DIGITAL SEARCH TREE

What is DST?

Digital search tree is a binary tree in which each node contains only binary data.

If the bit of DST starts with 0 then it is in left subtree and if the bit starts with 1 then it is in right subtree and this process works recursively.

All remaining pairs whose key begins with a 0 are in the left sub-tree.

All remaining pairs whose key begins with a 1 are in the right sub-tree.

Left and right sub-trees are digital sub-trees on remaining bits.

Example

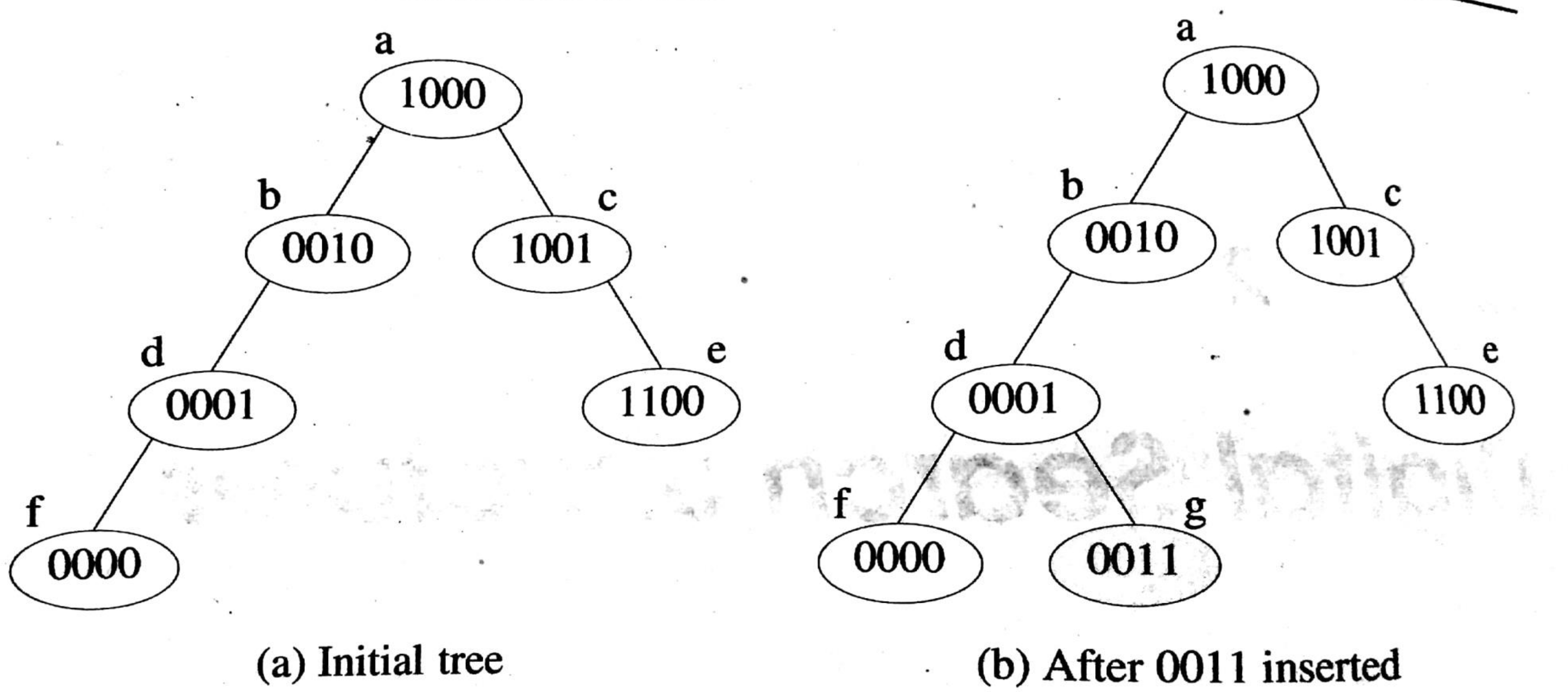
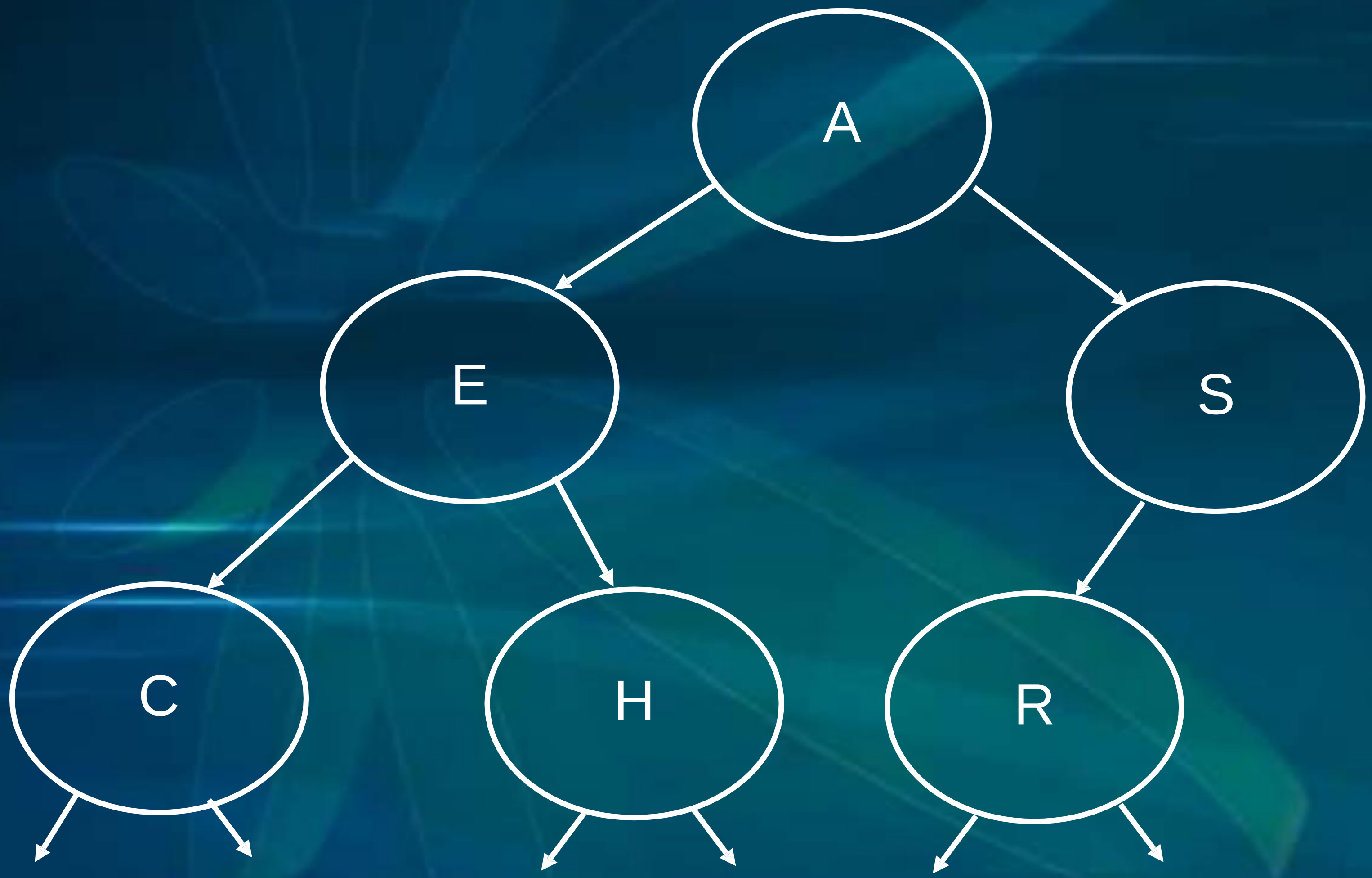


Figure 12.1: Digital search trees

Digital Search Tree Example

A 00001
S **1**0011
E **0**0101
R **10**010
C **00**011
H **01**000



Search Time Of DST

★ Searching is based on binary representation of data.

★ If the data are randomly distributed then the average search time per operation in O(log N), where N is the height of the tree.

★ However, Worst case is O(b), where b is the number of bits in the search key.

Application Of DST

- IP routing.
  - IPv4 – 32 bit IP address.
  - IPv6 – 128 bit IP address.

- Firewalls.

DST vs BST

- Insertion, search and deletion in DST are easier than the Binary search tree and AVL tree.
- This tree does not required additional information to maintain the balance of the tree because the depth of the tree is limited by the length of the key element.
- DST requires less memory than Binary search tree and AVL tree.

Drawbacks Of DST

- Bitwise operations are not always easy.
- Handling duplicates isproblematic.
- Similar problem with keys of different lengths.
- Data is not sorted.
- If a key is long search and comparisons are costly, this can be problem

Insertion of DST

- To insert an element in DST. There will be four(4) possible cases.
  - Tree Empty
  - If found ‘0’, then go left
  - If found ‘1’ then go right
  - If found same key, then insert with prefix equal

Insertion of DST (Cont.)

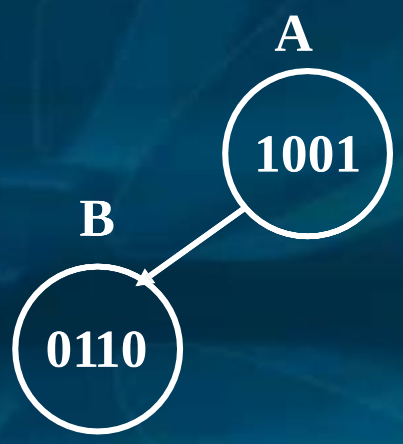
- Start with an empty digital search tree and insert a pair whose key is **1001**

Insertion of DST (Cont.)

- Start with an empty digital search tree and insert a pair whose key is **1001**

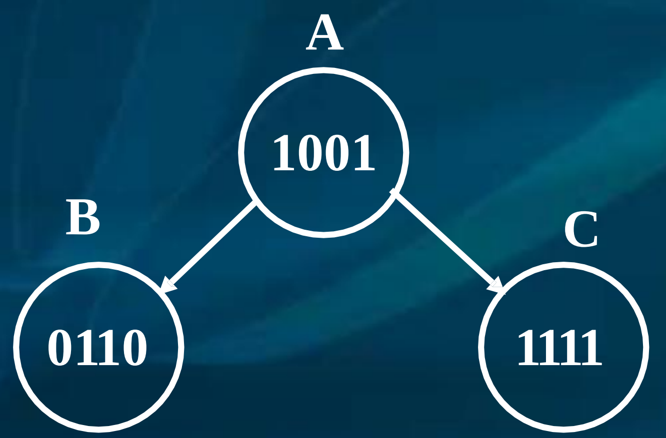
Now, insert a pair whose key is **0110**

Insertion of DST (Cont.)

