

# Data Structures and Algorithms

P and NP

# Introduction to Turing Machine

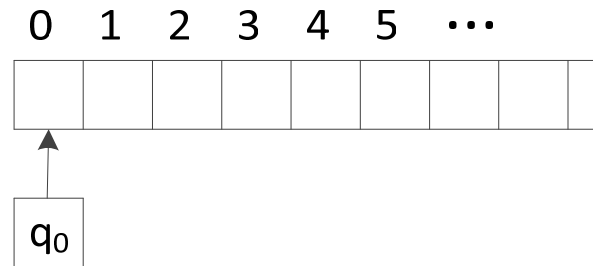
- Introduced by Alan Turing.
- A Turing machine is a formal computational model of finite-state computing machines.
- Has unlimited amount of time and memory available for computation.
- A Turing machine is a finite-state machine in which a transition reads and prints a symbol on the tape.
- A tape head may move in either direction.
- We will briefly discuss standard Turing machines.

# Introduction to Turing Machine

- Definition: A Turing machine is a quintuple  $M = (Q, \Sigma, \Gamma, \delta, q_0)$ , where
- $Q$  is a finite set of states.
- $\Gamma$  is a finite set called the *tape alphabet*.  $\Gamma$  contains a special symbol  $B$  that represents a blank.
- $\Sigma$  is a subset of  $\Gamma - \{B\}$  called the *input alphabet*.
- $\delta$  is a partial function  $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ .
- $q_0 \in Q$  is a distinguished state called the *start state*.

# Introduction to Turing Machine

- A tape extends indefinitely in one direction.
- Tape positions are numbered beginning with zero.
- A computation begins with the tape head in state  $q_0$  scanning the leftmost position.



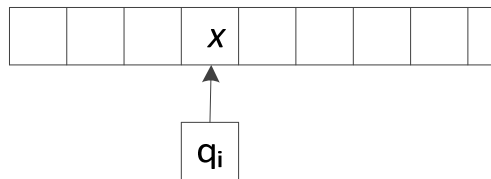
- Input string from  $\Sigma^*$  is written on the tape beginning at position one. Position zero and all other positions are blanks.

# Introduction to Turing Machine

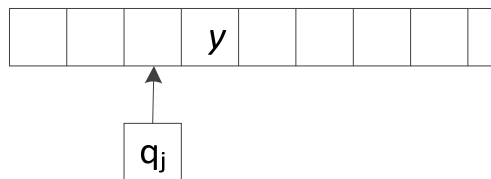
- A transition consists of three actions:
  - Change the state.
  - Write a symbol on the square scanned by the tape head.
  - Move the tap head. Direction of the move is indicated by  $L$  (left) or  $R$  (right).

# Introduction to Turing Machine

- If the machine configuration is



and the transition is  $\delta(q_i, x) = [q_j, y, L]$ , then the new configuration is



- Here, the transition changed from  $q_i$  to  $q_j$ , tape symbol  $y$  was written replacing  $x$ , and the tape head was moved to the left by one position.

# Introduction to Turing Machine

- A Turing machine *halts* when it encounters a state symbol pair for which no transition is defined.
- A transition from tape position zero may specify a move to the left (crossing the boundary of the tape). When this occurs, we say the computation *terminates abnormally*.
- When we say a computation *halts*, it means it terminates in a normal fashion.

# Introduction to Turing Machine

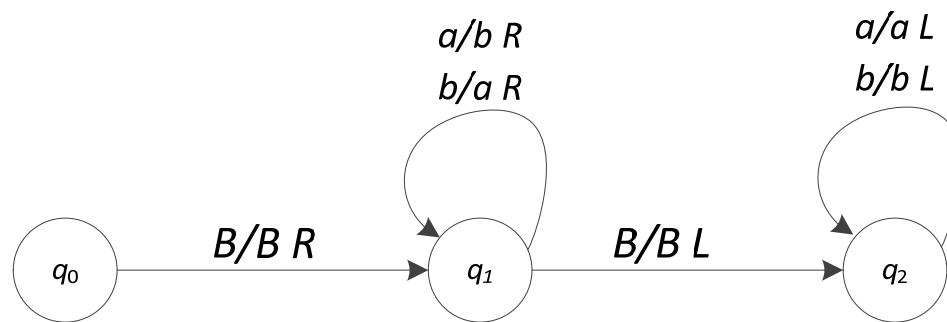
- Example: A transition function of a standard Turing machine with input alphabet  $\{a, b\}$ :

