Lecture 5

Hashing (contd.), Binary Search Trees

Independent Uniform Hashing

Independent Uniform Hashing

Suppose every element of the dynamic set has a distinct key from the universe

$U = \{0, 1, \ldots, n-1\}$

Independent Uniform Hashing

Suppose every element of the dynamic set has a distinct key from the universe $U = \{0,1,\ldots,n-1\}$ and chained hash table is $T[0:m-1]$, where $m < n$.

Independent Uniform Hashing

Suppose every element of the dynamic set has a distinct key from the universe $U = \{0,1,\ldots,n-1\}$ and chained hash table is $T[0:m-1]$, where $m < n$.

Defn: In independent uniform hashing, the hash function $h : U \to \{0,1,\ldots,m-1\}$ works in the following manner:

Independent Uniform Hashing

Suppose every element of the dynamic set has a distinct key from the universe $U = \{0,1,\ldots,n-1\}$ and chained hash table is $T[0:m-1]$, where $m < n$.

Defn: In independent uniform hashing, the hash function $h : U \to \{0,1,\ldots,m-1\}$ works in the following manner:

- For each key k, $h(k)$ is independently chosen uniformly randomly from $\{0,1,\ldots,m-1\}$.

Independent Uniform Hashing

Suppose every element of the dynamic set has a distinct key from the universe $U = \{0,1,\ldots,n-1\}$ and chained hash table is $T[0:m-1]$, where $m < n$.

Defn: In independent uniform hashing, the hash function $h : U \to \{0,1,\ldots,m-1\}$ works in the following manner:

- For each key k, $h(k)$ is independently chosen uniformly randomly from $\{0,1,\ldots,m-1\}$.
- Each subsequent call to h with the same input k yields the same output $h(k)$.

Independent Uniform Hashing

Suppose every element of the dynamic set has a distinct key from the universe $U = \{0,1,\ldots,n-1\}$ and chained hash table is $T[0 : m-1]$, where $m < n$.

Defn: In independent uniform hashing, the hash function $h : U \to \{0,1,\ldots,m-1\}$ works in the following manner:

- For each key k, $h(k)$ is independently chosen uniformly randomly from $\{0,1,\ldots,m-1\}$.
- Each subsequent call to h with the same input k yields the same output $h(k)$.

What is the expected length of a chain now?

Probability Recap

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Set of all possible outcomes of a random event.

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Set of all possible outcomes of a random event.

Defn: Expectation of a random variable, say X, is

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Set of all possible outcomes of a random event.

Defn: Expectation of a random variable, say X, is

$$\mathbf{Ex}(X) = \Sigma_{x \in range(X)}\ x \,.\, \mathbf{Pr}(X = x).$$

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Set of all possible outcomes of a random event.

Defn: Expectation of a random variable, say X, is

$$\textbf{Ex}(X) = \Sigma_{x \in range(X)} \; x \,.\, \textbf{Pr}(X = x).$$

Example: Two dice are rolled. Let X be the sum of the two numbers appearing on the dice.

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Set of all possible outcomes of a random event.

Defn: Expectation of a random variable, say X, is

$$\textbf{Ex}(X) = \Sigma_{x \in range(X)} \; x \, . \, \textbf{Pr}(X = x).$$

Example: Two dice are rolled. Let X be the sum of the two numbers appearing on the dice.

Sample Space :

X :

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Set of all possible outcomes of a random event.

Defn: Expectation of a random variable, say X, is

$$\mathbf{Ex}(X) = \Sigma_{x \in range(X)}\ x \,.\, \mathbf{Pr}(X = x).$$

Example: Two dice are rolled. Let X be the sum of the two numbers appearing on the dice.

Sample Space : $(1,1)$

X : $\downarrow$

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Set of all possible outcomes of a random event.

Defn: Expectation of a random variable, say X, is

$$\mathbf{Ex}(X) = \Sigma_{x \in range(X)}\ x \,.\, \mathbf{Pr}(X = x).$$

Example: Two dice are rolled. Let X be the sum of the two numbers appearing on the dice.

Sample Space : $(1,1)$

X : $\downarrow$ 2

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Set of all possible outcomes of a random event.

Defn: Expectation of a random variable, say X, is

$$\mathbf{Ex}(X) = \Sigma_{x \in range(X)}\ x \,.\, \mathbf{Pr}(X = x).$$

Example: Two dice are rolled. Let X be the sum of the two numbers appearing on the dice.

$$\begin{array}{rcccc} \text{Sample Space}: & (1,1) & \ldots & (2,3) \\ & \downarrow & & \downarrow \\ X: & 2 & \ldots & \end{array}$$

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Set of all possible outcomes of a random event.

Defn: Expectation of a random variable, say X, is

$$\mathbf{Ex}(X) = \Sigma_{x \in range(X)}\ x \,.\, \mathbf{Pr}(X = x).$$

Example: Two dice are rolled. Let X be the sum of the two numbers appearing on the dice.

$$\begin{array}{rcccc} \text{Sample Space} & : & (1,1) & \dots & (2,3) \\ & & \downarrow & & \downarrow \\ X & : & 2 & \dots & 5 \end{array}$$

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Set of all possible outcomes of a random event.

Defn: Expectation of a random variable, say X, is

$$\mathbf{Ex}(X) = \Sigma_{x \in range(X)} \; x \,.\, \mathbf{Pr}(X = x).$$

Example: Two dice are rolled. Let X be the sum of the two numbers appearing on the dice.

$$\begin{array}{lcccc} \text{Sample Space}: & (1,1) & \dots & (2,3) & (1,4) \\ & \downarrow & & \downarrow & \downarrow \\ X: & 2 & \dots & 5 & 5 \end{array}$$

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Set of all possible outcomes of a random event.

Defn: Expectation of a random variable, say X, is

$$\mathbf{Ex}(X) = \Sigma_{x \in range(X)}\ x \,.\, \mathbf{Pr}(X = x).$$

Example: Two dice are rolled. Let X be the sum of the two numbers appearing on the dice.

Sample Space :	(1,1)	…	(2,3)	(1,4)	…	(5,2)	…	(6,6)
	↓		↓	↓		↓		↓
X :	2	…	5	5	…	7	…	12

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Set of all possible outcomes of a random event.

Defn: Expectation of a random variable, say X, is

$$\textbf{Ex}(X) = \Sigma_{x \in range(X)}\ x \,.\, \textbf{Pr}(X = x).$$

Example: Two dice are rolled. Let X be the sum of the two numbers appearing on the dice.

Sample Space :	(1,1)	…	(2,3)	(1,4)	…	(5,2)	…	(6,6)
	$\downarrow$		$\downarrow$	$\downarrow$		$\downarrow$		$\downarrow$
X :	2	…	5	5	…	7	…	12

$$\textbf{Ex}(X) = 2.(1/36) + 3.(2/36) + \ldots\ldots\ldots + 12.(1/36)$$

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Set of all possible outcomes of a random event.

Defn: Expectation of a random variable, say X, is

$$\textbf{Ex}(X) = \Sigma_{x \in range(X)}\ x \,.\, \textbf{Pr}(X = x).$$

Example: Two dice are rolled. Let X be the sum of the two numbers appearing on the dice.

$$\begin{array}{lccccccccc}
\text{Sample Space}: & (1,1) & \dots & (2,3) & (1,4) & \dots & (5,2) & \dots & (6,6) \\
 & \downarrow & & \downarrow & \downarrow & & \downarrow & & \downarrow \\
X: & 2 & \dots & 5 & 5 & \dots & 7 & \dots & 12
\end{array}$$

$$\textbf{Ex}(X) = 2.(1/36) + 3.(2/36) + \dots\dots\dots + 12.(1/36) = 7$$

Probability Recap

Defn: A random variable is a function from sample space to $\mathbb{R}^+$.

Set of all possible outcomes of a random event.

Defn: Expectation of a random variable, say X, is

$$\mathbf{Ex}(X) = \Sigma_{x \in range(X)}\ x \,.\, \mathbf{Pr}(X = x).$$

Example: Two dice are rolled. Let X be the sum of the two numbers appearing on the dice.

Sample Space :	(1,1)	…	(2,3)	(1,4)	…	(5,2)	…	(6,6)
	$\downarrow$		$\downarrow$	$\downarrow$		$\downarrow$		$\downarrow$
X :	2	…	5	5	…	7	…	12

$$\mathbf{Ex}(X) = 2.(1/36) + 3.(2/36) + \ldots\ldots\ldots + 12.(1/36) = 7$$

Linearity of Expectation: $\mathbf{Ex}[X_1 + X_2 + \ldots + X_n] = \mathbf{Ex}[X_1] + \mathbf{Ex}[X_2] + \ldots + \mathbf{Ex}[X_n]$

Independent Uniform Hashing: Analysis

Independent Uniform Hashing: Analysis

Recall: Hash function h that maps every key from $\{0,1,\ldots,n-1\}$ to independently and uniformly randomly chosen value from $\{0,1,\ldots,m-1\}$.

Independent Uniform Hashing: Analysis

Recall: Hash function h that maps every key from $\{0,1,\ldots,n-1\}$ to independently and uniformly randomly chosen value from $\{0,1,\ldots,m-1\}$.

Fix a $j \in \{0,1,\ldots,m-1\}$**:**

Independent Uniform Hashing: Analysis

Recall: Hash function h that maps every key from $\{0,1,\ldots,n-1\}$ to independently and uniformly randomly chosen value from $\{0,1,\ldots,m-1\}$.

