**西南大学**

**计算机与信息科学学院**

课程设计报告

**课 程：**  数据结构课程设计

**题 目：**  基于晶格形式不规则体二分分裂算法的实现

**级、专业：** 2015 **级**  计科 **专业**  2 **班**

**学生姓名：**  杨浩

**提交日期：**  2017 **年** 6 **月** 21 **日**

|  |
| --- |
| **内容提要：**  **一、开发环境**  程序设计环境：Devcpp  图像显示环境:Matlab r2014a  运行环境：Windows 10  **二、设计目的**  运用数据结构相关知识，设计基于晶格形式不规则体二分分裂算法。  功能要求：   * 给定一个含有若干单粒的团簇，通过从三维坐标系的随机方向 （x,y,z）,以及随机位置（该位置必须在所有单粒在已确定分裂方向上的坐标最大值和最小值之间）上的连续分裂，每次将任意一个团簇一分为二，直到所有团簇都只含有一个单粒为止。 * 要求可以随时暂停程序，并将体系各粒子的坐标写入文件 * 读取文件中粒子的坐标信息，并在MATLAB中进行图形化显示   **三、设计内容**  **A.数据结构设计：**   1. 团簇：   用链表存储数据结构如下：  typedef struct LGroup{  int Xmin,Xmax,  int Ymin,Ymax;  int Zmin,Zmax;// 该团簇中单粒在x，y，z方向上下标的最大值和最小值  int sign,count,size;  //sign指团簇标号，count指该团簇所含单粒数目  //size指链接在该团簇之后的团簇数目  struct LNode \*head;//指向本团簇的第一个单粒  struct LGroup \*next;//指向下一个团簇  }Lgroup,\*LinkList;   1. 单粒：   typedef struct LNode  {  int x,y,z,sign;//结点坐标信息，sign表示单粒所属团簇的标号.  struct LNode \*next;//下一个结点  }Lnode,\*Node;   1. 存放被推动的团簇的栈：   typedef struct Stack\_LNode{  LinkList data;//存放一个团簇  struct Stack\_LNode \*next;  }Stack\_Lnode,\*Stack\_LinkList;  **B．算法设计：**  主要介绍两个重要算法，其余算法设计在算法描述中展示。   1. 统计团簇算法：   预先操作：在将文本文件中的坐标信息导入三维数组时，将坐标对应的元素赋值为noview(代表未访问)。  遍历三维数组，找到第一个值为noview的点,随后调用view函数；view函数部分结构如下:  void View(cubex Cube,int x,int y,int z,LinkList \*p)  {  Cube[x][y][z]=view;//首先将该节点由未访问设置为已访问。  if(z+1<=l-1)//z方向往上有相邻点  {  if(Cube[x][y][z+1]==noview)//该相邻点没有被访问  {  Node q1=(Node)malloc(sizeof(Lnode));  q1->x=x;q1->y=y;q1->z=z+1;  q1->sign=(\*p)->sign;  q1->next=(\*p)->head;(\*p)->head=q1;//将相邻点插入团簇  View(Cube,x,y,z+1,p);//递归访问相邻点  }  }  经过view函数的遍历能确定一个团簇，当view函数进行到周围已经没有相邻节点未访问时，函数退出运行。成功建立一个团簇，然后将该团簇链接至团簇链表头指针之后，继续遍历三维数组中下一个未访问的点。找到所有团簇。   1. 判断碰撞算法（简化）：   if(x-1>=0) //x-1未到达系统边界  {  if(cube[x-1][y][z]!=psign&&cube[x-1][y][z]!=0)  //x-1方向结点不属于本团簇的点，而且x-1方向有点。  {  Push(s,splitp1)//将原来的团簇入栈，由于原团簇碰撞了其他团簇，移动失败，需要入栈等待再次  lp=(\*L)->next;    while(lp)  {//在L指针后的团簇中找被碰撞的团簇  if(lp->sign==cube[x-1][y][z])  {  slp=(\*s);  while(slp)  {  if(slp->data->sign==lp->sign)  {  break;  }  else  {  slp=slp->next;    }  }//寻找栈中是否已有这个被撞的团簇，避免重复入栈  if(slp==NULL)  {  push(s,lp);//将被撞的团簇入栈  }  break;  }  else  {  lp=lp->next;  }  }  }  **四、算法描述**   1. **算法总体设计图如图所示：**     **2.分裂函数算法流程图：（分裂面两边的分裂函数大体相同）**    **五、程序结构**  **程序整体结构如图所示：**     1. **读文件模块**： 2. void ReadFile(cubex Cube,char\* filename)//读文件，将文件读入数组   将文件名为filename的文本文件中的坐标信息读入三维数组Cube，初始化元素值为未访问。   1. **统计团簇模块**： 2. void View(cubex Cube,int x,int y,int z,LinkList \*p)   递归遍历三维数组Cube中所有相邻的点，并将其链接到链表p的头节点之后   1. void countgroup(cubex Cube,LinkList \*L)   根据view函数得到的相邻信息建立团簇链表结构，每遍历完一个相邻点簇，新建一个团簇来储存它，并将这个新团簇链接到L指针之后。   1. **分裂模块**：   1). int View\_1(LinkList \*L,LinkList splitp1,cubex cube,Node p,Stack\_LinkList \*s,int k,int side,int sign)  判断团簇移动过程中是否发生碰撞，并将碰撞的团簇入栈  2). int split(LinkList \*L,LinkList \*splitp,cubex Cube,int i,int k)  分裂移动函数:首先进行碰撞检测，在碰撞检测完成后移动团簇在分裂面某一边的所有点，如果遇到系统边界，则换另外一边的团簇移动。若仍然遇到边界，则分裂无效，退出函数  3). int unionlist(LinkList \*up,cubex Cube,int i,int k)  合并移动函数:统计成功分裂以后形成的所有新团簇，先找出分裂面一边的最大团簇，保持该最大团簇不变，将其他团簇移回原位置。随后再次统计团簇，在统计过程中，实现了在小团簇移回原位置时发生碰撞而凝聚的过程。再对分裂面另一边的团簇做同样的操作，最终剩下两个团簇   1. 写入文件模块   1). void write\_file(cubex Cube,char \*filename)//将数组的坐标以及元素值信息写入文件。  **六、运行结果**  **运行Matlab团簇结构显示如下：**  **1）1个团簇**    **2）2个团簇**    **3）10个团簇**    **4）50个团簇**    **5）100个团簇**    **6）500个团簇**    **7)1000个团簇**    **8）2160个团簇**    **七、自我总结**  本次数据结构课程设计----基于晶格形式不规则体二分分裂算法的实现-----使我将c语言程序设计，数据结构等课程学到的理论知识充分地运用到了实践中去。课程设计中大量运用了数组，链表，栈等数据结构及其相关操作，巩固了我的课程所学。团簇的存储结构和处理碰撞的算法是设计的核心，经过自己的独立思考，成功调试运行。做设计的同时也领悟到了程序设计中每一个步骤的重要性，不再在没有需求分析和算法设计时直接开始编写程序。同时自己还有以下不足，程序模块化程度仍有待提高，算法的时间空间复杂度考虑得不够，部分算法仍有待优化。  **八、程序代码**  **链表操作以及存储结构头文件stack.h：**  #include<string.h>  #include<ctype.h>  #include<malloc.h> // malloc()等  #include<limits.h> // INT\_MAX等  #include<stdio.h> // EOF(=^Z或F6),NULL  #include<stdlib.h> // atoi()  #include<io.h> // eof()  #include<process.h> // exit()  //#include<iostream.h> // cout,cin  #define FALSE -1  #define ERROR -1  #define TRUE 1  #define OK 1  typedef struct LNode{  int x,y,z,sign,viewsignal;//结点坐标信息  struct LNode \*next;//下一个结点  }Lnode,\*Node;  typedef struct LGroup{  int Xmin,Xmax,Ymin,Ymax,Zmin,Zmax,sign,count,size;//size指团簇数目  struct LNode \*head;//指向本团簇的头指针  struct LGroup \*next;//指向下一个团簇  }Lgroup,\*LinkList;  //  typedef int Status;  typedef struct Stack\_LNode{  LinkList data;  struct Stack\_LNode \*next;  }Stack\_Lnode,\*Stack\_LinkList;  #define DestroyList ClearList  void