طراحی الگوریتم

جلسه پنجم

Binary search tree (B S T)

- Successor(P) : اولین گره بعد از P در پیمایش میان ترتیب
- Predecessor(P) : اولین گره قبل از P در پیمایش میان ترتیب.

$$BST \text{ توابع} \begin{cases} search \\ Minimom \\ maximom \\ predecessor \\ successor \\ Insert \\ Delete \end{cases} \rightarrow \theta \text{ (عمق درخت)}$$

$complete$: عمق درخت $= O(\log n)$

$liner\ chain$: عمق $= O\ (n) \rightarrow$

BST

$(Red - Black\ Tree)$: عمق $= O\ (n \log n)$

تفاوت عمق در دو سمت هر گره حداکثر یک می باشد $\rightarrow$ برای ذخیره اطلاعات $B - Tree$

Node:	Lchild	data	Rchild	Parent

```
Class Node {
   Private:
  Object data;
 Node*Lchild;
Node* Richild;
Node* Parent;
Public:
≡
};
```

```
class BST {
 Private:
  Node*root;
Public:
  ≡
 };
```

پیمایش میان ترتیب $\theta(n)$

P ریشه درخت

$$\text{INORDER} - \text{TREE} - \text{WALK}\ (Node^{*}P)$$

$$\{if\ (p! = NULL)$$

$$\{INORDER - TREE - WALK\ (P \rightarrow Lchild);$$

$$Print\ (P \rightarrow data);$$

$$INORDER - TREE - WALK\ (P\ \rightarrow Rchild);$$

- **بصورت بازگشتی:**

ریشه مقدار مورد نظر

↑ ↑

$Node^{*}\ Search\ (Node^{*}P, object\ K)\{$

$(if((P = NULL)\ \|\ (P \rightarrow data == k))$

$return\ (P)$:

$if\ (K < P \rightarrow data)$

$return\ \big(Search\ (P \rightarrow Lchild, k)\big)$:

$else$

$return\ \big(Search\ (P \rightarrow Rchild, k)\big)$;

}

- بدون بازگشتی: ancestor= جد

Predecessor (P)

Successor (P): اگر P فرزند راست، چپ داشته باشد که جواب مینیمم، ماکزیمم زیر درخت سمت راست، چپ P است اگر P فرزند راست، چپ نداشته باشد

در این صورت جواب پایین ترین P, ancector است که فرزند سمت راست، چپ آن نیز P, ancestor باشد.

```
Node*NoN − Rec − Search (Node*P, object k){
while (q! = NULL){
    if (q → data == k)          return (q);
  else if (q → data > k)        q = q → Lchild;
    eles                        q = q → Rchild;
}
return(NULL);
 }
```

```
Node*  Predecessor Successor (Node*P){
if (P → Lchild Rchild! = NULL)
return  (Maximom Minimmo (P → Lchild RChi ld);
q = P → Parent;
while ((q! = NULL)&& (q → Lchild Rchild == P){
      P = q;
       q = q → Parent
   }
 return (q);
}
```

```
void        INSERT (Node*z){
            if (root == NULL){root = z;return;}
            Node*P = root;
            while (ture){
                    if (z → data < P → data)
                            if(P → Lchild == NULL){
                                    P → Lchild = z;
                                    return;
                            }
                    else
                            P = P → Lchild;
                    else if (z → data > P → data)
                            if(P → Rchild == NULL)
                            P → Rchild = z;
                            return;
                             }
                     else
                             P = P → Rchild:
else
return;
} }
```

حذف Z می تواند به حذف y واقع شود. در هر حال x باید بجای yقرار گیرد. (y می تواند همان z و یا جایگزین z باشد.

$\boldsymbol{Delete\ (Node^{*}z)\{}$

$if((Z \rightarrow Rchild == NULL)\ \|\ (z \rightarrow Lchild = NULL)$ حداکثر یک فرزند دارد z

$y = z$

$Predecessor(z)$

$eles\ \ y \leftarrow Successor\ (z);$ دو فرزند دارد z

x برابر با فرزند غیر $NULL$ y می شود $\begin{cases} if\ (y \rightarrow\ \ Lchild! = NULL) & x = y \rightarrow Lchild; \\ else & x = y \rightarrow Rchild; \end{cases}$

X بجای y می نشیند.

$if\ (x! = NULL)x \rightarrow Parent = y \rightarrow parent);$

ریشه y $if\ (y \rightarrow Parent == NULL)root = x;$

$else\ \{\ if\ (y == y \rightarrow Parent\ \rightarrow Lchild)$

$y \rightarrow Parent \rightarrow \overset{Richild}{Lchild} = x;$

$else$

$y \rightarrow Parent\ \rightarrow \overset{Lchild}{Rchild} = x;$

}

$if(y! = z)\ \ Z \rightarrow data =\ \ y \rightarrow data;$

}

* تمرین

12 – 3 – 6

12 -2 , 12 – 4