Chapter 9 Trees 树

Lu Han

hl@bupt.edu.cn

9.1 Introduction 简介

Semifinals and finals of classic tennis competition Wimbledon.

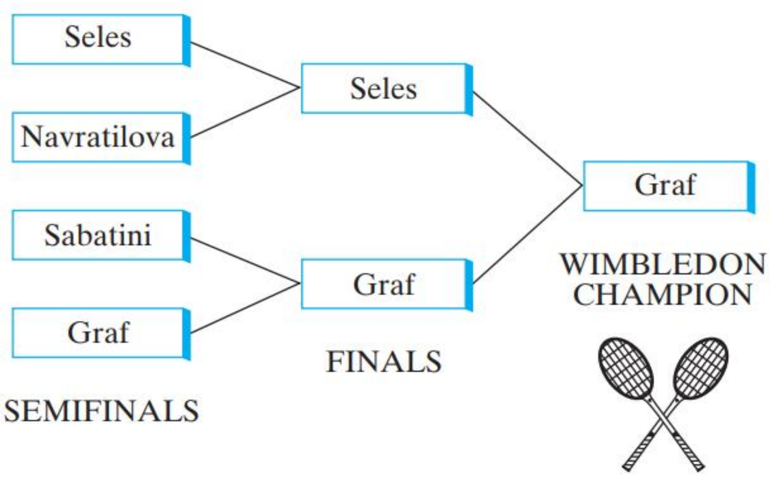


Figure 9.1.1 Semifinals and finals at Wimbledon.

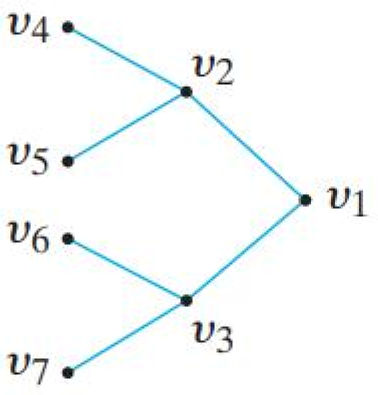


Figure 9.1.2 The tournament of Figure 9.1.1 as a tree.

9.1 Introduction 简介

Semifinals and finals of classic tennis competition Wimbledon.

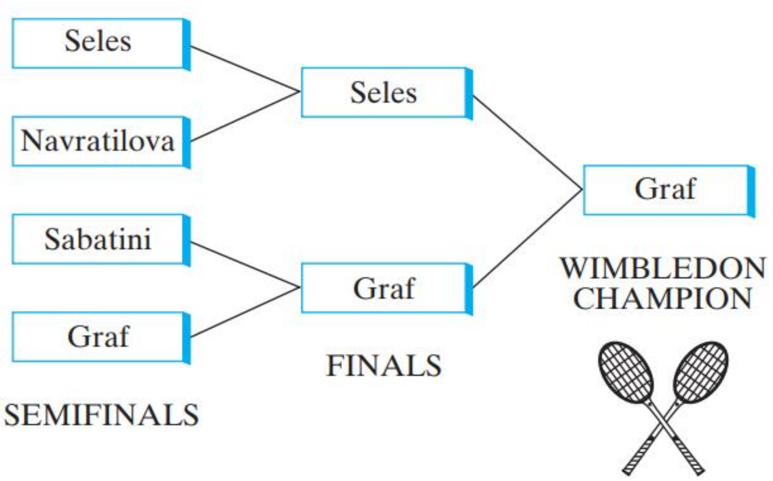


Figure 9.1.1 Semifinals and finals at Wimbledon.

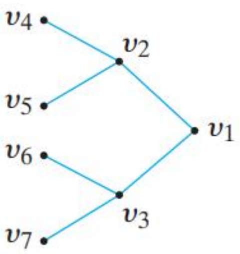


Figure 9.1.2 The tournament of Figure 9.1.1 as a tree.

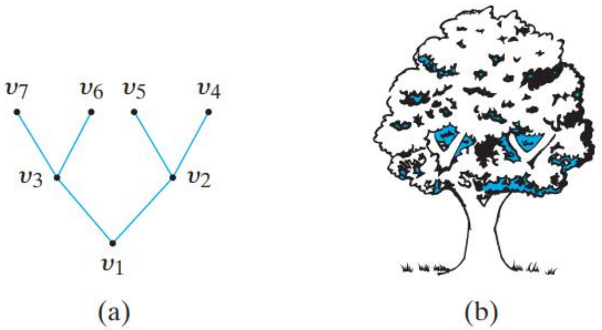
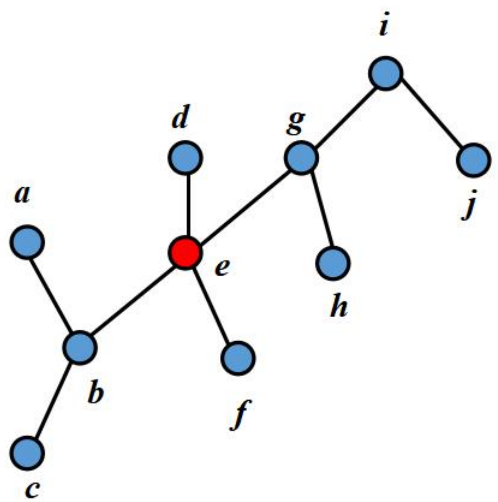


Figure 9.1.3 The tree of Figure 9.1.2 rotated (a) compared with a natural tree (b).

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.



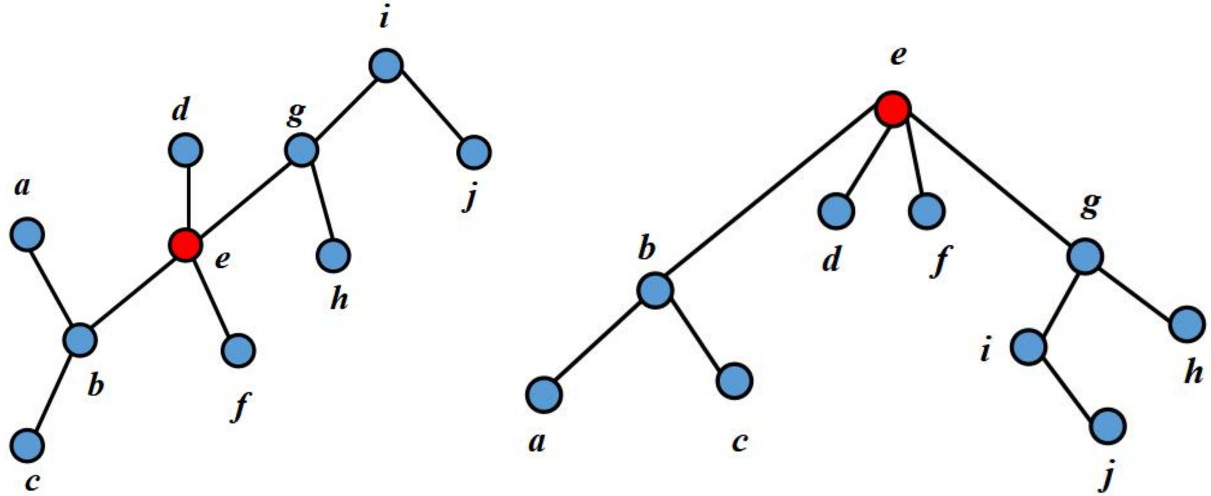
Designate e as the root in the tree.

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.

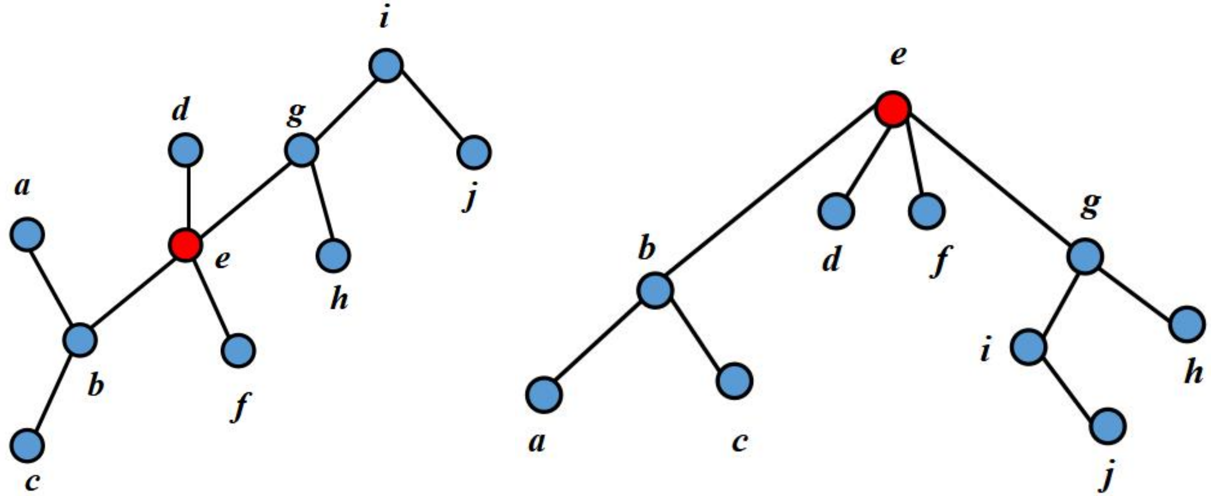


9.1 Introduction 简介

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.

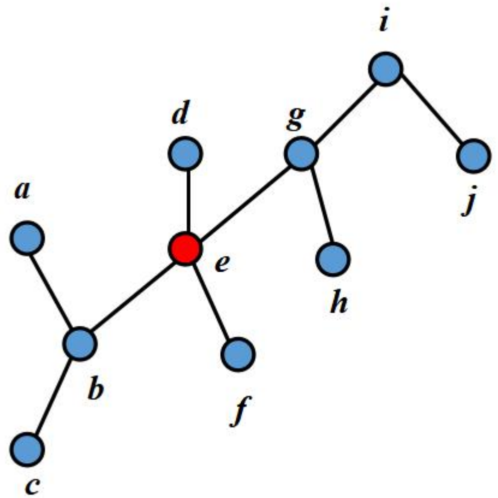
9.1 Introduction 简介

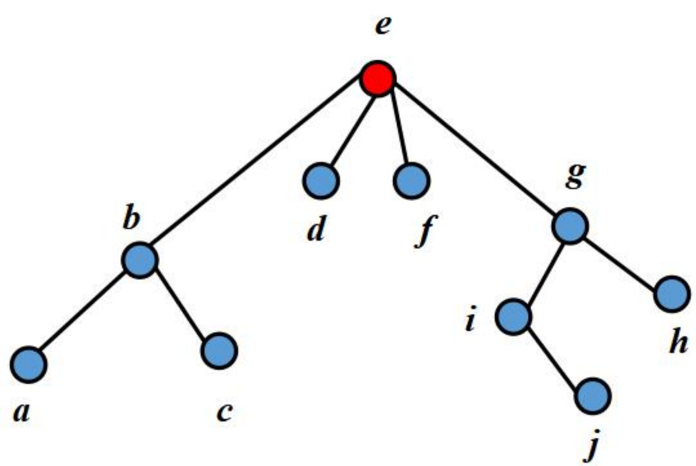
The **level of a vertex v (顶点 v 所在的层次)**: the length of the simple path from the root to v.

