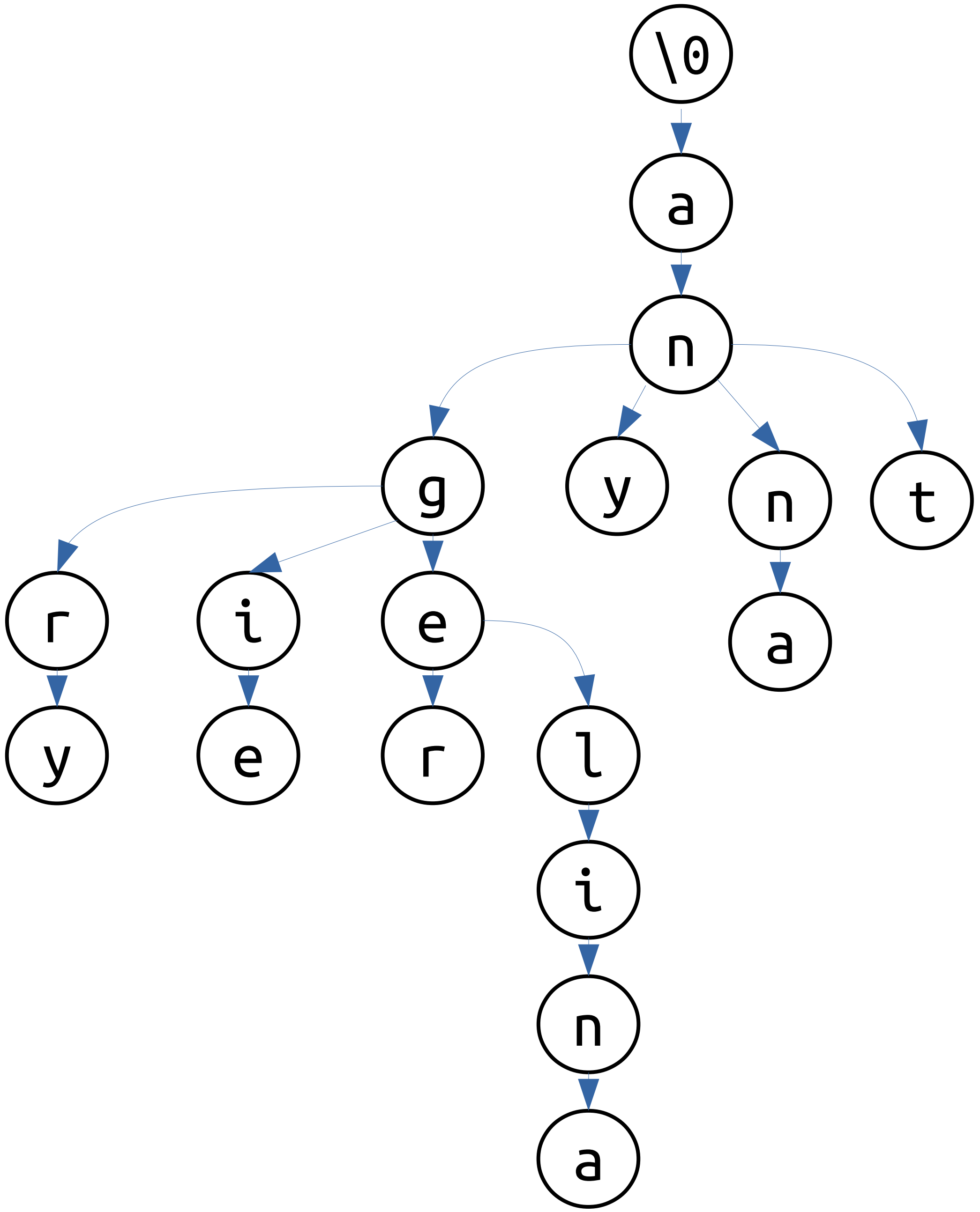
# Trie

**Search tree storing keys. Most often, we will consider strings as keys.**
**Nodes store characters of a key.**

any
ant
anna
anger
