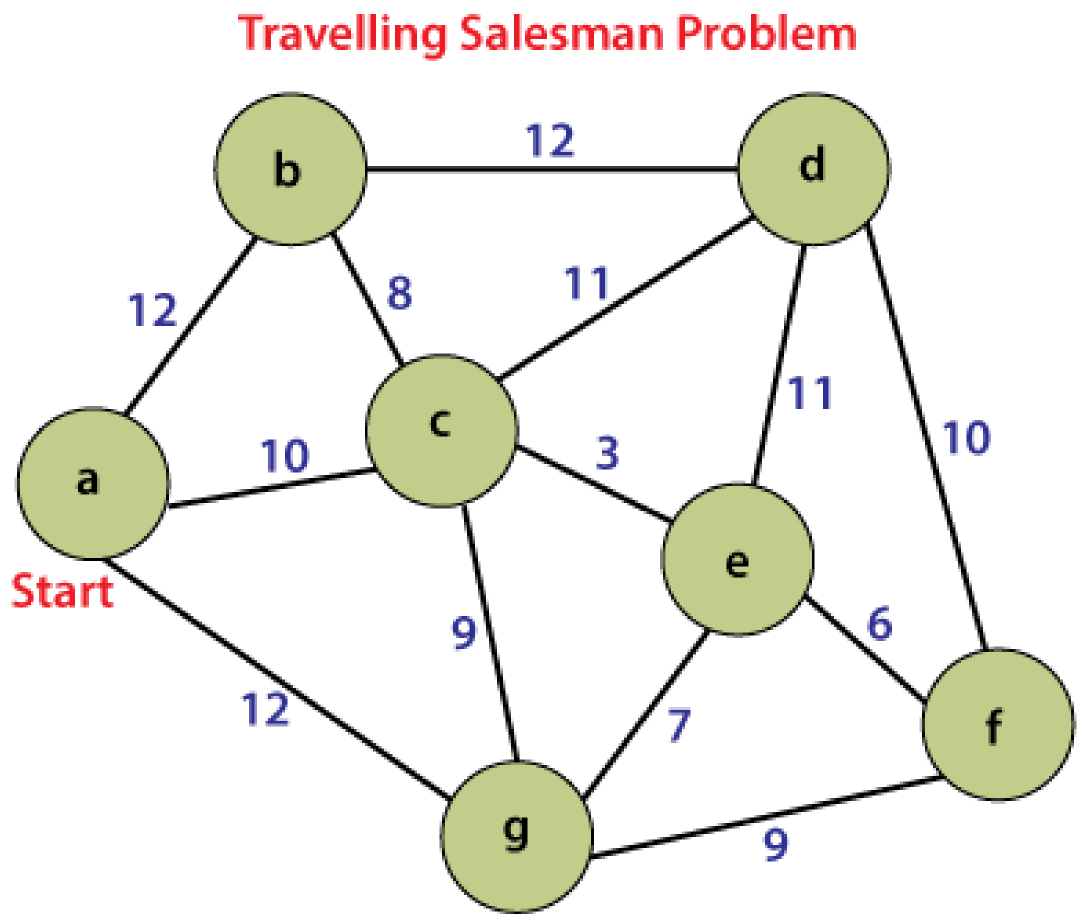
Python Algorithms

- **Travelling Salesman Problem (TSP)**



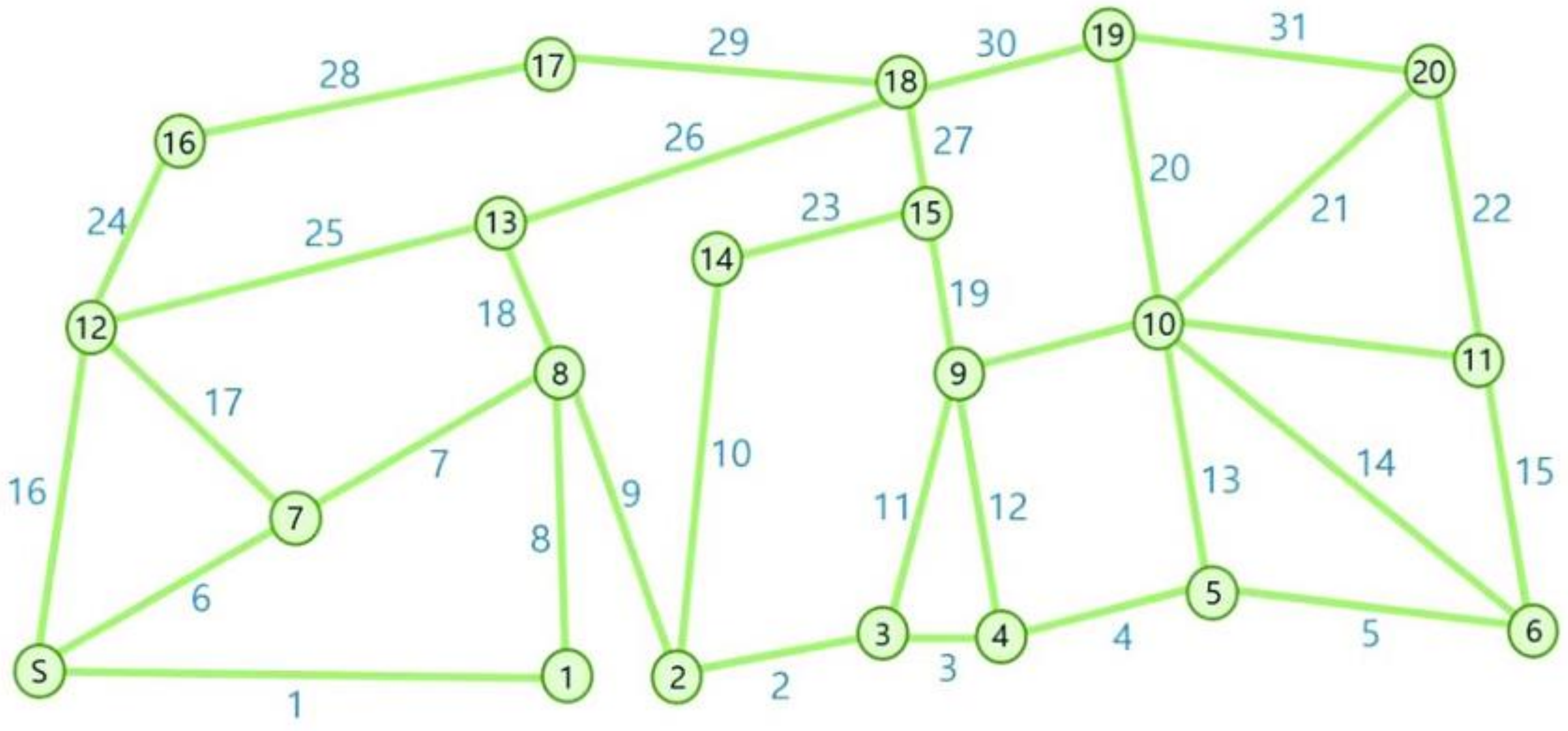
Python Algorithms

- **Travelling Salesman Problem (TSP)**



Python Algorithms

❑ **Travelling Salesman Problem (TSP)**

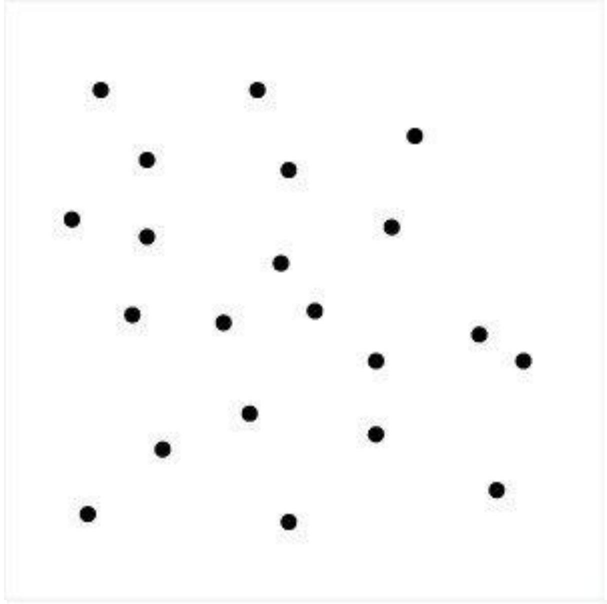


Python Algorithms

- **Travelling Salesman Problem (TSP)**

Python Algorithms

❑ Travelling Salesman Problem (TSP)

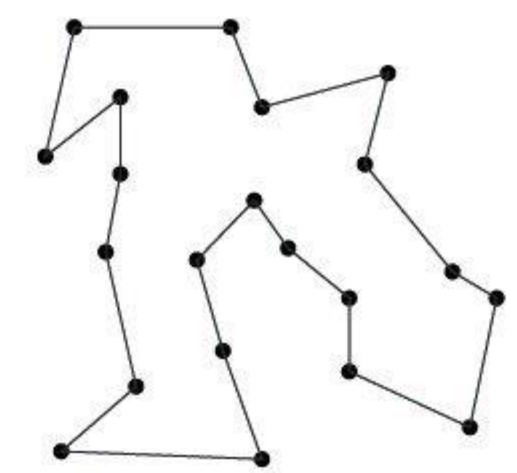


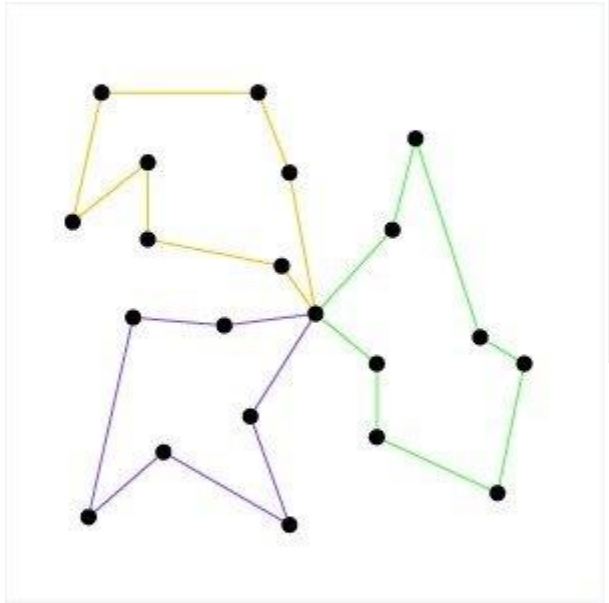
Label the cities with the numbers 1, …, *n* and define:

$$x_{ij} = \begin{cases} 1 & \text{the path goes from city } i \text{ to city } j \\ 0 & \text{otherwise} \end{cases}$$

For *i* = 1, …, *n*, let u_i be a dummy variable, and finally take $c_{ij} > 0$ to be the distance from city *i* to city *j*. Then TSP can be written as the following integer linear programming problem:

$$\begin{aligned} \min \sum_{i=1}^{n} \sum_{j\neq i, j=1}^{n} c_{ij} x_{ij} &: \\ x_{ij} \in \{0,1\} & \qquad i, j = 1, \ldots, n; \\ u_i \in \mathbf{Z} & \qquad i = 2, \ldots, n; \\ \sum_{i=1, i\neq j}^{n} x_{ij} = 1 & \qquad j = 1, \ldots, n; \\ \sum_{j=1, j\neq i}^{n} x_{ij} = 1 & \qquad i = 1, \ldots, n; \\ u_i - u_j + n x_{ij} \leq n - 1 & \qquad 2 \leq i \neq j \leq n; \\ 1 \leq u_i \leq n - 1 & \qquad 2 \leq i \leq n. \end{aligned}$$





Python Algorithms

- **Travelling salesman problem: Brute Force Method**

There are various ways to solve this problem. In this blog, we'll discuss the brute force approach. The travelling salesman problem is a permutation problem. There can be n! total ways to solve permutation problems.