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.

Huffman codes 霍夫曼编码

- Huffman coding is a form of statistical coding
- Not all characters occur with the same frequency

★ Use less bits to encode characters in general
★ a character that appears often should be encoded with few bits
★ on the contrary, a character that does not appear often should be encoded with more bits

TABLE 9.1.1 ■ A Portion of the ASCII Table

Character	*ASCII Code*
A	100 0001
B	100 0010
C	100 0011
1	011 0001
2	011 0010
!	010 0001
*	010 1010

- Compression without loss (e.g. zip, rar)
- Proposed by David A. Huffman in 1952: “A Method for the Construction of Minimum Redundancy Codes”
- bmp (bitmap) -> jpg : Compression with loss

Huffman codes 霍夫曼编码

Algorithm

1. Scan text to be compressed and count occurrence of all characters
2. Sort or prioritize characters based on number of occurrences in text
3. Build Huffman code tree based on prioritized list
4. Perform a traversal of tree to determine all code words
5. Scan text again and create new file using the Huffman codes

Huffman codes 霍夫曼编码

Algorithm

1. Scan text to be compressed and count occurrence of all characters
2. Sort or prioritize characters based on number of occurrences in text
3. Build Huffman code tree based on prioritized list
4. Perform a traversal of tree to determine all code words
5. Scan text again and create new file using the Huffman codes

Eerie eyes seen near lake.

E	e	r	i	y	s	n	a	l	k	“space”	.
1	8	2	1	1	2	2	2	1	1	4	1

Huffman codes 霍夫曼编码

Algorithm

1. Scan text to be compressed and count occurrence of all characters
2. Sort or prioritize characters based on number of occurrences in text
3. Build Huffman code tree based on prioritized list
4. Perform a traversal of tree to determine all code words
5. Scan text again and create new file using the Huffman codes

Eerie eyes seen near lake.

E	i	y	l	k	.	r	s	n	a	“space”	e
1	1	1	1	1	1	2	2	2	2	4	8

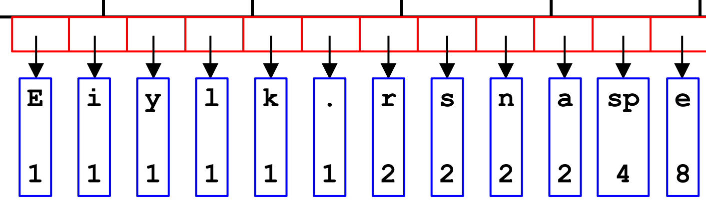
Huffman codes 霍夫曼编码

Algorithm

1. Scan text to be compressed and count occurrence of all characters
2. Sort or prioritize characters based on number of occurrences in text
3. Build Huffman code tree based on prioritized list
4. Perform a traversal of tree to determine all code words
5. Scan text again and create new file using the Huffman codes

Eerie eyes seen near lake.

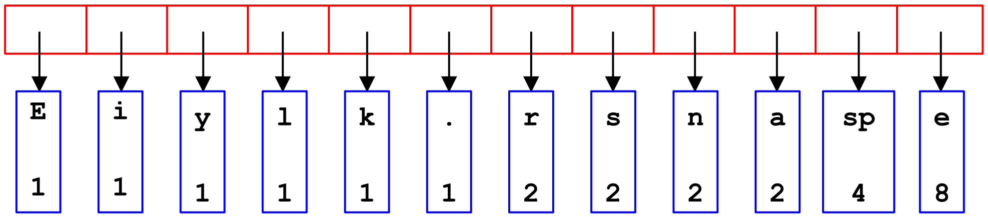
E	i	y	l	k	.	r	s	n	a	“space”	e
1	1	1	1	1	1	2	2	2	2	4	8



Huffman codes 霍夫曼编码

Building a tree

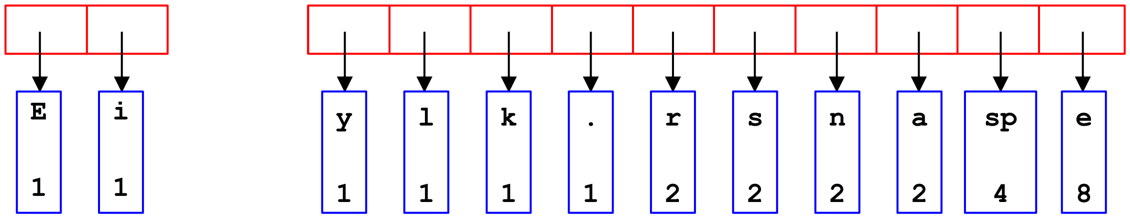
- Create new node
- Dequeue node and make it left subtree
- Dequeue next node and make it right subtree
- Frequency of new node equals sum of frequency of left and right children
- Enqueue new node back into queue



Huffman codes 霍夫曼编码

Building a tree

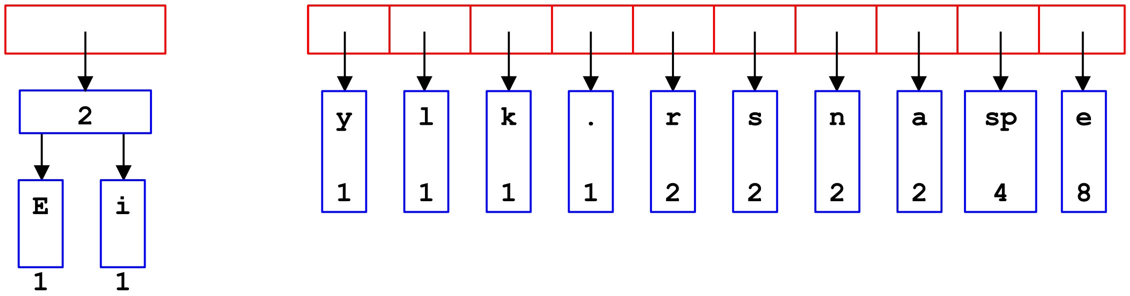
- Create new node
- Dequeue node and make it left subtree
- Dequeue next node and make it right subtree
- Frequency of new node equals sum of frequency of left and right children
- Enqueue new node back into queue



Huffman codes 霍夫曼编码

Building a tree

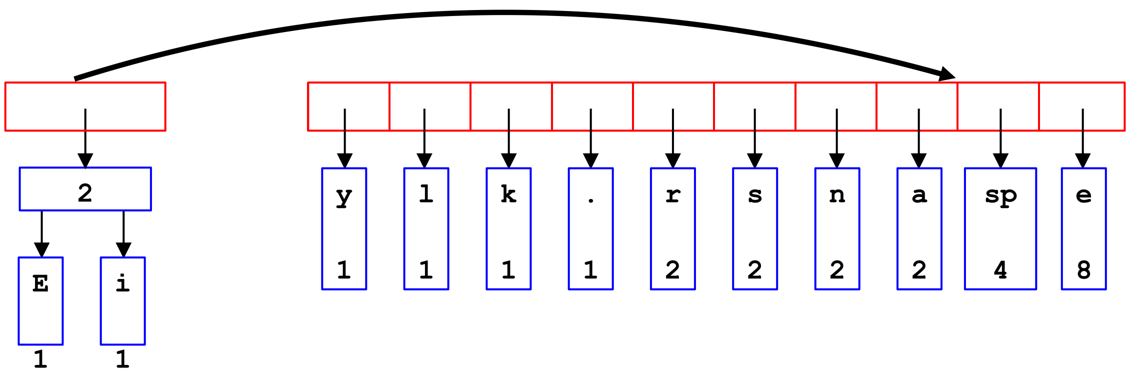
- Create new node
- Dequeue node and make it left subtree
- Dequeue next node and make it right subtree
- Frequency of new node equals sum of frequency of left and right children
- Enqueue new node back into queue



Huffman codes 霍夫曼编码

Building a tree

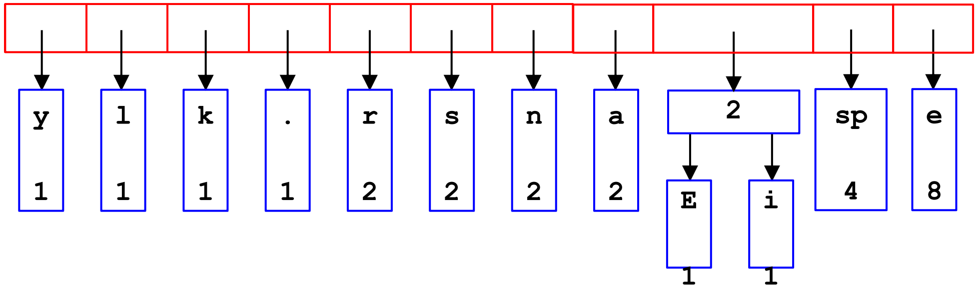
- Create new node
- Dequeue node and make it left subtree
- Dequeue next node and make it right subtree
- Frequency of new node equals sum of frequency of left and right children
- Enqueue new node back into queue



Huffman codes 霍夫曼编码

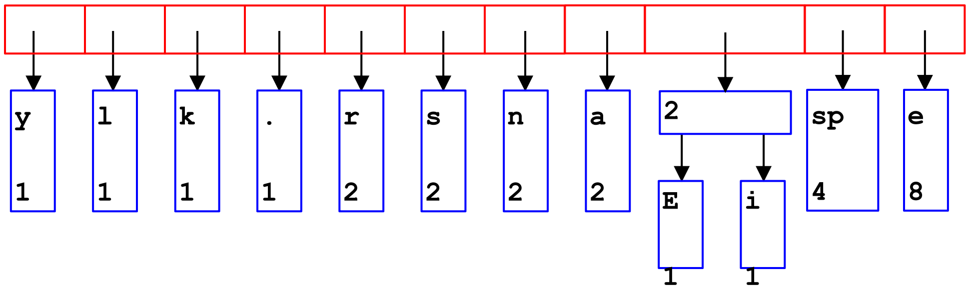
Building a tree

- Create new node
- Dequeue node and make it left subtree
- Dequeue next node and make it right subtree
- Frequency of new node equals sum of frequency of left and right children
- Enqueue new node back into queue



