数据结构作业第九章——查找

庄震丰 22920182204393

Dec. 2^{nd}, 2019

9-29

已知一非空有序表，表中记录元素按关键字顺序排列，以不带头节点的单循环链表作为存储结构，外设两个指针 h,t, 其中 h 始终指向刚刚查到的节点，查找算法的策略是，首先将定值 K 和 t->key 进行比较，若相等则查找成功，否则因 K 小于或大于 t->key 而从 h 所指结点或 t 所指结点的后继结点进行查找。

(1) 按上述查找过程编写查找算法。

(2) 画出查找过程的判定树，并分析在等概率查找的平均查找长度。

算法分析

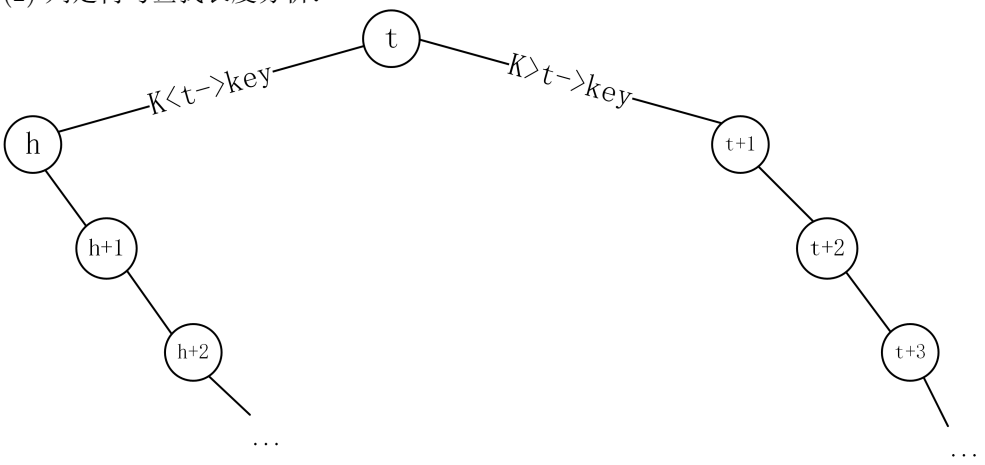
按照题目要求，如果首次查找失败 (小于)，从 h 开始往后进行查找，若大于则从 t 向后查找。如果查找长度超过了 n，则退出循环，改查找目标关键字不存在。

时间复杂度 $O(\frac{nm}{3})$, 空间复杂度 O(n), 其中 n 是顺序表长度，m 为查找次数。**code**

```
#include<bits/stdc++.h>
using namespace std;
struct node
{
    int key;
    node * next;
};
int n;
node *Build()
{
    cout<<"Please input the number:";
    cin>>n;
    node *h=(node *)malloc(sizeof(node));
    node *p1=h;
    node *p2=NULL;
    cin>>h->key;
    for (int i=1;i<=n-1;i++)
    {
        p2= (node *)malloc(sizeof(node));
        p1->next=p2;
        cin>>p2->key;
        p1=p2;
    }
    p1->next=h;
    return h;
}
node * Match(node *hp,int km)
{
    node *p=hp;
    for (int i=1;i<=n;i++)
        if (hp->key==km) return hp;
        else hp=hp->next;
    return NULL;
}
```

```
node * search(node* H,node* t,int K)
{
    if (t->key==K) return t;
    if (t->key>K) return Match(H,K);
    else return Match(t,K);
}
void init()
{
    int q;
    node *head=Build();
    node *t=head;
    cout<<"please input the query number:";
    cin>>q;
    for (int i=1;i<=q;i++)
        {
            int Key;
            cin>>Key;
            node *t1=search(head,t,Key);
            if (t1!=NULL)
            {
                t=t1;
                cout<<"Hash position is:"<<t<<endl;
            }
            else cout<<"Not found!"<<endl;
        }
}
int main()
{
    init();
    return 0;
}
```

(2) 判定树与查找长度分析：



平均查找长度分析：

设 f(i,j) 函数为 t=i,k=j 时的概率，则有 $f(i,j)=\frac{1}{n^2}$ 则对每种情况的查找长度分为一下几种情况：

$$L(i,j)=\begin{cases} j+1, & j<i \\ 1, & j=i \\ j-i+1, & j>i \ and \ j<n \end{cases}$$

则平均查找长度为：

$$\begin{aligned}&\sum\sum f(i,j)L(i,j)\\&=\sum_{i=1}^{n}\sum_{j=1}^{i-1}\frac{j+1}{n^2}+1+\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{j-i+1}{n^2}\\&=\frac{n^3+6n-4}{3n^2}\end{aligned}$$

