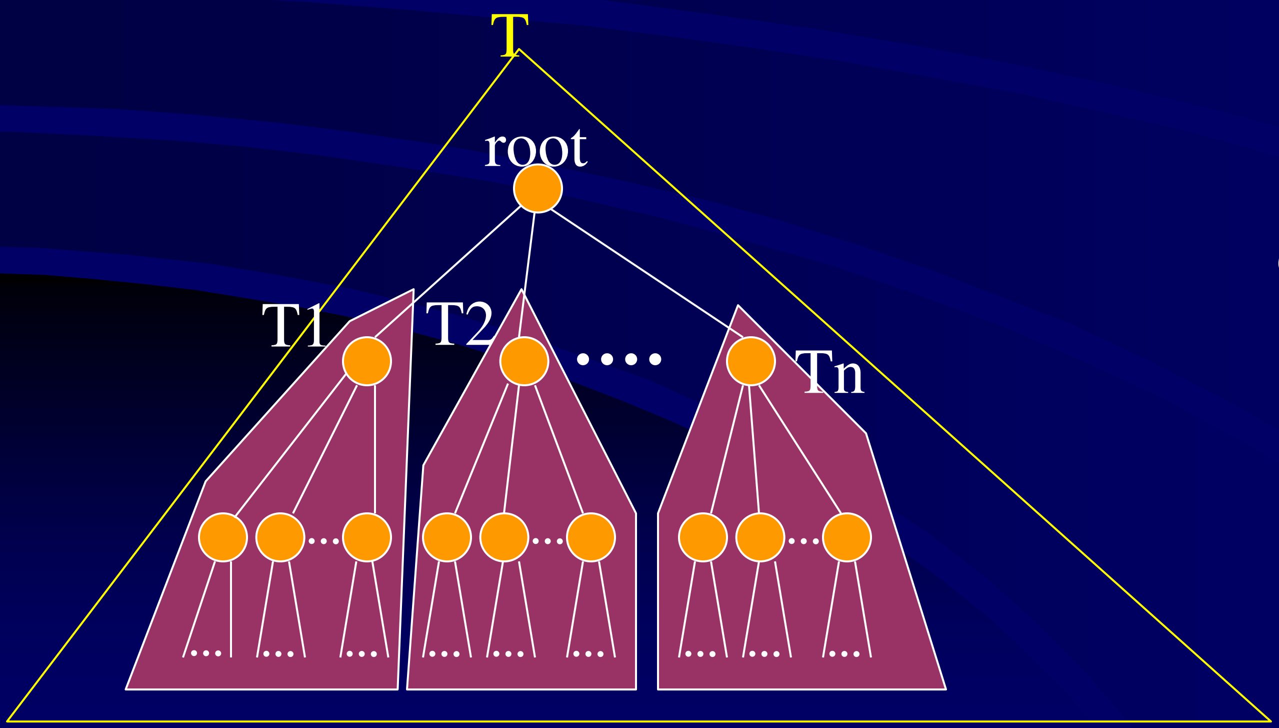
Tree

تعريف

- درخت يك ساختمان داده غيرخطي (Non linear) است
- تعريف:مجموعه اي از يك يا چند گره (node) است كه:
  - گره خاصي به نام ريشه (root) وجود دارد
  - بقيه گره ها به $n \geq 0$ مجموعه مجزاي $T_1, T_2, \ldots, T_n$ تقسيم مي شوند كه هركدام يك درخت هستند. $T_1, T_2, \ldots, T_n$ زيردرختان ريشه ناميده مي شوند



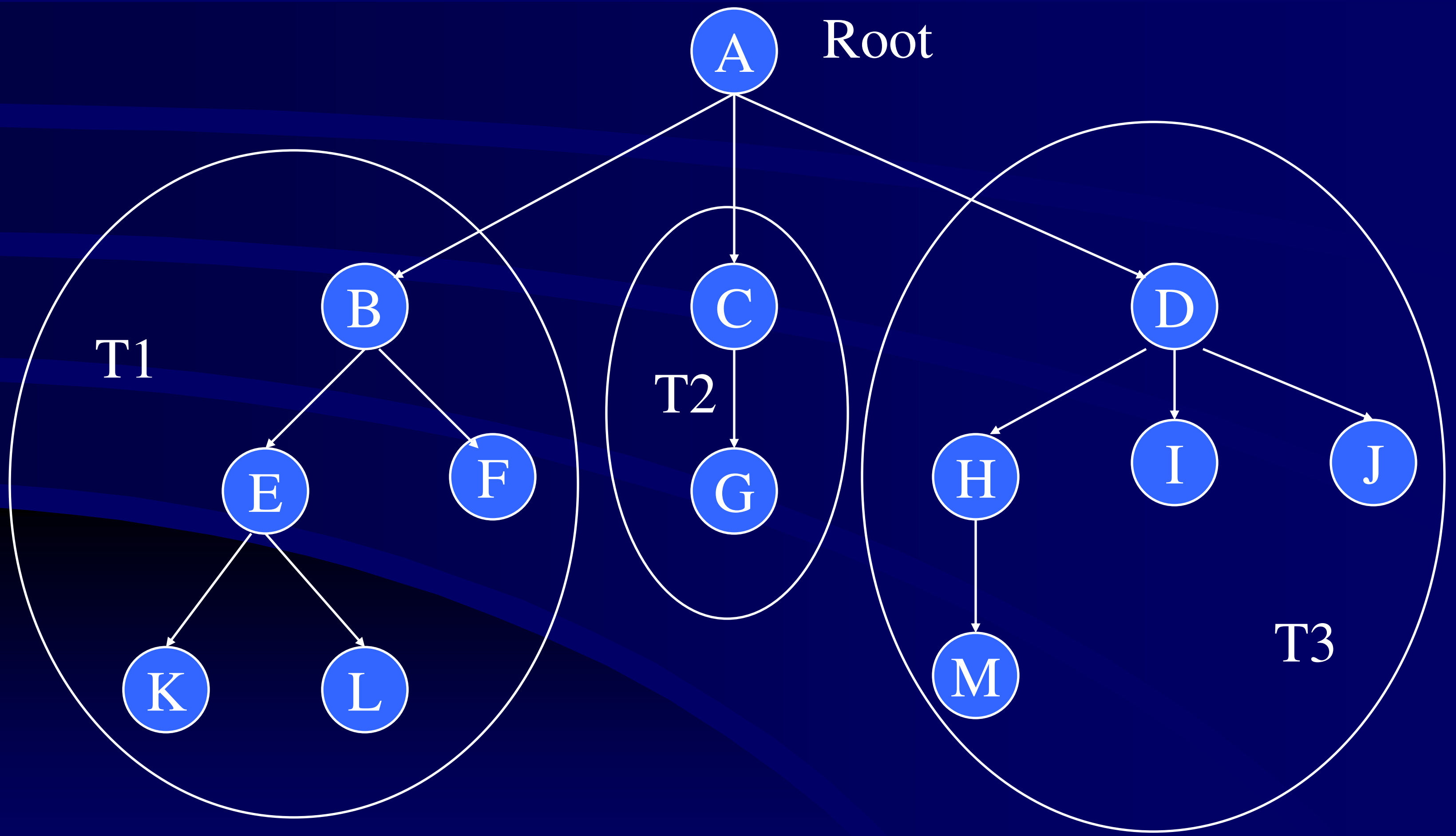
مثال:

-ساختار دايركتوري

-شجره نامه

- ساختار سازماني

مثال



چند اصطلاح

- درجه گره:
- درجه درخت:
- برگ (leaf) یا گره پایاني (terminal node):
- همزاد (Sibling):
- سطح (level) يك گره:

- ارتفاع (height) یا عمق (depth):
- جد (ancestor):

چند اصطلاح

- درجه گره: تعداد زيردرختهاي هر گره
- درجه درخت: حداكثر درجه گره هاي درخت
- برگ (leaf) يا گره پاياني (terminal node): گره با درجه صفر
- همزاد (Sibling): فرزندان يك گره، همزاد (يا برادر) هم هستند
- سطح (level) يك گره: ريشه در سطح صفر قرار دارد و سطح هر گره ديگر، سطح گره پدر به اضافه يك است
- ارتفاع (height) يا عمق (depth): بيشترين سطح گره هاي يك درخت
- جد (ancestor): جد يك گره، تمام گره ها در مسير از ريشه تا گره است

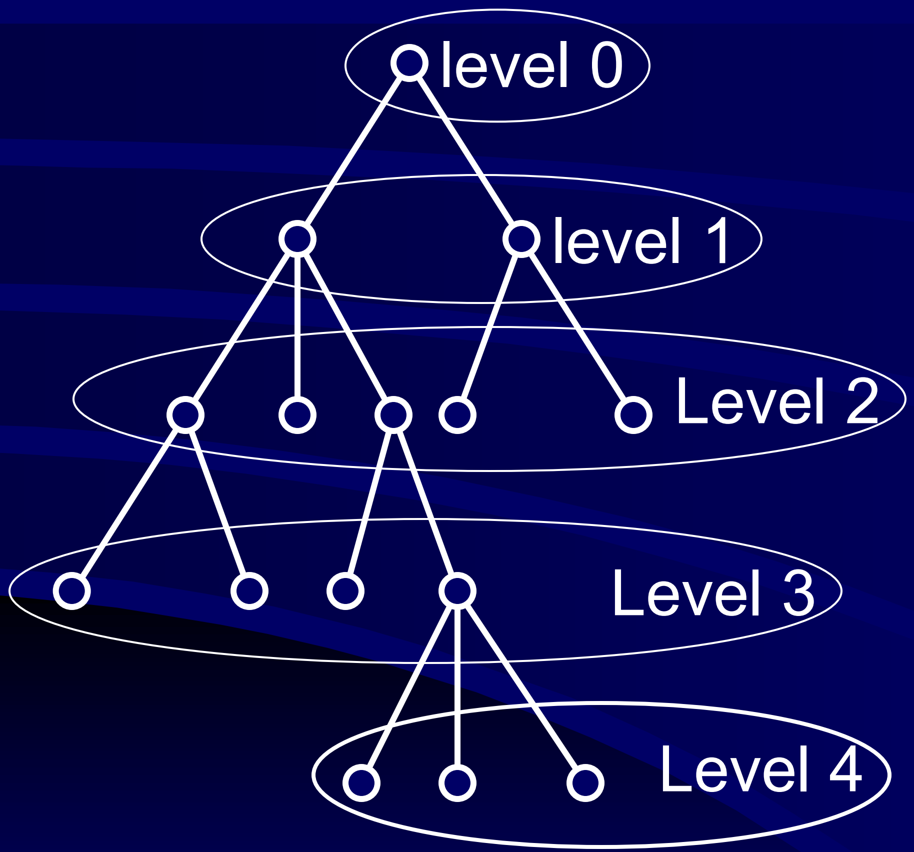
چند اصطلاح (ادامه)

- درخت مرتب (ordered tree):
- درخت نامرتب(unordered tree):
- درخت kتايي(k-ary tree):

- درخت kتايي كامل(complete k-ary tree):
- درخت متوازن (balanced tree):
- درخت كاملاً متوازن (completely balanced tree):
- جنگل (forest):

چند اصطلاح (ادامه)