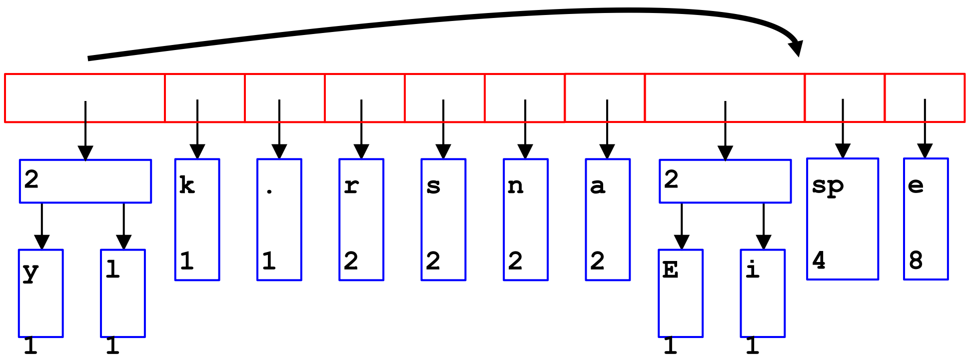
Huffman codes 霍夫曼编码

Building a tree



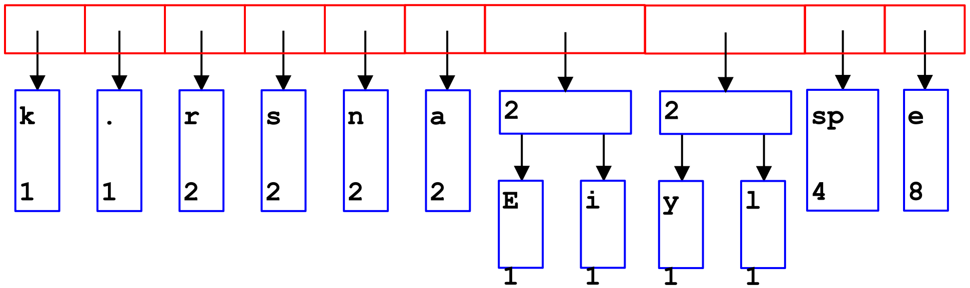
Huffman codes 霍夫曼编码

Building a tree



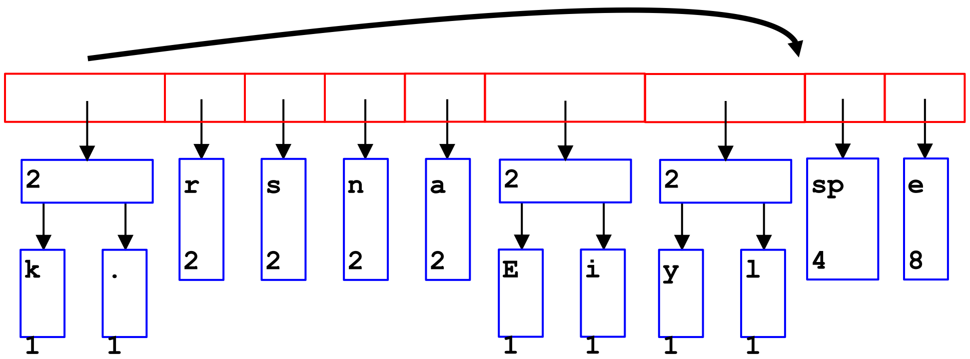
Huffman codes 霍夫曼编码

Building a tree



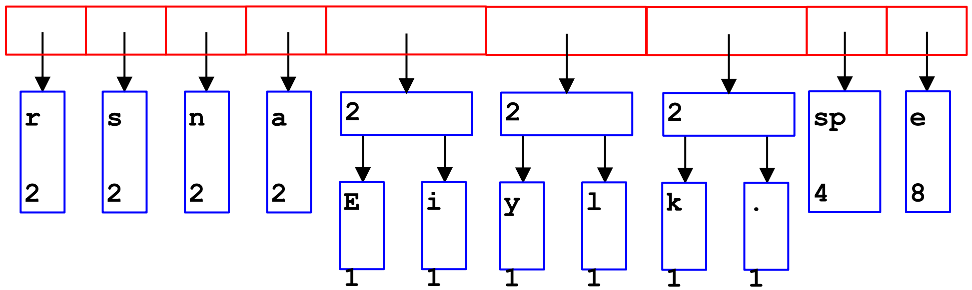
Huffman codes 霍夫曼编码

Building a tree



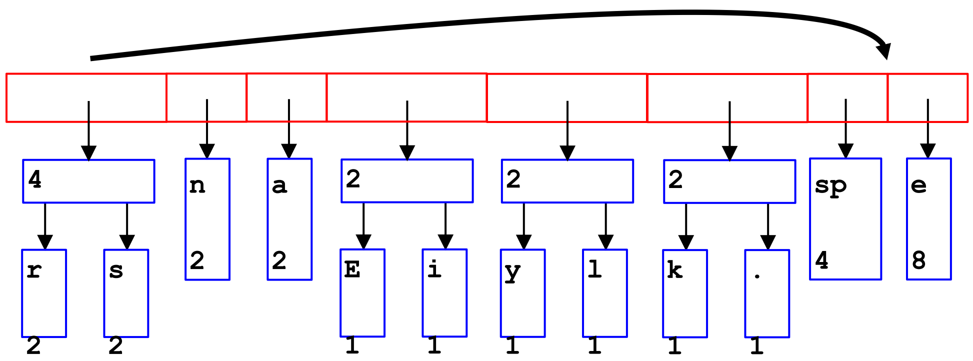
Huffman codes 霍夫曼编码

Building a tree



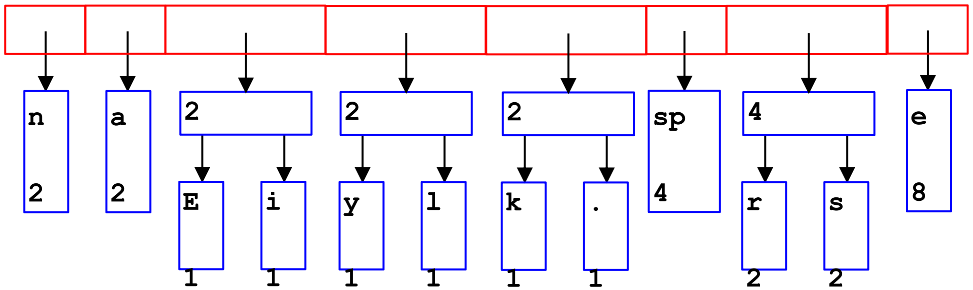
Huffman codes 霍夫曼编码

Building a tree



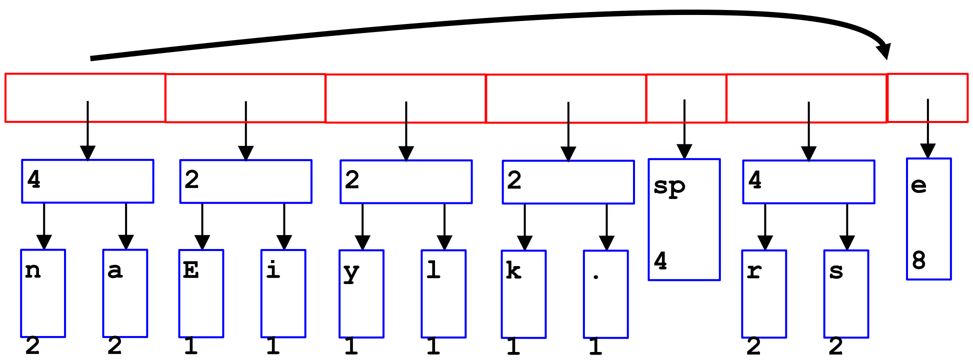
Huffman codes 霍夫曼编码

Building a tree



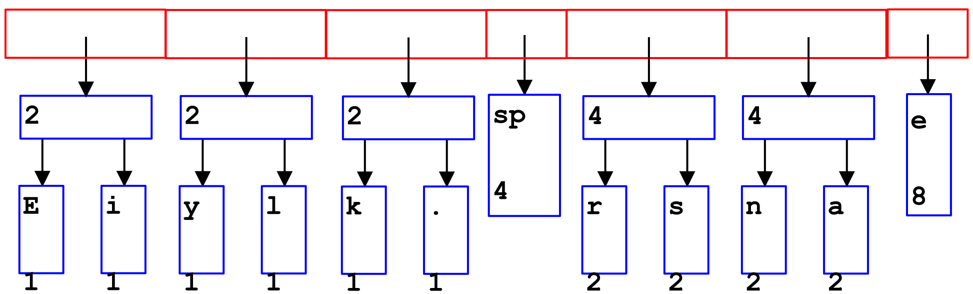
Huffman codes 霍夫曼编码

Building a tree



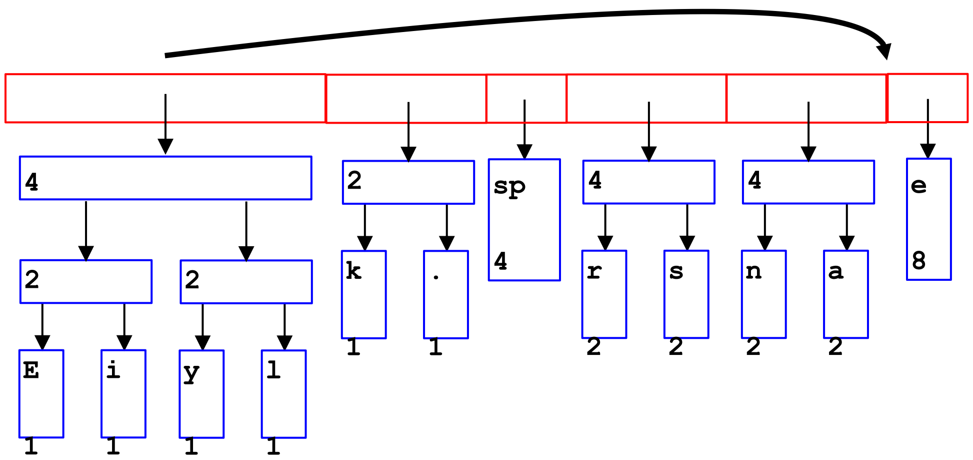
Huffman codes 霍夫曼编码

Building a tree



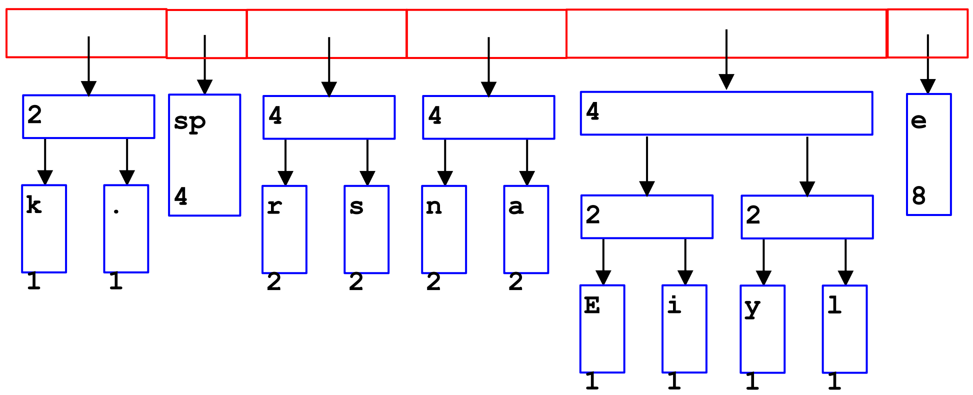
Huffman codes 霍夫曼编码

Building a tree



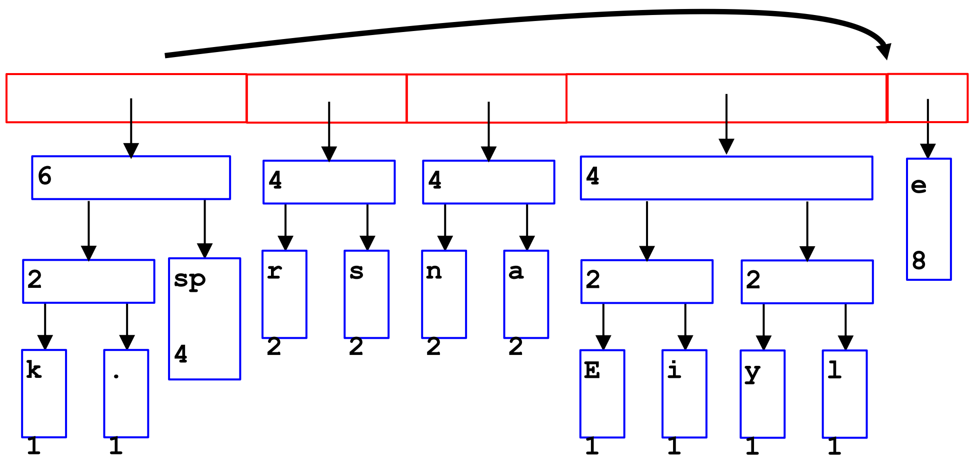
Huffman codes 霍夫曼编码

Building a tree



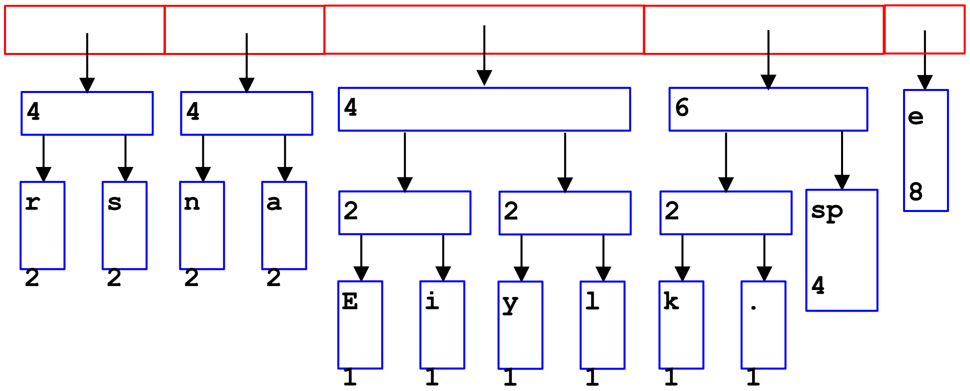
Huffman codes 霍夫曼编码

Building a tree



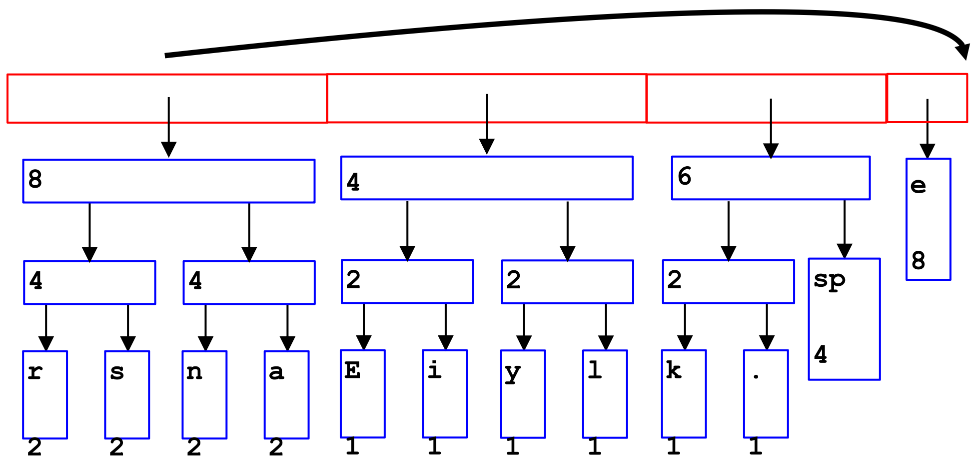
Huffman codes 霍夫曼编码

Building a tree



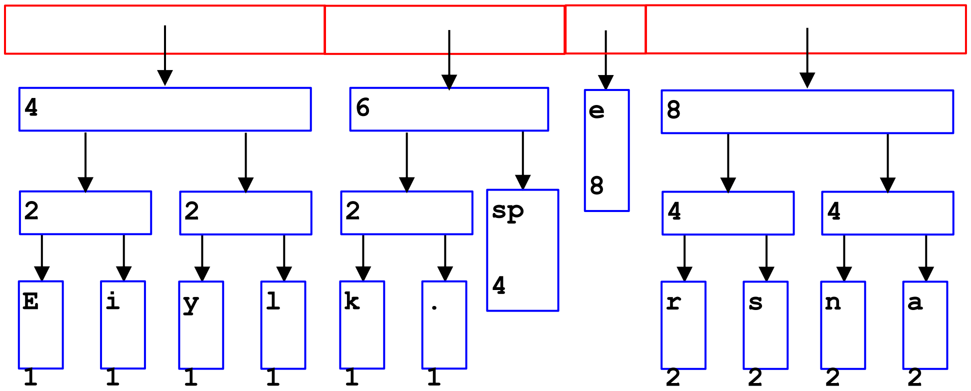
Huffman codes 霍夫曼编码

Building a tree



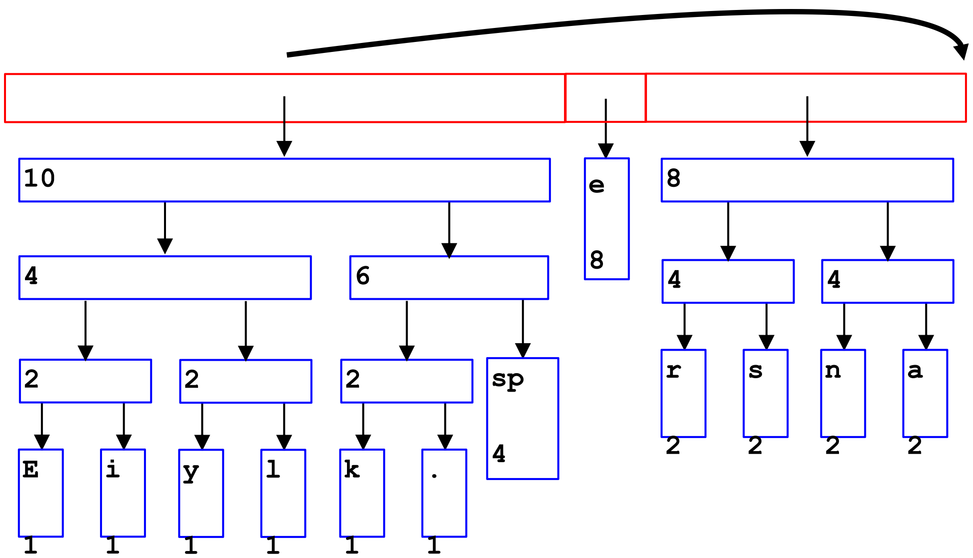
Huffman codes 霍夫曼编码

Building a tree



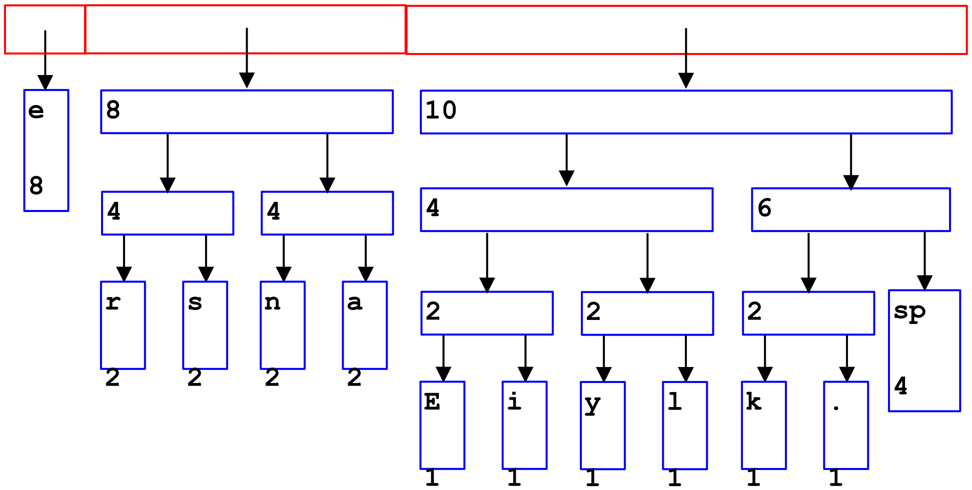
Huffman codes 霍夫曼编码

Building a tree



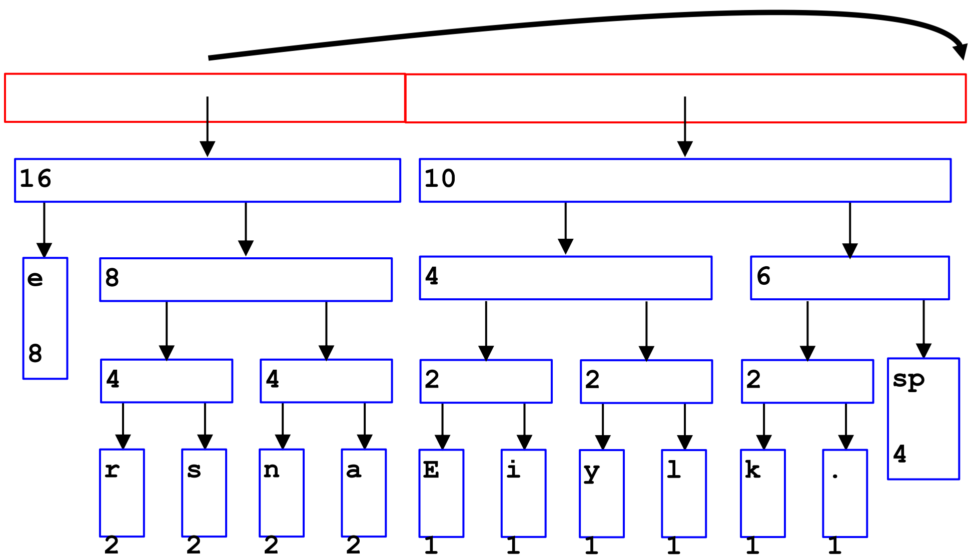
Huffman codes 霍夫曼编码

Building a tree



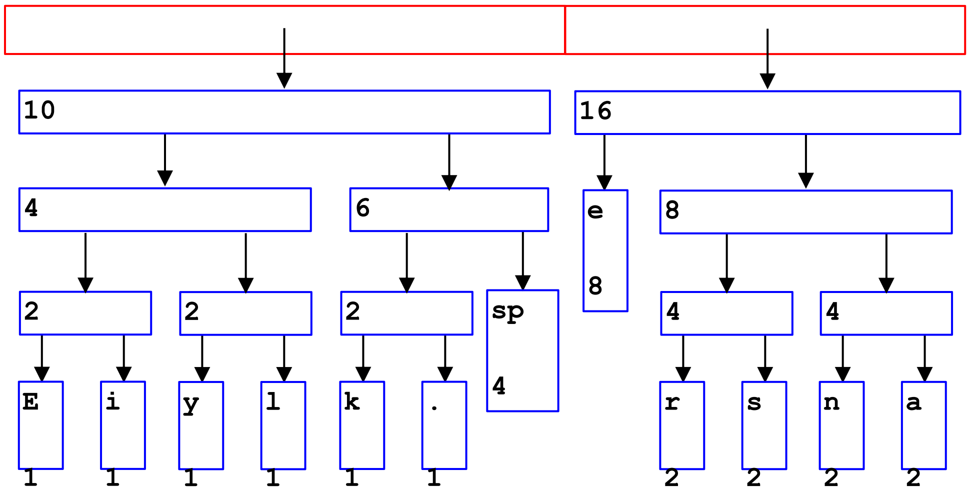
Huffman codes 霍夫曼编码

Building a tree



