网络异常检测方法

一、数据结构

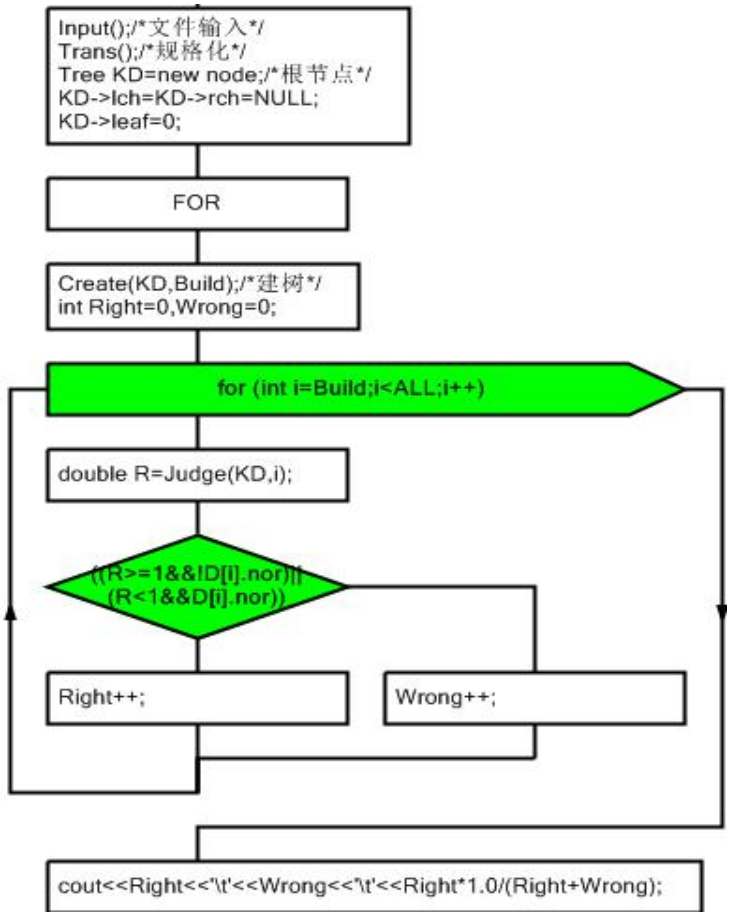
```
struct node/*节点*/
{     struct node *lch,*rch;/*子节点*/
       bool leaf;/*叶节点标志*/
       int dimen,mid;/*分割维和分割中点标号*/
       vector<int> index;/*数据标号*/};
typedef struct node* Tree;/*树的指针*/

struct data
{     double flow[10];/*流量属性集*/
       bool nor;/*正常 OR 攻击标志*/};

data D[ALL];/*数据集*/
```

二、主函数流程图

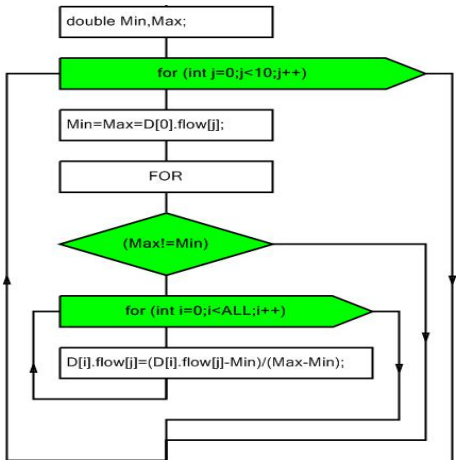
1.读入文件，筛选出流量属性集
2.读完之后对数据进行规格化处理
3.根据部分数据建树，测试剩余数据，并输出检测的正确率



三、规格化处理：void Trans()

