湖南科技大学计算机科学与工程学院

计算机网络 课程设计报告

**专业班级：** 22计科六班

**姓 名：** 周俊哲

**学 号：** 2205010711

**指导教师：** 何庭钦

**时 间**：

**地 点**： 逸夫楼430

|  |
| --- |
| **指导教师评语：**  **成绩： 等级：**  **签名：**  **年 月 日** |

1. **实验题目**

**1、网络聊天程序的设计与实现**

**二、实验目的**

**了解 Socket 通信的原理，在此基础上编写一个聊天程序**

**三、总体设计（含背景知识或基本原理与算法、或模块介绍、设**

**计步骤等）**

**该程序实现了基于 TCP 协议的简单客户端-服务器通信模型。服务器通过 Winsock2 库提供的 API 创建一个 TCP 套接字，监听本地端口，等待客户端连接；客户端通过同样的 API 创建一个连接至服务器的套接字并进行通信。  
基本原理是服务器和客户端通过特定的 IP 地址和端口号进行套接字的绑定与连接，利用 TCP 协议保证数据传输的可靠性。**

**模块介绍**

1. **服务器模块：负责监听客户端连接，接收消息并向客户端回传消息。**
2. **客户端模块：负责向服务器发送消息并接收服务器的回应。**

**设计步骤**

1. **初始化 Winsock 库，设置通信环境。**
2. **服务器端：**
   * **创建套接字，绑定 IP 地址与端口号。**
   * **监听客户端连接并接受连接请求。**
   * **接收客户端消息并返回响应。**
3. **客户端：**
   * **创建套接字，连接服务器的 IP 地址与端口号。**
   * **向服务器发送消息并接收回应。**
4. **清理环境，关闭套接字。**

