Chapter 9 Trees 树

Lu Han

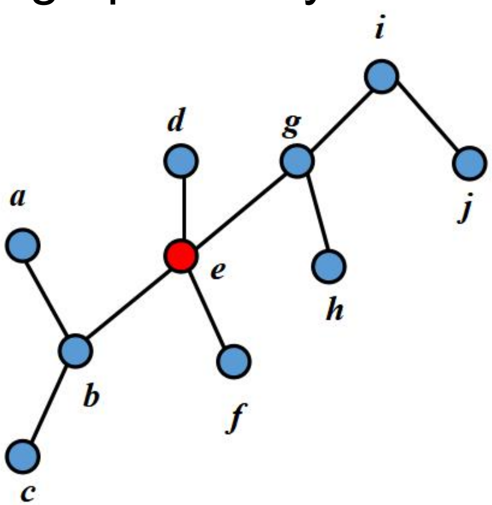
hl@bupt.edu.cn

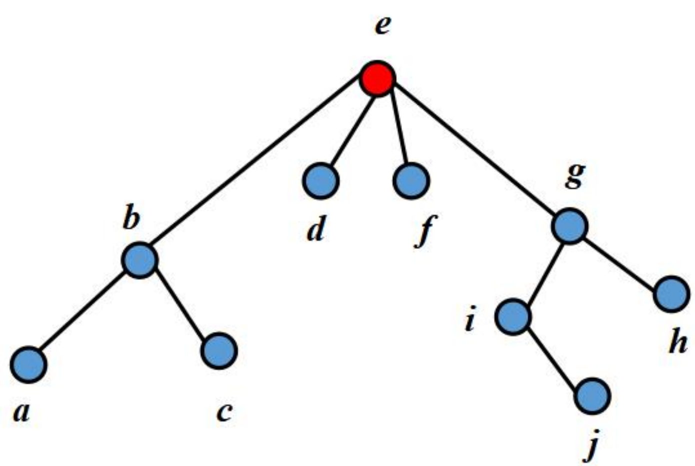
9.1 Introduction 简介

Definition 9.1.1 A **(free) tree (自由树)** T is a simple graph satisfying the following: If v and w are vertices in T, there is a unique simple path from v to w.

A **rooted tree (有根树)** is a tree in which a particular vertex is designated the root.

- In graph theory rooted trees are typically drawn with their roots at the top.





- We call the level of the root level 0.
- The vertices under the root are said to be on level 1, and so on.

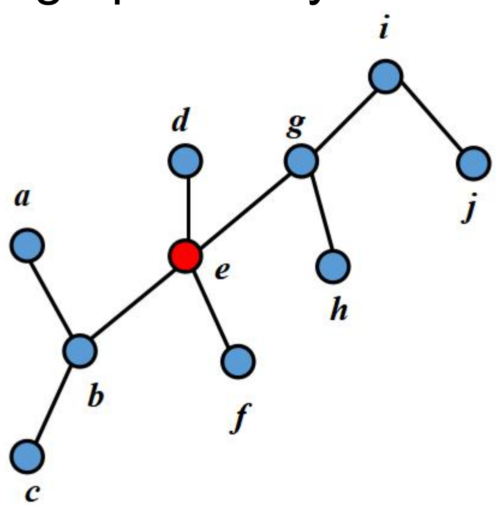
9.1 Introduction 简介

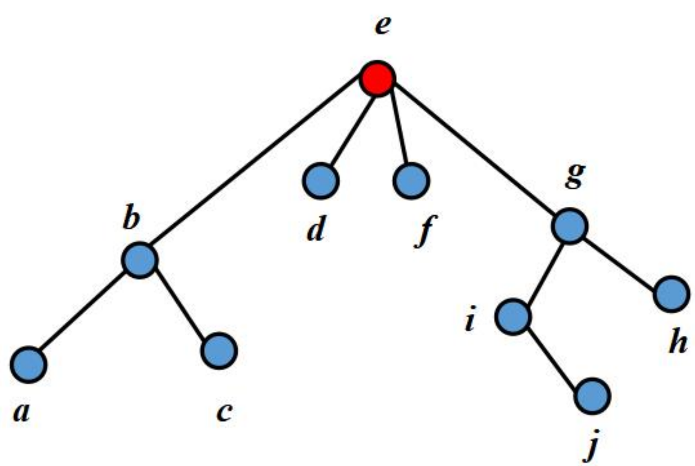
The **level of a vertex v (顶点 v 所在的层次)**: the length of the simple path from the root to v.

The **height (高度)** of a rooted tree: the maximum level number that occurs.

A **rooted tree (有根树)** is a tree in which a particular vertex is designated the root.

- In graph theory rooted trees are typically drawn with their roots at the top.





- We call the level of the root level 0.
- The vertices under the root are said to be on level 1, and so on.

9.2 Terminology and Characterizations of Trees 树的术语和性质

Definition 9.2.1 Let T be a tree with root v_0. Suppose that x, y, and z are vertices in T and that $(v_0, v_1, \ldots, v_n)$ is a simple path in T. Then

(a) v_{n-1} is the **parent (父节点)** of v_n.

(b) $v_0, \ldots, v_{n-1}$ are **ancestors (祖先节点)** of v_n.

(c) v_n is a **child (子节点)** of v_{n-1}.

(d) If x is an ancestor of y, y is a **descendant (后代节点)** of x.

(e) If x and y are children of z, x and y are **siblings (兄弟节点)**.

(f) If x has no children, x is a **terminal vertex (or a leaf) (终节点/叶节点)**.

(g) If x is not a terminal vertex, x is an **internal (or branch) vertex (中间节点/枝节点)**.

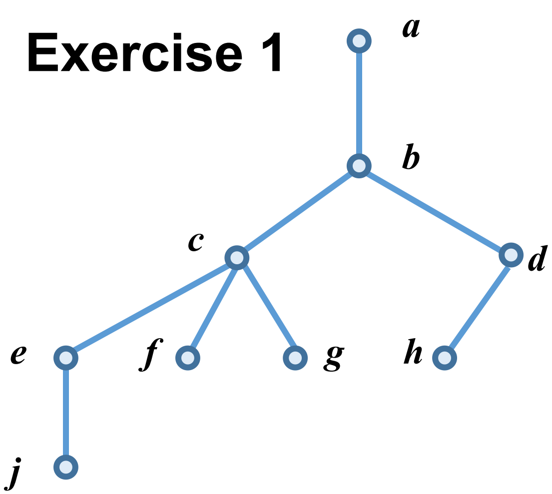
9.2 Terminology and Characterizations of Trees 树的术语和性质

Definition 9.2.1 Let T be a tree with root v_0. Suppose that x, y, and z are vertices in T and that $(v_0, v_1, \ldots, v_n)$ is a simple path in T. Then

(h) The **subtree (子树)** of T rooted at x is the graph with vertex set V and edge set E, where V is x together with the descendants of x and

$E = \{e \mid e$ is an edge on a simple path from x to some vertex in $V\}$.

Exercise 1



Draw the subtree rooted at c.

9.2 Terminology and Characterizations of Trees 树的术语和性质

Definition 9.1.1 A **(free) tree (自由树)** T is a simple graph satisfying the following: If v and w are vertices in T, there is a unique simple path from v to w.

A tree is connected.

A tree cannot contain a cycle.

A graph with no cycles is called an **acyclic graph (非循环图)**.

Definition 9.2.3 Let T be a graph with n vertices. The following are equivalent.

(a) T is a tree.

(b) T is connected and acyclic.

(c) T is connected and has $n-1$ edges.

(d) T is acyclic and has $n-1$ edges.

9.2 Terminology and Characterizations of Trees 树的术语和性质

Definition 9.1.1 A **(free) tree (自由树)** T is a simple graph satisfying the following: If v and w are vertices in T, there is a unique simple path from v to w.

A tree is connected.

A tree cannot contain a cycle.

A graph with no cycles is called an **acyclic graph (非循环图)**.

Definition 9.2.3 Let T be a graph with n vertices. The following are equivalent.

(a) T is a tree.

(b) T is connected and acyclic.

(c) T is connected and has $n-1$ edges.

(d) T is acyclic and has $n-1$ edges.

(a)→(b) √

9.2 Terminology and Characterizations of Trees 树的术语和性质

(b)→(c)

Definition 9.2.3 Let T be a graph with n vertices. The following are equivalent.
(a) T is a tree.
(b) T is connected and acyclic.
(c) T is connected and has $n-1$ edges.
(d) T is acyclic and has $n-1$ edges.

9.2 Terminology and Characterizations of Trees 树的术语和性质

(b)→(c)

[If (b), then (c).] Suppose that T is connected and acyclic. We will prove that T has $n-1$ edges by induction on n.

If $n=1$, T consists of one vertex and zero edges, so the result is true if $n=1$.

Now suppose that the result holds for a connected, acyclic graph with n vertices. Let T be a connected, acyclic graph with $n+1$ vertices. Choose a path P with no repeated edges of maximum length. Since T is acyclic, P contains no cycles. Therefore, P contains a vertex v of degree 1 (see Figure 9.2.4). Let T^* be T with v and the edge incident on v removed. Then T^* is connected and acyclic, and because T^* contains n vertices, by the inductive hypothesis T^* contains $n-1$ edges. Therefore, T contains n edges. The inductive argument is complete and this portion of the proof is complete.

