Data Structures and Algorithms

P and NP

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 $\mathbf{P_{SQ}}$: Determine whether an arbitrary number is a perfect square or not. This problem consists of the following questions:

  $\mathbf{p}_0$: Is 0 a perfect square?

  $\mathbf{p}_1$: Is 1 a perfect square?

  …

  Here, $\mathbf{p}_i$ 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

- An example of unsolvable 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.

aba
bbaba

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:

aaa	*baa*
aa	*abaaa*

The following sequence of dominoes is a solution:

aaa	*baa*	*aaa*
aa	*abaaa*	*aa*

Decision Problem

- Example 2. Given the following three dominoes:

ab	*bba*	*aba*
aba	*aa*	*bab*

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 **P** is Turing reducible to a problem **P’** if there is a Turing machine that takes any problem $\mathbf{p}_i \in$ **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 $\mathbf{p'}_i$.

$$p \in \mathbf{P} \rightarrow \boxed{\text{Reduction of } \mathbf{P} \text{ to } \mathbf{P'}} \rightarrow p' \in \mathbf{P'} \rightarrow \boxed{\text{Algorithm to solve } \mathbf{P'}} \rightarrow \text{yes/no}$$

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, $\boldsymbol{P} \subseteq \boldsymbol{NP}$.
- But it was never proved that $\boldsymbol{NP} \subseteq \boldsymbol{P}$. (If this is proved, then that proves ***P = NP***.)

P and NP

- If Q is reducible to L in polynomial time and $L \in$ ***P*****,** then $Q \in$ ***P***.

- A language L is called ***NP-hard*** if for every $Q \in$ ***NP*** Q is reducible to L in polynomial time.
- An ***NP-hard*** language that is also in ***NP*** is called ***NP-complete***.

- If there is an NP-complete language that is also in ***P***, then ***P = 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***.

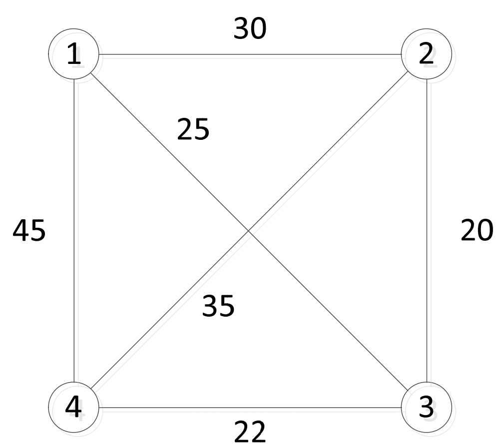
- But, it is decidable in nondeterministic polynomial time.
- Given a cycle in a graph, we can determine whether it is Hamiltonian cycle or not in polynomial time.
- So, Hamiltonian cycle problem is in ***NP***.
- In fact it is an ***NP-complete*** problem.

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 → 2 → 4 → 3 → 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 $S \mathrel{!}= \phi$

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

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

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

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

$v = w$;

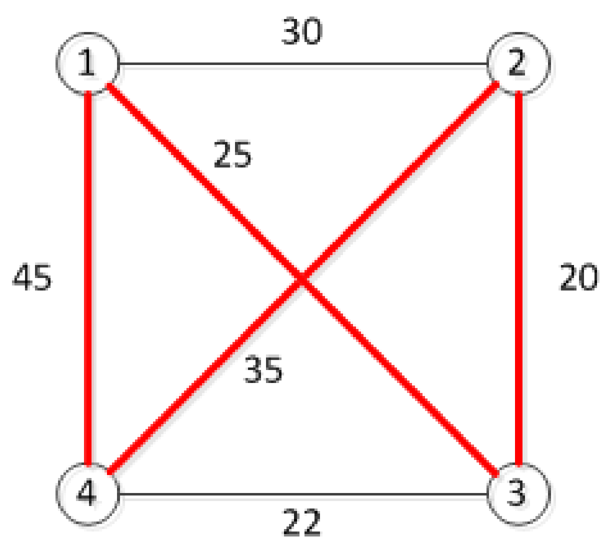
$C = C \cup \{(v, s)\}$;

