درختها- قسمت چهارم

سید مهدی وحیدی پور

با تشکر از دکتر جواد سلیمی

دانشگاه کاشان دانشکده برق و کامپیوتر

درختها

- مقدمه
  - بازنمایی درخت ها
- درخت های دودویی
- پیمایش درختهای دودویی
- عملیات دیگر روی درختهای دودویی
- درختهای دودویی نخ کشی شده
- Heap ها
- درختان جستجوی دودویی
- درختهای انتخاب
- جنگل ها
- نمایش مجموعه ها
- شمارش درخت های دودویی متمایز

صف های اولویت

- عضوی که باید حذف شود عضو با بالاترین یا پایین ترین اولویت است
- مثال : تخصیص منابع

هر وقت که کامپیوتری آزاد شد

- کاربری که متقاضی زمان کمتری است انتخاب شود : (صف اولویت مینیموم)

در هر عمل حذف عضوی که کوچکترین مقدار کلید را دارد حذف می شود

- کاربری که پول بیشتری پرداخته است انتخاب شود: (صف اولویت ماکزیموم)

در هر عمل حذف عضوی که بزگترین مقدار کلید را دارد حذف می شود

صف های اولویت

- صف اولویت ماکزیمم (صف مینیموم به صورت مشابه قابل تعریف)
- ساده ترین پیاده سازی : لیست خطی مرتب نشده

نمایش ترتیبی

اضافه کردن عضو به آسانی در انتهای صف O(1)

حذف یک عضو نیازمند جستجوی عضو با بزرگترین کلید O(n)

نمایش پیوندی

اضافه کردن عضو به آسانی در ابتدای زنجیر O(1)

حذف یک عضو نیازمند جستجوی عضو با بزرگترین کلید O(n)

صف های اولویت

- صف اولویت ماکزیمم (صف مینیموم به صورت مشابه قابل تعریف)
- ساده ترین پیاده سازی : لیست خطی مرتب نشده

نمایش ترتیبی Unordered array

اضافه کردن عضو به آسانی در انتهای صف O(1)

حذف یک عضو نیازمند جستجوی عضو با بزرگترین کلید O(n)

نمایش پیوندی Unordered linked list

اضافه کردن عضو به آسانی در ابتدای زنجیر O(1)

حذف یک عضو نیازمند جستجوی عضو با بزرگترین کلید O(n)

صف های اولویت

- صف اولویت ماکزیمم (صف مینیموم به صورت مشابه قابل تعریف)
- پیاده سازی با لیست خطی مرتب

نمایش ترتیبی **Sorted Array**

عضوها به صورت غیر نزولی

اضافه کردن عضو	O(n)
حذف یک عضو	O(1)

نمایش پیوندی **Sorted linked list**

عضوها به صورت غیر صعودی

اضافه کردن عضو	O(n)
حذف یک عضو	O(1)

صف های اولویت

- صف اولویت ماکزیمم (صف مینیموم به صورت مشابه قابل تعریف)

Representation	Insertion	Deletion
Unordered array	$\Theta(1)$	$\Theta(n)$
Unordered linked list	$\Theta(1)$	$\Theta(n)$
Sorted array	$O(n)$	$\Theta(1)$
Sorted linked list	$O(n)$	$\Theta(1)$
Max heap	$O(\log_2 n)$	$O(\log_2 n)$

