보고서 #7 (데이터구조 2) (기한 12/3)

2020. 12. 21

학번 : 20184646 성명 : 권석환

Table of Contents

[보고서 #7 (데이터구조 2) (기한 12/3) - 1 -](#_Toc59473622)

[[문제] - 2 -](#_Toc59473623)

[분석 : IOP 기반 분석 - 3 -](#_Toc59473624)

[알고리즘 작성 - 4 -](#_Toc59473625)

[코딩 : C 프로그램 작성 (주석 반드시 포함할 것) - 5 -](#_Toc59473626)

[테스트 : 프로그램의 올바름을 검증할 수 있도록 다양한 데이터에 대해서 테스트하고, 입력 값에 대해서 결과가 올바른지를 검증하라. - 19 -](#_Toc59473627)

[의견 : 문제 해결과정에서 느낀 점, 의견 등을 기술한다. - 21 -](#_Toc59473628)

# [문제]

A picture containing text, receipt

Description automatically generated

A picture containing calendar

Description automatically generated

# 분석 : IOP 기반 분석

|  |  |
| --- | --- |
| I | python으로 작성된 client 프로그램을 통해 출발지와 도착지를 입력한다.  해당된 내용을 c로 작성된 server로 전송한다. |
| O | 요청받은 경로를 통해 Dijkstra 알고리즘으로 최소거리를 찾을 때, 우선순위 큐의 힙을 이용하여 찾는다. 결과를 client로 전송한다. 이때, 로그기록을 서버에 남긴다. |
| P | client에서 최소 거리와 경로를 출력한다. |

알고리즘 작성

// main.c

Server를 생성한다.

port 번호를 설정해준다

소켓을 생성한다.

프로세스와 포트를 바인딩해준다.

무한루프속에서 클라이언트 요청을 리슨한다.

// heap.h

heap에대한 전반적인 내용

// heap.c

힙 삽입, 삭제에 관한 내용

// Dijkstra.h

정점에 관한 내용

// Dijkstra.c

힙을가지고 최소거리 찾기

// TCP\_Server.h

TCP 서버에 대한 전반적인 내용

// HandleTCPClient.c

클라이언트 요청을 받았을 때 처리하는 내용

// DieWithError.c

에러가 발생할 경우 종료할때 에러메시지 호출

// TCP\_Client\_python.py

python으로 tcp echo를 약간 변형한 요청을 보낸다.

# 코딩 : C 프로그램 작성 (주석 반드시 포함할 것)

서버파일 8개/ 클라이언트 파일 1개로 이루어져 있습니다.

//

// main.c

// Navigator

//

// Created by Seokhwan Kwon on 2020/12/20.

// 20184646 권석환

#include <stdio.h>

#include "heap.h"

#include "Dijkstra.h"

#include "TCP\_Server.h"

**int** main(**int** argc, **char** \*argv[])

{

**int** servSock; // Socket descriptor for server

**int** clntSock; // Socket descriptor for client

**struct** sockaddr\_in echoServAddr; // Local address

**struct** sockaddr\_in echoClntAddr; // Client address

**unsigned** **short** echoServPort; // Server port

**unsigned** **int** clntLen; // Length of client address data structure

**if**(argc != 2) // Test for correct number of arguments

{

fprintf(stderr, "Usage: %s <Server Port> \n", argv[0]);

}

// echoServPort = atoi(argv[1]); // First arg: local port

echoServPort = 8000; // First arg: local port

// Create socket for incoming connections

**if**((servSock = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP)) < 0)

DieWithError("socket() failed");

// Construct local address structure

memset(&echoServAddr, 0, **sizeof**(echoServAddr)); // Zero out structure

echoServAddr.sin\_family = AF\_INET; // Internet address family

echoServAddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); // Any incoming interface

echoServAddr.sin\_port = htons(echoServPort); // Local port

// Bind to the local address

**if**(bind(servSock, (**struct** sockaddr \*)&echoServAddr, **sizeof**(echoServAddr)) < 0)

DieWithError("bind() failed");

// Mark the socket so it will listen for incoming connections

**if**(listen(servSock, MAXPENDING) < 0)

DieWithError("listen() failed");

**for**(;;) // Run forever

{

// Set the size of the in-out parameter

clntLen = **sizeof**(echoClntAddr);

// Wait for a client to connect

**if**((clntSock = accept(servSock, (**struct** sockaddr \*)&echoClntAddr, &clntLen)) < 0)

DieWithError("accpet() failed");

// clntSock is connected to a client!

printf("\n-------------------------------------------------------\n");

printf("[ Handling client %s ] \n", inet\_ntoa(echoClntAddr.sin\_addr));

HandleTCPClient(clntSock);

}

// NOT REACHED

**return** 0;

}

//

// heap.h

// Navigator

//

// Created by Seokhwan Kwon on 2020/12/20.