In this approach, we'll generate all the possible permutations (routes). And find the one with the shortest distance.

The algorithm is as follows:

Step 1: Choose a city to act as a starting point(let's say city1).

Step 2: Find all possible routes using permutations. For n cities, it will be (n-1)!.

Step 3: Calculate the cost of each route and return the one with the minimum cost.

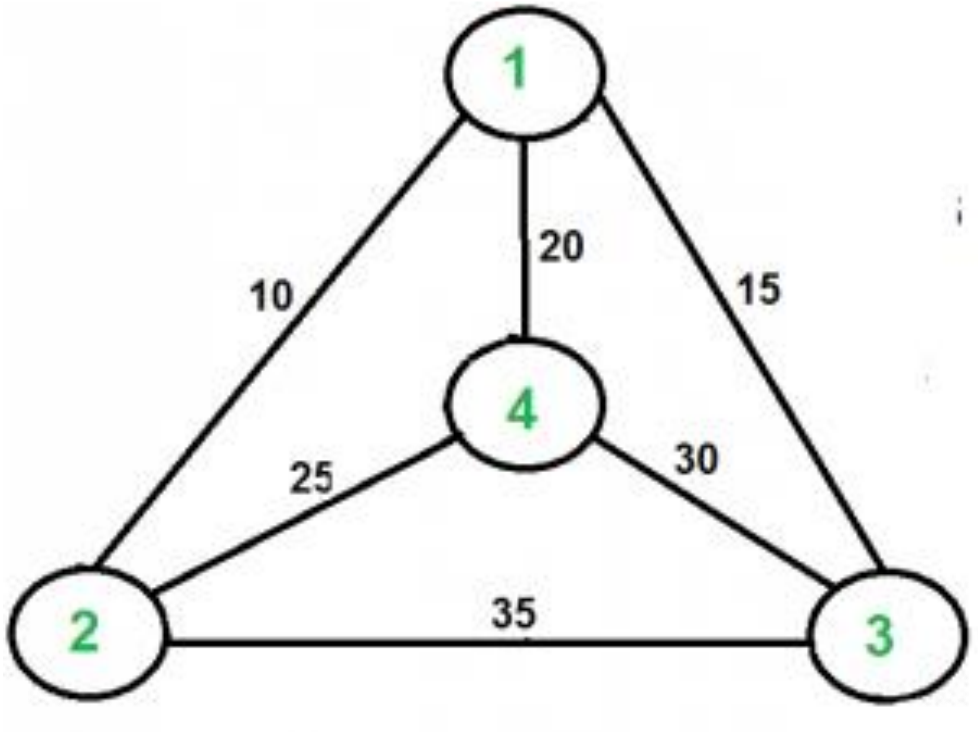
Python Algorithms

- ❑ **Travelling salesman problem:**

Testbed

For example, consider the graph shown in the figure on the right side.

- A TSP tour in the graph is 1-2-4-3-1.
- The cost of the tour is 10+25+30+15 which is 80.



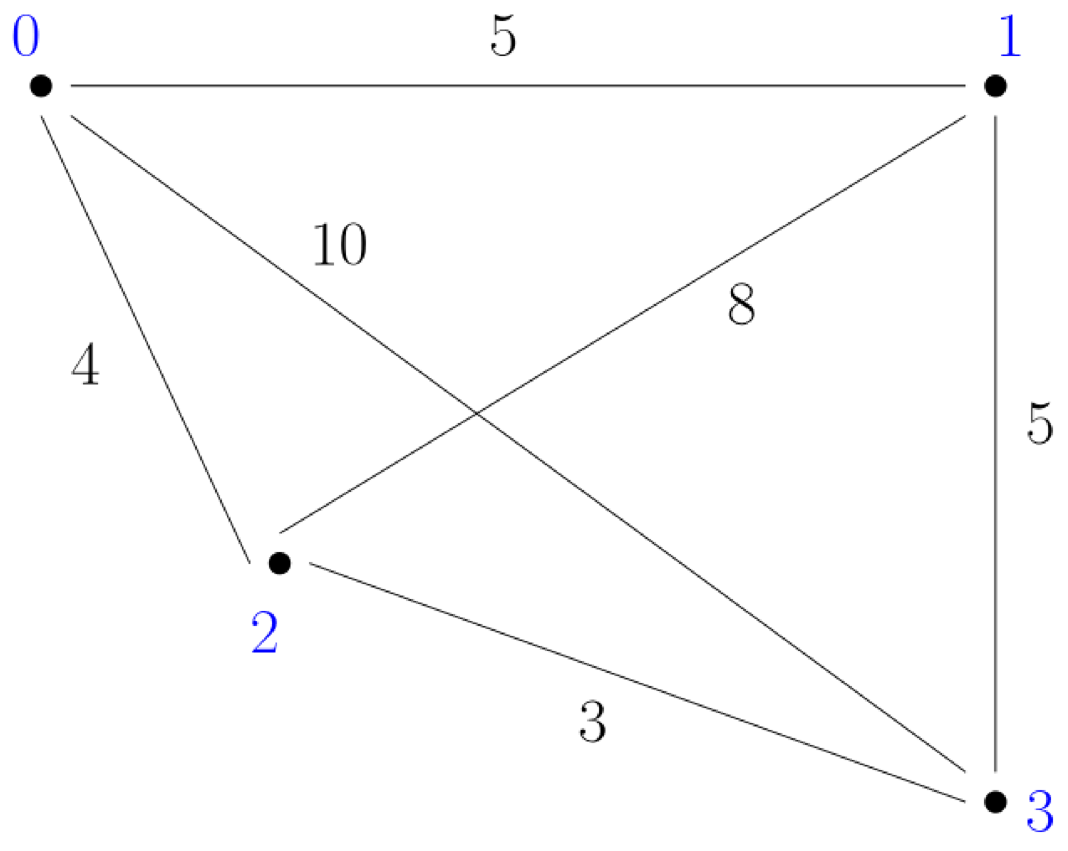
Python Algorithms

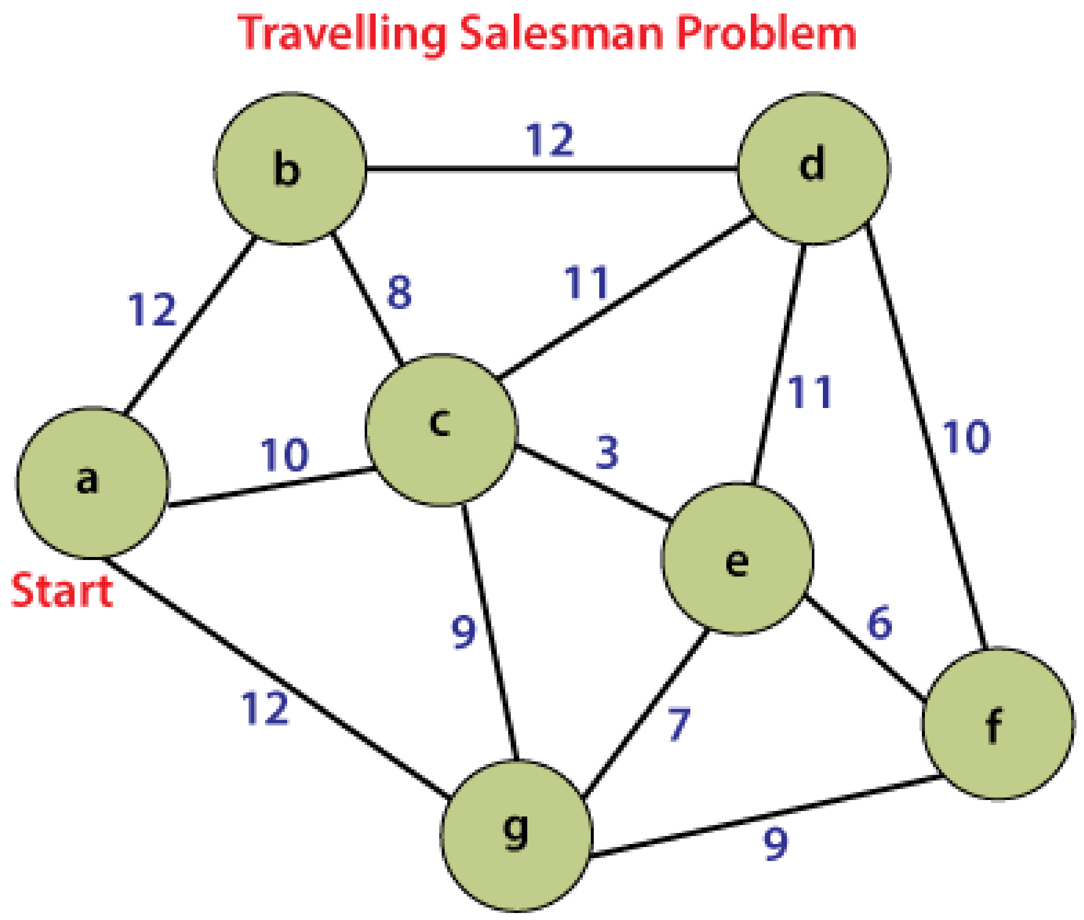
❑ Travelling salesman problem:

Testbed

For example, consider the graph shown in the figure.

