计算机网络实验报告-12

阮星程

2015K8009929047

1. 实验题目

网络路由实验二。

1. 实验内容

理解OSPF协议构建路由表的原理，完成构建的第二步——利用Dijkstra算法计算最短路径，基于实验一构建的完整链路数据库，得到完整的路由表。

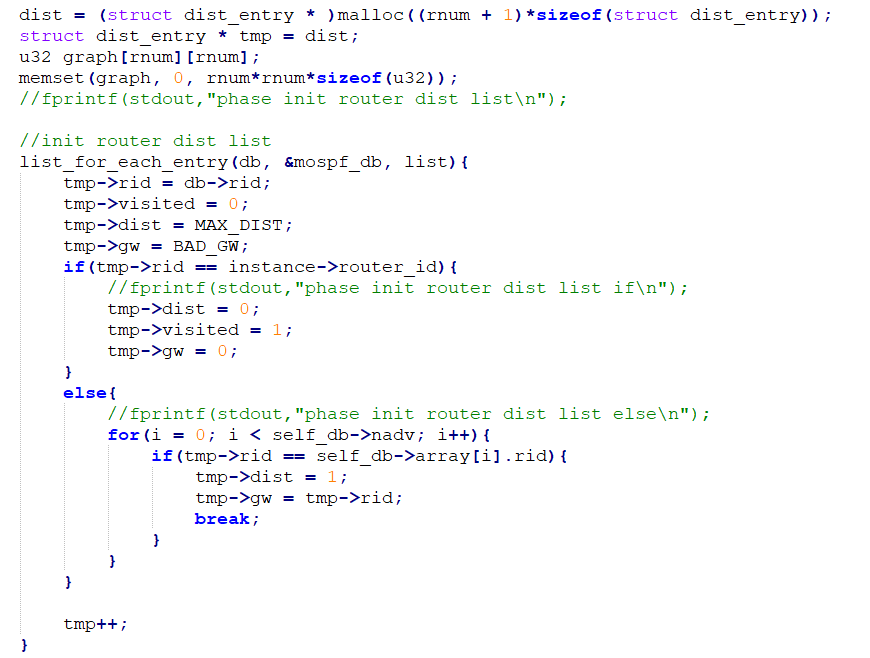
1. 实验过程

本次实验的主要代码在程序mospf\_daemon.c中，主要构架了一个新的函数，generate\_rt()，函数的作用是根据已有的链路数据库，构建出新的路由表，由于路由表条目在设计时没有超时的记录结构，一种做法是额外建立另一种表，与路由表同步维护，负责记录对应路由表项的存在时间；另一种是每次调用函数的时候不在意已经生成了的路由条目，直接重新生成。但第一种存在问题，不论这个新的数据结构效率怎样，在我们试图去添加路由条目的时候，为了决定这个路由条目是否在路由表中已有，这就要带来O(n2)的计算量，有些浪费，不如直接添加来的方便，所以选择了第二种做法。

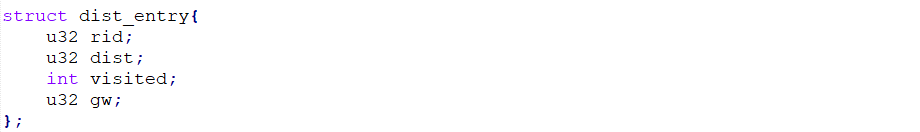
Dijkstra算法本身很简洁，在构建时也没有遇到什么问题，比较麻烦的反倒是构建起最初的距离条目以及邻接矩阵。由于本身数据结构的影响，在不考虑偷懒直接memcopy然后强行把原有的某些字段做其它用处的情况下，搬运与收集数据确实成了一个繁杂的工作，而且在链表数据结构的影响下，复杂度也是节节高升，让人总算是理解到了数据结构的重要性。