因此平均查找时间复杂度为 $O(\frac{n}{3})$。

9-35

假设二叉排序树以后继线索链表作存储结构，编写输出该二叉树中所有大于 a 并且小于 b 的关键字的算法。

算法分析

若要形成排序线索树，那么应该按照中序线索树建造线索，并且支持动态插入，当有一个询问时，调用 search 函数进行中序遍历（线索）查询。

时间复杂度 O(logn)(平衡二叉树),O(n)(普通二叉树，取决于插入顺序)，空间复杂度 O(n)。**code**

```
#include<bits/stdc++.h>
#define maxn 1000
using namespace std;
struct node
{
    int num;
    node * lson;
    node * rson;
};
int n,m;
void insert(int x,node *hp)
{
    if (x<hp->num)
    {
        if (hp->lson!=NULL)
        {
            insert(x,hp->lson);
            return;
        }
        else
        {
            node*p1=(node *)malloc(sizeof(node));
            hp->lson=p1;
            p1->lson=NULL;p1->rson=NULL;
            p1->num=x;
            return;
        }
    }
    else
    {
        if (hp->rson!=NULL)
        {
            insert(x,hp->rson);
            return;
        }
        else
```

```
37              {
38                  node*p1=(node *)malloc(sizeof(node));
39                  hp->rson=p1;
40                  p1->lson=NULL;p1->rson=NULL;
41                  p1->num=x;
42                  return;
43              }
44          }
45      }
46      void search(node *hp,int Min,int Max)
47      {
48          if (hp->lson!=NULL)
49              search(hp->lson ,Min,Max);
50          if (hp->num>=Min && hp->num<=Max)
51              cout<<hp->num<<" ";
52          if (hp->rson!=NULL)
53              search(hp->rson ,Min,Max);
54      }
55      void init()
56      {
57          cout<<"please input the number:"<<endl;
58          cin>>n;
59          int a;
60          node *h=(node *)malloc(sizeof(node));
61          h->lson=NULL;
62          h->rson=NULL;
63          cin>>h->num;
64          for (int i=1;i<n;i++)
65          {
66              cin>>a;
67              insert(a,h);
68          }
69          cout<<"please input the query number:"<<endl;
70          cin>>m;
71          for (int i=0;i<m;i++)
72          {
73              int A,B;
74              cin>>A>>B;
75              search(h,A,B);
76              cout<<endl;
77          }
78      }
79      int main()
80      {
81          init();
82      }
```

textbf9-38

试写一个算法，将两个排序二叉树合并为一个排序二叉树。

textbf 算法分析

1. 将结点少的树拆散一个个插入另外一棵树，时间复杂度 O(nlogn)(平衡二叉树)，空间复杂度 O(n)。

2. 同时中序遍历两颗二叉树的同时建立另外一棵树，设立两个指针指向两树的当前结点。
时间复杂度 O(N), 空间复杂度 O(N). code

```
#include<bits/stdc++.h>
using namespace std;
typedef struct BinNode
{
    int data;
    struct BinNode *lchild;
    struct BinNode *rchild;
}BinNode,*BinTree;

void CreateBinTree(BinTree *tree)
{
    int val;
    scanf("%d",&val);
    if(val==-1)
    {
        (*tree)=NULL;
    }
    else
    {
        *tree=(BinTree)malloc(sizeof(BinNode));
        (*tree)->data=val;
        CreateBinTree(&((*tree)->lchild));
        CreateBinTree(&((*tree)->rchild));
    }
}

void Inorder(BinTree T)
{
    if(T)
    {
        Inorder(T->lchild);
        printf("%d ",T->data);
        Inorder(T->rchild);
    }
}

void Insert(BinTree *T,int key)
{
    if(!(*T))
    {
        (*T)=(BinTree)malloc(sizeof(BinNode));
        (*T)->data=key;
        (*T)->lchild=(*T)->rchild=NULL;
        return;
    }
    if(key==(*T)->data)
        return;
    if(key>(*T)->data)
        Insert(&((*T)->rchild),key);
    else
```

```
51              Insert(&((*T)->lchild),key);
52      }
53
54      void InsertBST(BinTree T1,BinTree T2)
55      {
56          if(T2)
57          {
58              InsertBST(T1,T2->lchild);
59              Insert(&T1,T2->data);
60              InsertBST(T1,T2->rchild);
61          }
62      }
63
64      int main()
65      {
66          BinTree T1=NULL;
67          BinTree T2=NULL;
68          CreateBinTree(&T1);
69          CreateBinTree(&T2);
70          InsertBST(T1,T2);
71          Inorder(T1);
72          printf("\n");
73          return 0;
74      }
```