- درخت مرتب (ordered tree): ترتيب فرزندان هر گره مهم است
- درخت نامرتب(unordered tree): ترتيب فرزندان هر گره مهم نيست
- درخت kتايي(k-ary tree): اگر حداكثر فرزندان هر گره k باشد، آن درخت، درخت kتايي است
- درخت kتايي كامل(complete k-ary tree): تعداد فرزندان هرگره بجز برگها، k است
- درخت متوازن (balanced tree): سطح برگهاي درخت، حداكثر يك واحد اختلاف داشته باشد
- درخت كاملاً متوازن (completely balanced tree): درختي كه سطح برگهاي آن دقيقاً مساوي باشند
- جنگل (forest): مجموعه اي از $n \geq 0$ درخت مجزا



depth=height=4

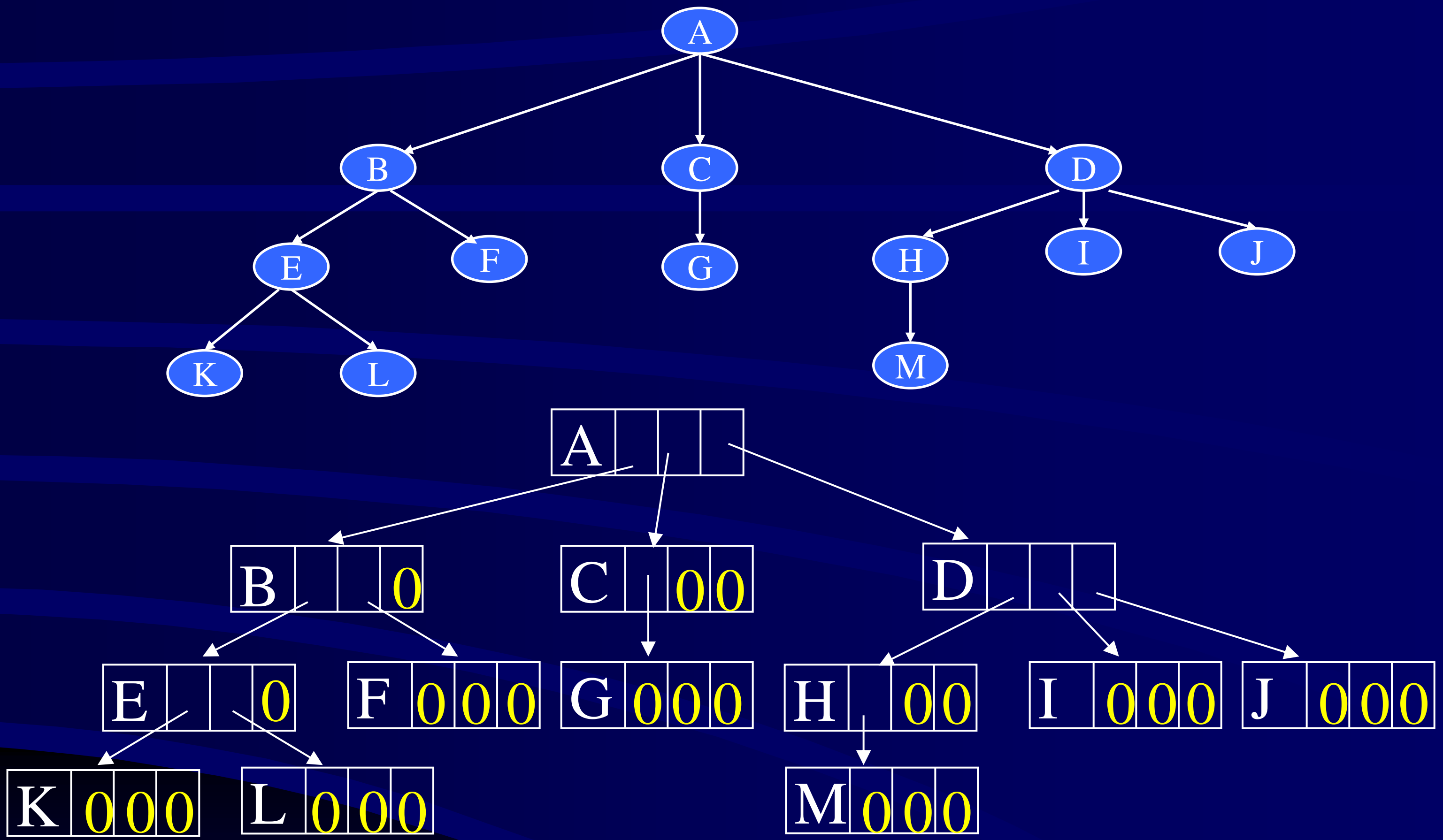
- بين هر دو گره درخت يك مسير يگانه وجود دارد
- اگر E تعداد لبه ها و n تعداد گره ها باشد، E=n-1
- مسئله: تعداد برگهاي درخت k تايي كامل با n رأس را بدست آوريد

$$E = n - 1$$

$$E = (n - l) * k$$

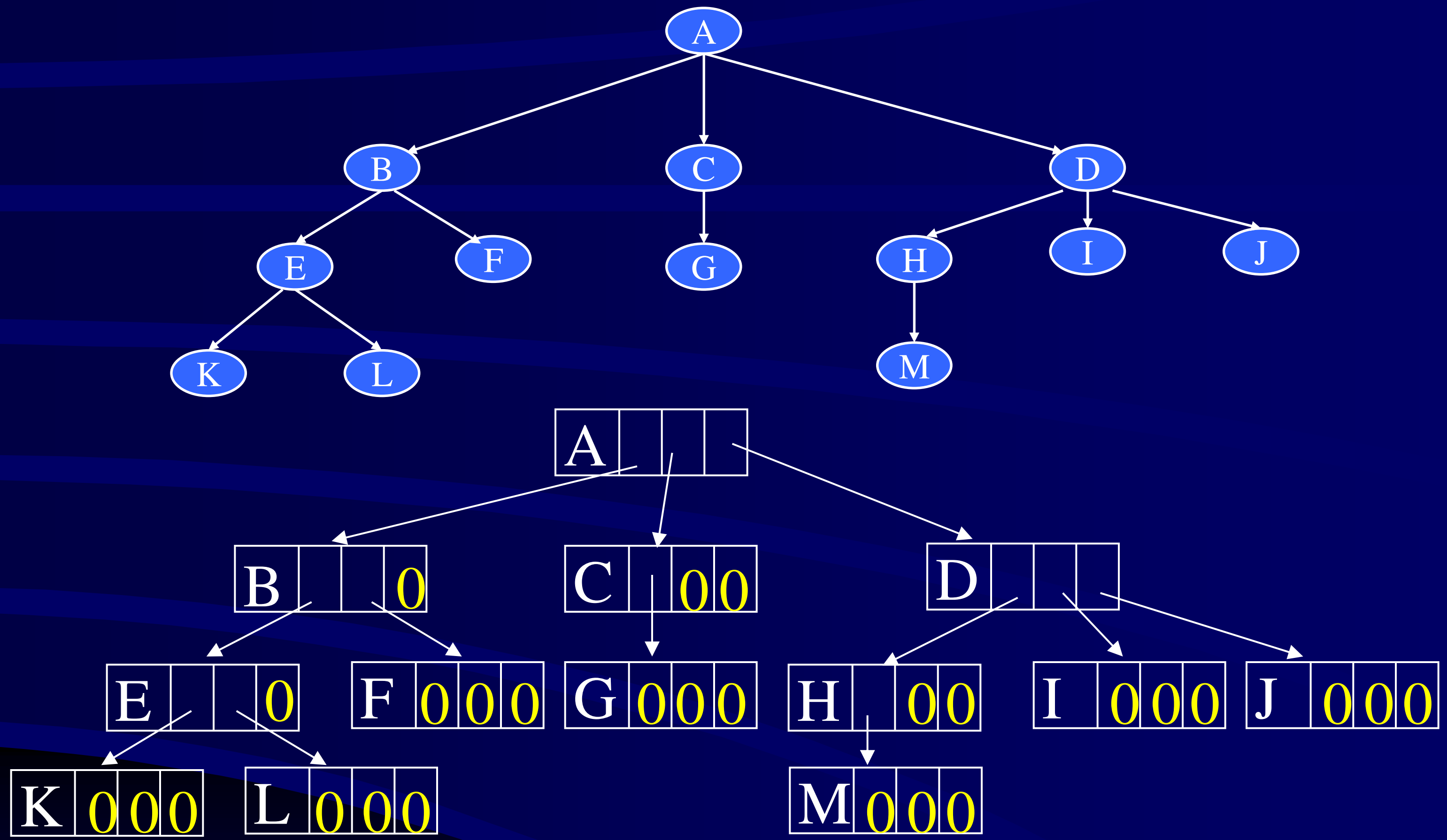
$$\Rightarrow n - 1 = nk - lk \Rightarrow l = \frac{nk - n + 1}{k} = \frac{n(k-1) + 1}{k}$$

استفاده از k اشاره گر براي درخت k تايي



تعداد اشاره گرهاي خالي بر اساس n و k :

استفاده از k اشاره گر براي درخت k تايي



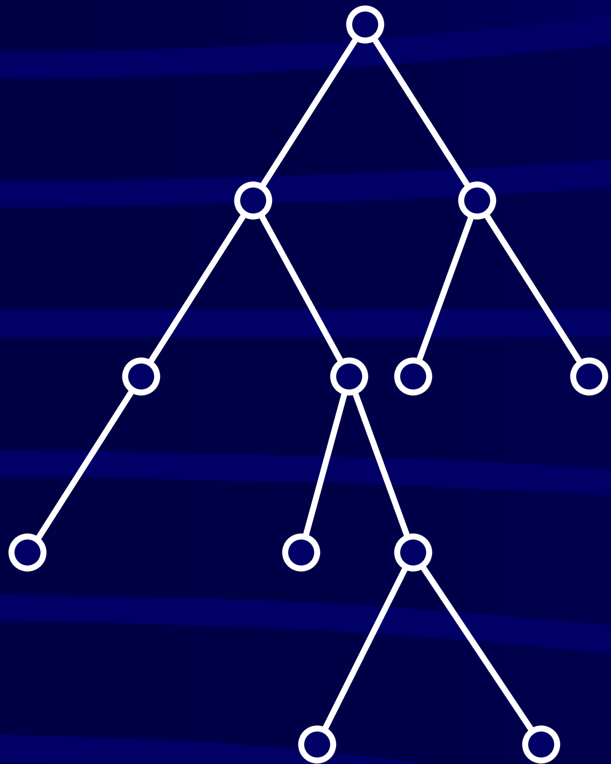
تعداد اشاره گرهاي خالي:

$$E = n - 1$$

$$nk - E = nk - (n-1) = n(k-1) + 1$$

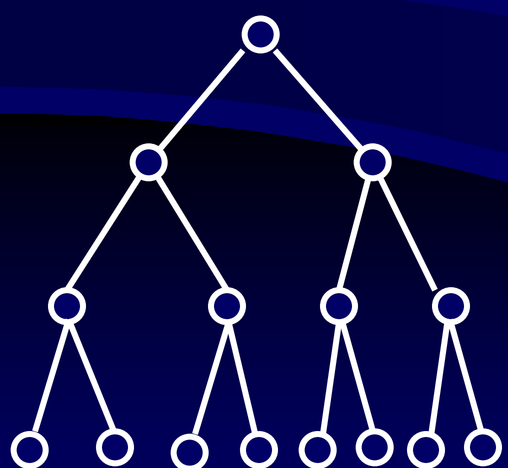
تعداد کل اشاره گرها در تمامي گره ها

درخت دودويي



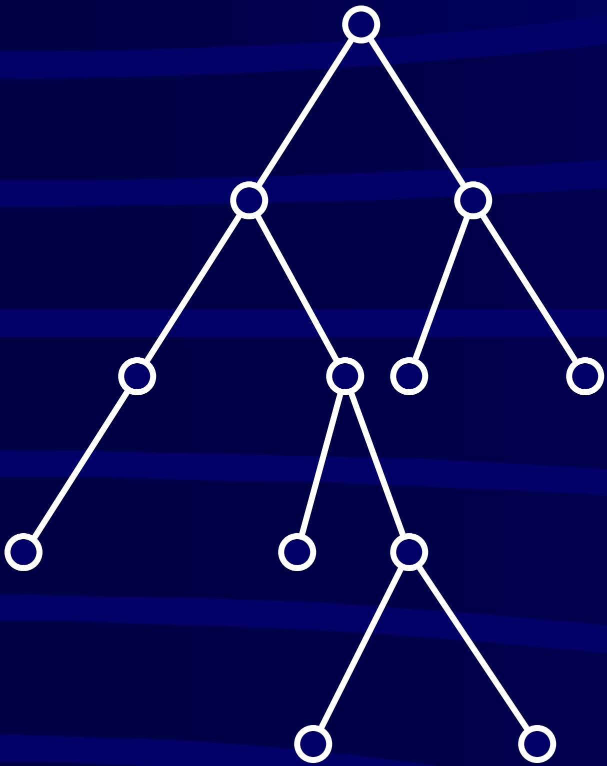
تعريف: مجموعه محدودي از گره ها كه يا تهي است و يا شامل يك ريشه و دو زيردرخت دودويي مجزا (زيردرخت چپ و زيردرخت راست) است.

هر گره حداكثر دو فرزند دارد



- تعداد گره هاي سطح i حداكثر 2^i است
- تعداد گره هاي درخت كامل با عمق k:

درخت دودويي

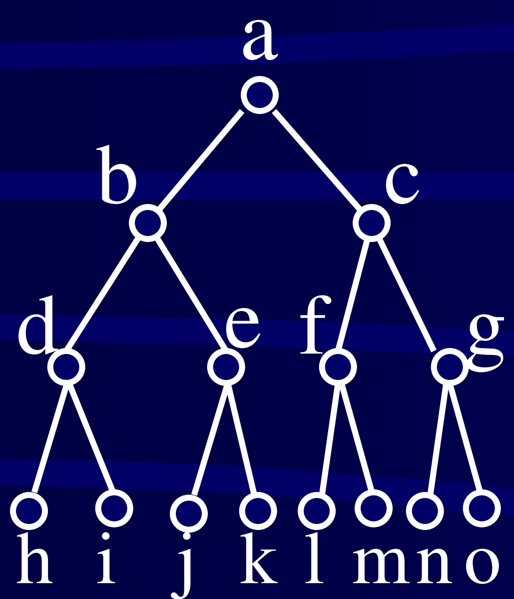


تعريف: مجموعه محدودي از گره ها كه يا تهي است و يا شامل يك ريشه و دو زيردرخت دودويي مجزا (زيردرخت چپ و زيردرخت راست) است.

هر گره حداكثر دو فرزند دارد

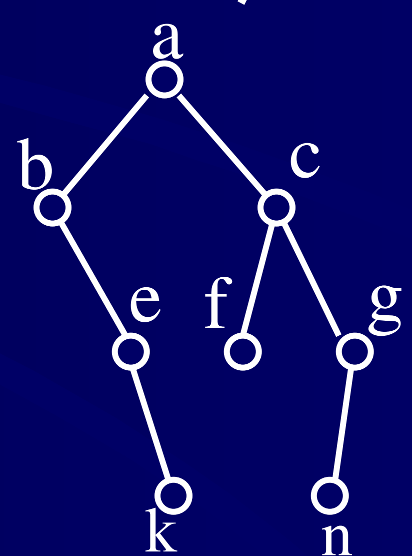
- تعداد گره هاي سطح i حداكثر 2^i است
- تعداد گره هاي درخت كامل با عمق k: $2^{k+1}-1$

پياده سازي درخت دودويي با استفاده از آرايه



- ريشه درخت در انديس شماره 1
- فرزند چپ i در $i*2$ قرار دارد
- فرزند راست i در $i*2+1$ قرار دارد
- براي $i>1$، پدر i در $\lfloor i/2 \rfloor$ قرار دارد

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o



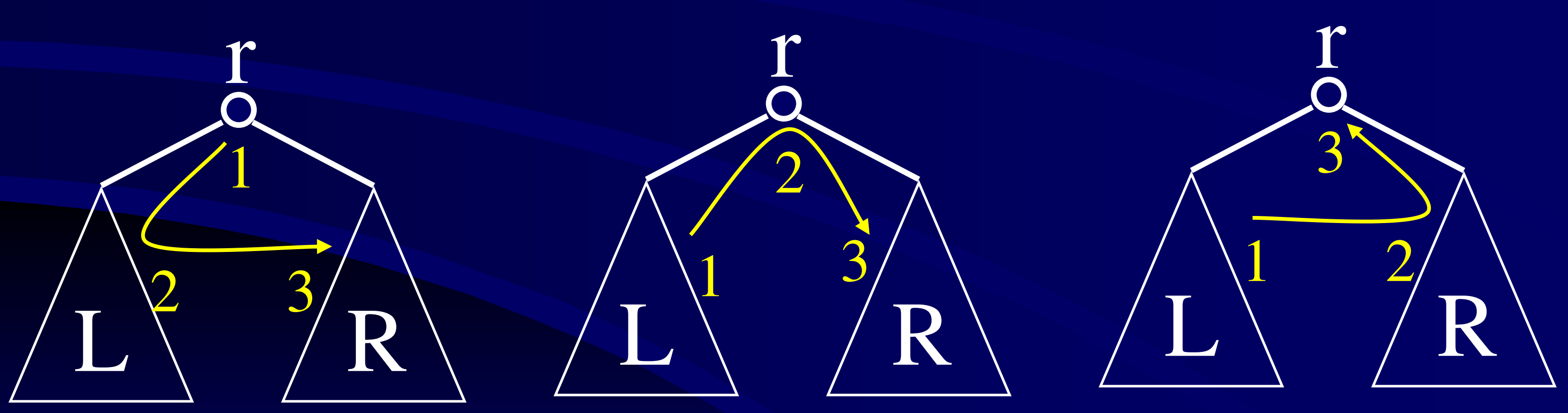
این روش براي نمايش درخت کامل بهينه است

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a	b	c		e	f	g				k			n	

پيمايش درخت دودويي

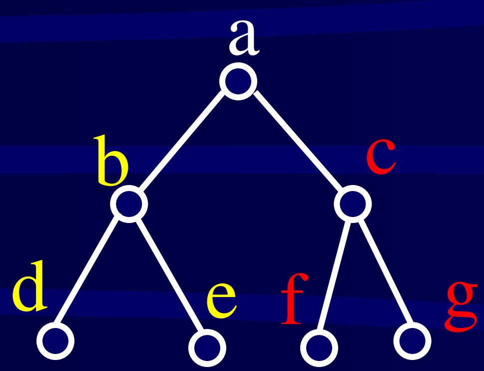
- طي كردن هر گره درخت يك و فقط يك بار
- 6 تركيب مختلف كنار هم قرار دادن r(root),L(Left),R(Right):
(rLR,rRL,LrR,LRr,RrL,RLr)

اگر زير درخت چپ قبل از زيردرخت راست پيمايش شود،
3 حالت مختلف ايجاد مي شود:



rLR preorder | LrR inorder | LRr postorder

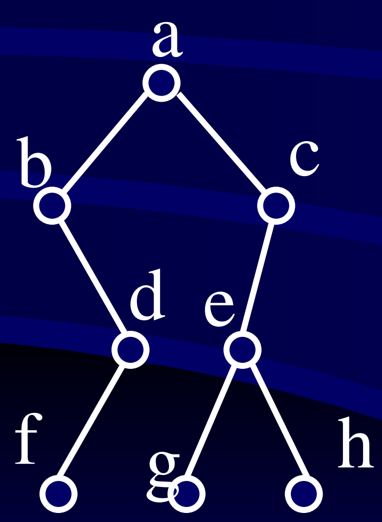
مثال



Inorder: d b e a f c g

Preorder: a b d e c f g

Postorder: d e b f g c a



Inorder: b f d a g e h c

Preorder: a b d f c e g h

Postorder: f d b g h e c a

توابع پیمایش درخت

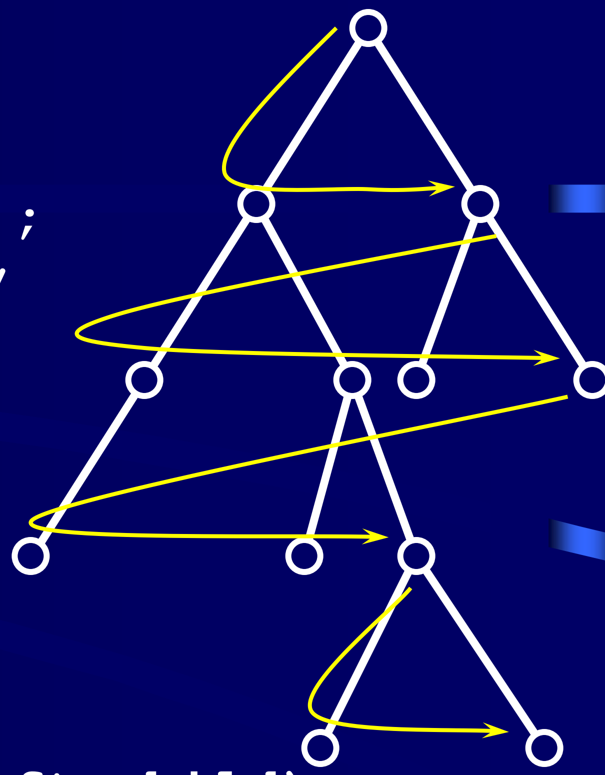
```
void inorder (tree_pointer ptr)
{
   if (ptr) {
      inorder(ptr->left_child);
      printf("%d",ptr->data);
      inorder(ptr->right_child);
   }
}

void preorder (tree_pointer ptr)
{
   if (ptr) {
      printf("%d",ptr->data);
      preorder(ptr->left_child);
      preorder(ptr->right_child);
   }
}
```

پيمايش غير بازگشتي inorder

```
void iter_inorder(tree_pointer node)
{
   int top= -1; /* initialize stack */
   tree_pointer stack[MAX_STACK_SIZE];
   for (;;) {
    for (; node; node=node->left_child)
       add(&top, node);/* add to stack */
    node= delete(&top);
                  /* delete from stack */
    if (!node) break; /* empty stack */
    printf("%d", node->data);
    node = node->right_child;
  }
}
```