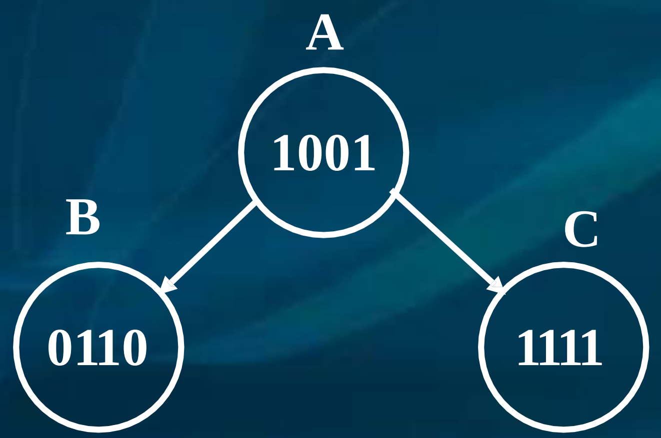


- Now, insert a pair whose key is **1111**

Insertion of DST (Cont.)

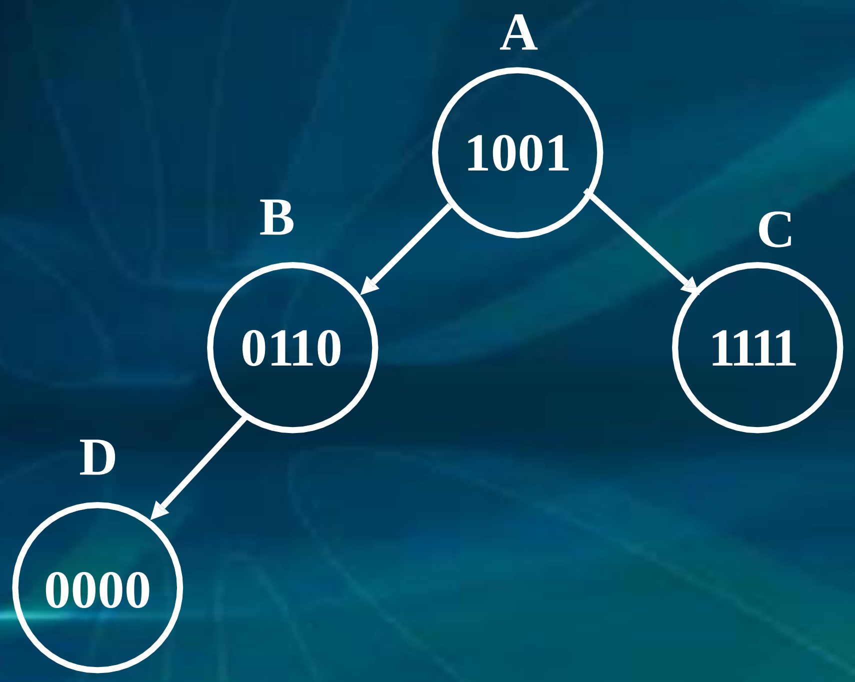


Insertion of DST (Cont.)

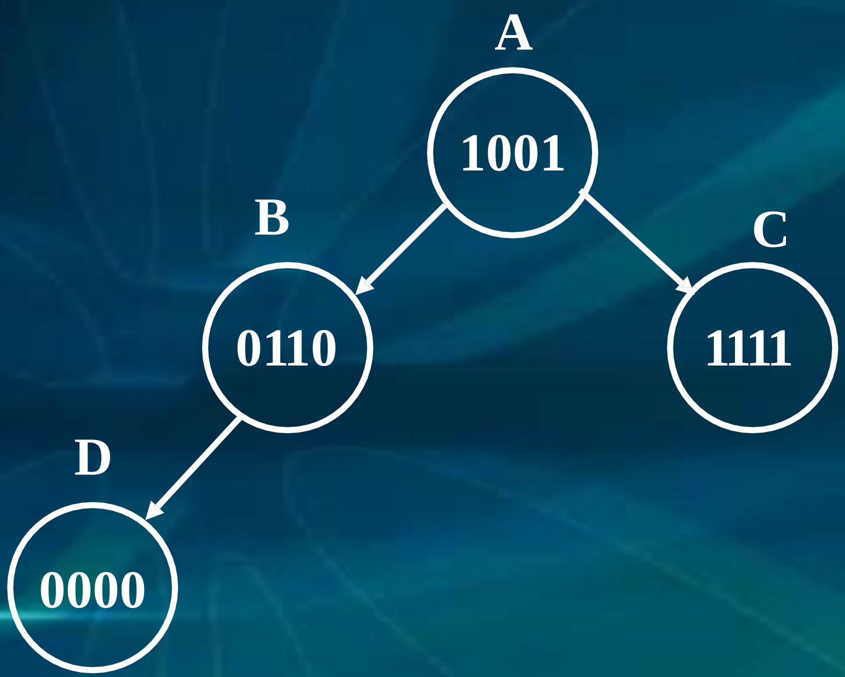


- Now, insert a pair whose key is **0000**

Insertion of DST (Cont.)

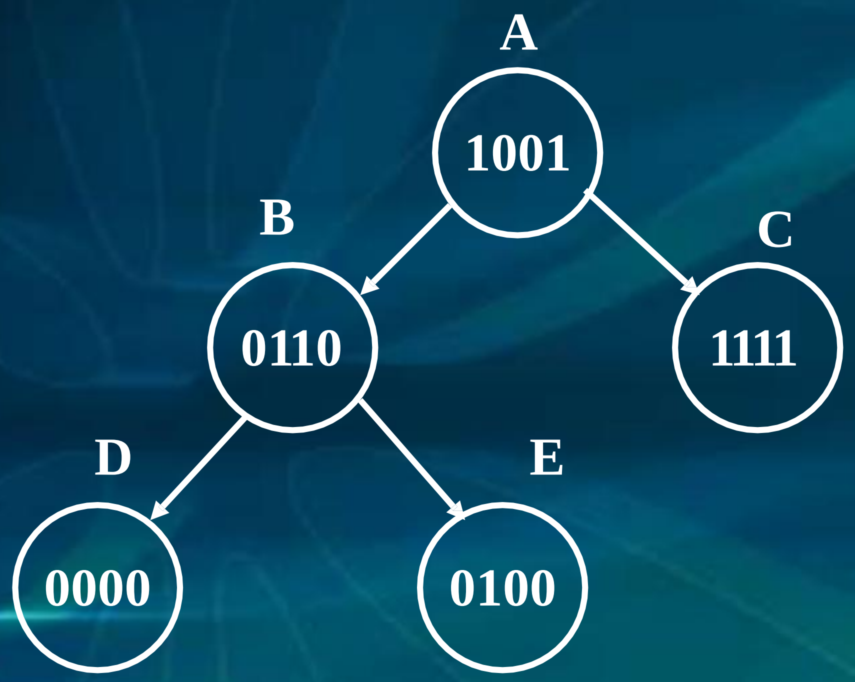


Insertion of DST (Cont.)

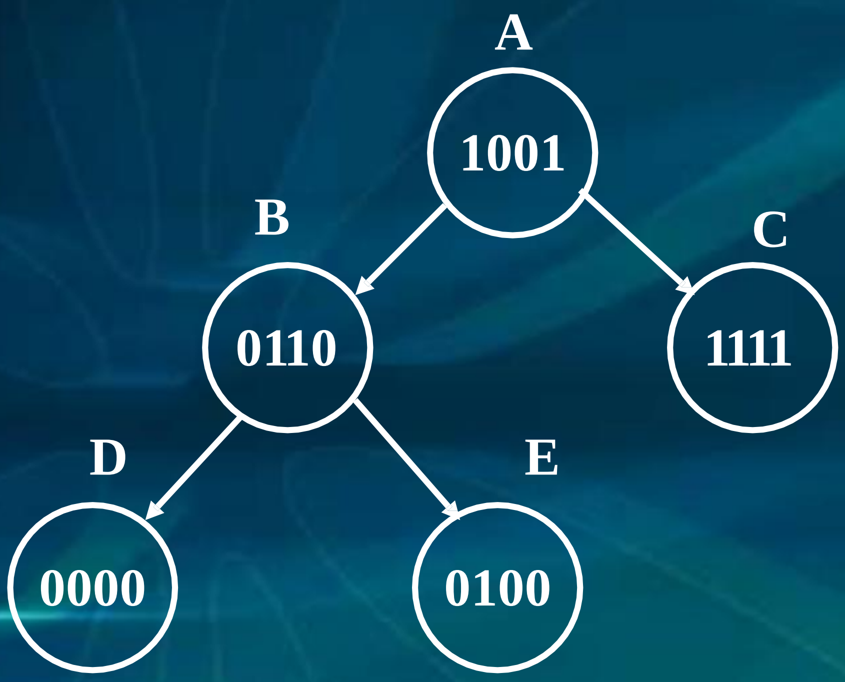


- Now, insert a pair whose key is **0100**

Insertion of DST (Cont.)

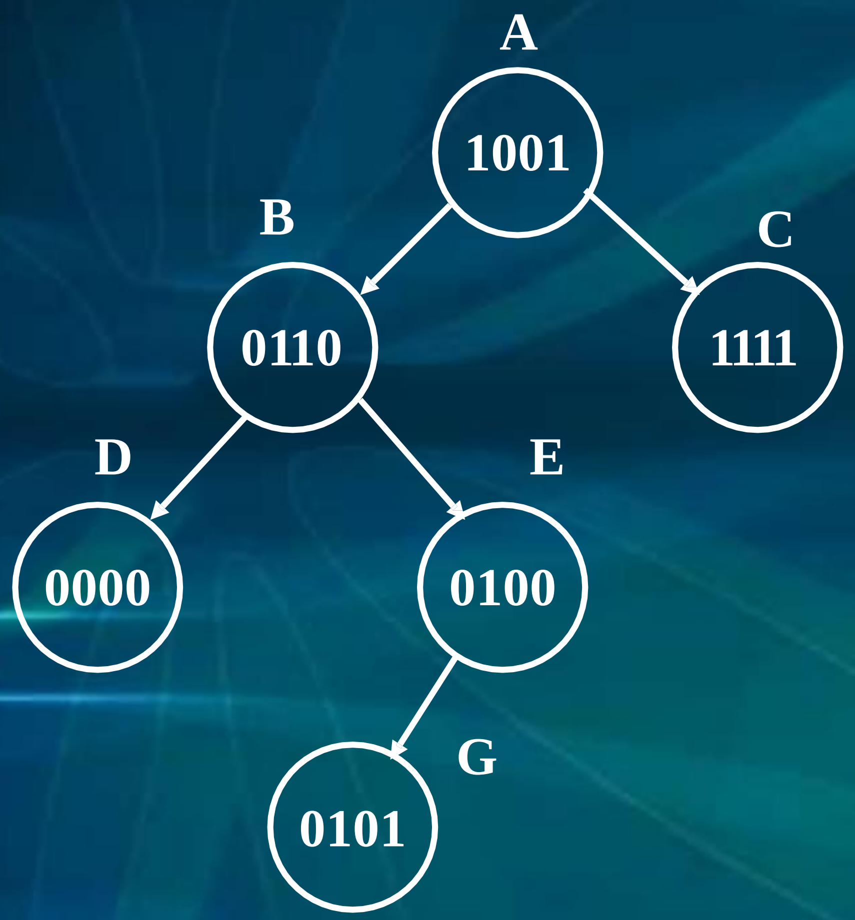


Insertion of DST (Cont.)