Heap

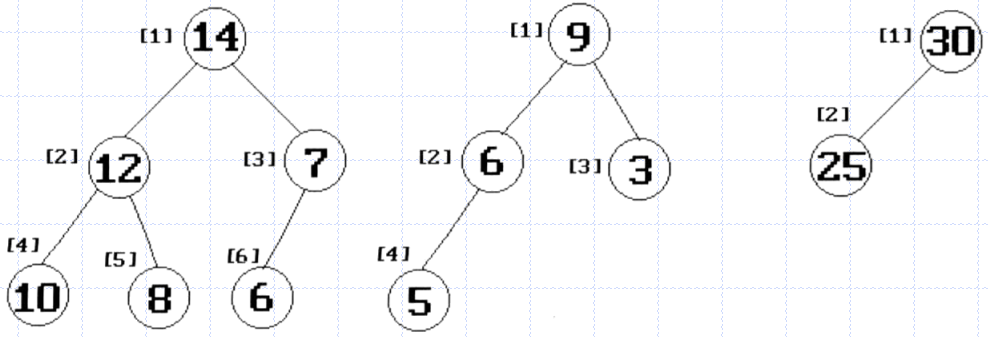
تعریف

- **max tree** درختی است که مقدار کلید هر گره آن کمتر از مقادیر کلیدهای فرزندانش نباشد.
- **max heap** یک درخت دودویی کامل است که همزمان max tree نیز می باشد.
- **min tree** درختی است که مقدار کلید هر گره آن بیشتر از مقادیر کلیدهای فرزندانش نباشد.
- **min heap** یک درخت دودویی کامل است که همزمان min tree نیز می باشد.

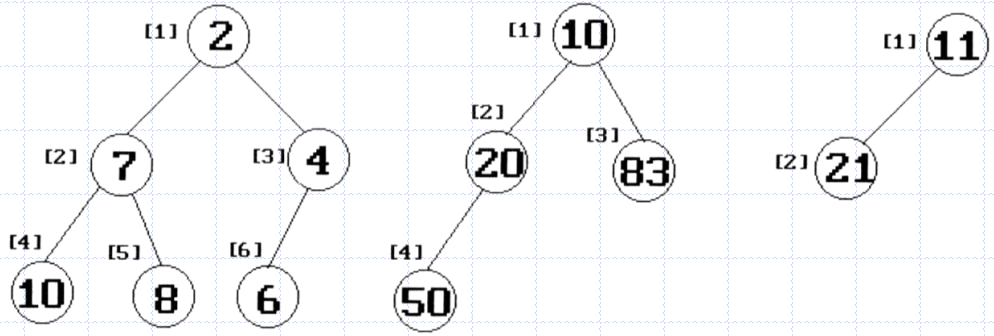
Heap

- مقدار کلید در گره ریشه در max heap (min heap) بزرگترین (کوچکترین) کلید در درخت است.
- **مثال**

max heap



min heap



Heap

عملیات اصلی روی یک Heap

- ایجاد یک Heap خالی
- اضافه کردن یک عنصر جدید به Heap
- حذف بزرگترین عنصر از Heap

Heap

- نوع داده مجرد Heap

structure *MaxHeap* is
objects: a complete binary tree of $n > 0$ elements organized so that the value in each node is at least as large as those in its children
functions:
for all *heap* ∈ *MaxHeap*, *item* ∈ *Element*, *n*, *max_size* ∈ integer

MaxHeap Create(*max_size*)	::=	create an empty heap that can hold a maximum of *max_size* elements.
Boolean HeapFull(*heap*, *n*)	::	**if** (*n* == *max_size*) **return** *TRUE* **else return** *FALSE*
MaxHeap Insert(*heap*, *item*, *n*)	::=	**if** (!HeapFull(*heap*, *n*)) insert *item* into *heap* and return the resulting heap **else return** error.
Boolean HeapEmpty(*heap*, *n*)	::	**if** ($n > 0$) **return** *TRUE* **else return** *FALSE*
Element Delete(*heap*, *n*)	::=	**if** (!HeapEmpty(*heap*, *n*)) **return** one instance of the largest element in the heap and remove it from the heap **else return** error.