**四、详细设计（含主要的数据结构、程序流程图、关键代码等）   
//创建套接字**

**SOCKET sockServer = socket(AF\_INET, SOCK\_STREAM, 0);**

**SOCKADDR\_IN addrSrv;**

**//设置服务器地址**

**addrSrv.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);**

**addrSrv.sin\_family = AF\_INET;**

**addrSrv.sin\_port = htons(6000);**

**//绑定套接字**

**bind(sockServer, (SOCKADDR\*)&addrSrv, sizeof(SOCKADDR));**

**if (err == SOCKET\_ERROR) {**

**printf("Bind failed\n");**

**closesocket(sockServer);**

**WSACleanup();**

**return 0;**

**}**

**//监听客户端连接**

**listen(sockServer, 5);**

**if (err == SOCKET\_ERROR) {**

**printf("Listen failed\n");**

**closesocket(sockServer);**

**WSACleanup();**

**return 0;**

**}**

**printf("Server is listening on port 6000...\n");**

**// 等待客户端连接**

**int addrSize, recvSize;**

**SOCKET sockClient;**

**SOCKADDR\_IN addrClient;**

**char recvBuf[1024];**

**addrSize = sizeof(SOCKADDR\_IN);**

**sockClient = accept(sockServer, (SOCKADDR\*)&addrClient, &addrSize);**

**if (sockClient == INVALID\_SOCKET) {**

**printf("Accept failed\n");**

**closesocket(sockServer);**

**WSACleanup();**

**return 0;**

**}**

**char ipStr[INET\_ADDRSTRLEN]; // 用于存储 IP 地址字符串**

**if (inet\_ntop(AF\_INET, &addrClient.sin\_addr, ipStr, sizeof(ipStr)) != nullptr) {**

**printf("Client connected: %s\n", ipStr);**

**}**

**else {**

**perror("inet\_ntop failed");**

**}**

**// 处理客户端消息**

**while (true) {**

**recvSize = recv(sockClient, recvBuf, sizeof(recvBuf), 0);**

**if (recvSize == SOCKET\_ERROR || recvSize == 0) {**

**printf("Client disconnected\n");**

**break;**

**}**

**recvBuf[recvSize] = '\0'; // 添加字符串结束符**

**printf("Client: %s\n", recvBuf);**

**// 回应客户端**

**printf("Server: ");**

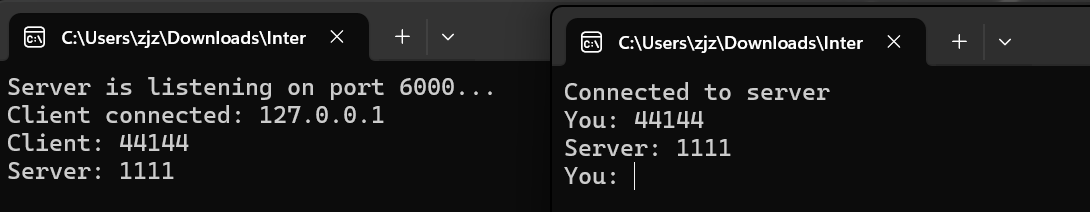
**string response;**

**getline(cin, response);**

**send(sockClient, response.c\_str(), response.length() + 1, 0);**

**}**

**五、实验结果与分析**

 **实验结果：**

1. **服务器成功启动后，在指定端口监听客户端的连接请求。**
2. **客户端成功连接服务器，发送消息“hello”，服务器返回响应。**
3. **通信过程稳定，能支持多次消息传递（如增加循环支持多次交互）。**

**分析：**

* **成功模拟了 TCP 通信的基本过程。**
* **客户端与服务器的连接正常，数据传递正确。**
* **不足之处：程序未实现异常处理的完善机制，例如连接失败或数据传输错误时的处理。**

**六、小结与心得体会  
熟悉了 Windows 网络通信接口及其工作流程。**

**理解了 TCP/IP 协议在数据传输中的可靠性保证。**

**收获了实际编程中如何处理套接字的生命周期管理和异常情况的经验。  
改进方向：可以尝试实现更多的功能，如多客户端支持、数据加密、长连接等。**

**一、实验题目   
2、Tracert 与 Ping 程序设计与实现**

**二、实验目的**

**了解 Tracert 程序的实现原理，并调试通过。**

**三、总体设计（含背景知识或基本原理与算法、或模块介绍、设**

**计步骤等）   
背景知识与基本原理**

**该程序主要实现了ICMP（Internet Control Message Protocol）的应用，使用了Windows Socket编程，通过原始套接字实现了网络子网状态的扫描功能。**

1. **ICMP协议：用于传输网络控制消息，例如PING命令中的回显请求和回显应答。**
2. **TTL（Time to Live）：IP数据包中的生存时间字段，用于限制数据包在网络中的生存周期。**
3. **网络状态检测：程序采用ICMP回显请求（类型值为8）来判断子网中的主机是否在线。**

**模块介绍**

1. **Checksum计算模块：用于计算ICMP校验和，确保数据包完整性。**
2. **ICMP报文处理模块：构造、发送和接收ICMP报文，处理回显应答和超时消息。**
3. **子网扫描模块：通过循环对特定子网范围的IP进行Ping操作，判断设备的在线状态。**
4. **结果输出模块：解析返回的数据包，并输出IP地址及在线状态。**

**设计步骤**

1. **初始化网络环境，设置子网扫描范围。**
2. **构造ICMP回显请求报文并设置TTL字段。**
3. **循环发送ICMP报文，并接收应答，解析数据包。**
4. **判断设备在线状态并记录结果。**
5. **输出子网状态扫描结果。**

**四、详细设计（含主要的数据结构、程序流程图、关键代码等）   
//对数据包进行解码**

**BOOL DecodeIcmpResponse(char\* pBuf, int iPacketSize, DECODE\_RESULT& DecodeResult, BYTE ICMP\_ECHO\_REPLY, BYTE ICMP\_TIMEOUT)**