$\delta$	B	a	b
$q_0$	$q_1, B, R$		
$q_1$	$q_2, B, L$	$q_1, b, R$	$q_1, a, R$
$q_2$		$q_2, a, L$	$q_2, b, L$



# Introduction to Turing Machine

- A Turing machine can be represented as a state diagram. In a state diagram, the transition  $\delta(q_i, x) = [q_j, y, d]$ ,  $d \in \{L, R\}$ , is represented by an edge from  $q_i$  to  $q_j$  labeled  $x/y d$ . The state diagram corresponding to the above transition function is:



# Introduction to Turing Machine

- A machine configuration consists of the state, the tape, and the position of the tape head.
- A configuration is denoted by  $uq_ivB$ , where  $uv$  is the string spelled on the tape from the left boundary to the rightmost nonblank symbol.
- The notation  $uq_ivB$  indicates the machine is in state  $q_i$  scanning the first symbol of  $v$ .
- The notation  $uq_ivB \vdash xq_jyB$  indicates the configuration  $xq_jyB$  is obtained from  $uq_ivB$  by a single transition. Here,  $u$ ,  $v$ ,  $x$ , and  $y$  are strings.
- The notation  $uq_ivB \vdash^* xq_jyB$  represents that  $xq_jyB$  can be obtained from  $uq_ivB$  by a finite number of, possibly zero, transitions.

# Introduction to Turing Machine

- The following sequence of configurations, or transitions, show the computation generated by tracing the input *abab* by the above Turing machine:

$$\begin{aligned} q_0BababB \vdash Bq_1ababB \vdash Bbq_1babB \vdash Bbaq_1abB \\ \vdash Bbabq_1bB \vdash Bbabaq_1B \vdash Bbabq_2aB \vdash Bbaq_2baB \\ \vdash Bbq_2abaB \vdash Bq_2babaB \vdash q_2BbabaB \end{aligned}$$

- This Turing machine exchanges *a*'s and *b*'s in the input string.

# Introduction to Turing Machine

- Turing machines can be used as language acceptors.
- A computation accepts or rejects the input string.
- A Turing machine is augmented with final states.
- A Turing machine need not read the entire input to accept the string.

# Introduction to Turing Machine

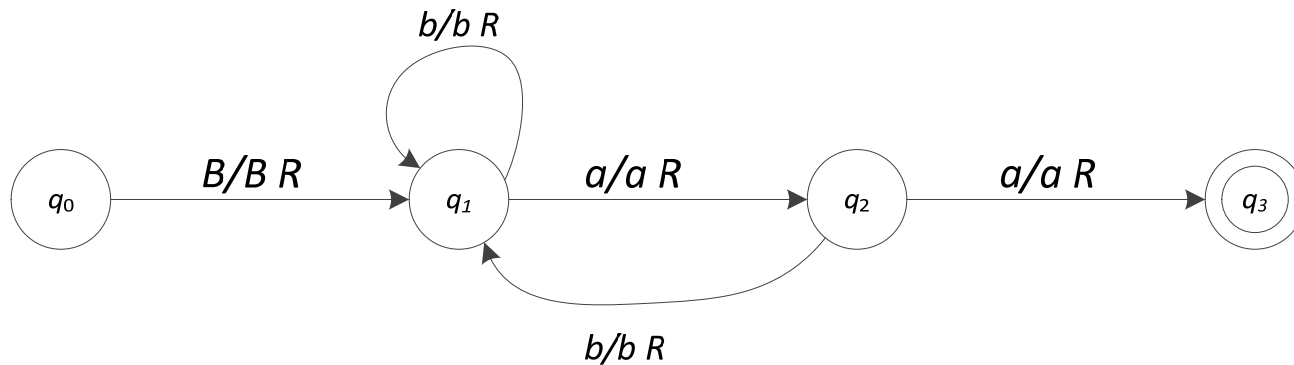
- Definition: Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  be a Turing machine. A string  $u \in \Sigma^*$  is accepted by final state if the computation of  $M$  with input  $u$  halts in a final state. A computation that terminates abnormally rejects the input. The language of  $M$ ,  $L(M)$ , is the set of all languages accepted by  $M$ .