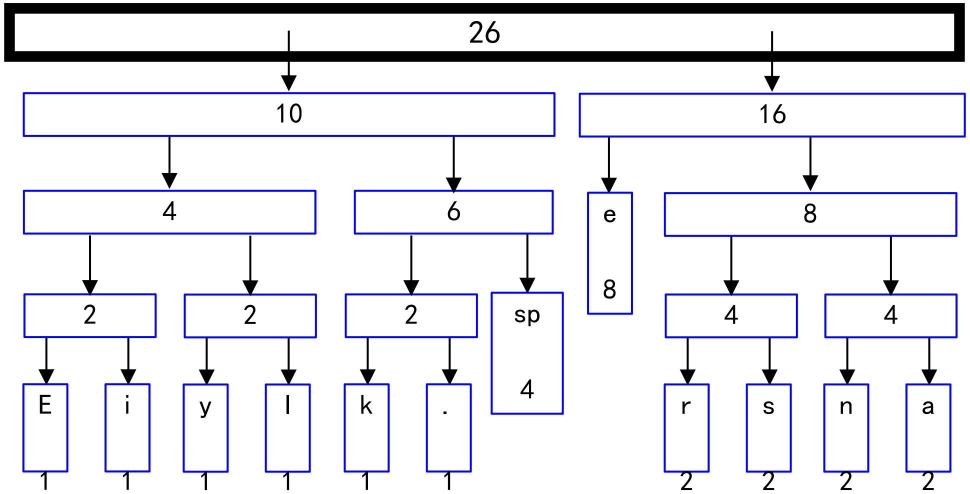
Huffman codes 霍夫曼编码

Building a tree



Huffman codes 霍夫曼编码

Building a tree

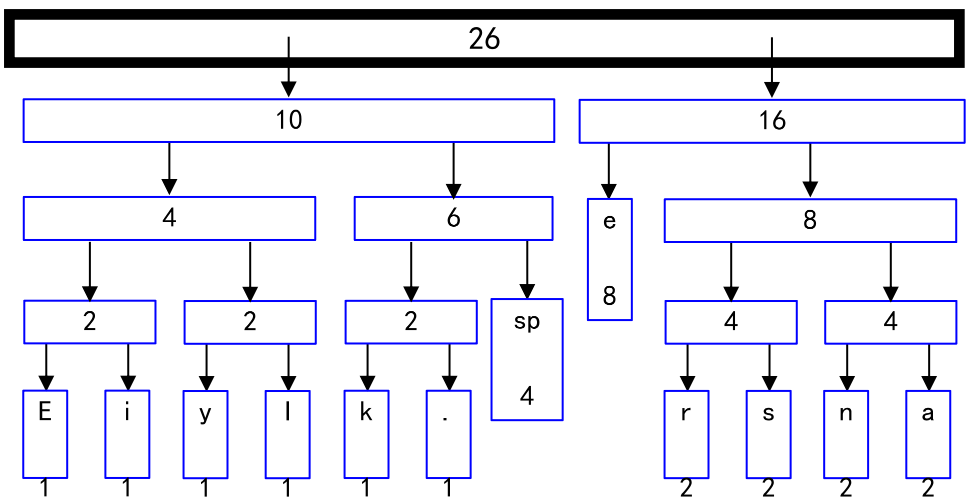


Huffman codes 霍夫曼编码

Algorithm

1. Scan text to be compressed and count occurrence of all characters
2. Sort or prioritize characters based on number of occurrences in text
3. Build Huffman code tree based on prioritized list
4. Perform a traversal of tree to determine all code words
5. Scan text again and create new file using the Huffman codes

- Perform a traversal of the tree to obtain new code words.
- Going left is a 0 going right is a 1.
- Code word is only completed when a leaf node is reached.

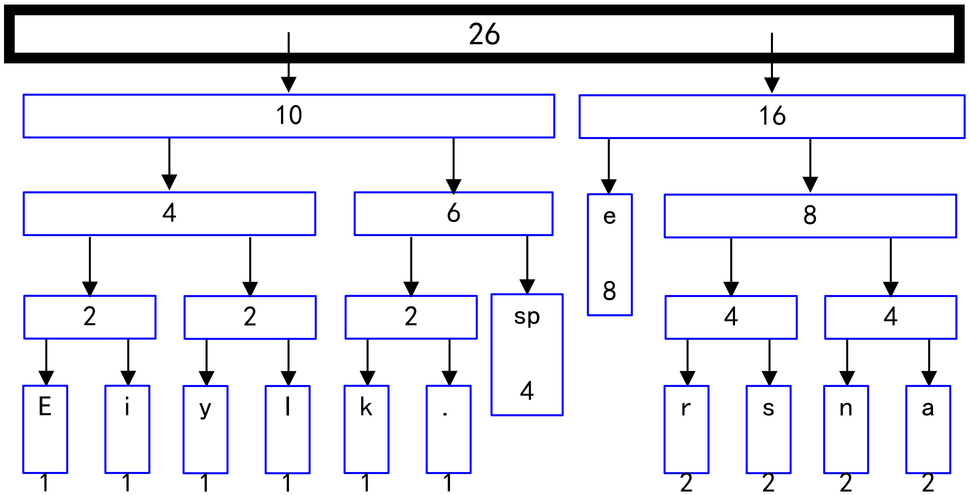


Huffman codes 霍夫曼编码

Algorithm

1. Scan text to be compressed and count occurrence of all characters
2. Sort or prioritize characters based on number of occurrences in text
3. Build Huffman code tree based on prioritized list
4. Perform a traversal of tree to determine all code words
5. Scan text again and create new file using the Huffman codes

Character	Code
E	0000
i	0001
y	0010
l	0011
k	0100
.	0101
"space"	011
e	10
r	1100
s	1101
n	1110
a	1111



Huffman codes 霍夫曼编码

Algorithm

1. Scan text to be compressed and count occurrence of all characters
2. Sort or prioritize characters based on number of occurrences in text
3. Build Huffman code tree based on prioritized list
4. Perform a traversal of tree to determine all code words