**{**

**//检查数据报大小的合法性**

**IP\_HEADER\* pIpHdr = (IP\_HEADER\*)pBuf;**

**int iIpHdrLen = pIpHdr->hdr\_len \* 4;**

**if (iPacketSize < (int)(iIpHdrLen + sizeof(ICMP\_HEADER)))**

**return FALSE;**

**//根据 ICMP 报文类型提取 ID 字段和序列号字段**

**ICMP\_HEADER\* pIcmpHdr = (ICMP\_HEADER\*)(pBuf + iIpHdrLen);**

**USHORT usID, usSquNo;**

**if (pIcmpHdr->type == ICMP\_ECHO\_REPLY) //ICMP 回显应答报文**

**{**

**usID = pIcmpHdr->id; //报文 ID**

**usSquNo = pIcmpHdr->seq; //报文序列号**

**}**

**else if (pIcmpHdr->type == ICMP\_TIMEOUT)//ICMP 超时差错报文**

**{**

**char\* pInnerIpHdr = pBuf + iIpHdrLen + sizeof(ICMP\_HEADER); //载荷中的 IP 头**

**int iInnerIPHdrLen = ((IP\_HEADER\*)pInnerIpHdr)->hdr\_len \* 4; //载荷中的 IP 头长**

**ICMP\_HEADER\* pInnerIcmpHdr = (ICMP\_HEADER\*)(pInnerIpHdr + iInnerIPHdrLen);//载荷中的 ICMP 头**

**usID = pInnerIcmpHdr->id; //报文 ID**

**usSquNo = pInnerIcmpHdr->seq; //序列号**

**}**

**else**

**{**

**return false;**

**}**

**//检查 ID 和序列号以确定收到期待数据报**

**if (usID != (USHORT)GetCurrentProcessId() || usSquNo != DecodeResult.usSeqNo)**

**{**

**return false;**

**}//记录 IP 地址并计算往返时间**

**DecodeResult.dwIPaddr.s\_addr = pIpHdr->sourceIP;**

**DecodeResult.dwRoundTripTime = GetTickCount() - DecodeResult.dwRoundTripTime;**

**//处理正确收到的 ICMP 数据报**

**if (pIcmpHdr->type == ICMP\_ECHO\_REPLY || pIcmpHdr->type == ICMP\_TIMEOUT)**

**{**

**//输出往返时间信息**

**if (DecodeResult.dwRoundTripTime)**

**cout << " " << DecodeResult.dwRoundTripTime << "ms" << flush;**

**else**

**cout << " " << "<1ms" << flush;**

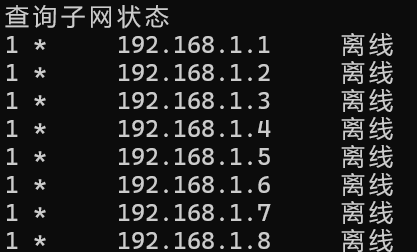
**}**

**return true;**

**}**

**五、实验结果与分析**

**实验结果**



1. **成功扫描子网范围内的IP地址。**
2. **输出了在线设备的IP地址和往返时间（RTT）。**
3. **离线设备被标记为“离线”。**

**六、小结与心得体会**

**本次实验通过Windows Socket编程实现了子网状态的扫描，能够高效检测在线设备并输出扫描结果。程序采用ICMP协议实现，与实际网络工具（如Ping）功能类似。**

1. **实验题目**

**3、滑动窗口协议仿真**

**二、实验目的   
 滑动窗口协议以基于分组的数据传输协议为特征，该协议适用于在数据链路层以及传输层中对按顺序传送分组的可靠性要求较高的环境。**

**三、总体设计（含背景知识或基本原理与算法、或模块介绍、设**

**计步骤等）**

**1. 背景知识与基本原理**

**本程序实现的是基于滑动窗口协议的传输模拟，包括发送方和接收方之间的通信。滑动窗口协议是一种流量控制机制，通过限制发送方和接收方之间未确认帧的数量，确保通信的可靠性与高效性。**

* **核心概念：**
  + **窗口大小：发送方在某一时间段内可以连续发送的帧数。**
  + **ACK机制：接收方发送确认信号（ACK）告知发送方已成功接收到某一帧。**
  + **超时与重传：如果发送方未在规定时间内收到ACK，会重新发送帧。**
  + **丢包与误码：通过随机数模拟帧丢失现象。**

**2. 模块介绍**

1. **发送方模块：**
   * **管理滑动窗口。**
   * **发送帧，并等待ACK。**
   * **检测ACK是否超时，并在需要时重新发送帧。**
2. **接收方模块：**
   * **接收发送方的帧。**
   * **检测是否存在帧丢失。**
   * **返回ACK信号。**
3. **多线程实现：**
   * **使用Windows API的线程机制，同时运行发送方与接收方。**

**3. 设计步骤**

1. **初始化发送方和接收方。**
2. **使用线程分别运行发送和接收模块。**
3. **模拟随机丢包和重传过程。**
4. **记录发送与接收的时间，计算效率。**