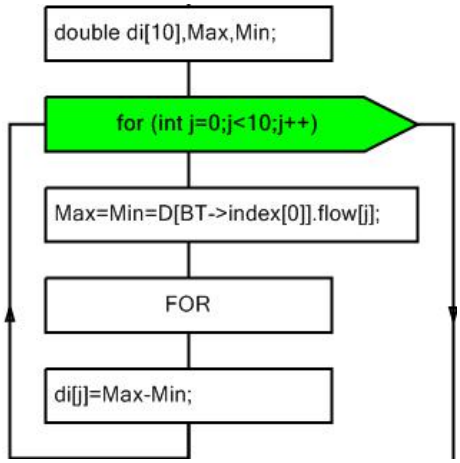
1.选出数据集某一属性的最大值 Max 和最小值 Min

2.如果 Max!=Min，根据最大值和最小值，将所有数据规格化到[0,1]

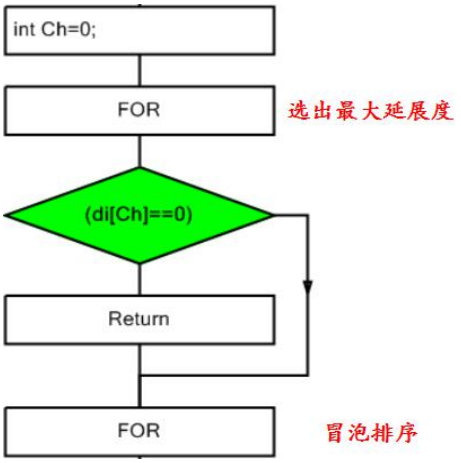


四、建 KD 树：void Create(Tree BT,int number)