5. Scan text again and create new file using the Huffman codes

Character	Code
E	0000
i	0001
y	0010
l	0011
k	0100
.	0101
“space”	011
e	10
r	1100
s	1101
n	1110
a	1111

Eerie eyes seen near lake.

```
0000101100000110011100010101101101001111101011111100011001111110100100101
```

Huffman codes 霍夫曼编码

- ASCII would take 7 * 26 = 182 bits
- Naive encoding

  12 characters to encode

  4 bits are enough for each character

  4 * 26 = 104 bits

Eerie eyes seen near lake.

- Huffman encoding

  73 bits

  average bits per character: 73/26≈ 2.81 **+ Also need to store the dictionary!**

Huffman codes 霍夫曼编码

- ASCII would take 7 * 26 = 182 bits
- Naive encoding

  12 characters to encode

  4 bits are enough for each character

  4 * 26 = 104 bits
- Huffman encoding

  73 bits

  average bits per character: 73/26≈ 2.81 **+ Also need to store the dictionary!**

Shortcoming of Huffman codes?

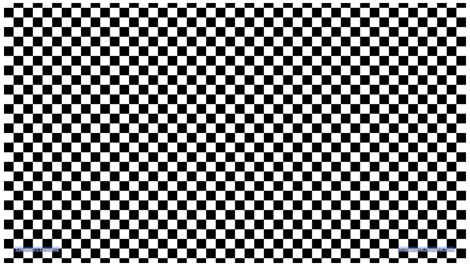
Huffman codes 霍夫曼编码

- ASCII would take 7 * 26 = 182 bits
- Naive encoding

  12 characters to encode

  4 bits are enough for each character

  4 * 26 = 104 bits



- Huffman encoding

  73 bits

  average bits per character: 73/26≈ 2.81 **+ Also need to store the dictionary!**

Shortcoming of Huffman codes?

It doesn’t work well for repeated pattern (for example alternate pixels picture).

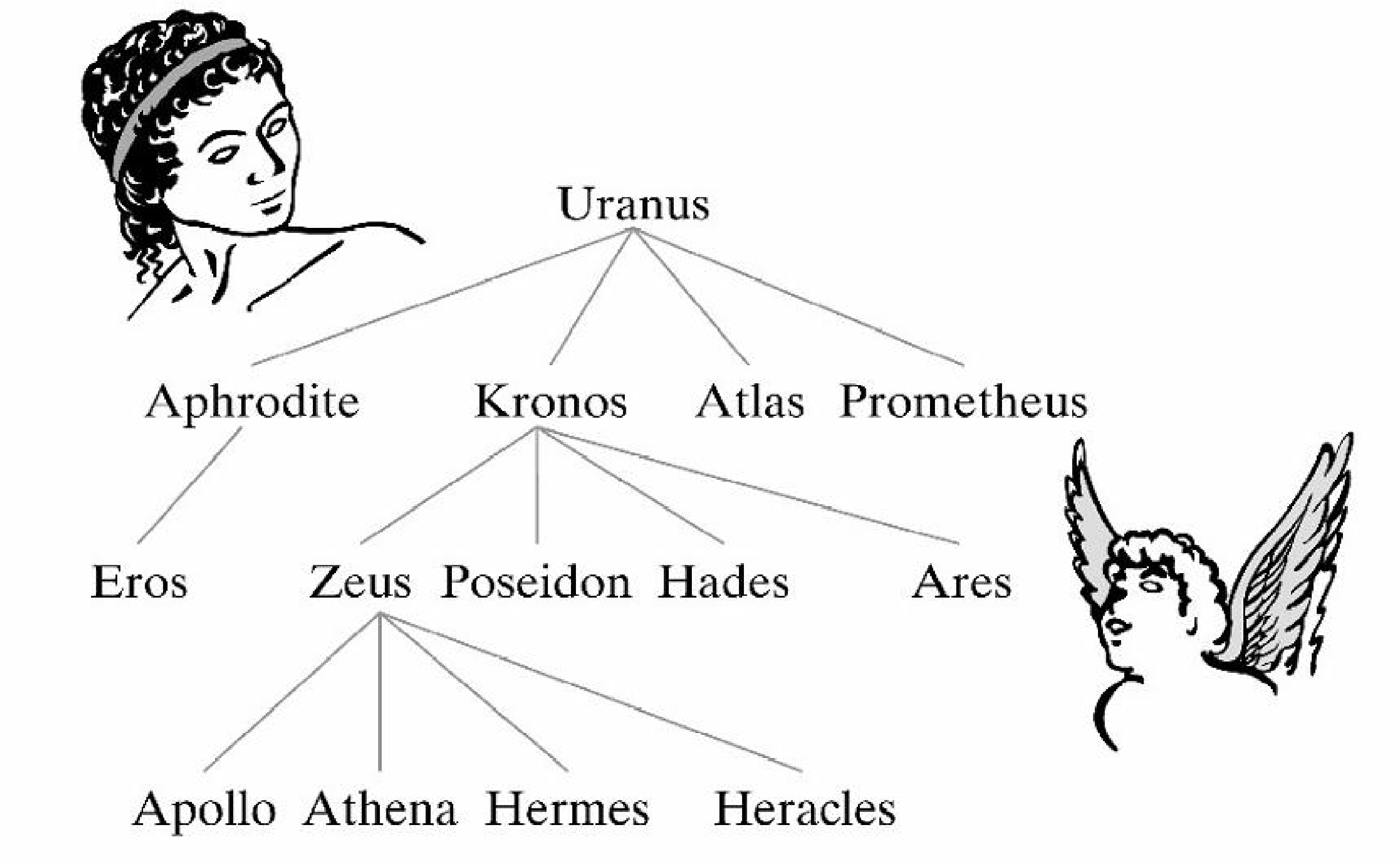
Huffman codes 霍夫曼编码

1. Encode the following text:

An illusory vision is a visionary illusion. Is it?