**四、详细设计（含主要的数据结构、程序流程图、关键代码等）**

**// 帧结构体，包含帧序号和是否被确认的状态**

**typedef struct {**

**int seq\_num; // 帧的序号**

**bool isAck; // 是否是确认帧**

**} Frame;**

**// 发送方结构体**

**typedef struct {**

**int windowStart; // 滑动窗口的起始位置**

**Frame frames[TOTAL\_FRAMES]; // 所有帧**

**bool ackReceived[TOTAL\_FRAMES]; // 存储ACK确认状态**

**} Sender;**

**// 接收方结构体**

**typedef struct {**

**bool receivedFrames[TOTAL\_FRAMES]; // 标记是否收到该帧**

**} Receiver;**

**// 更新窗口，滑动窗口的起始位置移动**

**void updateWindow(Sender\* sender) {**

**while (sender->ackReceived[sender->windowStart]) {**

**sender->ackReceived[sender->windowStart] = false; // 重置确认状态**

**sender->windowStart++;**

**if (sender->windowStart >= TOTAL\_FRAMES) break;**

**}**

**}**

**// 发送ACK**

**void sendAck(int seq\_num) {**

**printf("接收方:\t发送帧 %d 的ACT\n", seq\_num);**

**framesReceived++;**

**}**

**// 接收帧**

**void receiveFrame(Receiver\* receiver, Frame frame) {**

**if (frame.seq\_num < TOTAL\_FRAMES) {**

**printf("接收方:\t接收方收到帧: %d\n", frame.seq\_num);**

**receiver->receivedFrames[frame.seq\_num] = true;**

**// 模拟随机丢包，70%概率发送ACK**

**if (rand() % 10 < 7) {**

**sendAck(frame.seq\_num);**

**}**

**}**

**}**

**void receiveAck(Sender\* sender);**

**// 发送帧**

**void sendFrames(Sender\* sender, double& sendTime) {**

**cout << endl;**

**clock\_t startSend = clock(); // 记录发送开始时间**

**while (framesSent < TOTAL\_FRAMES) {**

**// 填充当前窗口**

**printf("发送方:\t发送窗口起始帧 %d\n", sender->windowStart);**

**for (int i = sender->windowStart; i < sender->windowStart + WINDOW\_SIZE && i < TOTAL\_FRAMES; i++) {**

**printf("发送方:\t发送帧: %d\n", sender->frames[i].seq\_num);**

**framesSent++;**

**}**

**cout << endl;**

**clock\_t endSend = clock(); // 记录发送结束时间**

**sendTime = double(endSend - startSend) / CLOCKS\_PER\_SEC; // 计算发送所用时间**

**// 模拟发送完所有帧后等待确认**

**Sleep(1000);**

**// 模拟接收ACK**

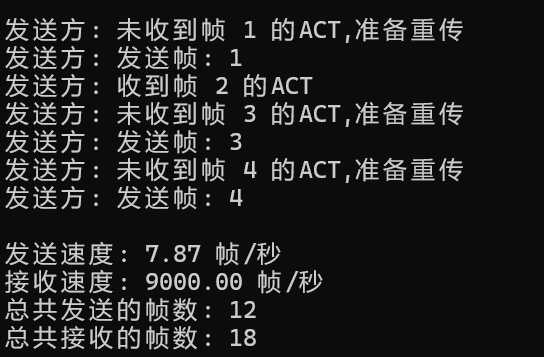
**receiveAck(sender);**

**}**

**}**

**五、实验结果与分析**

**实验结果:**

 **发送和接收的速度接近，表明程序运行效率较高。**

**模拟了丢包、重传等现象，展示了滑动窗口协议的可靠性。**

**六、小结与心得体会**

**通过实现滑动窗口协议，深入理解了流量控制和可靠性传输的原理。程序通过多线程模拟了实际网络中的数据传输过程。**

1. **实验题目**

**4、OSPF 路由协议原型系统设计与实现**

**二、实验目的**

**参考附录4及教材164页OSPF路由协议工作原理，在此基础上，实现一个简单的原型系统。**

**三、总体设计（含背景知识或基本原理与算法、或模块介绍、设**

**计步骤等）**

**本程序实现了基于邻接矩阵的无向图表示法，并使用 Dijkstra 算法 求解单源最短路径问题。**

* **图论基础：**
  + **图由顶点集和边集组成，用邻接矩阵表示顶点间的连接关系。**
  + **权值表示两顶点间的代价或距离。**
* **Dijkstra 算法：**
  + **适用于非负权值图，逐步扩展已找到最短路径的顶点集合，直到所有顶点的最短路径被确定。**
  + **核心思想是贪心策略，每次选择未访问的、距离起点最近的顶点更新路径。**
* **主要功能模块：**
  + **图的创建：支持用户输入顶点和边。**
  + **图的表示：以邻接矩阵形式显示图结构。**
  + **最短路径计算：基于 Dijkstra 算法计算从每个顶点到其他所有顶点的最短路径。**