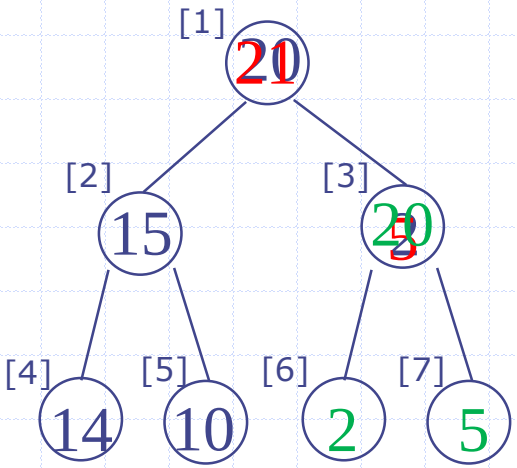
اضافه کردن یک عنصر در Max Heap

پیچیدگی اضافه کردن عنصر $O(\log_2 n)$

insert 21

*n= 5

i= 6




```
void insert_max_heap(element item, int *n)
{
/*insert item into a max heap of current size *n */
   int i;
   if (HEAP_FULL(*n)){
      fprintf(stderr, "The heap is full. \n");
      exit(1);
   }
   i = ++(*n);
   while ((i != 1) && (item.key > heap[i/2].key)) {
      heap[i] = heap[i/2];
      i /= 2;
   }
   heap[i] = item;
}
```

parent link

item up heap

حذف کردن یک عنصر از Max Heap

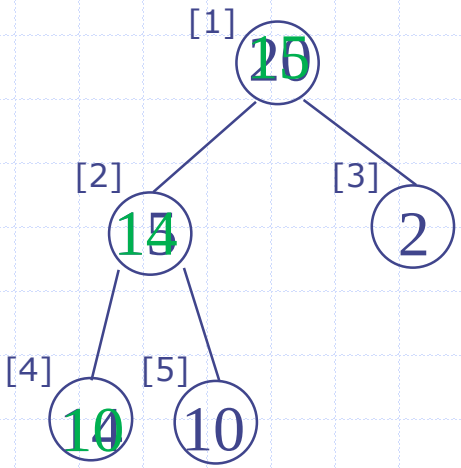
پیچیدگی حذف کردن عنصر $O(\log_2 n)$

```
element delete_max_heap(int *n)
{
/* delete element with the highest key from the heap */
   int parent, child;
   element item, temp;
   if (HEAP_EMPTY(*n)) {
      fprintf(stderr, "The heap is empty\n");
      exit(1);
   }
   /* save value of the element with the highest key */
   item = heap[1];
   /* use last element in heap to adjust heap */
   temp = heap[(*n)--];
   parent = 1;
   child = 2;
   while (child <= *n) {
      /* find the larger child of the current parent */
      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;
}
```