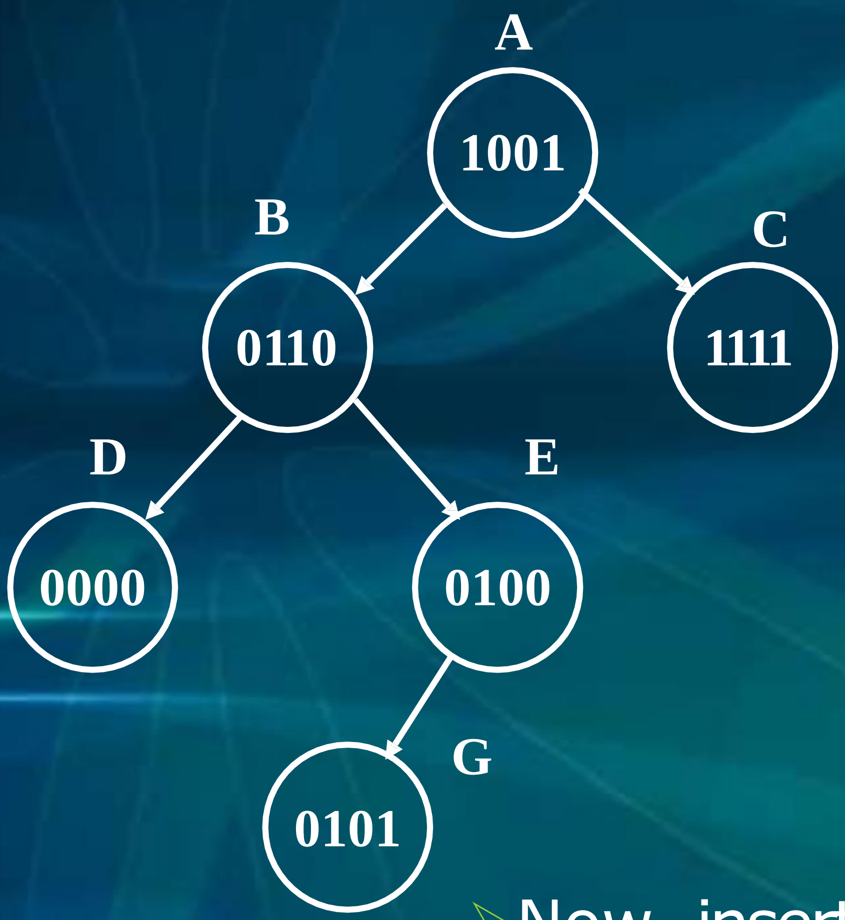


- Now, insert a pair whose key is **0101**

Insertion of DST (Cont.)

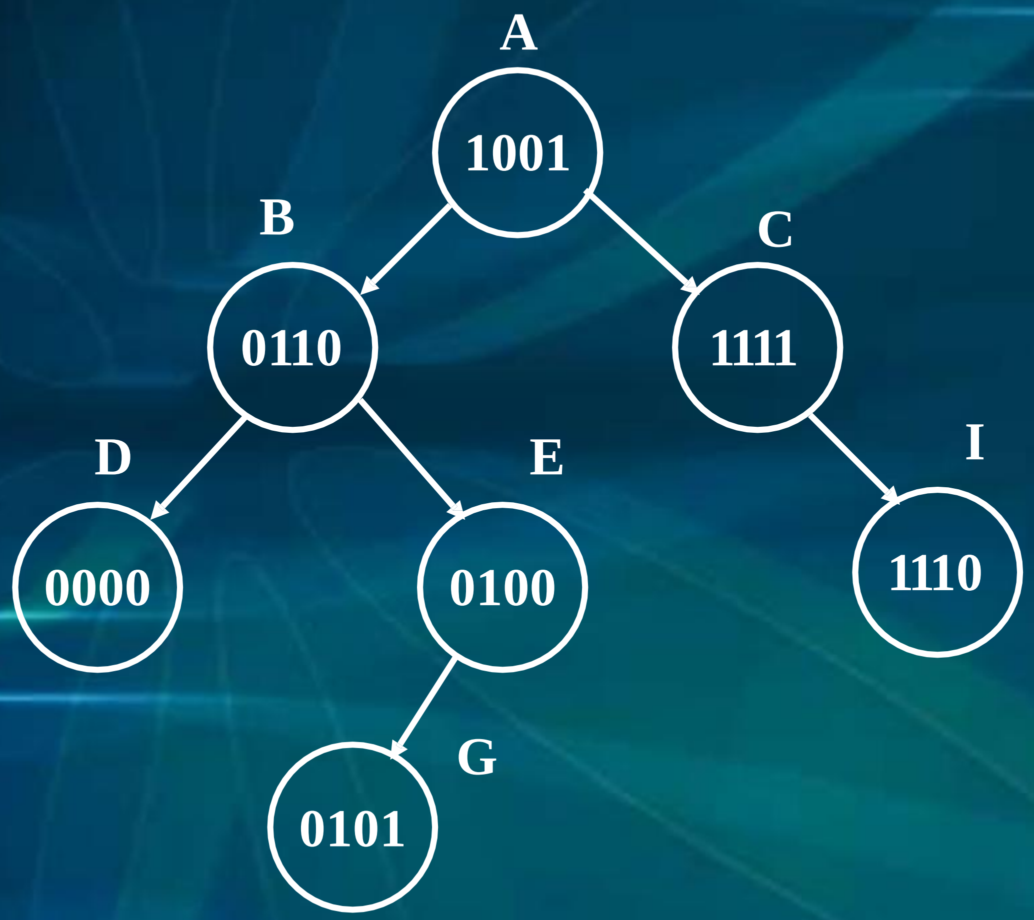


Insertion of DST (Cont.)

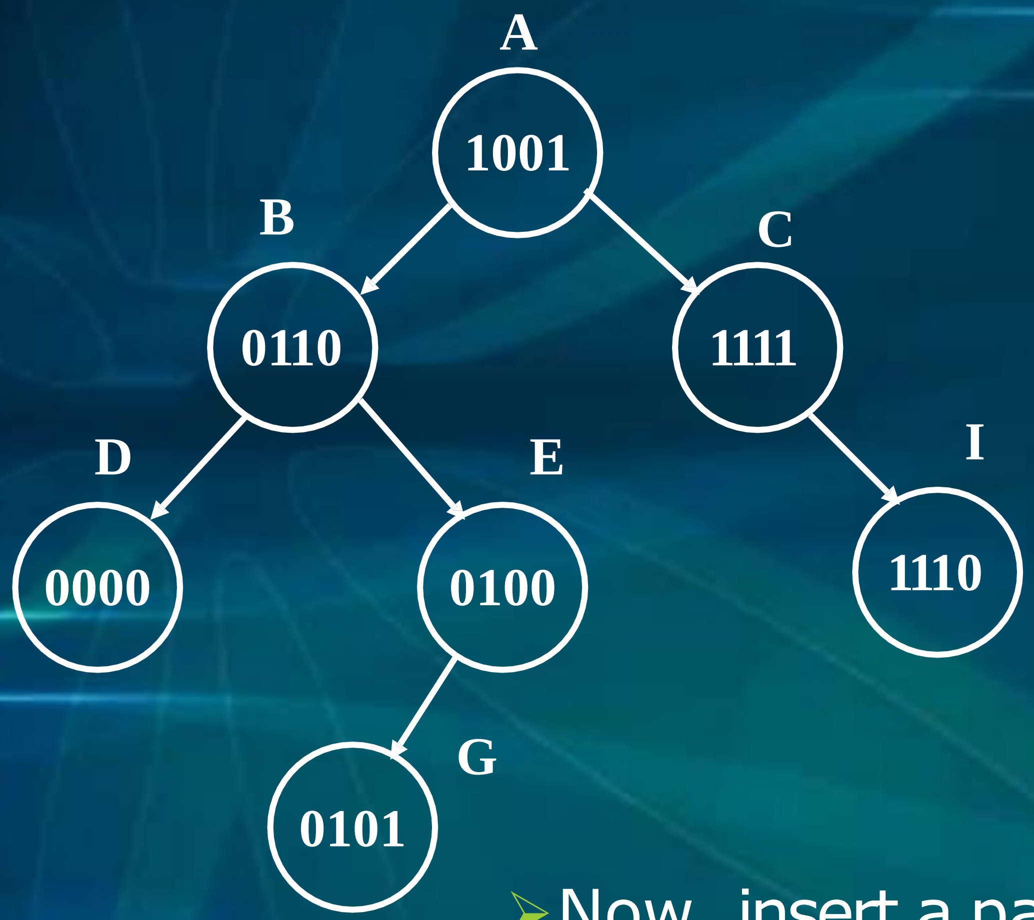


- Now, insert a pair whose key is **1110**

Insertion of DST (Cont.)