// 20184646 권석환

#ifndef heap\_h

#define heap\_h

#define TRUE 1

#define FALSE 0

#define HEAP\_MAX 700

**typedef** **int** Priority; // 우선순위

**typedef** **struct** \_hData { // 경로

**int** edge[HEAP\_MAX];

**int** count;

} HData;

// Dijkstra 기준

**typedef** **struct** \_heapElem {

Priority weight;

**int** vertex;

HData route; // [!] checker로 구현한걸 옮기자!! 리스트로 하면 좋을까.?

} HeapElem;

**typedef** **struct** \_heap {

**int** numOfData;

HeapElem heapArr[HEAP\_MAX];

} Heap;

**void** Heap\_Init(Heap\* ph);

**int** H\_Is\_Empty(Heap\* ph);

// void H\_Insert\_Min(Heap\* ph, HData data, Priority weight);

**void** H\_Insert\_Min(Heap\* ph, HeapElem newElem);

HeapElem H\_Delete(Heap\* ph);

**int** Get\_Hi\_Pri\_Child\_IDX(Heap\* ph, **int** idx);

#endif /\* heap\_h \*/

//

// heap.c

// Navigator

//

// Created by Seokhwan Kwon on 2020/12/20.

// 20184646 권석환

#include "heap.h"

// 힙의 초기화

**void** Heap\_Init(Heap\* ph)

{

ph->numOfData = 0;

}

// 힙이 비어있는지 확인

**int** H\_Is\_Empty(Heap\* ph)

{

**if**(ph->numOfData == 0)

**return** **TRUE**;

**else**

**return** **FALSE**;

}

// 부모 노드의 인덱스 값 반환

**int** Get\_Parent\_IDX(**int** idx)

{

**return** idx/2;

}

// 왼쪽 자식 노드의 인덱스 값 반환

**int** Get\_L\_Child\_IDX(**int** idx)

{

**return** idx\*2;

}

// 오른쪽 자식 노드의 인덱스 값 반환

**int** Get\_R\_Child\_IDX(**int** idx)

{

**return** Get\_L\_Child\_IDX(idx) + 1;

}

// 두 개의 자식 노드 중 높은 우선순위의 자식 노드 인덱스 값 반환

// (수가 낮을수록 높은 우선순위임)

**int** Get\_Hi\_Pri\_Child\_IDX(Heap\* ph, **int** idx)

{

**if**(Get\_L\_Child\_IDX(idx) > ph->numOfData) // 자식이 없는 경우

**return** 0;

**else** **if**(Get\_L\_Child\_IDX(idx) == ph->numOfData) // 왼쪽자식노드만 있는 경우

**return** Get\_L\_Child\_IDX(idx);

**else** // 자식이 두 개인 경우

{

**if**(ph->heapArr[Get\_L\_Child\_IDX(idx)].weight > ph->heapArr[Get\_R\_Child\_IDX(idx)].weight) // 오른쪽의 우선순위가 높은 경우

**return** Get\_R\_Child\_IDX(idx);

**else**

**return** Get\_L\_Child\_IDX(idx);

}

}

// weight가 작을 때 부모노드로 올리는 삽입

// void H\_Insert\_Min(Heap\* ph, HData data, Priority weight)

**void** H\_Insert\_Min(Heap\* ph, HeapElem newElem)

{

**int** idx = ph->numOfData + 1; // [!] 나중에 ++(ph->numOfData)로 바꾸어주자

// 배열의 계산 용이성을 위하여 빼고 1부터 시작한다.

**while**(idx != 1) // 루트 노드까지 반복

{

// 맨 마지막의 노드와 우선순위를 비교한다.

**if**(newElem.weight < (ph->heapArr[Get\_Parent\_IDX(idx)].weight)) // 부모노드보다 새로 추가하는 데이터의 우선순위가 높은 경우

{

// 부모노드 데이터의 위치를 자식노드 위치와 교환한다.

ph->heapArr[idx] = ph->heapArr[Get\_Parent\_IDX(idx)];

idx = Get\_Parent\_IDX(idx);

}

**else** // 새로운 노드의 weight값이 제 위치를 찾은 경우

**break**;

}

ph->heapArr[idx] = newElem;

ph->numOfData += 1;

}

// 힙에서 데이터 삭제

HeapElem H\_Delete(Heap\* ph)

{

HeapElem retData = (ph->heapArr[1]); // 루트노드의 데이터 받아오기

HeapElem lastElem = ph->heapArr[ph->numOfData]; // 맨마지막 노드를 루트로 올리기위해 선언

**int** parentIdx = 1;

**int** childIdx;

// 루트의 자식들 중에서 높은 우선순위를 가진것을 childIndx로 옮긴다.