InitList(Stack\_LinkList \*L)  {  \*L=NULL;  }  void ClearList(Stack\_LinkList \*L)  {  Stack\_LinkList p;  while(\*L)  {  p=\*L;  \*L=(\*L)->next;  free(p);  }  }  Status ListEmpty(Stack\_LinkList L)  {  if(L)  {  return FALSE;  }  else  {  return TRUE;  }  }  int ListLength(Stack\_LinkList L)  {  int i=0;  Stack\_LinkList p=L;  while(p)  {  p=p->next;  i++;  }  return i;  }  Status GetElem(Stack\_LinkList L,int i,LinkList \*e)  {  int j=1;  Stack\_LinkList p=L;  if(i<1)  {  return ERROR;  }  while(j<i&&p)  {  j++;  p=p->next;  }  if(j==i)  {  (\*e)=p->data;  return OK;  }  else  {  return ERROR;  }  }  Status ListInsert(Stack\_LinkList \*L, int i, LinkList e)  {  int j=1;  Stack\_LinkList p=(\*L);  Stack\_LinkList s;  if(i<1)  {  return ERROR;  }  s=(Stack\_LinkList)malloc(sizeof(Stack\_Lnode));  s->data=e;  if(i==1)  {  s->next=\*L;  (\*L)=s;  }  else  {  while(p&&j<i-1)  {  p=p->next;  j++;  }  if(!p)  {  return ERROR;  }  s->next=p->next;  p->next=s;  }  return OK;  }  Status ListDelete(Stack\_LinkList \*L,int i,LinkList \*e)  {  int j=0;  Stack\_LinkList p=(\*L),q;  if(i==1)  {  (\*L)=p->next;  (\*e)=p->data;  free(p);  }  else  {  while(p->next&&j<i-1)  {  p=p->next;  j++;  }  if(!p->next||j>i-1)  {  return ERROR;  }  q=p->next;  p->next=q->next;  (\*e)=q->data;  free(q);  }  return OK;  }  **Main.c:**  typedef Stack\_LinkList LinkStack;  #define InitStack InitList  #define DestroyStack DestroyList  #define ClearStack ClearList  #define StackEmpty ListEmpty  #define StackLength ListLength  Status GetTop(LinkStack S,LinkList \*e)  {  return GetElem(S,1,e);  }  Status push(LinkStack \*S,LinkList e)  {  return ListInsert(S,1,e);  }  Status pop(LinkStack \*S,LinkList \*e)  {  return ListDelete(S,1,e);  }  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <math.h>  #include "stack.h"  #define SElemType LinkList\*  #define l 100  #define noview 3000  #define view 3001  int counttime=0;  typedef int\* cubez;  typedef cubez\* cubey;  typedef cubey\* cubex;  cubex initcube(int i,int j,int k)//初始化三维数组  {  int x,y,z;  cubex Cube;  Cube=(cubex)malloc(i\*sizeof(cubey));  for(x=0;x<i;x++)  {  Cube[x]=(cubey)malloc(j\*sizeof(cubez));  for(y=0;y<j;y++)  {  Cube[x][y]=(cubez)malloc(k\*sizeof(int));  for(z=0;z<k;z++)  {  Cube[x][y][z]=0;//0表示没有点  }  }  }  return Cube;  }  void initcube\_1(cubex cube)  {  int x,y,z,count;  count=0;  for(x=0;x<l;x++)  for(y=0;y<l;y++)  for(z=0;z<l;z++)  if(cube[x][y][z]>0)  cube[x][y][z]=noview;  count++;  }  void initcube\_2(cubex cube)  {  int x,y,z,count;  count=0;  for(x=0;x<l;x++)  for(y=0;y<l;y++)  for(z=0;z<l;z++)  if(cube[x][y][z]!=0)  cube[x][y][z]=0;  count++;  }  void ReadFile(cubex Cube,char\* filename)//读文件，将文件读入数组  {  int i,x,y,z,c,count;  count=0;  FILE \*fp;  if ((fp=fopen(filename,"r"))==NULL)  {  printf("Read File failed! ");  return;  }  for(i=0;fscanf(fp,"%d %d %d",&x,&y,&z)!=EOF;i++)  {  Cube[x][y][z]=noview;  count++;  }  printf("readfile succes!,count:%d\n",count);  fclose(fp);  }  void View(cubex Cube,int x,int y,int z,LinkList \*p)  {  Cube[x][y][z]=view;  if(z+1<=l-1)  {  if(Cube[x][y][z+1]==noview)  {  Node q1=(Node)malloc(sizeof(Lnode));  q1->x=x;q1->y=y;q1->z=z+1;q1->sign=(\*p)->sign;  q1->next=(\*p)->head;(\*p)->head=q1;  View(Cube,x,y,z+1,p);  }  }  if(z-1>=0)  {  if(Cube[x][y][z-1]==noview)  {  Node q2;q2=(Node)malloc(sizeof(Lnode));  q2->x=x;q2->y=y;q2->z=z-1;q2->sign=(\*p)->sign;  q2->next=(\*p)->head;(\*p)->head=q2;  View(Cube,x,y,z-1,p);  }  }  if(x-1>=0)  {  if(Cube[x-1][y][z]==noview)  {  Node q3;q3=(Node)malloc(sizeof(Lnode));  q3->x=x-1;q3->y=y;q3->z=z;q3->sign=(\*p)->sign;  q3->next=(\*p)->head;(\*p)->head=q3;  View(Cube,x-1,y,z,p);  }  }  if(x+1<=l-1)  {  if(Cube[x+1][y][z]==noview)  {  Node q4;q4=(Node)malloc(sizeof(Lnode));  q4->x=x+1;q4->y=y;q4->z=z;q4->sign=(\*p)->sign;q4->next=(\*p)->head;  (\*p)->head=q4;View(Cube,x+1,y,z,p);  }  }  if(y+1<=l-1)  {  if(Cube[x][y+1][z]==noview)  {  Node q5;q5=(Node)malloc(sizeof(Lnode));  q5->x=x;q5->y=y+1;q5->z=z;q5->sign=(\*p)->sign;  q5->next=(\*p)->head;(\*p)->head=q5;  View(Cube,x,y+1,z,p);  }  }  if(y-1>=0)  {  if(Cube[x][y-1][z]==noview)  {  Node q6;q6=(Node)malloc(sizeof(Lnode));  q6->x=x;q6->y=y-1;q6->z=z;q6->sign=(\*p)->sign;  q6->next=(\*p)->head;(\*p)->head=q6;  View(Cube,x,y-1,z,p);  }  }  }  void minmax(LinkList \*lp)  {  Node p;  (\*lp)->Xmin=100;(\*lp)->Xmax=0;(\*lp)->Ymin=100;(\*lp)->Ymax=0;(\*lp)->Zmin=100;(\*lp)->Zmax=0;  p=(Node)malloc(sizeof(Lnode));p=(\*lp)->head;  while(p)  {  if((p->x)<(\*lp)->Xmin)  {  (\*lp)->Xmin=p->x;  }  if((p->x)>(\*lp)->Xmax)  {  (\*lp)->Xmax=p->x;  }  if((p->y)<(\*lp)->Ymin)  {  (\*lp)->Ymin=p->y;  }  if((p->y)>(\*lp)->Ymax)  {  (\*lp)->Ymax=p->y;  }  if((p->z)<(\*lp)->Zmin)  {  (\*lp)->Zmin=p->z;  }  if((p->z)>(\*lp)->Zmax)  {  (\*lp)->Zmax=p->z;  }  p=p->next;  }  }  void countgroup(cubex Cube,LinkList \*L)//在内循环中改变了x，y，z的值，内循环和外循环的值应该不同。  {  int x,y,z,x1,y1,z1,count,tcount;Node p;LinkList lp;  tcount=0;  int n=1;//当前新建团簇编号  for(x=0;x<l;x++)  {  for(y=0;y<l;y++)  {  for(z=0;z<l;z++)  {  if(Cube[x][y][z]==noview)//未访问，建新团簇  {  count=0;  lp=(LinkList)malloc(sizeof(Lgroup));  lp->sign=n;lp->next=NULL;  p=(Node)malloc(sizeof(Lnode));  p->x=x;p->y=y;p->z=z;p->sign=n;  p->next=NULL;  lp->head=p;  View(Cube,x,y,z,&lp);//遍历完所有相关结点  for(x1=0;x1<l;x1++)  {  for(y1=0;y1<l;y1++)  {  for(z1=0;z1<l;z1++)  {  if(Cube[x1][y1][z1]==view)  {  Cube[x1][y1][z1]=n;  count++;  }  }  }  }  lp->count=count;  tcount+=count;  minmax(&lp);  lp->next=(\*L)->next;  (\*L)->next=lp;  n++;  }  }  }  }  (\*L)->size=n-1;  printf("n:%d\n",n-1);  }  int random()//生成1-3的随机数 ,用于确定分裂面  {  int i;  srand((unsigned int)time(NULL));  i=rand()%3+1;  return i;  }  int locate(LinkList L,int k,int \*i)//确定分裂位置. BUG  {  int spliti,min,max;  if(k==1)  {  min=L->Zmin;max=L->Zmax;  }  else if(k==2)  {  min=L->Ymin;max=L->Ymax;  }  else{  min=L->Xmin;max=L->Xmax;  }  if(max!=min)  {  spliti=min+(rand()%(max-min));  return spliti;  }  else  {  k=rand()%3+1;  (\*i)=k;  return locate(L,k,i);  }  }  int Maxcount(LinkList \*L)  {  LinkList p;  p=(LinkList)malloc(sizeof(Lgroup));  int max;  p=(\*L)->next;  max=p->count;  while(p!=NULL)  {  if((p->count)>max)  {  max=p->count;  }  p=p->next;  }  return max;  }  LinkList choose(LinkList L)  {  int i=rand()%L->size+1;LinkList p;  p=(LinkList)malloc(sizeof(Lgroup));p=L;  while(p)  {  if(p->sign==i)  {  if(p->count>1)  {  return p;  }  else  {  return choose(L);  }  }  else  {  p=p->next;  }  }  }  void write\_file(cubex Cube,char \*filename)  {  FILE \*fp;  if ((fp=fopen(filename,"wt"))==NULL)  {  printf("Read File failed! ");  return;  }  int x,y,z;  for(x=0;x<l;x++)  {  for(y=0;y<l;y++)  {  for(z=0;z<l;z++)  {  if(Cube[x][y][z]>0)//未访问，建新团簇  {  fprintf(fp,"%d %d %d %d\n",x,y,z,Cube[x][y][z]);  }  }  }  }  fclose(fp);  }  void write\_file1(cubex Cube)  {  char filename[100]="01.txt";FILE \*fp;  if ((fp=fopen(filename,"wt"))==NULL)  {  printf("Read File failed! ");  return;  }  int x,y,z;  for(x=0;x<l;x++)  {  for(y=0;y<l;y++)  {  for(z=0;z<l;z++)  {  if(Cube[x][y][z]>0)//未访问，建新团簇  {  fprintf(fp,"%d %d %d\n",x,y,z);  }  }  }  }  fclose(fp);  }  int View\_1(LinkList \*L,LinkList splitp1,cubex cube,Node p,Stack\_LinkList \*s,int k,int side,int sign)  {  int x,y,z,peng,signal;  int i;  int psign=splitp1->sign;  LinkList lp,lp1;  lp=(LinkList)malloc(sizeof(Lgroup));  Stack\_LinkList slp;  peng=0;  signal=0;  if(k==1)  {  if(side==1)  {  x=p->x;y=p->y;z=p->z+1;  }  else {  x=p->x;y=p->y;z=p->z-1;  }  if(x-1>=0)  {  if(cube[x-1][y][z]!=psign&&cube[x-1][y][z]!=0&&cube[x-1][y][z]<=2160)  {  lp=(\*L)->next;  if(cube[x-1][y][z]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x-1][y][z])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  if(x+1<=l-1)  {  if(cube[x+1][y][z]!=psign&&cube[x+1][y][z]!=0&&cube[x+1][y][z]<=2160)  {  lp=(\*L)->next;  if(cube[x+1][y][z]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x+1][y][z])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {slp=slp->next;}  break;  }  else  {  lp=lp->next;  }  }  }  }  if(y-1>=0)  {  if(cube[x][y-1][z]!=psign&&cube[x][y-1][z]!=0&&cube[x][y-1][z]<=2160)  {  lp=(\*L)->next;  if(cube[x][y-1][z]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x][y-1][z])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;slp=(\*s);  while(slp)  {slp=slp->next;}  break;  }  else  {  lp=lp->next;  }  }  }  }  if(y+1<=l-1)  {  if(cube[x][y+1][z]!=psign&&cube[x][y+1][z]!=0&&cube[x][y+1][z]<=2160)  {  LinkList lp=(\*L)->next;  if(cube[x][y+1][z]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x][y+1][z])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {slp=slp->next;}  break;  }  else  {  lp=lp->next;  }  }  }  }  if(z-1>=0&&side==2)  {  if(cube[x][y][z-1]!=psign&&cube[x][y][z-1]!