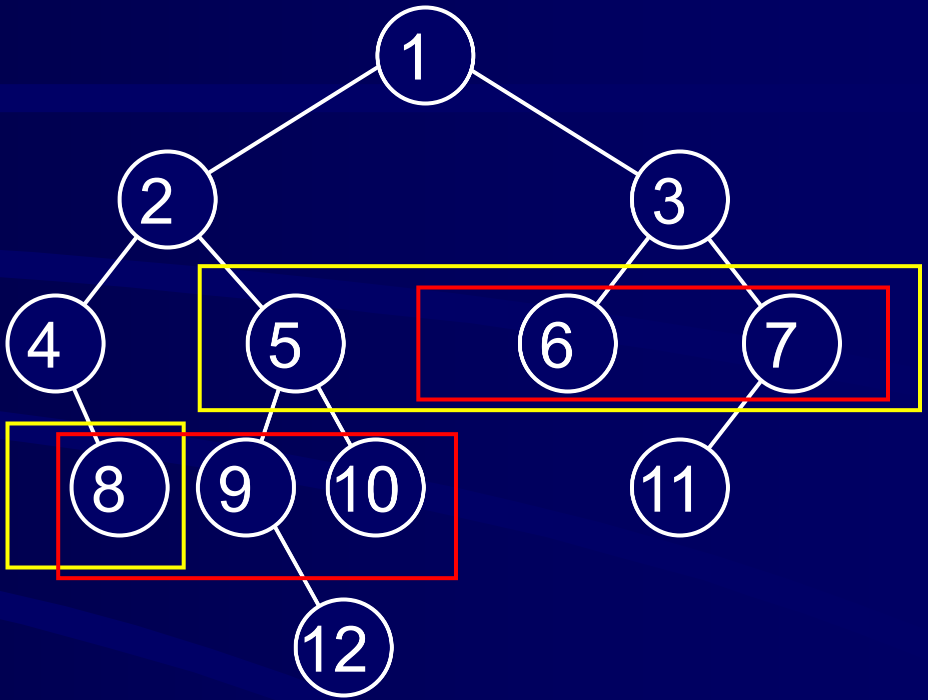
پيمايش سطحي

```
void level_order(tree_pointer ptr)
/* level order tree traversal */
{
  int front = rear = 0;
  tree_pointer queue[MAX_QUEUE_SIZE];
  if (!ptr) return; /* empty tree */
  addq(front, &rear, ptr);
  for (;;) {
    ptr = deleteq(&front, rear);
    if (ptr) {
       printf("%d", ptr->data);
       if (ptr->left_child)
          addq(front, &rear, ptr->left_child);
       if (ptr->right_child)
          addq(front, &rear, ptr-right_child);
    }
    else break;
  }
}
```



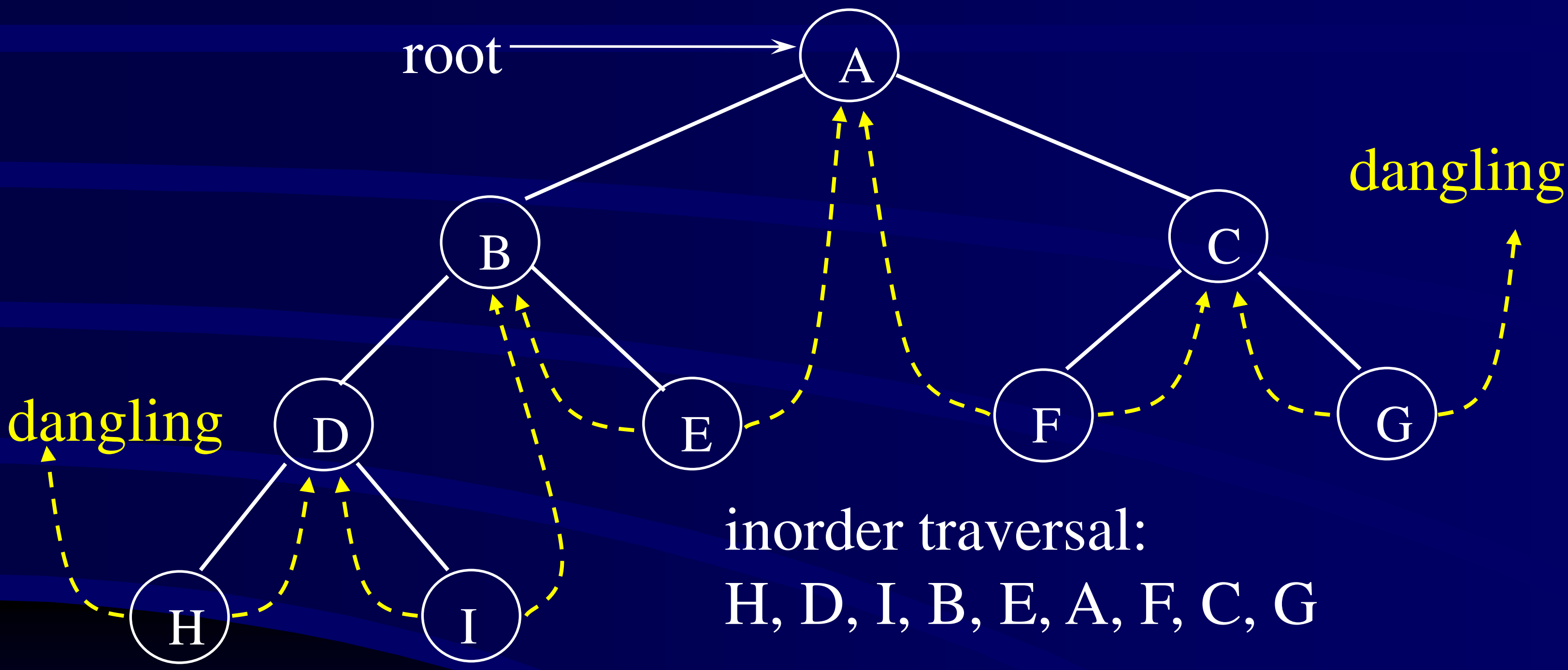
مثال پيمايش سطحي

queue:
1
2 3
3 **4 5**
4 5 **6 7**
5 6 7 **8**
6 7 8 **9 10**
7 8 9 10
8 9 10 **11**
9 10 11
10 11 **12**
11 12
12



محتواي صف پس از ديدن گره 4
محتواي صف پس از ديدن گره 5

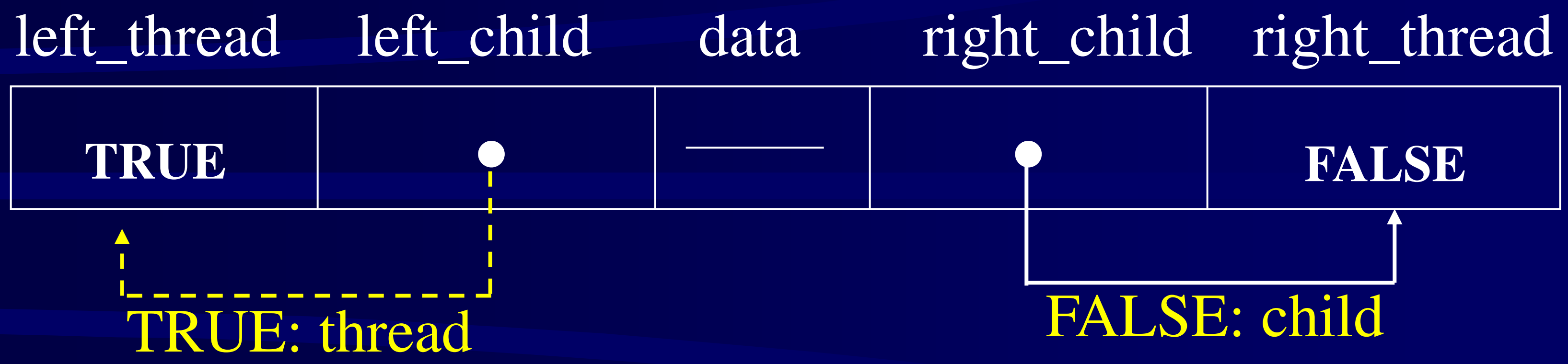
درخت دودويي نخ كشي شده (Threaded) (تسبيح واره)



inorder traversal:
H, D, I, B, E, A, F, C, G

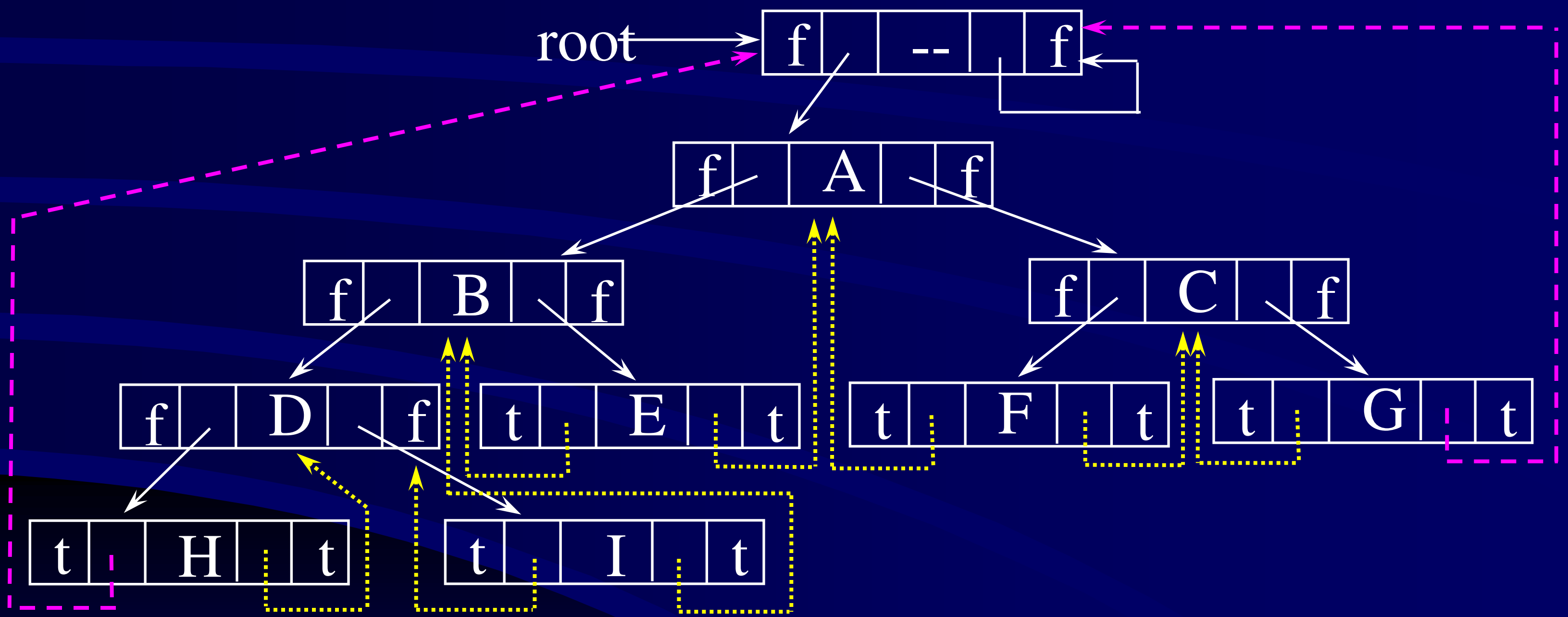
از اشاره گرهاي خالي براي اشاره به عنصر قبل و عنصر بعد در ترتيب inorder استفاده مي شود