Fix a $j \in \{0,1,\ldots,m-1\}$**:**

Define random variables $X_0, X_1, \ldots, X_{n-1}$

Independent Uniform Hashing: Analysis

Recall: Hash function h that maps every key from $\{0,1,\ldots,n-1\}$ to independently and uniformly randomly chosen value from $\{0,1,\ldots,m-1\}$.

Fix a $j \in \{0,1,\ldots,m-1\}$**:**

Define random variables $X_0, X_1, \ldots, X_{n-1}$, such that $X_i = 1$ if $h(i) = j$, else $X_i = 0$.

Independent Uniform Hashing: Analysis

Recall: Hash function h that maps every key from $\{0,1,\ldots,n-1\}$ to independently and uniformly randomly chosen value from $\{0,1,\ldots,m-1\}$.

Fix a $j \in \{0,1,\ldots,m-1\}$**:**

Define random variables $X_0, X_1, \ldots, X_{n-1}$, such that $X_i = 1$ if $h(i) = j$, else $X_i = 0$.

Define random variables T_j

Independent Uniform Hashing: Analysis

Recall: Hash function h that maps every key from $\{0,1,\ldots,n-1\}$ to independently and uniformly randomly chosen value from $\{0,1,\ldots,m-1\}$.

Fix a $j \in \{0,1,\ldots,m-1\}$**:**

Define random variables $X_0, X_1, \ldots, X_{n-1}$, such that $X_i = 1$ if $h(i) = j$, else $X_i = 0$.

Define random variables T_j, such that T_j is the length of the chain of the jth slot of hash table.

Independent Uniform Hashing: Analysis

Recall: Hash function h that maps every key from $\{0,1,\ldots,n-1\}$ to independently and uniformly randomly chosen value from $\{0,1,\ldots,m-1\}$.

Fix a $j \in \{0,1,\ldots,m-1\}$**:**

Define random variables $X_0, X_1, \ldots, X_{n-1}$, such that $X_i = 1$ if $h(i) = j$, else $X_i = 0$.

Define random variables T_j, such that T_j is the length of the chain of the jth slot of hash table.

$$T_j = X_0 + X_1 + \ldots + X_{n-1}$$

Independent Uniform Hashing: Analysis

Recall: Hash function h that maps every key from $\{0,1,\ldots,n-1\}$ to independently and uniformly randomly chosen value from $\{0,1,\ldots,m-1\}$.

Fix a $j \in \{0,1,\ldots,m-1\}$**:**

Define random variables $X_0, X_1, \ldots, X_{n-1}$, such that $X_i = 1$ if $h(i) = j$, else $X_i = 0$.

Define random variables T_j, such that T_j is the length of the chain of the jth slot of hash table.

$$T_j = X_0 + X_1 + \ldots + X_{n-1}$$

$$\textbf{Ex}[T_j] \ = \ \textbf{Ex}[X_0 + X_1 + \ldots + X_{n-1}]$$

Independent Uniform Hashing: Analysis

Recall: Hash function h that maps every key from $\{0,1,\ldots,n-1\}$ to independently and uniformly randomly chosen value from $\{0,1,\ldots,m-1\}$.

Fix a $j \in \{0,1,\ldots,m-1\}$**:**

Define random variables $X_0, X_1, \ldots, X_{n-1}$, such that $X_i = 1$ if $h(i) = j$, else $X_i = 0$.

Define random variables T_j, such that T_j is the length of the chain of the jth slot of hash table.

$$T_j = X_0 + X_1 + \ldots + X_{n-1}$$

$$\mathbf{Ex}[T_j] = \mathbf{Ex}[X_0 + X_1 + \ldots + X_{n-1}] = \mathbf{Ex}[X_0] + \mathbf{Ex}[X_1] + \ldots + \mathbf{Ex}[X_{n-1}]$$

Independent Uniform Hashing: Analysis

Recall: Hash function h that maps every key from $\{0,1,\ldots,n-1\}$ to independently and uniformly randomly chosen value from $\{0,1,\ldots,m-1\}$.

Fix a $j \in \{0,1,\ldots,m-1\}$**:**

Define random variables $X_0, X_1, \ldots, X_{n-1}$, such that $X_i = 1$ if $h(i) = j$, else $X_i = 0$.

Define random variables T_j, such that T_j is the length of the chain of the jth slot of hash table.

$$T_j = X_0 + X_1 + \ldots + X_{n-1}$$

$$\mathbf{Ex}[T_j] = \mathbf{Ex}[X_0 + X_1 + \ldots + X_{n-1}] = \mathbf{Ex}[X_0] + \mathbf{Ex}[X_1] + \ldots + \mathbf{Ex}[X_{n-1}] = n \cdot \left(\frac{1}{m}\right) = n/m$$

Independent Uniform Hashing: Analysis

Recall: Hash function h that maps every key from $\{0,1,\ldots,n-1\}$ to independently and uniformly randomly chosen value from $\{0,1,\ldots,m-1\}$.

Fix a $j \in \{0,1,\ldots,m-1\}$**:**

Define random variables $X_0, X_1, \ldots, X_{n-1}$, such that $X_i = 1$ if $h(i) = j$, else $X_i = 0$.

Define random variables T_j, such that T_j is the length of the chain of the jth slot of hash table.

$$T_j = X_0 + X_1 + \ldots + X_{n-1}$$

$$\textbf{Ex}[T_j] = \textbf{Ex}[X_0 + X_1 + \ldots + X_{n-1}] = \textbf{Ex}[X_0] + \textbf{Ex}[X_1] + \ldots + \textbf{Ex}[X_{n-1}] = n \cdot \left(\frac{1}{m}\right) = n/m$$

Search's Time Complexity: $O(1)$, when $n = m$.

Open Addressing

Open Addressing

In open addressing is an alternative to chaining in hash tables in which:

Open Addressing

In open addressing is an alternative to chaining in hash tables in which:

- All elements occupy the hash tables itself, i.e., $T[i] = x$ or $T[i] = NIL$.

Open Addressing

In open addressing is an alternative to chaining in hash tables in which:

- All elements occupy the hash tables itself, i.e., $T[i] = x$ or $T[i] = NIL$.
- Collision are handled as follows:

Open Addressing

In open addressing is an alternative to chaining in hash tables in which:

- All elements occupy the hash tables itself, i.e., $T[i] = x$ or $T[i] = NIL$.
- Collision are handled as follows:
  - A new element is inserted in "first-choice" location, if it's free.

Open Addressing

In open addressing is an alternative to chaining in hash tables in which:

- All elements occupy the hash tables itself, i.e., $T[i] = x$ or $T[i] = NIL$.
- Collision are handled as follows:
  - A new element is inserted in "first-choice" location, if it's free.
  - If "first-choice" location is already filled, then place it in "second-choice" location if it's free.

Open Addressing

In open addressing is an alternative to chaining in hash tables in which:

- All elements occupy the hash tables itself, i.e., $T[i] = x$ or $T[i] = NIL$.
- Collision are handled as follows:
  - A new element is inserted in "first-choice" location, if it's free.
  - If "first-choice" location is already filled, then place it in "second-choice" location if it's free.
  - If "second-choice" location is already filled, then place it in "third-choice" location if it's free.

Open Addressing

In open addressing is an alternative to chaining in hash tables in which:

- All elements occupy the hash tables itself, i.e., $T[i] = x$ or $T[i] = NIL$.
- Collision are handled as follows:
  - A new element is inserted in "first-choice" location, if it's free.
  - If "first-choice" location is already filled, then place it in "second-choice" location if it's free.
  - If "second-choice" location is already filled, then place it in "third-choice" location if it's free.
  - ....

Open Addressing

In open addressing is an alternative to chaining in hash tables in which:

- All elements occupy the hash tables itself, i.e., $T[i] = x$ or $T[i] = NIL$.
- Collision are handled as follows:
  - A new element is inserted in "first-choice" location, if it's free.
  - If "first-choice" location is already filled, then place it in "second-choice" location if it's free.
  - If "second-choice" location is already filled, then place it in "third-choice" location if it's free.
  - ....
- Number of elements cannot exceed the size of hash table.

Open Addressing

Open Addressing

The sequence through which free slots are searched is called probe-sequence.

Open Addressing

The sequence through which free slots are searched is called probe-sequence.

Hash functions are defined as:

Open Addressing

The sequence through which free slots are searched is called probe-sequence.

Hash functions are defined as:

$$h : U \times \{0,1,\ldots,m-1\} \rightarrow \{0,1,\ldots,m-1\}$$

Open Addressing

The sequence through which free slots are searched is called probe-sequence.

Hash functions are defined as:

$$h : U \times \{0,1,\ldots,m-1\} \to \{0,1,\ldots,m-1\}$$

Such that for all the keys k:

Open Addressing

The sequence through which free slots are searched is called probe-sequence.