parent = 1

child = 2

*n= 5



item.key = 20

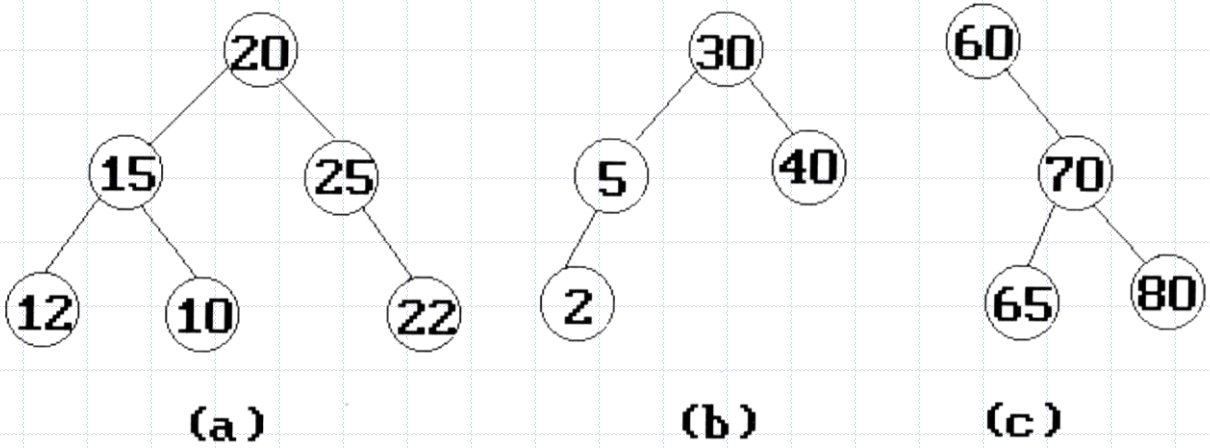
temp.key = 10

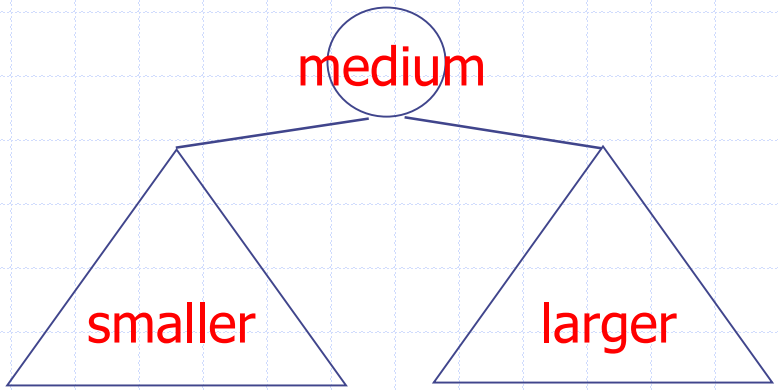
درختهای جستجوی دودویی

- دلیل نیاز به درختهای جستجوی دودویی
  - Heap برای کاربردهایی که نیاز به حذف یک عنصر دلخواه دارند اصلا مناسب نیست.
    - حذف کوچکترین (بزرگترین) عنصر $O(\log_2 n)$
    - حذف یک عنصر دلخواه $O(n)$
    - جستجوی یک عنصر دلخواه $O(n)$
- **تعریف** درخت جستجوی دودویی
  - هر عنصر دارای یک کلید است.
  - کلیدهای زیردرخت غیرتهی چپ (راست) کوچکتر (بزرگتر) از کلید واقع در ریشه هستند.
  - زیردرختهای چپ و راست نیز خود درختهای جستجوی دودویی هستند.