9.2 Terminology and Characterizations of Trees 树的术语和性质

(c)→(d)

Definition 9.2.3 Let T be a graph with n vertices. The following are equivalent.
(a) T is a tree.
(b) T is connected and acyclic.
(c) T is connected and has $n-1$ edges.
(d) T is acyclic and has $n-1$ edges.

9.2 Terminology and Characterizations of Trees 树的术语和性质

(c)→(d)

[If (c), then (d).] Suppose that T is connected and has $n-1$ edges. We must show that T is acyclic.

Suppose that T contains at least one cycle. Since removing an edge from a cycle does not disconnect a graph, we may remove edges, but no vertices, from cycle(s) in T until the resulting graph T^* is connected and acyclic. Now T^* is an acyclic, connected graph with n vertices. We may use our just proven result, (b) implies (c), to conclude that T^* has $n-1$ edges. But now T has more than $n-1$ edges. This is a contradiction. Therefore, T is acyclic. This portion of the proof is complete.

9.2 Terminology and Characterizations of Trees 树的术语和性质

(d)→(a)

Definition 9.2.3 Let T be a graph with n vertices. The following are equivalent.
(a) T is a tree.
(b) T is connected and acyclic.
(c) T is connected and has $n-1$ edges.
(d) T is acyclic and has $n-1$ edges.

9.2 Terminology and Characterizations of Trees 树的术语和性质

[If (d), then (a).] Suppose that T is acyclic and has $n-1$ edges. We must show that T is a tree, that is, that T is a simple graph and that T has a unique simple path from any vertex to any other vertex.

The graph T cannot contain any loops because loops are cycles and T is acyclic. Similarly, T cannot contain distinct edges e_1 and e_2 incident on v and w because we would then have the cycle (v, e_1, w, e_2, v). Therefore, T is a simple graph.

Suppose, by way of contradiction, that T is not connected (see Figure 9.2.5). Let $T_1, T_2, \ldots, T_k$ be the components of T. Since T is not connected, $k > 1$. Suppose that T_i has n_i vertices. Each T_i is connected and acyclic, so we may use our previously proven result, (b) implies (c), to conclude that T_i has $n_i - 1$ edges. Now

$$
\begin{aligned}
n - 1 &= (n_1 - 1) + (n_2 - 1) + \cdots + (n_k - 1) && \text{(counting edges)} \\
&< (n_1 + n_2 + \cdots + n_k) - 1 && \text{(since } k > 1\text{)} \\
&= n - 1, && \text{(counting vertices)}
\end{aligned}
$$

which is impossible. Therefore, T is connected.

9.2 Terminology and Characterizations of Trees 树的术语和性质

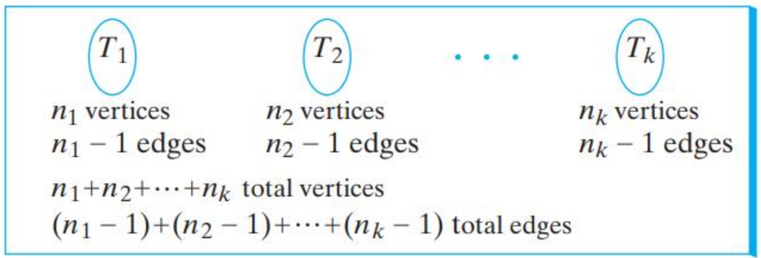


Figure 9.2.5 The proof of Theorem 9.2.3 [if (d), then (a)]. The T_i are components of T. T_i has n_i vertices and $n_i - 1$ edges. A contradiction results from the fact that the total number of edges must equal $n - 1$.

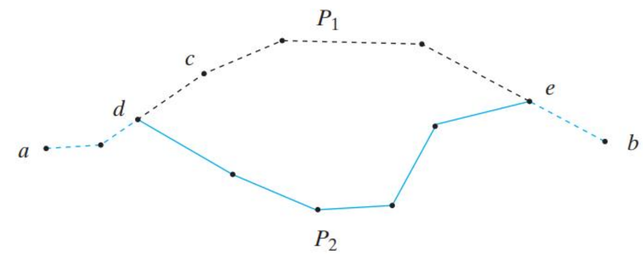


Figure 9.2.6 The proof of Theorem 9.2.3 [if (d), then (a)]. P_1 (shown dashed) and P_2 (shown in color) are distinct simple paths from a to b. c is the first vertex after a on P_1 not in P_2. d is the vertex preceding c on P_1. e is the first vertex after d on P_1 that is also on P_2. As shown, a cycle results, which gives a contradiction.

Suppose that there are distinct simple paths P_1 and P_2 from a to b in T (see Figure 9.2.6). Let c be the first vertex after a on P_1 that is not in P_2; let d be the vertex preceding c on P_1; and let e be the first vertex after d on P_1 that is also on P_2. Let $(v_0, v_1, \ldots, v_{n-1}, v_n)$ be the portion of P_1 from $d = v_0$ to $e = v_n$. Let $(w_0, w_1, \ldots, w_{m-1}, w_m)$ be the portion of P_2 from $d = w_0$ to $e = w_m$. Now

$$(v_0, \ldots, v_n = w_m, w_{m-1}, \ldots, w_1, w_0) \tag{9.2.1}$$

is a cycle in T, which is a contradiction. [In fact, (9.2.1) is a simple cycle since no vertices are repeated except for v_0 and w_0.] Thus there is a unique simple path from any vertex to any other vertex in T. Therefore, T is a tree. This completes the proof.

9.2 Terminology and Characterizations of Trees 树的术语和性质

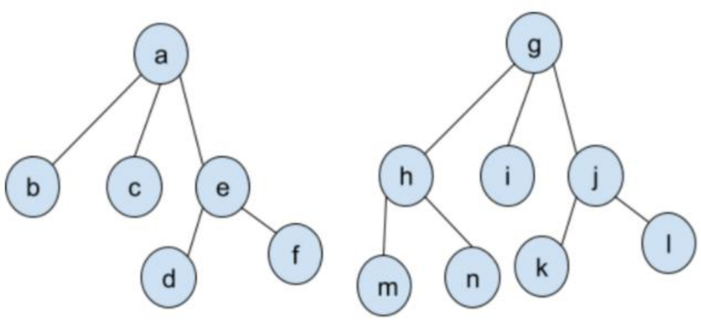
If T is a graph with n vertices, the following are equivalent (Theorem 9.2.3):
(a) T is a tree.
(b) If v and w are vertices in T, there is a unique simple path from v to w.
(c) T is connected and acyclic.
(d) T is connected and has $n-1$ edges.
(e) T is acyclic and has $n-1$ edges.

A **forest (森林)** is a simple graph with no cycles.

9.2 Terminology and Characterizations of Trees 树的术语和性质

If T is a graph with n vertices, the following are equivalent (Theorem 9.2.3):
(a) T is a tree.
(b) If v and w are vertices in T, there is a unique simple path from v to w.
(c) T is connected and acyclic.
(d) T is connected and has $n - 1$ edges.
(e) T is acyclic and has $n - 1$ edges.

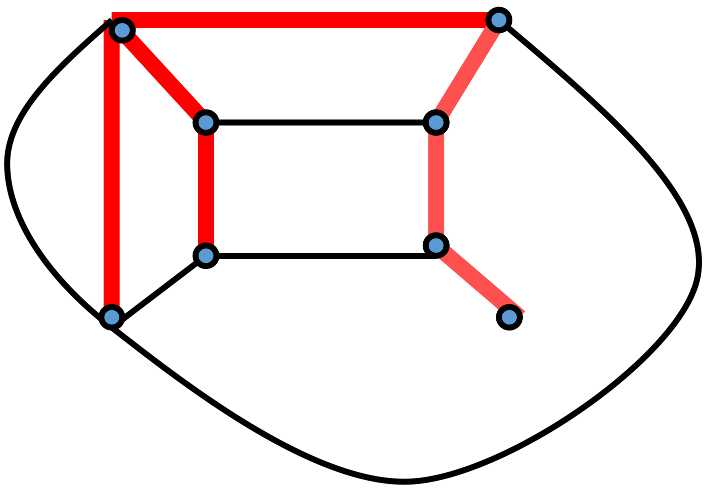
A **forest (森林)** is a simple graph with no cycles.



Forest

9.3 Spanning Trees 生成树

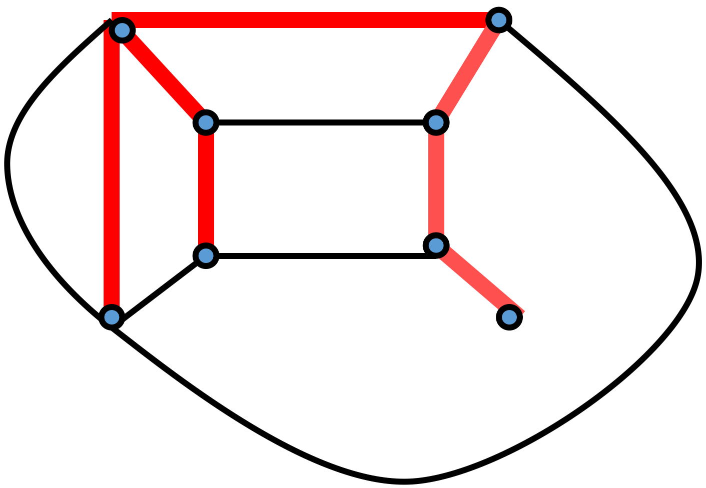
Definition 9.3.1 A tree T is a **spanning tree (生成树)** of a graph G if T is a subgraph of G that contains all of the vertices of G.

