emulating RIP emulator 说明文档

目录

1. RIP 仿真器设计简介 2

1.1 前端界面 2

1.2 后端程序 2

2. 图形化界面设计 3

2.1 图形化界面图形 3

2.2 图形化界面简介 3

3. 距离向量算法程序设计 4

3.1 问题分析 4

3.2 设计思路 5

3.3 预处理 5

3.4 路由表更新 9

4. 前后端交互设计 11

4.1 前后端交互简介 11

4.2 前后端交互具体逻辑 12

5. 功能展示 12

5.1 右键操作 12

5.2 报错提示 15

5.3 主题切换 16

6. 开发环境 17

6.1 前端界面开发 17

6.2 后端程序开发 17

1. RIP 仿真器设计简介

该系统的名称是 emulating RIP emulator，意为奋斗向上的 RIP 仿真器，它是 Python 和 C++的融合体：PyQt5 构建前端界面，C++支撑后端程序。奋斗向上，这四个字在于：它是从一个空白页面开始，不断打磨，不断报错，在逆境中，惟有“奋斗向上”四个字铭记在心中，不断成长，最终成为了一个可以正确运行的小程序。 但他的成长之路不止于此，他还会继续奋斗向上，成为一个能够真正能学习使用的 RIP 仿真器程序。

1.1 前端界面

前端界面采用 PyQt5 程序设计和全英文的模式，它的左侧的表格是当前路由器的路由信息，右侧的表格是待更新的路由表的路由信息，在下面是相邻路由表的路由信息，然后点击运行按钮，即可运行。在修改路由信息中，可以通过相应的操作进行增添路由信息，按目的网络进行排序，删除路由信息，初始化路由表信息等，也可以在路由信息中通过右键，进行这些操作。

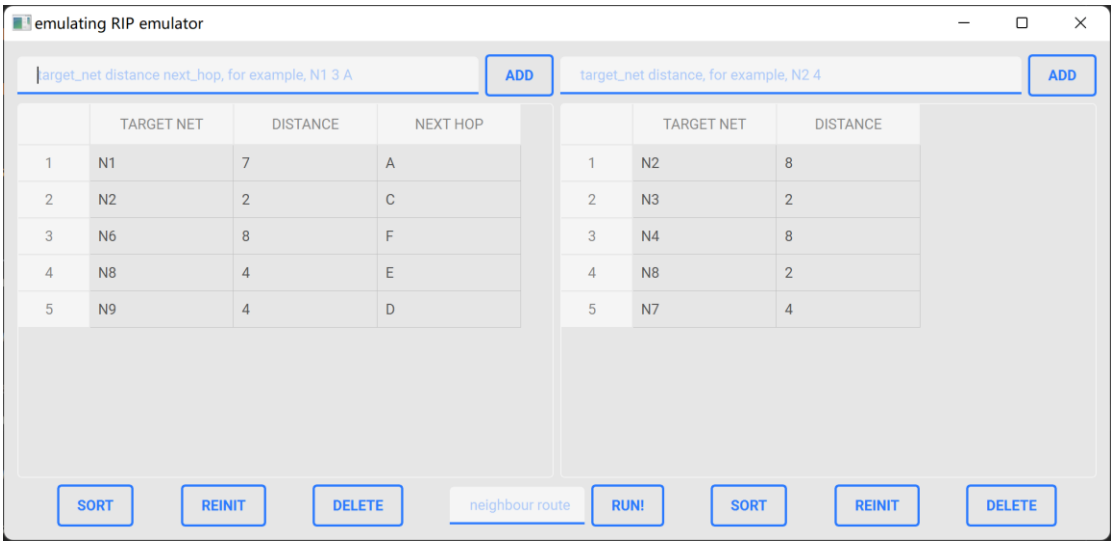
1.2 后端程序

后端程序采用 C++程序设计，因为距离向量算法是基于 Bellman-ford 算法上的，并且 C++拥有丰富的函数库，操作相对于其他语言比较方便，完全适用于 C++。C++程序中在结构体引入了含有目的网络，距离和下一跳，使路由信息封装成一个整体。在更新路由时，采用了 Bellman-ford 算法， 与当前路由器逐一比对，满足要求的路由信息进行更新。