**2. 设计步骤**

1. **输入与初始化：**
   * **用户输入图的顶点数和边数。**
   * **初始化邻接矩阵，边的权值初始化为最大值（表示无连接）。**
2. **构建图：**
   * **输入每条边的起点、终点及权重，更新邻接矩阵。**
3. **最短路径求解：**
   * **遍历所有顶点，使用 Dijkstra 算法计算其到其他顶点的最短路径。**
   * **输出路径及对应权重。**

**四、详细设计（含主要的数据结构、程序流程图、关键代码等）**

1. **图的创建：**

**void CreateGraph(Graph& g) {**

**cout << "请输入顶点数：";**

**cin >> g.vexnum;**

**cout << "请输入边数：";**

**cin >> g.arcnum;**

**// 初始化邻接矩阵**

**for (int i = 0; i < g.vexnum; i++) {**

**for (int j = 0; j < g.vexnum; j++) {**

**g.arcs[i][j] = (i == j) ? 0 : Max;**

**}**

**}**

**// 输入边信息**

**for (int k = 0; k < g.arcnum; k++) {**

**char v1, v2;**

**int w;**

**cin >> v1 >> v2 >> w;**

**int i = LocateVex(g, v1), j = LocateVex(g, v2);**

**g.arcs[i][j] = g.arcs[j][i] = w;**

**}**

**}**

1. **Dijkstra 算法：**

**void shortDIJ(Graph g, int v0) {**

**vector<int> s(g.vexnum, 0), d(g.vexnum, Max), p(g.vexnum, -1);**

**s[v0] = 1;**

**d[v0] = 0;**

**for (int v = 0; v < g.vexnum; v++) {**

**if (g.arcs[v0][v] < Max) p[v] = v0;**

**}**

**for (int i = 1; i < g.vexnum; i++) {**

**int Min = Max, v;**

**for (int w = 0; w < g.vexnum; w++) {**

**if (!s[w] && d[w] < Min) {**

**v = w;**

**Min = d[w];**

**}**

**}**

**s[v] = 1;**

**for (int w = 0; w < g.vexnum; w++) {**

**if (!s[w] && d[v] + g.arcs[v][w] < d[w]) {**

**d[w] = d[v] + g.arcs[v][w];**

**p[w] = v;**

**}**

**}**

**}**

**// 输出最短路径**

**for (int i = 0; i < g.vexnum; i++) {**

**if (i != v0 && d[i] < Max) {**

**cout << "从 " << g.vexs[v0] << " 到 " << g.vexs[i] << " 的最短路径为：" << d[i] << endl;**

**}**

**}**

**}**

1. **结果展示：**