=0&&cube[x][y][z-1]<=2160)  {  lp=(\*L)->next;  if(cube[x][y][z-1]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x][y][z-1])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  if(z+1<=l-1&&side==1)  {  if(cube[x][y][z+1]!=psign&&cube[x][y][z+1]!=0&&cube[x][y][z+1]<=2160)  {  lp=(\*L)->next;  if(cube[x][y][z+1]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x][y][z+1])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  }  if(k==2)  {  if(side==1)  {  x=p->x;y=p->y+1;z=p->z;  }  else{  x=p->x;y=p->y-1;z=p->z;  }  if(x-1>=0)  {  if(cube[x-1][y][z]!=psign&&cube[x-1][y][z]!=0&&cube[x-1][y][z]<=2160)  {  lp=(\*L)->next;  if(cube[x-1][y][z]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x-1][y][z])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  if(x+1<=l-1)  {  if(cube[x+1][y][z]!=psign&&cube[x+1][y][z]!=0&&cube[x+1][y][z]<=2160)  {  lp=(\*L)->next;  if(cube[x+1][y][z]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x+1][y][z])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  if(z-1>=0)  {  if(cube[x][y][z-1]!=psign&&cube[x][y][z-1]!=0&&cube[x][y][z-1]<=2160)  {  lp=(\*L)->next;  if(cube[x][y][z-1]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x][y][z-1])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  if(z+1<=l-1)  {  if(cube[x][y][z+1]!=psign&&cube[x][y][z+1]!=0&&cube[x][y][z+1]<=2160)  {  lp=(\*L)->next;  if(cube[x][y][z+1]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x][y][z+1])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  if(y-1>=0&&side==2)  {  if(cube[x][y-1][z]!=psign&&cube[x][y-1][z]!=0&&cube[x][y-1][z]<=2160)  {  lp=(\*L)->next;  if(cube[x][y-1][z]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x][y-1][z])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  if(y+1<=l-1&&side==1)  {  if(cube[x][y+1][z]!=psign&&cube[x][y+1][z]!=0&&cube[x][y+1][z]<=2160)  {  LinkList lp=(\*L)->next;  if(cube[x][y+1][z]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x][y+1][z])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  }  if(k==3)  {  if(side==1)  {  x=p->x+1;y=p->y;z=p->z;  }  else  {  x=p->x-1;y=p->y;z=p->z;  }  if(y-1>=0)  {  if(cube[x][y-1][z]!=psign&&cube[x][y-1][z]!=0&&cube[x][y-1][z]<=2160)  {  lp=(\*L)->next;  if(cube[x][y-1][z]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x][y-1][z])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  if(y+1<=l-1)  {  if(cube[x][y+1][z]!=psign&&cube[x][y+1][z]!=0&&cube[x][y+1][z]<=2160)  {  LinkList lp=(\*L)->next;  if(cube[x][y+1][z]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x][y+1][z])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  if(z-1>=0)  {  if(cube[x][y][z-1]!=psign&&cube[x][y][z-1]!=0&&cube[x][y][z-1]<=2160)  {  lp=(\*L)->next;  if(cube[x][y][z-1]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x][y][z-1])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  if(z+1<=l-1)  {  if(cube[x][y][z+1]!=psign&&cube[x][y][z+1]!=0&&cube[x][y][z+1]<=2160)  {  lp=(\*L)->next;  if(cube[x][y][z+1]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x][y][z+1])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  if(x-1>=0&&side==2)  {  if(cube[x-1][y][z]!=psign&&cube[x-1][y][z]!=0&&cube[x-1][y][z]<=2160)  {  lp=(\*L)->next;  if(cube[x-1][y][z]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x-1][y][z])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  if(x+1<=l-1&&side==1)  {  if(cube[x+1][y][z]!=psign&&cube[x+1][y][z]!=0&&cube[x+1][y][z]<=2160)  {  lp=(\*L)->next;  if(cube[x+1][y][z]==sign)  {  return -1;  }  while(lp)  {  if(lp->sign==cube[x+1][y][z])  {  if(signal==0)  {  push(s,splitp1);  signal=1;  }  slp=(\*s);  i=1;  while(slp)  {  if(slp->data->sign==lp->sign&&signal==1&&i==2)  {  pop(s,&lp1);  signal=0;  break;  }  else if(slp->data->sign==lp->sign&&signal==1&&i!=2)  {  return -1;  break;  }  else  {  slp=slp->next;  i++;  }  }  if(slp==NULL)  {  push(s,lp);  }  peng=1;  slp=(\*s);  while(slp)  {  slp=slp->next;  }  break;  }  else  {  lp=lp->next;  }  }  }  }  }  return peng;  }  int split\_2(LinkList \*L,LinkList \*splitp,cubex Cube,int i,int k)  {  int totalsign=(\*splitp)->sign;  LinkList splitp1,lp;  lp=(LinkList)malloc(sizeof(Lgroup));  splitp1=(LinkList)malloc(sizeof(Lgroup));  int success;//分裂成功的标志变量  int zhuang;//是否碰撞  int splitp\_sign;//是否为被碰撞的团簇  int zhuang1;//被撞的团簇是否再被撞  LinkStack s,moves;  moves=(Stack\_LinkList)malloc(sizeof(Stack\_Lnode));  s=(Stack\_LinkList)malloc(sizeof(Stack\_Lnode));  InitStack(&s);  InitStack(&moves);  push(&s,\*splitp);  while(ListLength(s)!