# Introduction to Turing Machine

- A language accepted by a Turing machine is called a *recursively enumerable language*.
- If the Turing machine halts for all input string of a language, the language is said to be *recursive*.
- The computations of a Turing machine provide a decision procedure for membership in a recursive language.

# Introduction to Turing Machine

- Example: The following Turing machine accepts the language  $(a \cup b)^*aa(a \cup b)^*$ .



# Introduction to Turing Machine

- The computation on the input  $aabb$  is:

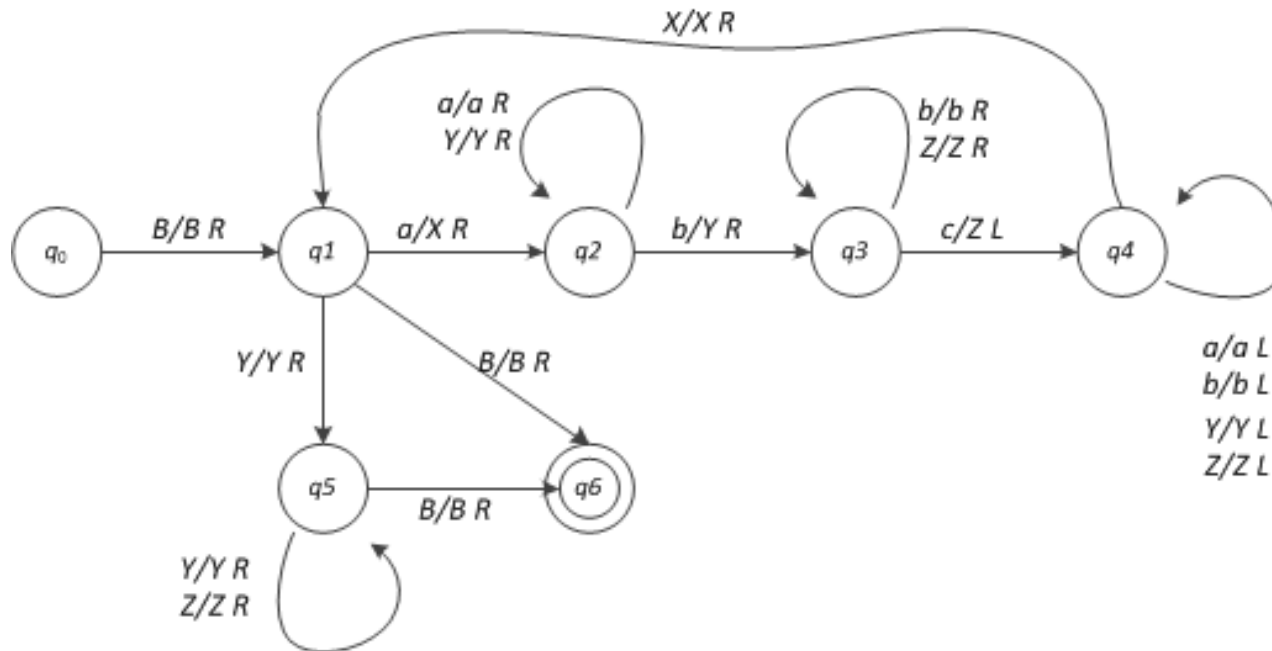
$$q_0BaabbB$$
$$\vdash Bq_1aabbB$$
$$\vdash Baq_2abbB$$
$$\vdash Baaq_3bbB$$

- Note that only the first half of the input string is examined before accepting it.



# Introduction to Turing Machine

- Exercise: Construct a Turing machine that accepts the language  $\{a^i b^j c^i \mid i \geq 0\}$ .



# Decision Problem

- Decision problem: A decision problem **P** is a set of questions each of which has a yes or no answer.
- Example: A decision problem **P<sub>sq</sub>**: Determine whether an arbitrary number is a perfect square or not. This problem consists of the following questions:

**p<sub>0</sub>**: Is 0 a perfect square?

**p<sub>1</sub>**: Is 1 a perfect square?

...

Here, **p<sub>i</sub>** is also called an instance of **P**.

# Decision Problem

- A solution to a decision problem is an algorithm that determines the answer to every question  $\mathbf{p}_i \in \mathbf{P}$ .
- An algorithm that solves a decision problem should be
  - *complete* – it produces an answer, either positive or negative, to each question in the problem domain
  - *mechanistic* – it consists of a finite sequence of instructions each of which can be carried out without requiring insight, ingenuity, or guesswork
  - *deterministic* – when presented with identical input, it always produces the same result.