**void showGraph(Graph g) {**

**cout << "\t";**

**for (int i = 0; i < g.vexnum; i++) cout << g.vexs[i] << "\t";**

**cout << endl;**

**for (int i = 0; i < g.vexnum; i++) {**

**cout << g.vexs[i] << "\t";**

**for (int j = 0; j < g.vexnum; j++) {**

**cout << ((g.arcs[i][j] == Max) ? "∞" : to\_string(g.arcs[i][j])) << "\t";**

**}**

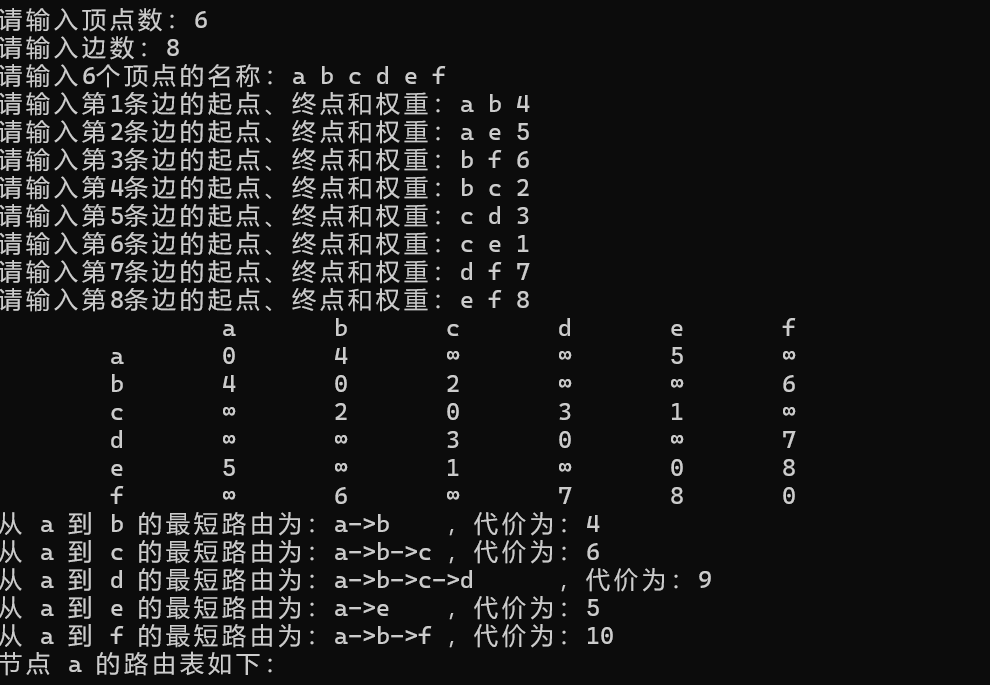
**cout << endl;**

**}**

**}**

**五、实验结果与分析**

**实验结果:**



**六、小结与心得体会**

**理解了 Dijkstra 算法的核心思想和实现方式。**

**学会了如何使用邻接矩阵高效表示图。**

**掌握了在图论问题中处理边权与路径计算的技巧。**

1. **实验题目**

**5、网络路由模拟器设计与实现**

**二、实验目的**

**设计并实现一个可以模拟不同网络拓扑结构下的路由选择和数据包转发的路由模拟器。**

**三、总体设计（含背景知识或基本原理与算法、或模块介绍、设**

**计步骤等）**

**1. 背景知识与基本原理**

**本程序模拟了网络路由中最短路径的计算和数据包的传输。主要依赖以下背景知识：**

* **图论：使用邻接矩阵表示图，其中顶点表示路由器，边表示路由器间的连接，权重表示连接的代价。**
* **Dijkstra算法：求解单源最短路径，用于动态更新路由表。**
* **递归传输：模拟数据包从源路由器到目标路由器的逐跳转发。**

**2. 模块介绍**

1. **图创建模块：**
   * **构造一个无向加权图，表示路由器之间的网络拓扑。**
2. **路由表更新模块：**
   * **使用Dijkstra算法填充每个路由器的路由表，包括目的地、最短距离、下一跳。**
3. **数据包传输模块：**
   * **模拟数据包从源路由器到目标路由器的传输，输出每一跳的传输情况。**
4. **可视化模块：**
   * **打印邻接矩阵展示网络拓扑结构。**

**3. 设计步骤**

1. **初始化网络拓扑和路由器的初始信息。**
2. **构建图的邻接矩阵表示，调用 CreateGraph 函数生成图。**
3. **使用Dijkstra算法计算最短路径，填充每个路由器的路由表。**
4. **模拟数据包的递归传输，调用 send\_packet 输出传输过程。**