=0)  {  splitp\_sign=0;  zhuang=0;  zhuang1=0;  success=1;//分裂成功的标志变量  GetTop(s,&splitp1);  pop(&s,&lp);  Node p=lp->head;  if(k==1)  {  while(p)  {  if(p->z-1>=0)  {  if(p->sign==totalsign)  {  if(p->z<=i)  {  if(View\_1(L,splitp1,Cube,p,&s,k,2,lp->sign)==1)//与其他团簇相撞  {  zhuang=1;//相撞标记为1  success=0;//分裂失败  break;//退出循环  }  else if(View\_1(L,splitp1,Cube,p,&s,k,2,lp->sign)==-1)  {  return -1;  }  else  {  p=p->next;  }  }  else  {  p=p->next;  }  }  else  {  splitp\_sign=1;//此次移动的是被动推动的团簇  if(View\_1(L,splitp1,Cube,p,&s,k,2,lp->sign)==1)//与其他团簇相撞  {  zhuang1=1;//相撞标记为1  success=0;//分裂失败  break;//退出循环  }  else if(View\_1(L,splitp1,Cube,p,&s,k,2,lp->sign)==-1)  {  return -1;  }  else  {  p=p->next;  }  }  }  else  {  while(ListLength(moves)!=0)  {  pop(&moves,&lp);  Node p1=lp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=0;  p1->z+=1;  p1=p1->next;  }  p1=lp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=lp->sign;  }  }  ClearStack(&s);ClearStack(&moves);  return 0;  }  }  if(!zhuang)  {  if(lp->sign==totalsign)  {  Node p1=lp->head;  while(p1)  {  if(p1->z<=i&&p1->z-1>=0)  {  Cube[p1->x][p1->y][p1->z]=0; ///////20170605 23:22 分裂一部分发生碰撞数组的值问题  p1->z-=1;  p1=p1->next;  }  else  {  p1=p1->next;  }  }  }  }  if(!zhuang1&&splitp\_sign)  {  Node p1=lp->head;  while(p1)  {  if(p1->z-1>=0)  {  Cube[p1->x][p1->y][p1->z]=0;  p1->z-=1;  p1=p1->next;  }  else  {  p1=p1->next;  }  }  push(&moves,lp);  }  }  else if(k==2)  {  while(p)  {  if(p->y-1>=0)  {  if(p->sign==totalsign)  {  if(p->y<=i)  {  if(View\_1(L,splitp1,Cube,p,&s,k,2,lp->sign)==1)//与其他团簇相撞  {  zhuang=1;//相撞标记为1  success=0;//分裂失败  break;//退出循环  }  else if(View\_1(L,splitp1,Cube,p,&s,k,2,lp->sign)==-1)  {  return -1;  }  else  {  p=p->next;  }  }  else  {  p=p->next;  }  }  else  {  splitp\_sign=1;  if(View\_1(L,splitp1,Cube,p,&s,k,2,lp->sign)==1)//与其他团簇相撞  {  zhuang1=1;//相撞标记为1  success=0;//分裂失败  break;//退出循环  }  else if(View\_1(L,splitp1,Cube,p,&s,k,2,lp->sign)==-1)  {  return -1;  }  else  {  p=p->next;  }  }  }  else  {  while(ListLength(moves)!=0)  {  pop(&moves,&lp);  Node p1=lp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=0;  p1->y+=1;  p1=p1->next;  }  p1=lp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=lp->sign;  }  }  ClearStack(&s);ClearStack(&moves);  return 0;  }  }  if(!zhuang)  {  if(lp->sign==totalsign)  {  Node p1=lp->head;  while(p1)  {  if(p1->y<=i&&p1->y-1>=0)  {  Cube[p1->x][p1->y][p1->z]=0; ///////20170605 23:22 分裂一部分发生碰撞数组的值问题  p1->y-=1;  p1=p1->next;  }  else  {  p1=p1->next;  }  }  }  }  if(!zhuang1&&splitp\_sign)  {  Node p1=lp->head;  while(p1)  {  if(p1->y-1>=0)  {  Cube[p1->x][p1->y][p1->z]=0;  p1->y-=1;  p1=p1->next;  }  else  {  p1=p1->next;  }  }  push(&moves,lp);  }  }  else  {  while(p)  {  if(p->x-1>=0)  {  if(p->sign==totalsign)  {  if(p->x<=i)  {  if(View\_1(L,splitp1,Cube,p,&s,k,2,lp->sign)==1)//与其他团簇相撞  {  zhuang=1;//相撞标记为1  success=0;//分裂失败  break;//退出循环  }  else if(View\_1(L,splitp1,Cube,p,&s,k,2,lp->sign)==-1)  {  return -1;  }  else  {  p=p->next;  }  }  else  {  p=p->next;  }  }  else  {  splitp\_sign=1;  if(View\_1(L,splitp1,Cube,p,&s,k,2,lp->sign)==1)//与其他团簇相撞  {  zhuang1=1;//相撞标记为1  success=0;//分裂失败  break;//退出循环  }  else if(View\_1(L,splitp1,Cube,p,&s,k,2,lp->sign)==-1)  {  return -1;  }  else  {  p=p->next;  }  }  }  else  {  while(ListLength(moves)!=0)  {  pop(&moves,&lp);  Node p1=lp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=0;  p1->x+=1;  p1=p1->next;  }  p1=lp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=lp->sign;  }  }  ClearStack(&s);ClearStack(&moves);  return 0;  }  }  if(!zhuang)  {  if(lp->sign==totalsign)  {  Node p1=lp->head;  while(p1)  {  if(p1->x<=i&&p1->x-1>=0)  {  Cube[p1->x][p1->y][p1->z]=0; ///////20170605 23:22 分裂一部分发生碰撞数组的值问题  p1->x-=1;  p1=p1->next;  }  else  {  p1=p1->next;  }  }  }  }  if(!zhuang1&&splitp\_sign)  {  Node p1=lp->head;  while(p1)  {  if(p1->x-1>=0)  {  Cube[p1->x][p1->y][p1->z]=0;  p1->x-=1;  p1=p1->next;  }  else  {  p1=p1->next;  }  }  push(&moves,lp);  }  }  if(success)  {  Node p1=lp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=lp->sign;  p1=p1->next;  }  }  }  return success;  }  int split(LinkList \*L,LinkList \*splitp,cubex Cube,int i,int k) //分裂函数之分裂 （碰撞就平移）需要用到栈  {  LinkList splitp1,lp,\*lp2;  int success;//分裂成功的标志  int zhuang;//是否碰撞  int nextsign;  int splitp\_sign;//是否为被碰撞的团簇  int zhuang1;//被撞的团簇是否再被撞  int totalsign=(\*splitp)->sign;  lp=(LinkList)malloc(sizeof(Lgroup));  lp2=(LinkList\*)malloc(sizeof(Lgroup));  splitp1=(LinkList)malloc(sizeof(Lgroup));  LinkStack s,moves;  moves=(Stack\_LinkList)malloc(sizeof(Stack\_Lnode));  s=(Stack\_LinkList)malloc(sizeof(Stack\_Lnode));  InitStack(&s);  InitStack(&moves);  push(&s,\*splitp);  while(ListLength(s)!