- *What is the tour for minimizing the travel cost?*



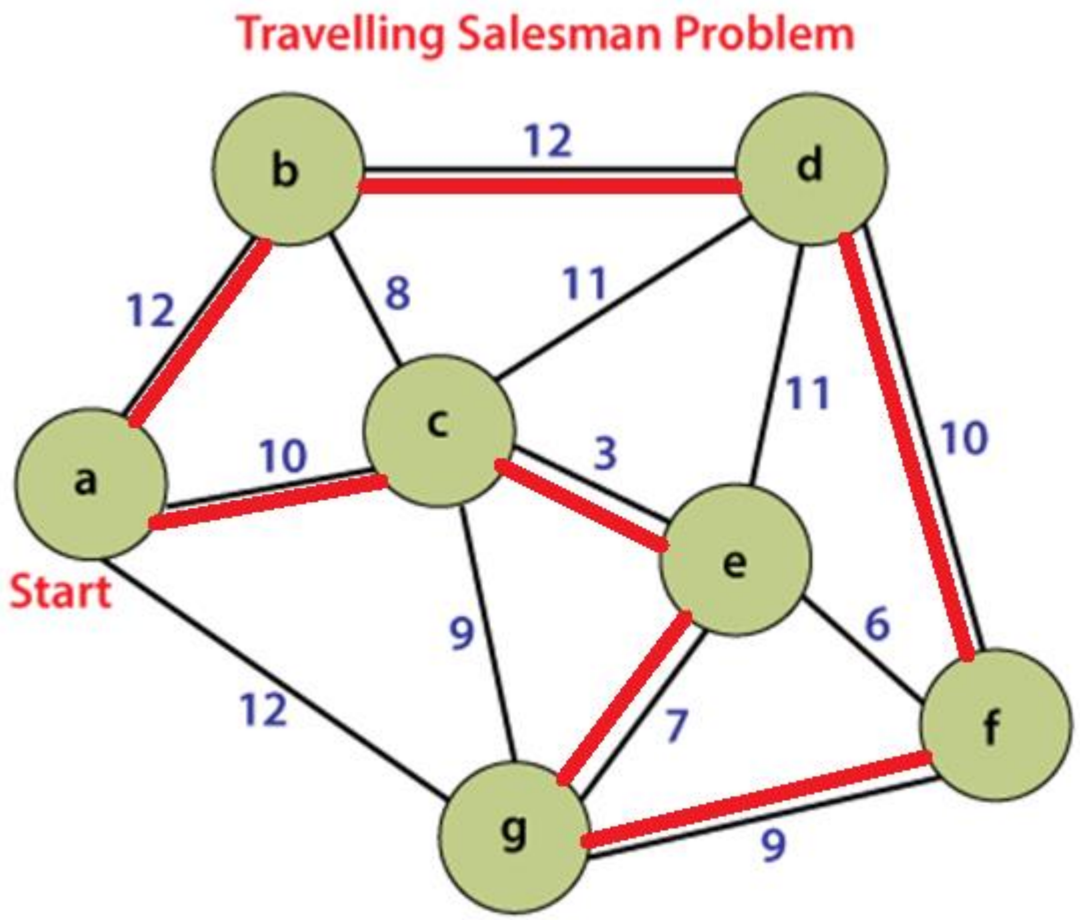


Python Algorithms

❑ Travelling salesman problem:

Testbed

Tour	Cost
a-b-d-f-g-e-c-a (*)	63
a-b-c-d-e-f-g-a	69
a-g-c-e-f-d-b-a	64

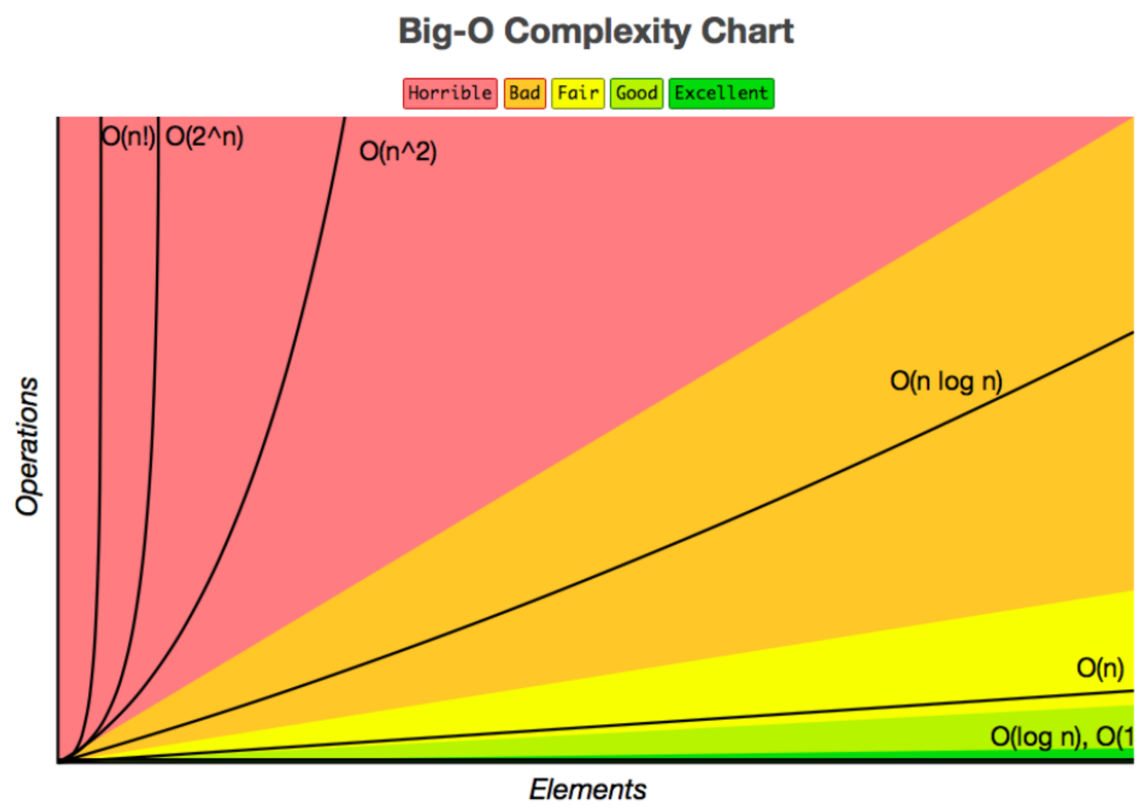


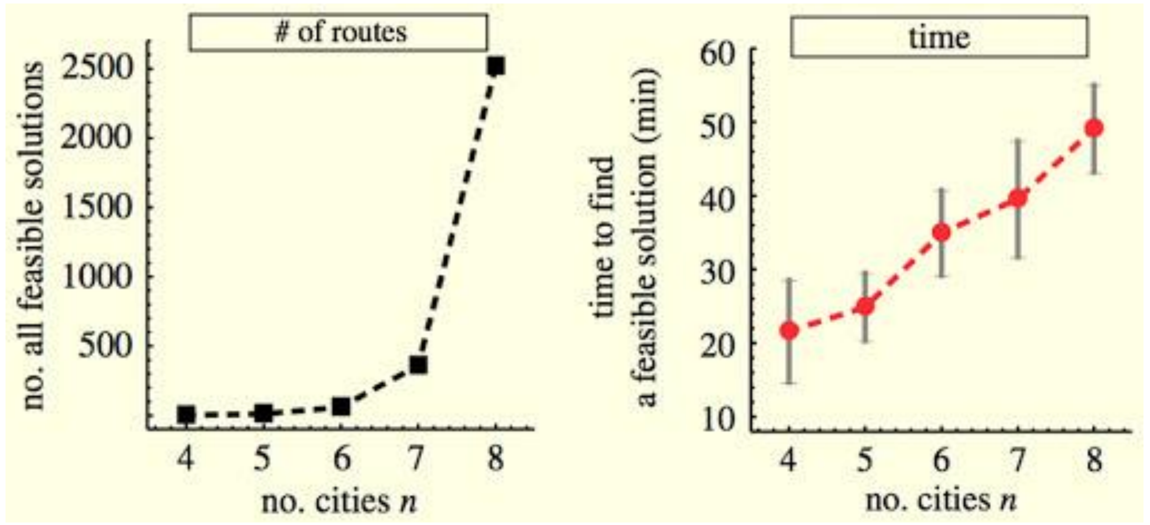
Python Algorithms

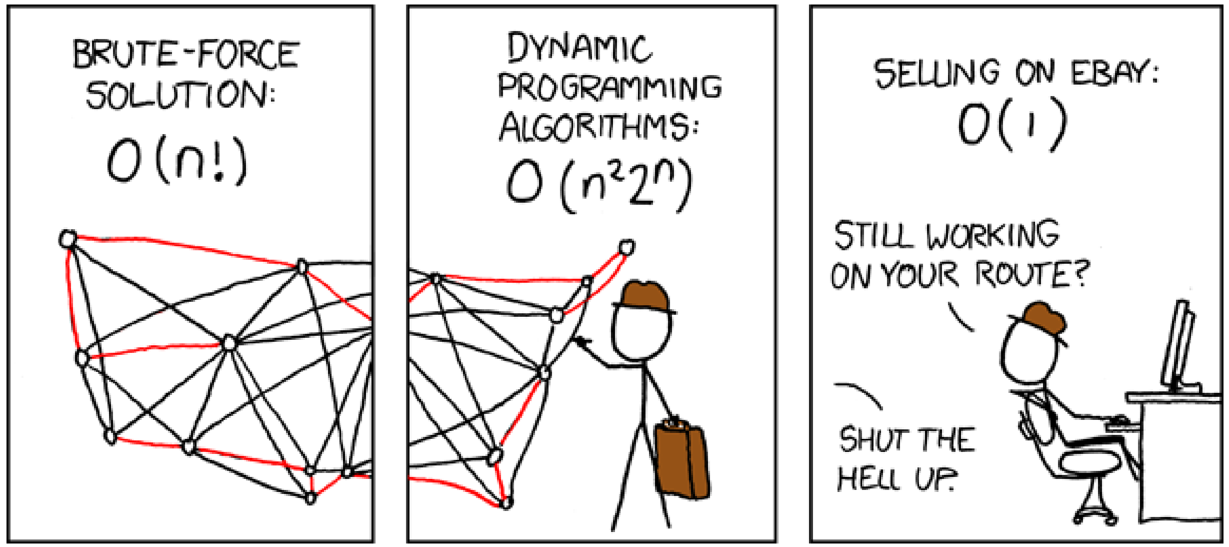
❑ TSP – Complexity:

Number of Cities	Number of Routes	Brute Force Time*
4	3	$^1/_3$ millisecond
5	12	1 millisecond
10	181 440	18 seconds
13	239 500 800	7 hours
15	43 589 145 600	50 days
16	653 837 184 000	2 years
20	60 822 550 000 000 000	193 millennia

*Assuming 10 000 routes mapped per second.