# Decision Problem

- Decision problems:
  - *Unsolvable* (or *undecidable*)
  - *Solvable*:
    - *Tractable*: A decision problem is said to be tractable if there is at least one polynomially bounded algorithm that solves the problem. Such an algorithm is called an *efficient* algorithm.
    - *Intractable*: A decision problem is said to be intractable if there is no polynomially bounded algorithm (or no efficient algorithm) that solves the problem

# Decision Problem

- Two examples of unsolvable problems: *the halting problem for Turing machines* and *the post correspondence problem*.
- The halting problem for Turing machines: Given an arbitrary Turing machine  $M$  with an input alphabet  $\Sigma$  and a string  $w \in \Sigma^*$ , will the computation of  $M$  with input  $w$  halt?

# Decision Problem

- Note this problem is different from determining whether a particular Turing machine will halt for a given string.
- This problem requires a general algorithm that answers the halting question for every possible combination of Turing machine and input string.
- Theorem: The halting problem for Turing machines is undecidable.

# Decision Problem

- Post correspondence problem: Instead of formally stating the problem, we will illustrate the problem as a simple game of manipulating dominoes.
- A domino consists of two strings from a fixed alphabet, one on the top half of the domino and the other on the bottom.

<i>aba</i>
<i>bbaba</i>

# Decision Problem

- We are given a finite set of different types of dominoes.
- We assume that there are an unlimited number of each type of dominoes.
- The game begins when a domino is placed on a table. Another domino is placed to the immediate right of the domino. This process is repeated making a sequence of dominoes on the table.



# Decision Problem

- The *top string* is obtained by concatenating the strings in the top halves of the sequence of dominoes.
- The *bottom string* is obtained by concatenating the strings in the bottom halves of the sequence of dominoes.
- The goal of the game (or the solution to a Post correspondence problem) is to come up with a sequence of dominoes where the top string is identical to the bottom string.

# Decision Problem

- Example 1. Given the following two dominoes:

<i>aaa</i>	<i>baa</i>
<i>aa</i>	<i>abaaa</i>

The following sequence of dominoes is a solution:

<i>aaa</i>	<i>baa</i>	<i>aaa</i>
<i>aa</i>	<i>abaaa</i>	<i>aa</i>

# Decision Problem

- Example 2. Given the following three dominoes:

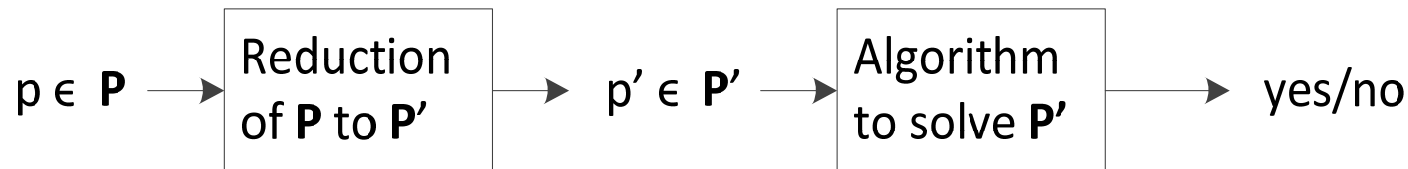
<i>ab</i>	<i>bba</i>	<i>aba</i>
<i>aba</i>	<i>aa</i>	<i>bab</i>

There is no solution.

- Theorem: There is no algorithm that determines whether an arbitrary finite set of dominoes has a solution.
- Since solvable problems are equivalent to recursive languages, *decision problems* and *languages* are used interchangeably.

# Reducibility

- A decision problem  $\mathbf{P}$  is Turing reducible to a problem  $\mathbf{P}'$  if there is a Turing machine that takes any problem  $\mathbf{p}_i \in \mathbf{P}$  as input and produces an associated problem  $\mathbf{p}'_i \in \mathbf{P}'$  where the answer to the original problem  $\mathbf{p}_i$  can be obtained from the answer to  $\mathbf{p}'_i$ .



# P and NP

- A language  $L$  is decidable in polynomial time if there is a standard (or deterministic) Turing machine  $M$  that accepts  $L$  in polynomial time, or  $O(n^r)$ , where  $r$  is a natural number independent of  $n$ .
- The family of languages decidable in polynomial time is denoted  **$P$** .