2. 图形化界面设计

2.1 图形化界面图形

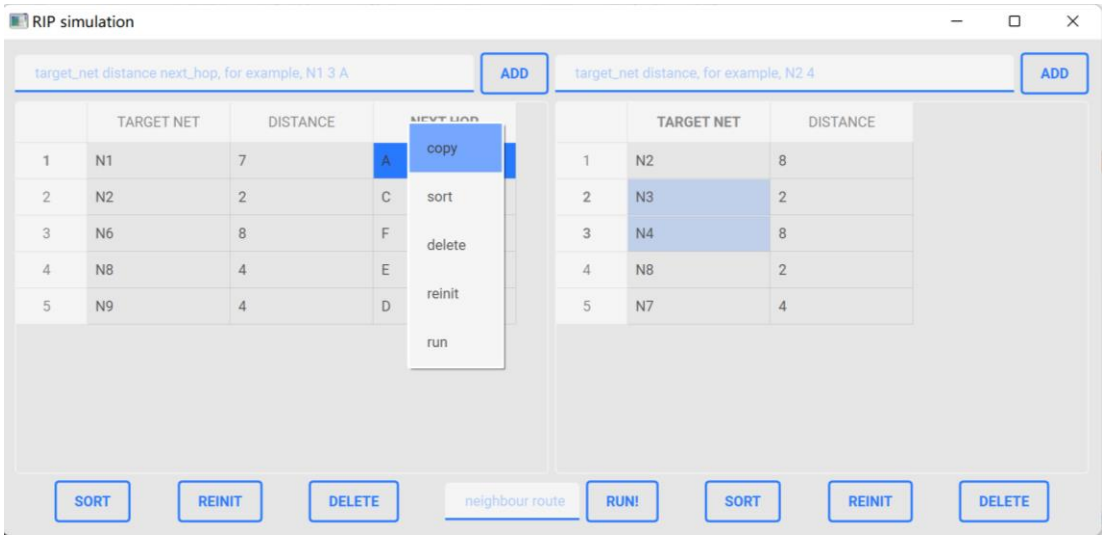
界面采用PyQt5的方法，然后内容采用全英文的方式，设计出界面，如图一所示



图一 界面

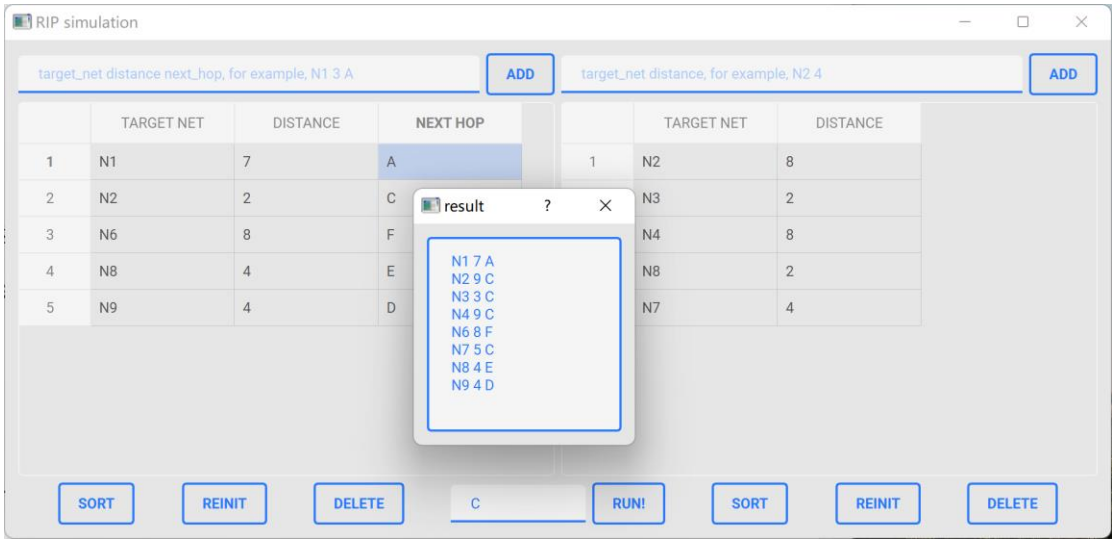
2.2 图形化界面简介

该界面包括当前路由表，相邻路由器以及该路由表信息，同时还设有相关的添加操作，在界面中间的表格部分也可以通过右键，可以进行复制、排序、删除、初始化、运行操作，如图二所示。



图二 右键操作

在界面的底部有排序，初始化和删除的按钮，可以触发点击，响应对应的操作，最关键的，是运行按钮，如果所有信息合法，便能正确运行，并跳出结果弹窗，如图三所示。



图三 运行弹窗

3. 距离向量算法程序设计

3.1 问题分析

路由表更新算法的关键在于如何实现每个路由器路由表的动态更新过程。让每个路由器维护一张路由表，表中给出了到每个目的地已知的最佳距离和路径。通过与相邻路由器之间

周期性地相互交换信息，来更新表中的信息。当网络拓扑结构发生变化时，路由器之间也将及时地相互通知有关变更信息。

3.2 设计思路

基于距离矢量路由算对路由表更新算法的设计与实现。距离向量路由算法要求，每个结点都参与定期交换整个路由表，即把路由表传递给自己与之相连的结点。为了让路由更新的过程便于观察，将每个路由更新的过程分离出来。先建立两张路由表：当前路由表和相邻路由表，然后采用距离向量算法，通过相邻路由表对当前路由表进行更新。对于程序设计采用C++语言，设计一个类，该类中对应两张路由表，即当前路由表与相邻路由表，先输入待更新的当前路由表和相邻路由表的路由信息，再对相邻路由器的跳数进行更新，接着再使用距离向量算法，从而更新当前路由表，最终能够运行实现路由表更新算法的设计。