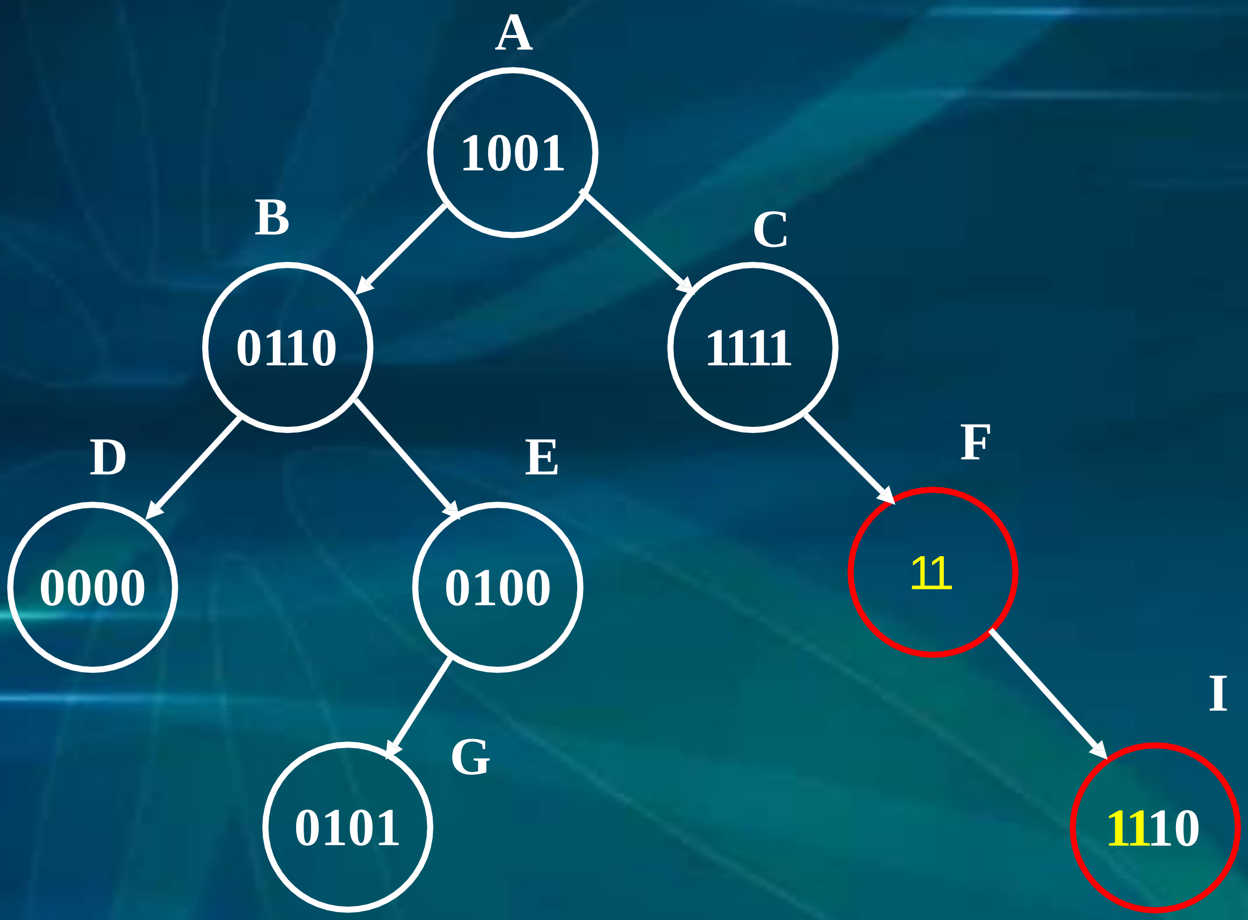


Insertion of DST (Cont.)

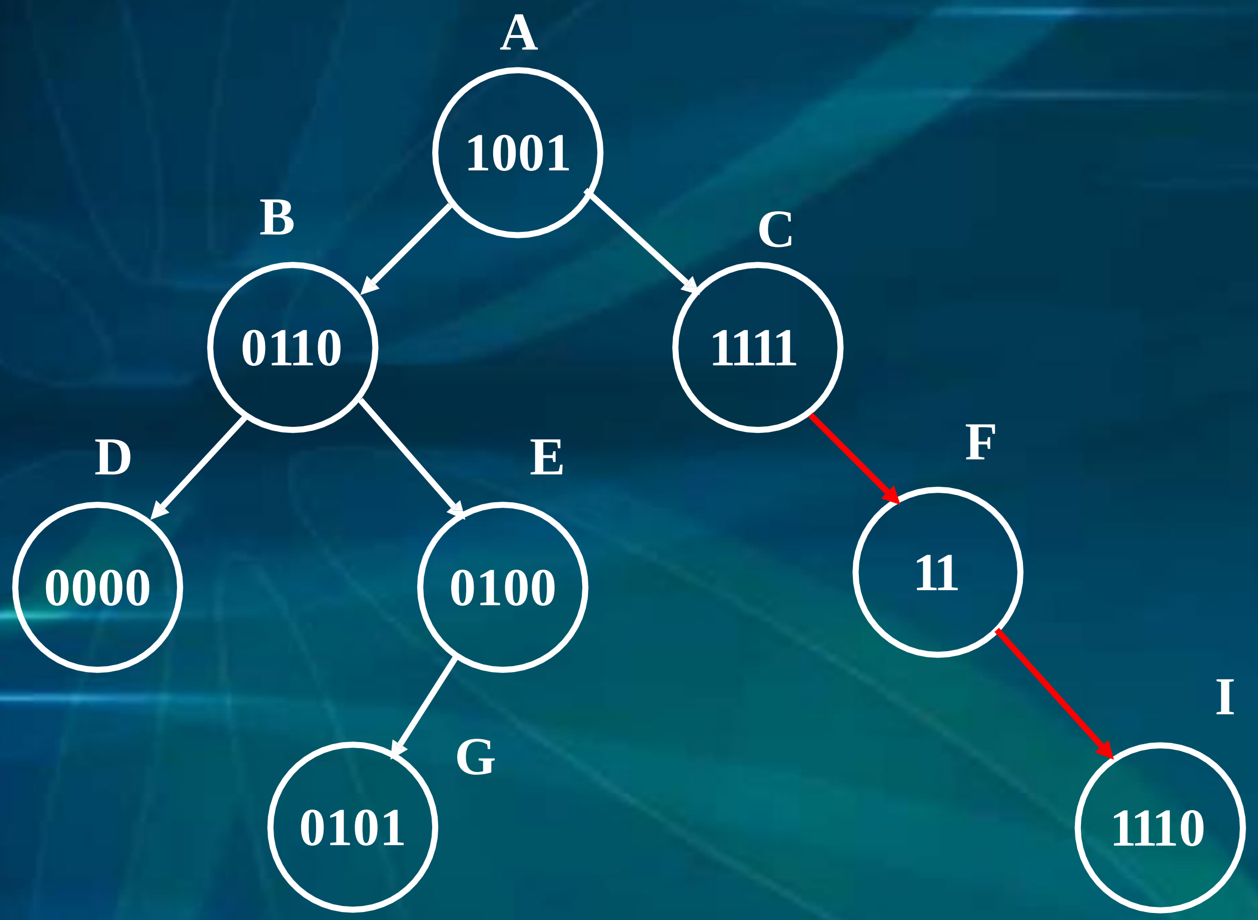


- Now, insert a pair whose key is **11**

Insertion of DST (Cont.)



Insertion of DST (Cont.)



Insertion Pseudo Code of DST

insert()

To insert an item, with a key, k, we begin a search from the root node to locate the insertion position for the item.

```
➢ if t->root is null then
        {
        t->root = new node for the item with key k;
        return null;
        }
    p = t->root;
    i = max_b;
```

Insertion Pseudo Code of DST

```
loop
        {
        if p->key == k then a matching item has been found
        return p->item;
        i = i - 1; /*Traverse left or right branch, depending on the current bit.*/

let j be the value of the (i)th bit of k;
                if p->a[j] is null then
                {
                        p->a[j] = new node for the item with key k;
                        return null;
                 }
        p = p->a[j];
        }
```

Insertion Pseudo Code of DST

In the above pseudo-code, insertion fails if there is already an item with key k in the tree, and a pointer to the matching item will be returned.

Otherwise, when insertion is successful, a null pointer is returned. When the new node, x, is created, its fields are initialized as follows.

```
x->key = k;
x->item = item;
x->a[0] = x->a[1] = NULL;
```

Search

DST search for key K

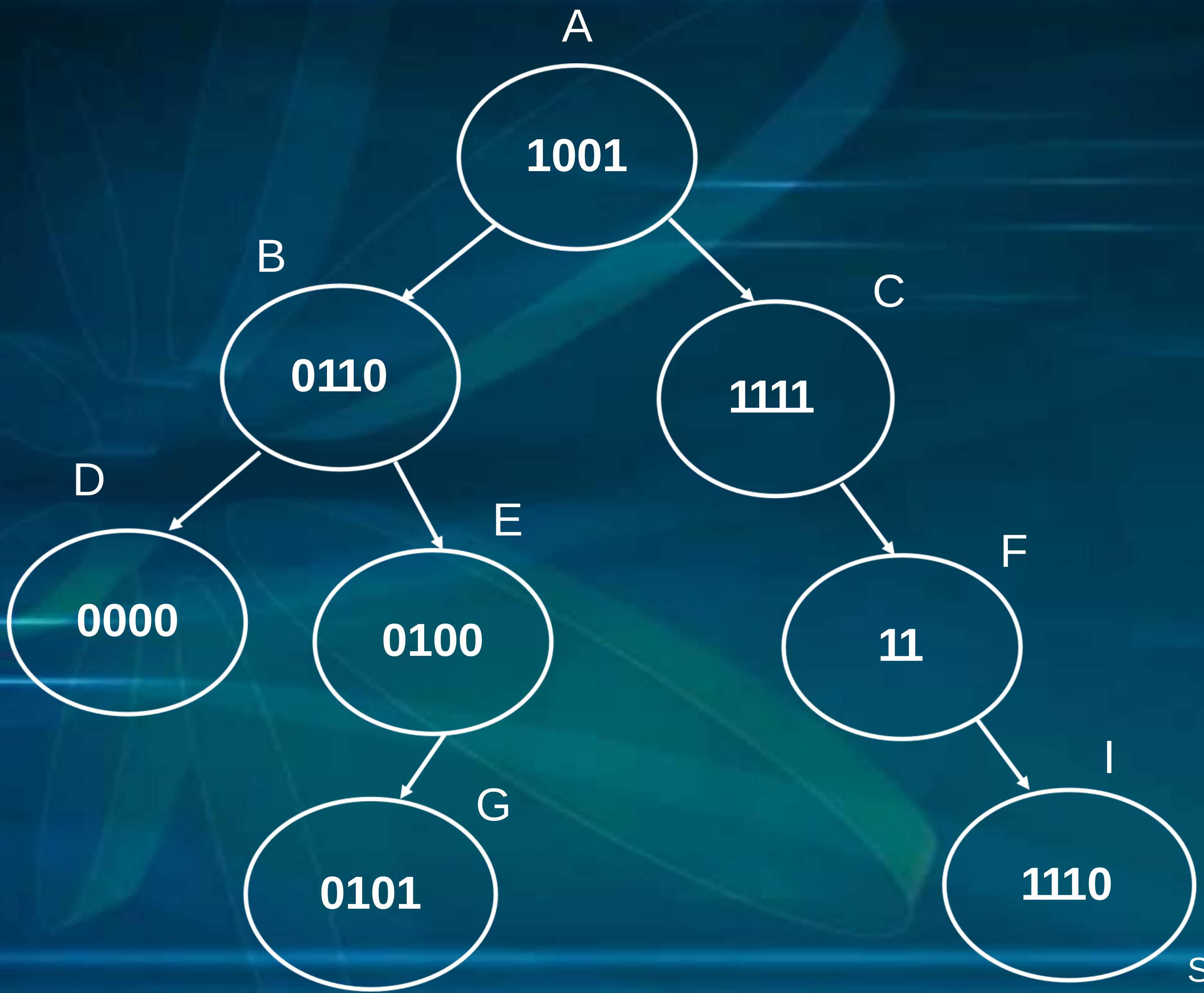
For each node T in the tree we have 4 possible results

- T is empty
- K matches T
- Current bit of K is a 0 and go to left child
- Current bit of K is a 1 and go to right child