=0)  {  splitp\_sign=0;  zhuang=0;  zhuang1=0;  success=1;//分裂成功的标志变量  pop(&s,&lp);  if(ListLength(s)!=0)  {  GetTop(s,lp2);  nextsign=(\*lp2)->sign;  }  else  {  nextsign=2163;  }  splitp1=lp;  Node p=lp->head;  if(k==1) //移动点之前不用考虑点四周是否有点，因为团簇已经备统计过，只需计算移动后的情况  {  while(p) //预先分裂处理 判断是否相撞  {  if(p->z+1<=99)  {  if(p->sign==totalsign)  {  if(p->z>i)  {  if(View\_1(L,splitp1,Cube,p,&s,k,1,nextsign)==1)//与其他团簇相撞  {  zhuang=1;//相撞标记为1  success=0;//分裂失败  break;//退出循环  }  else if(View\_1(L,splitp1,Cube,p,&s,k,1,nextsign)==-1)  {  return -1;  }  else  {  p=p->next;  }  }  else  {  p=p->next;  }  }  else  {  splitp\_sign=1;  if(View\_1(L,splitp1,Cube,p,&s,k,1,nextsign)==1)//与其他团簇相撞  {  zhuang1=1;//相撞标记为1  success=0;//分裂失败  break;//退出循环  }  else if(View\_1(L,splitp1,Cube,p,&s,k,1,nextsign)==-1)  {  return -1;  }  else  {  p=p->next;  }  }  }  else  {  while(ListLength(moves)!=0)  {  pop(&moves,&lp);  Node p1=lp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=0;  p1->z-=1;  p1=p1->next;  }  p1=lp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=lp->sign;  p1=p1->next;  }  }  ClearStack(&s);ClearStack(&moves);  return 0;  }  }  if(!zhuang)  {  if(lp->sign==totalsign)  {  Node p1=lp->head;  while(p1)  {  if(p1->z>i&&p1->z+1<=l-1)  {  Cube[p1->x][p1->y][p1->z]=0; ///////20170605 23:22 分裂一部分发生碰撞数组的值问题  p1->z+=1;  p1=p1->next;  }  else  {  p1=p1->next;  }  }  }  }  if(!zhuang1&&splitp\_sign)  {  Node p1=lp->head;  while(p1)  {  if(p1->z+1<=l-1)  {  Cube[p1->x][p1->y][p1->z]=0;  p1->z+=1;  p1=p1->next;  }  else  {  p1=p1->next;  }  }  push(&moves,lp);  }  }  else if(k==2)  {  while(p)  {  if(p->y+1<=99)  {  if(p->sign==totalsign)  {  if(p->y>i)  {  if(View\_1(L,splitp1,Cube,p,&s,k,1,nextsign)==1)//与其他团簇相撞  {  zhuang=1;//相撞标记为1  success=0;//分裂失败  break;//退出循环  }  else if(View\_1(L,splitp1,Cube,p,&s,k,1,nextsign)==-1)  {  return -1;  }  else  {  p=p->next;  }  }  else  {  p=p->next;  }  }  else  {  splitp\_sign=1;  if(View\_1(L,splitp1,Cube,p,&s,k,1,nextsign)==1)//与其他团簇相撞  {  zhuang1=1;//相撞标记为1  success=0;//分裂失败  break;//退出循环  }  else if(View\_1(L,splitp1,Cube,p,&s,k,1,nextsign)==-1)  {  return -1;  }  else  {  p=p->next;  }  }  }  else  {  while(ListLength(moves)!=0)  {  pop(&moves,&lp);  Node p1=lp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=0;  p1->y-=1;  p1=p1->next;  }  p1=lp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=lp->sign;  p1=p1->next;  }  }  ClearStack(&s);ClearStack(&moves);  return 0;  }  }  if(!zhuang)  {  if(lp->sign==totalsign)  {  Node p1=lp->head;  while(p1)  {  if(p1->y>i&&p1->y+1<=l-1)  {  Cube[p1->x][p1->y][p1->z]=0; ///////20170605 23:22 分裂一部分发生碰撞数组的值问题  p1->y+=1;  p1=p1->next;  }  else  {  p1=p1->next;  }  }  }  }  if(!zhuang1&&splitp\_sign)  {  Node p1=lp->head;  while(p1)  {  if(p1->y+1<=l-1)  {  Cube[p1->x][p1->y][p1->z]=0;  p1->y+=1;  p1=p1->next;  }  else  {  p1=p1->next;  }  }  push(&moves,lp);  }  }  else  {  while(p)  {  if(p->x+1<=99)  {  if(p->sign==totalsign)  {  if(p->x>i)  {  if(View\_1(L,splitp1,Cube,p,&s,k,1,nextsign)==1)//与其他团簇相撞  {  zhuang=1;//相撞标记为1  success=0;//分裂失败  break;//退出循环  }  else if(View\_1(L,splitp1,Cube,p,&s,k,1,nextsign)==-1)  {  return -1;  }  else  {  p=p->next;  }  }  else  {  p=p->next;  }  }  else  {  splitp\_sign=1;  if(View\_1(L,splitp1,Cube,p,&s,k,1,nextsign)==1)//与其他团簇相撞  {  zhuang1=1;//相撞标记为1  success=0;//分裂失败  break;//退出循环  }  else if(View\_1(L,splitp1,Cube,p,&s,k,1,nextsign)==-1)  {  return -1;  }  else  {  p=p->next;  }  }  }  else  {  while(ListLength(moves)!