其中，每个路由信息的形式如下：

< 目的网络 N，跳数，下一跳地址 >

3.3 预处理

设计一个class类，作为路由表RTable，分别输入当前路由器的路由表和相邻路由器的路由表，将下一跳的路由地址改为当前路由器，并且跳数加1。

```
//定义路由表结构 RTable
typedef struct node{
    char dstNet[5];
    int   distance;
    char nextSkip[5];
}RTable;
```

```
RTable RT1[1000];//当前路由器路由表
RTable RT2[1000];//相邻路由器的路由表


int i,l1,l2;
char nearR1[5],nearR2[5];



bool cmp(node a, node b)
{
    return strcmp(a.dstNet, b.dstNet) < 0;
}


//基于当前路由器的路由表初始化函数
void InitRTable( RTable* RT ){
    printf("请为当前路由器添加路由表项(目的网络  距离  下一跳路由器)：\n");
    for( i=0;i<1000;++i ){
        scanf("%s%d%s",RT[i].dstNet,&RT[i].distance,RT[i].nextSkip);

        if( RT[i].distance==0 ){
            break;
        }
    }
```

```
    l1 = i;//记录 RT1[]的长度
}

//相邻路由器添加函数
void AddNearRouter(){
    printf("\n 请输入相邻路由器名称：\n");
    scanf("%s",nearR1);
}

//基于相邻路由器的路由表初始化函数
void InitNearRTable(){
    printf("\n 请为相邻路由器%s 添加路由表项(目的网络  距离  下一跳路由器)：
\n",nearR1);
    for(i=0;i<1000;++i){
        scanf("%s%d%s",RT2[i].dstNet,&RT2[i].distance,RT2[i].nextSkip);
        if(T2[i].distance==0){

            break;
        }
    }
    l2 = i;//记录 RT2[]的长度
}
```

```
//路由信息修改函数
void UpdateNearRTable ( RTable* RT2,char* nearR ){
    int p;
    for( p=0;p<l2;++p ){
    RT2[p].distance = RT2[p].distance + 1;
    strcpy( RT2[p].nextSkip,nearR );
    }
}
```

```
-----------距离向量算法的过程模拟------------

请为当前路由器添加路由表项(目的网络 距离 下一跳路由器):
N1 7 A
N2 2 C
N6 8 F
N8 4 E
N9 4 D
0 0 0

-----------是否有新的路由信息（0为否，1为是）------------

1

请输入相邻路由器名称:
C

请为相邻路由器C添加路由表项(目的网络 距离 下一跳路由器):
N2 8 C
N3 2 C
N4 8 C
N8 2 C
N7 4 C
0 0 0
```

图四 路由表路由信息输入

```
----------当前路由器的路由表---------
    目的网络     距离     下一跳路由器
      N1         7          A
      N2         2          C
      N6         8          F
      N8         4          E
      N9         4          D
-------------------------------------

----------相邻路由器的路由表---------
    目的网络     距离     下一跳路由器
      N2         8          C
      N3         2          C
      N4         8          C
      N8         2          C
      N7         4          C
-------------------------------------

-------修改后的路由信息------
    目的网络     距离     下一跳路由器
      N2         9          C
      N3         3          C
      N4         9          C
      N8         3          C
      N7         5          C
-------------------------------------
```

图五 相邻路由器的路由表更新

3.4 路由表更新

整个程序的关键，就是采用距离向量算法，设计一个路由表更新函数，它运用了Bellman-Ford 算法的思想，通过当前路由器的路由表与相邻路由器路由表的路由信息逐一比对，在以下三种情况下，做出不同的处理方式：

（1）若当前表中没有找到这条路由信息，直接添加至原路由表中；

（2）若当前表中找到这条路由信息，且下一跳路由器正好是这个相邻路由器，直接添加至原路由表中，对其进行更新；

（3）若当前表中找到这条路由信息，但下一跳路由器不是这个相邻路由器，选择跳数少的替换。如果两者跳数一样，则保留原路由表的项。

最后，路由表更新算法得以实现，如图六所示。

```
//路由表更新函数
void UpdateRTable( RTable* RT1,RTable* RT2 ){
    int p,q;//p——RT2[]，q——RT1[]
    for( p=0;p<l2;++p ){
        int finded=0;
        for( q=0;q<l1;++q ){
            if( strcmp( RT2[p].dstNet,RT1[q].dstNet )==0 ){//当前表中找到与发来的表目的网络相同的一条路由信息
                finded = 1;
                if( strcmp( RT1[q].nextSkip,RT2[q].nextSkip )==0 ){//下一跳路由器正好是这个相邻路由器
                    RT1[q].distance = RT2[p].distance;
                }
                else{//下一跳路由器不是这个
                    if( RT2[p].distance<RT1[q].distance ){
                        RT1[q].distance = RT2[p].distance;
                        strcpy( RT1[q].nextSkip,RT2[q].nextSkip );
                    }
                }
            }
        }
```

```
        if( !finded ){//当前表中没有这条路由信息，就加上
            strcpy( RT1[l1].dstNet,RT2[p].dstNet );
            RT1[l1].distance = RT2[p].distance;
            strcpy( RT1[l1].nextSkip,RT2[p].nextSkip );

            ++l1;
        }
    }
}
```

```
-------当前路由器更新后的路由表------
    目的网络     距离     下一跳路由器
      N1         7          A
      N2         9          C
      N3         3          C
      N4         9          C
      N6         8          F
      N7         5          C
      N8         3          C
      N9         4          D
-------------------------------------
```