Example

Search 0001

NOT FOUND

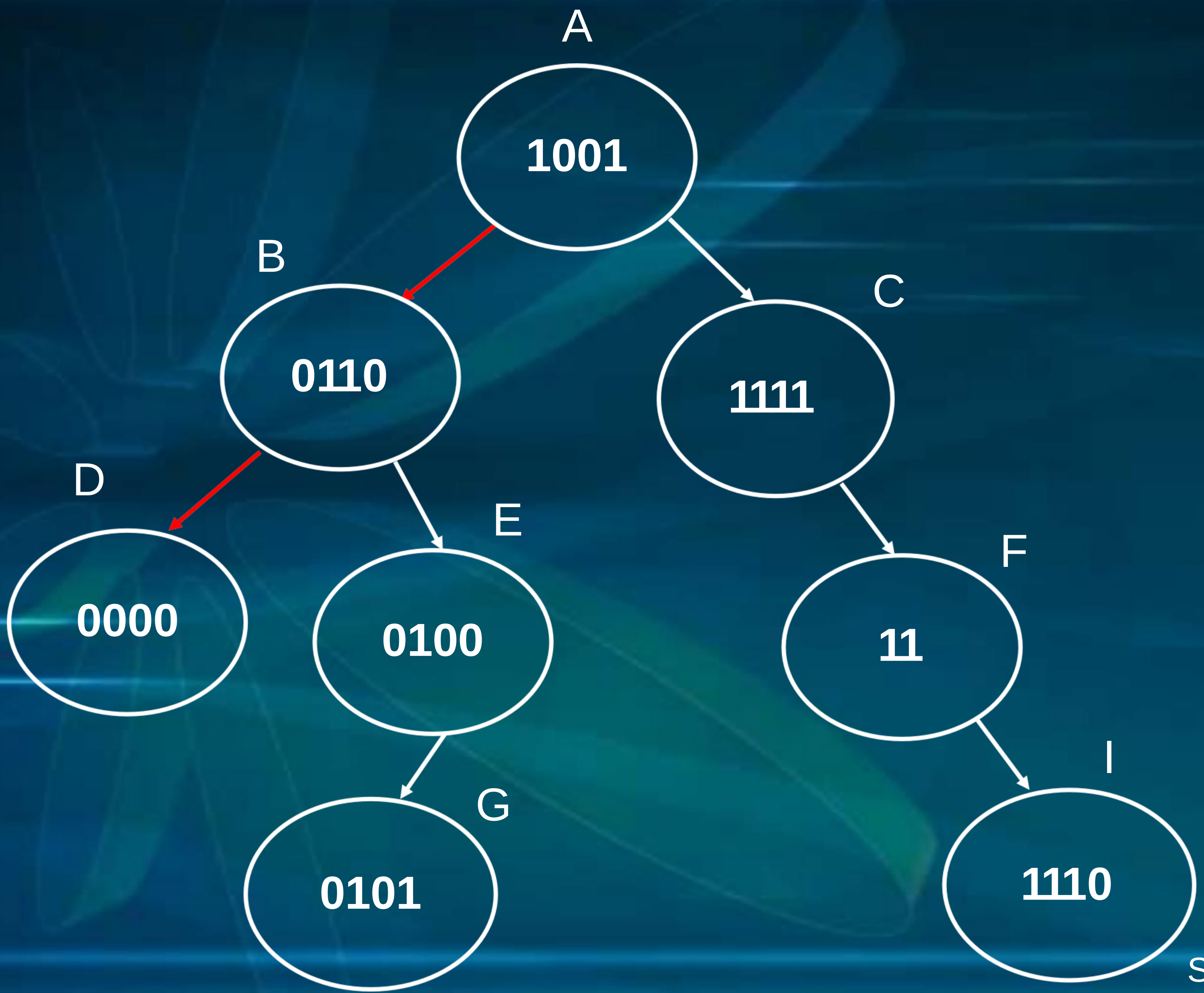


Example

Search 0000

Now 0000=D

So K found

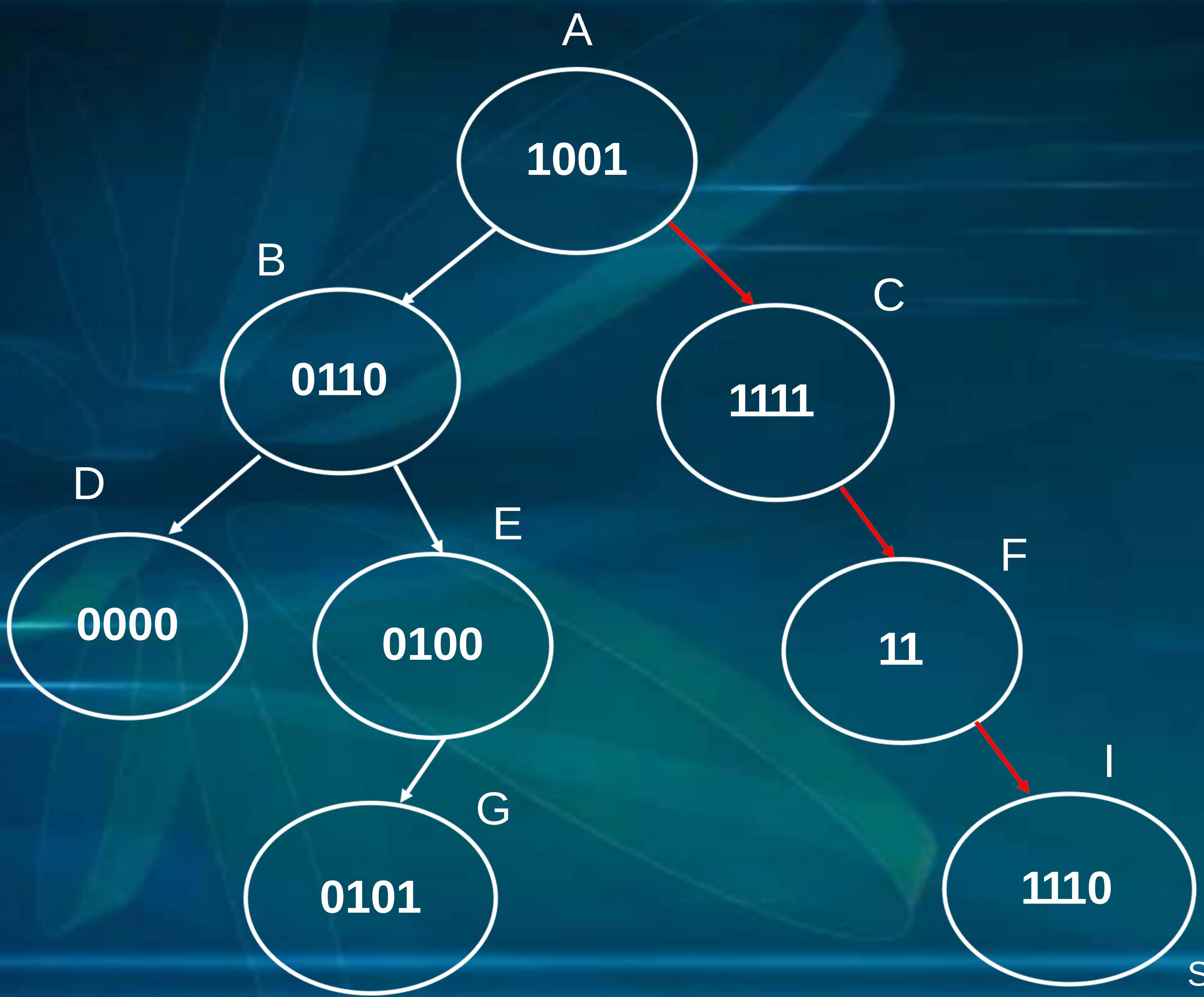


Example

Search 1110

Now 1110=I

So K found

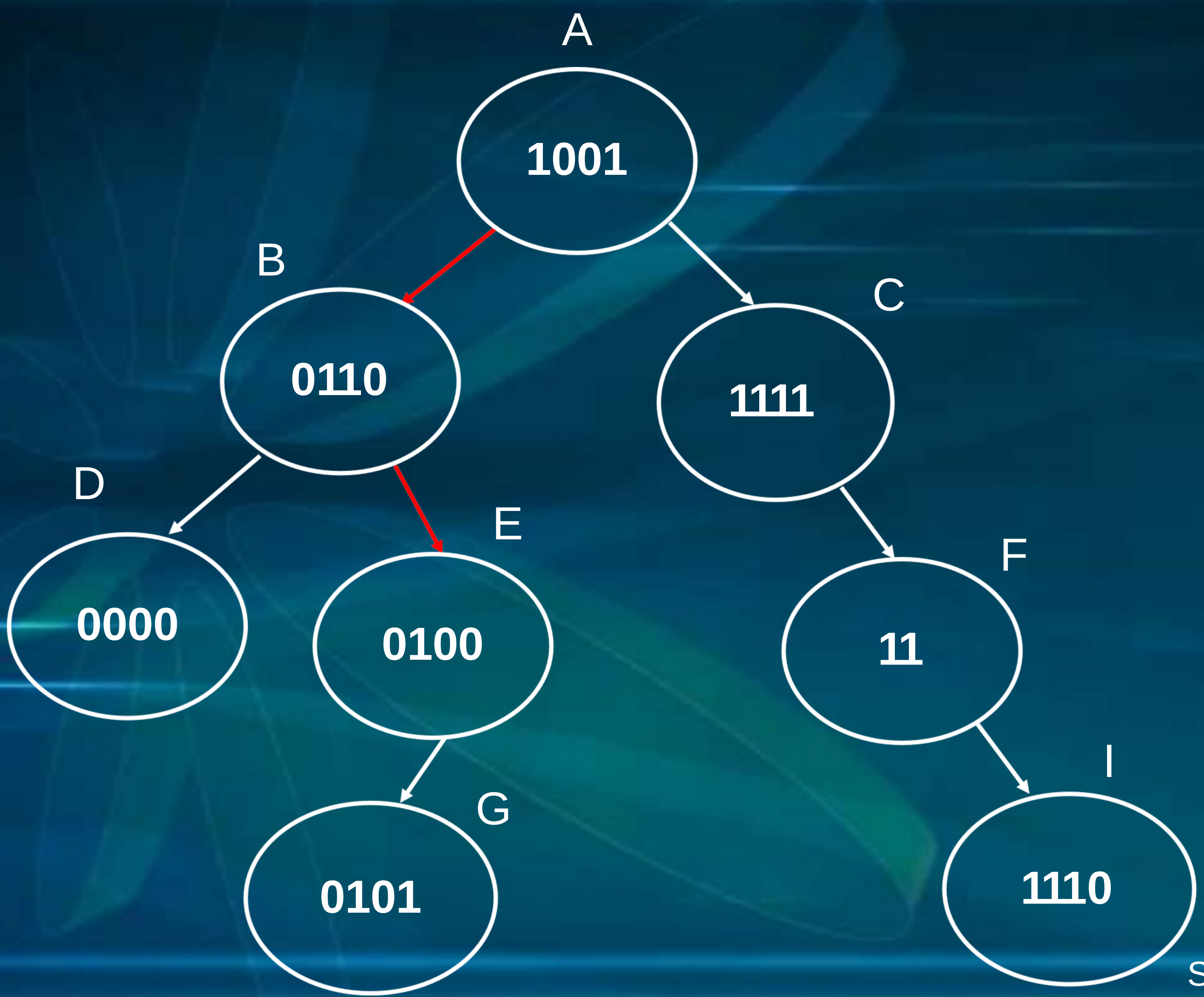


Example

Search 0100

Now 0100=E

So K found



C code of DST Search

```
Struct node{
        int key,info;
        struct node *l,*r ;
}
  static struct node *head,*z;

  unsigned bits(unsigned x, int k, int j)
        return (x>>k) & ~(~0<<j);
Int digital_search(int v)
{
      struct node *x=head;
      int b=maxb;  // maxb is the number of bits in the key to be sorted
      z->key=v;
      while(v!=x->key)
          x=(bits(v,b--,1) ? x->r : x->l;
      return x->info;
}
```