angelina
angie
angry
angel

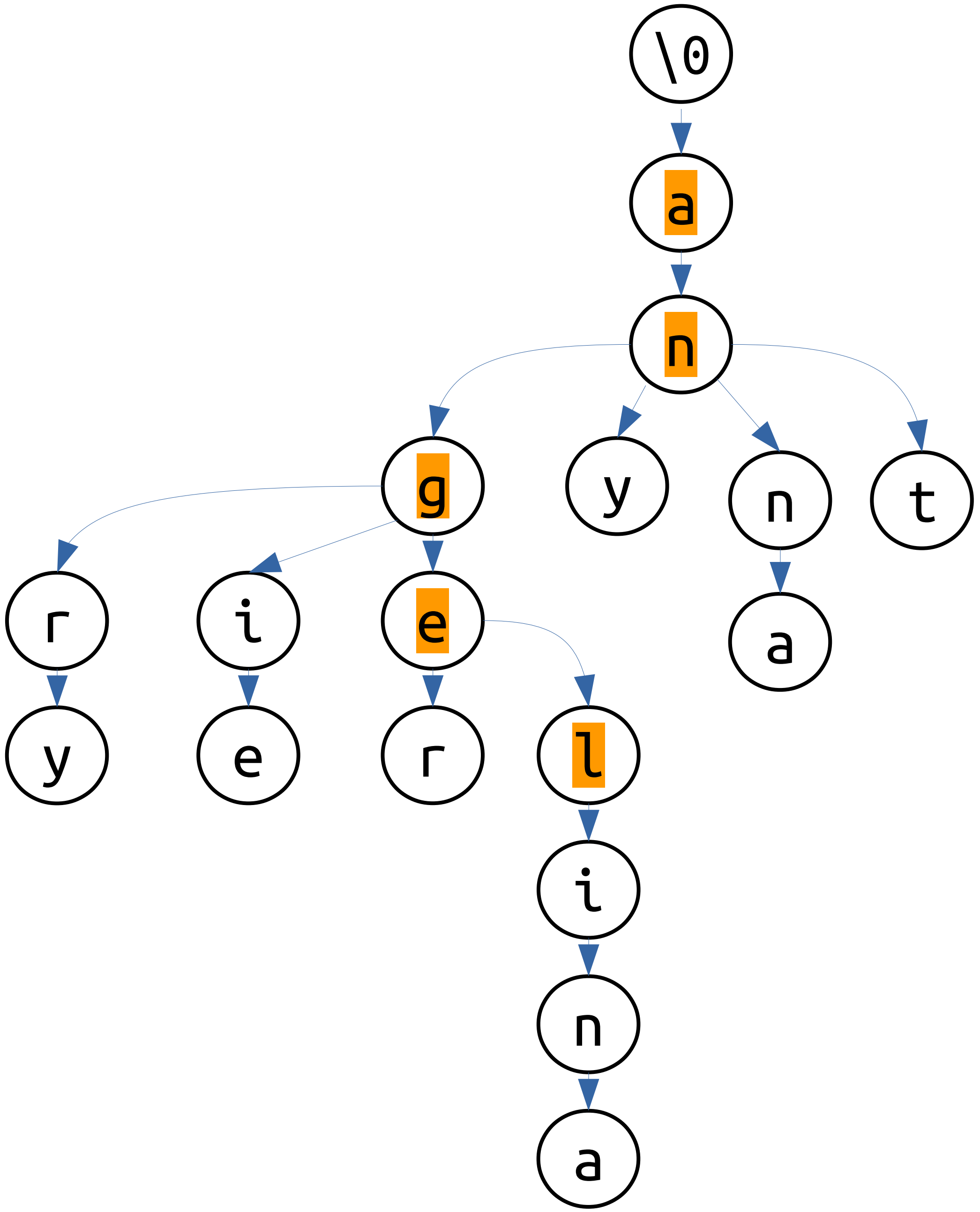


## Trie

**Search tree storing keys. Most often, we will consider strings as keys.**

Nodes store characters of a key.

