实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 实验名称 | 实验一 递归与分治算法设计 | | |
| 实验日期 | 2023年3月28日 |  |  |
| 学 号 | 2021213193 | 姓 名 | 李田 |
| 专业班级 | 2021级软件工程 2班 | | |
| 指导教师 | 谷志新 | | |

东北林业大学

软件工程专业

|  |
| --- |
| 1. 实验目的   掌握分治算法的基本思想，掌握分治算法的设计步骤及用递归技术实现分治策略。 |
| 1. 实验环境   Windows 7 以上操作系统，PC机，codeblocks环境 |
| 三、实验内容及结果  算法总体思想：对这k个子问题分别求解。如果子问题的规模仍然不够小，则再划分为k个子问题，如此递归的进行下去，直到问题规模足够小，很容易求出其解为止。将求出的小规模的问题的解合并为一个更大规模的问题的解，自底向上逐步求出原来问题的解。  分治法的设计思想是，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之，如下图所示。  **1、棋盘覆盖问题**  在一个2k×2k 个方格组成的棋盘中，恰有一个方格与其它方格不同，称该方格为一特殊方格，且称该棋盘为一特殊棋盘。在棋盘覆盖问题中，要用4种不同形态的L型骨牌覆盖给定的特殊棋盘上除特殊方格以外的所有方格，且任何2个L型骨牌不得重叠覆盖。  #include<bits/stdc++.h>  using namespace std;  typedef long long LL;  #define PLL pair<LL,LL>  #define PII pair<int,int>  #define rep(i,a,b) for(int i=a;i<=b;i++)  #define dep(i,a,b) for(int i=a;i>=b;i--)  #define lowbit(x) (x&-x)  #define x first  #define y second  const double eps=1e-8;  clock\_t startTime;  #define l first  #define r second  double getCurrentTime() {      return (double)(clock() - startTime) / CLOCKS\_PER\_SEC;  }  int timestamp=0;//用作标记L型骨牌编号  int a[101][101];//用以表示棋盘  //tr，tc表示棋盘的起始位置(第tr行，第tc列)，dr，dc表示特殊格子所在位置(第dr行，第dc列)，4\*4的棋盘tr，tc取值范围为0,1,2,3  //size表示棋盘大小，4\*4的棋盘size为4  void ChessBoard(int tr,int tc,int dr,int dc,int size)  {      if(size==1) return ;      size/=2;      timestamp++;      if(dr<=tr+size-1&&dc<=tc+size-1)      {        ChessBoard(tr,tc,dr,dc,size);      }else      {        a[tr+size-1][tc+size-1]=timestamp;        ChessBoard(tr,tc,tr+size-1,tc+size-1,size);      }      if(dr>tr+size-1&&dc>tc+size-1)      {        ChessBoard(tr+size,tc+size,dr,dc,size);      }else      {        a[tr+size][tc+size]=timestamp;        ChessBoard(tr+size,tc+size,tr+size,tc+size,size);      }      if(dr<=tr+size-1&&dc>tc+size-1)      {        ChessBoard(tr,tc+size,dr,dc,size);      }else      {        a[tr][tc+size]=timestamp;        ChessBoard(tr,tc+size,tr,tc+size,size);      }      if(dr>tr+size-1&&dc<=tc+size-1)      {        ChessBoard(tr+size,tc,dr,dc,size);      }else      {        a[tr+size][tc]=timestamp;        ChessBoard(tr+size,tc,tr+size,tc,size);      }  }  void solve()  {    int k;    cin>>k;    int fx,fy;    cin>>fx>>fy;    ChessBoard(0,0,fx,fy,1LL<<k);    for(int i=0;i<(1<<k);i++)    {      for(int j=0;j<(1<<k);j++)      {        cout<<a[i][j]<<" ";      }      cout<<endl;    }  }    int main()  {      int tt;      cin>>tt;      startTime = clock();      while(tt--)      {          solve();          //printf("%.2lf",getCurrentTime());      }  }  //运行结果      **2、合并排序问题**  对n个元素组成的序列进行排序。  基本思想：将待排序元素分成大小大致相同的两个子集合，分别对两个集合进行排序，最终将排序好的子集合合并成所要求的排好序的集合。  #include<bits/stdc++.h>  using namespace std;  const int N=1e6+10;  int a[N],temp[N];  int n;  void merge\_sort(int a[] ,int l,int r )  {        if(l>=r)  return ;        int mid=(l+r)>>1;  // 根据mid将数组划分为两部分递归处理        merge\_sort(a,mid+1,r),   merge\_sort(a,l,mid);        int i=l,j=mid+1,k=0;        while(i<=mid&&j<=r)        {            if(a[i]<=a[j])  temp[k++]=a[i++];          else  temp[k++]=a[j++];       }         while(i<=mid) temp[k++]=a[i++];         while(j<=r)  temp[k++]=a[j++];         for(int i=l,j=0;i<=r;i++,j++)  a[i]=temp[j];  }  int main()  {    cin>>n;      for(int i=0;i<n;i++)  cin>>a[i];    merge\_sort(a,0,n-1);    for(int i=0;i<n;i++)    cout<<a[i]<<' ';            return 0;  }  //运行结果：  8  Input:  99 77 1  3 4 222  721 33  Sorted\_Array:  1 3 4 33 77 99 222 721  **3、集合最大元问题**  在规模为n的数据元素集合中找出最大元。当n=2时，一次比较就可以找出两个数据元素的最大元和最小元。当n>2时，可以把n个数据元素分为大致相等的两半，一半有n/2个数据元素，而另一半有n/2个数据元素。 先分别找出各自组中的最大元，然后将两个最大元进行比较，就可得n个元素的最大元  #include<bits/stdc++.h>  using namespace std;  typedef long long LL;  #define PLL pair<LL,LL>  #define PII pair<int,int>  #define rep(i,a,b) for(int i=a;i<=b;i++)  #define dep(i,a,b) for(int i=a;i>=b;i--)  #define lowbit(x) (x&-x)  #define x first  #define y second  const double eps=1e-8;  clock\_t startTime;  double getCurrentTime() {      return (double)(clock() - startTime) / CLOCKS\_PER\_SEC;  }  const int N=1010;  int a[N];  int Iter\_Max(int a[],int l,int r)  {      if(l>r) return 0;      if(l==r) return a[l];      else      {        int mid=(l+r)>>1;        // 将区间一分为二，递归处理左右两部分最大值，然后取Max        return max(Iter\_Max(a,l,mid),Iter\_Max(a,mid+1,r));      }  }  void solve()  {    int n;    cin>>n;    cout<<"输入结果："<<endl;    rep(i,1,n)cin>>a[i];      int mx=Iter\_Max(a,1,n);    cout<<"最大值为："<<endl;    cout<<mx<<endl;  }  int main()  {      int tt;      tt=1;      startTime = clock();      while(tt--)      {          solve();          //printf("%.2lf",getCurrentTime());      }  }  输出结果：  8  Input:  99 77 1  3 4 222  721 33  Max\_element:  721 |
|  |

|  |
| --- |
| 四、实验过程分析与讨论  在解决棋盘覆盖问题时，要注意特殊点的位置，特殊点被分成了四个部分，而且这四个点的位置时不同的，开始时如果没有确定好这四个点的位置，很容易使得解决方案提前退出。 |
| 五、指导教师意见  指导教师签字：  年 月 日 |