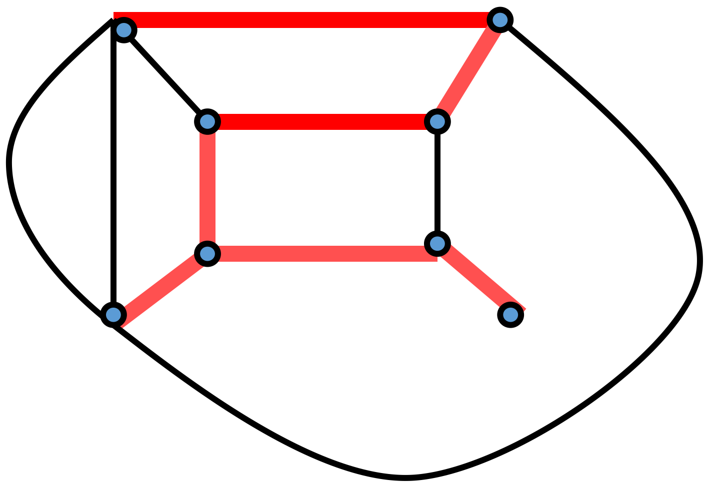


9.3 Spanning Trees 生成树

Definition 9.3.1 A tree T is a **spanning tree (生成树)** of a graph G if T is a subgraph of G that contains all of the vertices of G.

In general, a graph will have several spanning trees.





9.3 Spanning Trees 生成树

Theorem 9.3.4 A graph G has a spanning tree if and only if G is connected.

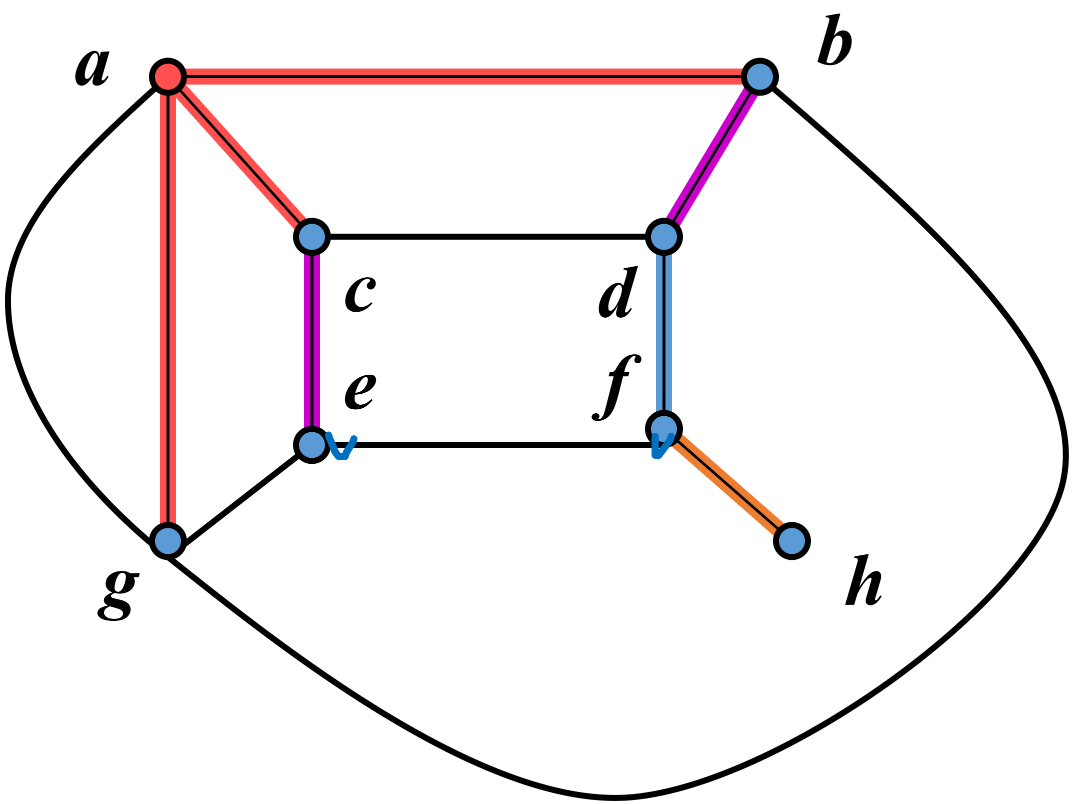
9.3 Spanning Trees 生成树

Breadth-First Search 广度优先搜索

The idea of breadth-first search is to process all the vertices on a given level before moving to next-higher level.

9.3 Spanning Trees 生成树

Breadth-First Search 广度优先搜索



- Select an ordering, say $abcdefgh$, of the vertices of G.
- Select the first vertex a and label it the root. Let T consist of the single vertex a and no edges.
- Add to T all edges (a, x) and vertices on which they are incident, for $x = b$ to h, that do not produce a cycle when added to T.
- Repeat this procedure with the vertices on level 1 (2, 3, ...) by examing each in order.
- Since no edge can be added to the single vertex h on lever 4, the procedure ends.

9.3 Spanning Trees 生成树

Breadth-First Search 广度优先搜索

Input: A connected graph G with vertices ordered

$$v_1, v_2, \ldots, v_n$$

Output: A spanning tree T

```
bfs(V, E) {
  // V = vertices ordered v1, ..., vn; E = edges
  // V' = vertices of spanning tree T; E' = edges of spanning tree T
  // v1 is the root of the spanning tree
  // S is an ordered list
  S = (v1)
  V' = {v1}
  E' = ∅
  while (true) {
    for each x ∈ S, in order,
      for each y ∈ V − V', in order,
        if ((x, y) is an edge)
          add edge (x, y) to E' and y to V'
    if (no edges were added)
      return T
    S = children of S ordered consistently with the original vertex ordering
  }
}
```

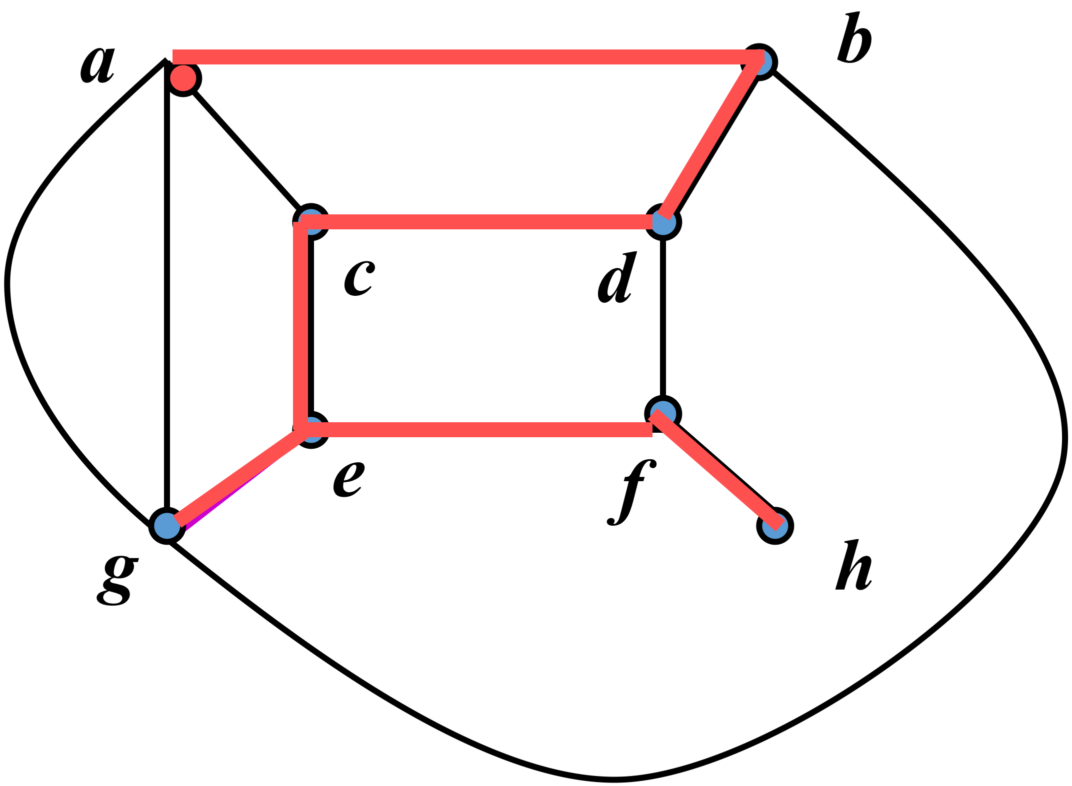
9.3 Spanning Trees 生成树

Depth-First Search 深度优先搜索

The idea of depth-first search is to proceeds to successive levels in a tree at the earliest possible opportunity.