9-40

在平衡二叉树的每个结点增设一个 lsize 域，其值为它左子树的结点数 +1，试写一个时间复杂度为 O(logn) 的算法，确定树中的第 k 小结点的位置。

算法分析

利用中序遍历现初始化 setup left(lsize 域)，当有询问 k 时，直接一层层比较，向叶子结点递归即可。由于树高度不超过 logn+1

时间复杂度 O(logn), 空间复杂度 O(n).

```
1       #include<bits/stdc++.h>
2       #define maxn 1000
3       using namespace std;
4       struct node
5       {
6           int num;
7           node * lson;
8           node * rson;
9           node * fa;
10          int left;
11      };
12      int n,m;
13      bool flag;
14      int number;
15      void insert(int x,node *hp)
16      {
17          if (x<hp->num)
18          {
```

```
19              if (hp->lson!=NULL)
20              {
21                  insert(x,hp->lson);
22                  return;
23              }
24              else
25              {
26                  node*p1=(node *)malloc(sizeof(node));
27                  hp->lson=p1;
28                  hp->left=0;
29                  p1->lson=NULL;p1->rson=NULL;p1->fa=hp;
30                  p1->num=x;
31                  return;
32              }
33          }
34          else
35          {
36              if (hp->rson!=NULL)
37              {
38                  insert(x,hp->rson);
39                  return;
40              }
41              else
42              {
43                  node*p1=(node *)malloc(sizeof(node));
44                  hp->rson=p1;
45                  hp->left=0;
46                  p1->lson=NULL;p1->rson=NULL;p1->fa=hp;
47                  p1->num=x;
48                  return;
49              }
50          }
51      }
52      void search(node *hp,int Min,int Max)
53      {
54          if (hp->lson!=NULL)
55              search(hp->lson ,Min,Max);
56          if (hp->num>=Min && hp->num<=Max)
57              cout<<hp->num<<" ";
58          if (hp->rson!=NULL)
59              search(hp->rson ,Min,Max);
60      }
61      void setup(node *hp,int k)
62      {
63          if (hp->lson!=NULL)
64              setup(hp->lson);
65              number++;
66              hp->left=number;
67              if (hp->left==k)
68          if (hp->rson!=NULL)
69              setup(hp->rson);
70      }
```

```
71      void del(node *hp,int inx)
72      {
73          if (flag) return;
74          if (inx<hp->num)
75              del(hp->lson,inx);
76          if (inx>hp->num)
77              del(hp->rson,inx);
78          if (inx==hp->num)
79              {
80                  flag=true;
81                  if  (hp->lson==NULL)
82                  {
83                      hp->lson->fa=hp->fa;
84                      if (hp->fa->lson==hp)
85                          hp->fa->lson=hp->rson;
86                      if (hp->fa->rson==hp)
87                          hp->fa->rson=hp->rson;
88                  }
89                  if (hp->lson!=NULL)
90                  {
91                       node*p1=hp->lson;
92                       if (p1->rson==NULL)
93                       {
94                           hp->lson->fa=hp->fa;
95                           if (hp->fa->lson==hp)
96                               hp->fa->lson=hp->lson;
97                           if (hp->fa->rson==hp)
98                               hp->fa->rson=hp->lson;
99                               hp->lson->rson=hp->rson;
100                              hp->rson->fa=hp->lson;
101                      }
102                      else
103                      {
104                          while(p1->rson!=NULL)
105                              p1=p1->rson;
106                          p1->fa->rson=p1->lson;
107                          p1->lson->fa=p1->fa;
108                          p1->lson=hp->lson;
109                          p1->rson=hp->rson;
110                          p1->fa=hp->fa;
111                          if (hp->fa->lson==hp)
112                              hp->fa->lson=p1;
113                          if (hp->fa->rson==hp)
114                              hp->fa->rson=p1;
115                          hp->rson->fa=p1;
116                          hp->lson->fa=p1;
117                      }
118                      free(hp);
119                  }
120              }
121          return;
122      }
```

```
123     void init()
124     {
125         cout<<"please input the number:";
126         cin>>n;
127         int a;
128         node *root=(node *)malloc(sizeof(node));
129         node *h=(node *)malloc(sizeof(node));
130         h->lson=NULL;
131         h->rson=NULL;
132         h->fa=root;
133         cin>>h->num;
134         for (int i=1;i<n;i++)
135         {
136             cin>>a;
137             insert(a,h);
138         }
139         cout<<"please input the query number:";
140         cin>>m;
141         for (int i=0;i<m;i++)
142         {
143             int A;
144             cin>>A;
145             setup(h,A);
146         }
147     }
148     int main()
149     {
150         init();
151     }
```