代码模板

快速排序分两步走：1.主分治函数

1. 次交换函数

代码模板：

第一种写法：void quick\_sort(int it[],int lef,int rig){//主排序

      if(lef>=rig){

        return;

      }

      int pos=back\_pos(it,lef,rig);//返回枢轴的位置->枢轴在数组中的位置已经确定

      quick\_sort(it,lef,pos-1);//只需要排枢轴左区间 和右区间

      quick\_sort(it,pos+1,rig);

 }

 int back\_pos(int it[],int lef,int rig){//交互函数 将比枢轴大的数字放枢轴右边 比枢轴小的位置放枢轴左边

       int tem=it[lef];

       while(lef<rig){

        while(lef<rig&&tem<=it[rig]){

               rig--;

        }

        if(lef<rig){

          swap(it,lef,rig);

        }

        while(lef<rig&&it[lef]<=tem){

          lef++;

        }

        if(lef<rig){

          swap(it,lef,rig);

        }

       }

       return lef;//返回枢轴的位置

 }

第二种模板：

void quick\_sort(int it[],int lef,int rig){//主排序

      if(lef>=rig){

        return;

      }

     //返回枢轴的位置->枢轴在数组中的位置已经确定

      int kl=lef;

      int kr=rig;

      int tem=it[lef];

       while(lef<rig){

        while(lef<rig&&tem<=it[rig]){

               rig--;

        }

        if(lef<rig){

          swap(it,lef,rig);

        }

        while(lef<rig&&it[lef]<=tem){

          lef++;

        }

        if(lef<rig){

          swap(it,lef,rig);

        }

       }

       int pos=lef;

      quick\_sort(it,kl,pos-1);//只需要排枢轴左区间 和右区间

      quick\_sort(it,pos+1,kr);

 }

归并排序：分两步走：1.拆分函数 2.合并函数 ------》 需要辅助数组帮助

别学大话：直接利用原数组操作，help数组还 转两次 容易出问题

辅助数组直接在合并中使用更高效。

void mergesort(int uions[],int lef ,int rig){

  if(lef>=rig){

     return;

  }

  else{

    int mid=(lef+rig)/2;//分割点

  mergesort(uions,lef,mid);//递归的将former[lef---mid]归并为有序的help[lef--mid]

  mergesort(uions,mid+1,rig);

  mergy(uions,lef,rig,mid);

  }//将help[lef--mid]和help[mid+1--rig]归并成有序的uions[lef--rig];

 }

 void mergy(int uions[],int lef,int rig, int mid){

  int help[rig-lef+1];//辅助数组

  int k=0;

  int i=lef;//合并的左半区间起始

  int j=mid+1;//合并的右半区间起始

  for( i=lef,j=mid+1;lef<=mid&&j<=rig;k++){

    if(uions[i]>=uions[j]){

       help[k]=uions[j++];

    }

    else{

       help[k]=uions[i++];

    }

  }

  //此时的hlpe[i]没被使用

  if(i<=mid){//如果左半区间还剩元素

       for(int p=i;p<=mid;k++){

        help[k]=uions[p++];//放入剩下的元素

       }

  }

  //此时的hlpe[j]没被使用

  if(j<=rig){//如果右半区间还剩元素

    for(int p=j;p<=rig;k++){

     help[k]=uions[p++];//放入剩下的元素

    }

  }

  for(int l=0;l<rig-lef+1;l++){

    uions[lef+l]=help[l];

  }

return;

 }

总结：两个排序都是运用来分治的思想，对于分治思想常常运用于 题基是一维数组的问题并且这类问题可以拆成多个无关联的小问题。

堆排序：

两步走：1.建立堆 (从第一个叶子节点的前一个节点出发)

1. 交换 堆顶和堆尾（每交换一次要从堆顶开始维护堆）

void build\_heap(int it[],int n,int start){//堆排序第一步建立堆这种数据结构

         if(start<0){

          return;

         }

         int  lagest=start;

         int lef=start\*2+1;

         int right=start\*2+2;

         if(lef<n&&it[lef]>it[lagest]){

            lagest=lef;

         }

         if(right<n&&it[right]>it[lagest]){

          lagest=right;

         }

         if(lagest!=start){

          int tem=it[lagest];

          it[lagest]=it[start];

          it[start]=tem;

          build\_heap(it,n,lagest);

         }

         return;

 }

 void head\_sort(int it[]){

  int n=10;

  for(int i=n/2-1;i>=0;i--){//建立堆

    build\_heap(it,n,i);

  }

  for(int i=n-1;i>0;i--){

    int tep=it[0];

    it[0]=it[i];

    it[i]=tep;

    build\_heap(it,i,0);//交换完之后必须维护堆的性质,此时维护堆的性质要从堆顶开始

  }

 }

构建一颗字典树的代码模板

1. 首先定义一个类
2. 在类里面定义 节点的孩子数组 以及判断当前节点是否是字符串的末尾的变量
3. 巧妙的利用this 指针 指向当前节点的指针 （创建的对象被抽象成头节点了）然后利用数组的查找功能根据元素是否为空来创建节点

class Trie {

public:

     // 利用this 的特性在当前类上直接构建一颗 字典树

    //字典树 多叉树-》要用到指针数组来存储子树根节点

    vector<Trie\*> children;//C++ 会自动将字符初始化为空指针

    bool isend;

    Trie():children(26),isend(false) {

    }

    Trie \* searchPrefix(string prefix){

        Trie \* node=this;

        for(auto i:prefix){

            i-='a';

            if(node->children[i]==nullptr){

              return nullptr;

            }

            node=node->children[i];//迭代法查找多叉映射树

        }

        return node;

    }

    void insert(string word) {

        Trie \* node=this;

        for(auto i:word){

            i-='a';

            if(node->children[i]==nullptr){

                node->children[i]=new Trie();//new Trie();C++ 允许的写法 分配一Trie 大小的空间 并且调用构造函数 对其进行初始化

            }

            node=node->children[i];

        }

        node->isend=true;

    }

    bool search(string word) {

        Trie \*node =this->searchPrefix(word);

        return node!=nullptr&&node->isend;

    }

    bool startsWith(string prefix) {

            return this->searchPrefix(prefix)!=nullptr;

    }

};

使用结构题的字典树构造模板(使用map来构造)

struct Trie{

    unordered\_map<char,Trie\*> children; //先声明一个字典树的节点，里面包含节点数组 key 表示节点的值

};

Trie\* creat(vector<string> &dictionary){//迭代法构建一棵字典树

        Trie \*node =new Trie();

        for(auto key:dictionary){

            Trie \*cur=node;

            for(auto &kl:key){

               if(cur->children.count(kl)==0){

                   cur->children[kl]=new Trie();

               }

               cur=cur->children[kl];

            }

            cur->children['#']=new Trie;

//’#’ 表示 检查到前缀的末尾

        }

        return node;

}

C++ 实现split函数模板

 vector<string> split(string & str,char ch){//c++实现字符串分割模板

//双指针法

        vector<string> ans;

        int n=str.size();

        int pos=0;

        int start=0;

        while(pos<n){

            while(pos<n&&str[pos]==ch){//越过分割的字符

                pos++;

            }

            start=pos;

            while(pos<n&&str[pos]!=ch){

                pos++;

            }

            if(start<n){

                ans.emplace\_back(str.substr(start,pos-start));

            }

        }

        return ans;

    }