BRUTE-FORCE SOLUTION: O(n!)
DYNAMIC PROGRAMMING ALGORITHMS: O(n²2ⁿ)
SELLING ON EBAY: O(1)
STILL WORKING ON YOUR ROUTE?
SHUT THE HELL UP.

A story of Alogrithm

from

Travelling Salesman Problem

to

Routing Problem

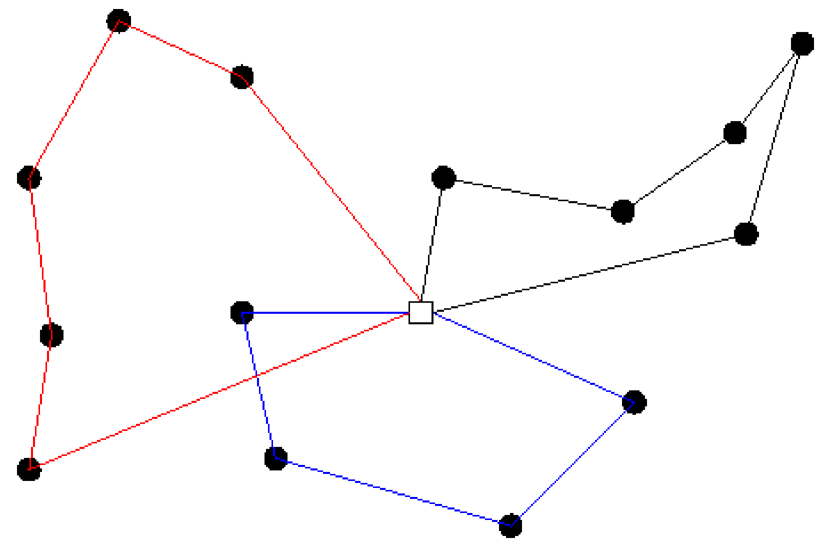
https://en.wikipedia.org/wiki/Vehicle_routing_problem

Python Algorithms

Vehicle routing problem (VRP)

The vehicle routing problem (VRP) is a combinatorial optimization and integer programming problem which asks:

"What is the optimal set of routes for a fleet of vehicles to traverse in order to deliver to a given set of customers?"



- It generalizes the travelling salesman problem (TSP). It first appeared in a paper by George Dantzig and John Ramser in 1959, in which the first algorithmic approach was written and was applied to petrol deliveries.
- Often, the context is that of delivering goods located at a central depot to customers who have placed orders for such goods. The objective of the VRP is to minimize the total route cost.
- Determining the optimal solution to VRP is NP-hard, so the size of problems that can be optimally solved using mathematical programming or combinatorial optimization may be limited. Therefore, commercial solvers tend to use heuristics due to the size and frequency of real world VRPs they need to solve.

Python Algorithms

Vehicle routing problem (VRP)

There are three main different approaches to modelling the VRP

- Vehicle flow formulations—this uses integer variables associated with each arc that count the number of times that the edge is traversed by a vehicle. This is good for cases where the solution cost can be expressed as the sum of any costs associated with the arcs. However it can't be used to handle many practical applications.
- Commodity flow formulations—additional integer variables are associated with the arcs or edges which represent the flow of commodities along the paths travelled by the vehicles. This has recently been used to find an exact solution.
- Set partitioning problem—These have an exponential number of binary variables which are each associated with a different feasible circuit. The VRP is then instead formulated as a set partitioning problem which asks what is the collection of circuits with minimum cost that satisfy the VRP constraints. This allows for very general route costs.

$$\min \sum_{i\in V}\sum_{j\in V} c_{ij}x_{ij}$$

subject to

$$\sum_{i\in V} x_{ij} = 1 \quad \forall j \in V \backslash \{0\}$$

$$\sum_{j\in V} x_{ij} = 1 \quad \forall i \in V \backslash \{0\}$$

$$\sum_{i\in V\backslash\{0\}} x_{i0} = K$$

$$\sum_{j\in V\backslash\{0\}} x_{0j} = K$$

$$\sum_{i\notin S}\sum_{j\in S} x_{ij} \geq r(S), \ \ \forall S \subseteq V \setminus \{0\}, S \neq \emptyset$$

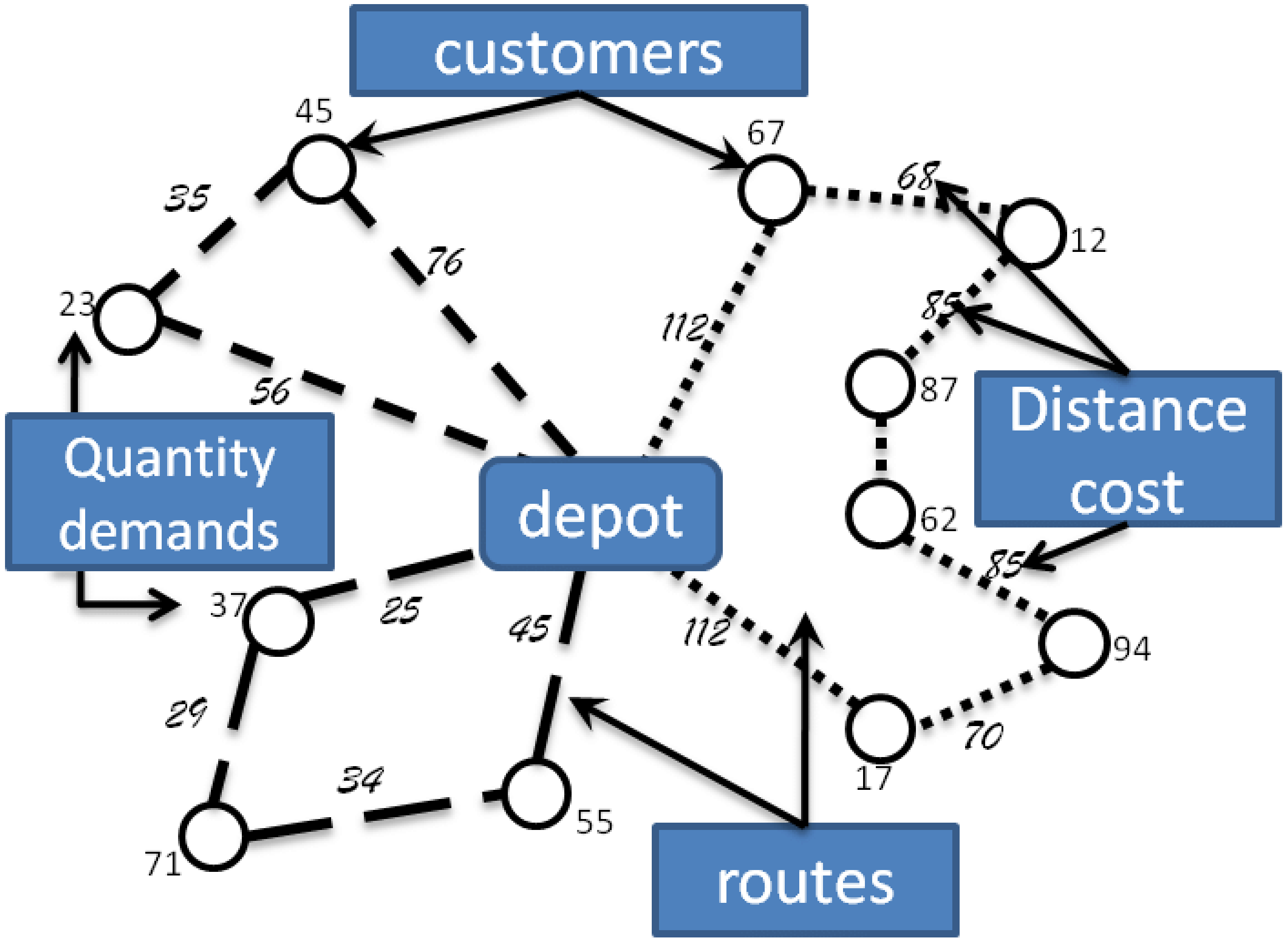
$$x_{ij} \in \{0,1\} \quad \forall i,j \in V$$

c_{ij} represents the cost of going from node i to node j,
x_{ij} is a binary variable that has value 1 if the arc going from i to j is considered as part of the solution and 0 otherwise,
K is the number of available vehicles and r(S) corresponds to the minimum number of vehicles needed to serve set S.
We are also assuming that 0 is the depot node.