# P and NP

- Nondeterministic computation:
  - A deterministic machine solves a decision problem by generating a solution.
  - A nondeterministic machine needs only determine if one of possibilities is a solution.
- A language  $L$  is said to be accepted in nondeterministic polynomial time if there is a nondeterministic Turing machine that accepts  $L$  in polynomial time, or  $O(n^r)$ , where  $r$  is a natural number independent of  $n$ .

# P and NP

- The family of languages accepted in nondeterministic polynomial time is denoted **NP**.
- Another definition: A problem is in **NP** if it is “verifiable” in polynomial time.
- What “verifiable” means is that given a possible solution (which is also called **certificate**) we can verify whether it is a solution or not in polynomial time.

# P and NP

- **$P = NP$  ?**
- Unsolved question.
- Since every deterministic machine is also nondeterministic,  **$P \subseteq NP$ .**
- But it was never proved that  **$NP \subseteq P$ .** (If this is proved, then that proves  **$P = NP$ .**)



# P and NP

- If  $Q$  is reducible to  $L$  in polynomial time and  $L \in \mathbf{P}$ , then  $Q \in \mathbf{P}$ .
- A language  $L$  is called ***NP-hard*** if for every  $Q \in \mathbf{NP}$   $Q$  is reducible to  $L$  in polynomial time.
- An ***NP-hard*** language that is also in  $\mathbf{NP}$  is called ***NP-complete***.
- If there is an NP-complete language that is also in  $\mathbf{P}$ , then  $\mathbf{P} = \mathbf{NP}$ .

# P and NP

- Two examples of NP-complete problems: Hamiltonian cycle problem and traveling salesman problem.

# Hamiltonian Cycle Problem

- A Hamiltonian cycle of an undirected graph  $G = (V, E)$  is a simple cycle that contains each vertex in  $V$ .
- Note: A cycle is simple if a node, except the first node, is visited only once.
- A graph that contains a Hamiltonian cycle is called “Hamiltonian.”
- ***Hamiltonian Cycle Problem:*** Does a graph  $G$  have a Hamiltonian cycle?

# Hamiltonian Cycle Problem

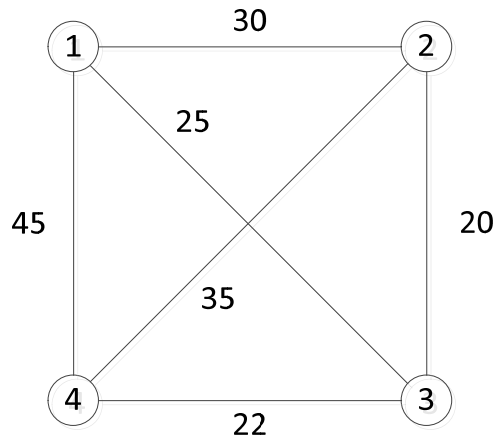
- It can be shown that the Hamiltonian cycle problem can be decidable by a Turing machine in *exponential* time, but not in *polynomial* time. This means Hamiltonian cycle problem is not in **P**.
- But, it is decidable in nondeterministic polynomial time.
- Given a cycle in a graph, we can determine whether it is Hamiltonian cycle or not in polynomial time.
- So, Hamiltonian cycle problem is in **NP**.
- In fact it is an **NP-complete** problem.

# Traveling Salesman Problem

- Given a complete, non-negative weighted graph, find a Hamiltonian cycle of minimum weight.
- This problem is ***NP-complete***.
- Will briefly discuss three approximate algorithms.

# Traveling Salesman Problem

- Consider the following graph:



minimum weight cycle =  $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$ .

total weight =  $30 + 35 + 22 + 25 = 112$

# Traveling Salesman Problem

- Nearest-neighbor strategy

NEAREST-TSP ( $G, f$ ) /\*  $f$  is a cost function, or a weight function \*/

select an arbitrary vertex  $s$ ;

$v = s$ ;  $Q = \{v\}$ ;  $S = G.V - Q$ ;  $C = \phi$ ;

**while**  $S \neq \phi$

select an edge  $(v, w)$  of minimum weight, where  $w \in S$ ;

$C = C \cup \{(v, w)\}$ ;

$Q = Q \cup \{w\}$ ;

$S = S - \{w\}$ ;

$v = w$ ;

