최단거리 데이크스트라

준비 기간 2019년 04월 14일 ~ 2019년 04월 20일
발표 일자 2019년 05월 01일

발표 동기

네트워크를 공부하다 최단거리 알고리즘이 네비게이션에도 사용된다는 점에 호기심이 생겨 기하와 벡터 수업시간에 발표 주제로 선정하게 됨

준비 과정

- 데이크스트라 알고리즘을 활용한 C언어 예제 조사 (인터넷, 관련서적)
- 발표에 사용할 C언어 예제 코딩
- 발표 PPT 제작
- 참여 인원 : 단독
- 준비 기간 : 7일

발표

기하와 벡터 교과 수업시간 중 학우들에게 발표

나의 변화

자신 있는 분야를 설명하는 과정에서 얻게 되는 성취감과 뿌듯함을 알게 되었고, 발표 및 스피치 능력을 향상시킬 수 있었던 계기가 됨

보고서 작성 · 심재빈

개인 발표자료

네트워크 공부 중 네비게이션에 사용되는 최단거리 알고리즘에 대해 기하와 벡터 교과 수업 시간에 발표함

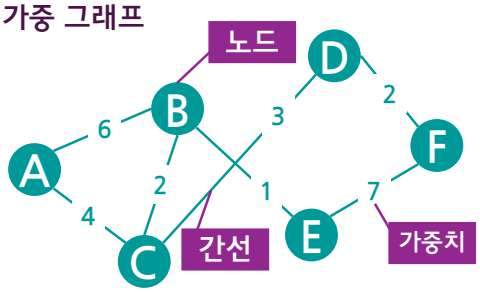
1. 발표 동기

평소 관심 있던 네트워크를 공부하다 라우팅 프로토콜 중 하나인 OSPF 에 최단거리 알고리즘인 데이크스트라 알고리즘이 사용 된다는 내용을 보고 원리가 궁금해졌는데, 이 알고리즘이 기하학을 바탕으로 만들어졌다는 점을 통해 발표 주제로 선정하였다.

발표 준비를 위해 위키백과를 참고하였으나 고등학생 수준에서는 이해하기 어려운 내용이다.
그러다 인터넷을 통해 C 언어로 개발된 데이크스트라 예제 코드를 발견하였고, 이를 디버거의 breakpoint 기능을 통해 메모리에 저장되는 변수와 식 값들을 직접 확인해보면서 알고리즘의 흐름을 이해할 수 있게 되었다.

2. 선행연구

■ 가중 그래프



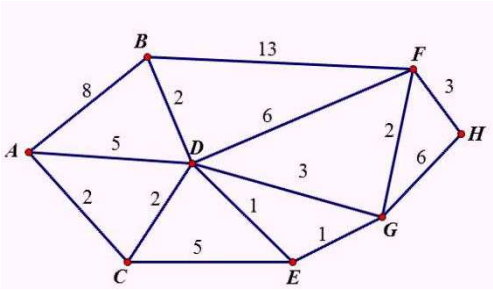
노드(꼭짓점)와 노드 사이를 잇는 변(간선)에 가중치(시간, 거리, 비용)이 주어진 그래프를 말한다.

각 용어를 지도에 치환하면 다음과 같다
노드 = 건물
간선 = 도로
가중치 = 이동하는데 걸리는 비용(거리, 시간)

■ 최단 경로 문제

지도 상의 한 지점에서 다른 지점으로 갈 때 가장 빠른 길을 찾는 것과 비슷한 문제이며, 가중 그래프에서는 구성하는 변들의 가중치 합이 최소가 되도록 하는 경로를 찾는 문제이다. 최단 경로 문제를 해결하는 알고리즘은 대표적으로 데이크스트라 알고리즘, 벨먼 - 포드 알고리즘, A* 탐색 알고리즘, 플로이드-와셜 알고리즘이 있다. 이 중 데이크스트라 알고리즘에 대해 조사하였다.