Python Algorithms

❑ **Vehicle routing problem (VRP)**

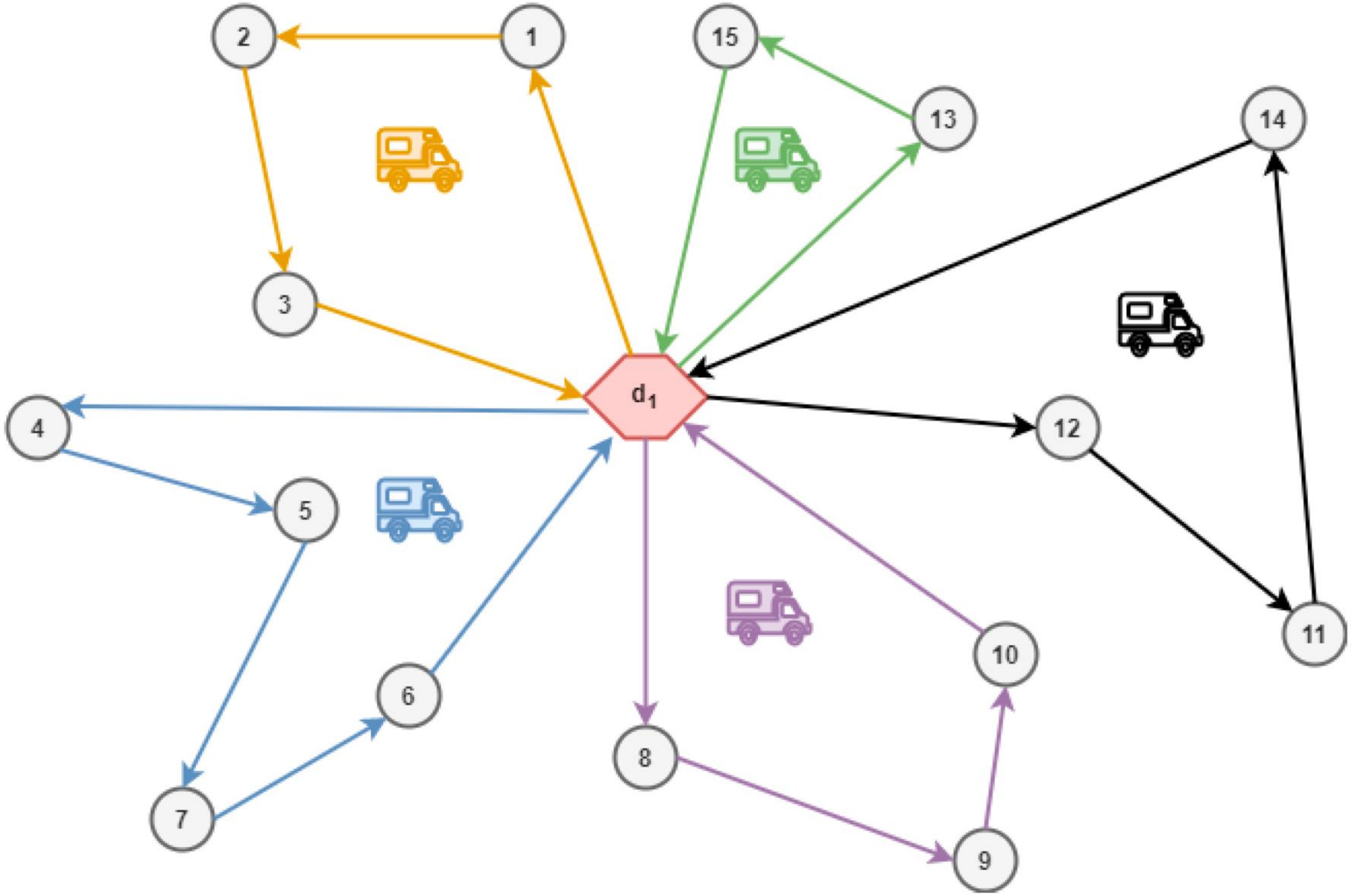
Example



Python Algorithms

❑ **Vehicle routing problem (VRP)**

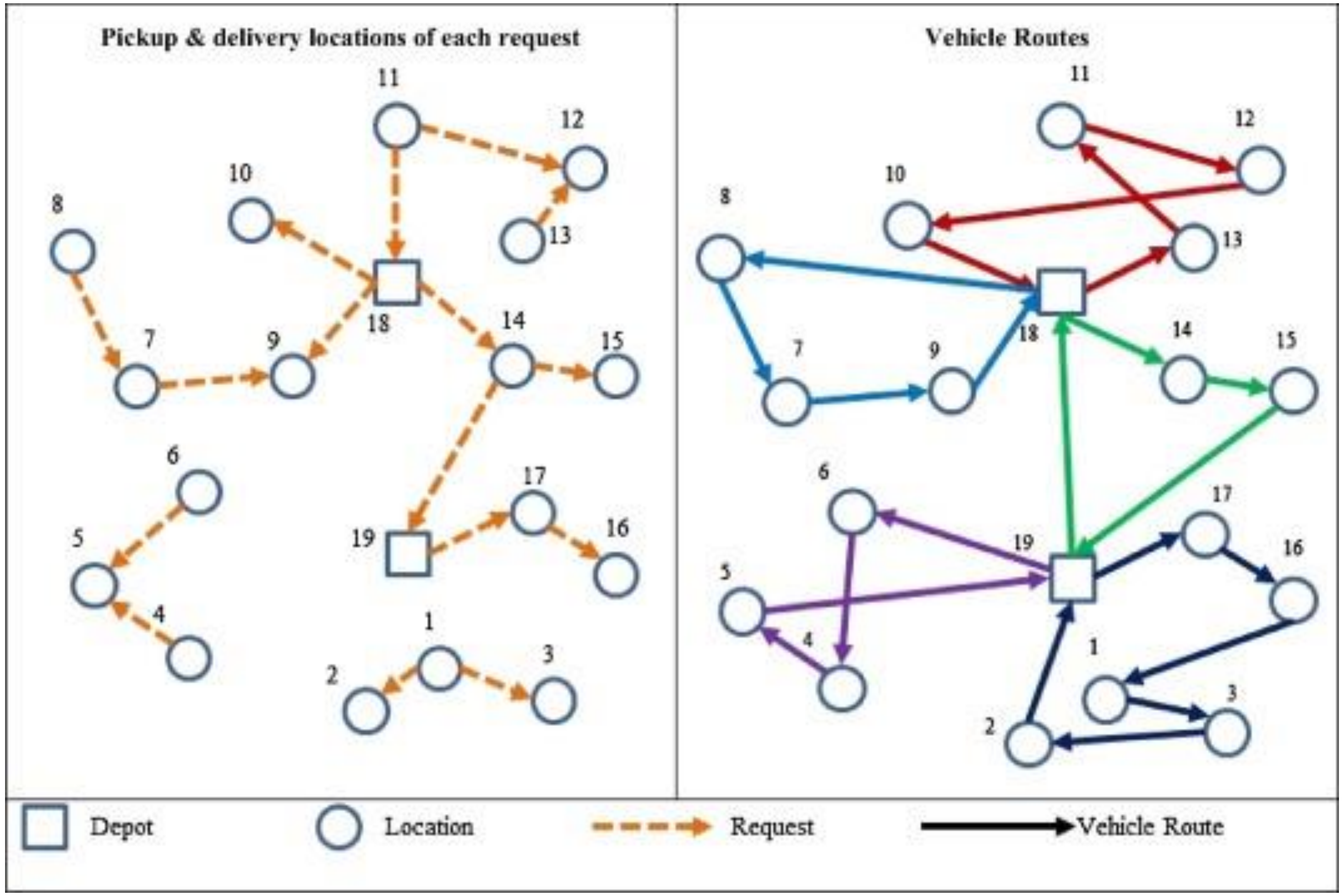
Example



Python Algorithms

❑ **Vehicle routing problem (VRP)**

Example

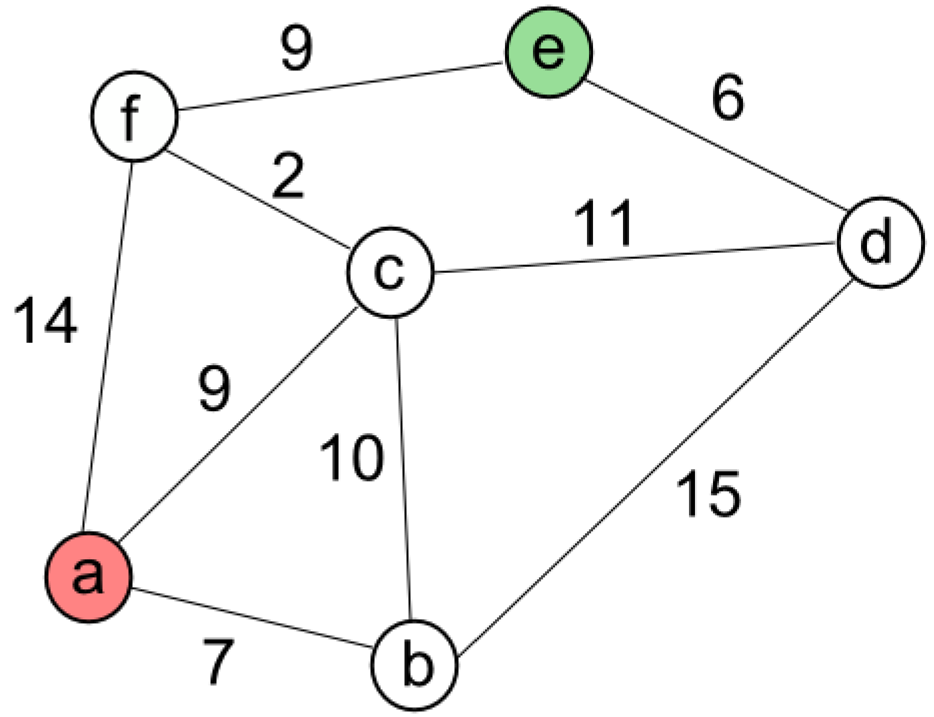


Python Algorithms

- **Vehicle routing problem (VRP)**

Testbed:

find the path with the smallest total weight among the possible paths we can take



Python Algorithms

- **Vehicle routing problem (VRP): Shortest path algorithm**

Our algorithm starts by defining a list of possible paths. A “path” is a list of connected nodes. Initially, we have only one “path” possible: [node1], because we start traversing the graph from that node. We also define a set of previously visited nodes to avoid backtracking.

- Initial steps the algorithm does the following:

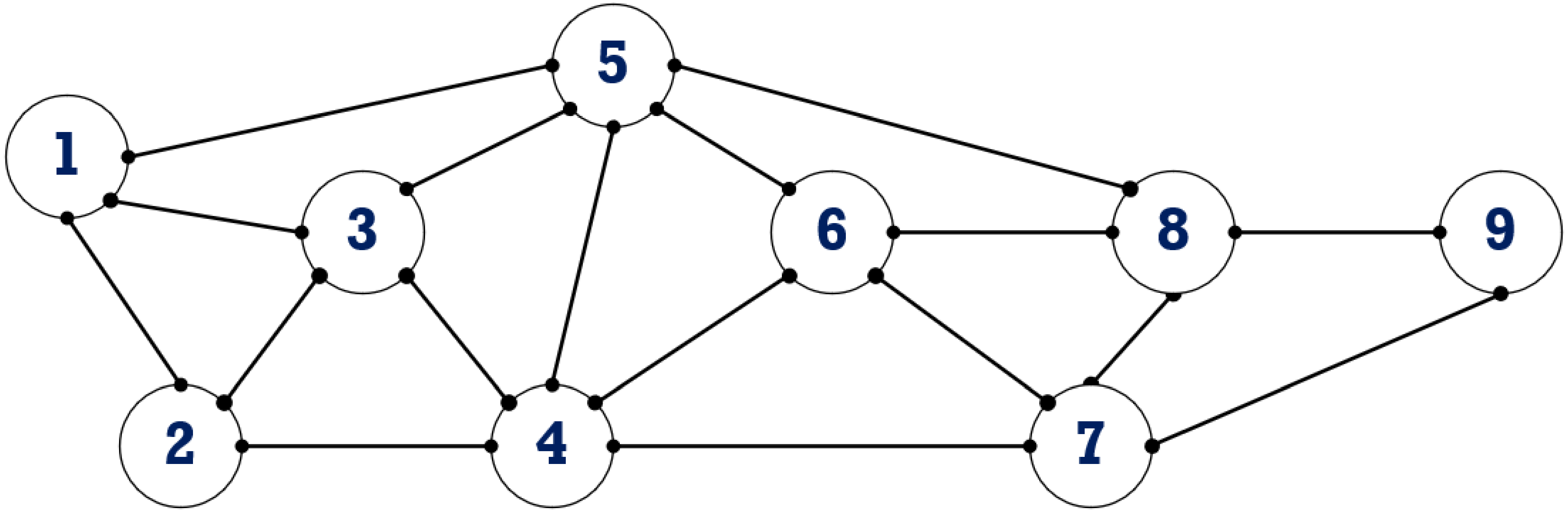
1. Take the next path from the list of paths.
   - If all possible paths have been traversed, stop. No path was found.
2. Set the “current node” to the last node in the current path.
3. Search the goal node, node2, between the nodes that are connected to the current node.
   - If node2 is connected to the current node, we have found path from node1 to node2. Stop.
4. If node2 isn’t connected to the current node, update the list of paths to traverse.
   - Add a new path from node1 to each one of the connected nodes to traverse next.
5. Go back to step 1.