1.选出数据集所有属性的最大值和最小值，算出延展度



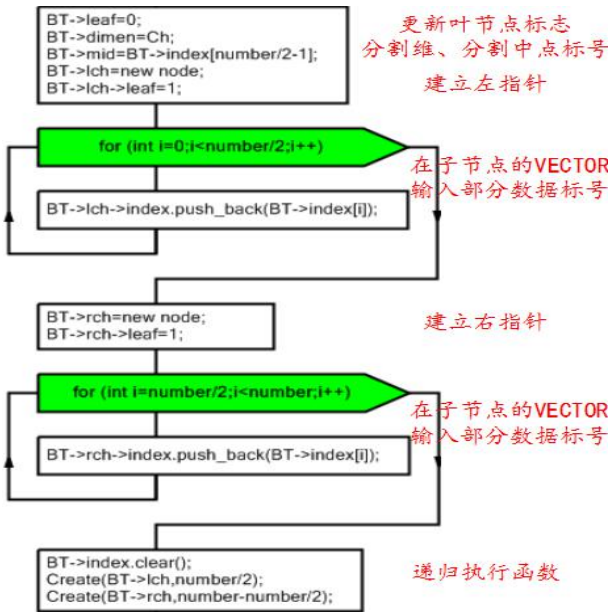
2.选出最大延展度，如果不为 0，则按照当前分割维对数据标号进行升序排序



3.更新当前节点的叶节点标志、分割维和分割中点标号

4.建立左右子节点，并在子节点的 VECTOR 中加入数据标号

5.清空当前节点的 VECTOR，递归执行函数

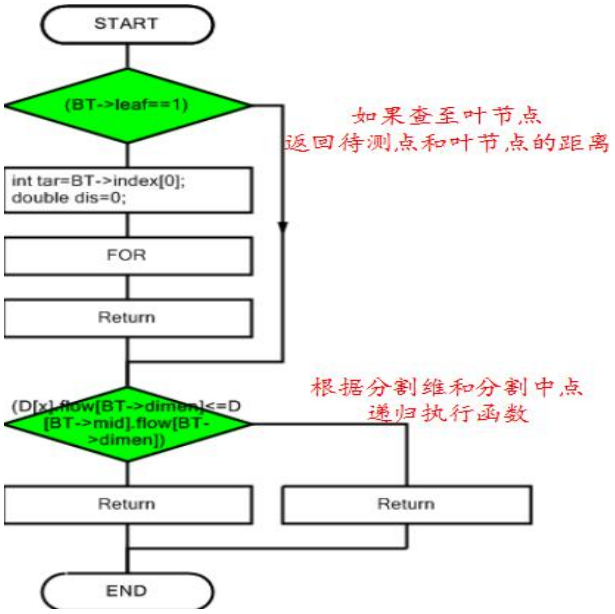


6.KD 树的高度不超过 $Log_2(n)$；叶节点的 VECTOR 中存储的数据点流量属性集相同

五、检测：double Judge(Tree BT,int x)

1.若当前点是叶节点，则返回待测点和叶节点的欧氏距离

2.若当前点不是叶节点，则根据分割维和分割中点，递归执行函数



六、实验结果

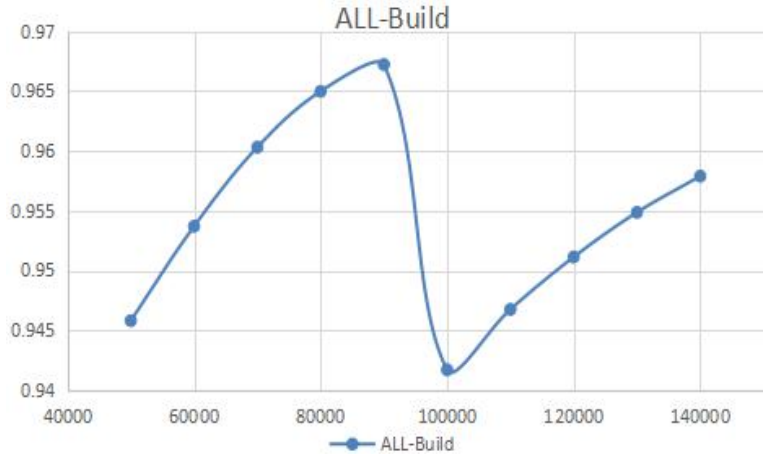
1.读入"KDDCUP.data_10_percent"前 ALL=200000 条，其中前 Build 条用于建 KD 树

2.测试程序，输出正确数、误报数、正确率

Build	10000	20000	30000	40000	50000
正确率	0.9657	0.80585	0.962588	0.963531	0.960567
Build	60000	70000	80000	90000	100000
正确率	0.969564	0.967192	0.966392	0.875955	0.76299

3.Build=50000 时，通过修改 ALL，测试程序，输出正确数、误报数、正确率

ALL-Build	50000	60000	70000	80000	90000
正确率	0.9458	0.9537	0.960314	0.964988	0.967233
ALL-Build	100000	110000	120000	130000	140000
正确率	0.94169	0.946745	0.951125	0.954862	0.957893



七、代码

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<iostream>
#include<algorithm>
#include<vector>
#include<new>
#define ALL 20000
#define Build 10000
using namespace std;

struct node
{     struct node *lch,*rch;/*子节点*/
       bool leaf;/*叶节点标志*/
       int dimen;/*分割维*/
       int mid;/*分割中点标号*/
       vector<int> index;/*数据标号*/};
typedef struct node* Tree;/*树的指针*/

struct data
{     double flow[10];/*流量属性集*/
       bool nor;/*正常标志*/};

data D[ALL];/*数据集*/

void Input()/*文件输入*/
{     FILE *F=fopen("KDDCUP.data_10_percent","r");
       if(!F)
       {     return;}
       string in[42],s;
       char ch;
       for(int i=0;i<ALL;i++)/*D 下标*/
       {     ch=fgetc(F);
              int k=0;
              while(1)/*多维点*/
              {     s="";
                     while(ch!=',')/*间隔符*/
                     {     if(ch=='\n')/*读完*/
                            {     in[k]=s;
                                   goto Next;}
                            s+=ch;
```