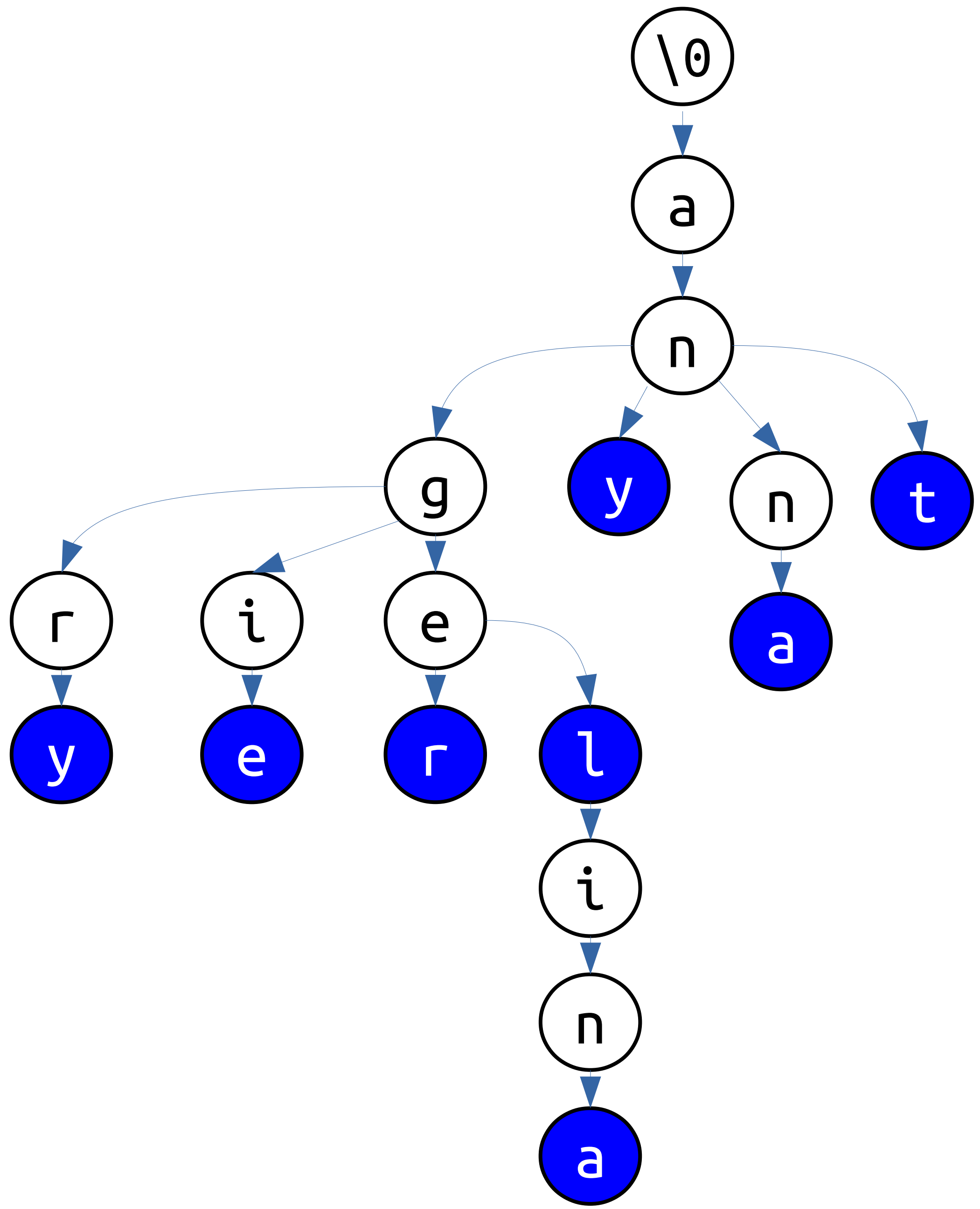
any
ant
anna
anger
angelina
angie
angry
angel