Python Algorithms

❑ **Vehicle routing problem (VRP)**

Testbed:

find the path (1 → 9) with the smallest total weight among the possible paths we can take



A story of Alogrithm

Assignment Problem
(Hungarian Algorithm)

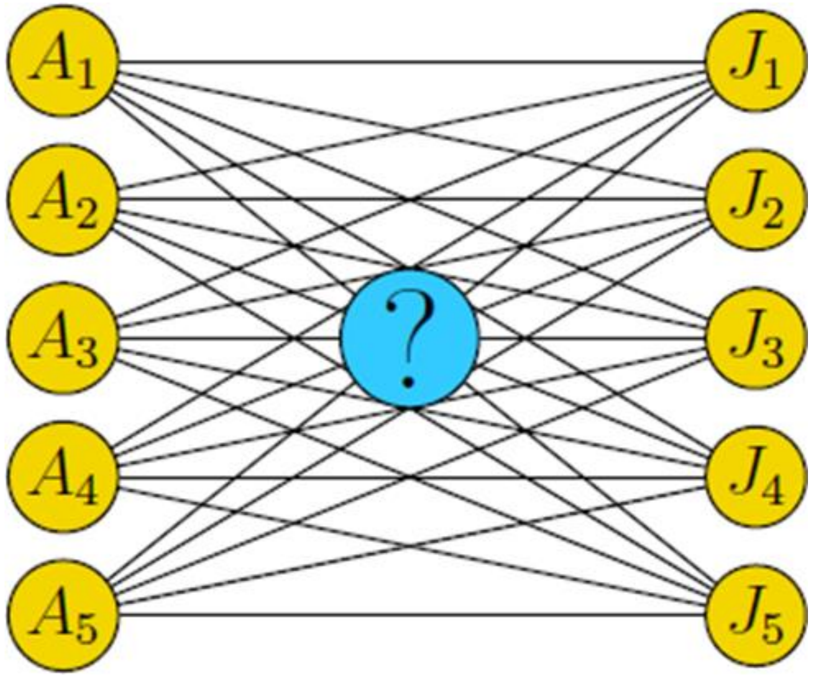
https://en.wikipedia.org/wiki/Assignment_problem

Python Algorithms

- **Assignment Problem (AP)**

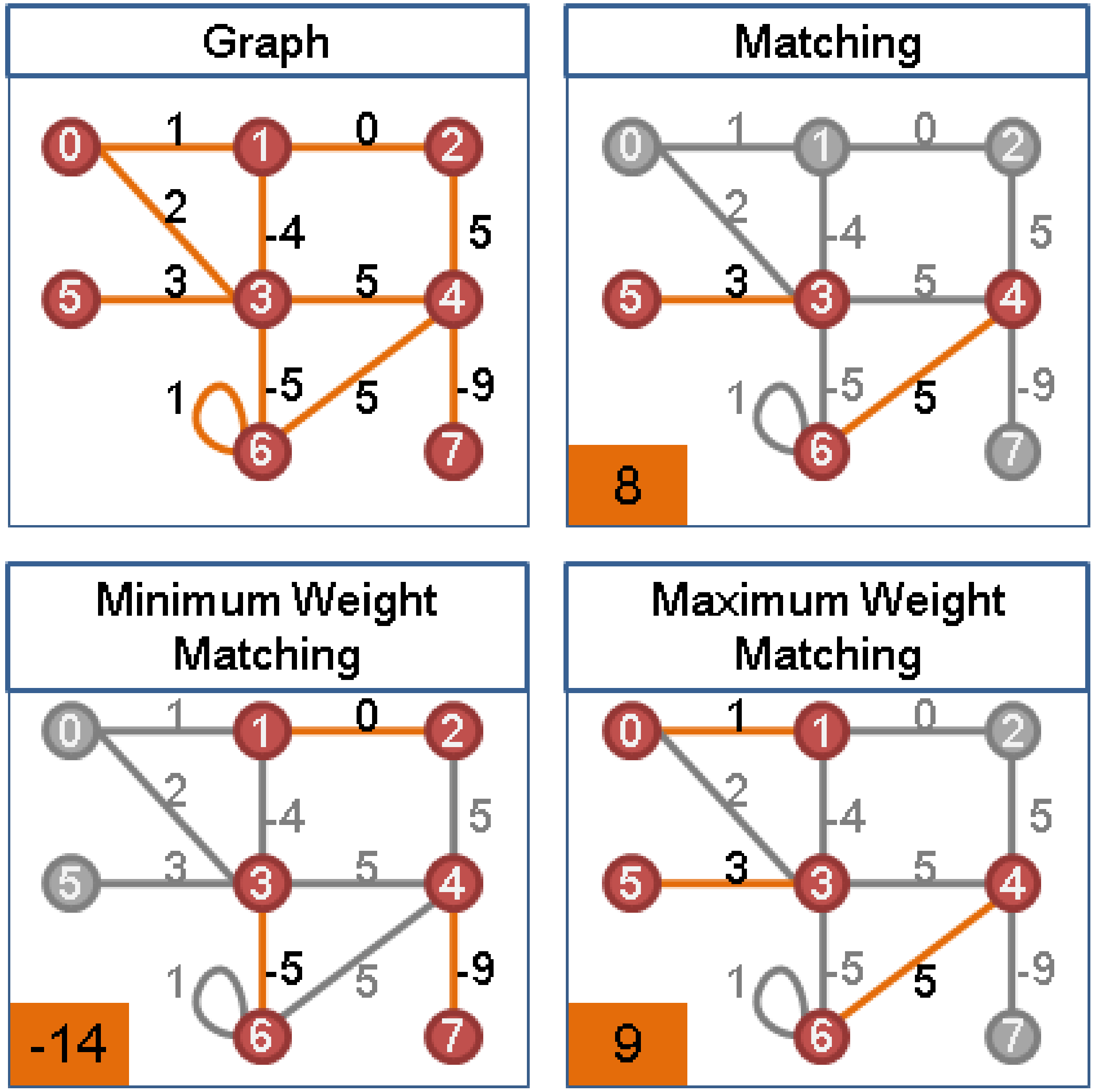
The assignment problem is a fundamental combinatorial optimization problem.
In its most general form, the problem is as follows:

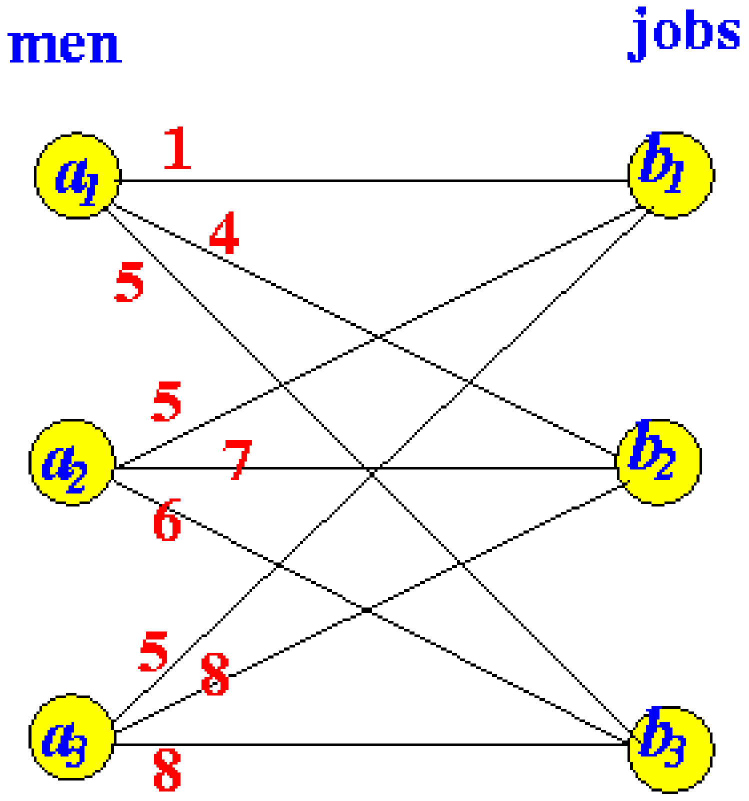
"The problem instance has a number of agents and a number of tasks. Any agent can be assigned to perform any task, incurring some cost that may vary depending on the agent-task assignment.
It is required to perform as many tasks as possible by assigning at most one agent to each task and at most one task to each agent, in such a way that the total cost of the assignment is minimized."



Python Algorithms

❑ Assignment Problem (AP)





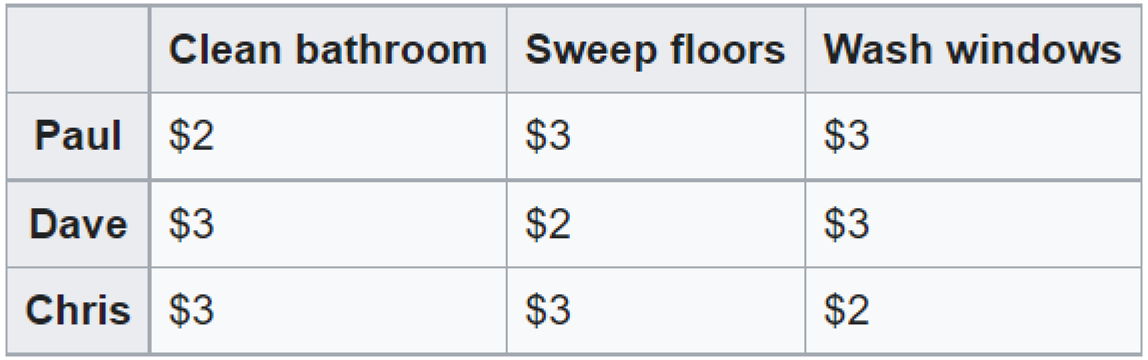
Python Algorithms

❑ Method of Assignment Problem (AP): Hungarian Algorithm

In this simple example there are three workers: Paul, Dave, and Chris.