Hash functions are defined as:

$$h : U \times \{0,1,\ldots,m-1\} \to \{0,1,\ldots,m-1\}$$

Such that for all the keys k:

Probe sequence $\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ is a permutation of $\langle 0,1,\ldots,m-1 \rangle$

Open Addressing: Operations

Open Addressing: Operations

OA-INSERT(T, x)**:**

Open Addressing: Operations

OA-INSERT(T, x)**:**

1. $i = 0$

Open Addressing: Operations

OA-INSERT(T, x)**:**

1. $i = 0$
2. **do**

Open Addressing: Operations

OA-INSERT(T, x)**:**

1. $i = 0$
2. **do**
3. $\quad q = h(x\,.\,key, i)$

Open Addressing: Operations

OA-INSERT(T, x)**:**

$i = 0$

do

    $q = h(x\,.\,key, i)$

    if $T[q] == NIL$

Open Addressing: Operations

OA-INSERT(T, x):

$i = 0$
do
    $q = h(x\,.\,key, i)$
    if $T[q] == NIL$
        $T[q] = x$

Open Addressing: Operations

```
OA-INSERT(T, x):
i = 0
do
    q = h(x.key, i)
    if T[q] == NIL
        T[q] = x
        return q
```

Open Addressing: Operations

OA-INSERT(T, x):

$i = 0$
do
    $q = h(x.key, i)$
    if $T[q] == NIL$
        $T[q] = x$
        return q
    else

Open Addressing: Operations

OA-INSERT(T, x):

$i = 0$
do
    $q = h(x\,.\,key, i)$
    if $T[q] == NIL$
        $T[q] = x$
        return q
    else
        $i = i + 1$

Open Addressing: Operations

```
OA-INSERT(T, x):
i = 0
do
    q = h(x.key, i)
    if T[q] == NIL
        T[q] = x
        return q
    else
        i = i + 1
while i ≠ m
```

Open Addressing: Operations

OA-INSERT(T, x)**:**

```
i = 0
do
    q = h(x.key, i)
    if T[q] == NIL
        T[q] = x
        return q
    else
        i = i + 1
while i ≠ m
error "Hash table overflow"
```

Open Addressing: Operations

Open Addressing: Operations

OA-SEARCH(T, k)**:**

Open Addressing: Operations

OA-SEARCH(T, k)**:**

1. $i = 0$

Open Addressing: Operations

OA-SEARCH(T, k)**:**

1. $i = 0$
2. **do**

Open Addressing: Operations

OA-SEARCH(T, k)**:**

1. $i = 0$
2. **do**
3. $\quad q = h(k, i)$

Open Addressing: Operations

OA-SEARCH(T, k)**:**

1. $i = 0$
2. **do**
3. $\quad q = h(k, i)$
4. $\quad$ **if** $T[q].key == k$

Open Addressing: Operations

OA-SEARCH(T, k)**:**

$i = 0$

do

    $q = h(k, i)$

    if $T[q].key == k$

        return $T[q]$

Open Addressing: Operations

OA-SEARCH(T, k)**:**

$i = 0$
do
    $q = h(k, i)$
    if $T[q].key == k$
        return $T[q]$
    else

Open Addressing: Operations

OA-SEARCH(T, k)**:**

$i = 0$

do

  $q = h(k, i)$

  if $T[q].key == k$

    return $T[q]$

  else

    $i = i + 1$

Open Addressing: Operations

OA-SEARCH(T, k):

$i = 0$
do
    $q = h(k, i)$
    if $T[q].key == k$
        return $T[q]$
    else
        $i = i + 1$
while $T[q] \neq NIL$ **and** $i \neq m$

Open Addressing: Operations

OA-SEARCH(T, k):

```
i = 0
do
    q = h(k, i)
    if T[q].key == k
        return T[q]
    else
        i = i + 1
while T[q] ≠ NIL and i ≠ m
return NIL
```

Open Addressing: Operations

Open Addressing: Operations

OA-DELETE(T, x)**:**

Open Addressing: Operations

OA-DELETE(T, x)**:**

1. $i = 0$

Open Addressing: Operations

OA-DELETE(T, x)**:**

1. $i = 0$
2. **do**

Open Addressing: Operations

OA-DELETE(T, x)**:**

1. $i = 0$
2. **do**
3. $\quad q = h(x, i)$

Open Addressing: Operations

OA-DELETE(T, x)**:**

1. $i = 0$
2. **do**
3. $\quad q = h(x, i)$
4. $\quad$ **if** $T[q].key == x.key$

Open Addressing: Operations

OA-DELETE(T, x)**:**

$i = 0$

do

    $q = h(x, i)$

    if $T[q] \,.\, key == x \,.\, key$

        $T[q] = NIL$

Open Addressing: Operations

OA-DELETE(T, x)**:**

$i = 0$

do

    $q = h(x, i)$

    if $T[q].key == x.key$

        $T[q] = NIL$

        return

Open Addressing: Operations

OA-DELETE(T, x)**:**

```
i = 0
do
    q = h(x, i)
    if T[q].key == x.key
        T[q] = NIL
        return
    else
```

Open Addressing: Operations

OA-DELETE(T, x)**:**

```
i = 0
do
    q = h(x, i)
    if T[q].key == x.key
        T[q] = NIL
        return
    else
        i = i + 1
```

Open Addressing: Operations

OA-DELETE(T, x)**:**

```
i = 0
do
    q = h(x, i)
    if T[q].key == x.key
        T[q] = NIL
        return
    else
        i = i + 1
while T[q] ≠ NIL and i ≠ m
```

Open Addressing: Operations

```
OA-DELETE(T, x):
    i = 0
    do
        q = h(x, i)
        if T[q] . key == x . key
            T[q] = NIL
            return
        else
            i = i + 1
    while T[q] ≠ NIL and i ≠ m
    return NIL
```

Open Addressing: Operations

OA-DELETE(T, x):

$i = 0$
do
  $q = h(x, i)$
  if $T[q].key == x.key$
    $T[q] = NIL$
    return
  else
    $i = i + 1$
while $T[q] \neq NIL$ **and** $i \neq m$
return NIL

Flaw: Deletion like this can affect the search or deletion of previously inserted element.

Probing Sequence

Probing Sequence

Linear Probing:

Probing Sequence

Linear Probing:

$$h(k, i) = (h'(k) + i) \,\%\, m$$

Probing Sequence

Linear Probing:

$h(k, i) = (h'(k) + i) \,\%\, m$, where $h'(k)$ is an ordinary hash function.

Probing Sequence

Linear Probing:

$h(k, i) = (h'(k) + i) \,\%\, m$, where $h'(k)$ is an ordinary hash function.

Double Hashing:

Probing Sequence

Linear Probing:

$h(k, i) = (h'(k) + i) \,\%\, m$, where $h'(k)$ is an ordinary hash function.

Double Hashing:

$h(k, i) = (h_1(k) + ih_2(k)) \,\%\, m$

Probing Sequence

Linear Probing:

$h(k, i) = (h'(k) + i) \,\%\, m$, where $h'(k)$ is an ordinary hash function.

Double Hashing:

$h(k, i) = (h_1(k) + ih_2(k)) \,\%\, m$, where $h_1(k)$ and $h_2(k)$ are ordinary hash functions

Probing Sequence

Linear Probing:

$h(k, i) = (h'(k) + i) \,\%\, m$, where $h'(k)$ is an ordinary hash function.

Double Hashing:

$h(k, i) = (h_1(k) + ih_2(k)) \,\%\, m$, where $h_1(k)$ and $h_2(k)$ are ordinary hash functions

and $gcd(h_2(k), m) = 1$ for all k.

Probing Sequence

Linear Probing:

$h(k, i) = (h'(k) + i) \,\%\, m$, where $h'(k)$ is an ordinary hash function.

Double Hashing:

$h(k, i) = (h_1(k) + ih_2(k)) \,\%\, m$, where $h_1(k)$ and $h_2(k)$ are ordinary hash functions

and $\underbrace{gcd(h_2(k), m) = 1}$ for all k.

To hit all the slots

Binary Search Trees

Binary Search Trees

Binary Search Trees (BSTs) are used to maintain a dynamic set that supports operations such as:

Binary Search Trees

Binary Search Trees (BSTs) are used to maintain a dynamic set that supports operations such as:

- Insert an element.

Binary Search Trees

Binary Search Trees (BSTs) are used to maintain a dynamic set that supports operations such as:

- Insert an element.
- Search for an element with the key k.

Binary Search Trees

Binary Search Trees (BSTs) are used to maintain a dynamic set that supports operations such as:

- Insert an element.
- Search for an element with the key k.
- Delete an element.

Binary Search Trees

Binary Search Trees (BSTs) are used to maintain a dynamic set that supports operations such as:

- Insert an element.
- Search for an element with the key k.
- Delete an element.
- Minimum or Maximum of the set.