**while**(childIdx = Get\_Hi\_Pri\_Child\_IDX(ph, parentIdx))

{

**if**(lastElem.weight <= ph->heapArr[childIdx].weight) // 맨 마지막 노드와 루트의 자식노드중 높은 우선순위가 무엇인지 확인

**break**; // 마지막노드가 루트인 조건과 부합할 경우

// 자식 노드가 우선순위가 높을경우 옮겨준다.

ph->heapArr[parentIdx] = ph->heapArr[childIdx];

parentIdx = childIdx;

}

ph->heapArr[parentIdx] = lastElem; // 마지막 노드가 제자리를 찾은경우 위치시킨다.

ph->numOfData -= 1; // 전체 크기를 알기 위하여 -1만큼 줄여준다

**return** retData; // 맨처음 받아온 루트노드값을 리턴해준다

}

//

// Dijkstra.h

// Navigator

//

// Created by Seokhwan Kwon on 2020/12/20.

// 20184646 권석환

#ifndef Dijkstra\_h

#define Dijkstra\_h

#include "heap.h"

#define TRUE 1

#define FALSE 0

#define MAX\_VERTICES 700

#define INF 1000000 // 무한대 (연결이 없는 경우)

**typedef** **struct** GraphType {

**int** n; // 정점의 개수

**int** weight[MAX\_VERTICES][MAX\_VERTICES];

} GraphType;

HeapElem PQueue\_Dijkstra(GraphType\* g, **int** start, **int** end);

#endif /\* Dijkstra\_h \*/

//

// Dijkstra.c

// Navigator

//

// Created by Seokhwan Kwon on 2020/12/20.

// 20184646 권석환

#include "Dijkstra.h"

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

HeapElem PQueue\_Dijkstra(GraphType\* g, **int** start, **int** end)

{

// [0] 출발지는 found를 TURE로 distance는 0로...

// 아 그냥 출발지 제외하거 넣는걸로 변경하자.

// [1] 값을 healElem으로 옮겨주자.

// [2] H\_Delete를 실행한다 is\_empty까지 반복 while문을 활용하자.

// [3] 현재 상태 출력 (스텝과, 인덱스, 거리를 표시해준다)

// [4] 방문되지 않은 정점중에서 distance값이 제일 짧은 값 u를 가져온다

// 이것은 Heap-delete로 구현 가능

// [5] 꺼내온 경로를 found로 바꾸어준다 // 어차피 heap tree에서 꺼낸 것이니 found는 무의미 하지 않는가?

// [6] 그다음으로 짧은 경로 w를 가져온다.

// [7] w의 거리와 u->w의 거리 중 짧은게 무엇인지 비교한다

// [8] u->w가 더 짧은 경우 w의 최소 거리를 갱신해준다.

// [9] 다시 insert로 자료를 넣어준다.

HeapElem u; // 확인된 정점

HeapElem w; // 도착 정점

// 전체 노드의 개수만큼 heapElem 생성하고

Heap heap, support\_Heap;

HeapElem ret[g->n];

Heap\_Init(&heap);

Heap\_Init(&support\_Heap);

**for**(**int** i = 0; i < g->n; i++) // 초기화

{

// start에서 시작하여 가장 짧은길 찾기

u.vertex = i;

u.weight = g->weight[start][i];

u.route.count = 1;

u.route.edge[0] = i; // 도착지가 0번지로

H\_Insert\_Min(&heap, u); // [!] i를 나중에 checker로 구현한거 옮기자

}

**while**(!H\_Is\_Empty(&heap))

{

u = H\_Delete(&heap);

// distance[u.vertex] = u.weight;

ret[u.vertex] = u; // 결정된 값을 넣어준다.