至于代码方面，这次我觉得没有太多需要展示的部分，毕竟只是按照Dijkstra算法逐步实现罢了，值得一提的是在rtable中增加了一个锁，避免了在发包时出现程序进入无限循环的情况。按道理来说这里没有死锁可以构成，找了一下之后还是没有注意到什么特殊的地方。在rtable中增加锁后，各程序就能正常有序地运行了。

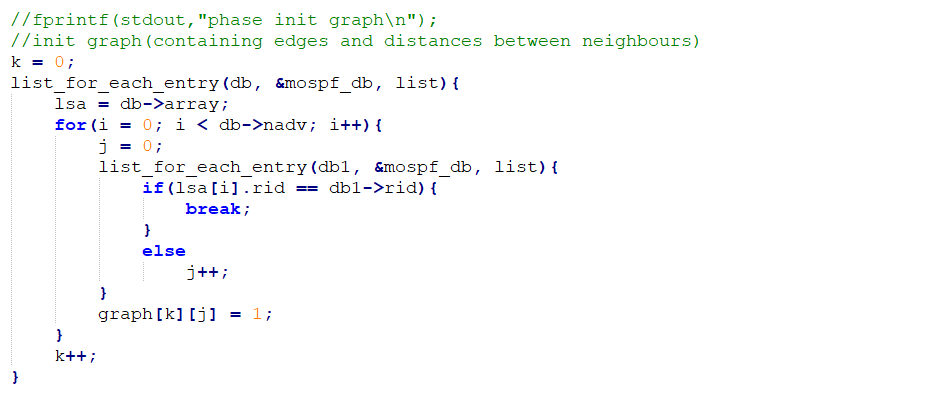
接下来还是按照常规把写的核心代码放上来，加以简单的解释。



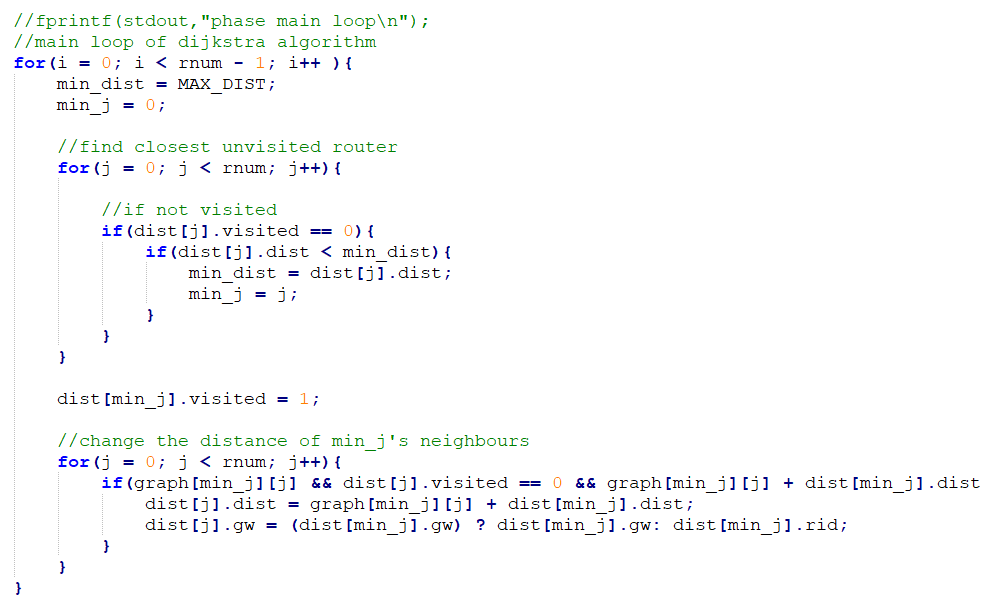