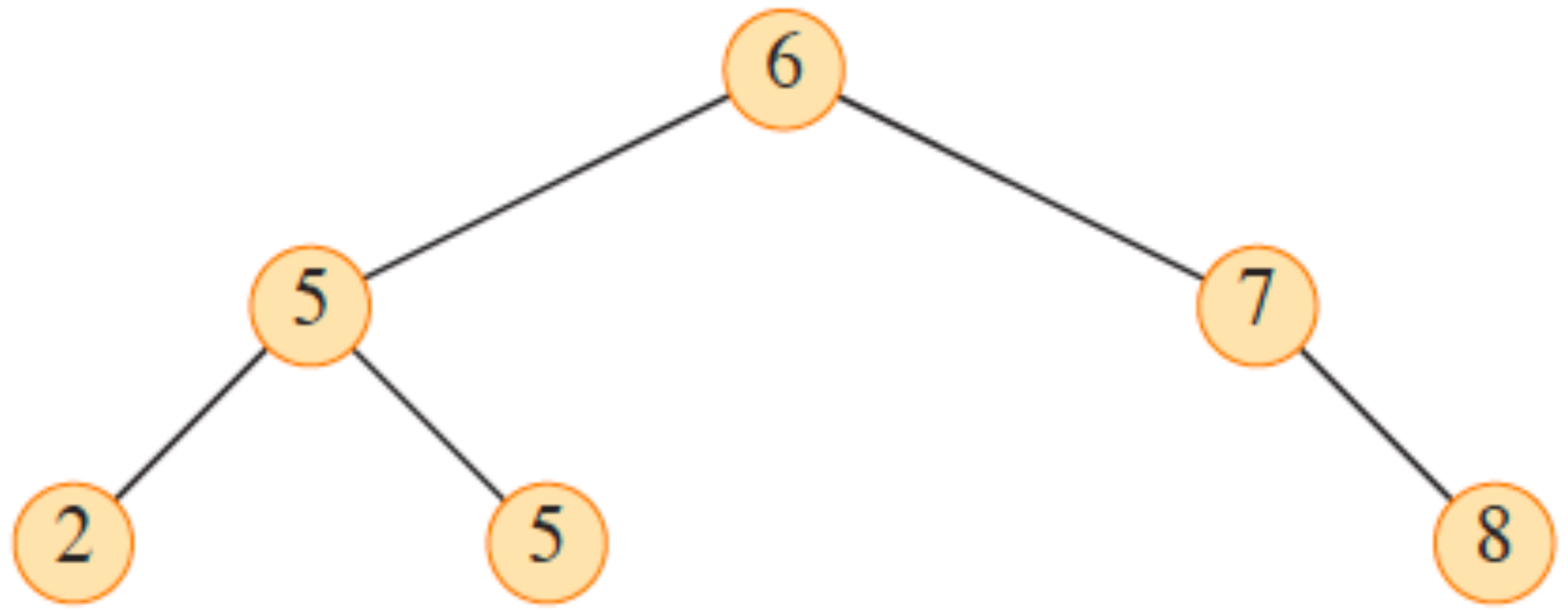
Binary Search Trees

Binary Search Trees (BSTs) are used to maintain a dynamic set that supports operations such as:

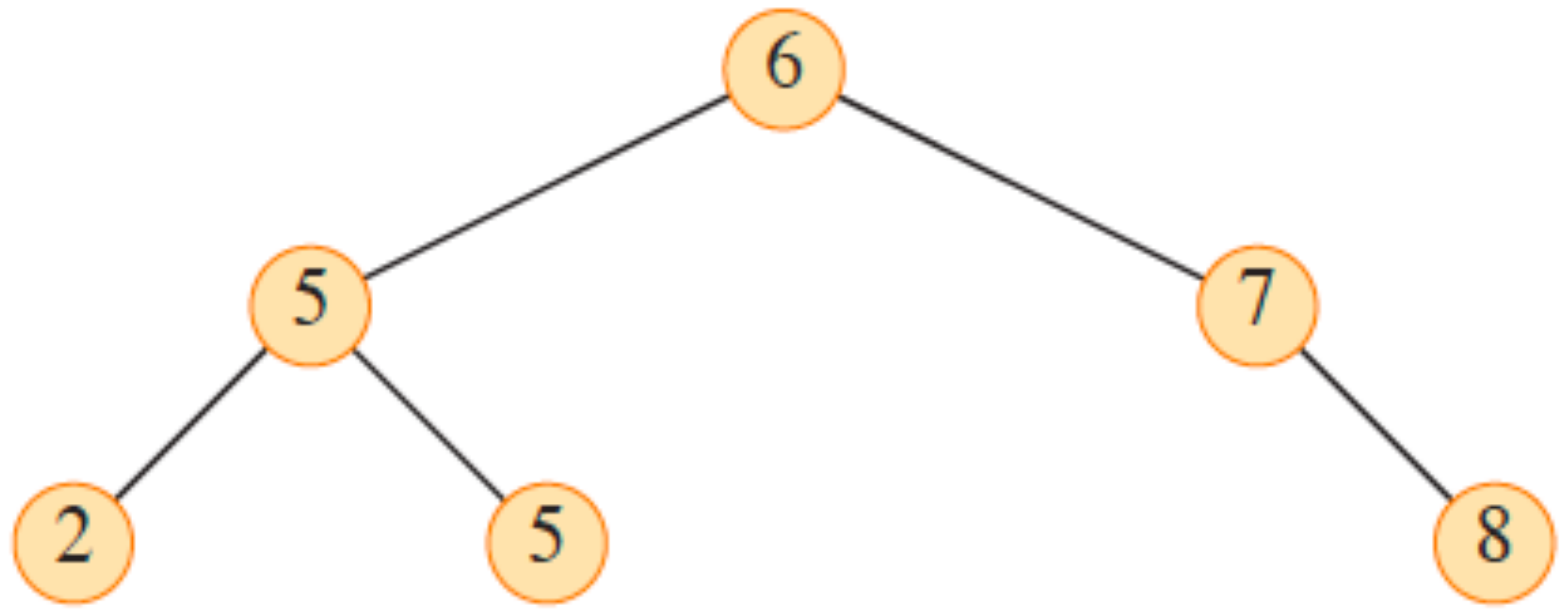
- Insert an element.
- Search for an element with the key k.
- Delete an element.
- Minimum or Maximum of the set.
- Successor or Predecessor of an element of the set.

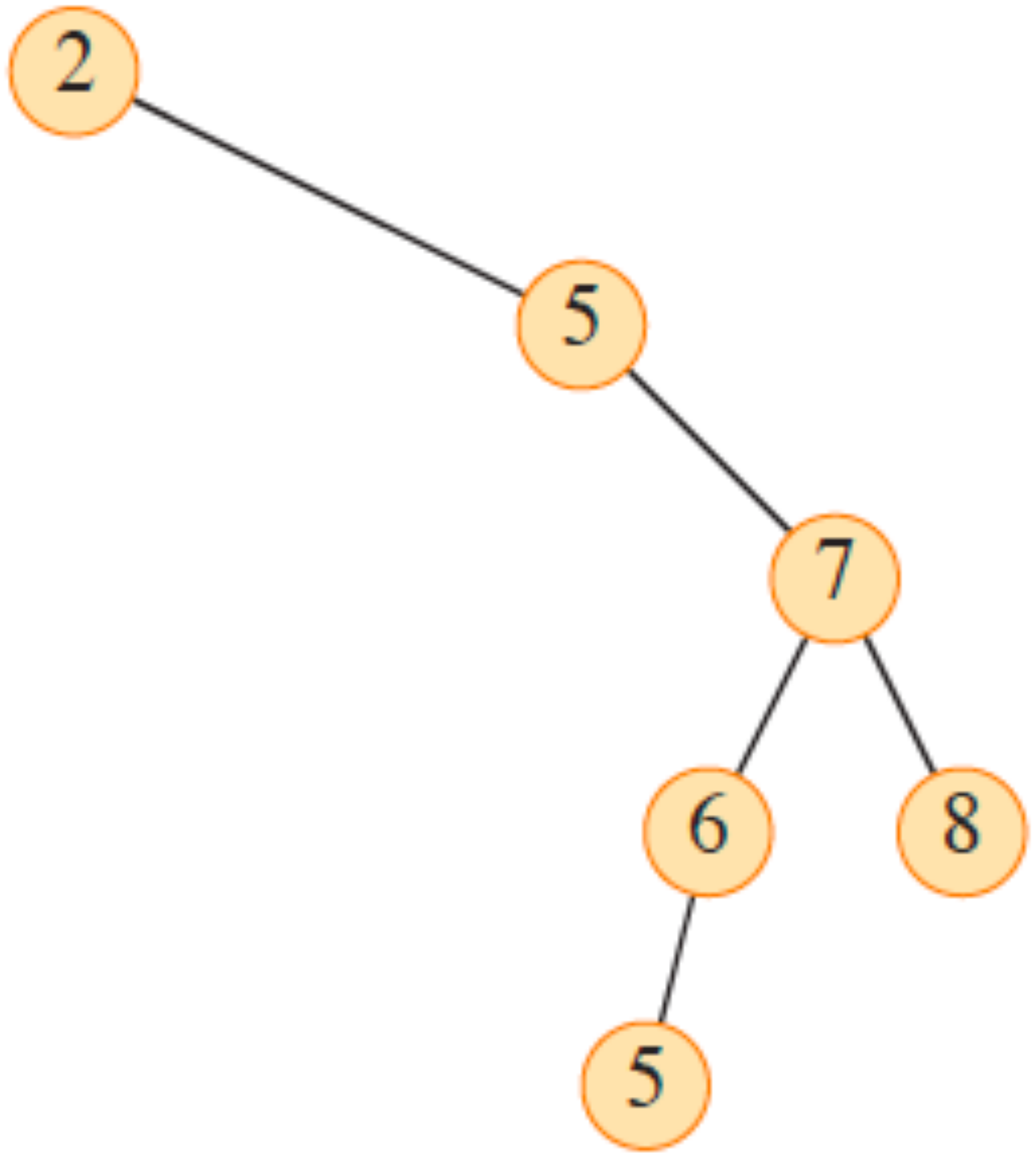
What is a Binary Search Tree?

What is a Binary Search Tree?