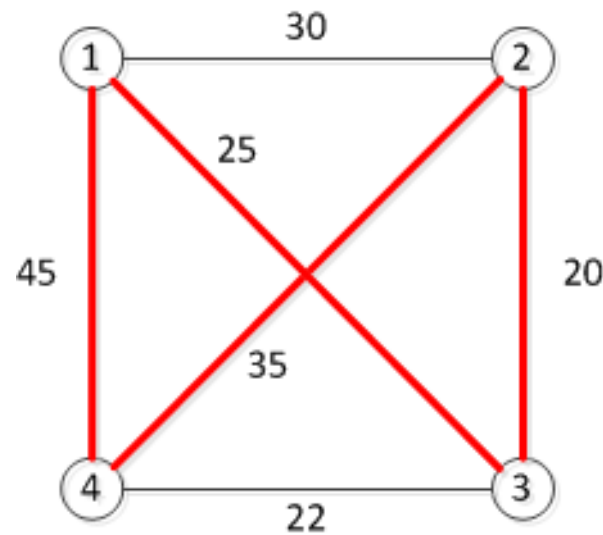
$C = C \cup \{(v, s)\}$ ;

**return**  $C$ ;

Running time: $O(V^2)$
------------------------

# Traveling Salesman Problem

- Nearest-neighbor strategy



Starting at vertex 1: (1, 3), (3, 2), (2, 4), (4, 1)

Total weight =  $25 + 20 + 35 + 45 = 125$



# Traveling Salesman Problem

- Shortest-link strategy

SHORTEST-LINK-TSP ( $G, f$ )

$R = G.E;$

$C = \phi;$

**while**  $R \neq \phi$

    choose the shortest edge  $(v, w)$  from  $R$ ;

$R = R - \{(v, w)\};$

**if**  $(v, w)$  does not make a cycle with edges in  $C$  and  $(v, w)$  would  
        not be the third edge in  $C$  incident on  $v$  or  $w$

**then**

$C = C + \{(v, w)\};$

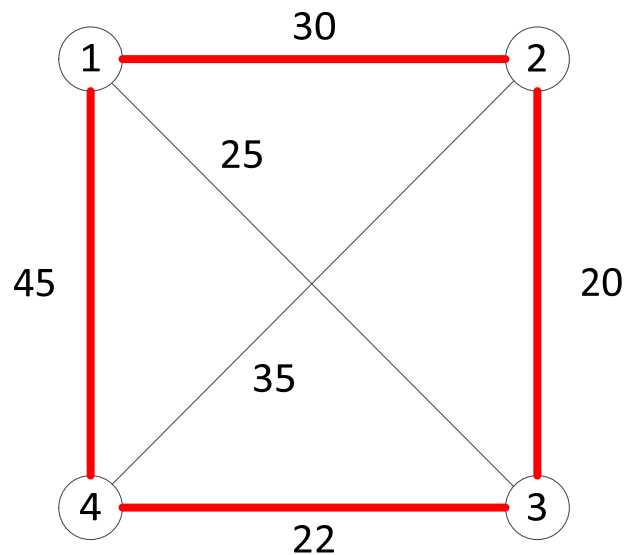
add the edge connecting the end points of the path in  $C$ ;

return  $C$ ;

Running time: $O(E \log V)$
-----------------------------

# Traveling Salesman Problem

- Shortest-link strategy



Edges added: (2, 3), (3, 4), (2, 1), (1, 4)

Total weight =  $20 + 22 + 30 + 45 = 117$

# Traveling Salesman Problem

- In general, we cannot establish a bound on how much the weight of an approximate algorithm differ from the weight of a minimum tour.
- If we assume the triangle inequality holds on distances among vertices, we can develop an approximate algorithm that has an upper bound on the weight.
- Triangle inequality:  
$$f(u, v) \leq f(u, w) + f(w, v), \text{ for all } u, v, w \in G.V.$$
- Euclidean distance has the triangle inequality property.