درختهای جستجوی دودویی

- مثال: b و c درخت جستجوی دودویی هستند

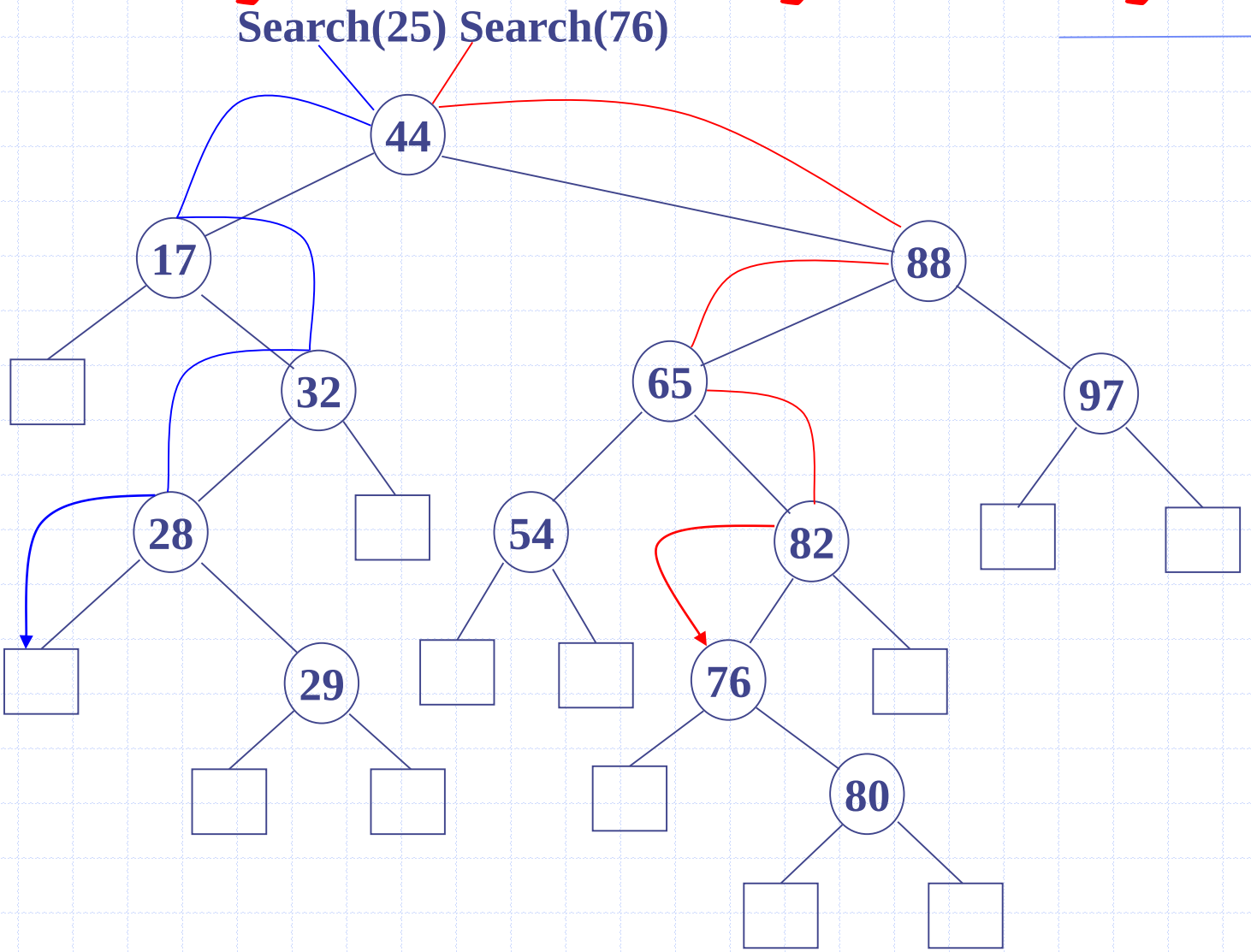




جستجوی یک عضو در درخت جستجوی دودویی

- فرض کنید می خواهیم مکان عنصری با کلید x را پیدا کنیم. کار خود را از ریشه (root) شروع می کنیم ، اگر ریشه تهی باشد، درخت جستجو فاقد هر عنصری بوده و جستجو ناموفق خواهد بود. در غیر این صورت x را با مقدار کلید ریشه مقایسه می کنیم :
- اگر x برابر با این کلید باشد آنگاه جستجو به طور موفقیت آمیز به پایان می رسد.
- اگر x کوچکتر از مقدار کلید ریشه باشد، هیچ عنصری در زیردرخت راست نمی تواند که مقدار کلید x را دارا باشد و فقط زیر درخت چپ باید جستجو شود.
- اگر x بزرگتر از مقدار کلید ریشه باشد، فقط لازم است زیردرخت راست را جستجو می کنیم.

جستجوی یک عضو در درخت جستجوی دودویی



جستجوی یک عضو در درخت جستجوی دودویی

جستجوی بازگشتی

```
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);
}
```

جستجوی تکراری

```
tree_pointer search2(tree_pointer tree, int key)
{
/* return a pointer to the node that contains key.  If
there is no such node, return NULL. */
   while (tree) {
      if (key == tree->data) return tree;
      if (key < tree->data)
         tree = tree->left_child;
      else
         tree = tree->right_child;
      }
   return NULL;
}
```

O(h)

اگر h ارتفاع یا عمق یک درخت جستجوی دودویی باشد، عمل جستجو در مدت O(h) انجام می شود.