9.3 Spanning Trees 生成树

Depth-First Search 深度优先搜索



- Select an ordering, say $abcdefgh$, of the vertices of G.
- Select the first vertex a and label it the root. Let T consist of the single vertex a and no edges.
- Add to T the edge (a, x) with minimal x and the vertex x, which is incident and does not produce a cycle when added to T.
- Repeat this procedure with the vertex on the next level until we cannot add an edge.
- Backtrack to the parent of the current vertx and try to add an edge.
- When no more edges can be added, we finally backtrack to the root and algorithm ends.

9.3 Spanning Trees 生成树

Depth-First Search 深度优先搜索

Input: A connected graph G with vertices ordered

$$v_1, v_2, \ldots, v_n$$

Output: A spanning tree T

```
dfs(V, E) {
  // V′ = vertices of spanning tree T; E′ = edges of spanning tree T
  // v1 is the root of the spanning tree
  V′ = {v1}
  E′ = ∅
  w = v1
  while (true) {
    while (there is an edge (w, v) that when added to T does not create a cycle
      in T) {
      choose the edge (w, vk) with minimum k that when added to T
        does not create a cycle in T
      add (w, vk) to E′
      add vk to V′
      w = vk
    }
    if (w == v1)
      return T
    w = parent of w in T // backtrack
  }
}
```

9.3 Spanning Trees 生成树

Depth-First Search 深度优先搜索

Input: A connected graph G with vertices ordered

$$v_1, v_2, \ldots, v_n$$

Output: A spanning tree T

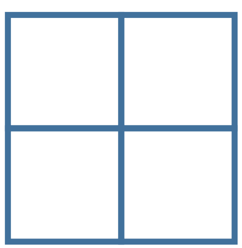
```
dfs(V, E) {
  // V′ = vertices of spanning tree T; E′ = edges of spanning tree T
  // v1 is the root of the spanning tree
  V′ = {v1}
  E′ = ∅
  w = v1
  while (true) {
    while (there is an edge (w, v) that when added to T does not create a cycle
      in T) {
      choose the edge (w, vk) with minimum k that when added to T
        does not create a cycle in T
      add (w, vk) to E′
      add vk to V′
      w = vk
    }
    if (w == v1)
      return T
    w = parent of w in T // backtrack
  }
}
```

Backtracking
回溯

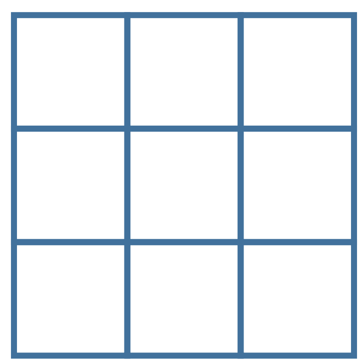
9.3 Spanning Trees 生成树

Four-Queens Problem 4皇后问题

To place for tokens on a 4×4 grid so that no two tokens are on the same row, column, or diagonal.



Two-Queens Problem

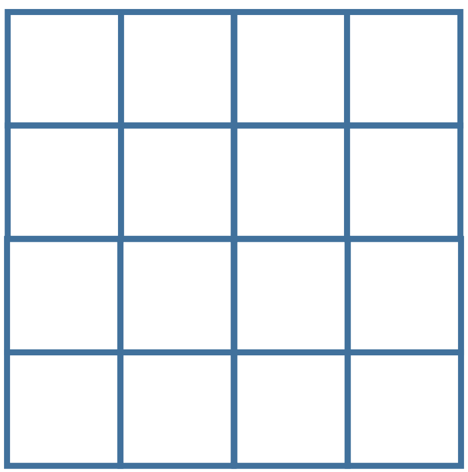


Three-Queens Problem

9.3 Spanning Trees 生成树

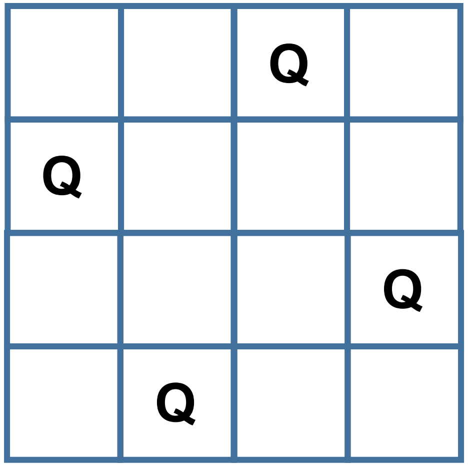
Four-Queens Problem 4皇后问题

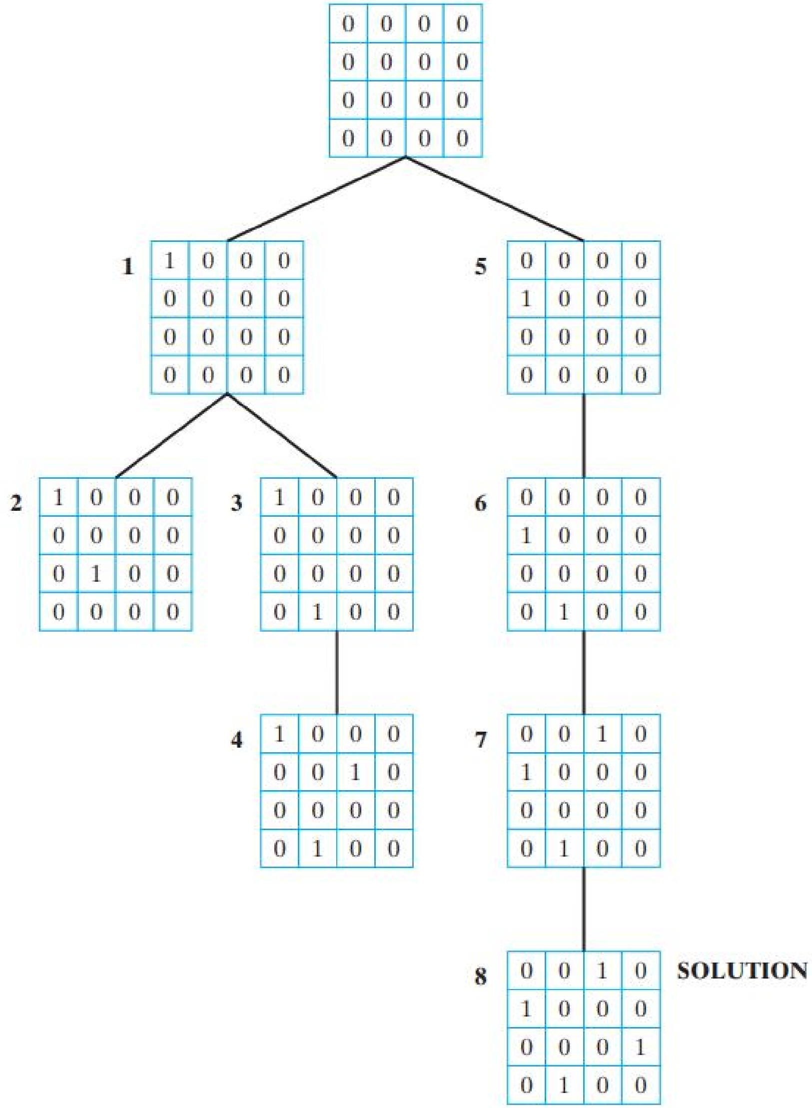
To place for tokens on a 4×4 grid so that no two tokens are on the same row, column, or diagonal.



9.3 Spanning Trees 生成树

Four-Queens Problem 4皇后问题



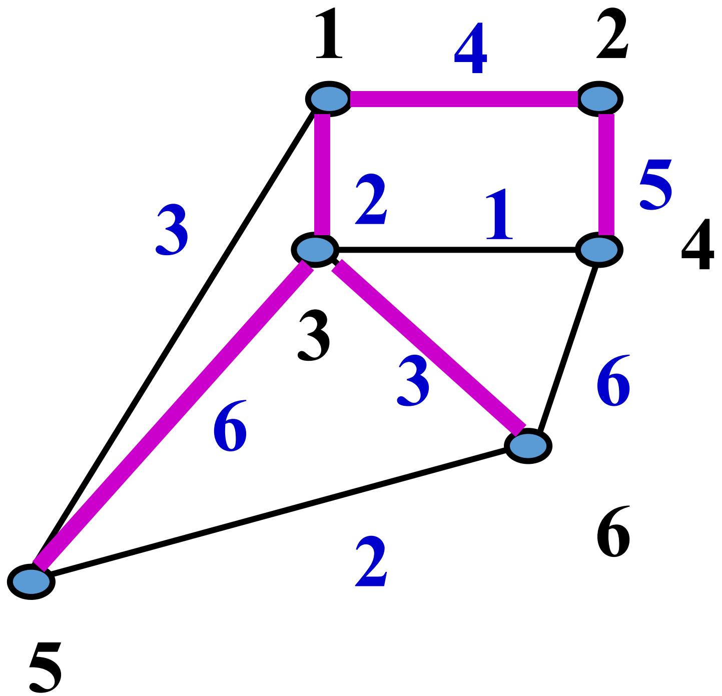


9.4 Minimal Spanning Trees 最小生成树

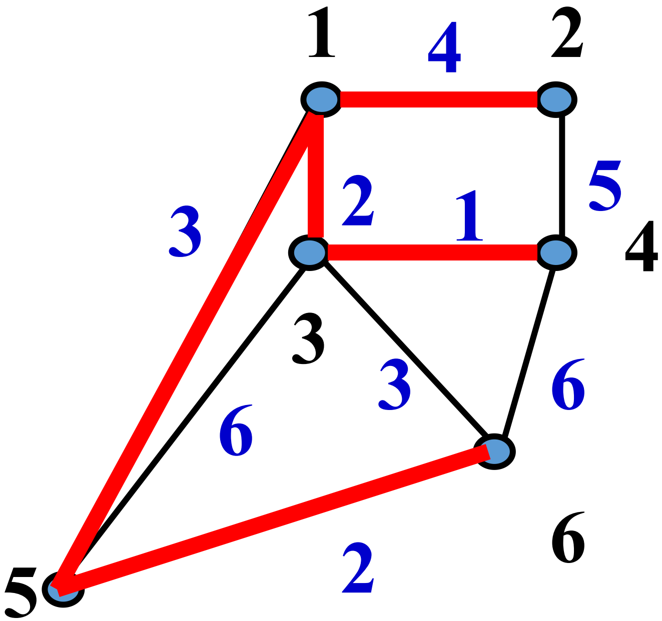
Definition 9.4.1 Given a weighted graph G, a **minimal spanning tree (最小生成树)** of G is a spanning tree of G that has minimum weight.