2. Give the corresponding encoding table
3. What is the average number of bits per character?

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



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)**.

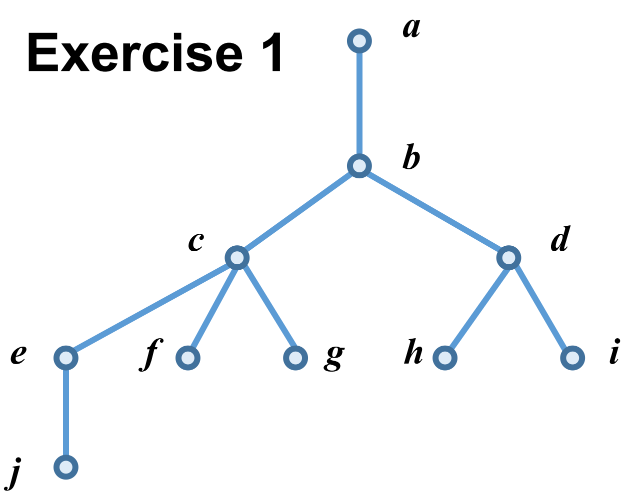
(f) If x has no children, x is a **terminal vertex (or a leaf) (终节点/叶节点)**.

(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

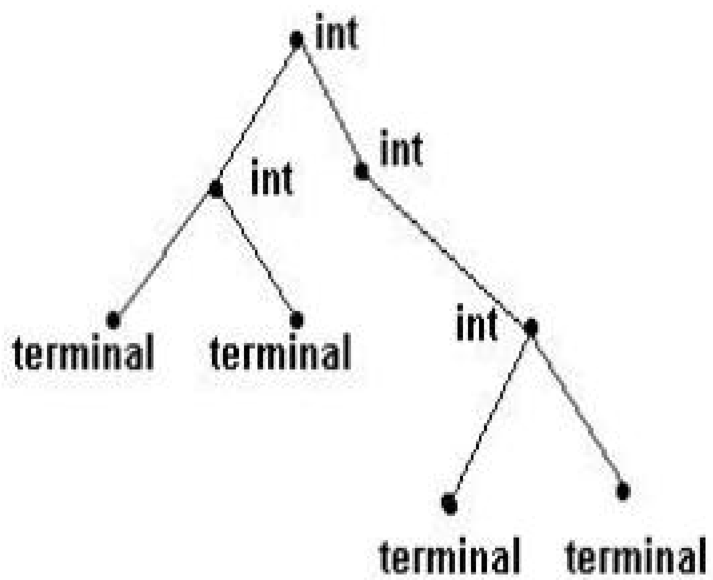
Definition 9.2.1 Let T be a tree with root v_0. Suppose that x, y, and z are vertices in T and that $(v_0, v_1, \ldots, v_n)$ is a simple path in T. Then
(a) v_{n-1} is the **parent (父节点)** of v_n.
(b) $v_0, \ldots, v_{n-1}$ are **ancestors (祖先节点)** of v_n.
(c) v_n is a **child (子节点)** of v_{n-1}.
(d) 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)**.

Exercise 1



(1) Find the parent of c.
(2) Find the children of c.
(3) Find the ancestors of c.
(4) Find the descendants of c.
(5) Find the siblings of e.
(6) Find the terminal vertices.
(7) Find the internal vertices.

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



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

- An **internal vertex (中间节点)** is a vertex that has at least one child.
- A **terminal vertex (终节点)** is a vertex that has no children

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)**.

Exercise 2

(1) What can you say about two vertices in a rooted tree that have the same parent?

(2) What can you say about a vertex in a rooted tree that has no ancestors?

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)**.

Exercise 2

(3) What can you say about a vertex in a rooted tree that has no descendants?

(4) What can you say about two vertices in a rooted tree that have a descendant in common?

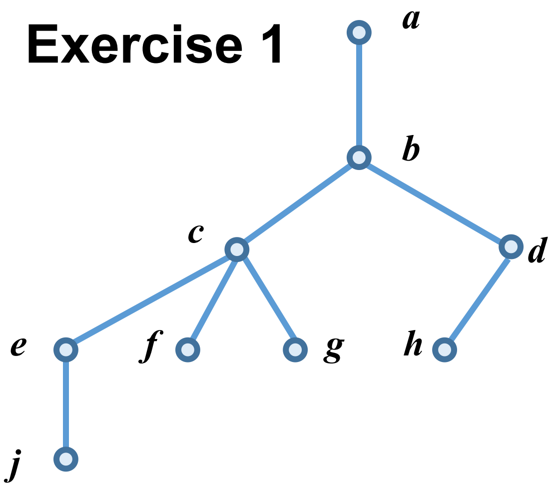
9.2 Terminology and Characterizations of Trees 树的术语和性质

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.

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

(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

(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

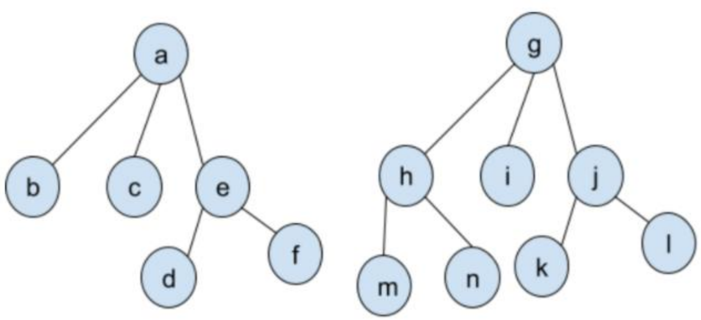
If T is a graph with n vertices, the following are equivalent (Theorem 9.2.3):
(a) T is a tree.
(b) If v and w are vertices in T, there is a unique simple path from v to w.
(c) T is connected and acyclic.
(d) T is connected and has $n - 1$ edges.
(e) T is acyclic and has $n - 1$ edges.

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.

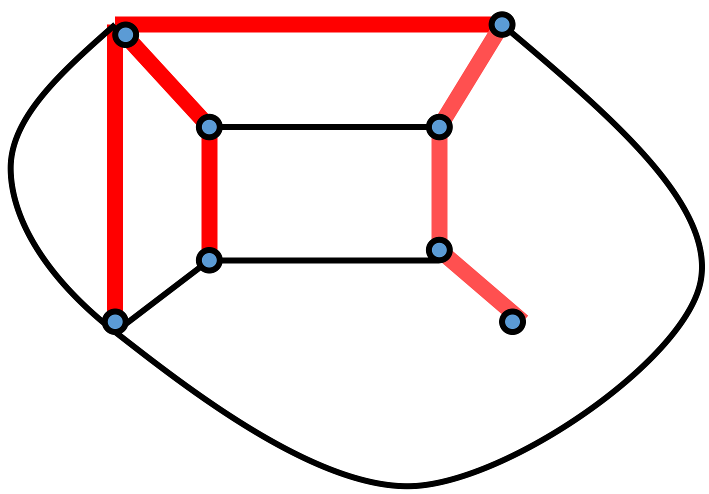
A **forest (森林)** is a simple graph with no cycles.



Forest

9.3 Spanning Trees 生成树

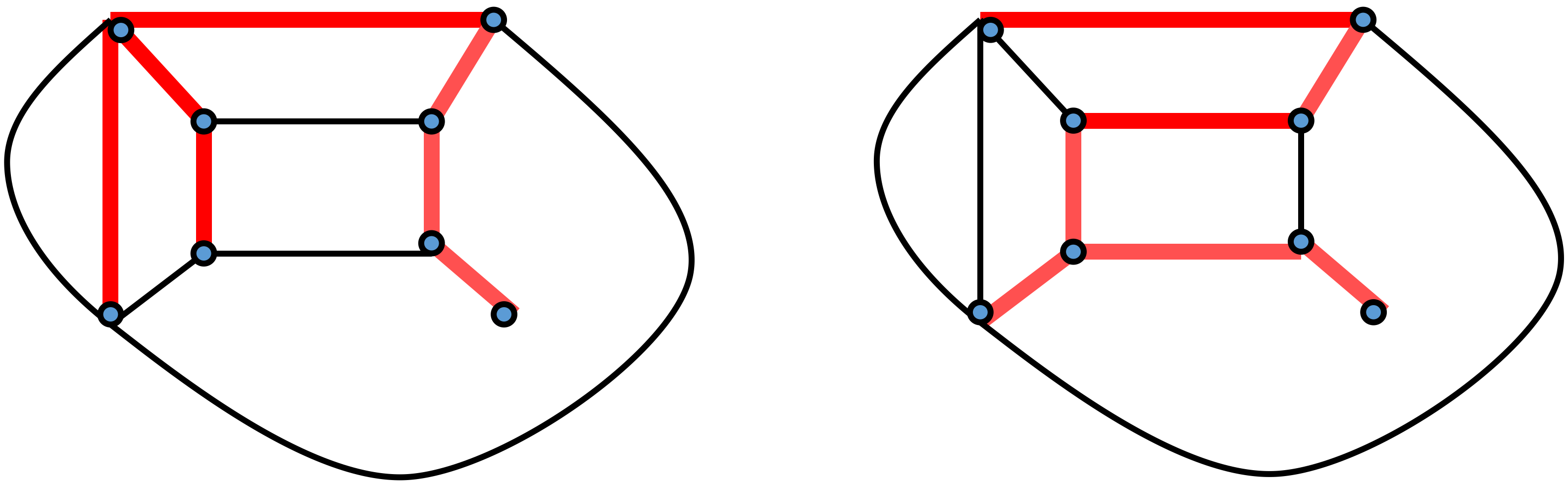
Definition 9.3.1 A tree T is a **spanning tree (生成树)** of a graph G if T is a subgraph of G that contains all of the vertices of G.