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 $R \mathrel{!=} \phi$

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

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

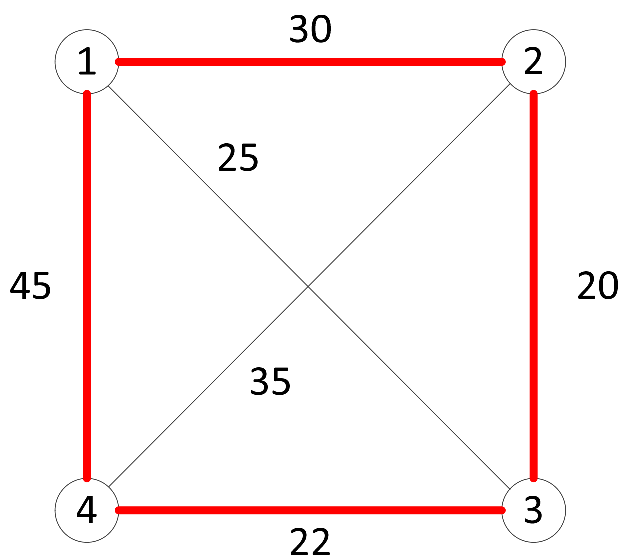
add the edge connecting the end points of the path in C;

return C;

$\boxed{\text{Running time: } O(E \log V)}$

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:

  $f(u, v) \le f(u, w) + f(w, v)$, for all $u, v, w \in G.V$.

- Euclidean distance has the triangle inequality property.

Traveling Salesman Problem

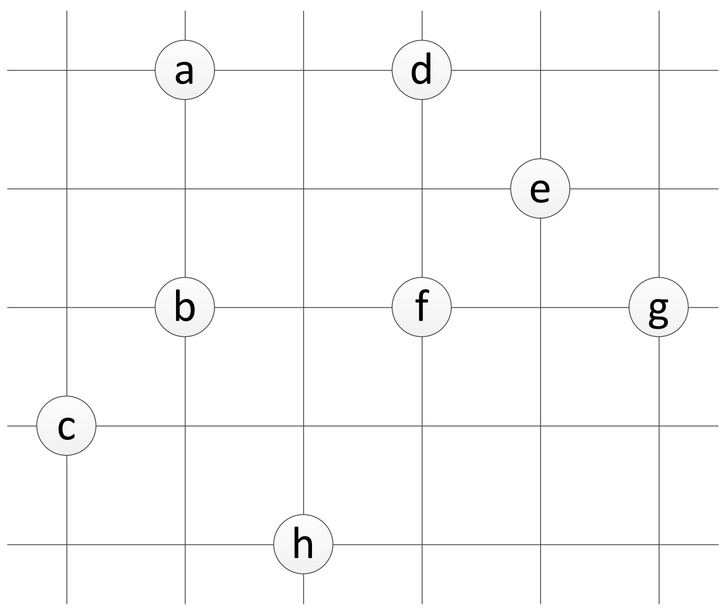
- The following approximate algorithm has an upper bound on the weight: total weight of a cycle is no more than the twice that of the minimum spanning tree's weight