ساختمان داده براي گره هاي درخت نخ کشي شده

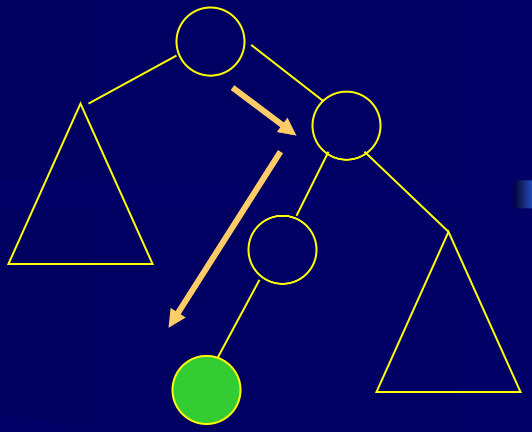



```
typedef struct threaded_tree *threaded_pointer;
typedef struct threaded_tree {
    short int left_thread;
    threaded_pointer left_child;
    char data;
    threaded_pointer right_child;
    short int right_thread;
};
```

درخت نخ کشي شده: مثال

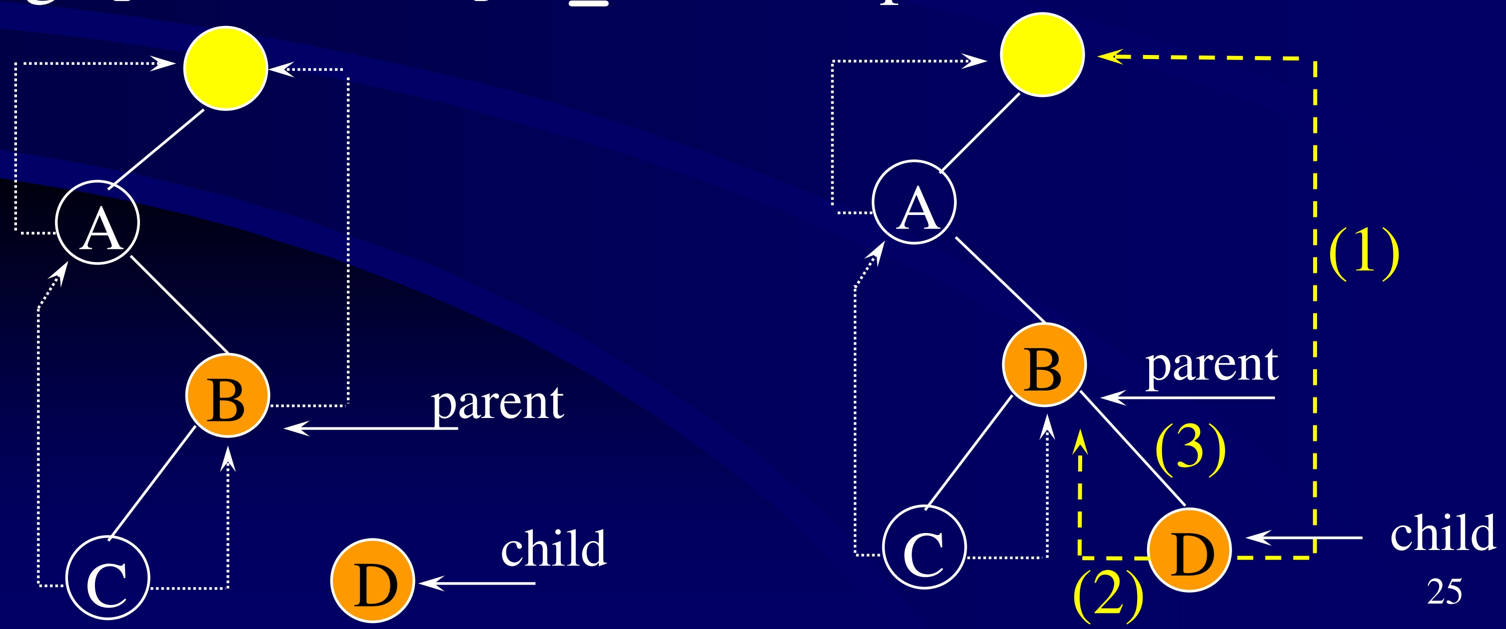


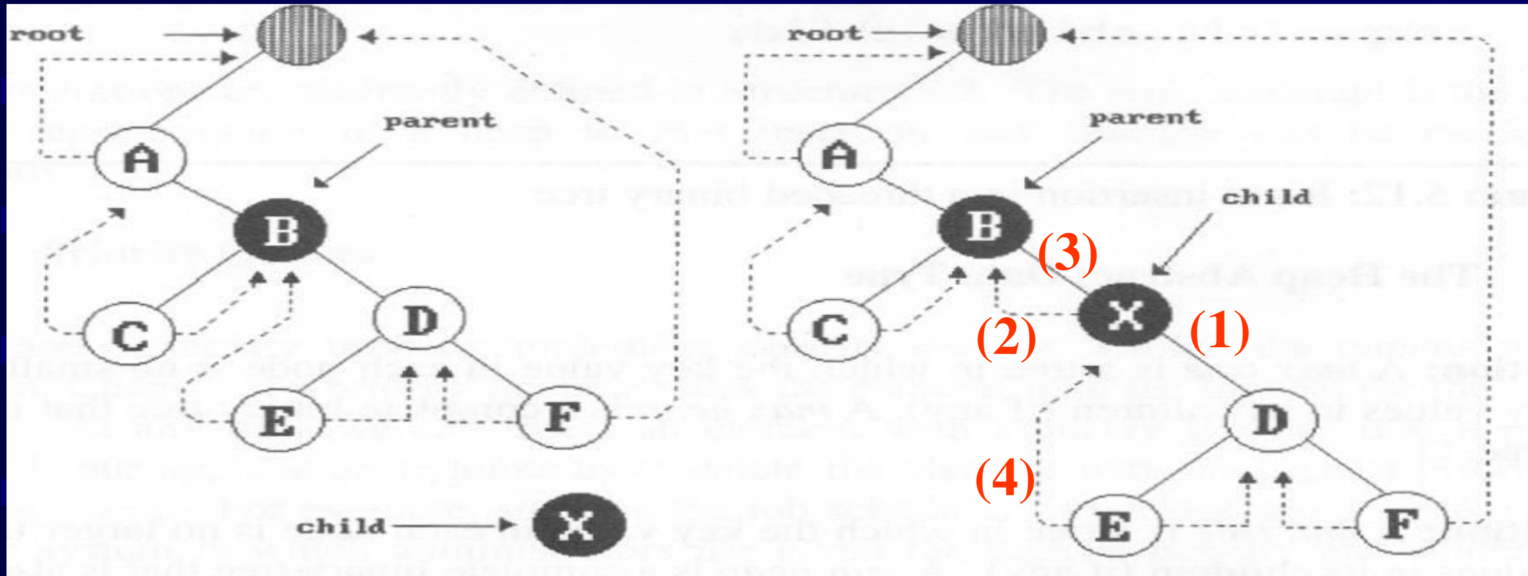
```
thread_ptr insucc(thread_ptr tree)
{ thread_ptr temp;
  temp = tree->right_child;
  if (!tree->right_thread)
    while (!temp->left_thread)
      temp = temp->left_child;
  return temp;
}
void tinorder(thread_ptr tree)/*traverse inorder*/
{   threaded_pointer temp = tree;
    for (;;) {
        temp = insucc(temp);
        if (temp==tree) break;
        printf("%3c", temp->data);
    }
}
```



اضافه كردن گره به درخت نخ كشي شده

- Insert **`child`** as the right child of node **`parent`**
  - change **`parent->right_thread`** to FALSE
  - set **`child->left_thread`** and **`child->right_thread`** to TRUE
  - set **`child->right_child`** to **`parent->right_child`**
  - set **`child->left_child`** to point to **`parent`**
  - change **`parent->right_child`** to point to **`child`**






```
void insert_right(thrd_ptr parent, thrd_ptr child)
{ threaded_pointer temp;
(1)  child->right_child = parent->right_child;
     child->right_thread = parent->right_thread;
(2)  child->left_child = parent;
     child->left_thread = TRUE;
(3)  parent->right_child = child;
     parent->right_thread = FALSE;
  if (!child->right_thread){/*parent had a child*/
(4)    temp = insucc(child);
       temp->left_child = child;
  }
}
```

كد هافمن (Huffman Code)

- مسئله: بدست آوردن كد بهينه براي كاراكترهاي يك متن با داشتن احتمال وقوع هر كاركتر (بدست آوردن كدي كه طول متوسط آن كمترين باشد)

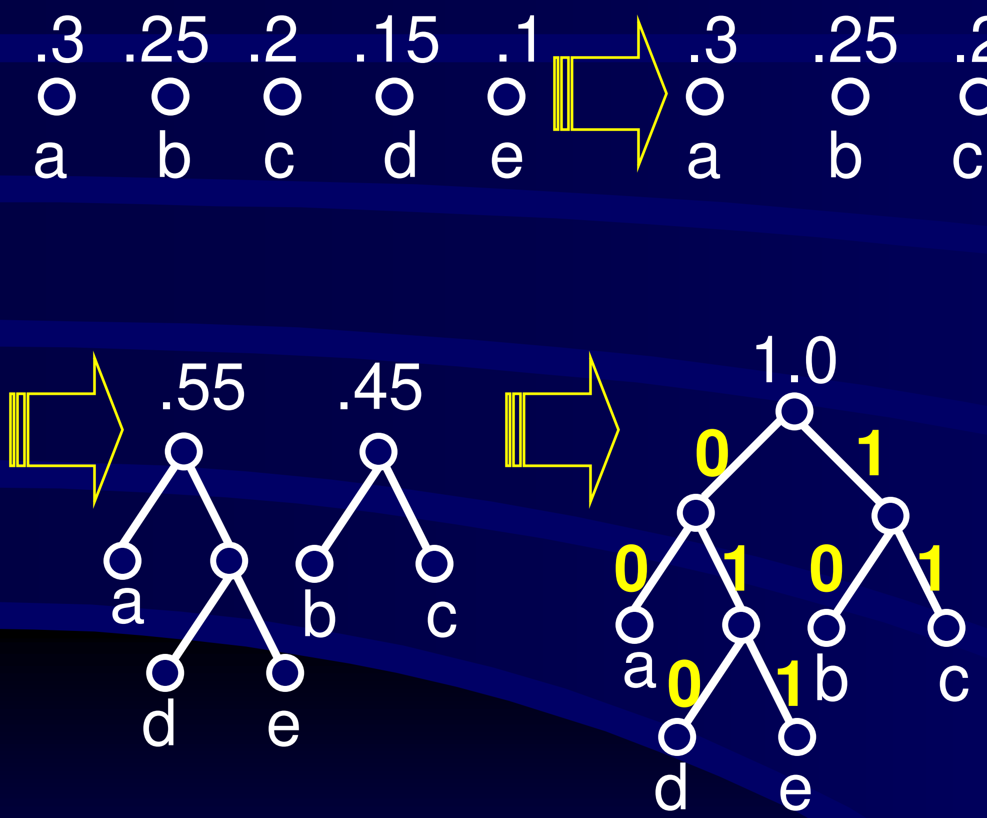
احتمال وقوع كاراكترهاي مختلف: تعداد كل كاراكترها/تعداد وقوع كاركتر x = P(x)