9.4 Minimal Spanning Trees 最小生成树

Definition 9.4.1 Given a weighted graph G, a **minimal spanning tree (最小生成树)** of G is a spanning tree of G that has minimum weight.



Weight=20

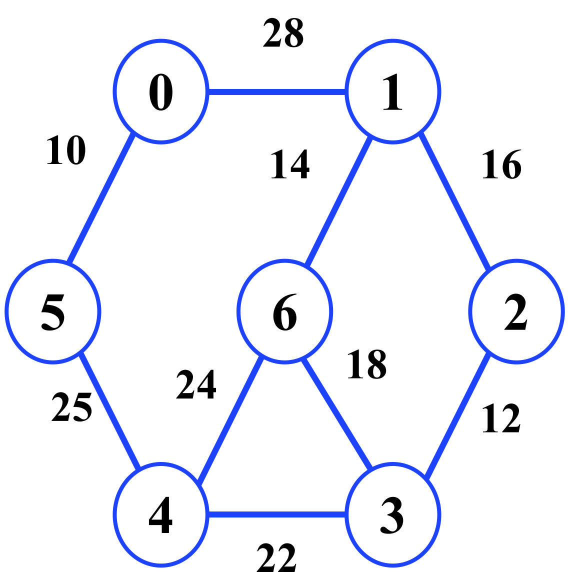


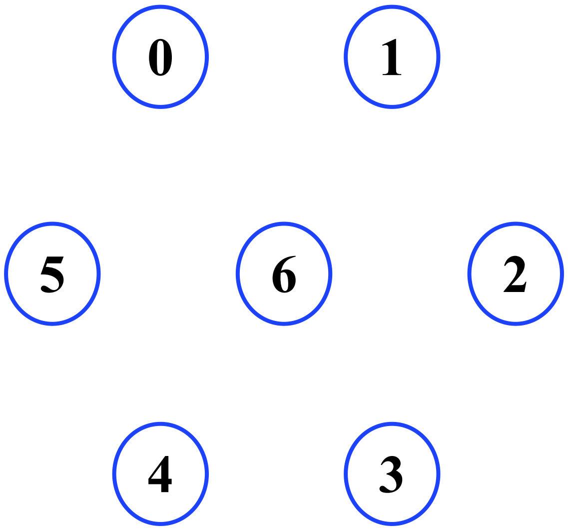
Weight=12

9.4 Minimal Spanning Trees 最小生成树

Prim’s Algorithm 普里姆算法

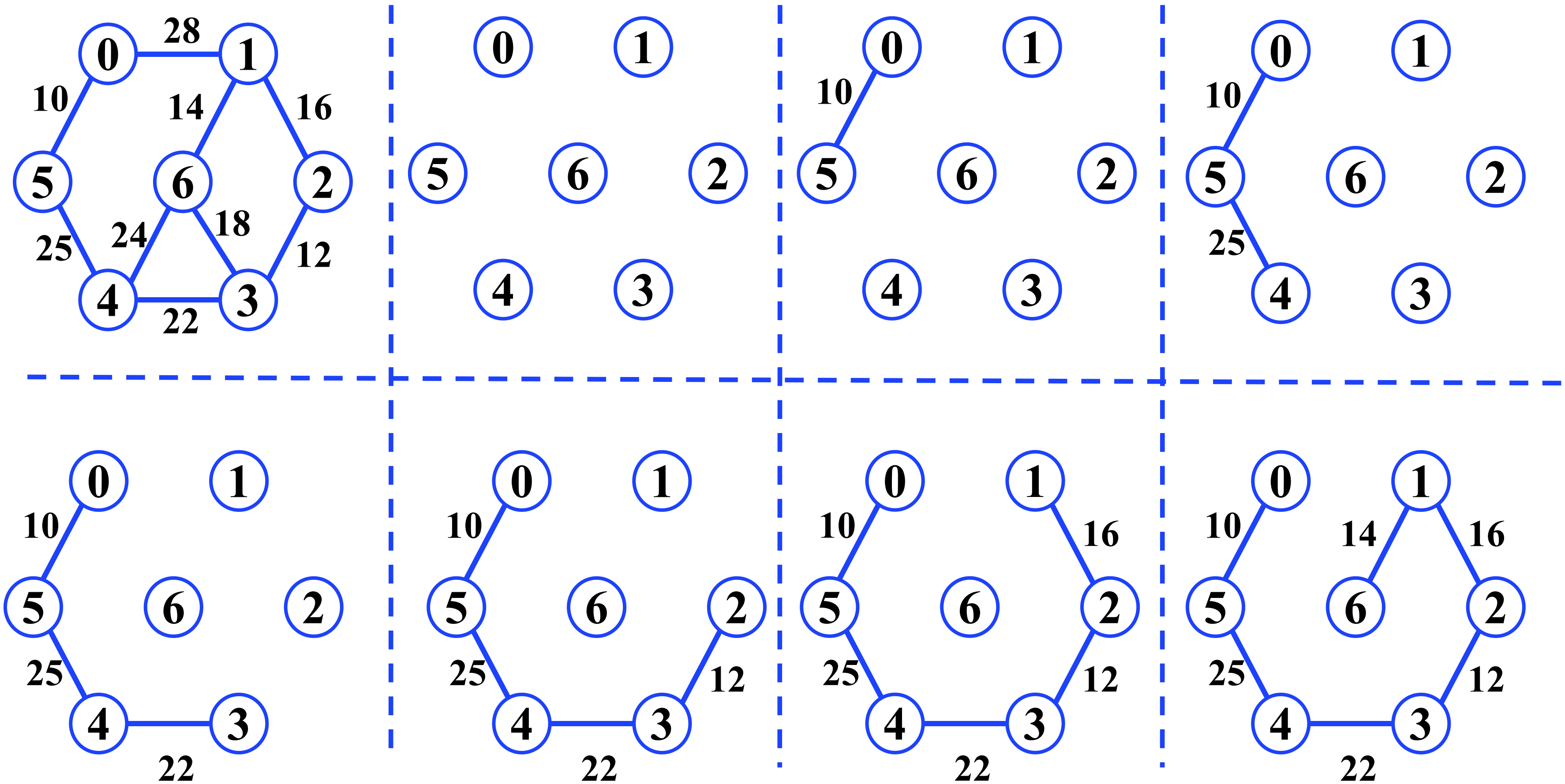
The algorithm begins with a single vertex. Then at each iteration, it adds to the current tree a minimum-weight edge that does not complete a cycle.





Keep finding the minimum-weight edge with one vertex in the tree and one vertex not in the tree.