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.

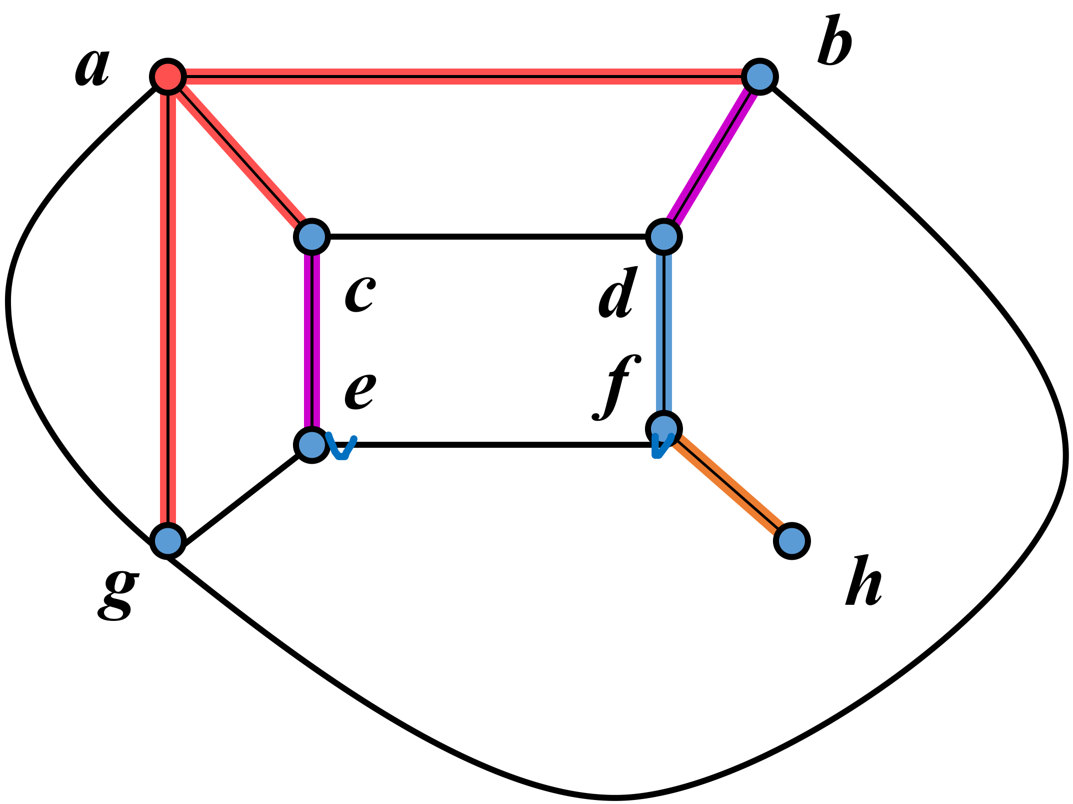
9.3 Spanning Trees 生成树

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
  }
}
```

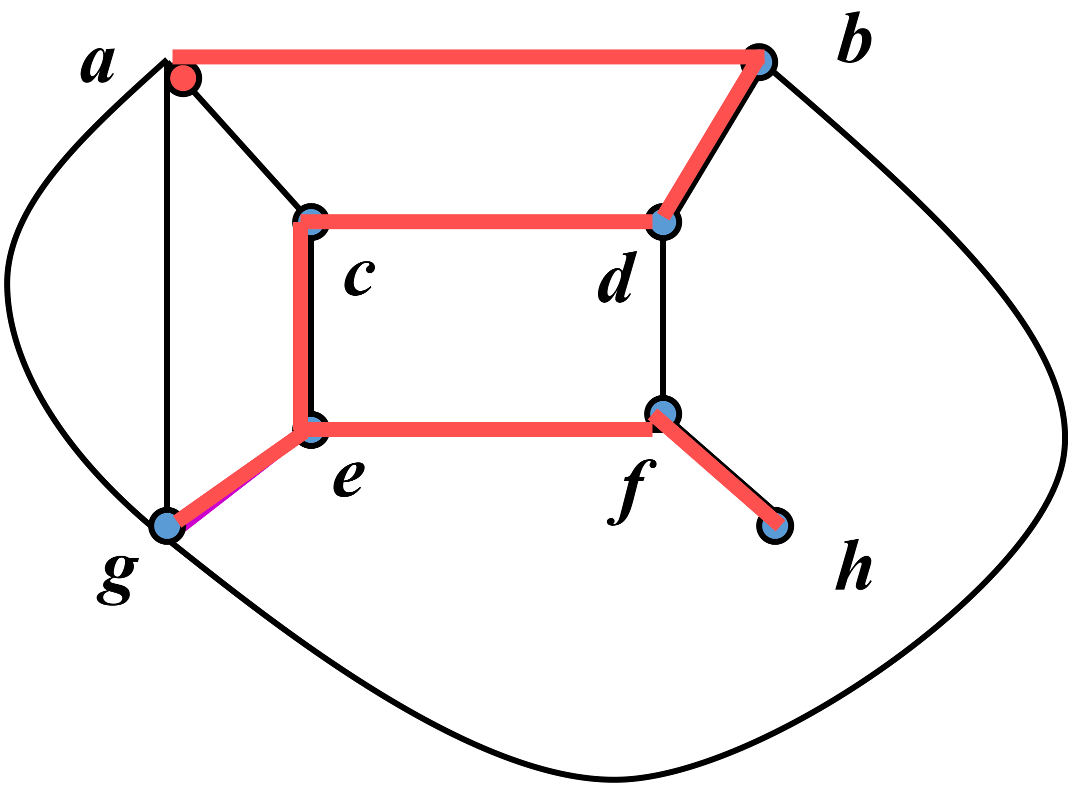
9.3 Spanning Trees 生成树

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

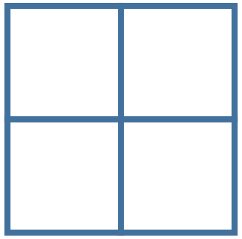
```
dfs(V, E) {
  // V′ = vertices of spanning tree T; E′ = edges of spanning tree T
  // v1 is the root of the spanning tree
  V′ = {v1}
  E′ = ∅
  w = v1
  while (true) {
    while (there is an edge (w, v) that when added to T does not create a cycle
      in T) {
      choose the edge (w, vk) with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

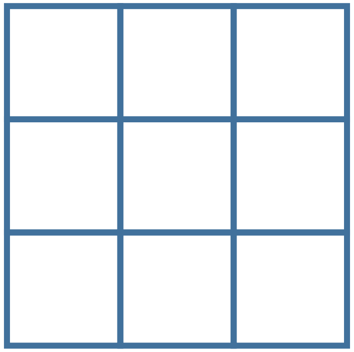
9.3 Spanning Trees 生成树

Four-Queens Problem 4皇后问题

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



Two-Queens Problem

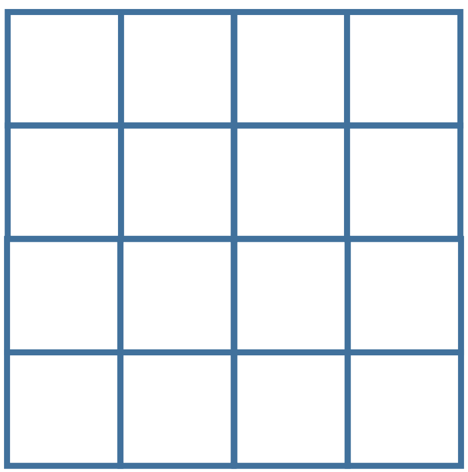


Three-Queens Problem

9.3 Spanning Trees 生成树

Four-Queens Problem 4皇后问题

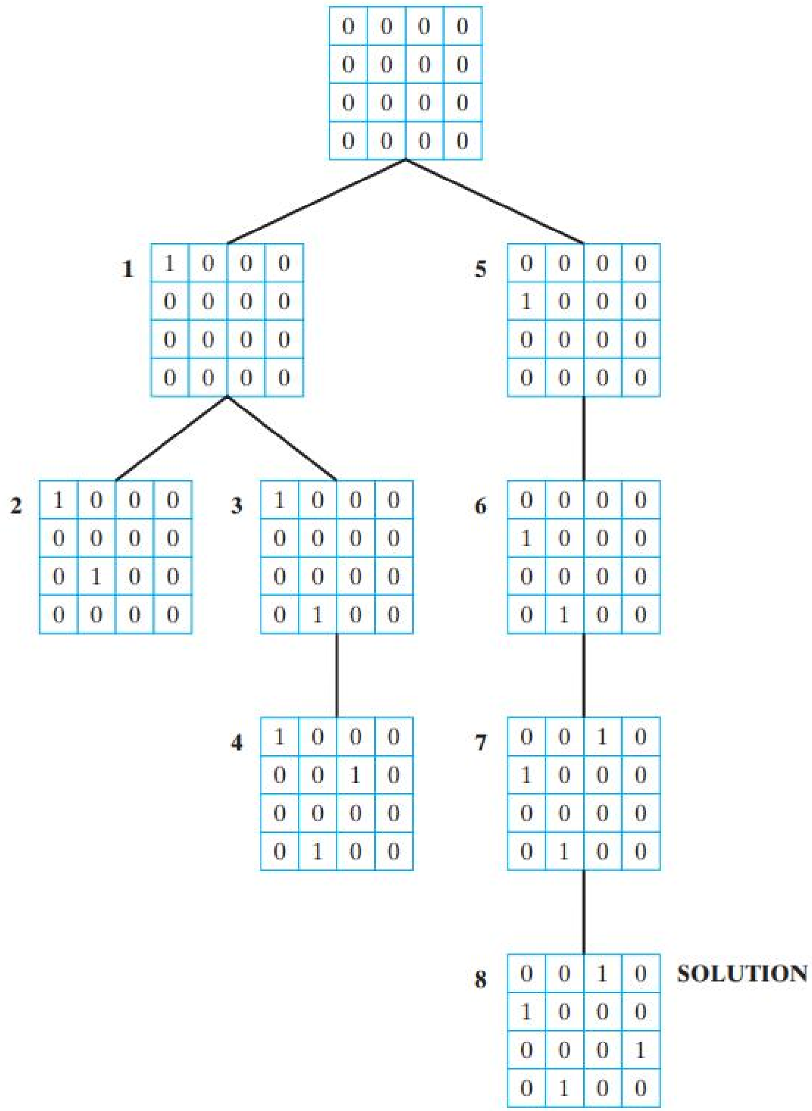
To place for tokens on a 4×4 grid so thta no two tokens are on the same row, column, or diagonal.



