# 问题分析

题目给定了一个规则集以及数据，规则级有5项组成：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 源IP地址 | 目的IP地址 | 源端口范围 | 目的端口范围 | 协议范围 |

其中源IP地址给定子网掩码，目的IP地址给定子网掩码。

待匹配的五元组如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 源IP地址 | 目的IP地址 | 源端口号 | 目的端口号 | 协议号 |

题目的要求就是将待匹配的五元组在规则集中找到与之匹配的规则。

**匹配成功：**

五元组的源IP地址网络号与目的IP地址网络号和规则集中的一样。

并且端口号范围和协议号范围一致。

默认顺序下来最先匹配到的。

# 暴力做法

对于每个待匹配的5元组，从第一行开始逐个扫描规则集中的元素，若匹配成功则返回，若在规则集中找不到可以匹配的规则，则返回-1。若规则集个数为n，待匹配的5元组个数为m，则时间复杂度为O（nm），如果n = 10000， m = 100000，则最坏的时间将达到1000000000.。

# 字典树做法：

考虑什么情况能够匹配成功：

**192.168.0.1/24**

**192.168.0.129/24**

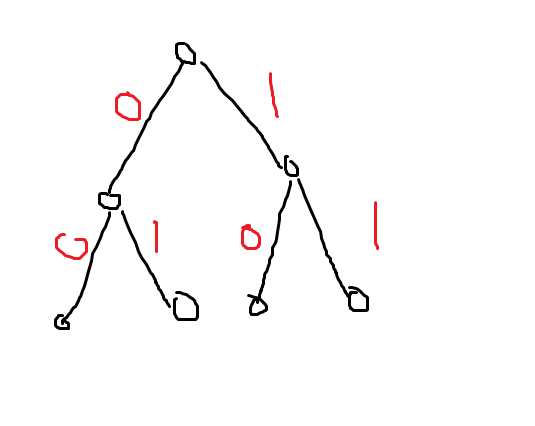
上述两个IP地址能够匹配成功，这是因为将他们转化成二进制之后能够保证前24位一致。

在朴素做法中，暴力对每个规则逐个判断，时间复杂度和规则集的规则个数成正比。

我们可以发现不管规则集个数如何变化，IP地址总是32位的，现在假定规则集有四个IP地址分别为：00/2， 01/2， 10/2， 11/2；

待匹配的IP地址为：11

暴力做法是逐个扫描，上面的例子在4次扫描之后能够得到答案。

优化做法：  
 

利用字典树的思想，对规则集建立字典树，可知只要经过2次比较就可以知道答案。

题目中规则集给定了5项，其中端口是给定了范围的，对规则集建立字典树较为麻烦，考虑到待匹配五元组个数多，而规则集相对少。我们可以反着对待匹配的5元组建立字典树。

假设待匹配的五元组个数为n，而规则集元素为m

建立字典树复杂度O(n \* 64)；

规则集查询时间复杂度少于n \* 64 \*m；

利用字典树可以快速找到该条规则可以匹配的五元组。

一个五元组可以匹配多个多个规则，由于我们是反着匹配，即通过规则来搜索五元组，所以会有多个规则匹配到同一个五元组，这时候就可以在匹配成功后比较该五元组之前被匹配到的规则，通过比较该规则与之前的规则的优先级来决定选择哪个。我们可以将五元组的源IP地址和目的IP地址合在一起，先对源IP地址建立字典树，然后再从上一个结点开始建立目的IP地址字典树，该字典树的深度为64位（因为两个IP地址二进制加在一起位64位），在叶子节点记录端口号等信息。

# 程序模块：

程序由下面几个模块组成：

Main.cpp：用于执行输出结果

Ipmatch.h:字典树建立以及查找封装

Test.cpp：用于比较输出答案与标准答案是否一致。

使用Makefile进行项目管理。

solve.exe rule.txt packet.txt//通过rule规则匹配packet.txt的结果，并将结果放在同目录下的ans.txt中

debug.exe// 用于调试

test.exe ans1.txt ans2.txt//用于判断两个答案是否一致，不一致返回第一个错误答案信息。

make main:生成主文件

make debug：生成调试文件

make test：生成答案判断文件

## 基本类型：

struct \_node{

int proto;//协议号

int s\_port;//源端口

int t\_port;//目的端口

int index;//该字典树节点对应原答案数组的下标

};

//字典树核心数据结构

typedef struct \_UpdateTrieNode{

\_UpdateTrieNode\* left;//表示左边是0

\_UpdateTrieNode\* right;//表示右边是1

std::vector<\_node> res;//相同的源地址和目的地址，可能存在多个不同的源端口号和目的端口号以及不同的协议号。

}UpdateTrieNode;

//匹配规则

typedef struct \_IPRULE{

//源IP地址

std::string s\_ip;

//目的IP地址

std::string t\_ip;

//源端口号范围

int spoint\_left;

int spoint\_right;

//目的端口号范围

int tpoint\_left;

int tpoint\_right;

//协议范围

std::string proto;

//优先级

int level;

}IPRULE;

//查找路由

typedef struct \_IPIMP{

ull s\_ip;

ull t\_ip;

//协议号

int proto;

//源端口号

int s\_point;

//目的端口号

int t\_point;

}IPIMP;

## IPMatch类

它用于生成字典树，并通过规则进行匹配。

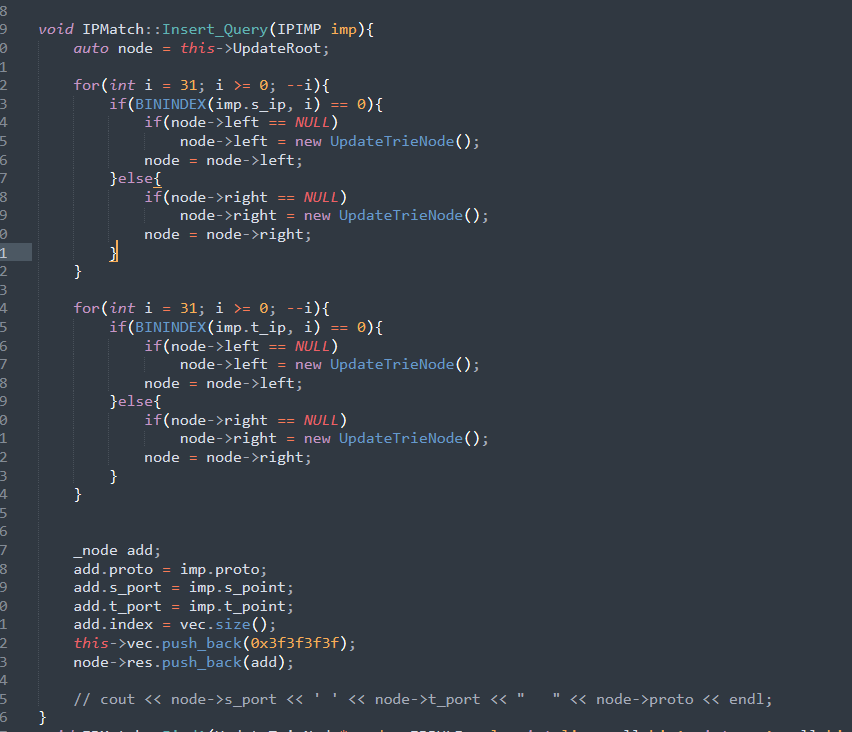
Insert\_Query:插入一条五元组

Search(rule, line);对于一条rule，查找与之匹配的五元组



### Insert\_Query(IMP imp):

用于插入一个五元组



### Search：

给定一个规则在字典树当中匹配五元组。