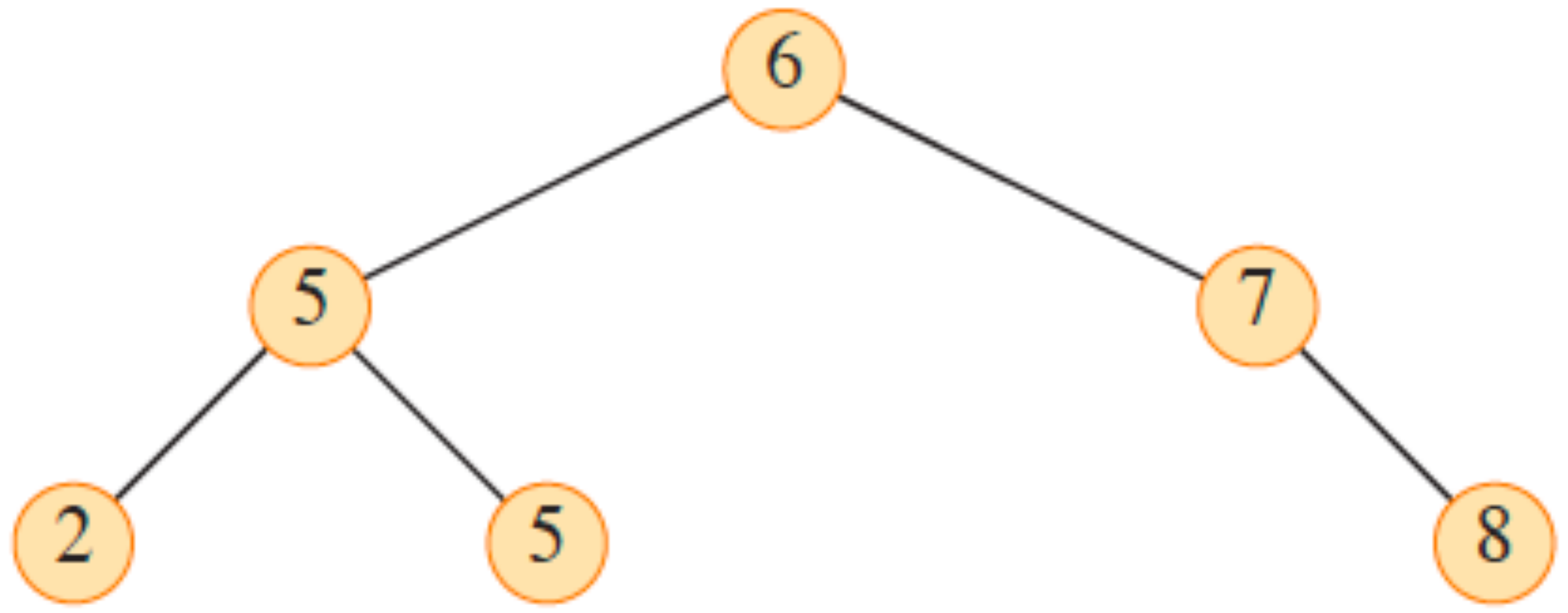


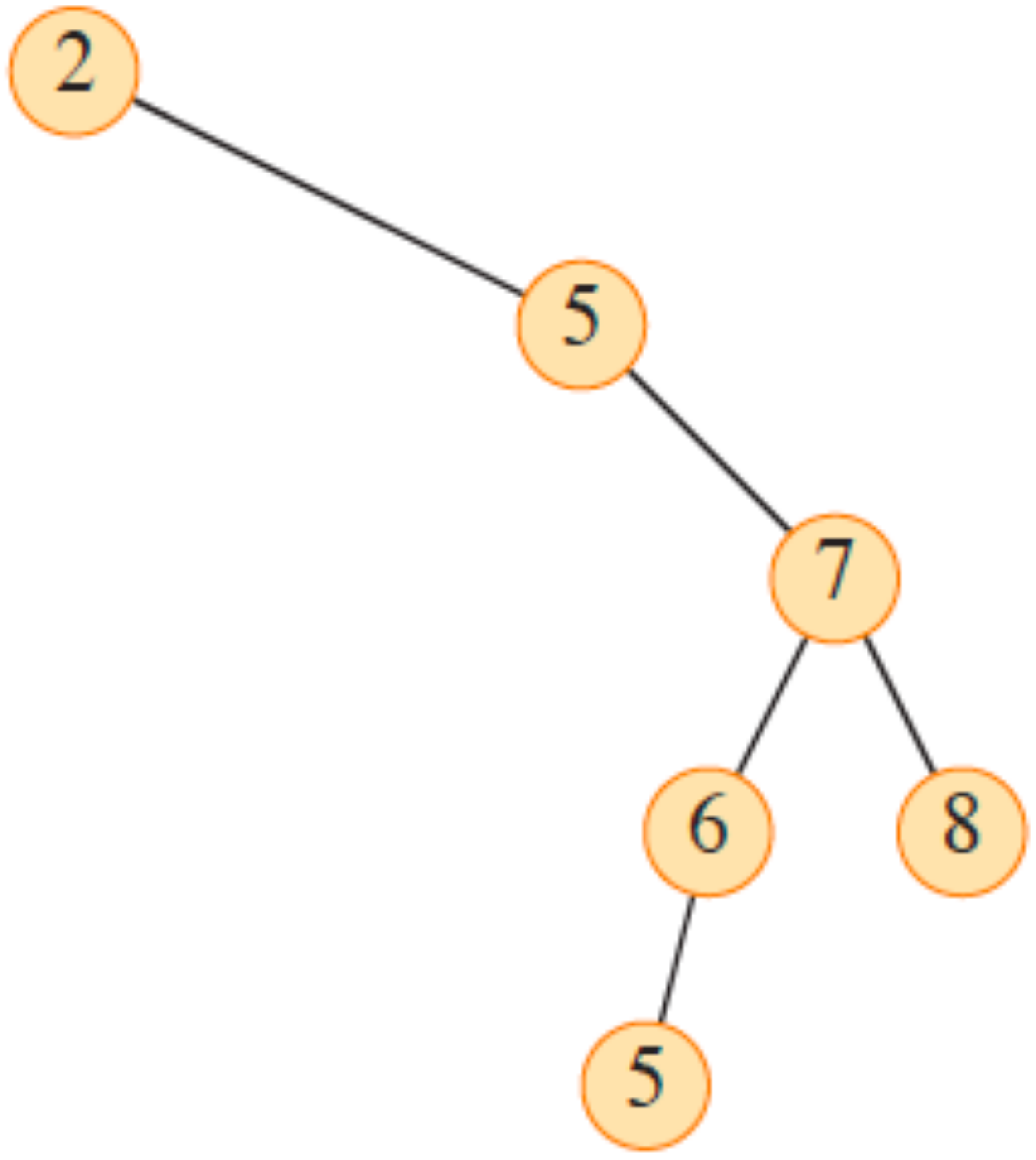
What is a Binary Search Tree?





What is a Binary Search Tree?





Source: CLRS

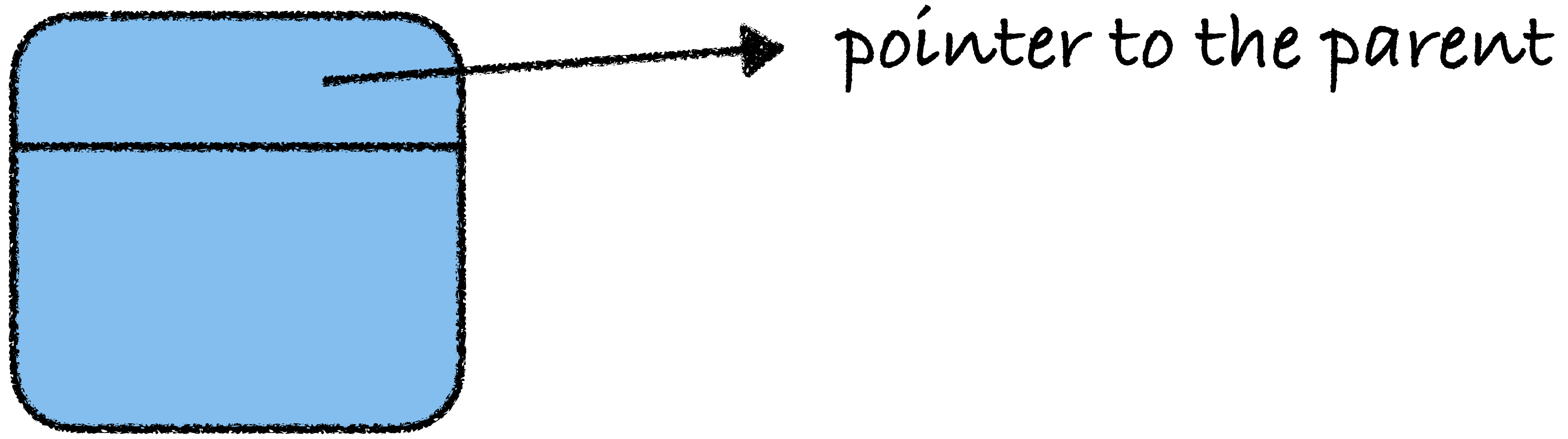
What is a Binary Search Tree?

What is a Binary Search Tree?

- A binary search tree is a collection of nodes of the following type:

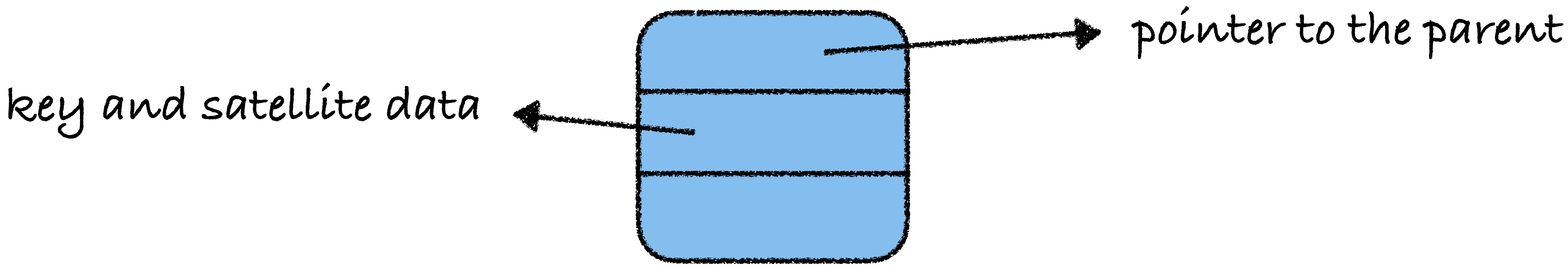
What is a Binary Search Tree?