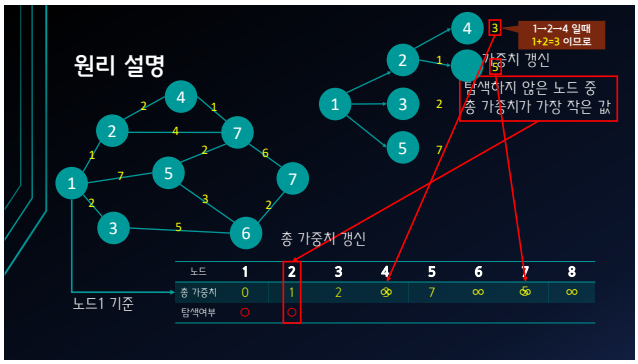
■ 데이크스트라 알고리즘

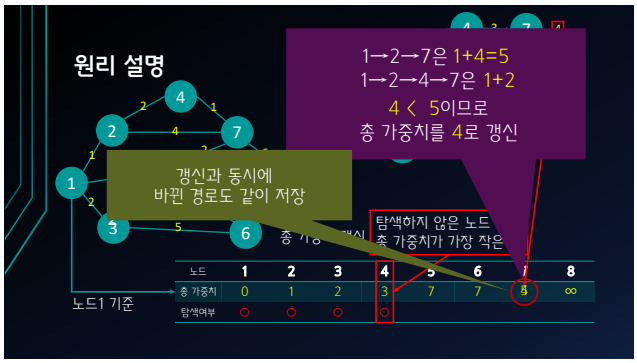


V	A	B	C	D	E	F	G	H
A	0_A	8_A	2_A	5_A	∞	∞	∞	∞
C		8_A	2_A	4_C	7_C	∞	∞	∞
D		6_D		4_C	5_D	10_D	7_D	∞
					5_D			

컴퓨터 과학자 에츠허르 데이크스트라가 1956 년에 고안한 최단 경로를 찾는 알고리즘이다.
가중 그래프에서 한 노드(꼭짓점)를 기준으로 정하고 다른 모든 노드까지의 최단 경로 트리를 만들어 최단 경로를 찾는다.

3. 예제 프로그램 코딩





데이크스트라 알고리즘 설명에 관한 내용은 PPT 애니메이션을 통해 표현하였지만,해당 내용이 실제로 적용될 수 있는지 신뢰성을 부여하기 위해 발표에서 설명한 내용을 바탕으로 C 언어를 이용해 데이크스트라 알고리즘 예제를 코딩하였다.

■ 가중 그래프 생성

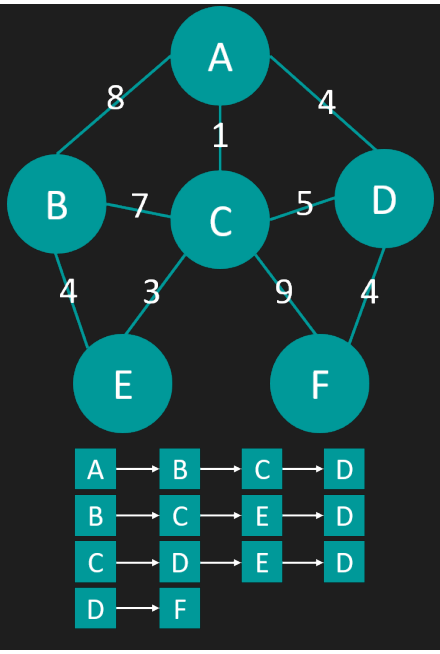
```
void initGraph() {
        // A의 연결점
        insertEdge(0, 1, 8);        // A-B
        insertEdge(0, 2, 1);        // A-C
        insertEdge(0, 3, 4);        // A-D

        // B의 연결점
        insertEdge(1, 2, 7);        // B-C
        insertEdge(1, 4, 4);        // B-E

        // C의 연결점
        insertEdge(2, 3, 5);        // C-D
        insertEdge(2, 4, 3);        // C-E
        insertEdge(2, 5, 9);        // C-F

        // D의 연결점
        insertEdge(3, 5, 4);        // D-F

        // E, F의 연결점 완료
}
```



다음과 같이 노드(알파벳)와 가중치가 주어진 가중그래프를 임의로 정의하였다.

insertEdge (시점노드, 종점노드, 가중치)

C 언어의 인접 행렬과 인접 리스트를 이용하여 다음과 같이 코드상에 가중그래프를 정의해주는 initGraph() 함수를 만들었다.

■ 데이크스트라 함수

Dijkstra() 함수

코드는 4 페이지의 Dijkstra() 함수 참조.

Dijkstra(시점, 종점)으로 사용하며, 위에서 정의한 가중그래프에서 시점과 종점을 인자로 주면
시점 → 종점으로 가는 최단거리의 가중치와, 최단거리를 구하기 위해 탐색한 경로의 개수를 반환하는 함수를 만들었다.

