Coco_young

Splay tree 的实现

博客分类： 数据结构

splay 数据结构 tree

Splay tree 伸展树

1.基本思想：把本次访问的结点通过一系列的旋转操作，变换为根结点，同时要保持树为二叉查找树（BST）。

2.旋转操作:Zig,Zag.(代码注释中有说明)

3.核心操作：Splay(伸展).

4.5个基本操作：

Find(x,&Spt); //查找操作,在Spt中查找值为x的元素,然后把x所在的结点变为Splay tree的根结点

Insert(x,&Spt);//在spt中插入值为x的结点,并把x所在的结点变为Splay tree的根结点

Delete(x,&Spt);//在spt中删除值为x的结点

Join(&s1,&s2);//合并s1,s2两棵Splay tree

Split(x,&s,&s1,&s2);//把值为 x的 结点左右子树分离成2个Splay tree（s1,s2）

5.实现：(参考LRJ的书)

5(1).需要用到的数据结构:

int right[].left[],next[],father[];

DataType data[];

5(2).说明:

right[p],left[p]记录的是结点p的右儿子和左儿子.

father[p]是p 的父亲结点.

next[] 是存放结点的表，手动实现内存分配.

- 更多存档...

评论排行榜

- 通信——实现多人聊天室
- POJ_2337_欧拉回路+贪心

最新评论

Flywarrior：很喜欢楼主的编程风格，学习了。
POJ_2337_欧拉回路+贪心

igdnss：static List<Home> symbol= ...
我的多线程小游戏——坦克大战

yinger_fei：赞一个！！
我的多线程小游戏——坦克大战

Coco_young：25262875 写道您好楼主，我下来试了试，服务器启动不了。 ...
通信——实现多人聊天室

Coco_young：25262875 写道您好楼主，我下来试了试，服务器启动不了。 ...
通信——实现多人聊天室

data[p]对应p结点存放的数据.

5(3).实现代码:

Cpp代码

```cpp
#include<iostream>
#include<queue>
using namespace std;
/**
** author: yfr
** date: 2012-1-10
** project: splay tree
** reference: LRJ's Book
**/
#define SIZE 100
int Left[SIZE],Right[SIZE],father[SIZE],next[SIZE],data[SIZE];

/**
基本函数声明
**/
void Init();
int newnode(int d);
void delnode(int p);
//*********************************
void Zig(int &);
void Zag(int &);
void Splay(int &x,int &s);
int BST_find(int x,int p);
int SPT_find(int x,int &p);
int BST_insert(int x,int &p);
void SPT_insert(int x,int &p);
void remove(int x,int &p);
//*********************************
int findmax(int &s);
int findmin(int &s);
int join(int &s1,int &s2);
void split(int x,int &s,int &s1,int &s2);
//*********************************
/**
       /                              \
      p                               x
     / \          Zig(x)             / \
    x  <>   ----------------->     <>  p
   / \                                / \
  <> <>                              <> <>
**/
//zig zag 函数 注意指针修改时要成对去修改
void Zig(int &x)
{
     int p = father[x];
     Left[p] = Right[x];
     if(Right[x])//如果右子树存在
     father[Right[x]] = p;
     Right[x] = p;
     father[x] = father[p];
     father[p] = x;
     //这步很关键!!
     if(data[father[x]]>data[x])
     Left[father[x]] = x;
     else
     Right[father[x]] = x;
}
```

```
/**
       /                            \
      x                             p
     / \         Zag(x)            / \
    p  <>   <-----------------     <>  x
   / \                               / \
  <> <>                             <> <>
**/
void Zag(int &x)
{
     int p = father[x];
     Right[p] = Left[x];
     if(Left[x])//如果左子树存在
     father[Left[x]] = p;
     Left[x] = p;
     father[x] = father[p];
     father[p] = x;
     //这步很关键!!
     if(data[father[x]]>data[x])
     Left[father[x]] = x;
     else
     Right[father[x]] = x;
}
//s是树根，x是待伸长的结点
void Splay(int &x,int &s)
{
    int p;
    while(father[x])
    {
       p = father[x];
       if(!father[p])//如果p是根
       {
           if( x == Left[p])
           Zig(x);
           else if( x == Right[p])
           Zag(x);
       }
       else//如果不是树根
       {
           int g = father[p];//祖先结点
           if(Left[g]==p && Left[p]==x) //LL的情况
           {
               Zig(p);
               Zig(x);
           }
           else if(Left[g]==p&&Right[p]==x) //LR的情况
           {
               Zag(x);
               Zig(x);
           }
           else if(Right[g]==p&&Left[p]==x) //RL的情况
           {
               Zig(x);
               Zag(x);
           }
           else if(Right[g]==p&&Right[p]==x) //RR的情况
           {
               Zag(p);
               Zag(x);
           }
       }
    }
    s = x;//变为树根
}
```

```
122.  //初始化各容器
123.  void Init()
124.  {
125.       memset(Left,0,sizeof(Left));
126.       memset(Right,0,sizeof(Right));
127.       memset(father,0,sizeof(father));
128.       for(int i=0;i<SIZE;i++)
129.       next[i] = i+1;
130.  }
131.
132.  //模拟内存分配函数
133.  int newnode(int d)
134.  {
135.      int p = next[0];
136.      next[0] = next[p];
137.      data[p] = d;
138.      return p;
139.  }
140.  void delnode(int p)
141.  {
142.       Left[p] = Right[p] = father[p] = 0;
143.       next[p] = next[0];
144.       next[0] = p;
145.  }
146.
147.  //*********返回插入结点的编号,以便用来Splay*************//
148.  int BST_insert(int x,int &p)
149.  {
150.      if(!p){ p = newnode(x); return p;}
151.      else if(x < data[p])
152.      {
153.           int q = BST_insert(x,Left[p]);
154.           father[Left[p]] = p;//修改父亲指针
155.           return q;
156.      }
157.      else if(x >= data[p])
158.      {
159.           int q = BST_insert(x,Right[p]);
160.           father[Right[p]] = p;//修改父亲指针
161.           return q;
162.      }
163.  }
164.  //SPT的insert操作
165.  void SPT_insert(int x,int &p)
166.  {
167.      int q = BST_insert(x,p);
168.      Splay(q,p);//伸展
169.  }
170.  int BST_find(int x,int p)
171.  {
172.      if(!p)return 0;//空树
173.      if(x == data[p]) return p;
174.      else if(x < data[p]) return BST_find(x,Left[p]);
175.      else return BST_find(x,Right[p]);
176.  }
177.  int SPT_find(int x,int &s)
178.  {
179.      int q = BST_find(x,s);
180.      if(q)//如果查找成功
181.      Splay(q,s);
182.      return q;
183.  }
184.  int findmax(int &s)
185.  {
```

```
186.        int p = s;
187.        while(Right[p]) p = Right[p];
188.        Splay(p,s);
189.        return p;
190.  }
191.  int findmin(int &s)
192.  {
193.        int p = s;
194.        while(Left[p]) p = Left[p];
195.        Splay(p,s);
196.        return p;
197.  }
198.
199.  /*******************************************************/
200.  int join(int &s1,int &s2)
201.  {
202.      father[s1] = father[s2] = 0;//断开与公共先祖结点的连接 ,此步很关键
203.      if(!s1) return s2;
204.      if(!s2) return s1;
205.      int q = findmax(s1);
206.      Right[s1] = s2;
207.      father[s2] = s1;
208.      return s1;
209.  }
210.  void split(int x,int &s,int &s1,int &s2)
211.  {
212.       int p = SPT_find(x,s);
213.       s1 = Left[p];
214.       s2 = Right[p];
215.  }
216.  void remove(int x,int &p)
217.  {
218.       int q = SPT_find(x,p);
219.       if(q){//如果找到了x
220.       int temp = p;
221.       p = join(Left[p],Right[p]);
222.       delnode(temp);
223.       }
224.  }
225.  /*************************************************************/
226.
227.  //辅助函数
228.  void order(int p)
229.  {
230.     if(!p)return;
231.     order(Left[p]);
232.     cout<<data[p]<<endl;
233.     order(Right[p]);
234.  }
235.  void bfs(int p)
236.  {
237.     if(!p)return;
238.     queue<int> q;
239.     q.push(p);
240.     while(!q.empty())
241.     {
242.        int v = q.front();
243.        q.pop();
244.        if(Left[v]) q.push(Left[v]);
245.        if(Right[v]) q.push(Right[v]);
246.        cout<<data[v]<<endl;
247.     }
248.  }
249.  int main()
250.  {
```

```
251.     freopen("dataout.txt","w",stdout);
252.     Init();
253.     int ROOT = 0;
254.     SPT_insert(12,ROOT);//build SPT
255.     SPT_insert(8,ROOT);
256.     SPT_insert(2,ROOT);
257.     SPT_insert(7,ROOT);
258.     SPT_insert(13,ROOT);
259.     remove(13,ROOT);
260.     order(ROOT);
261.     cout<<"--------------"<<endl;
262.     bfs(ROOT);
263.     cout<<"------------"<<endl;
264.     cout<<father[ROOT]<<endl;
265.     cout<<"-------------"<<endl;
266.     int s1,s2;
267.     split(7,ROOT,s1,s2);
268.     bfs(s1);
269.     cout<<"----------"<<endl;
270.     bfs(s2);
271.     return 0;
272. }
```