为了方便Dijkstra算法的实现，我新增加了一个数据结构



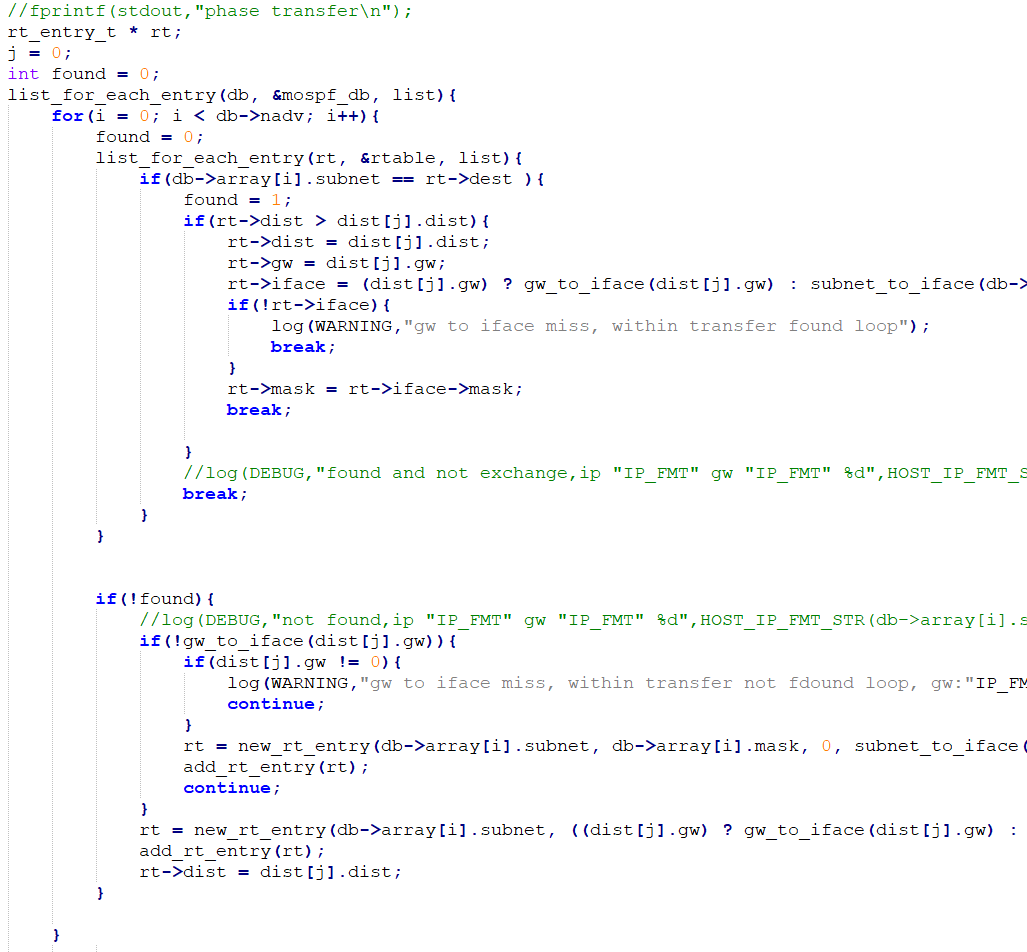
在这里对这个距离条目进行初始化，gw项负责记录到达这个节点所要通过的网关。从上图中可知，对于远处的节点，初始化rid，visited置0，dist置为最大距离，gw先置为BAD\_GW，对于源节点，dist置0，visited置1，gw置0。对于源节点的邻居节点，gw置为他们自身，距离置1，作为初始化条件，方便我们在之后的循环中扩散各自对应的gw。