图六 当前路由器的路由表更新

4. 前后端交互设计

4.1 前后端交互简介

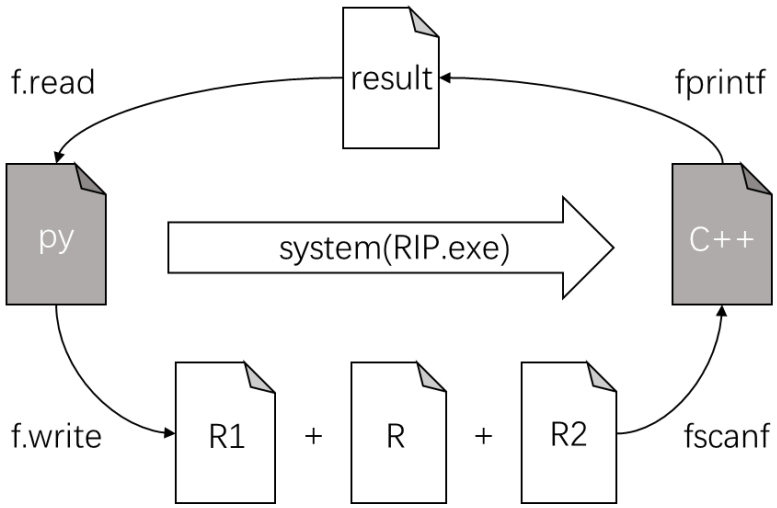
由于前端界面是通过 python 进行设计，后端程序是通过 C++程序进行设计，他们无法在同一个编译环境下进行运行。这时，我们要设计出一个方法，让他们能够交互。

我们可以通过采用 python 的文件读写和以及 C++的文件读写，通过 txt 文件进行双方的输入输出连接。同时，在 python 中采用 system()函数，来调用 C++的 exe 文件。

4.2 前后端交互具体逻辑

它们具体的逻辑如图所示，R1，R，R2 分别表示当前路由表信息，相邻路由器以及其路由信息，这些数据由 python 设计的前端提供，并且通过 f.write 写入对应的 txt 文件中，再用 system(RIP.exe)触发由 C++设计的后端，运行距离向量算法，将 R1，R，R2 通过 fscanf 读入到程序中，运行后通过 fprintf 写入 result 的 txt 文件中，最后，在 python 代码中的 f.read 将距离向量算法运行后的结果加入到图形化界面进行展示。