=0)  {  pop(&moves,&lp);  Node p1=lp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=0;  p1->x-=1;  p1=p1->next;  }  p1=lp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=lp->sign;  p1=p1->next;  }  }  ClearStack(&s);ClearStack(&moves);  return 0;  }  }  if(!zhuang)  {  if(lp->sign==totalsign)  {  Node p1=lp->head;  while(p1)  {  if(p1->x>i&&p1->x+1<=l-1)  {  Cube[p1->x][p1->y][p1->z]=0; ///////20170605 23:22 分裂一部分发生碰撞数组的值问题  p1->x+=1;  p1=p1->next;  }  else  {  p1=p1->next;  }  }  }  }  if(!zhuang1&&splitp\_sign)  {  Node p1=lp->head;  while(p1)  {  if(p1->x+1<=l-1)  {  Cube[p1->x][p1->y][p1->z]=0;  p1->x+=1;  p1=p1->next;  }  else  {  p1=p1->next;  }  }  push(&moves,lp);  }  }  if(success)  {  Node p1=lp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=lp->sign;  p1=p1->next;  }  }  }  return success;  }  int count\_group\_by\_list\_2(LinkList \*up,cubex Cube)//不考虑后续团簇  {  int n1;int count;  Node p,p1;LinkList lp1,ulp;  p=(\*up)->head;  while(p)  {  Cube[p->x][p->y][p->z]=noview;  p=p->next;  }  n1=1;  (\*up)->next=NULL;  p=(\*up)->head;  while(p)  {  if(Cube[p->x][p->y][p->z]==noview)  {  count=0;  lp1=(LinkList)malloc(sizeof(Lgroup));  lp1->sign=n1;lp1->next=NULL;  p1=(Node)malloc(sizeof(Lnode));  p1->x=p->x;p1->y=p->y;p1->z=p->z;p1->sign=lp1->sign;  p1->next=NULL;  lp1->head=p1;  View(Cube,p->x,p->y,p->z,&lp1);//遍历完所有相关结点  p1=lp1->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=n1;  count++;  p1=p1->next;  }  lp1->count=count;  minmax(&lp1);  lp1->next=(\*up)->next;  (\*up)->next=lp1;  n1++;  }  p=p->next;  }//找出了分裂后形成的所有新团簇 第二步  return n1-1;  }  int unionlist(LinkList \*up,cubex Cube,int i,int k) //分裂函数之合并 （碰撞就汇聚）  {  int n=1;int groupNum;  int ucount;  int init\_sign=(\*up)->sign; //第一步  int dots;Node p; //用于存储最大团簇结点数  LinkList lp;  LinkList ulp,ulp1;  ulp1=(LinkList)malloc(sizeof(Lgroup));  int x,y,z,x1,y1,z1;  groupNum=count\_group\_by\_list\_2(up,Cube);  if(groupNum==2)  {  return 1;  }  dots=0;  int lq=0;  ulp=(\*up)->next;//up后跟的团簇变了  while(ulp)  {  if(k==1)  {  if(ulp->Zmin>i)  {  if(ulp->count>dots)//找出最大团簇  {  dots=ulp->count;  lq=ulp->sign;  }  }  }  else if(k==2)  {  if(ulp->Ymin>i)  {  if(ulp->count>dots)//找出最大团簇  {  dots=ulp->count;  lq=ulp->sign;  }  }  }  else  {  if(ulp->Xmin>i)  {  if(ulp->count>dots)//找出最大团簇  {  dots=ulp->count;  lq=ulp->sign;  }  }  }  ulp=ulp->next;  }  ulp=(\*up)->next;  while(ulp)  {  if(k==1)  {  if(ulp->Zmin>i)  {  if(ulp->sign!=lq)//不是最大团簇  {  p=ulp->head;  while(p)  {  Cube[p->x][p->y][p->z]=0;//合并  (p->z)-=1;  p=p->next;  }  }  }  }  if(k==2)  {  if(ulp->Ymin>i)  {  if(ulp->sign!=lq)//不是最大团簇  {  p=ulp->head;  while(p)  {  Cube[p->x][p->y][p->z]=0;//合并  (p->y)-=1;  p=p->next;  }  }  }  }  if(k==3)  {  if(ulp->Xmin>i)  {  if(ulp->sign!=lq)//不是最大团簇  {  p=ulp->head;  while(p)  {  Cube[p->x][p->y][p->z]=0;//合并  //if(View\_2(p,k,2,Cube));  (p->x)-=1;  p=p->next;  }  }  }  }  ulp=ulp->next;  }//第三步  ulp=(\*up)->next;  int t=0;  while(ulp)  {  Node p1=ulp->head;  while(p1)  {  Cube[p1->x][p1->y][p1->z]=noview;  p1=p1->next;  t++;  }  ulp=ulp->next;  }  (\*up)->next=NULL;  countgroup(Cube,up);  lq=0;// 第六步  dots=0;  ulp=(\*up)->next;  while(ulp)  {  if(k==1)  {  if(ulp->Zmin<=i)  {  if(ulp->count>dots)//找出最大团簇  {  dots=ulp->count;  lq=ulp->sign;  }  }  }  if(k==2)  {  if(ulp->Ymin<=i)  {  if(ulp->count>dots)//找出最大团簇  {  dots=ulp->count;  lq=ulp->sign;  }  }  }  if(k==3)  {  if(ulp->Xmin<=i)  {  if(ulp->count>dots)//找出最大团簇  {  dots=ulp->count;  lq=ulp->sign;  }  }  }  ulp=ulp->next;  }  ulp=(\*up)->next;//第一次分裂合并 大团簇不动 小团簇向相反方向移动一个单元格  while(ulp)  {  if(k==1)  {  if(ulp->Zmin<=i)  {  if(ulp->sign!=lq)//不是最大团簇  {  p=ulp->head;  while(p!=NULL)  {  Cube[p->x][p->y][p->z]=0;  (p->z)+=1;//合并  p=p->next;  }  }  }  }  if(k==2)  {  if(ulp->Ymin<=i)  {  if(ulp->sign!=lq)//不是最大团簇  {  p=ulp->head;  while(p)  {  Cube[p->x][p->y][p->z]=0;  (p->y)+=1;//合并  p=p->next;  }  }  }  }  if(k==3)  {  if(ulp->Xmin<=i)  {  if(ulp->sign!=lq)//不是最大团簇  {  p=ulp->head;  while(p)  {  Cube[p->x][p->y][p->z]=0;  (p->x)+=1;//合并  p=p->next;  }  }  }  }  ulp=ulp->next;  }  LinkList ulp2;  ulp2=(\*up)->next;  while(ulp2)  {  Node p2=ulp2->head;  while(p2)  {  Cube[p2->x][p2->y][p2->z]=noview;  p2=p2->next;  }  ulp2=ulp2->next;  }  (\*up)->next=NULL;  for(x=0;x<l;x++)  {  for(y=0;y<l;y++)  {  for(z=0;z<l;z++)  {  if(Cube[x][y][z]==noview)//未访问，建新团簇  {  ucount=0;  lp=(LinkList)malloc(sizeof(Lgroup));  lp->sign=n;lp->next=NULL;  p=(Node)malloc(sizeof(Lnode));  p->x=x;p->y=y;p->z=z;p->sign=n;  p->next=NULL;  lp->head=p;  View(Cube,x,y,z,&lp);//遍历完所有相关结点  for(x1=0;x1<l;x1++)  {  for(y1=0;y1<l;y1++)  {  for(z1=0;z1<l;z1++)  {  if(Cube[x1][y1][z1]==view)  {  Cube[x1][y1][z1]=n;  ucount++;  }  }  }  }  minmax(&lp);  lp->count=ucount;  lp->next=(\*up)->next;  (\*up)->next=lp;  n++;  }  }  }  }  if(n-1!