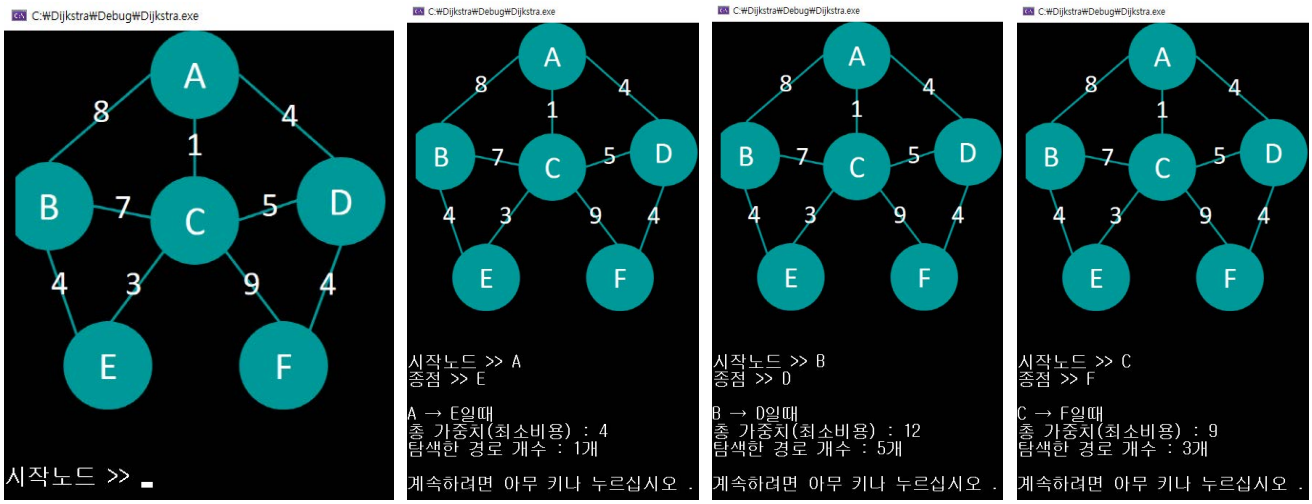
■ 가중 그래프 출력

showGraph() 함수

코드는 3 페이지의 showGraph() 함수 참조.

위에서 정의한 가중 그래프 그림을 콘솔 프로그램으로 실행되는 프로그램에 출력되도록 만들어주는 함수이다.
윈도우 API 함수를 이용해 비트맵(bmp) 이미지로 만든 가중그래프를 비트맵 메모리를 할당해 저장하고,
비트맵 이미지를 버퍼에 고속 복사하여 콘솔 창에 출력하도록 하였다.