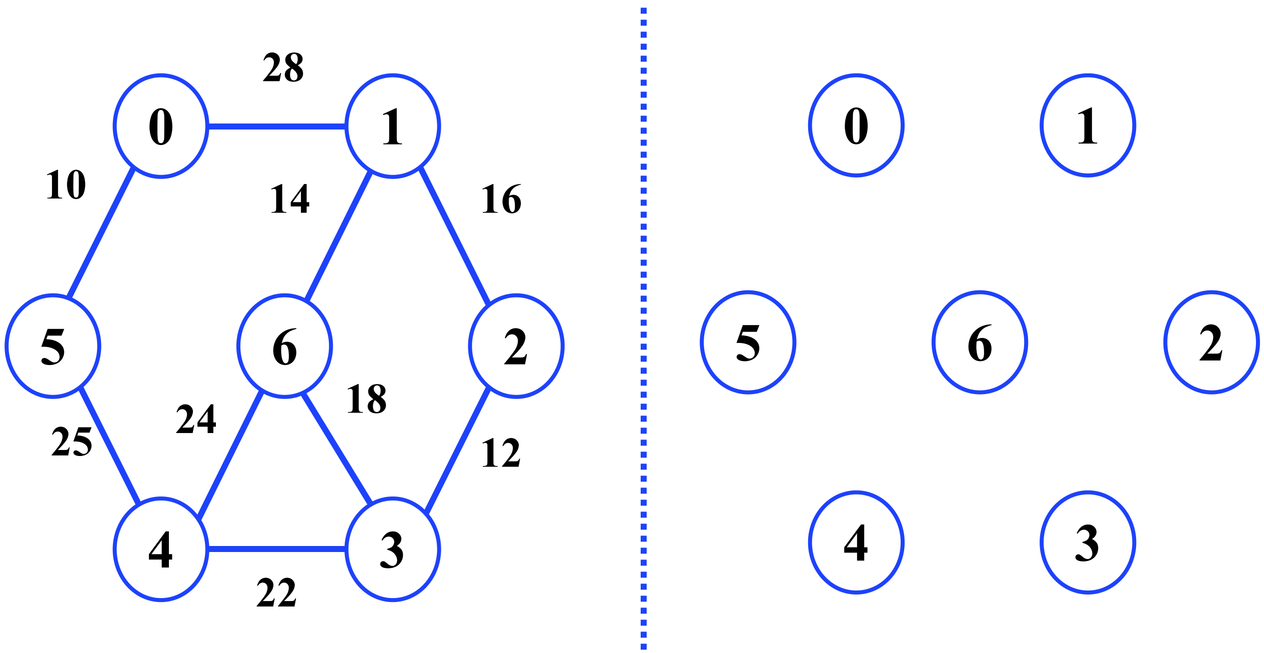
Prim’s Algorithm 普里姆算法



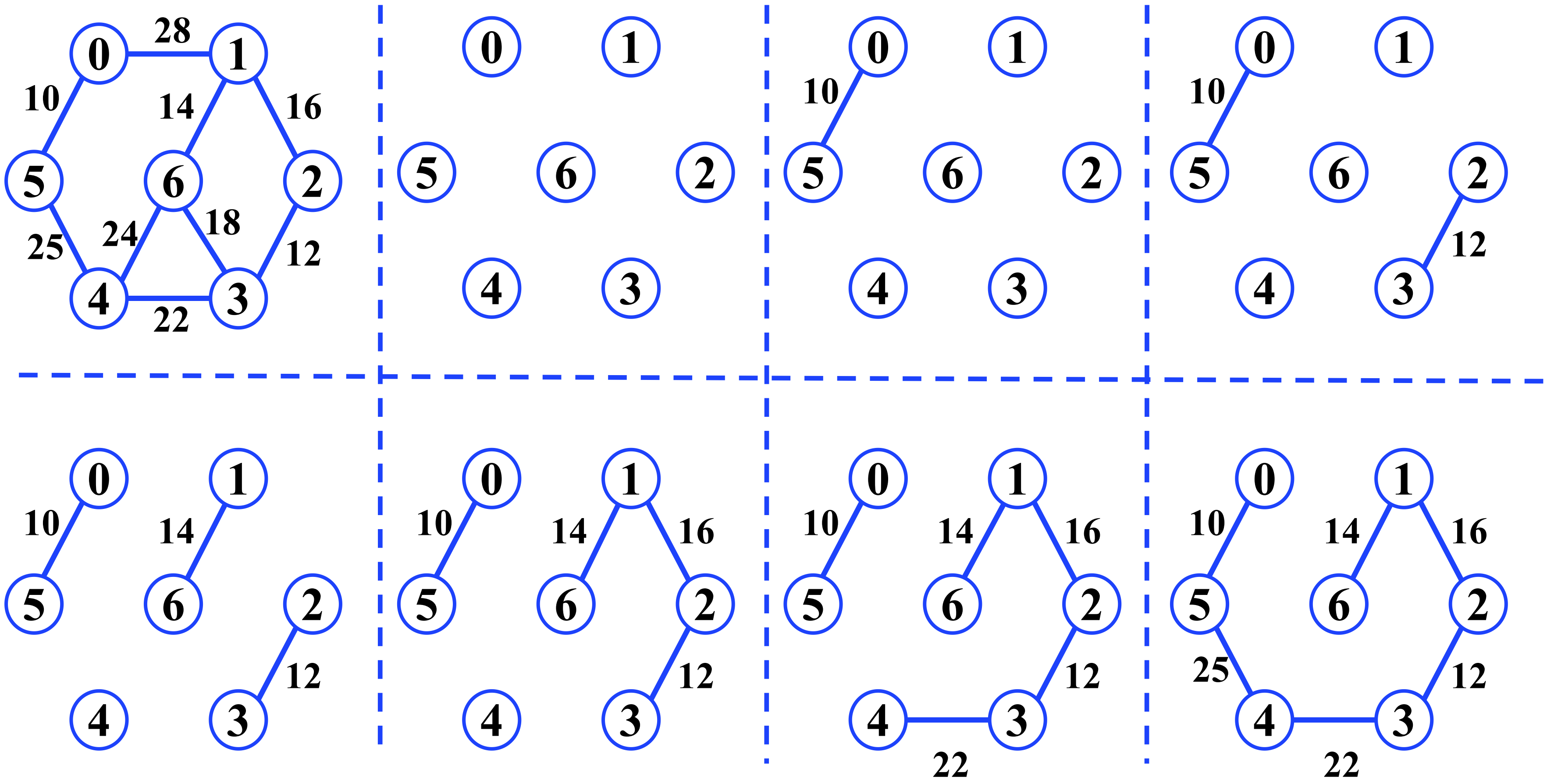
9.4 Minimal Spanning Trees 最小生成树

Kruskal's Algorithm 克鲁斯卡尔算法

It is a greedy algorithm in graph theory as in each step it adds the next lowest-weight edge that will not form a cycle to the minimum spanning forest.



Kruskal's Algorithm 克鲁斯卡尔算法



Minimal Spanning Trees and Metric TSP

Traveling salesman problem (TSP)

Given a completem graph with nonnegative edge costs, find a minimum cost cycle visiting every vertex exactly once.

Metric TSP

The edge costs satisf triangle inequality.

Minimal Spanning Trees and Metric TSP

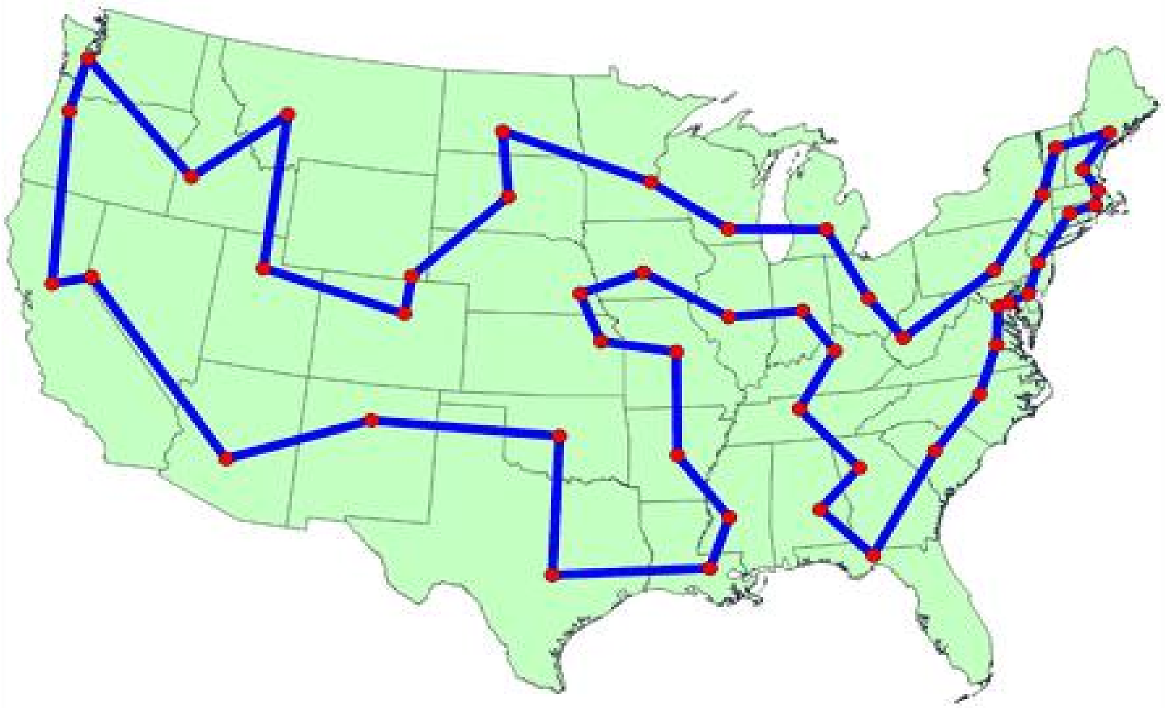
Traveling salesman problem (TSP)
Given a completem graph with nonnegative edge costs, find a minimum cost cycle visiting every vertex exactly once.

Metric TSP
The edge costs satisf triangle inequality.

The Metric TSP is an NP-hard problem.
If $\mathrm{P} \neq \mathrm{NP}$, we can’t simultaneously have algorithms that
(1) find optimal solutions
(2) in polynomial-time
(3) for any instance.
At least one of these requirements must be relaxed in any approach to dealing with an NP-hard optimization problem.