**四、详细设计（含主要的数据结构、程序流程图、关键代码等）**

**1. 主要数据结构**

**typedef struct {**

**string destination; // 目的地**

**int distance; // 距离**

**string nextHop; // 下一跳**

**} Route;**

**typedef struct {**

**string name; // 路由器名称**

**Route table[MAX\_DESTINATIONS]; // 路由表**

**int tableSize; // 路由表大小**

**} Router;**

**typedef struct {**

**int info; // 附加信息**

**string vexs[MVNum]; // 顶点表**

**int arcs[MVNum][MVNum]; // 邻接矩阵**

**int vexnum, arcnum; // 顶点和边的数量**

**} Graph;**

**关键代码**

**Dijkstra 算法更新路由表：**

**static void shortDIJ(Graph g) {**

**vector<int> s(g.vexnum, 0), d(g.vexnum, Max), p(g.vexnum, -1);**

**vector<string> temp(g.vexnum);**

**for (int v0 = 0; v0 < g.vexnum; v0++) {**

**int router = LocalRouter(g.vexs[v0]);**

**fill(d.begin(), d.end(), Max);**

**fill(p.begin(), p.end(), -1);**

**s[router] = 1;**

**d[router] = 0;**

**for (int i = 1; i < g.vexnum; i++) {**

**int Min = Max, v = -1;**

**for (int w = 0; w < g.vexnum; w++) {**

**if (!s[w] && d[w] < Min) {**

**v = w; Min = d[w];**

**}**

**}**

**if (v == -1) break;**

**s[v] = 1;**

**for (int w = 0; w < g.vexnum; w++) {**

**if (!s[w] && d[v] + g.arcs[v][w] < d[w]) {**

**d[w] = d[v] + g.arcs[v][w];**

**p[w] = v;**

**}**

**}**

**}**

**for (int i = 0; i < g.vexnum; i++) {**

**if (i != router && d[i] < Max) {**

**string nextHop = g.vexs[p[i]];**

**network[router].table[network[router].tableSize++] = {g.vexs[i], d[i], nextHop};**

**}**

**}**

**}**

**}**

**数据包传输递归函数：**

**int send\_packet(Router\* current\_router, string source, string destination, string packet) {**

**cout << "Packet from " << source << " to " << destination << ": " << packet << endl;**

**string nextHop = find\_best\_path(current\_router, destination);**

**if (nextHop == "Destination Unreachable") {**

**cout << "Packet cannot be delivered." << endl;**

**return -1;**

**} else {**

**Router\* next\_router = find\_router(nextHop);**

**if (nextHop == destination) {**

**cout << "Packet successfully delivered to " << destination << endl;**

**return 0;**

**} else {**

**return send\_packet(next\_router, nextHop, destination, packet);**

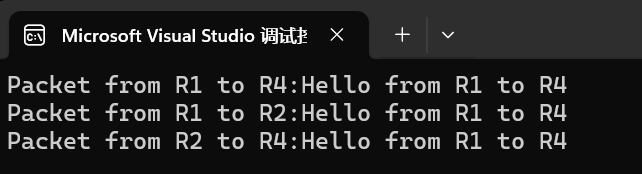
**}**

**}**

**}**

**五、实验结果与分析**

**实验结果:**



**六、小结与心得体会**

**算法实践：通过实现Dijkstra算法，深化了对图论算法的理解。**

**递归应用：递归调用用于数据包传输，简化了逐跳传输的实现。**

1. **实验题目**

**6、编程模拟NAT网络地址转换**

**二、实验目的**

**模拟NAT路由器的工作过程，主要有2个步骤的工作：1、将收到的来自 内网报文中的私有源IP地址转换为NAT的外部合法IP地址，同时将传输层源端口号转换为NAT路 由器分配的端口号，建立转换映射表；2、将收到的来自外网的应答报文提取其目的IP地址及端口号， 查找映射表，找到其对应的内网机器的IP地址及端口号并替换。**

**三、总体设计（含背景知识或基本原理与算法、或模块介绍、设**

**计步骤等）**

**1. 背景知识与基本原理**

**网络地址转换 (NAT, Network Address Translation) 是一种通过修改 IP 地址和端口号在本地网络和外部网络之间建立通信的方法。它在路由器或防火墙中执行，用于：**

* **实现私有地址到公共地址的映射。**
* **节省 IPv4 地址。**
* **提高网络安全性，隐藏内部网络结构。**

**2. 基本算法和实现原理**

* **使用两个映射表：**
  + **Maps\_In：外部地址到内部地址的映射。**
  + **Maps\_Out：内部地址到外部地址的映射。**
* **提供基本功能：地址转换、地址新增、地址修改、地址删除、映射表显示。**
* **通过查找键值映射 (std::map) 实现高效的数据存储与检索。**

**3. 模块介绍**

1. **数据初始化模块 (Init\_data)**
   * **初始化内部和外部地址映射表。**
2. **映射表显示模块 (show\_data)**
   * **显示内部到外部和外部到内部的映射表。**
3. **地址转换模块 (Net\_address\_translation)**
   * **根据用户输入的地址，查找映射表，显示对应的内部或外部地址。**
4. **新增地址模块 (Net\_address\_add)**
   * **添加新的内部与外部地址映射。**
5. **删除地址模块 (Net\_address\_delect)**
   * **删除指定的地址及其映射关系。**
6. **修改地址模块 (Net\_address\_revise)**
   * **修改指定的内部或外部地址。**