用户在实现距离向量算法后，会在 C++写入 result 文件的同时，还会写入 record 文件中，通过记录每次用户执行的结果，如图七所示。

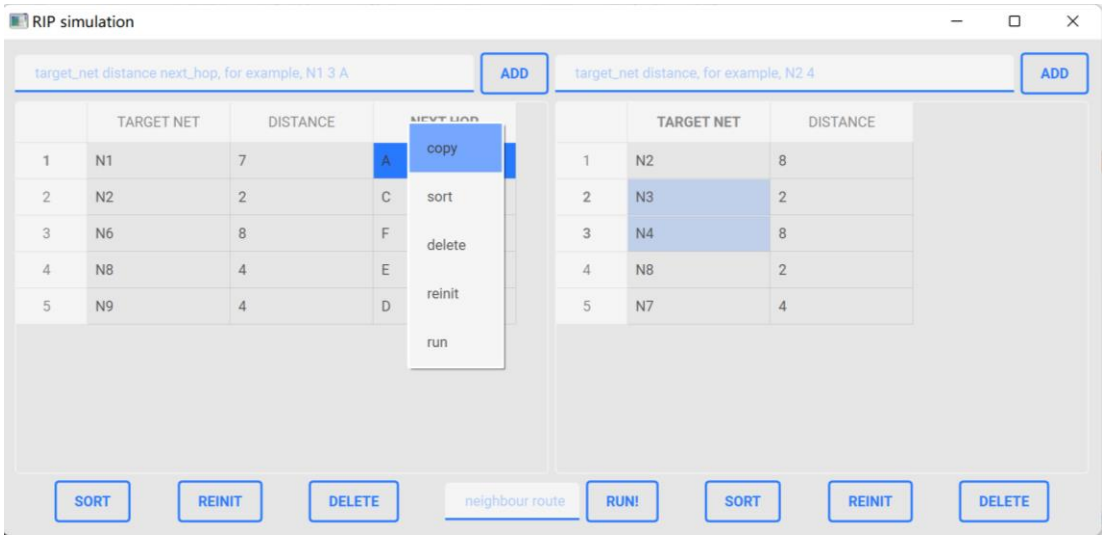


图七 前后端交互

5. 功能展示

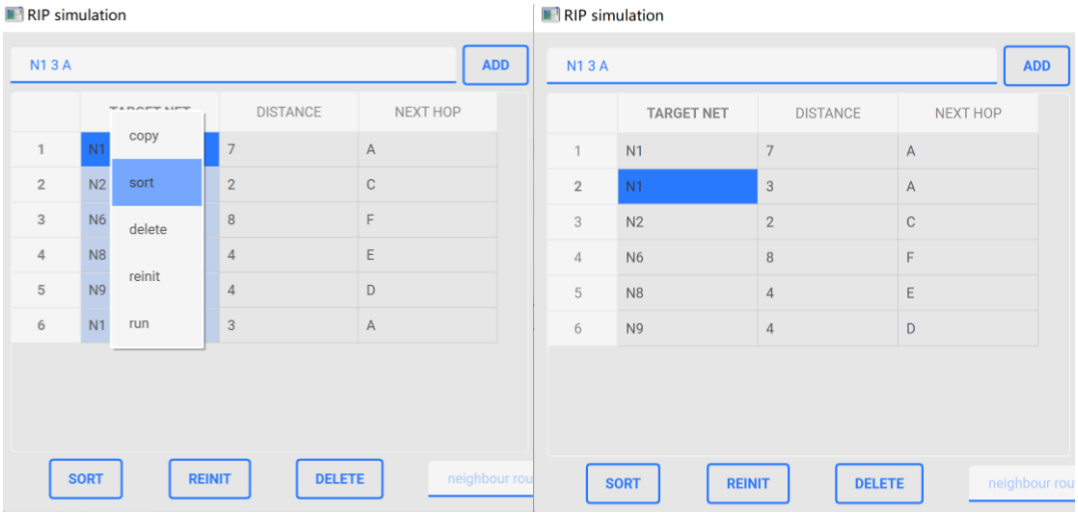
5.1 右键操作

通过复制按钮可以复制该单元格内容， 如图八所示。



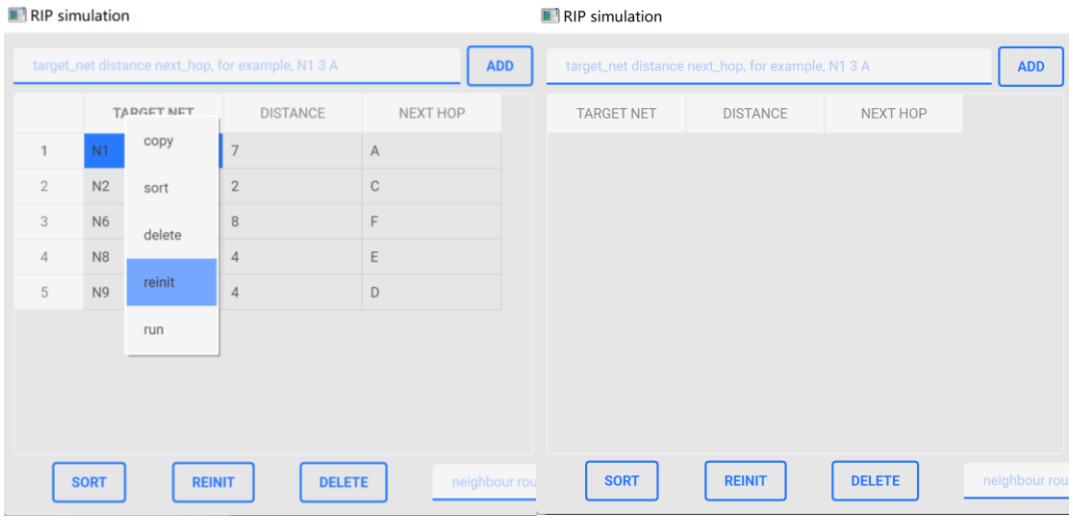
图八 复制

通过排序按钮可以将该路由信息，按照目的网络进行排序， 如图九所示。



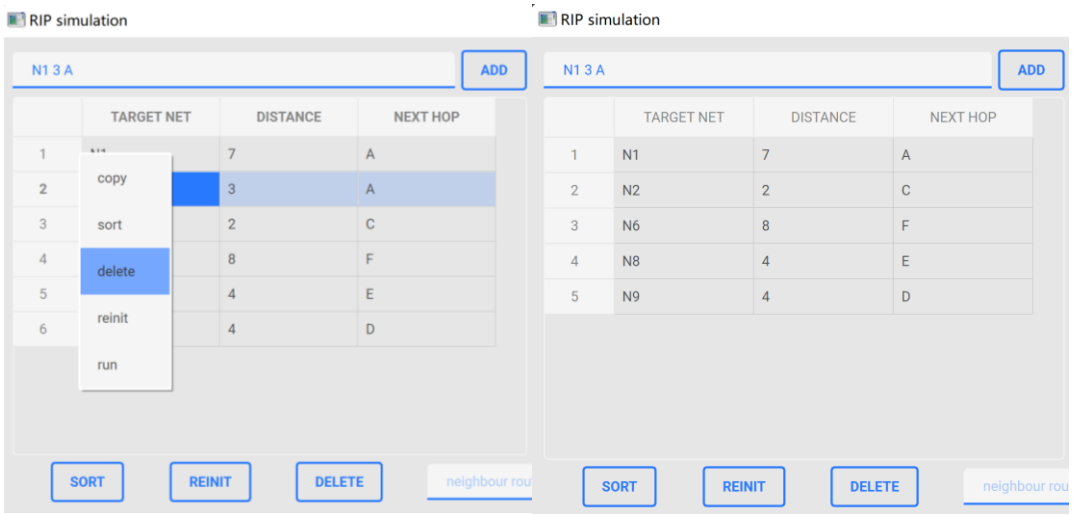
图九 排序

通过删除按钮进行删除行操作， 如图十所示。



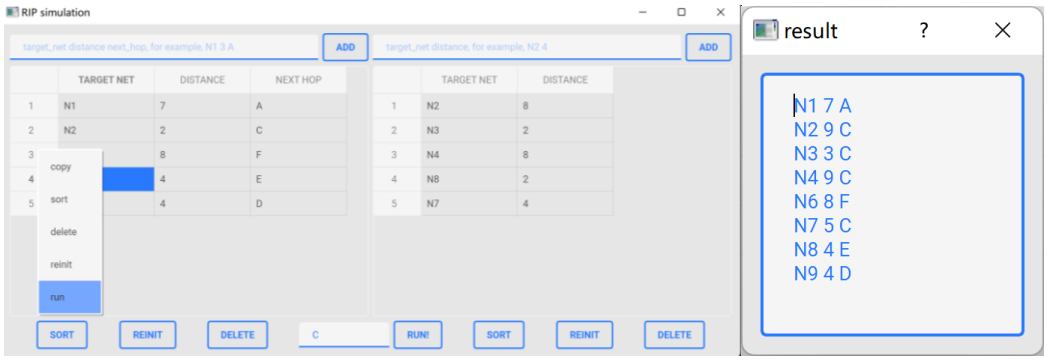
图十 删除