**while**(!H\_Is\_Empty(&heap))

{

w = H\_Delete(&heap);

**if**(u.weight + g->weight[u.vertex][w.vertex] < w.weight)

{

w.weight = u.weight + g->weight[u.vertex][w.vertex];

w.route.count = 1;

**for**(**int** i = 0; i < u.route.count; i++)

w.route.edge[(w.route.count)++] = u.route.edge[i];

}

H\_Insert\_Min(&support\_Heap, w);

}

**while**(!H\_Is\_Empty(&support\_Heap)) // heap에서 u를 제외한 나머지를 복귀시켜준다.

{

w = H\_Delete(&support\_Heap);

H\_Insert\_Min(&heap, w);

}

}

**return** ret[end];

}

//

// TCP\_Server.h

// Navigator

//

// Created by Seokhwan Kwon on 2020/12/20.

// 20184646 권석환

#ifndef TCP\_Server\_h

#define TCP\_Server\_h

#include <stdio.h> // for printf() and fprintf()

#include <sys/socket.h> // for socket(), bind(), and connect()

#include <arpa/inet.h> // for sockaddr\_in and inet\_ntoa()

#include <stdlib.h> // for atoi() and exit()

#include <string.h> // for memset()

#include <unistd.h> // for close()

#define MAXPENDING 5 // Maximum outstanding connection requests

**void** DieWithError(**char** \*errorMessage); // Error handling function

**void** HandleTCPClient(**int** clntSocket); // TCP client handling function

#endif /\* TCP\_Server\_h \*/

//

// HandleTCPClient.c

// Navigator

//

// Created by Seokhwan Kwon on 2020/12/20.

// 20184646 권석환

#include "TCP\_Server.h"

#include "heap.h"

#include "Dijkstra.h"

#include <stdio.h> // for printf() and fprintf()

#include <sys/socket.h> // for recv() and send()

#include <unistd.h> // for close()

#include <string.h> // for str\_\_\_()

#include <stdlib.h> // for itoa()

#define RCVBUFSIZE 32 // Size of receive buffer

#define CHAR\_MAX 128

**void** DieWithError(**char** \*errorMessage); // Error handling function

**typedef** **char** Station\_Type;

**void** HandleTCPClient(**int** clntSocket)

{

**int** start, end;

GraphType g = { 7,

{ // 0 1 2 3 4 5 6

{ 0, 7, INF, INF, 3, 10, INF }, // 0

{ 7, 0, 4, 10, 2, 6, INF }, // 1

{ INF, 4, 0, 2, INF, INF, INF }, // 2

{ INF, 10, 2, 0, 11, 9, 4 }, // 3

{ 3, 2, INF, 11, 0, INF, 5 }, // 4

{ 10, 6, INF, 9, INF, 0, INF }, // 5

{ INF, INF, INF, 4, 5, INF, 0 } // 6

}

};

HeapElem u;

**char** stringTemp[RCVBUFSIZE]; // 반환값

**char** echoBuffer[RCVBUFSIZE]; // Buffer for echo string

**int** recvMsgSize; // Size of received message

Station\_Type station[7][40];

strcpy(station[0], "서울");

strcpy(station[1], "용산");

strcpy(station[2], "청량리");

strcpy(station[3], "부산");

strcpy(station[4], "대구");

strcpy(station[5], "대전");

strcpy(station[6], "전주");

// Receive message from client

**if**((recvMsgSize = recv(clntSocket, echoBuffer, RCVBUFSIZE, 0)) < 0)

DieWithError("recv() failed");

**char** \*s1 = malloc(**sizeof**(echoBuffer));

strcpy(s1, echoBuffer);

**char** \*ptr = strtok(s1, "/");

**for**(**int** i = 0; i < 7; i++)

{

**if**(strcmp(ptr, station[i]) == 0)

start = i;

}

// start = atoi(ptr);

**while**(ptr != **NULL**)

{

**for**(**int** i = 0; i < 7; i++)

{

**if**(strcmp(ptr, station[i]) == 0)

end = i;

}

//end = atoi(ptr);

ptr = strtok(**NULL**, "/");

}

free(s1);

printf("요청 : %d <- %d \n", end, start);

u = PQueue\_Dijkstra(&g, start, end);

// 서버 로그 기록용

printf("최소경로 : ");

**for**(**int** i = 0; i < u.route.count; i++)

printf("%d <- ", u.route.edge[i]);

printf("%d\n", start);

printf("거리 : %2d\n", u.weight);

// 전달용

strcpy(echoBuffer, "");

**for**(**int** i = 0; i < u.route.count; i++)

{

//sprintf(stringTemp, "%d", u.route.edge[i]);

//strcat(echoBuffer, stringTemp);

strcat(echoBuffer, station[u.route.edge[i]]);

strcat(echoBuffer, "<-");

}

//sprintf(stringTemp, "%d", start);