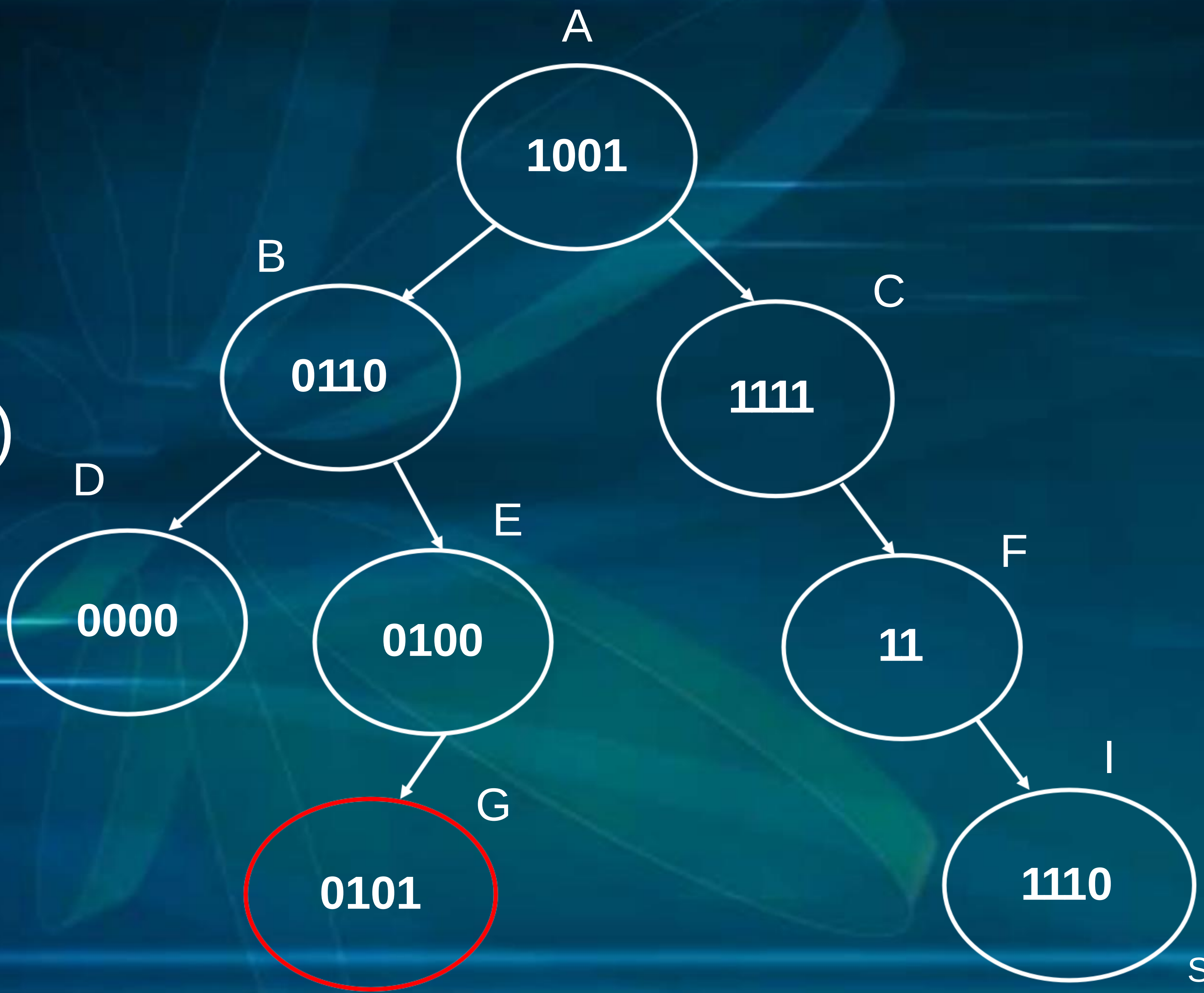
Delete

For each node T in the tree we have 3 possible cases.

- No child
- One child
- Two children

Example

Delete G(0101)



Example

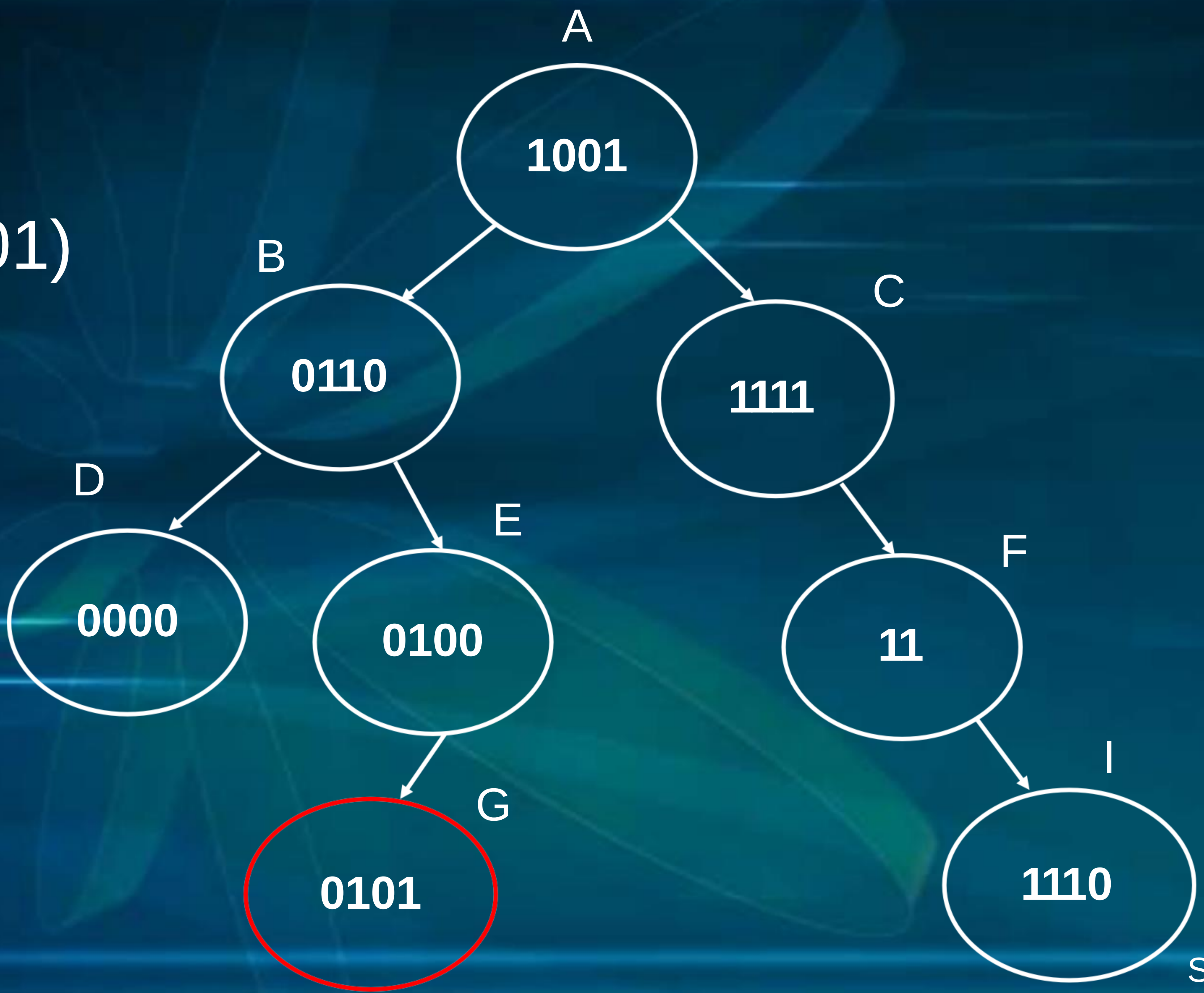
Delete G(0101)