=2)  {  initcube\_2(Cube);  ReadFile(Cube,"01.txt");  }  return 1;  }  int unionlist\_2(LinkList \*up,cubex Cube,int i,int k) //分裂函数之合并 （碰撞就汇聚）  {  int ucount,groupNum;  int init\_sign=(\*up)->sign; //第一步  (\*up)->next=NULL;  int dots=0; //用于存储最大团簇结点数  int n=1;  LinkList lp;Node p;  int x,y,z,x1,y1,z1;  LinkList ulp1=(\*up);  groupNum=count\_group\_by\_list\_2(up,Cube);  if(groupNum==2)  {  return 1;  }  LinkList ulp;  int lq=0;  ulp=(LinkList)malloc(sizeof(Lgroup));  ulp=(\*up)->next; //up后跟的团簇变了  while(ulp!=NULL)  {  if(k==1)  {  if(ulp->Zmax<i)  {  if(ulp->count>dots)//找出最大团簇  {  dots=ulp->count;  lq=ulp->sign;  }  }  }  else if(k==2)  {  if(ulp->Ymax<i)  {  if(ulp->count>dots)//找出最大团簇  {  dots=ulp->count;  lq=ulp->sign;  }  }  }  else  {  if(ulp->Xmax<i)  {  if(ulp->count>dots)//找出最大团簇  {  dots=ulp->count;  lq=ulp->sign;  }  }  }  ulp=ulp->next;  }  ulp=(\*up)->next;  while(ulp!=NULL)  {  if(k==1)  {  if(ulp->Zmax<i)  {  if(ulp->sign!=lq)//不是最大团簇  {  Node p=ulp->head;  while(p!=NULL)  {  Cube[p->x][p->y][p->z]=0;//合并  (p->z)+=1;  p=p->next;  }  }  }  }  if(k==2)  {  if(ulp->Ymax<i)  {  if(ulp->sign!=lq)//不是最大团簇  {  Node p=ulp->head;  while(p!=NULL)  {  Cube[p->x][p->y][p->z]=0;//合并  (p->y)+=1;  p=p->next;  }  }  }  }  if(k==3)  {  if(ulp->Xmax<i)  {  if(ulp->sign!=lq)//不是最大团簇  {  Node p=ulp->head;  while(p!=NULL)  {  Cube[p->x][p->y][p->z]=0;//合并  (p->x)+=1;  p=p->next;  }  }  }  }  ulp=ulp->next;  } //第三步  ulp=(\*up)->next;  while(ulp!=NULL)  {  Node p1=ulp->head;  while(p1!=NULL)  {  Cube[p1->x][p1->y][p1->z]=noview;  p1=p1->next;  }  ulp=ulp->next;  } //第四步  (\*up)->next=NULL;  countgroup(Cube,up);  lq=0;// 第六步  dots=0;  ulp=(\*up)->next;  while(ulp!=NULL)  {  if(k==1)  {  if(ulp->Zmax>=i)  {  if(ulp->count>dots)//找出最大团簇  {  dots=ulp->count;  lq=ulp->sign;  }  }  }  if(k==2)  {  if(ulp->Ymax>=i)  {  if(ulp->count>dots)//找出最大团簇  {  dots=ulp->count;  lq=ulp->sign;  }  }  }  if(k==3)  {  if(ulp->Xmax>=i)  {  if(ulp->count>dots)//找出最大团簇  {  dots=ulp->count;  lq=ulp->sign;  }  }  }  ulp=ulp->next;  }  ulp=(\*up)->next;//第一次分裂合并 大团簇不动 小团簇向相反方向移动一个单元格  while(ulp!=NULL)  {  if(k==1)  {  if(ulp->Zmax>=i)  {  if(ulp->sign!=lq)//不是最大团簇  {  Node p=ulp->head;  while(p!=NULL)  {  Cube[p->x][p->y][p->z]=0;  (p->z)-=1;//合并  p=p->next;  }  }  }  }  if(k==2)  {  if(ulp->Ymax>=i)  {  if(ulp->sign!=lq)//不是最大团簇  {  Node p2;  p2=ulp->head;  while(p2)  {  Cube[p2->x][p2->y][p2->z]=0;  (p2->y)-=1;//合并  p2=p2->next;  }  }  }  }  if(k==3)  {  if(ulp->Xmax>=i)  {  if(ulp->sign!=lq)//不是最大团簇  {  Node p3=ulp->head;  while(p3!=NULL)  {  Cube[p3->x][p3->y][p3->z]=0;  (p3->x)-=1;//合并  p3=p3->next;  }  }  }  }  ulp=ulp->next;  }  ulp=(\*up)->next;  while(ulp!=NULL)  {  Node p1=ulp->head;  while(p1!=NULL)  {  Cube[p1->x][p1->y][p1->z]=noview;  p1=p1->next;  }  ulp=ulp->next;  }  (\*up)->next=NULL;  n=1;  for(x=0;x<l;x++)  {  for(y=0;y<l;y++)  {  for(z=0;z<l;z++)  {  if(Cube[x][y][z]==noview)//未访问，建新团簇  {  Node p;  ucount=0;  lp=(LinkList)malloc(sizeof(Lgroup));  lp->sign=n;  p=(Node)malloc(sizeof(Lnode));  p->x=x;p->y=y;p->z=z;p->sign=n;  p->next=NULL;  lp->head=p;  View(Cube,x,y,z,&lp);//遍历完所有相关结点  for(x1=0;x1<l;x1++)  {  for(y1=0;y1<l;y1++)  {  for(z1=0;z1<l;z1++)  {  if(Cube[x1][y1][z1]==view)  {  Cube[x1][y1][z1]=n;  ucount++;  }  }  }  }  minmax(&lp);  lp->count=ucount;  lp->next=(\*up)->next;  (\*up)->next=lp;  n++;  }  }  }  }//  if(n-1!=2)  {  initcube\_2(Cube);  ReadFile(Cube,"01.txt");  }  return 1;  }  int sub\_divide(LinkList \*L,LinkList \*p,cubex Cube) //子分裂函数  {  int i,k;  int \*k1=&k;  k=random();//k表示分裂方向  i=locate(\*p,k,k1);  k=(\*k1);  if(split(L,p,Cube,i,k))  {  if(unionlist(p,Cube,i,k))  {  return 1;  }  else  {  return 0;  }  }  else if(split(L,p,Cube,i,k)==-1)  {  return 0;  }  else if(split\_2(L,p,Cube,i,k))  {  if(unionlist\_2(p,Cube,i,k))  {  return 1;  }  else  {  return 0;  }  }  else if(split\_2(L,p,Cube,i,k)==-1)  {  return 0;  }  else  {  return 0;  }  }  void divide(cubex cube,LinkList \*L) //总分裂函数  {  LinkList p;  p=(LinkList)malloc(sizeof(Lgroup));  while((\*L)->size<=2160)  {  p=choose(\*L);//选择可分裂的团簇（以编号随机选）；  if(sub\_divide(L,&p,cube))  {  write\_file1(cube);  initcube\_1(cube);  (\*L)->next=NULL;  system("cls");  printf("团簇数量:%d\n",(\*L)->size);  countgroup(cube,L);  }  else  {  continue;  }  }  printf("分裂完成!\n");  countgroup(cube,L);  }  int main()  {  LinkList L;  char \*filename="break\_init\_1.txt";//"01.txt";//  cubex Cube;  Cube=initcube(l,l,l);  ReadFile(Cube,filename);  L=(LinkList)malloc(sizeof(Lgroup));  L->next=NULL;  countgroup(Cube,&L);  divide(Cube,&L);  system("pause");  } |
| **关键词：** |
| **参考书目：** |
| **成绩评定：**  **指导教师（签字）：**  **年 月 日** |