9.3 Spanning Trees 生成树

Four-Queens Problem 4皇后问题

		Q	
Q			
			Q
	Q		

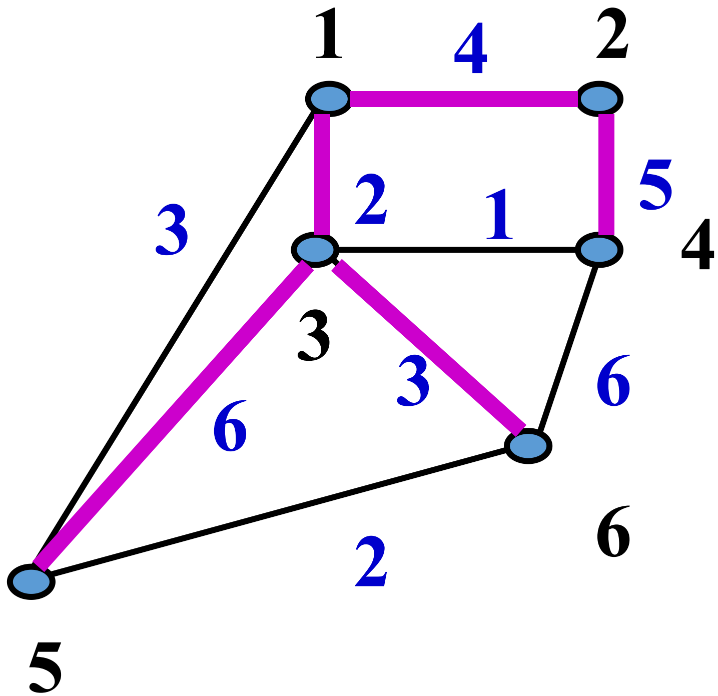


9.4 Minimal Spanning Trees 最小生成树

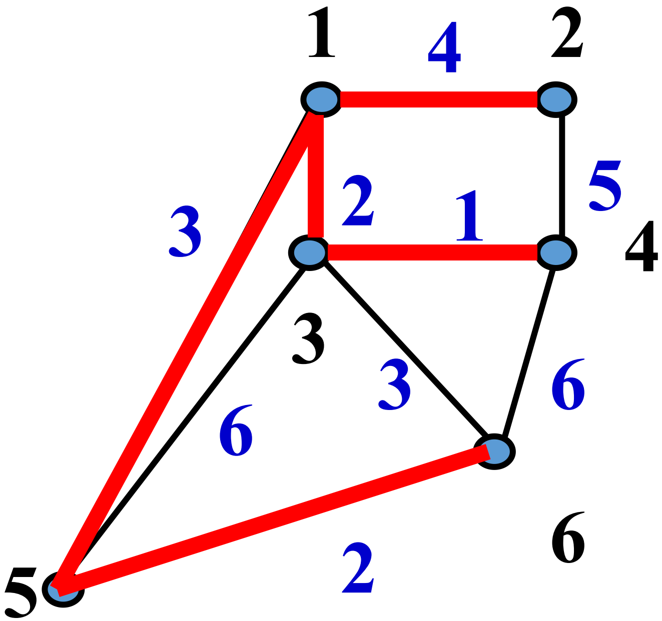
Definition 9.4.1 Given a weighted graph G, a **minimum spanning tree (最小生成树)** of G is a spanning tree of G that has minimum weight.

9.4 Minimal Spanning Trees 最小生成树

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



Weight=20

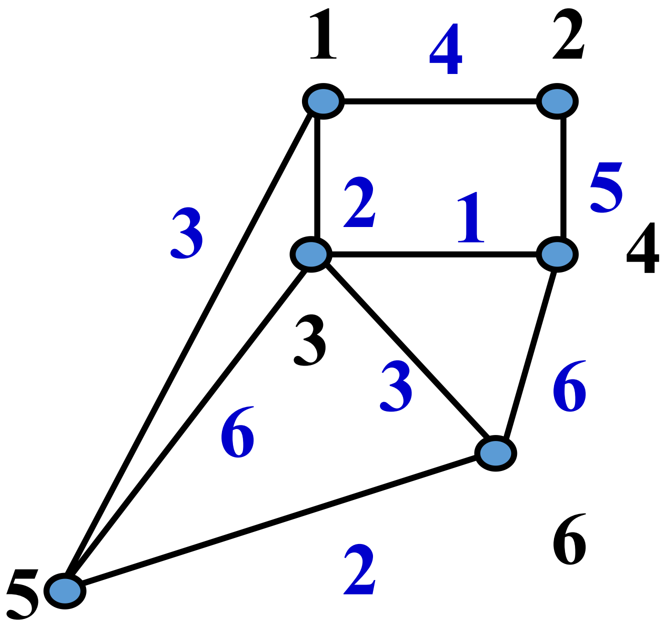


Weight=12

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.

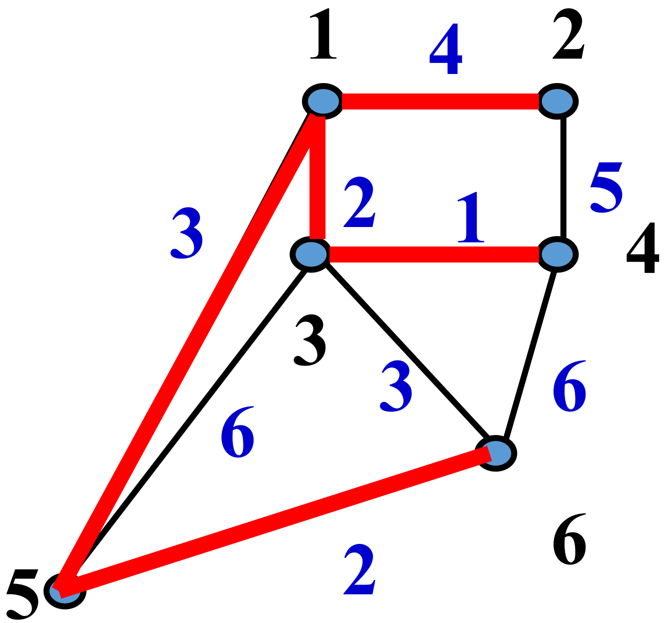


Keep finding the minimum-weight edge with one vertex in the tree and one vertex not in the tree.

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.

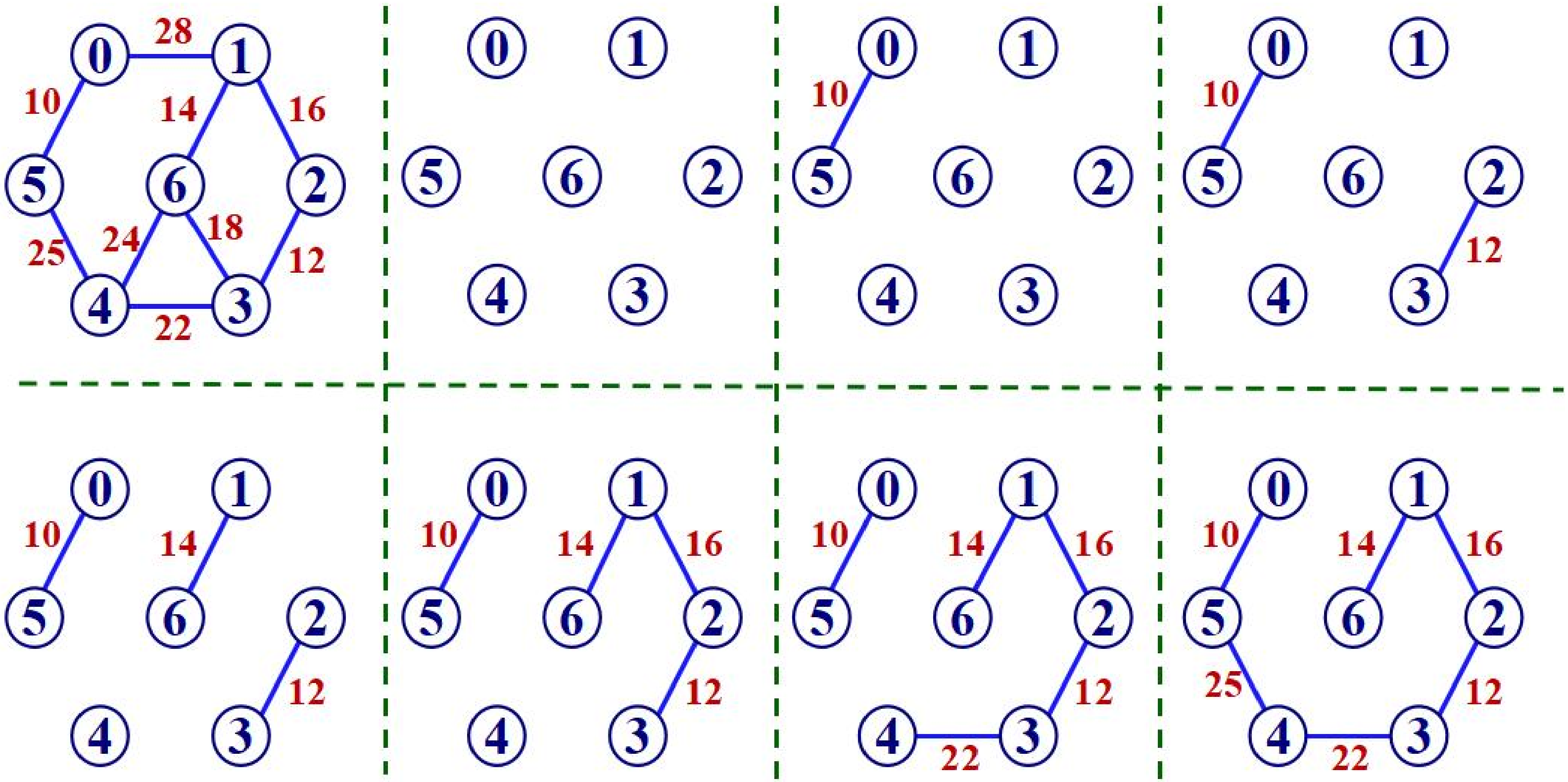
9.4 Minimal Spanning Trees 最小生成树

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.

9.4 Minimal Spanning Trees 最小生成树

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.

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 $\Rightarrow$ 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

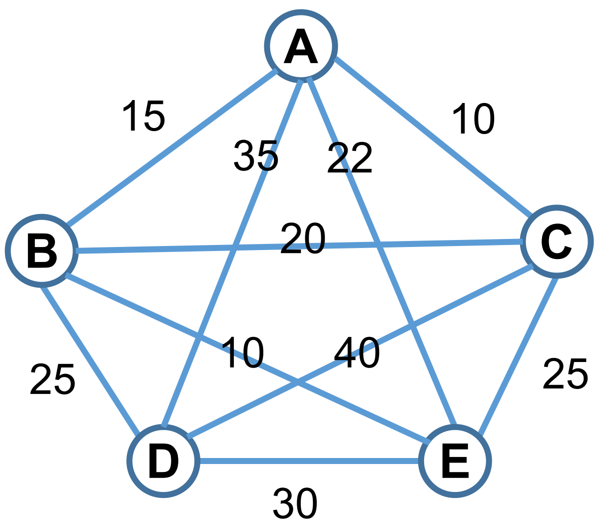
Input：A completem graph with nonnegative edge costs that satisfy the triangle inequality.
Ouptput: A cycle C visiting every vertex exactly once.

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 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 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.

Proof