初始化邻接矩阵。

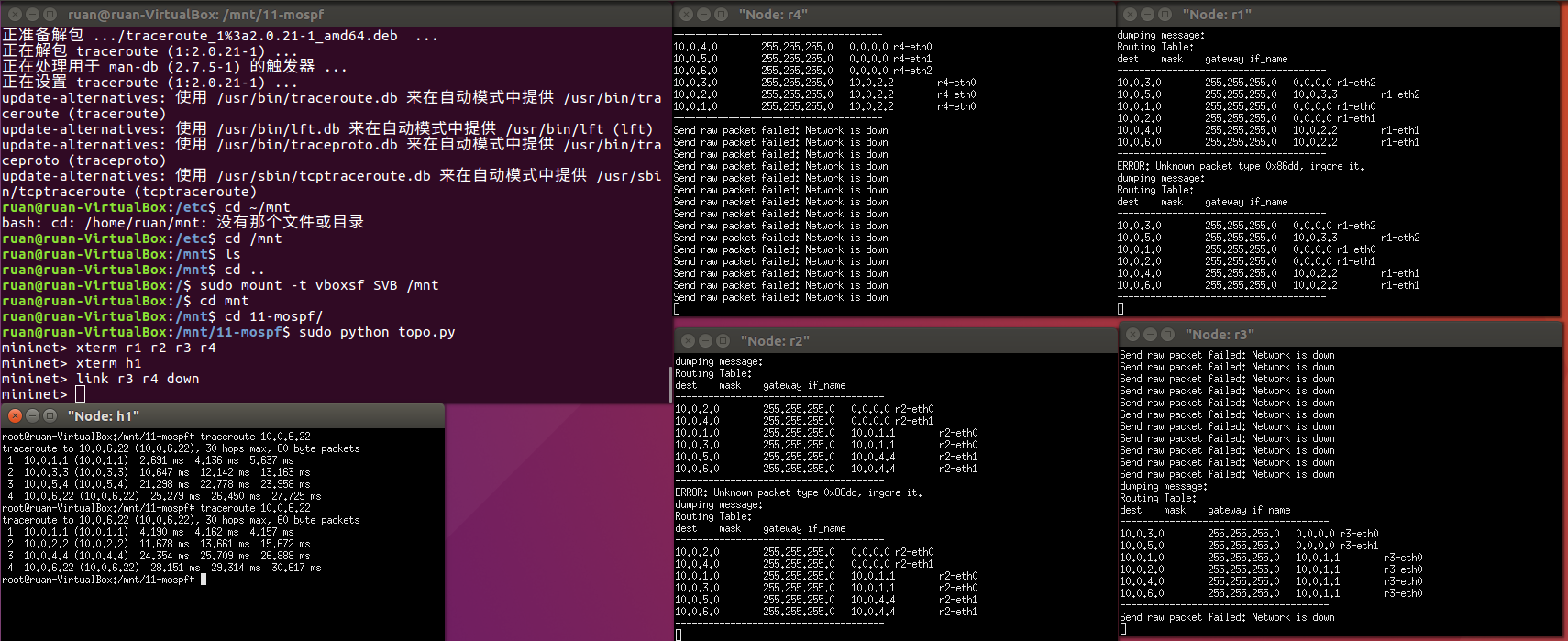


Dijkstra算法的核心循环，构建起了到各路由器的最短路径。



最后再将路径转化为对应的路由表条目。

1. 实验结果



可见上图，traceroute在删除路径前后得到了对应的正确路径，右边各个路由器的路由表项也正确，达成实验目的。

1. 实验总结

本次实验的核心Dijkstra算法很简单，但当你真正想要把它运用到对应的情境时，需要考虑的东西确实不少，要怎么把原有的东西转换成适合的数据结构，得到最短路径后要怎样得到我们想要的路由表，这些都不比核心的已有算法要麻烦不少，实验的时长也算是十分出人意料，居然用了一整天。其中三分之一的时间在考虑如何构建和协调数据结构，三分之一的时间在书写代码，三分之一的时间在调试，调试的话虽然只是一个很简单的问题，当第二次再看代码的时候，仔细读了三分钟，便发现了问题所在，可以说DEBUG这事，确实急不得，得静下心来弄，不过令人庆幸的是bug越来越少了，初次完成的代码便已经有了相应的质量，希望在之后这点能继续发扬光大。