# Traveling Salesman Problem

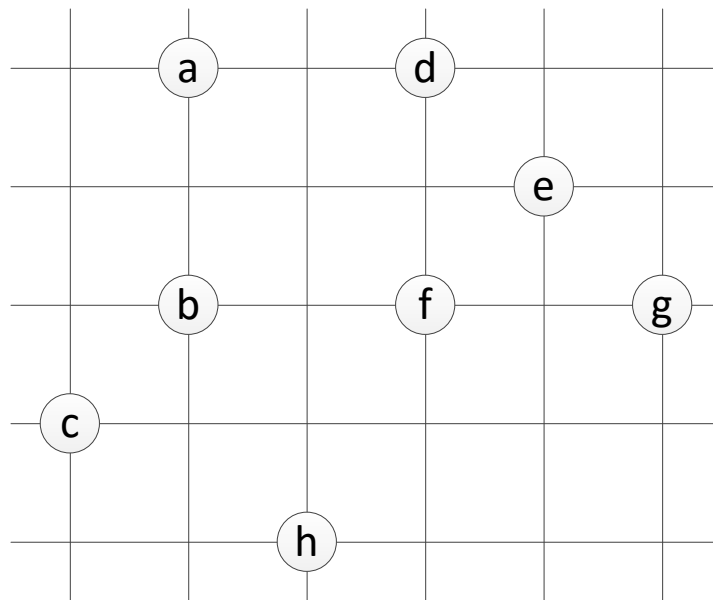
- The following approximate algorithm has an upper bound on the weight: total weight of a cycle is no more than the twice that of the minimum spanning tree's weight

APPROX-TSP-TOUR ( $G, f$ )

```
select a vertex  $r \in G.V$  to be the root;  
compute MST  $T$  from  $r$  using MST-PRIM( $G, f, r$ );  
let  $H$  be a list of vertices, ordered according to when they are  
    first visited in a preorder tree walk of  $T$ ;  
return  $H$ 
```

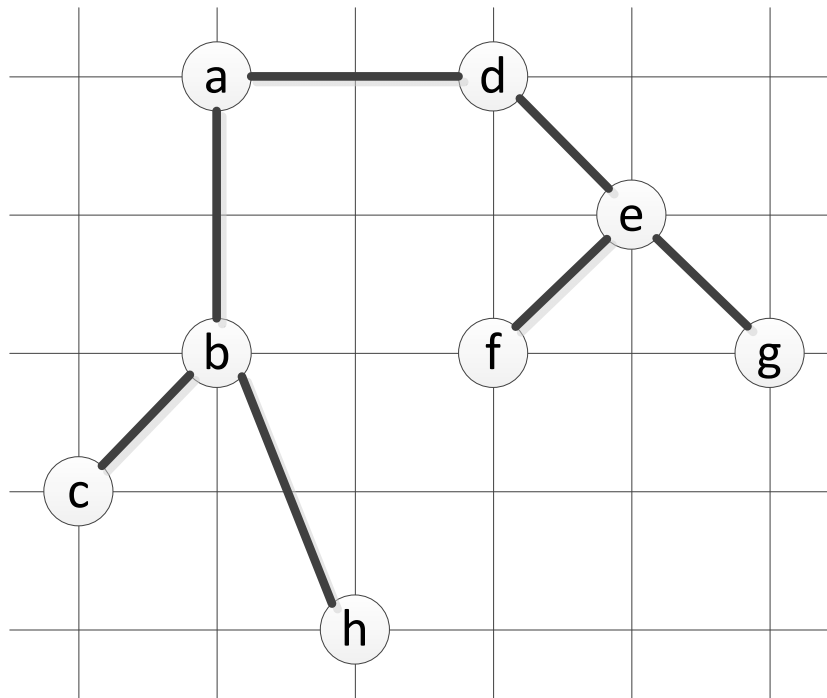
# Traveling Salesman Problem

- Example (refer to Figure 35.2): Given the following complete graph (There are edges from each node to all other nodes though edges are not shown in the graph below).