- A binary search tree is a collection of nodes of the following type:



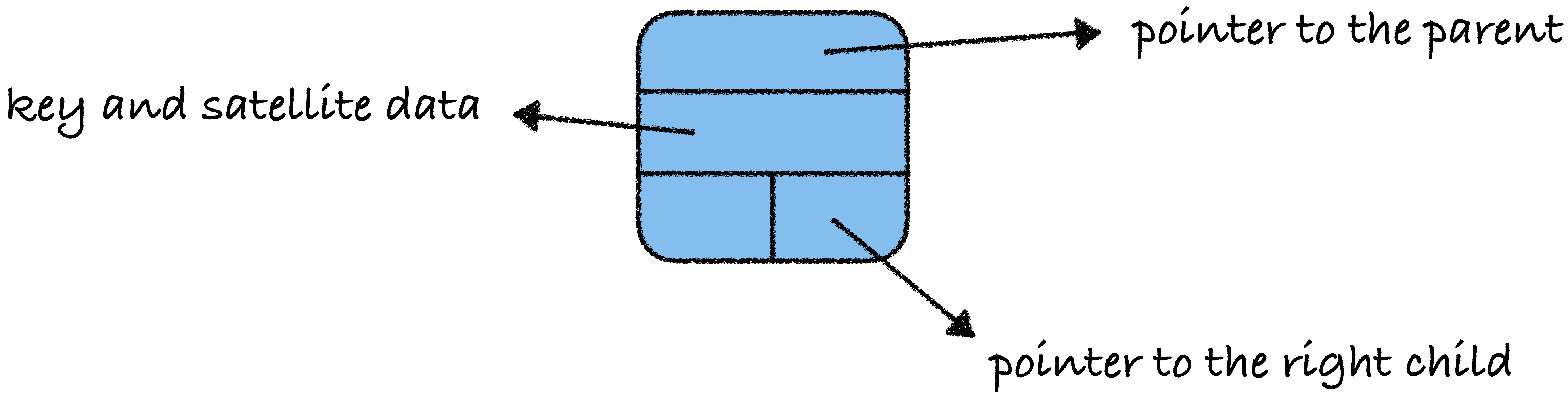
What is a Binary Search Tree?

- A binary search tree is a collection of nodes of the following type:



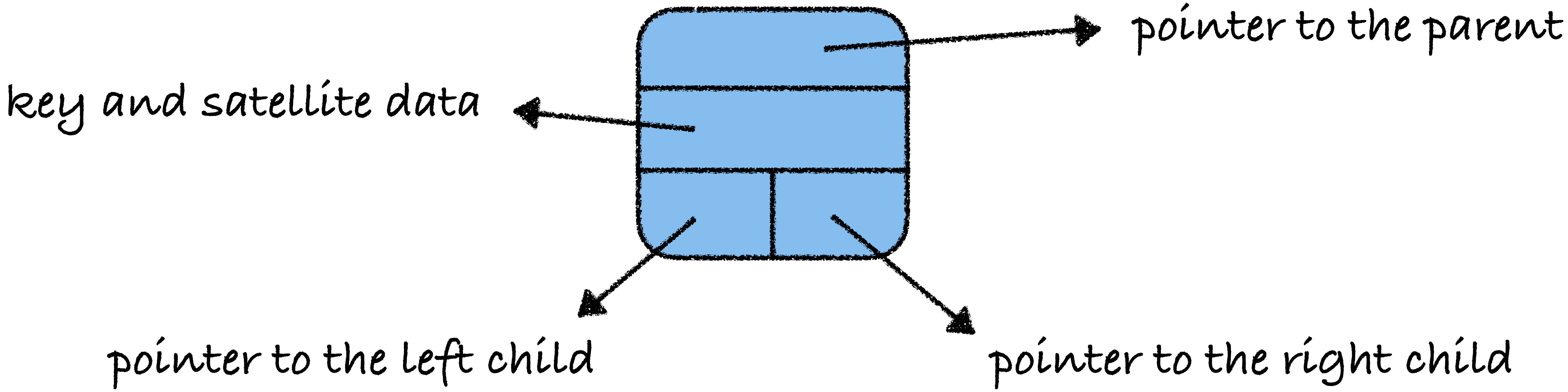
What is a Binary Search Tree?

- A binary search tree is a collection of nodes of the following type:



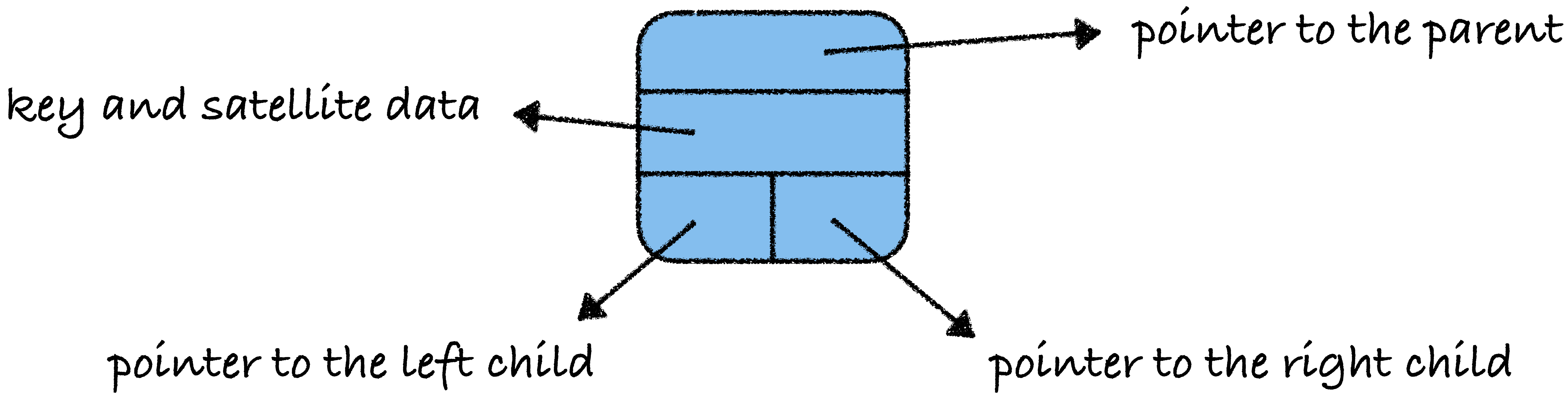
What is a Binary Search Tree?

- A binary search tree is a collection of nodes of the following type:



What is a Binary Search Tree?

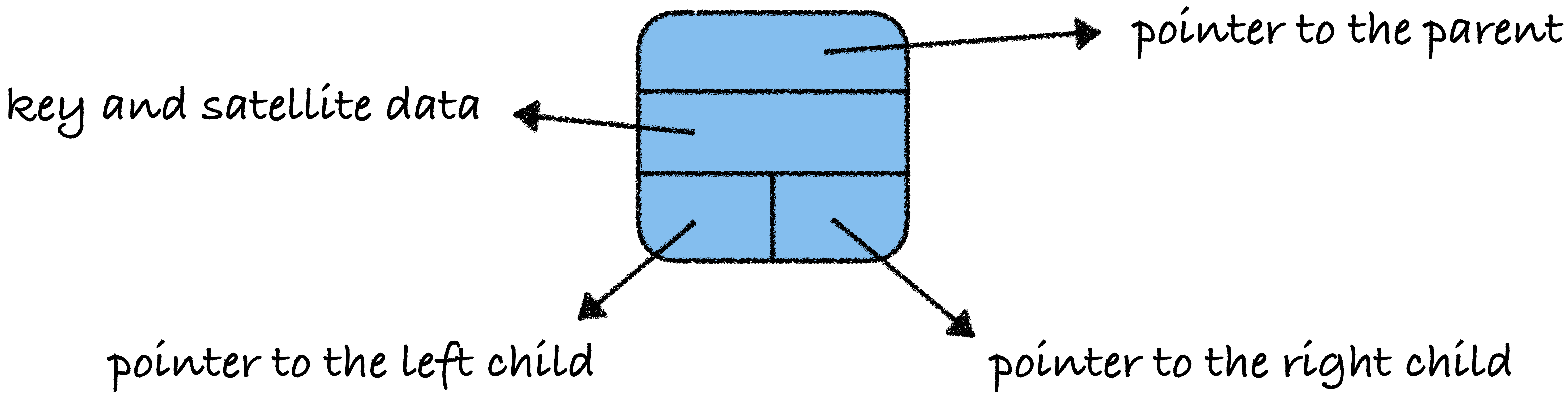
- A binary search tree is a collection of nodes of the following type:



- Every tree has a root.

What is a Binary Search Tree?