通过初始化按钮进行该路由信息表初始化操作， 如图十一所示。



图十一 初始化

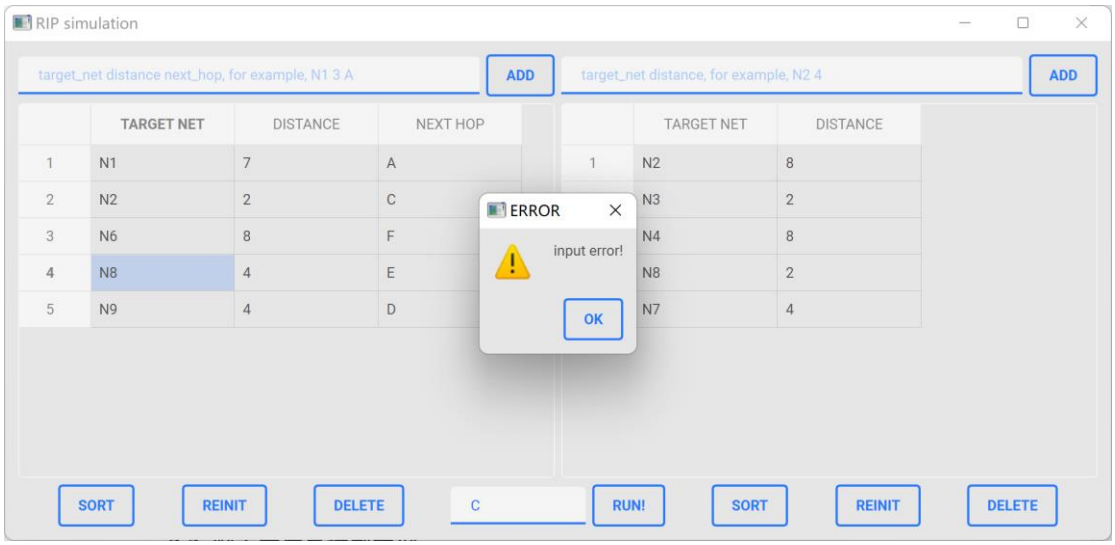
通过运行按钮，即可运行该程序， 如图十二所示。



图十二 运行

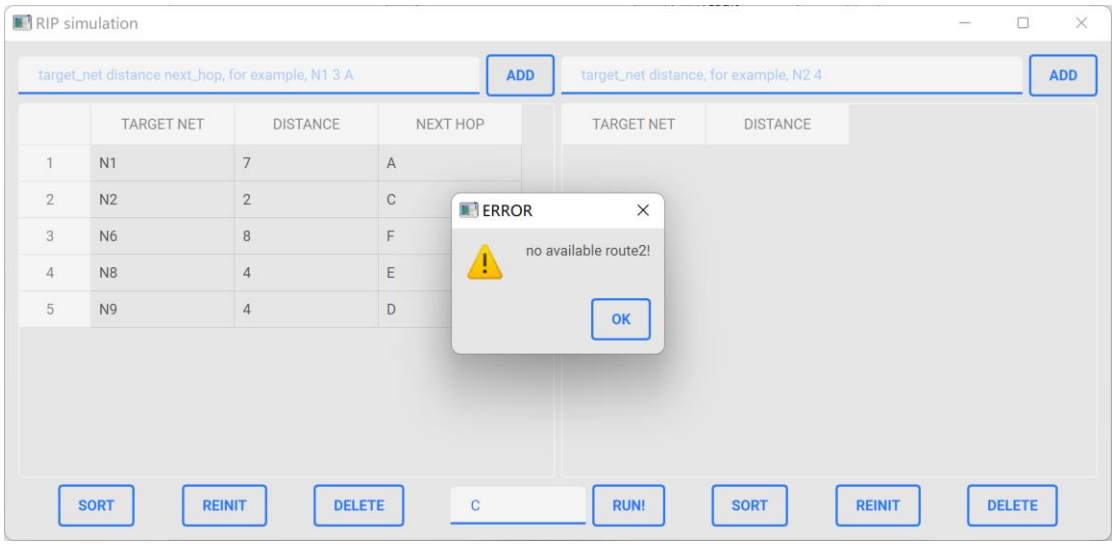
5.2 报错提示

当添加内容为空或者输入格式错误时，进行添加会出现输入错误报错， 如图十三所示。



图十三 输入错误报错

当当前路由信息为空时，相邻路由器的路由信息为空时或者相邻路由器为空时，会出现没有有效路由信息报错， 如图十四所示。

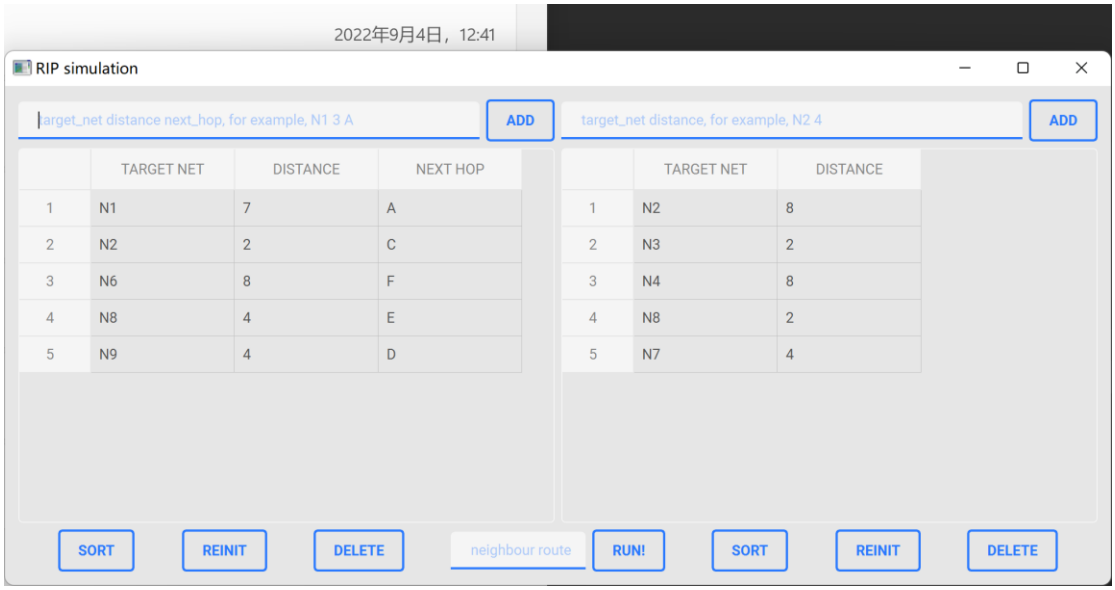


图十四 无有效信息报错

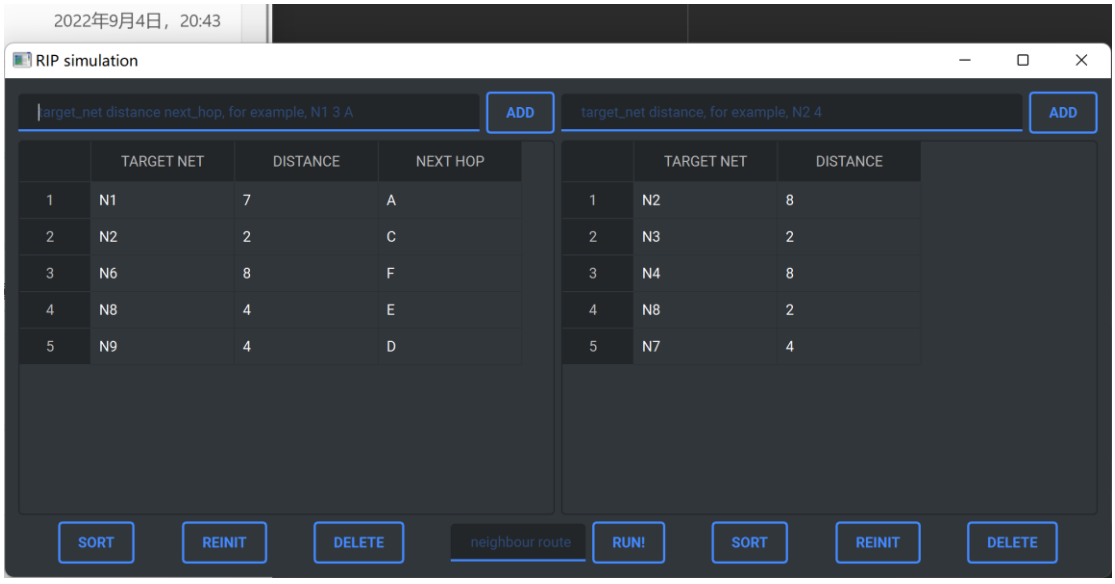
5.3 主题切换

该图形化界面子在不同的季节不同的时段，显示不同的主题。