G has no child,

So simply remove G

And replace

by a NIL pointer

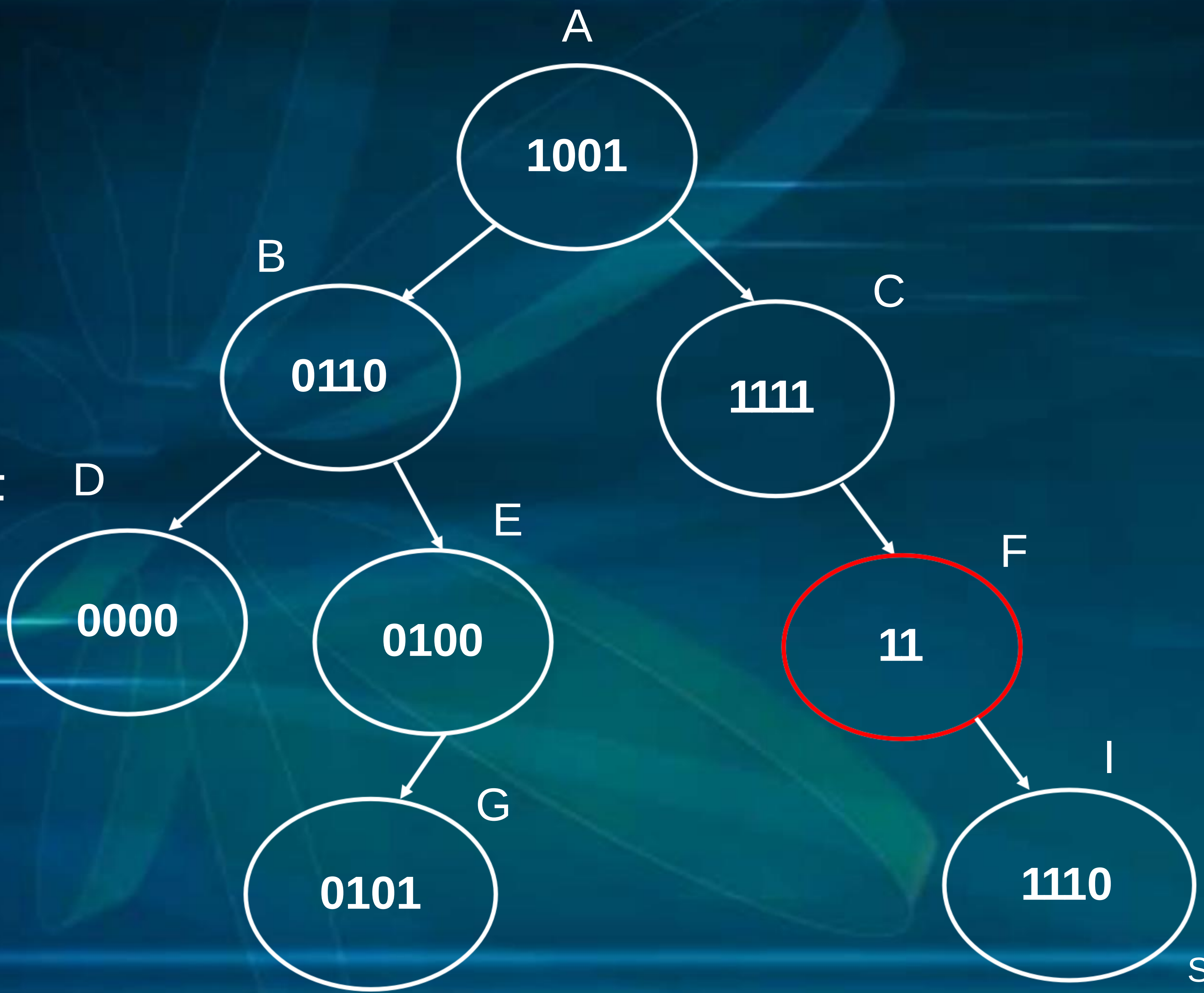


Example

Delete F(11)

F has one child,

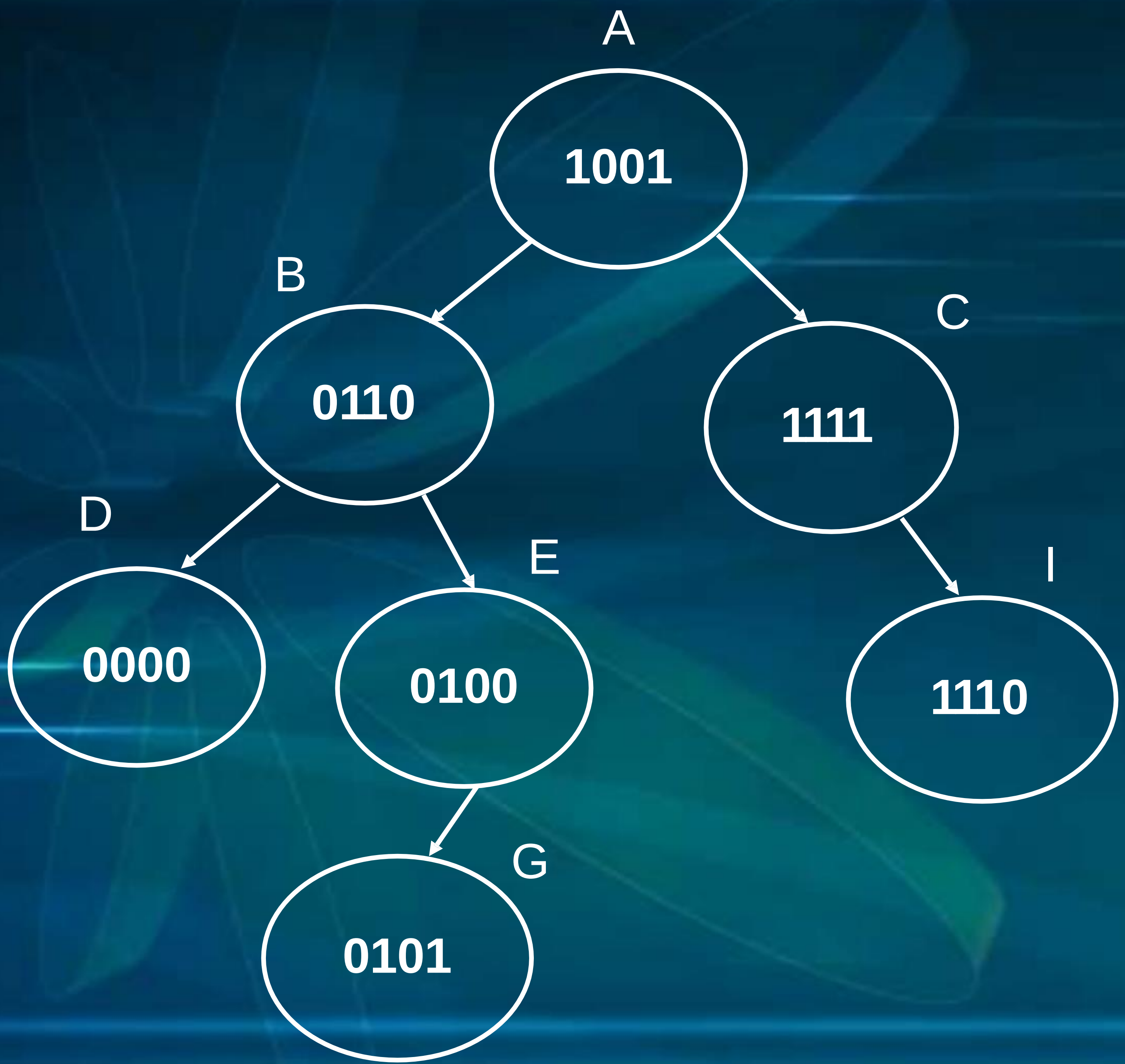
So, first remove F



Example

Delete F(11)

And link C with I

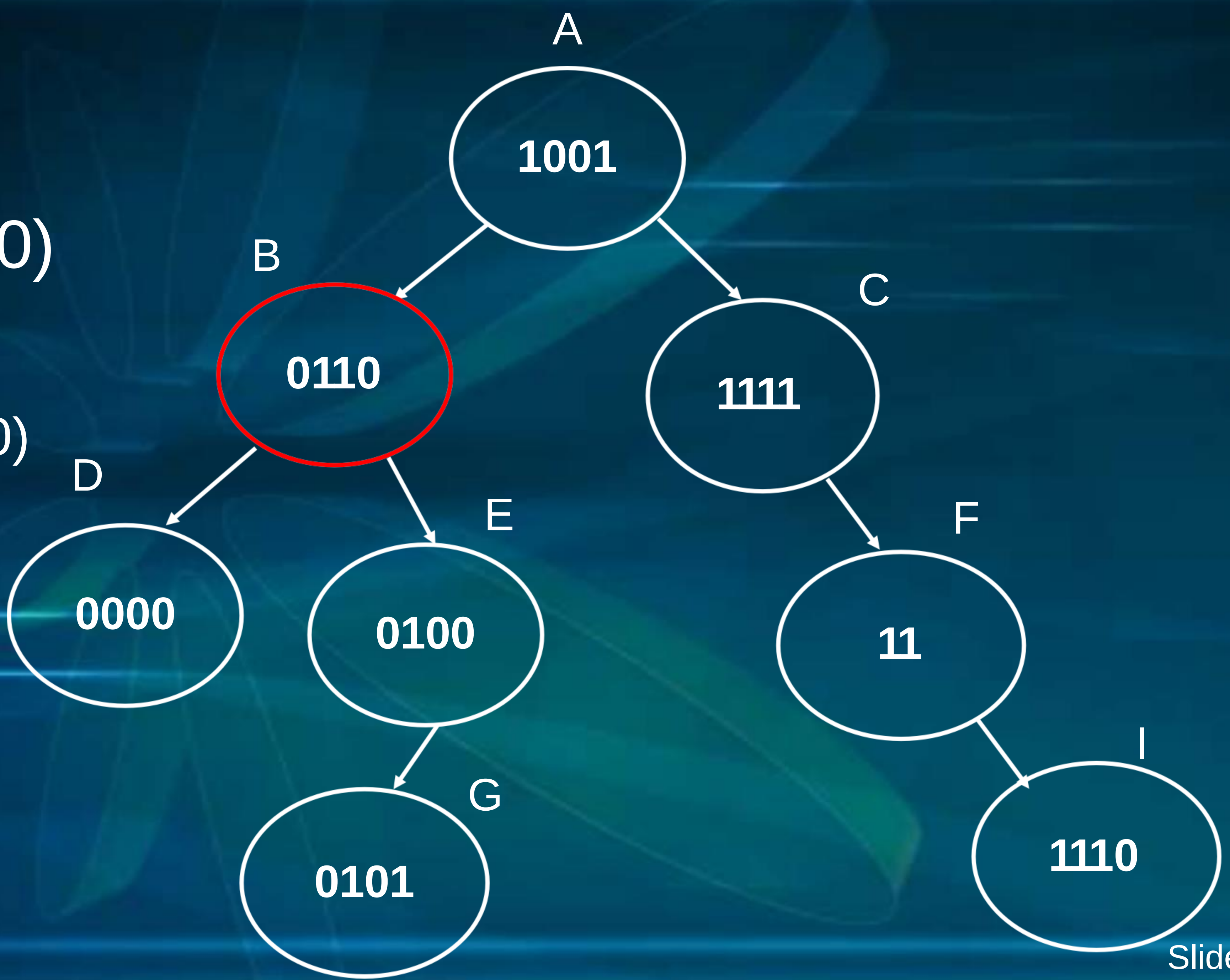


Example

Delete B(0110)