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

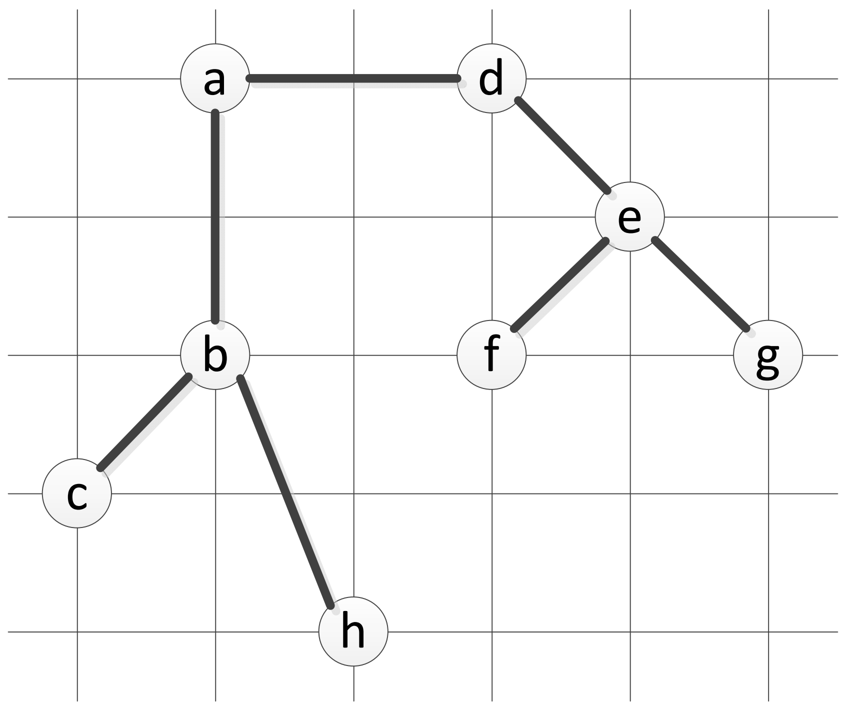
Traveling Salesman Problem

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



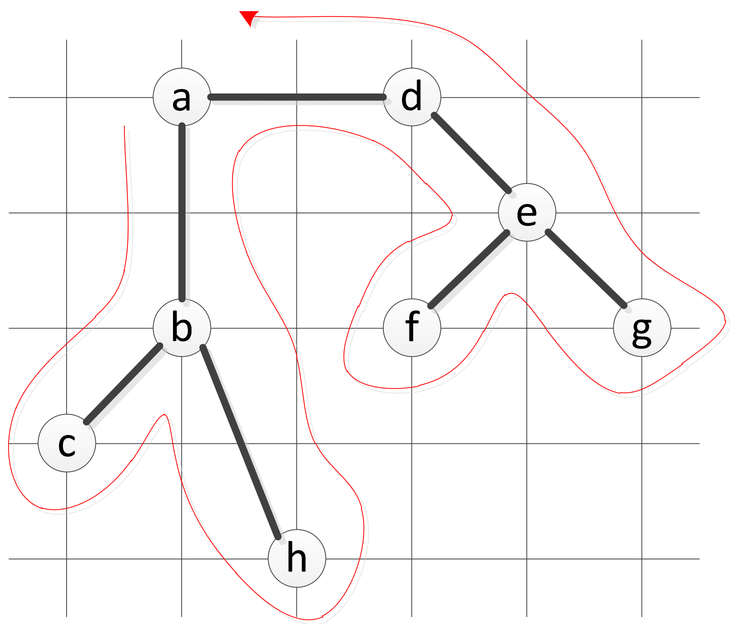
Traveling Salesman Problem

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



Traveling Salesman Problem

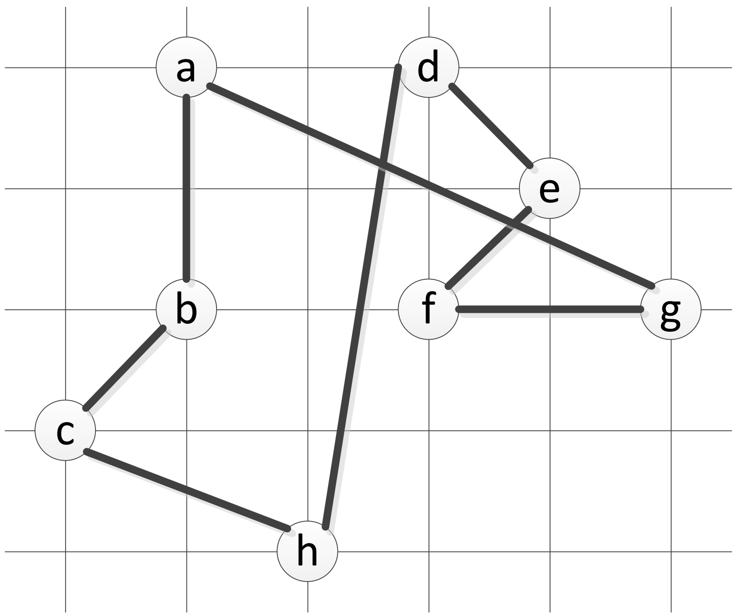
- A minimum spanning tree *T* (*a* is the root)



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

Traveling Salesman Problem

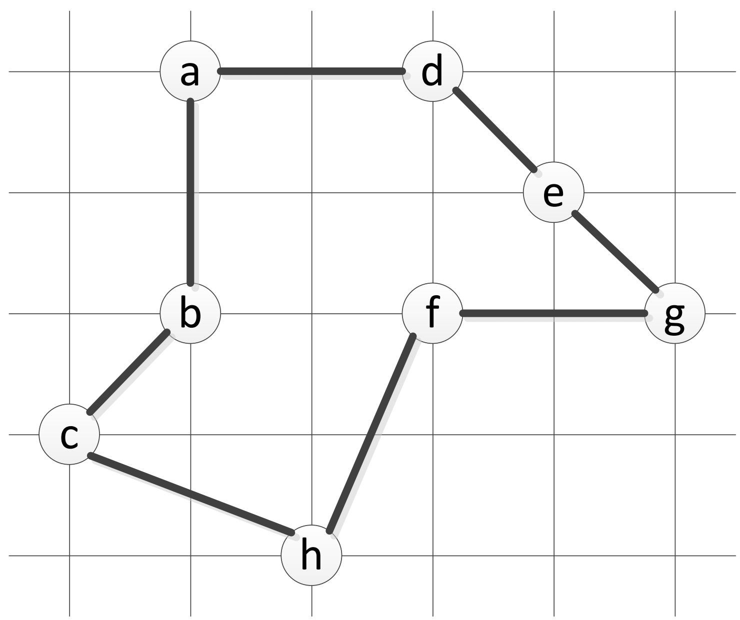
- *H* returned by APPROX-TSP-TOUR is



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.