//strcat(echoBuffer, stringTemp);

strcat(echoBuffer, station[start]);

strcat(echoBuffer, "/");

sprintf(stringTemp, "%d", u.weight);

strcat(echoBuffer, stringTemp);

// Send received string and receive again util end of transmission

**while**(recvMsgSize > 0) // Zero indicates end of transmission

{

// Echo message back to client

//if(send(clntSocket, echoBuffer, strlen(echoBuffer), 0) != recvMsgSize)

// DieWithError("send() failed");

send(clntSocket, echoBuffer, strlen(echoBuffer), 0);

// See if there is more data to receive

**if**((recvMsgSize = recv(clntSocket, echoBuffer, RCVBUFSIZE, 0)) < 0)

DieWithError("recv() failed");

}

close(clntSocket); // Close client socket

}

//

// DieWithError.c

// Navigator

//

// Created by Seokhwan Kwon on 2020/12/20.

// 20184646 권석환

#include "TCP\_Server.h"

#include <stdio.h> // for perror()

#include <stdlib.h> // for exit()

**void** DieWithError(**char** \*errorMessage)

{

perror(errorMessage);

exit(1);

}

### 20184646 권석환

### Navigator Client 프로그램 python

import socket

import tkinter as tk

# main window of our calculator

window = tk.Tk()

window.title("Created by 20184646 권석환")

window.geometry("300x250")

# search function for finding the shortest path

def search():

res\_box.delete(1.0,"end")

HOST = '127.0.0.1'

PORT = 8000

path = first\_nr.get() + '/' + second\_nr.get()

path = path.encode('UTF-8')

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

s.connect((HOST, PORT))

s.sendall(path)

data = s.recv(1024)

res\_box.insert(tk.INSERT, data.decode('UTF-8'))

# all elements inside the window

lbl\_nr\_one = tk.Label(window, text="Enter the START:", bg="green")

lbl\_nr\_one.pack(padx=5, pady=5)

first\_nr = tk.Entry(window, width=10)

first\_nr.pack(padx=5, pady=5)

lbl\_nr\_two = tk.Label(window, text="Enter the END:", bg="red")

lbl\_nr\_two.pack(padx=10, pady=5)

second\_nr = tk.Entry(window, width=10)

second\_nr.pack(padx=10, pady=5)

button\_search = tk.Button(window, text="Search shortest path!", command=search)

button\_search.pack()

lbl\_res = tk.Label(window, text="경로 / 거리")

lbl\_res.pack(padx=10, pady=5)

res\_box = tk.Text(window, width=15, height=1)

res\_box.pack(padx=10, pady=5)

window.mainloop()

# 테스트 : 프로그램의 올바름을 검증할 수 있도록 다양한 데이터에 대해서 테스트하고, 입력 값에 대해서 결과가 올바른지를 검증하라.

위 소스코드는 아래 그림을 기준으로 작성 되었습니다.

Diagram

Description automatically generated

[1] 실행전 client모습과 server의 모습

Graphical user interface, application

Description automatically generated

[2] 실행 후 Client와 서버의 모습

Graphical user interface

Description automatically generated

경로와 거리가 GUI로 표현된 클라이언트 창에 올바르게 표시된 것을 볼 수 있었습니다.

server에도 로그를 남기기 위하여 접속한 ip주소와 어떤 요청을 하고 어떤 결과값을 보냈는지 로그를 남겼습니다.

# 의견 : 문제 해결과정에서 느낀 점, 의견 등을 기술한다.

C로 작성한 heap과 dijkstar알고리즘을 어떻게 GUI로 표현할까 고민하다가, C로 interface를 만드는 것보다 다른 언어로 작성하는게 더 편할 것 같아서 python의 tkinter라는 모듈을 가지고 client를 만들었습니다. 그리고 Server/Client프로그램을 만들게 되면 소켓을 만들어서 통신을 해야하는데, 이때 저는 TCP ECHO를 약간 변형하여 구현하였습니다. 요청과 응답에 대한 내용들은 UTF-8라는 유니코드 한글버전으로 보내고 받았습니다.

하면서 어려웠던 점은 정말 많은 수도권 지하철 레퍼런스를 찾고 있는 와중에 제가 습득한 파일들은 KRUSKAL기준 데이터 밖에 없었습니다. 공개데이터이기 때문에 보고서에 같이 첨부하겠습니다.

만일, 역간의 연결성을 보여줄 수 있는 엑셀파일 하나라도 있었으면 그걸 C로 읽어와서 real world에서 적용 가능한 프로그램을 만들 수 있었는데 그게 많이 아쉽습니다.