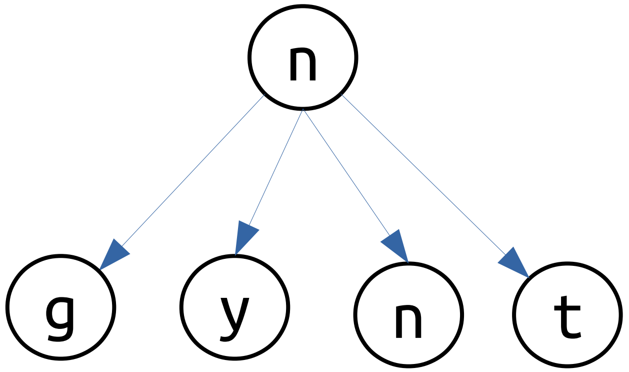
## Trie

any
ant
anna
anger
angelina
angie
angry
angel

**Nodes have an endOfWord boolean variable to know if they are end of a key**



## Trie - Nodes



```
class Node
{
```

**Attributes:**

**- array of Node pointers to children**
**- a character c**
**- a boolean endOfWord**

```
};
```

**How many children?**

**When coding the constructor, which should be the initial values of a node?**

## Trie – Insertion

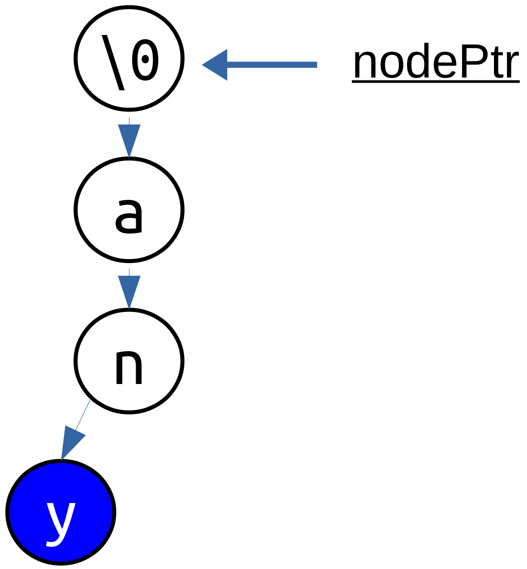
```
void insert(std::string key)
{

   1. Initialize a pointer nodePtr as the root of the trie

   2. Loop through each i-th character of the string key
   {

        if(nodePtr’s child with letter key[i] is null)
             nodePtr’s child with letter key[i] = new Node(key[i]);

        move nodePtr to child with nodePtr’s child with letter key[i];
   }

   3. mark nodePtr→endOfWord = true;

}
```

# Trie – Insertion

```
void insert(std::string key)
{

   1. Initialize a pointer nodePtr as the root of the trie

   2. Loop through each i-th character of the string key
   {

        if(nodePtr’s child with letter key[i] is null)
             nodePtr’s child with letter key[i] = new Node(key[i]);

        move nodePtr to child with nodePtr’s child with letter key[i];
   }

   3. mark nodePtr→endOfWord = true;
}
```

```
insert(“anna”);
```

*key* = anna

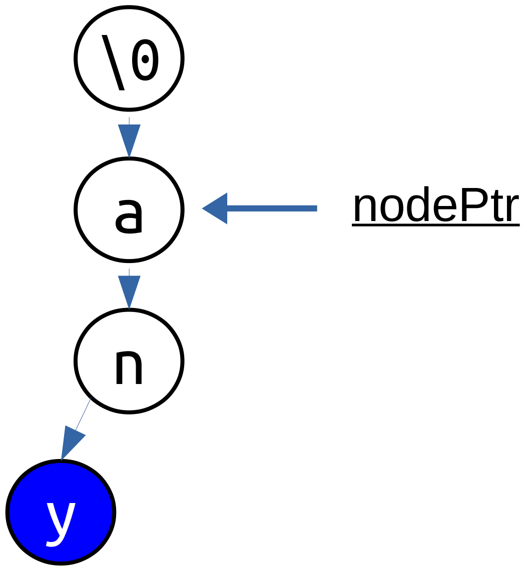


## Trie – Insertion

```
void insert(std::string key)
{

   1. Initialize a pointer nodePtr as the root of the trie

   2. Loop through each i-th character of the string key
   {

        if(nodePtr’s child with letter key[i] is null)
             nodePtr’s child with letter key[i] = new Node(key[i]);

        move nodePtr to child with nodePtr’s child with letter key[i];
   }

   3. mark nodePtr→endOfWord = true;
}
```

```
insert(“anna”);
```

*key* = anna

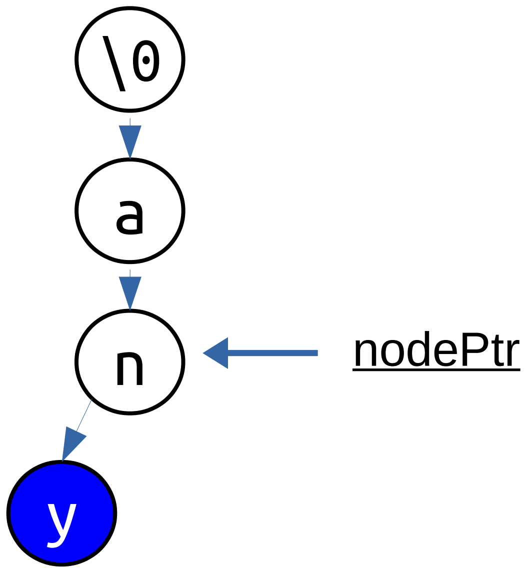


## Trie – Insertion

```
void insert(std::string key)
{

   1. Initialize a pointer nodePtr as the root of the trie

   2. Loop through each i-th character of the string key
   {

        if(nodePtr’s child with letter key[i] is null)
             nodePtr’s child with letter key[i] = new Node(key[i]);

        move nodePtr to child with nodePtr’s child with letter key[i];
   }

   3. mark nodePtr→endOfWord = true;

}
```

```
insert(“anna”);
```