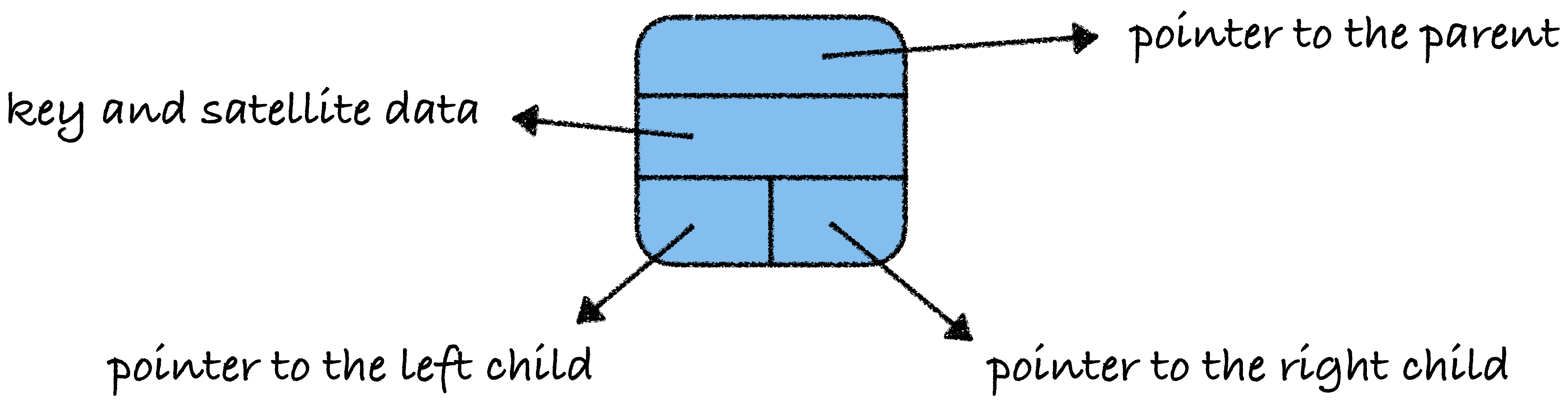
- A binary search tree is a collection of nodes of the following type:



- Every tree has a root.
- A BST satisfies the following property:

What is a Binary Search Tree?

- A binary search tree is a collection of nodes of the following type:



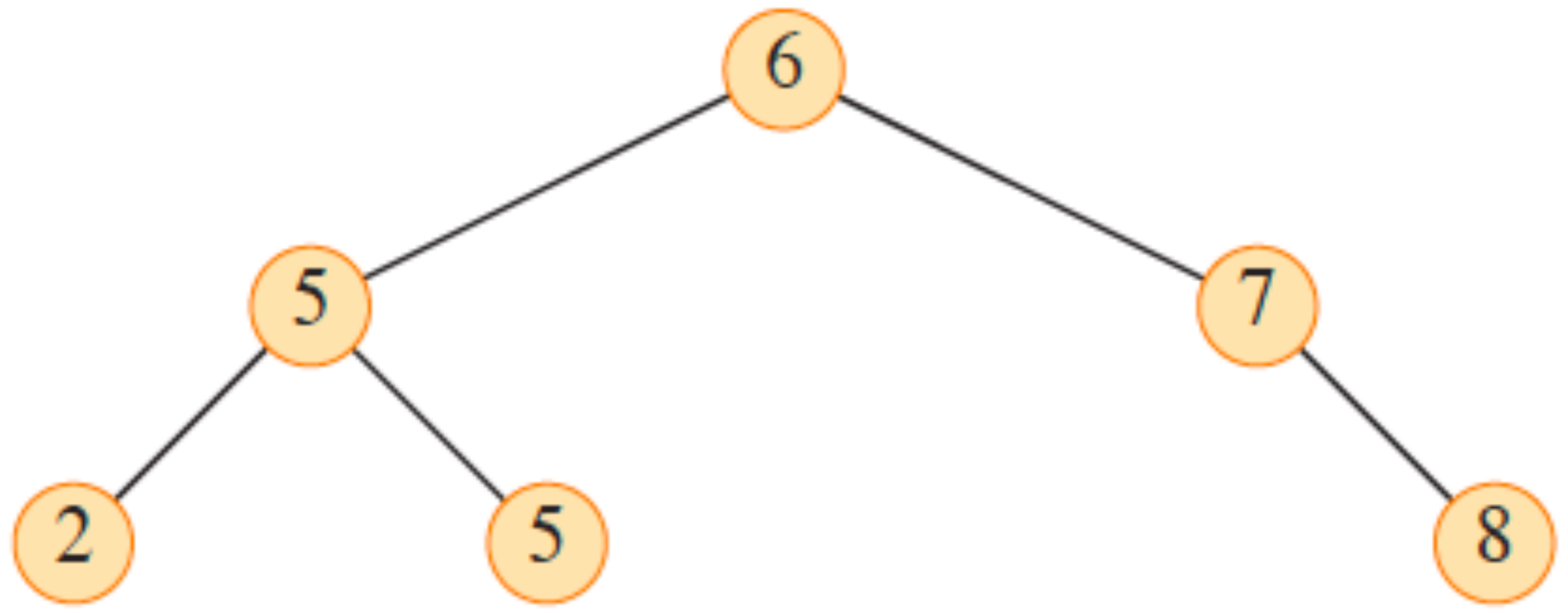
- Every tree has a root.
- A BST satisfies the following property:

  Let x be a node in BST and y, z be the nodes in its left, right subtree, respectively.

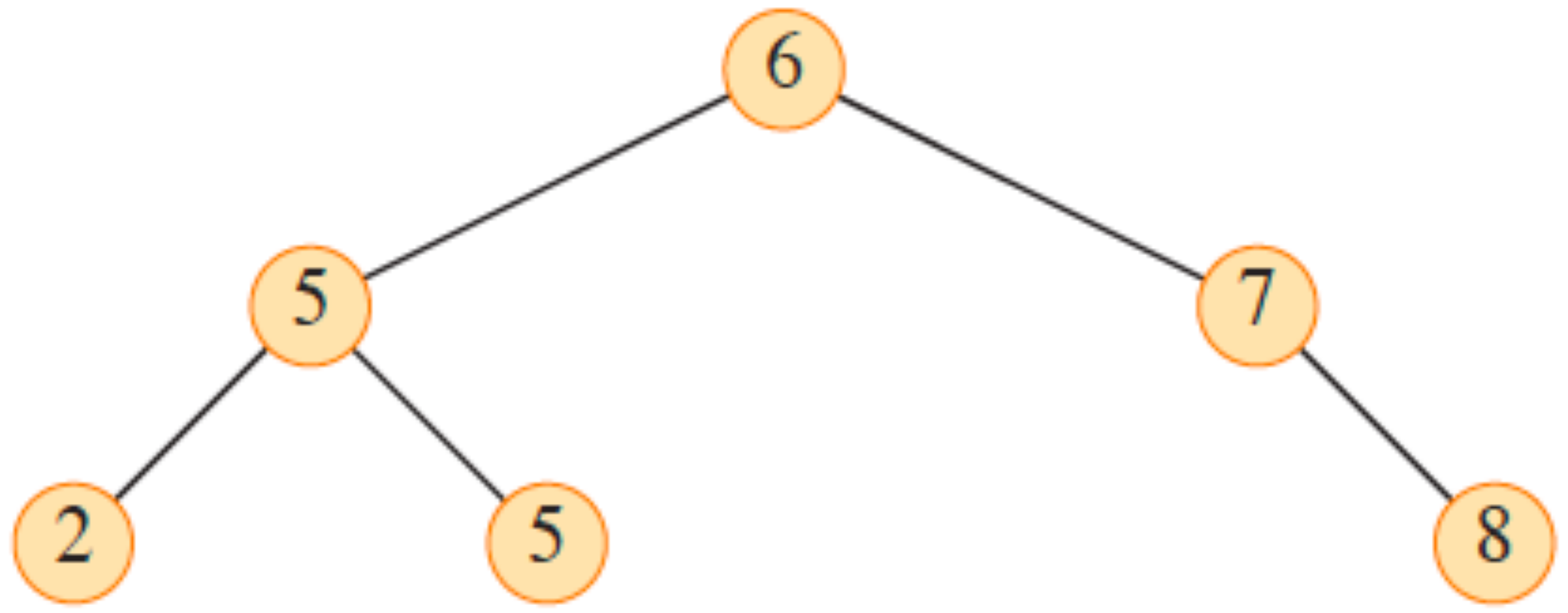
  Then, $y.key \leq x.key \leq z.key$.