NP-hard Problem

Traveling Salesman Problem



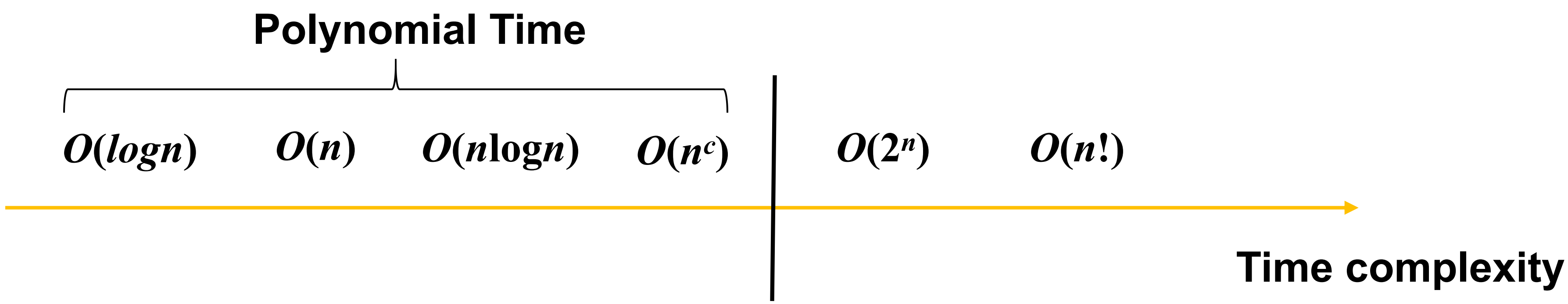
P & NP

P： (Decision) problems solvable in Polynomial time

P & NP

P：(Decision) problems solvable in Polynomial time

Why polynomial time is important?



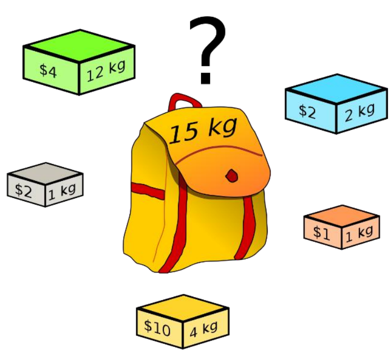
P & NP

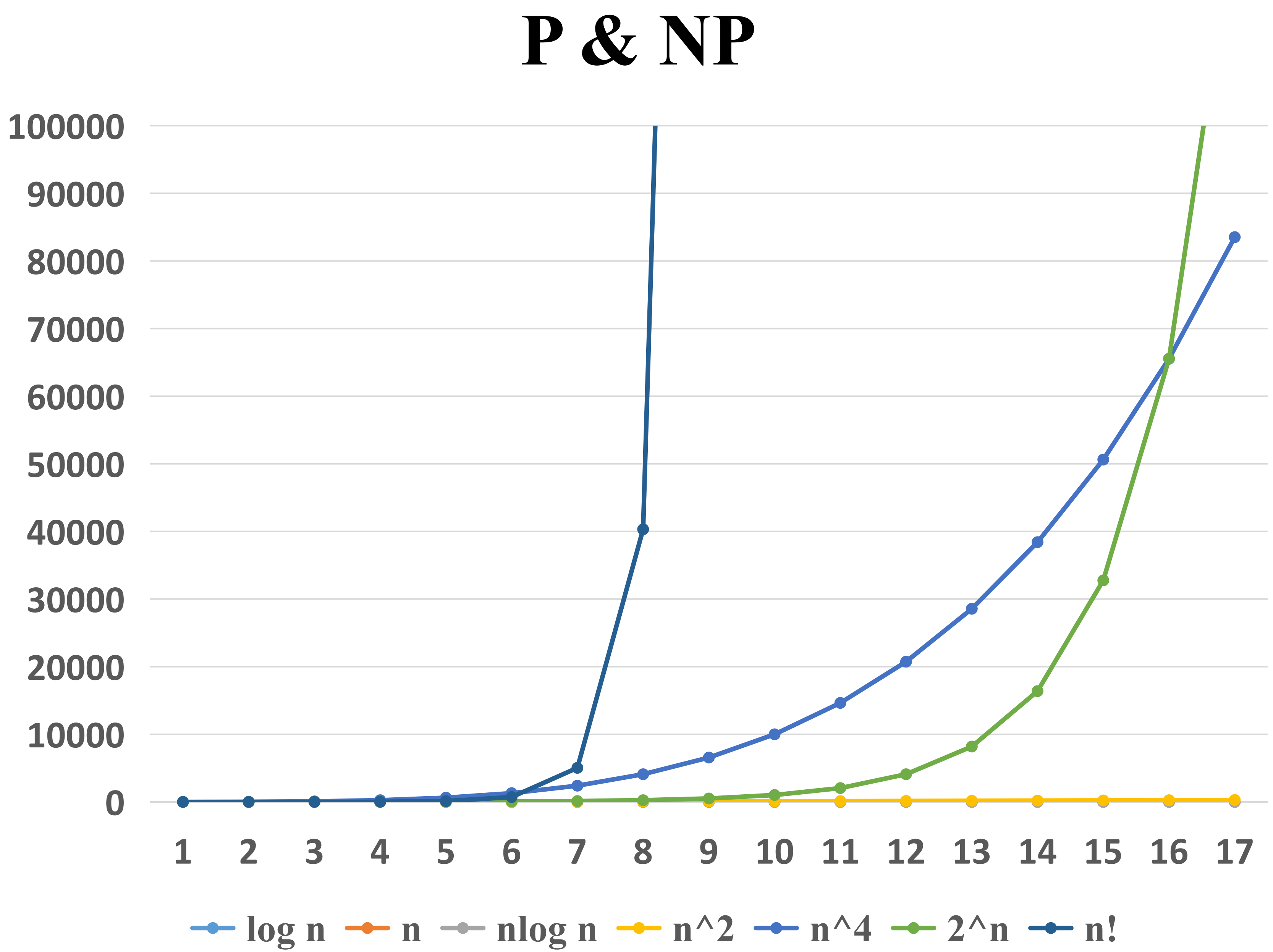
P：(Decision) problems solvable in Polynomial time

Why polynomial time is important?

$$\overbrace{O(\log n) \quad O(n) \quad O(n\log n) \quad O(n^c)}^{\text{Polynomial Time}} \quad \Big| \quad O(2^n) \quad O(n!)$$

Time complexity




P & NP
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
log n
n
nlog n
n^2
n^4
2^n
n!

P & NP

	$f(n)$	$n = 20$	$n = 40$	$n = 60$
算法 1	$\log_2 n$	4.32×10^{-6}秒	5.32×10^{-6}秒	5.91×10^{-6}秒
算法 2	$\sqrt{n}$	4.47×10^{-6}秒	6.32×10^{-6}秒	7.75×10^{-6}秒
算法 3	n	20×10^{-6}秒	40×10^{-6}秒	60×10^{-6}秒
算法 4	$n \log_2 n$	86×10^{-6}秒	213×10^{-6}秒	354×10^{-6}秒
算法 5	n^2	400×10^{-6}秒	1600×10^{-6}秒	3600×10^{-6}秒
算法 6	n^4	0.16秒	2.56秒	-------秒
算法 7	2^n	1.05秒	12.73天	-------年
算法 8	$n!$	77147年	2.56×10^{34}年	2.64×10^{68}年

P & NP

P： **(Decision) problems solvable in Polynomial time**

NP： **Decision Problems verifiable in Polynomial time**

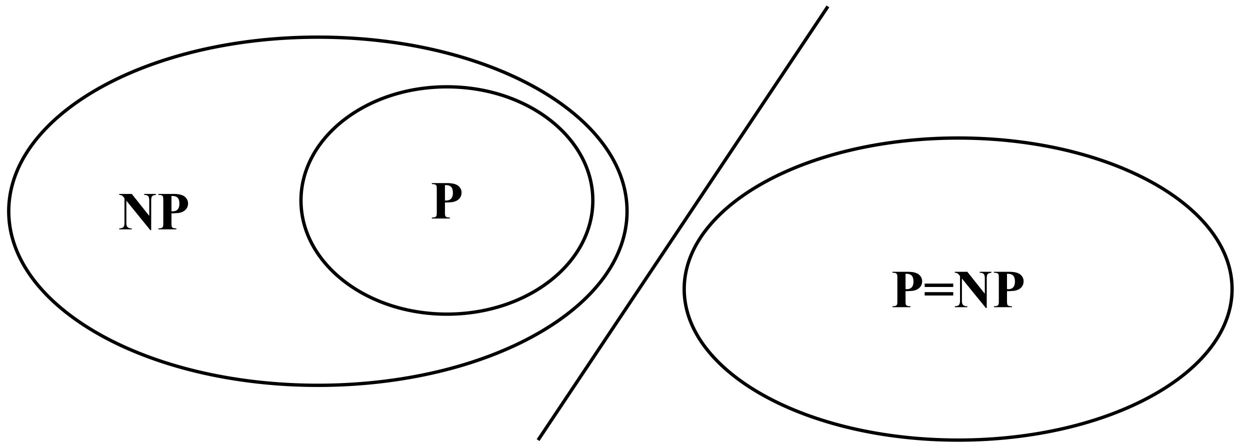
P & NP

P： (Decision) problems solvable in Polynomial time

NP： Decision Problems verifiable in Polynomial time

P & NP

P： **(Decision) problems solvable in Polynomial time**
NP： **Decision Problems verifiable in Polynomial time**