- روشهاي كد كردن كاراكترها:
  - با طول ثابت (مثل ASCII)
  - با طول متغير

- در كد كردن با طول متغير، كدها نبايد **خاصيت پيشوندي** داشته باشند.
  خاصيت پيشوندي (Prefix Property) : يك كد، زيررشته اي از كد ديگر باشد.
  مثال: 011 , 01

مثال درخت هافمن

a	b	c	d	e
.3	.25	.2	.15	.1



a: 00 **.3 * 2 = .6**

b: 10 **.25* 2 = .5**

c: 11 **.2 * 2 = .4**

d: 010 **.15* 3 = .45**

e: 011 **.1 * 3 = .3**

Avg length = 2.25 bit

1011010000011

b c d a e

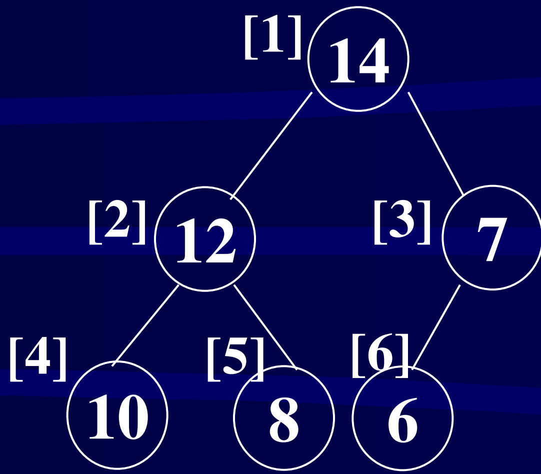
Heap

تعريف: max tree (min tree) درختي است كه مقدار كليد هر گره از فرزندانش (در صورت وجود) كمتر (بيشتر) نباشد.

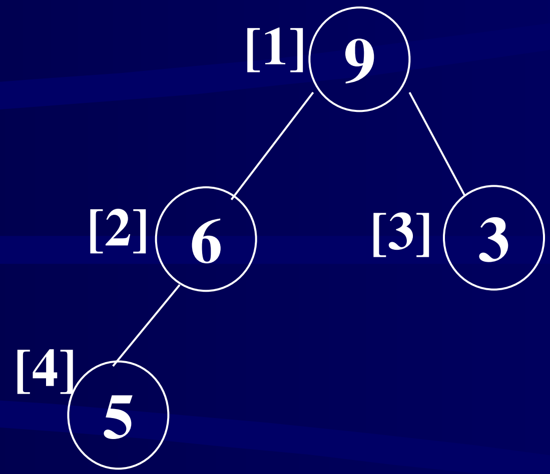
تعريف: max heap (min heap) درخت دودويي كامل است كه max tree (min tree) نيز باشد. (بجز حداكثر يک نود با يک فرزند) و برگها از سمت چپ چيده شده اند.

heap ساختمان داده مناسبي براي پياده سازي صف با اولويت (priority queue) است.

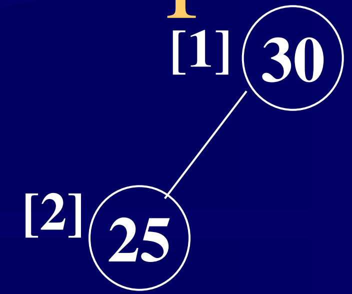
Heap مثال



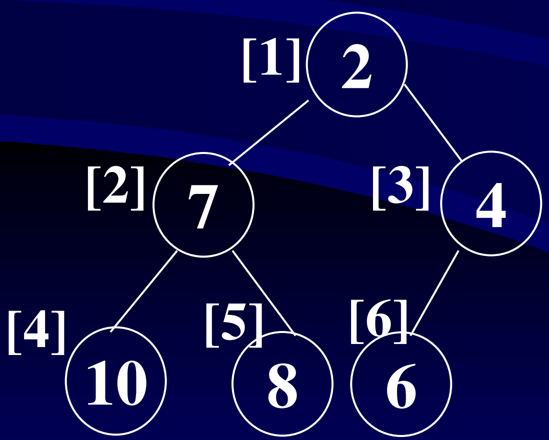
14	12	7	10	8	6
1	2	3	4	5	6



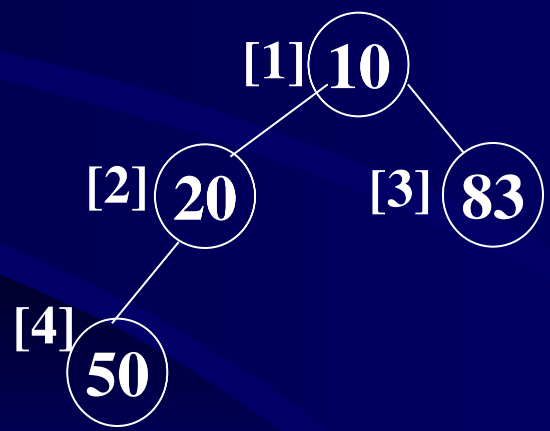
9	6	3	5
1	2	3	4



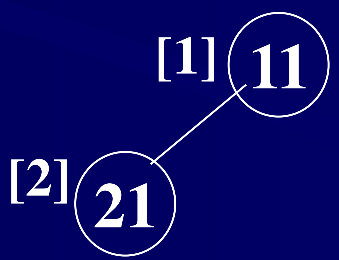
30	25
1	2



2	7	4	10	8	6
1	2	3	4	5	6

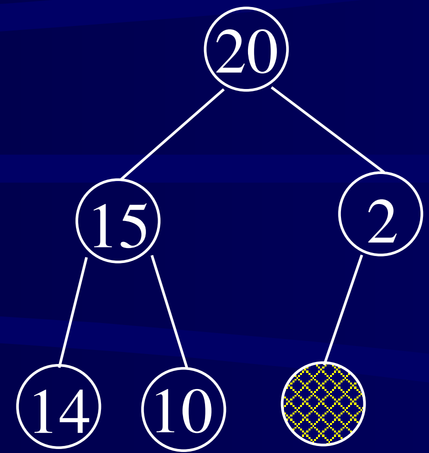


10	20	83	50
1	2	3	4



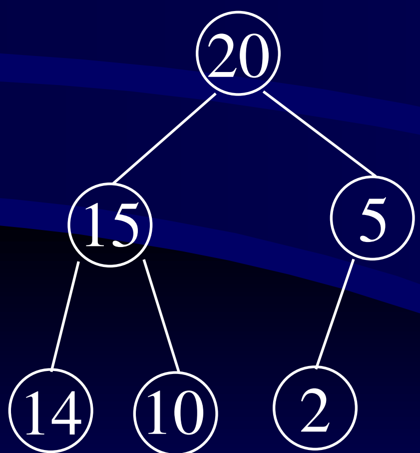
11	21
1	2

عمليات روي heap : insert

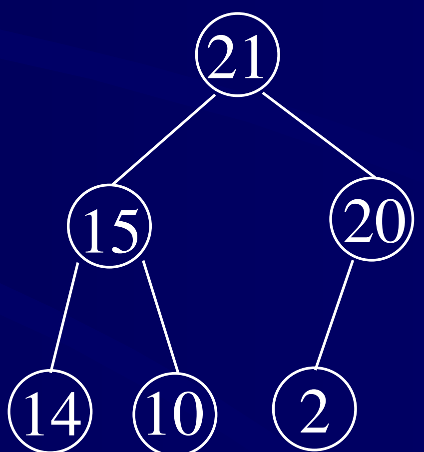


20	15	2	14	10	
1	2	3	4	5	6

محل اضافه شدن گره جديد



insert 5 into heap



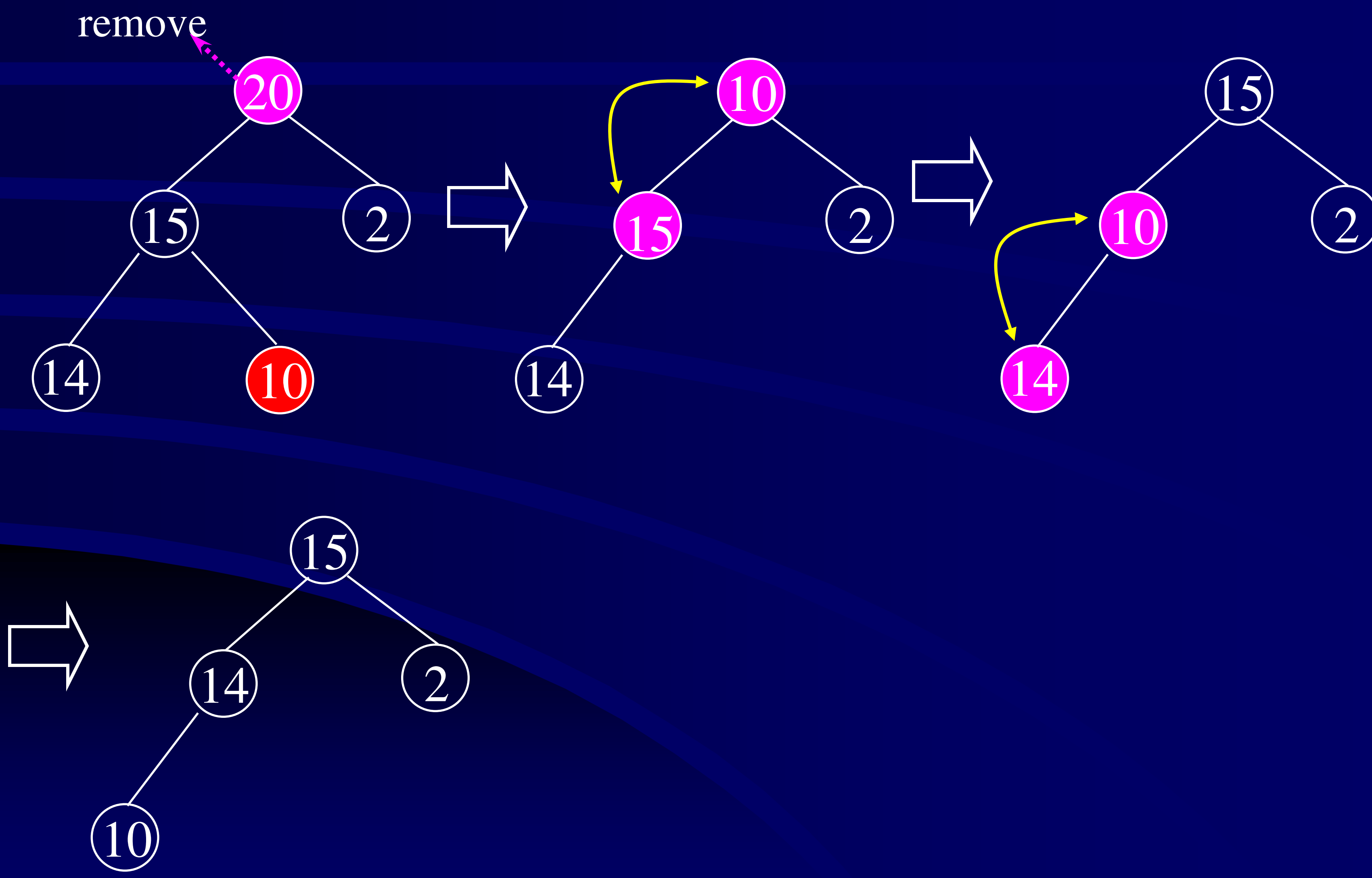
insert 21 into heap

الگوریتم insert به heap

```
void insert_max_heap(element item, int *n)
{
  i = ++(*n);
  while ((i!=1)&&(item.key>heap[i/2].key)) {
    heap[i] = heap[i/2];
    i /= 2;
  }
  heap[i]= item;
}
```