B has two children

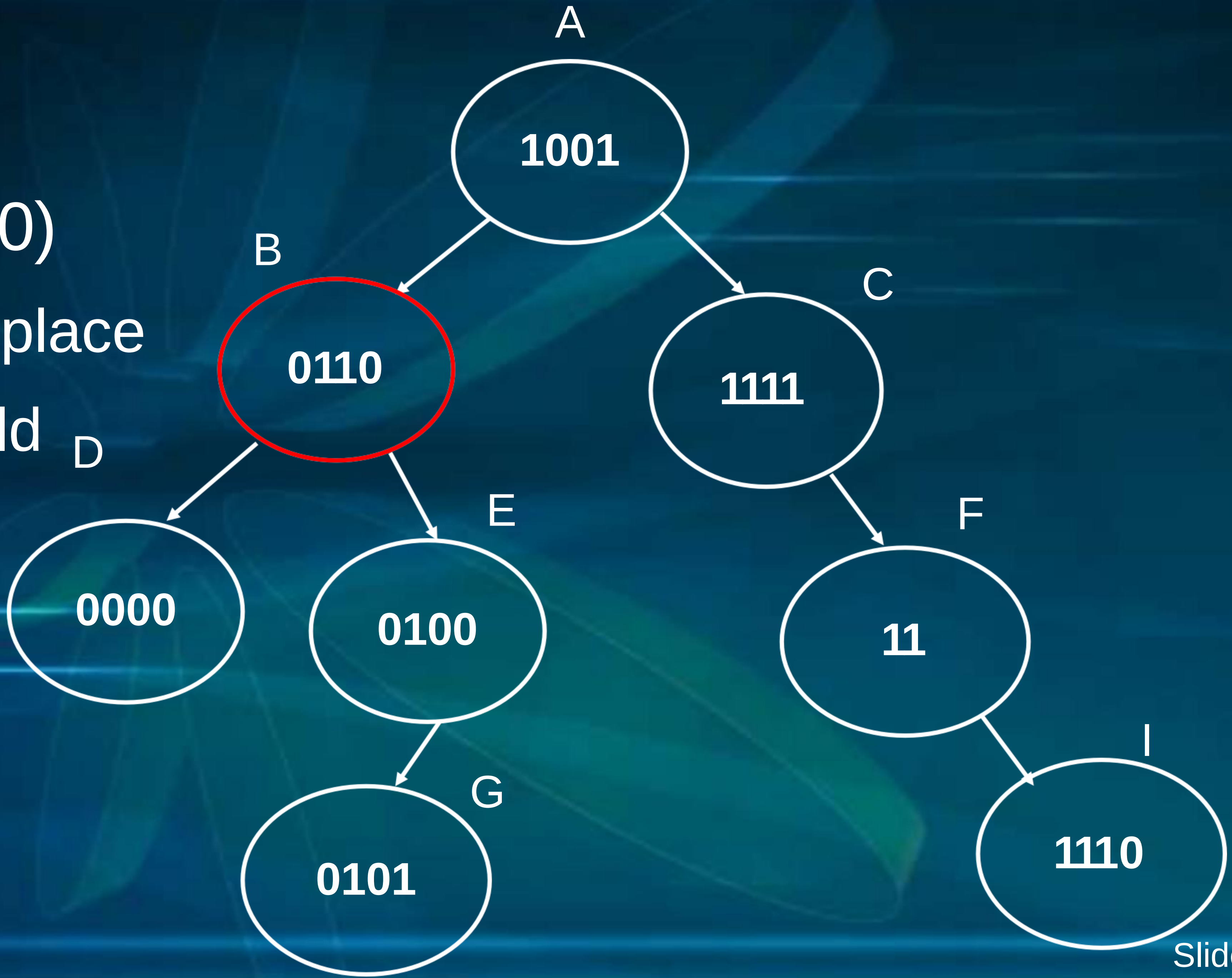
D(0000) and E(0100)



Example

Delete B(0110)

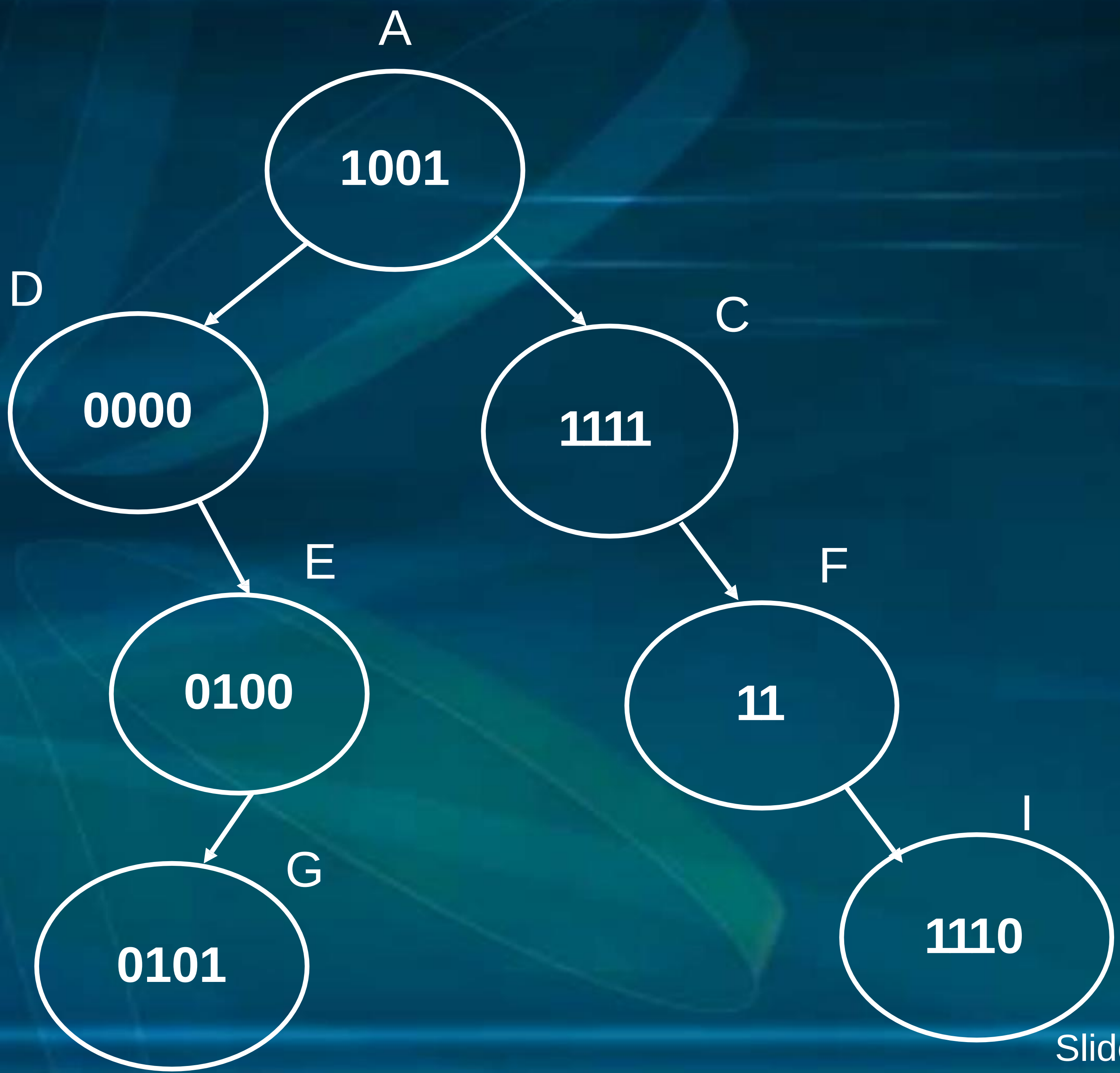
Remove B and replace with one of its child



Example

Delete B(0110)

Replace B with D

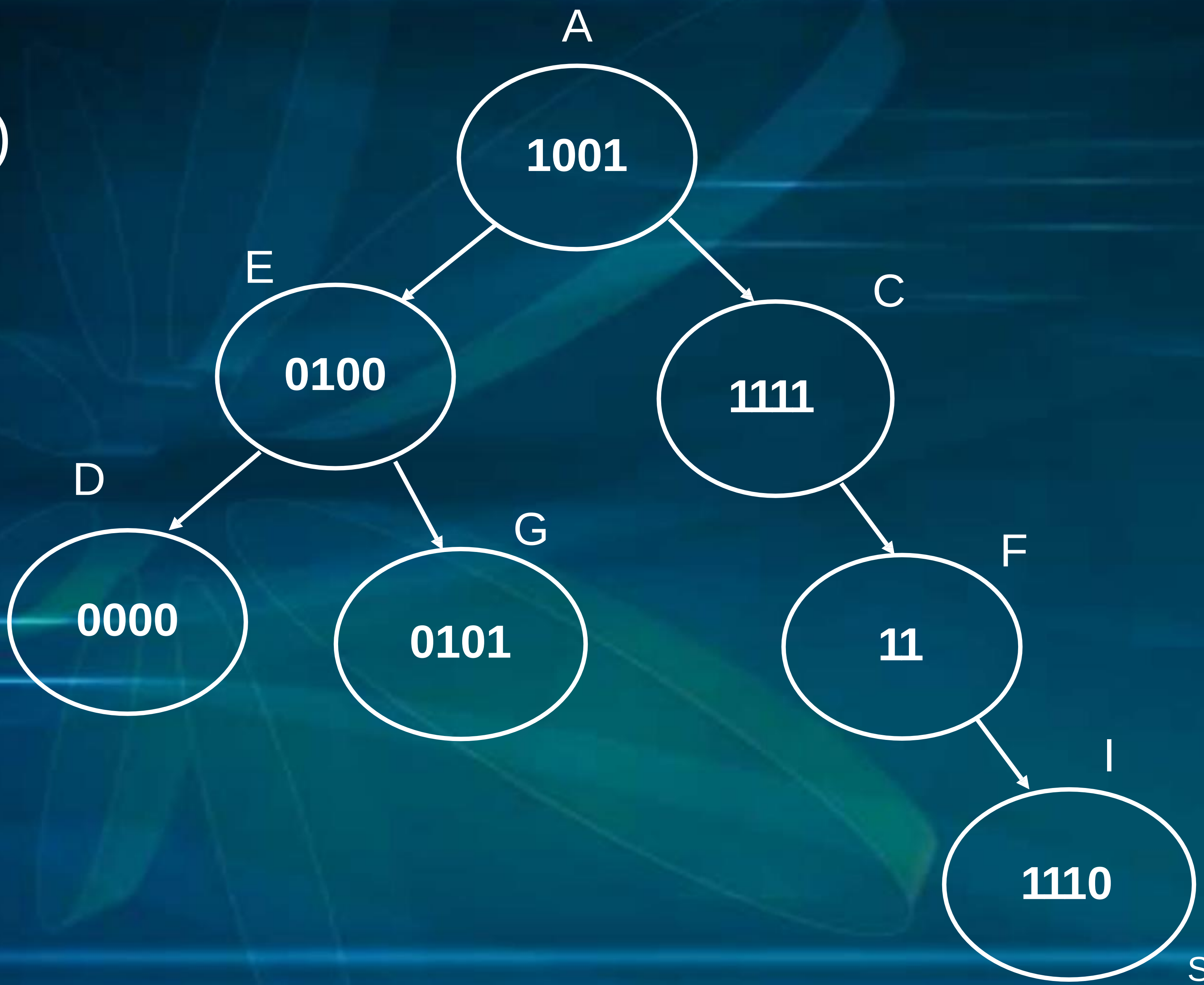


Example

Delete B(0110)

OR

Replace B with E



Delete Pseudo code of DST

```
1.Search key
2. if(key==Node)
     free(Node);
      if(Node->left==NULL && Node->right==NULL
           Do Nothing;
       else if(Node->left!=NULL)
             Replace Node with next left Node
       else if(Node->right!=NULL)
             Replace Node with next right Node
       else if(Node->right!=NULL && Node->left!=NULL )
             Replace Node with next any Node
```

Conclusion

- The digital search trees can be recommended for use whenever the keys stored are binary data.
- Character strings of fixed width.
- DSTs are also suitable in many cases when the keys are strings of variable length.