■ 프로그램 실행 모습



프로그램 시작 모습 | A → B 입력 | B → D 입력 | C → F 입력

콘솔 프로그램으로 제작하였으며 위와 같이 가중 그래프 그림이 출력되고 시작 노드와 종점을 입력 받아 최단 가중치가 계산되도록 함.

```c
#define _CRT_SECURE_NO_WARNINGS // 보안경고 해제
#include <stdio.h>
#include <stdlib.h>
#include <Windows.h>

#define INFINITE 9999   // 거리의 초기값
#define NUM_VERTICES 6  // 정점의 개수

// 인접리스트 구조체
typedef struct Adjacency
{
    struct Adjacency *link;     // 연결
    int vertex;                 // 정점
    int weight;                 // 무게
}Adjacency;

// 인접 리스트
Adjacency *Graph_list[NUM_VERTICES];

void initGraph();
void showGraph();
void insertEdge(int, int, int);
void Dijkstra(int, int);

void main()
{
    char start, end;

    // 그래프 생성.
    initGraph();
    showGraph();
    printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

    // 시작노드, 종점 입력
    printf("시작노드 >> ");
    scanf("%c", &start);
    getchar();
    printf("종점 >> ");
    scanf("%c", &end);

    Dijkstra(start - 'A', end - 'A'); // 데이크스트라 알고리즘 실행
}

void showGraph() { // 가중그래프 비트맵 표시
    {
        HINSTANCE hInstance = GetModuleHandle(NULL);
        HWND hWnd = FindWindow("ConsoleWindowClass", NULL);
        HBITMAP hImage, hOldBitmap;

        HDC hdc = GetDC(hWnd);
        HDC hMemDC = CreateCompatibleDC(hdc);
        // 비트맵 불러오기
        hImage = (HBITMAP)LoadImage(NULL,
            TEXT("graph.bmp"),
            IMAGE_BITMAP,
            0,
            0,
            LR_LOADFROMFILE ¦ LR_CREATEDIBSECTION);
        // 비트맵 출력
        hOldBitmap = (HBITMAP)SelectObject(hMemDC, hImage);
        BitBlt(hdc, 0, 0, 400, 398, hMemDC, 0, 0, SRCCOPY);

        // 메모리 해제
        SelectObject(hMemDC, hOldBitmap);
        DeleteObject(hImage);
        DeleteDC(hMemDC);
        ReleaseDC(hWnd, hdc);
    }
}

// 가중그래프 초기화 (간선 수만큼 연결)
void initGraph() {
    // A의 연결점
    insertEdge(0, 1, 8);    // A-B
    insertEdge(0, 2, 1);    // A-C
    insertEdge(0, 3, 4);    // A-D

    // B의 연결점
    insertEdge(1, 2, 7);    // B-C
    insertEdge(1, 4, 4);    // B-E

    // C의 연결점
    insertEdge(2, 3, 5);    // C-D
    insertEdge(2, 4, 3);    // C-E
    insertEdge(2, 5, 9);    // C-F

    // D의 연결점
    insertEdge(3, 5, 4);    // D-F

    // E, F의 연결점 완료

}

void insertEdge(int start, int end, int weight)
{
```

```
9       Adjacency *node;    // 연결노드
9
98      // 양방향으로 연결
99      node = (Adjacency *)malloc(sizeof(Adjacency));
100     node->vertex = end;
101     node->weight = weight;
102     node->link = Graph_list[start];
103     Graph_list[start] = node;
104
105     // 반대쪽에서 연결
106     node = (Adjacency*)malloc(sizeof(Adjacency));
107     node->vertex = start;
108     node->weight = weight;
109     node->link = Graph_list[end];
110     Graph_list[end] = node;
111 }
112
113 void Dijkstra(int start, int end){
114
115     int distance[NUM_VERTICES];     // 거리
116     int pathcnt[NUM_VERTICES];      // 경로 수
117     int check[NUM_VERTICES];        // 가방 안에 있는지 여부
118     Adjacency *tmp;                 // 최근에 들어온 정점을 저장하기 위한 구조체 포인터
119     int cycle, min, now, i;         // cycle : 작업 횟수,
120                                     // min : 최단거리를 위한 비교값,
121                                     // now : 최근에 들어온 정점
122
123     // 초기화 과정
124     for (i = 0; i < NUM_VERTICES; i++)
125     {
126         distance[i] = INFINITE;     // 거리를 무한으로 초기화
127         pathcnt[i] = 0;             // 경로수를 0으로 초기화
128         check[i] = 0;               // PQ에 있는지 없는지 여부 ( PQ에 들어있다면 0 )
129     }
130     distance[start] = 0;            // 초기 start 0으로 (자신까지 거리 0)
131     cycle = 0;
132
133     while (cycle < NUM_VERTICES - 1)        // 모든 정점에 대해서
134     {
135         min = INFINITE;     // 최소값 무한으로 초기화
136         for (i = 0; i < NUM_VERTICES; i++)
137             if (distance[i] < min && !check[i])     // 가방 밖의 정점 중 거리가 최소인 정점으로부터 시작
138             {
139                 min = distance[i];
140                 now = i;    // 가방에 넣을 정점 위치
141             }
142
143         check[now] = 1;         // 가방 안에 넣기
144         tmp = Graph_list[now];  // 가방에 새로 들어온 정점의 구조체 포인터
145
146         while (tmp != NULL)
147         {
148             if (!check[tmp->vertex])    // 가방 밖의 정점이라면
149             {
150                 // 수행한 거리 = 기존 최단거리 -- 다른 경로를 통한 같은 최단거리
151                 if (min + tmp->weight == distance[tmp->vertex])
152                     pathcnt[tmp->vertex] += pathcnt[now];       // 최단경로 개수 증가
153
154                 //  수행한 거리 < 기존 최단거리 --> 간선 완화
155                 if (min + tmp->weight < distance[tmp->vertex])
156                 {
157                     distance[tmp->vertex] = min + tmp->weight;  // 최단거리 갱신
158                     if (now == start) pathcnt[tmp->vertex] = 1;             // 만약 시작점에서의 연결점이라면 경로수 1
159                     else            pathcnt[tmp->vertex] = pathcnt[now];    // 그 이외에는 부모위치의 경로수
160                 }
161             }   // if
162             tmp = tmp->link;    // 다음 인접 정점, 간선 검사
163         }   // while
164
165         cycle++;
166         if (now == end) break;      // 현재 정점이 도착 정점과 같다면 종료
167     }   // while
168
169     // 결과물 출력
170     printf("\n%c → %c일때 \n총 가중치(최소비용) : %d\n탐색한 경로 개수 : %d개\n\n", start + 'A', end + 'A', distance[end], pathcnt[end]);
171     system("pause"); // 콘솔 창 정지
172 }
```