One of them has to clean the bathroom, another sweep the floors and the third washes the windows, but they each demand different pay for the various tasks.

The problem is to find the lowest-cost way to assign the jobs? The problem can be represented in a matrix of the costs of the workers doing the jobs.



	Clean bathroom	Sweep floors	Wash windows
Paul	\$2	\$3	\$3
Dave	\$3	\$2	\$3
Chris	\$3	\$3	\$2

Python Algorithms

❑ Method of Assignment Problem (AP): Hungarian Algorithm

The Hungarian method is a combinatorial optimization algorithm that solves the assignment problem in polynomial time and which anticipated later primal–dual methods.

It was developed and published in 1955 by Harold Kuhn, who gave the name "Hungarian method" because the algorithm was largely based on the earlier works of two Hungarian mathematicians: Dénes Kőnig and Jenő Egerváry.

Jobs

$$\begin{array}{cc|cccccccccc|}
 & & 1 & 2 & 3 & & j & & n \\
\hline
 & 1 & C_{11} & C_{12} & C_{13} & \cdots\cdots & C_{ij} & \cdots & C_{1n} \\
 & 2 & C_{21} & C_{22} & C_{23} & \cdots\cdots & C_{2j} & \cdots & C_{2n} \\
 & 3 & C_{31} & C_{32} & C_{33} & \cdots\cdots & C_{3j} & \cdots & C_{3n} \\
\text{Persons} & \cdot & & & & & & & \\
 & i & C_{11} & C_{12} & C_{13} & \cdots\cdots & C_{ij} & \cdots & C_{in} \\
 & n & C_{n1} & C_{n2} & C_{n3} & \cdots\cdots & C_{nj} & \cdots & C_{nn}
\end{array}$$

Mathematically an assignment problem can be stated as follows

Minimise the total cost

$$Z = \sum_{i=i}^{n} \sum_{j=1}^{n} \; c_{ij} \, . \, x_{ij}$$

where

$$x_{ij} = \begin{array}{l} 1, \text{ if } i^{\text{th}} \text{ person is assgned to the } j^{\text{th}} \text{ job} \\ 0, \text{ if } i^{\text{th}} \text{ person is that assigned to the } j^{\text{th}} \text{ job} \end{array}$$

subject to the constraints

(i) $\sum_{i=1}^{n} x_{ij} = 1, j = 1,2 \text{ --- } n$

which means that only one job is done by the i-th person, i =1,2 ---*n*

(ii) $\sum_{j=1}^{n} x_{ij} = 1, i = 1,2\, n$

which means that only one person should be assigned to the j^{th} job, j = 1,2 --- *n*

Python Algorithms

Method of Assignment Problem (AP): Hungarian Algorithm

- **Step 1**: Subtract row minima

For each row, find the lowest element and subtract it from each element in that row.

- **Step 2**: Subtract column minima

Similarly, for each column, find the lowest element and subtract it from each element in that column.

- **Step 3**: Cover all zeros with a minimum number of lines

Cover all zeros in the resulting matrix using a minimum number of horizontal and vertical lines. If n lines are required, an optimal assignment exists among the zeros. The algorithm stops.

If less than n lines are required, continue with Step 4.

- **Step 4**: Create additional zeros

Find the smallest element (call it k) that is not covered by a line in Step 3. Subtract k from all uncovered elements, and add k to all elements that are covered twice.

Python Algorithms

❑ Method of Assignment Problem (AP): Hungarian Algorithm

We consider an example where four jobs (J1, J2, J3, and J4) need to be executed by four workers (W1, W2, W3, and W4), **one job per worker**.

The matrix below shows the cost of assigning a certain worker to a certain job.

The objective is to minimize the total cost of the assignment.