$O(\log_2 n)$

delete : heap عمليات روي



الگوریتم حذف از heap

```
element delete_max_heap(int *n)
{
  int parent, child;
  element item, temp;
  item=heap[1];/*save value of the element with the highest key*/
  temp=heap[(*n)--];/*use last element in heap to adjust heap*/
  parent = 1;
  child = 2;
  while (child <= *n) {
   if((child<*n)&&(heap[child].key<heap[child+1].key))
      child++;
   if (temp.key >= heap[child].key) break;
    /* move to the next lower level */
   heap[parent] = heap[child];
   parent = child;
   child *= 2;
  }
  heap[parent] = temp;
  return item;
}
```

$O(\log_2 n)$

تبدیل یک آرایه به max-heap

تبدیل یک آرایه به max-heap

$\textsc{Build-Heap}(A)$

1 **for** $i \leftarrow \lfloor \frac{length[A]}{2} \rfloor$ **downto** 1

2 **do** $\textsc{Heapify}(A, i)$

به صورت max-heap در آوردن A[i..length[A]]

$\textsc{Max-Heapify}(A, i)$

$l \leftarrow \textsc{LeftChild}\ (i)$

$r \leftarrow \textsc{RightChild}\ (i)$

if $l \leq length[A]$ **and** $A[l] > A[i]$

  then $bigchild \leftarrow l$

  else $bigchild \leftarrow i$

if $r \leq length[A]$ **and** $A[r] > A[bigchild]$

  then $bigchild \leftarrow r$

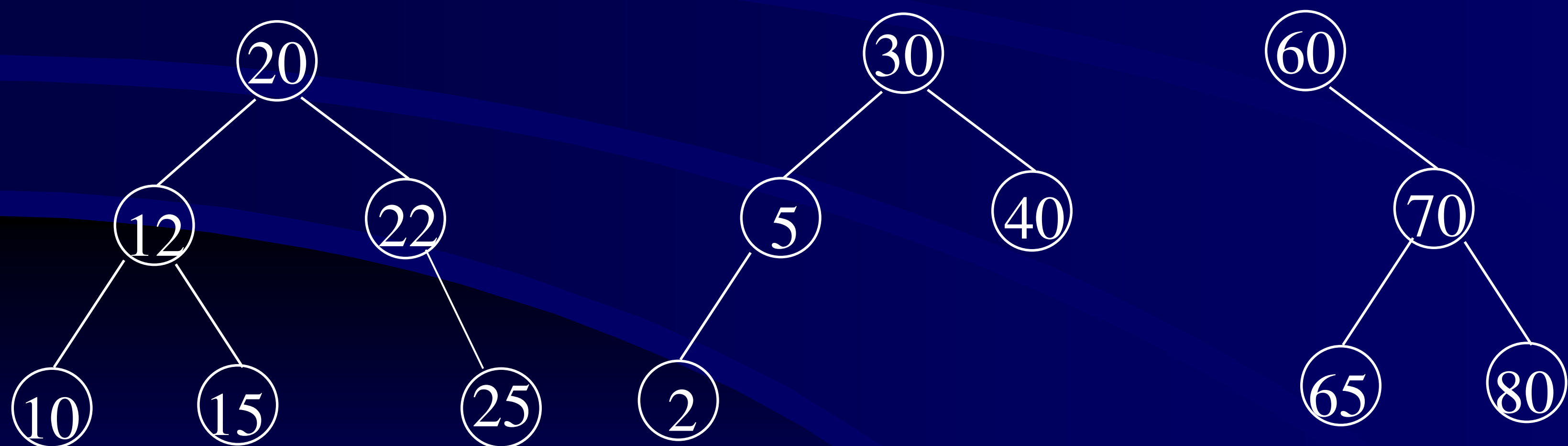
if $bigchild \neq i$

  then swap$(A[i], A[bigchild])$

    $\textsc{Max-Heapify}\ (A, bigchild)$

درخت دودويي جستجو (BST: Binary Search Tree)

- هر عنصر يك كليد يگانه دارد (كليد تكراري وجود ندارد)
- كليدهاي زيردرخت چپ (در صورت وجود) از كليد ريشه كوچكتر است
- كليدهاي زيردرخت راست(در صورت وجود) از كليد ريشه بزرگتر است
- زيردرختهاي چپ و راست، درخت دودويي جستجو هستند



<u>Inorder traverse</u>

10 12 15 20 22 25

<u>Inorder traverse</u>

2 5 30 40

<u>Inorder traverse</u>

60 65 70 85

جستجوي درخت دودويي جستجو(بازگشتي)

```
tree_pointer search(tree_pointer root, int key)
{
/* return a pointer to the node that contains
   key. If there is no such node, return NULL */

   if (!root) return NULL;
   if (key == root->data) return root;
   if (key < root->data)
        return search(root->left_child, key);
   return search(root->right_child,key);
}
```

هزينه الگوريتم: O(h)

(h ارتفاع درخت، در بدترين حالت h=n)

جستجوي درخت دودويي جستجو (غيربازگشتي)

```
tree_pointer search2(tree_pointer tree, int key)
{
  while (tree) {
    if (key == tree->data) return tree;
    if (key < tree->data)
        tree = tree->left_child;
    else
        tree = tree->right_child;
  }
  return NULL;
}
```

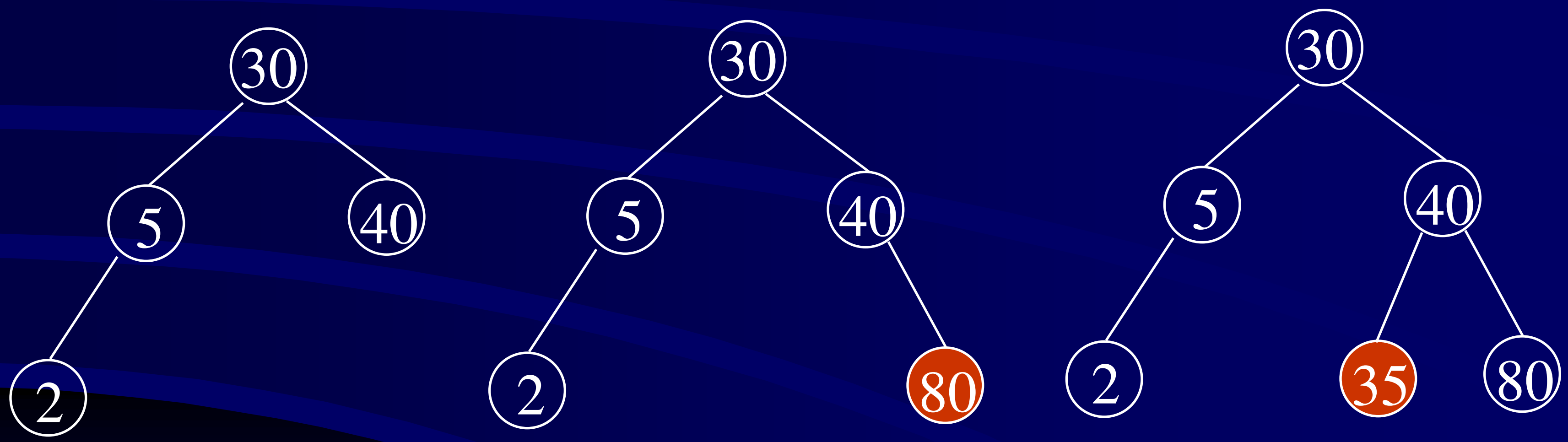
هزينه الگوريتم: O(h)

(h ارتفاع درخت، در بدترين حالت h=n)

درج (Insert) گره در BST

الگوريتم:

- جستجو براي كليد انجام مي شود
- اگر كليد در درخت پيدا نشد، به عنوان فرزند آخرين گره در جستجو به درخت اضافه مي شود. (بطوريكه شرط BST نقض نشود)

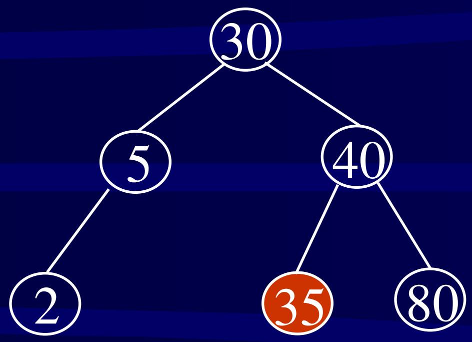


Insert 80　　Insert 35

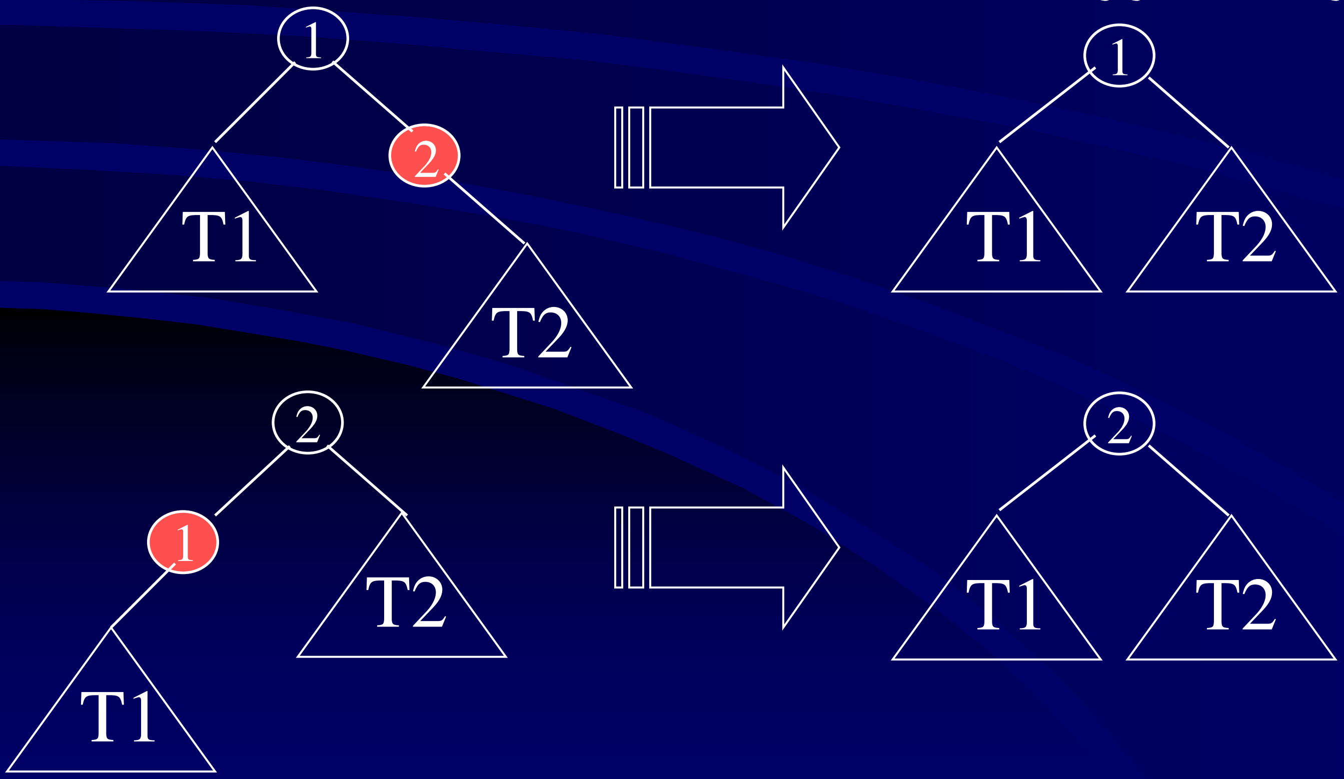
حذف (Delete) گره از BST

الگوريتم:

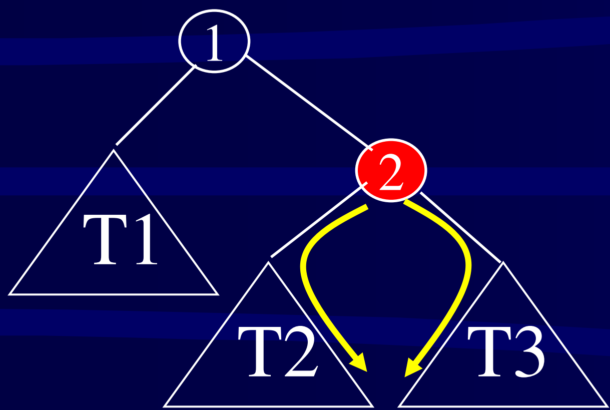
- حالت اول: حذف يك برگ



- حالت دوم: حذف گره با يك فرزند

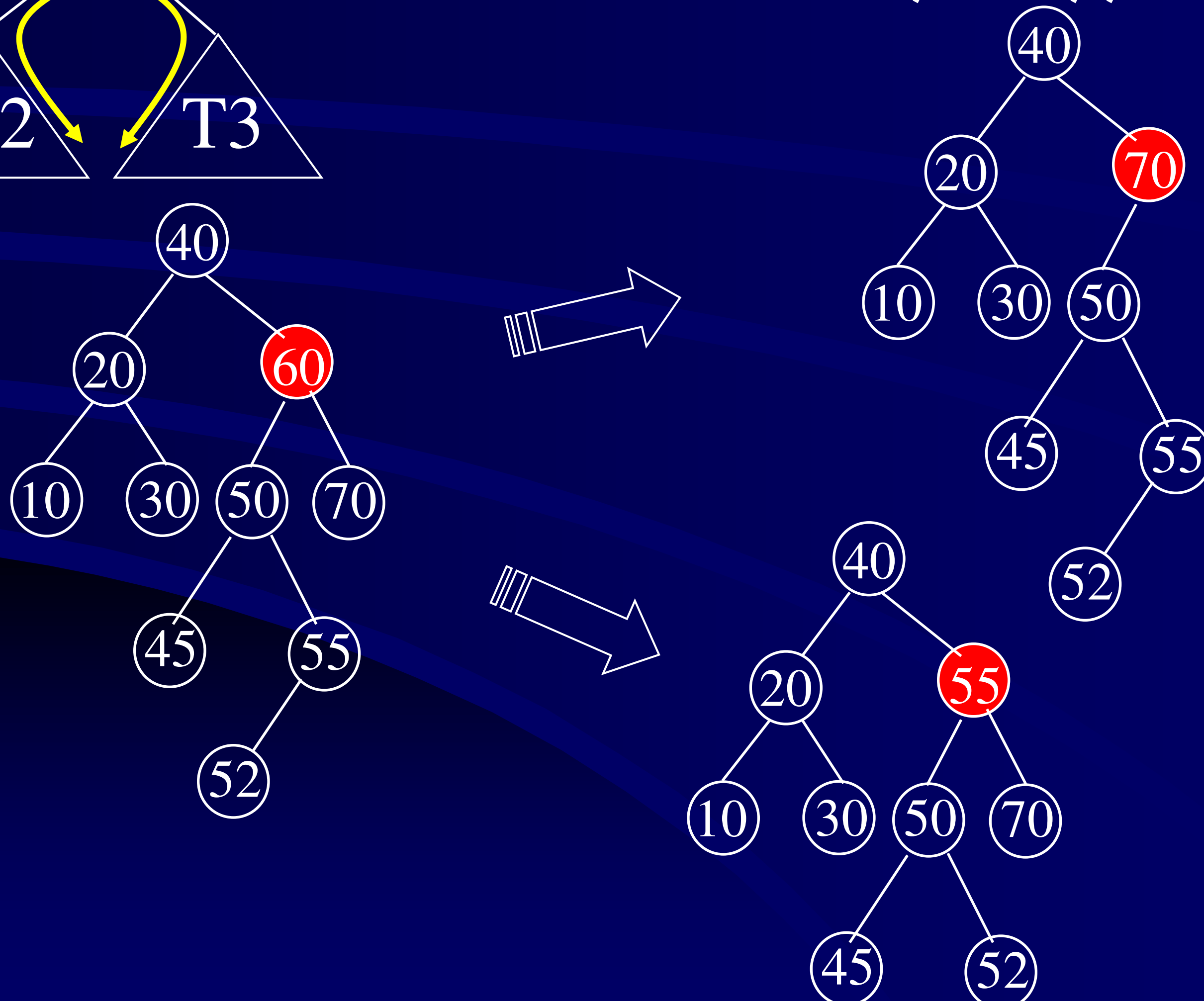


حذف (Delete) گره از BST



ـ حالت سوم: حذف گره با دو فرزند

جايگزين كردن گره اي كه بايد حذف شود با بزرگترين
گره زيردرخت چپ يا كوچكترين گره زيردرخت راست



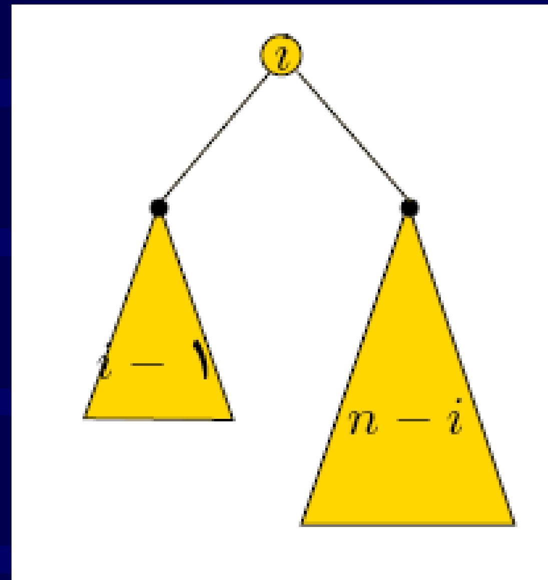
تعداد درخت های دودویی جست و جو با n عنصر و ارتفاع n-1 ؟

تعداد درخت های دودویی جست و جو با n عنصر و ارتفاع n-1 ؟

2^{n-1}

تعداد درخت های دودویی جست و جو

با $a_1 < a_2 < \cdots < a_n$ چند تا درخت متفاوت می‌توان ساخت؟

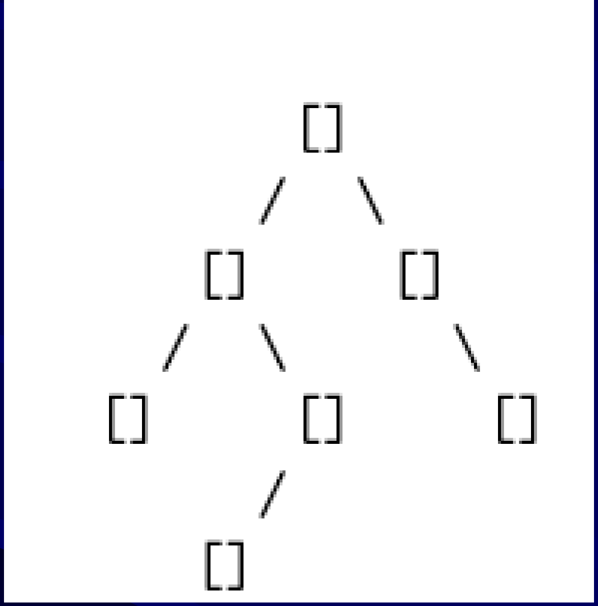


$$T(n) = \sum_{i=۱}^{n} T(i-۱)T(n-i),$$
$$T(۰) = ۱$$

$$T(n) = \frac{۱}{n+۱}\binom{۲n}{n}$$

عدد کاتالان

اعداد {8,1,13,14,9,22,6} را به درخت زیر طوری نسبت دهید که درخت دودویی جستجو حاصل شود



```
        [13]
       /    \
    [6]      [14]
   /   \         \
[1]     [9]       [22]
       /
    [8]
```

این دنباله‌های درج در یک درخت تهی درخت فوق را می‌سازد

13, 6, 9, 1, 14, 8, 22

13, 14, 6, 22, 1, 9, 8

13, 6, 1, 9, 8, 14, 22

به چند حالت می توان اعداد فوق را وارد یک درخت تهی کرد تا
در انتها درخت فوق حاصل شود؟

مسیر جستجوی x در بازه 1 تا 1000

<2, 252,401,398,330,344,397, 362>
<924, 220, 911, 244, 898, 258, 362, 363>

<925, 202, 911, 240, 912, 245, 363>

الگوریتمی کارا ارایه دهید که مشخص نمایید آیا A یک دنباله جستجو است؟

درخت انتخاب (Selection tree)

مسئله: تركيب k ليست مرتب شده (run) و توليد يك ليست مرتب شده

راه حل اول:

- پيدا كردن كمترين عنصر بين k كوچكترين عنصر (k-1 مقايسه)
- حذف كوچكترين عنصر از ليست مربوطه و افزودن آن به ليست نتيجه
- تكرار مراحل فوق متناسب با تعداد كل عناصر (n)

هزينه الگوريتم: O(nk)

run 1	run 2	run 3	run 4	run 5	run 6	run 7	run 8
10	9	20	6	8	9	90	17
15	20	20	15	15	11	95	18
16	38	30	25	50	16	99	20
			28				

(Selection tree) درخت انتخاب

-هر گره حاوي كوچكترين فرزندانش است

-هزينه: $O(n*lg_2k)$