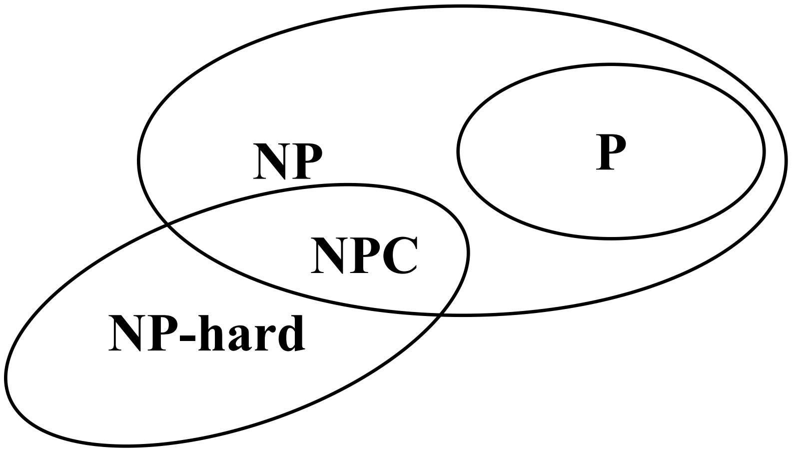


千禧年大奖难题
Millennium Prize Problems

P & NP

P： (Decision) problems solvable in Polynomial time

NP： Decision Problems verifiable in Polynomial time



Assume P≠NP

Minimal Spanning Trees and Metric TSP

Traveling salesman problem (TSP)
Given a completem graph with nonnegative edge costs, find a minimum cost cycle visiting every vertex exactly once.

Metric TSP
The edge costs satisf triangle inequality.

The Metric TSP is an NP-hard problem.
If $\mathrm{P} \neq \mathrm{NP}$, we can't simultaneously have algorithms that
(1) find optimal solutions
(2) in polynomial-time
(3) for any instance.
At least one of these requirements must be relaxed in any approach to dealing with an NP-hard optimization problem.

Minimal Spanning Trees and Metric TSP

Traveling salesman problem (TSP)
Given a completem graph with nonnegative edge costs, find a minimum cost cycle visiting every vertex exactly once.

Metric TSP
The edge costs satisf triangle inequality.

The Metric TSP is an NP-hard problem.
If $\mathrm{P} \neq \mathrm{NP}$, we can't simultaneously have algorithms that
(1) find optimal solutions ⇒ 2-approximation algorithm
(2) in polynomial-time
(3) for any instance.
At least one of these requirements must be relaxed in any approach to dealing with an NP-hard optimization problem.

Minimal Spanning Trees and Metric TSP

Traveling salesman problem (TSP)

Given a completemgraph with nonnegative edge costs, find a minimum cost cycle visiting every vertex exactly once.

Metric TSP

The edge costs satisf triangle inequality.

Approximation Algorithms

Definition: An α-appproximation algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of α of the value of an optimal solution.

极小化问题

$$\sup_{\text{实例 } I} \frac{\text{近似算法解的值 Cost A (I)}}{\text{最优解的值 Opt (I)}} \leq \alpha$$

Minimal Spanning Trees and Metric TSP

A 2-approximation algorithm for the Metric TSP

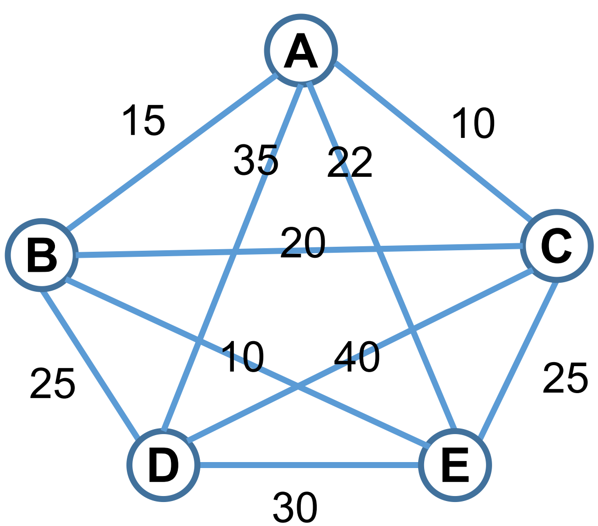
Input：A completem graph with nonnegative edge costs that satisfy the triangle inequality.
Ouptput: A cycle C visiting every vertex exactly once.

1. Find an MST, T of G.
2. Double every edge of the MST to obtain an Euler graph.
3. Find an Euler cycle, T_{eu} , on this graph.
4. Output the cycle that visits vertices of G in the order of their first appearance in T_{eu}. Let C be this cycle.

A 2-approximation algorithm for the Metric TSP

Input： A complete graph with nonnegative edge costs that satisfy the triangle inequality.
Ouptput: A cycle C visiting every vertex exactly once.

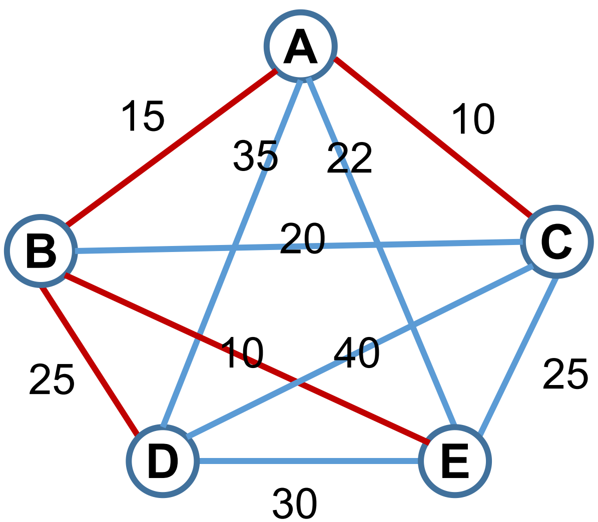
1. Find an MST, T of G.
2. Double every edge of the MST to obtain an Euler graph.
3. Find an Euler cycle, T_{eu} , on this graph.
4. Output the cycle that visits vertices of G in the order of their first appearance in T_{eu}. Let C be this cycle.



A 2-approximation algorithm for the Metric TSP

Input: A complete graph with nonnegative edge costs that satisfy the triangle inequality.
Ouptput: A cycle C visiting every vertex exactly once.

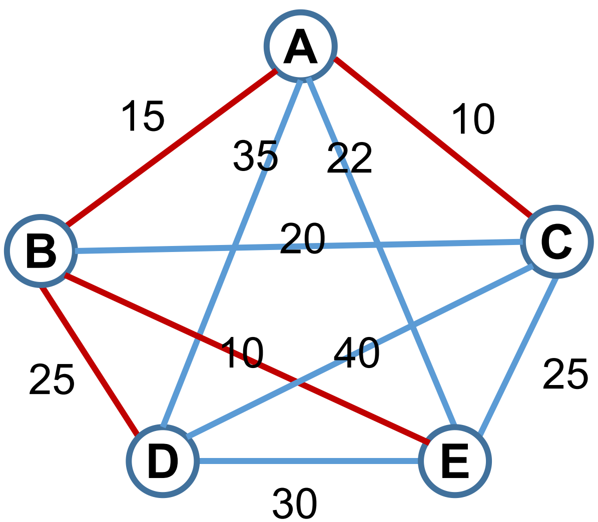
1. Find an MST, T of G.
2. Double every edge of the MST to obtain an Euler graph.
3. Find an Euler cycle, T_{eu} , on this graph.
4. Output the cycle that visits vertices of G in the order of their first appearance in T_{eu}. Let C be this cycle.

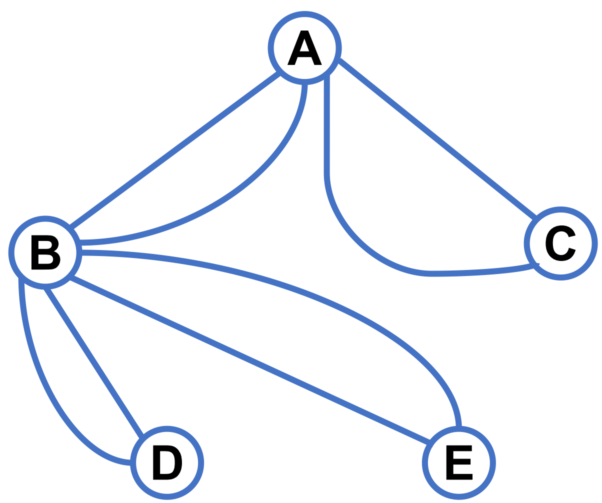


A 2-approximation algorithm for the Metric TSP

Input： A complete graph with nonnegative edge costs that satisfy the triangle inequality.
Ouptput: A cycle C visiting every vertex exactly once.

1. Find an MST, T of G.
2. Double every edge of the MST to obtain an Euler graph.
3. Find an Euler cycle, T_{eu} , on this graph.
4. Output the cycle that visits vertices of G in the order of their first appearance in T_{eu}. Let C be this cycle.





A 2-approximation algorithm for the Metric TSP

Input： A complete graph with nonnegative edge costs that satisfy the triangle inequality.
Ouptput: A cycle C visiting every vertex exactly once.

1. Find an MST, T of G.
2. Double every edge of the MST to obtain an Euler graph.
3. Find an Euler cycle, T_{eu} , on this graph.
4. Output the cycle that visits vertices of G in the order of their first appearance in T_{eu}. Let C be this cycle.

Proof