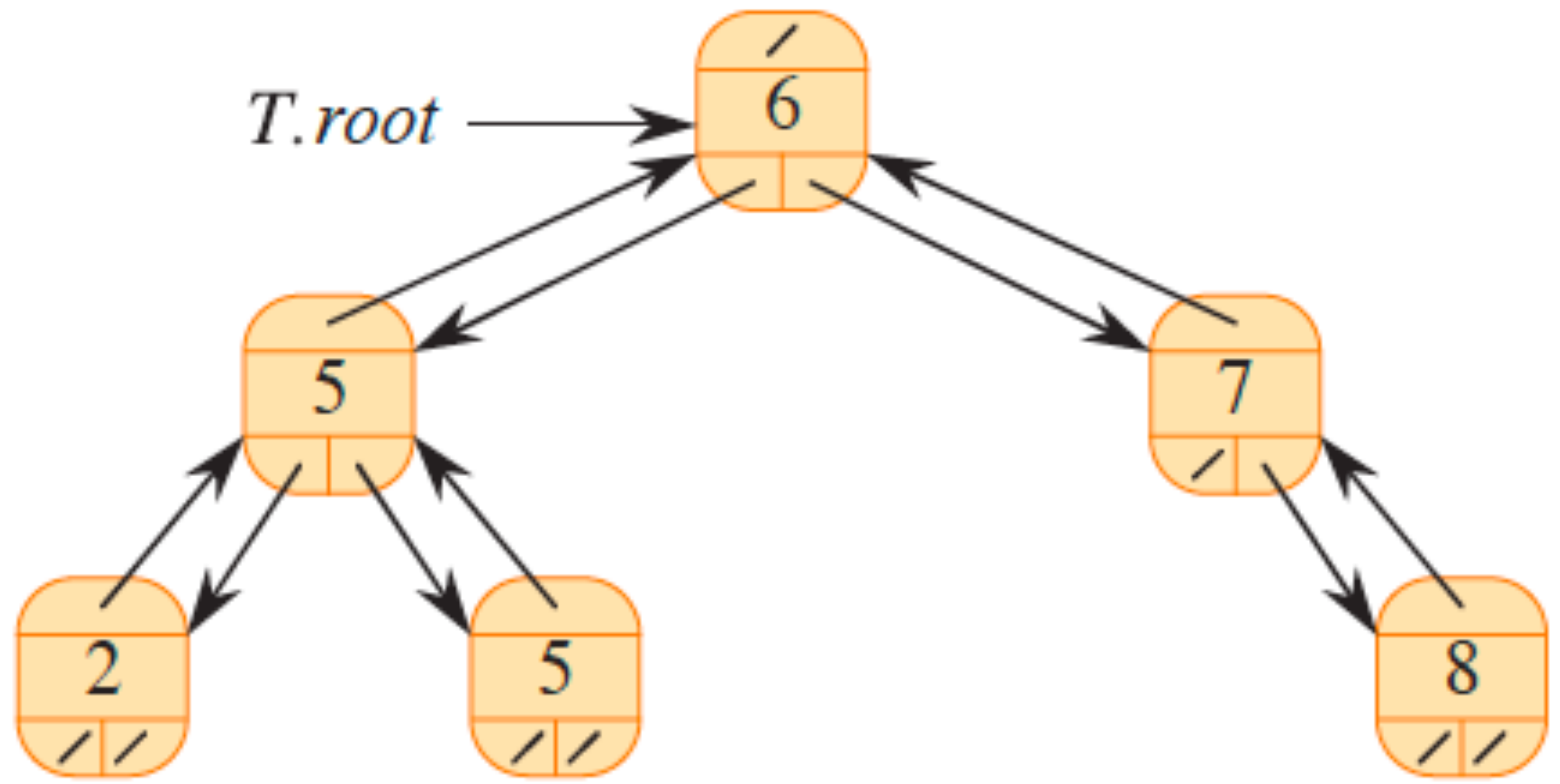
What is a Binary Search Tree?

What is a Binary Search Tree?

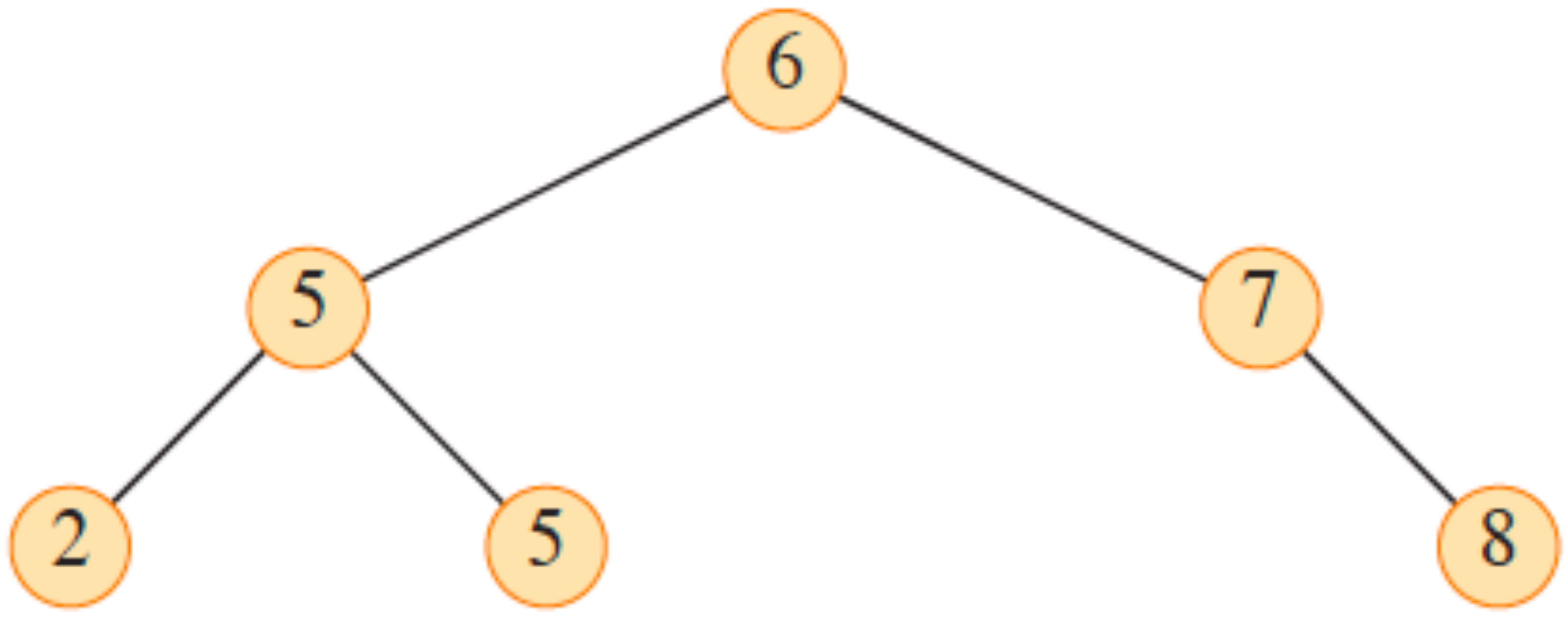


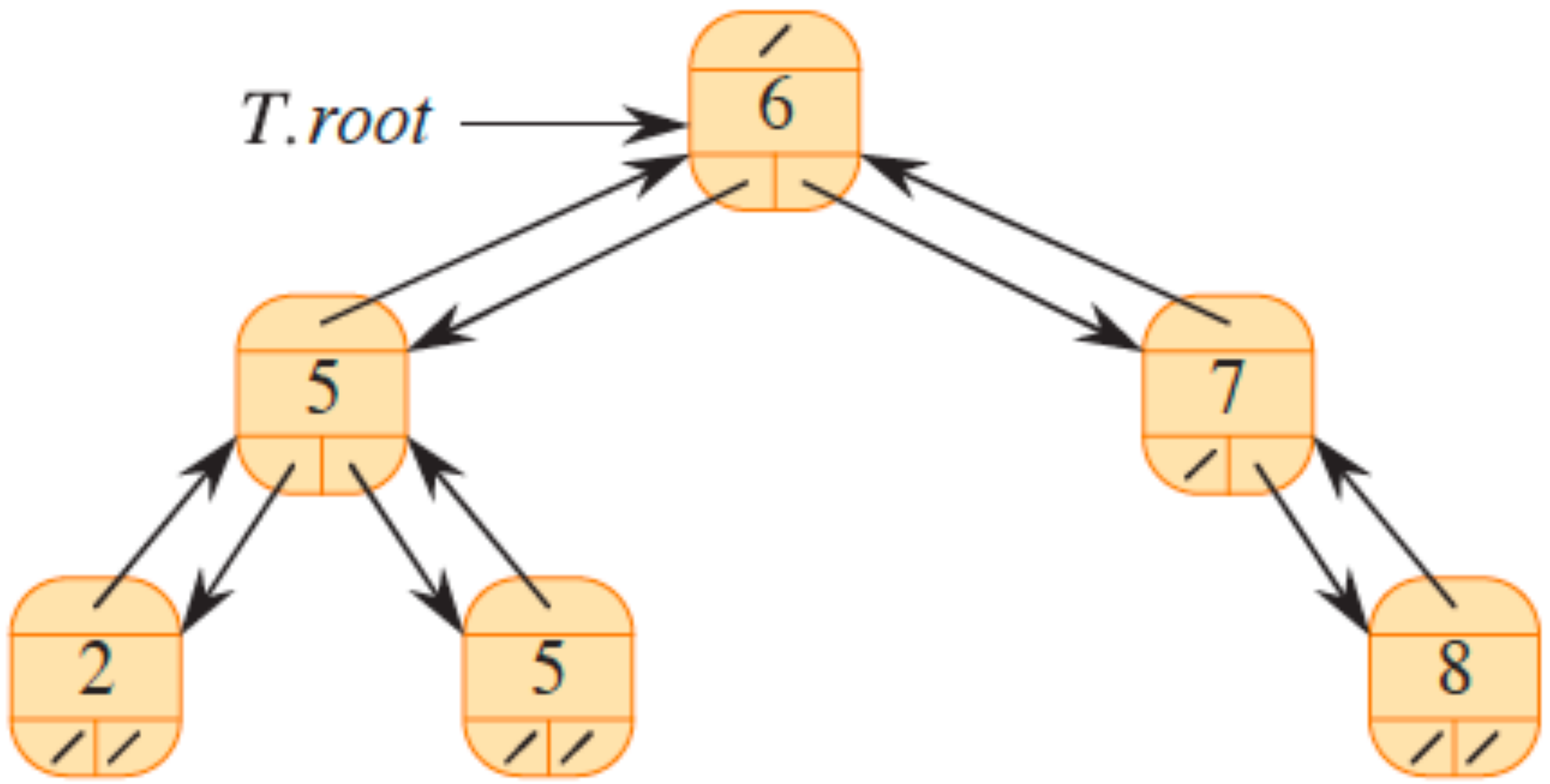
What is a Binary Search Tree?





What is a Binary Search Tree?





Source: CLRS