اضافه کردن یک عضو در درخت جستجوی دودویی

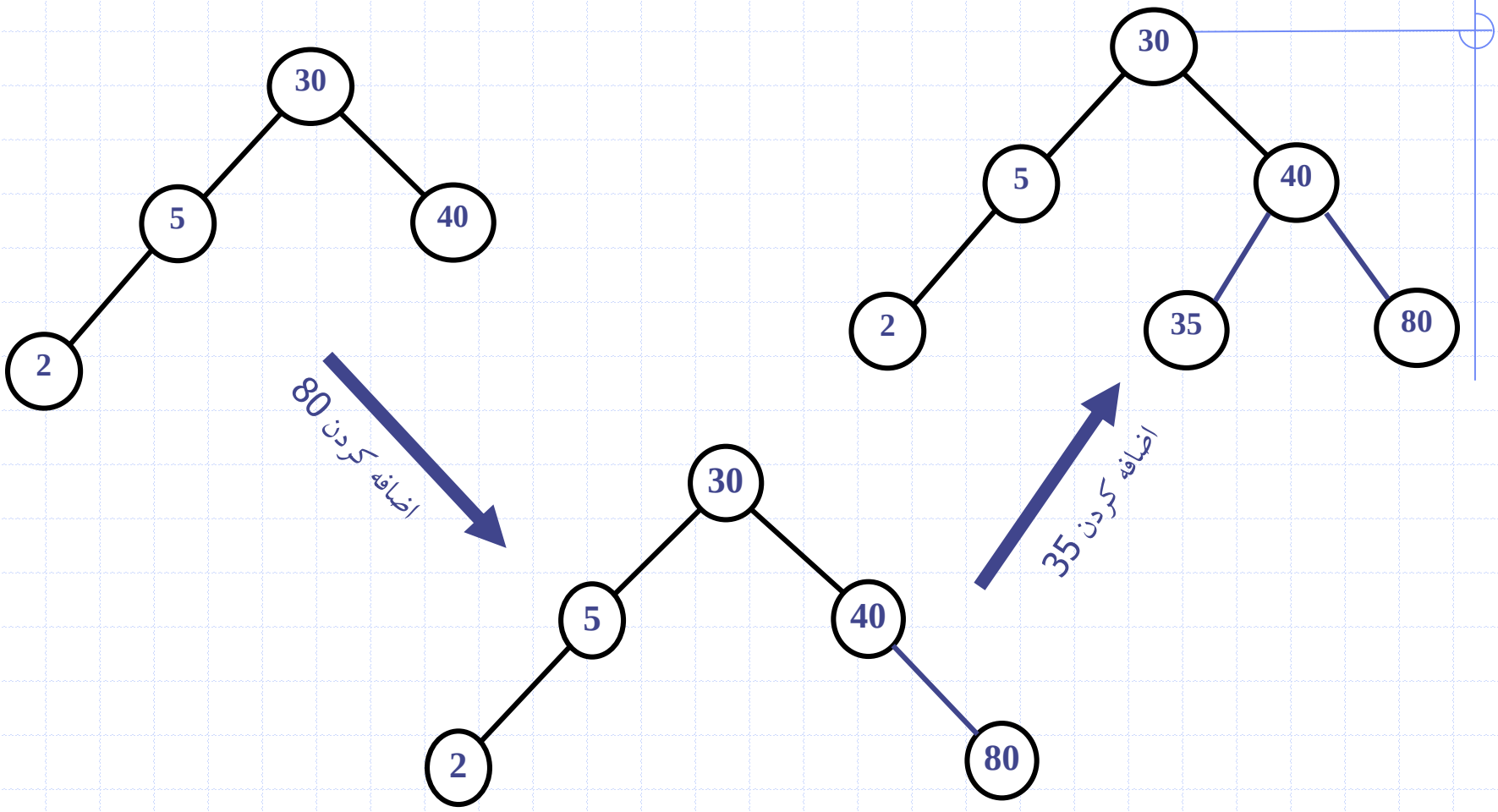
- برای درج عنصر جدید X در درخت ، ابتدا باید مشخص کنیم که آیا با کلید عناصر موجود در درخت متفاوت است یا خیر. برای انجام این کار باید درخت را جستجو کرد. اگر جستجو ناموفق باشد ، عنصر را در محلی که جستجو خاتمه پیدا نموده است اضافه می کنیم.