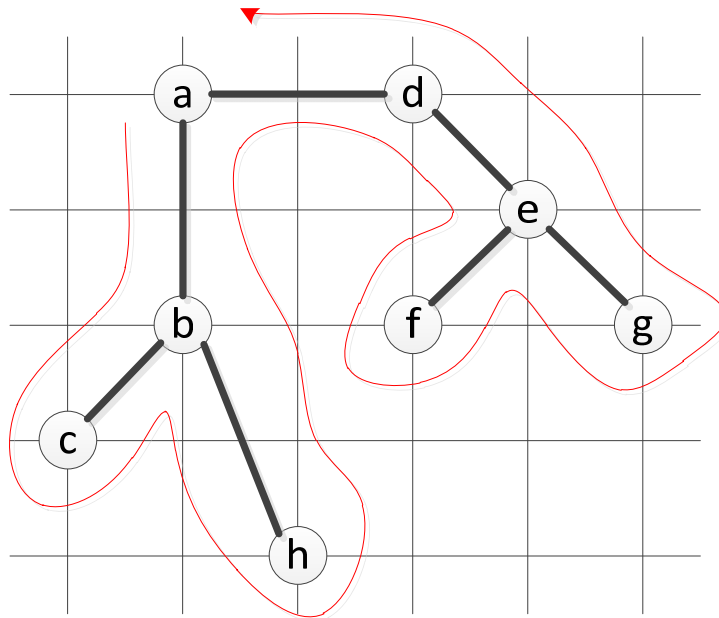
# Traveling Salesman Problem

- A minimum spanning tree  $T$  ( $a$  is the root)



# Traveling Salesman Problem

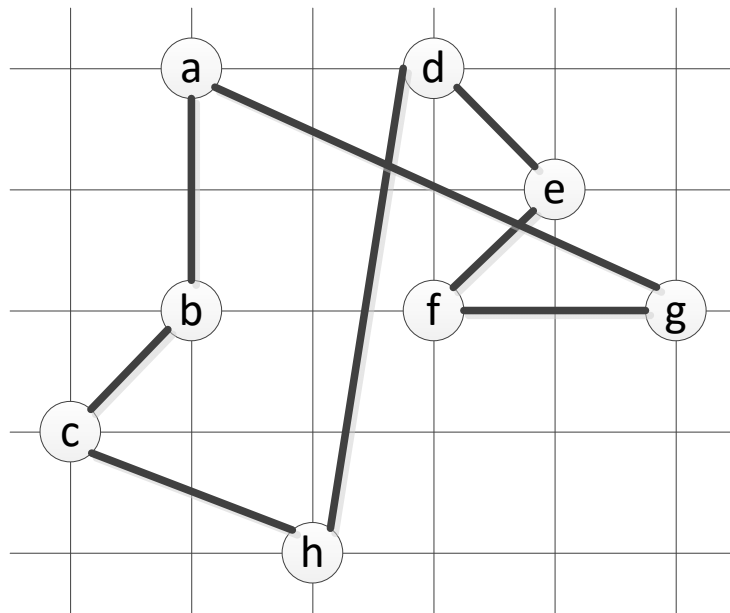
- A minimum spanning tree  $T$  ( $a$  is the root)



$a \rightarrow b \rightarrow c \rightarrow h \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow a$

# Traveling Salesman Problem

- $H$  returned by APPROX-TSP-TOUR is

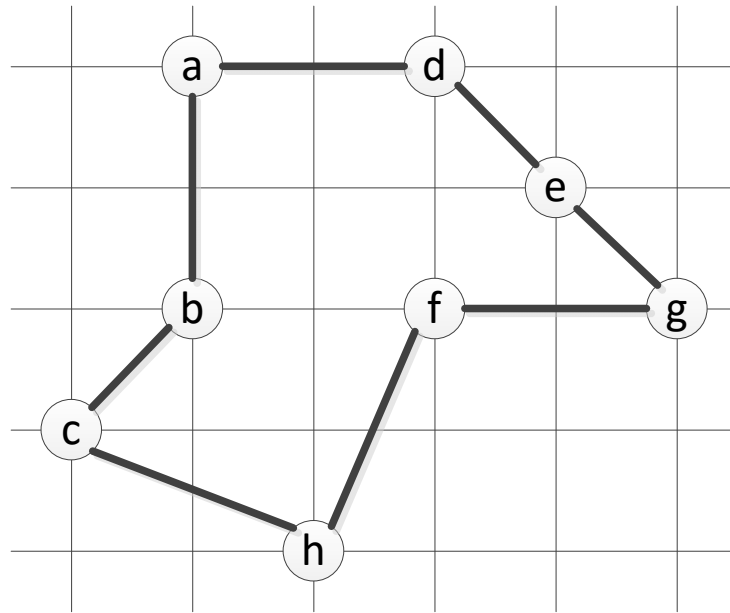


total weight = approx. 19.074



# Traveling Salesman Problem

- An optimal tour (or Hamiltonian cycle with minimum weight)



total weight = approx. 14.715

# Reference

- T.A. Sudkamp, “Languages and Machines,” 1988, Addison Wesley.
- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, “Introduction to Algorithms,” 3rd Ed., 2009, MIT Press.