在 python 程序中获取了当前的时间，并且通过判断季节，选择启用白天模式和夜间模式的时间，如图十五和图十六所示。



图十五 白天模式



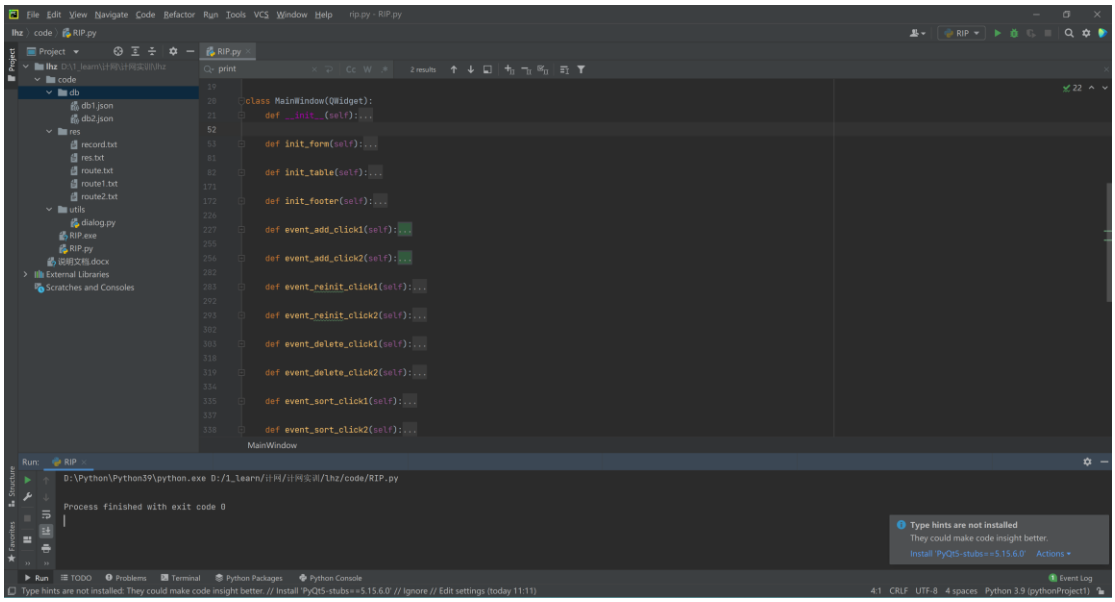
图十六 夜间模式

6. 开发环境

本项目使用 Python 语言和 C++语言进行开发，前端界面与后端程序分开进行开发。

6.1 前端界面开发

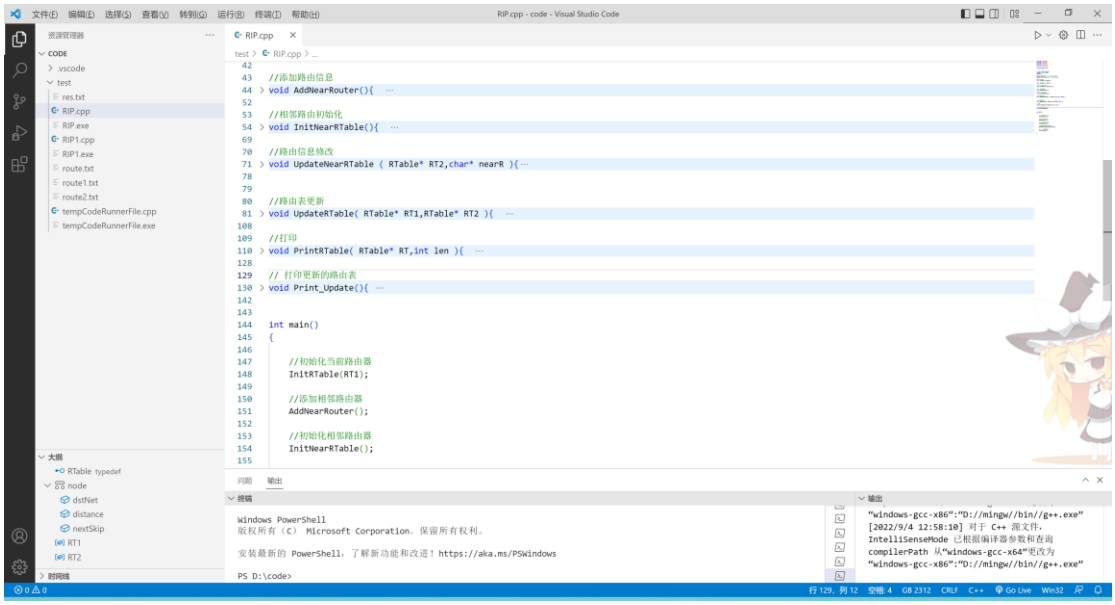
前端界面采用 Pycharm 2021 进行开发，如图十七所示。



图十七 前端界面开发

6.2 后端程序开发

后端程序采用版本 1.71.0 的 Visual Studio Code 进行开发，如图十八所示。


RIP.cpp - code - Visual Studio Code
RIP.cpp
//添加路由信息
void AddNearRouter(){
//相邻路由初始化
void InitNearRTable(){
//路由信息修改
void UpdateNearRTable (RTable* RT2,char* nearR){
//路由表更新
void UpdateRTable(RTable* RT1,RTable* RT2){
//打印
void PrintRTable(RTable* RT,int len){
// 打印更新的路由表
void Print_Update(){
int main()
{
//初始化当前路由器
InitRTable(RT1);
//添加相邻路由器
AddNearRouter();
//初始化相邻路由器
InitNearRTable();
Windows PowerShell
版权所有（C） Microsoft Corporation。保留所有权利。
安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows
PS D:\code>

图十八后端程序开发