*key* = anna

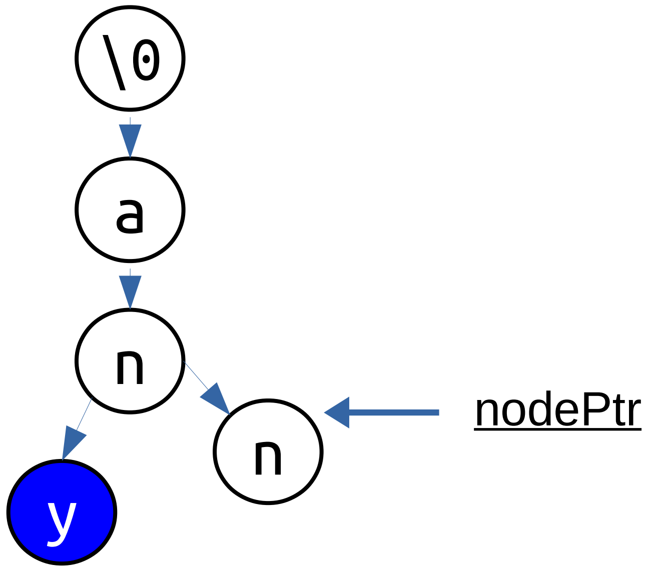


## Trie – Insertion

```
void insert(std::string key)
{

   1. Initialize a pointer nodePtr as the root of the trie

   2. Loop through each i-th character of the string key
   {

        if(nodePtr’s child with letter key[i] is null)
             nodePtr’s child with letter key[i] = new Node(key[i]);

        move nodePtr to child with nodePtr’s child with letter key[i];
   }

   3. mark nodePtr→endOfWord = true;

}
```

```
insert(“anna”);
```

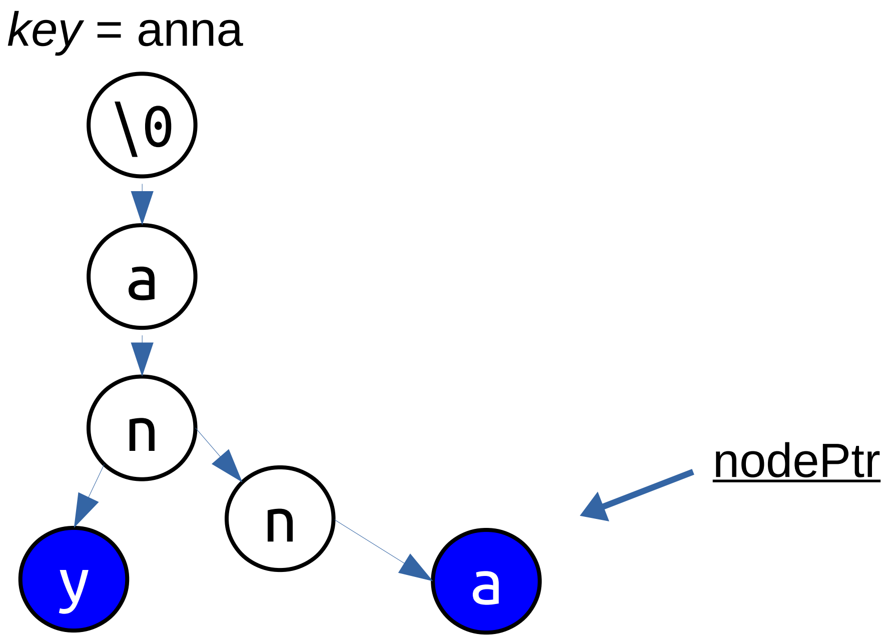
*key* = anna



## Trie – Insertion

```
void insert(std::string key)
{

   1. Initialize a pointer nodePtr as the root of the trie

   2. Loop through each i-th character of the string key
   {

        if(nodePtr’s child with letter key[i] is null)
             nodePtr’s child with letter key[i] = new Node(key[i]);

        move nodePtr to child with nodePtr’s child with letter key[i];
   }

   3. mark nodePtr→endOfWord = true;

}
```

```
insert(“anna”);
```



## Trie – Search

```
bool search(std::string key)
{
   1. Initialize a pointer nodePtr as the root of the trie

   2. Loop through each i-th character of the string key
   {

       if(nodePtr’s child with letter key[i] is null)
           return false;

       move nodePtr to child with nodePtr’s child with letter key[i];
   }

}  return nodePtr→endOfWord;
```