**4. 设计步骤**

1. **定义数据结构：**
   * **使用 std::map 实现地址映射表。**
2. **实现功能模块：**
   * **按照功能划分模块，实现地址的新增、删除、修改、查询及映射表显示。**
3. **提供主菜单交互：**
   * **循环显示菜单，根据用户选择调用相应模块。**
4. **优化代码结构：**
   * **使用清屏函数和模块化设计，增强用户体验。**

**四、详细设计（含主要的数据结构、程序流程图、关键代码等）**

**关键代码**

* **地址新增：**

**void Net\_address\_add() {**

**string Input\_interior\_addrs, Input\_exterior\_addrs;**

**cout << "请输入内部地址: ";**

**cin >> Input\_interior\_addrs;**

**cout << "请输入外部地址: ";**

**cin >> Input\_exterior\_addrs;**

**Maps\_In[Input\_exterior\_addrs] = Input\_interior\_addrs;**

**Maps\_Out[Input\_interior\_addrs] = Input\_exterior\_addrs;**

**cout << "新增映射成功！" << endl;**

**}**

* **地址转换：**

**void Net\_address\_translation() {**

**string Search\_addrs;**

**cout << "请输入地址: ";**

**cin >> Search\_addrs;**

**if (Maps\_In.count(Search\_addrs)) {**

**cout << "内部地址: " << Maps\_In[Search\_addrs] << endl;**

**} else if (Maps\_Out.count(Search\_addrs)) {**

**cout << "外部地址: " << Maps\_Out[Search\_addrs] << endl;**

**} else {**

**cout << "地址未找到！" << endl;**

**}**

**}**

* **地址删除：**

**void Net\_address\_delect() {**

**string Search\_addrs;**

**cout << "请输入需要删除的地址: ";**

**cin >> Search\_addrs;**

**if (Maps\_In.count(Search\_addrs)) {**

**Maps\_Out.erase(Maps\_In[Search\_addrs]);**

**Maps\_In.erase(Search\_addrs);**

**cout << "删除成功！" << endl;**

**} else if (Maps\_Out.count(Search\_addrs)) {**

**Maps\_In.erase(Maps\_Out[Search\_addrs]);**

**Maps\_Out.erase(Search\_addrs);**

**cout << "删除成功！" << endl;**

**} else {**

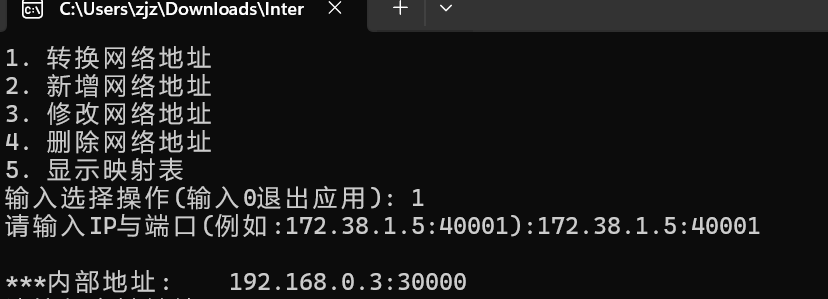
**cout << "地址未找到！" << endl;**

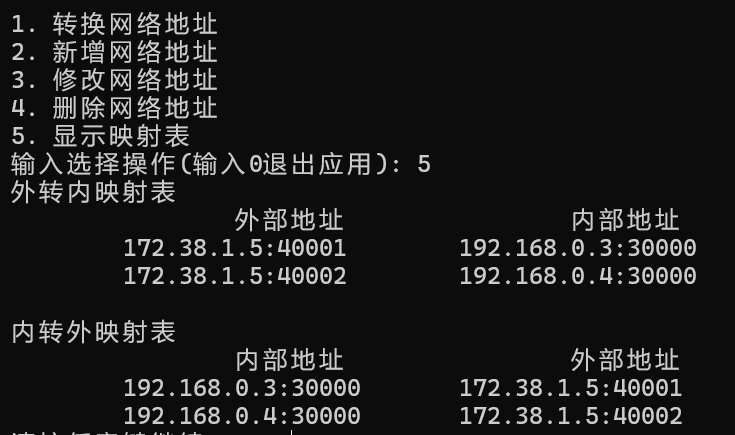
**}**

**}**

**五、实验结果与分析**

**实验结果:**





**六、小结与心得体会**

**学会了NAT转换机制.**