	J1	J2	J3	J4
W1	82	83	69	92
W2	77	37	49	92
W3	11	69	5	86
W4	8	9	98	23

Step 1: Subtract row minima

Step 2: Subtract column minima

Step 3: Cover all zeros with a minimum number of lines

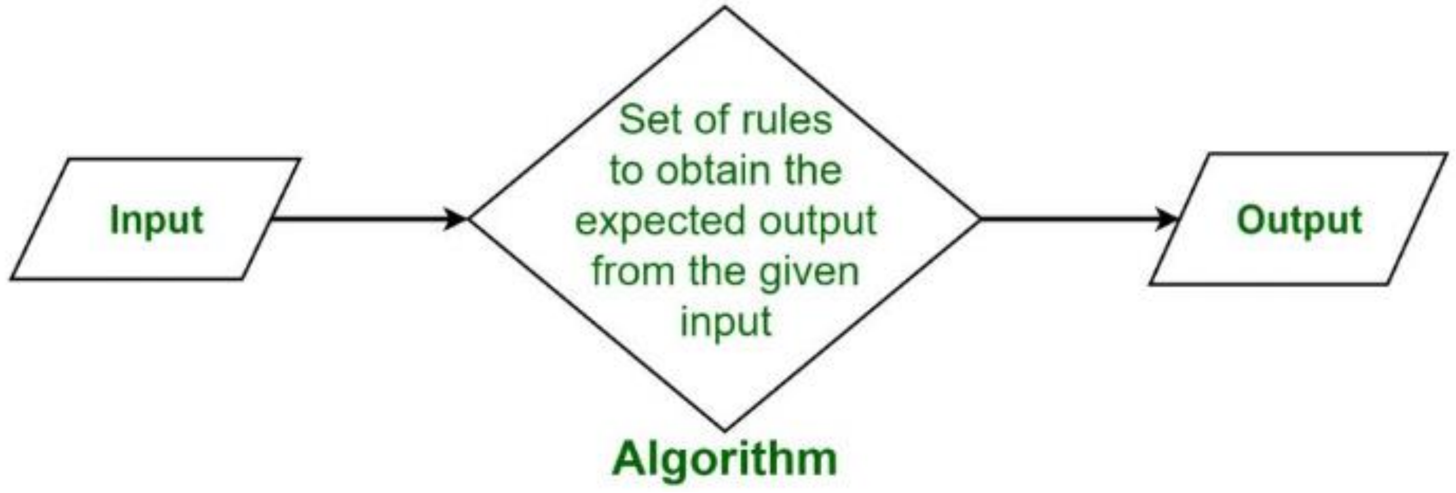
Step 4: Create additional zeros

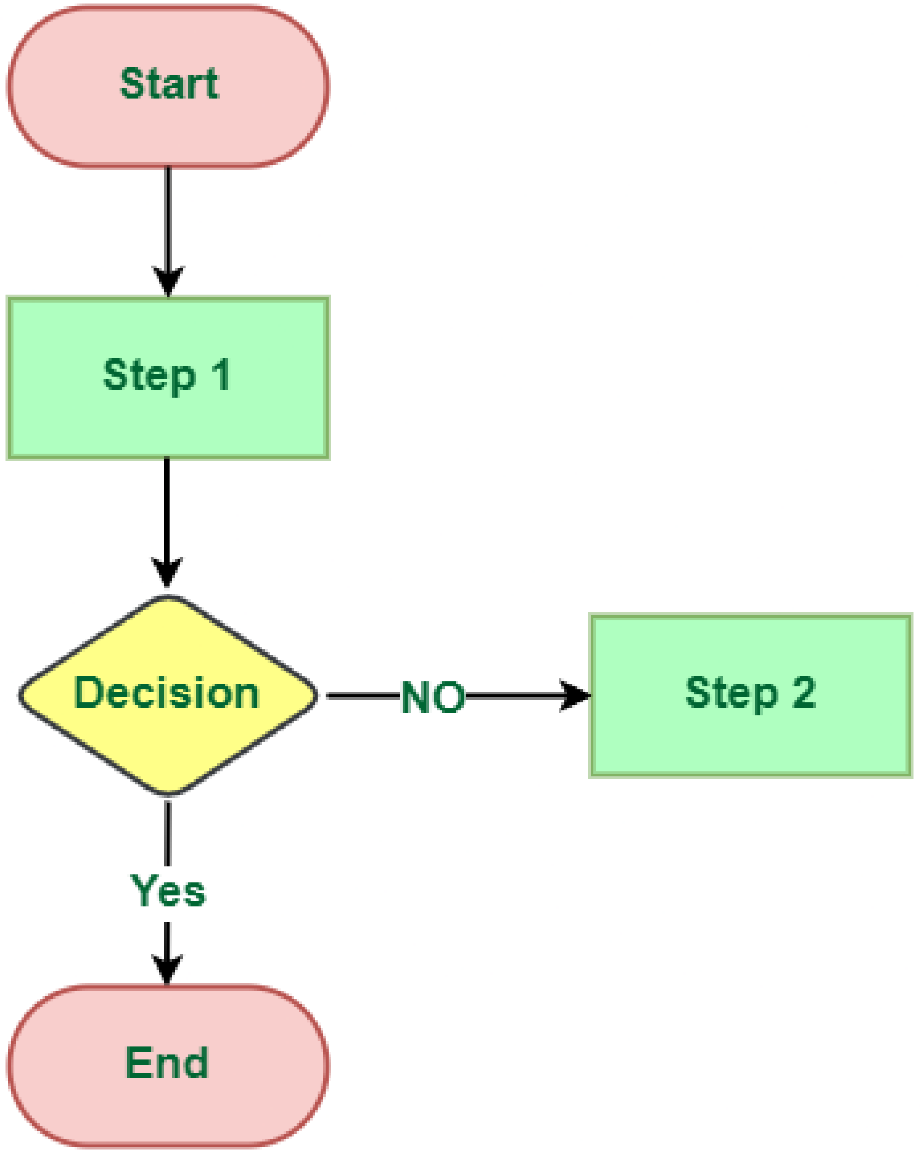
Practices

Algorithm Designs in Application Programming

Practices: Algorithms

Design of Algorithm in Python Programming

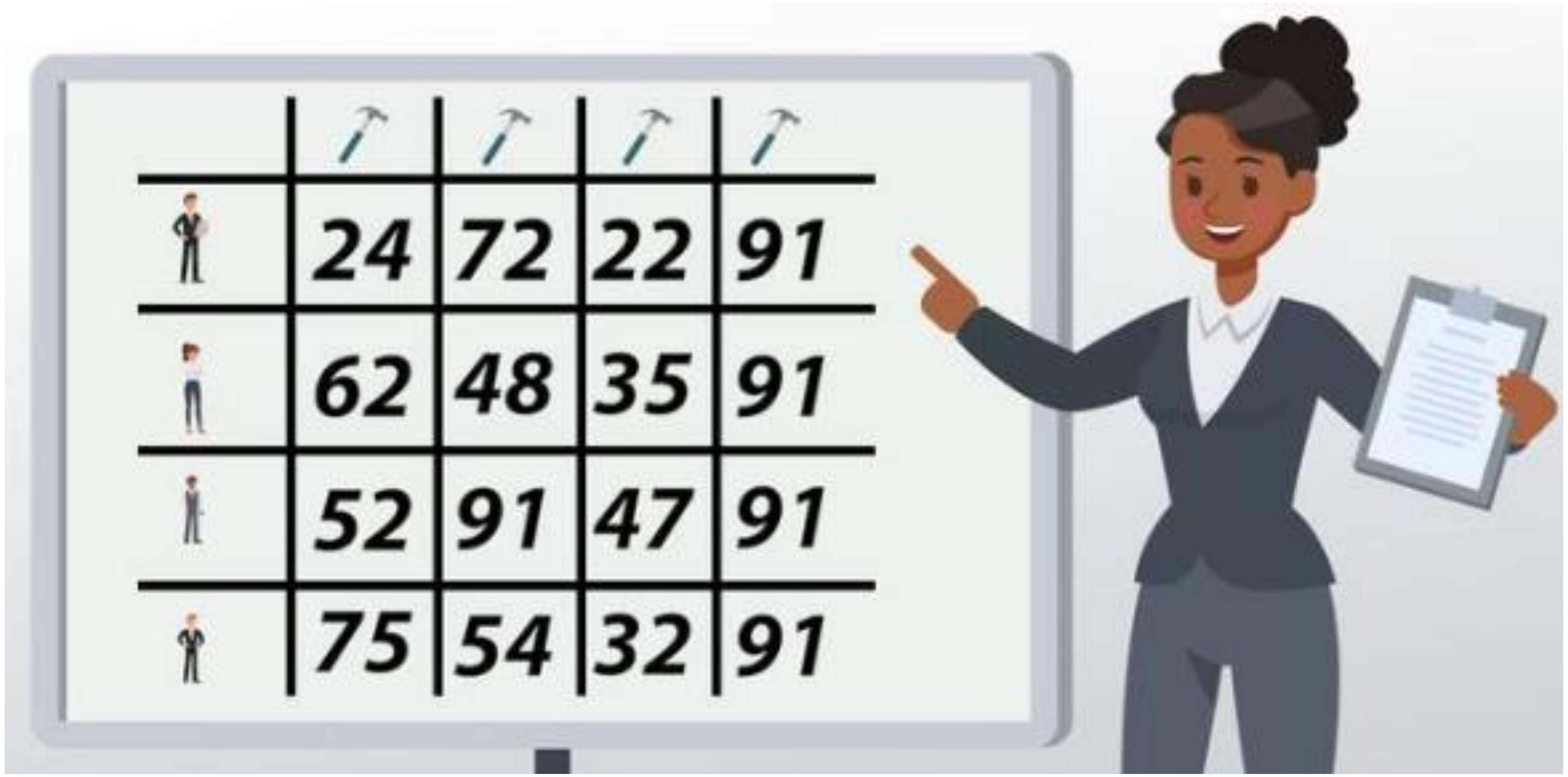

Input
Set of rules to obtain the expected output from the given input
Output
Algorithm



Practices 03

❑ **Assignment problem:**

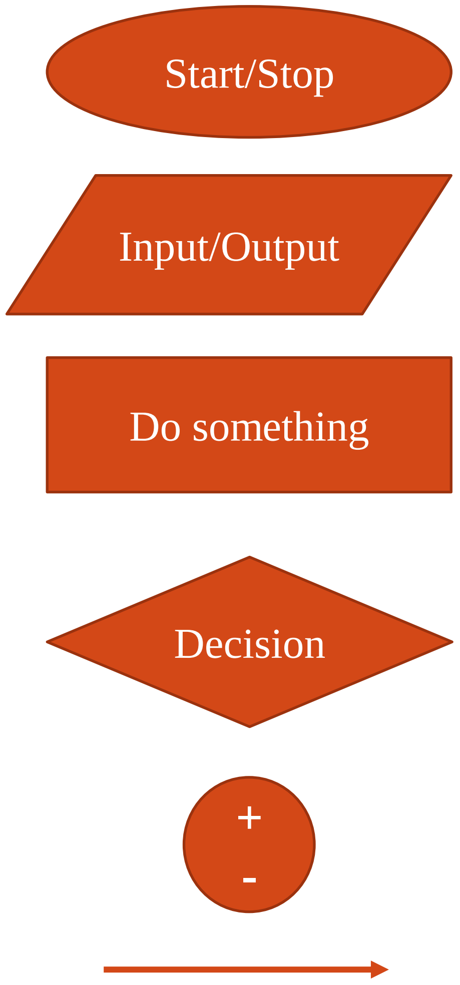
Using Hungarian Algorithm → find matching “Person-Job” with [MIN. Cost]



24	72	22	91
62	48	35	91
52	91	47	91
75	54	32	91

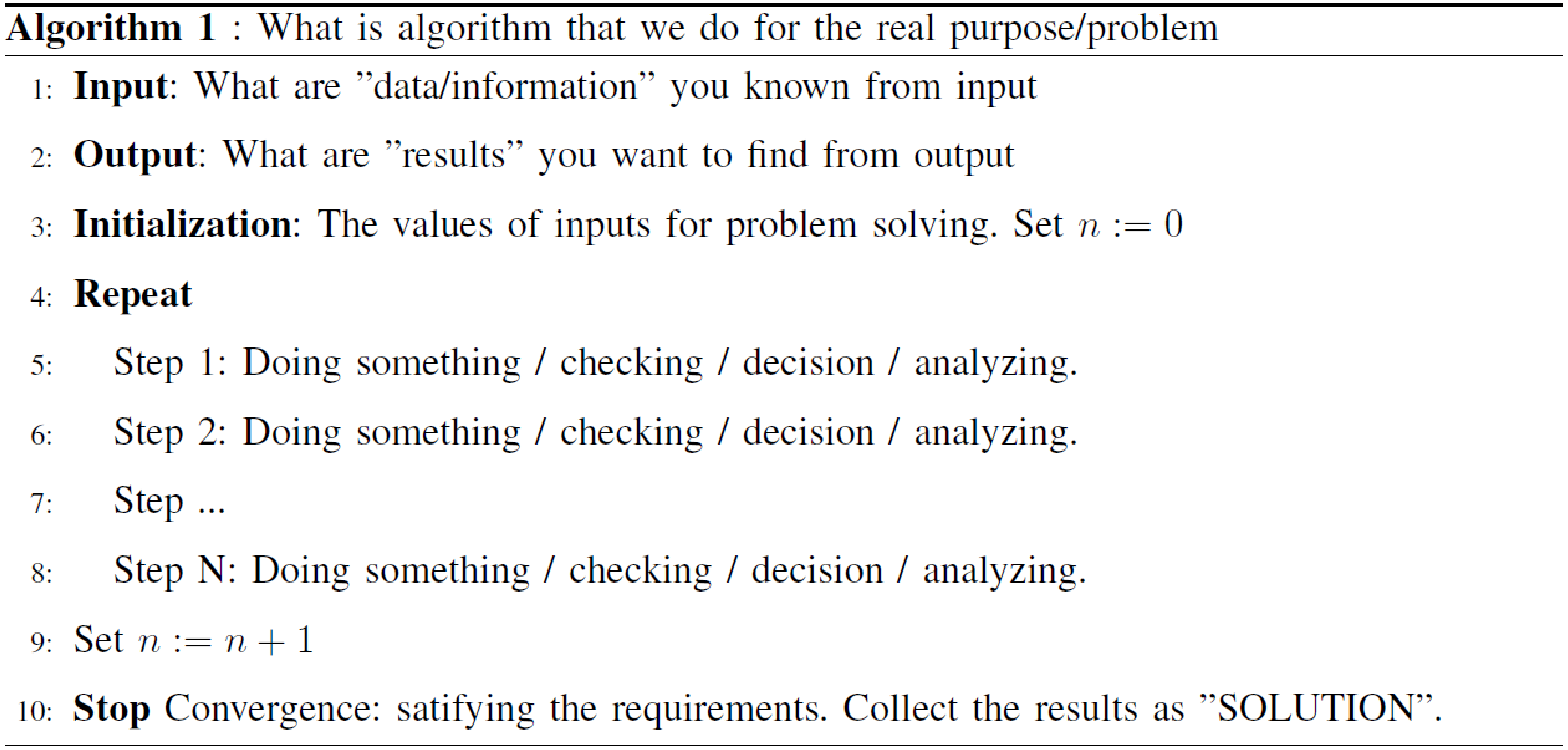
Practices 03

- ❑ Design an Algorithm (steps) để giải quyết “Assignment problem”.
- ❑ Giải quyết vấn đề → tìm ra solution (bằng tay)
- ❑ Viết code lập trình giải thuật: (đánh giá theo từng mức)
  - ✓ Function chứa giải thuật (*Hungarian algorithm*) cho bài toán “Assignment”
  - ✓ Thể hiện rõ ràng các bước thực hiện giải thuật *Hungarian algorithm*
  - ✓ Viết code giải bài toán Assignment problem đã đề cập (free code)
  - ✓ Code có thể mở rộng với bất kỳ giá trị “input” phù hợp
  - ✓ Testbed và tính toán thời gian thực thi giải thuật với các “input size” khác nhau



Practices: Algorithms

Design of an Algorithm in Python Programming (steps)



Algorithm 1 : What is algorithm that we do for the real purpose/problem

```
Input: What are ”data/information” you known from input
Output: What are ”results” you want to find from output
Initialization: The values of inputs for problem solving. Set $n := 0$
Repeat
    Step 1: Doing something / checking / decision / analyzing.
    Step 2: Doing something / checking / decision / analyzing.
    Step ...
    Step N: Doing something / checking / decision / analyzing.
Set $n := n + 1$
Stop Convergence: satifying the requirements. Collect the results as ”SOLUTION”.
```