```
                    ch=fgetc(F);}
                ch=fgetc(F);
                in[k++]=s;}
Next:   for(int j=21;j<31;j++)/*选择流量属性集*/
        {   D[i].flow[j-21]=atof(in[j].c_str());}
        D[i].nor=(in[41]=="normal.")?1:0;}}
```

void Trans()/*规格化*/

```
{   double Min,Max;
    for(int j=0;j<10;j++)
    {   Min=Max=D[0].flow[j];
        for(int i=1;i<ALL;i++)
        {   if(D[i].flow[j]>Max)
            {   Max=D[i].flow[j];}
            if(D[i].flow[j]<Min)
            {   Min=D[i].flow[j];}}
        if(Max!=Min)
        {   for(int i=0;i<ALL;i++)
            {   D[i].flow[j]=(D[i].flow[j]-Min)/(Max-Min);}}}}
```

void Create(Tree BT,int number)/*建树*/

```
{   double di[10],Max,Min;
    for(int j=0;j<10;j++)
    {   Max=Min=D[BT->index[0]].flow[j];
        for(int i=1;i<number;i++)
        {   if(D[BT->index[i]].flow[j]>Max)
            {   Max=D[BT->index[i]].flow[j];}
            if(D[BT->index[i]].flow[j]<Min)
            {   Min=D[BT->index[i]].flow[j];}}
        di[j]=Max-Min;}
    int Ch=0;/*分割维*/
    for(int i=1;i<10;i++)
    {   if(di[i]>di[Ch])
        {   Ch=i;}}
    if(di[Ch]==0)
    {   return;}
    for(int i=0;i<number-1;i++)/*冒泡排序*/
    {   for(int j=i+1;j<number;j++)
        {   if(D[BT->index[i]].flow[Ch]>D[BT->index[j]].flow[Ch])
            {   int tmp=BT->index[i];
                BT->index[i]=BT->index[j];
                BT->index[j]=tmp;}}}
    BT->leaf=0;
    BT->dimen=Ch;
```

```
    BT->mid=BT->index[number/2-1];
    BT->lch=new node;
    BT->lch->leaf=1;
    for(int i=0;i<number/2;i++)
    {   BT->lch->index.push_back(BT->index[i]);}
    BT->rch=new node;
    BT->rch->leaf=1;
    for(int i=number/2;i<number;i++)
    {   BT->rch->index.push_back(BT->index[i]);}
    BT->index.clear();
    Create(BT->lch,number/2);
    Create(BT->rch,number-number/2);}
```

double Judge(Tree BT,int x)/*判断最近点并返回欧氏距离*/

```
{   if(BT->leaf==1)
    {   int tar=BT->index[0];
        double dis=0;
        for(int i=0;i<10;i++)
        {   dis+=pow(D[x].flow[i]-D[tar].flow[i],2);}
        return sqrt(dis);}
    //cout<<BT->dimen<<'\t'<<BT->mid<<endl;
    if(D[x].flow[BT->dimen]<=D[BT->mid].flow[BT->dimen])
    {   return Judge(BT->lch,x);}
    else
    {   return Judge(BT->rch,x);}}
```

int main()

```
{   Input();/*文件输入*/
    Trans();/*规格化*/
    Tree KD=new node;/*根节点*/
    KD->lch=KD->rch=NULL;
    KD->leaf=0;
    for(int i=0;i<Build;i++)
    {   (KD->index).push_back(i);}
    Create(KD,Build);/*建树*/
    int Right=0,Wrong=0;
    for(int i=Build;i<ALL;i++)
    {   double R=Judge(KD,i);
        if((R>=1&&!D[i].nor)||(R<1&&D[i].nor))
        {   Right++;}
        else
        {   Wrong++;}}
    printf("ALL:%d\tBuild:%d\n",ALL,Build);
    printf("Right:%d\tWrong:%d\t%lf",Right,Wrong,Right*1.0/(Right+Wrong));}
```