اضافه کردن یک عضو در درخت جستجوی دودویی

```
int Insert_node(tree-pointer  tree, int x)
//Insert x into the binary search tree
{
// search for x, q is the parent of p
tree –pointer  p = tree, q=0;
while ( p) {
    q=p;
    if (x==p-> data) return false; //x is already in tree
    if (x <p->data) p=p->leftchild;
    else p=p->rightchild;
}
//perform insertion
p= (tree-pointer) malloc (sizeof (node));
p->leftchild=p->rightchild=0; p->data=x;
if  (!tree) tree=p;
else if (x< q-> data) q->leftchild =p;
else q-> rightchild=p;
return true;
}
```

اگر h ارتفاع یا عمق یک درخت جستجوی دودویی باشد، عمل اضافه کردن در مدت O(h) انجام می شود.

اضافه کردن یک عضو در درخت جستجوی دودویی

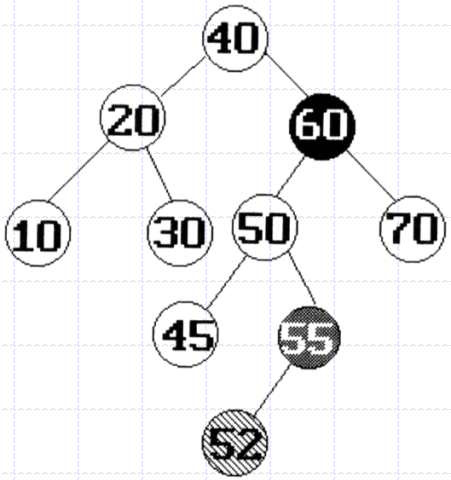


حذف یک عضو از درخت جستجوی دودویی

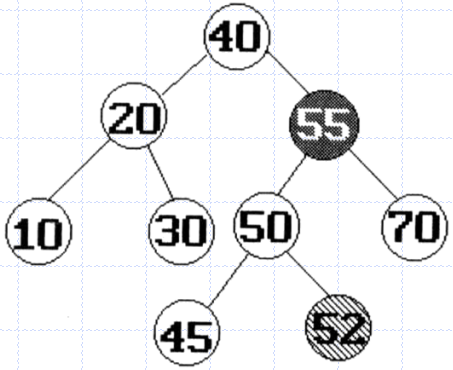
- ۳ حالت باید در نظر گرفته شود:
  - حالت۱: حذف عضو برگ
  - حالت ۲: حذف عضو غیر برگ که فقط یک بچه دارد
  - حالت ۳: حذف عنصر غیر برگ که دو بچه دارد. بزرگترین عضو در زیر درخت چپ یا کوچکترین عضو در زیر درخت راست (نزدیکترین عددها به عدد مذکور) جایگزین آن می شوند. حال خود عنصر جایگزین شونده باید از محل قبلی خود حذف شود. عنصر جایگزین شونده همواره درجه حداکثر یک دارد (حالت ۱ یا ۲) پس حذف آن ساده است.

حذف یک عضو از درخت جستجوی دودویی

- حالت ۳:
  - عنصر جایگزین شونده عنصر قبلی یا بعدی در پیمایش میانوندی است.
  - عنصر قبلی یا بعدی در پیمایش میانوندی نزدیکترین عنصرها به عنصر حذف شونده هستند.



(a) tree before deletion of 60



(b) tree after deletion of 60

اگر h ارتفاع یا عمق یک درخت جستجوی دودویی باشد، عمل حذف کردن در مدت O(h) انجام می شود.

درخت های جستجوی دودویی

- ارتفاع / عمق درخت جستجوی دودویی

- ارتفاع یک درخت جستجوی دودویی n عضوی می تواند به بزرگی n باشد.
- وقتی که عمیات اضافه و حذف کردن به صورت تصادفی انجام شود ارتفاع درخت جستجوی دودویی به طور متوسط $O(\log_2 n)$ است.
- درختهای جستجویی که در بدترین حالت عمق $O(\log_2 n)$ دارند درختهای جستجوی متوازن balanced search trees نامیده می شوند.
- درختان جستجوی متوازنی وجود دارند که عمل جستجو، درج و حذف در آنها در زمان $O(h)$ انجام می شود از جمله درختان red_black، 2-3، AVL