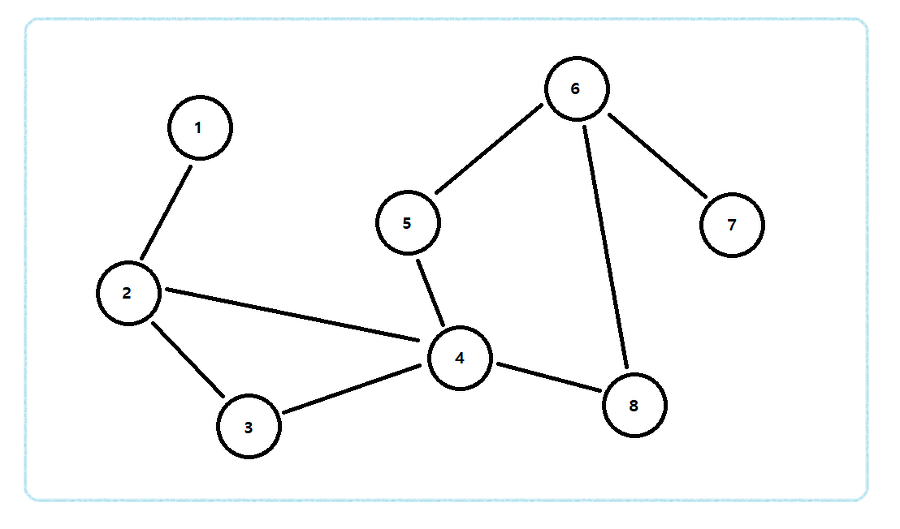
**알고리즘 프로젝트**

**(OBST , Bridge Test)**

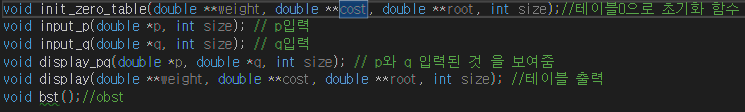


|  |  |  |
| --- | --- | --- |
| 이름 | 학번 | 학과 |
| 이정찬 | 2014154031 | 컴퓨터공학과 |
| 박현규 | 2014154020 | 컴퓨터공학과 |

**OBST(Optimal Binary Search Tree)**

접근 방법

정점의 개수를 입력 받고 p, q ,cost table, weight table, root table을 입력 받은 개수의 값에 따라 동적 할당하여 p, q 데이터 입력과 각 테이블을 0으로 초기화 시킨 후 p, q 데이터 값에 따라서 cost table, weight table, root table을 구하는 방식으로 접근



**void init\_zero\_table(double \*\*, double \*\*, double \*\*, int)**

**반환 형: void**

**역할: table, weight, root 테이블을 0으로 초기화한다.**

**void input\_p(double \*,int) , input\_q(double \*,int)**

**반환 형: void**

**역할: p와q의 값을 입력받는다**

**void display\_pq(double \*,double \*,int)**

**반환 형:void**

**역할: 입력한 p,q의 값을 출력한다**

**void display(double \*\*, double \*\*, double \*\*, int)**

**반환 형:void**

**역할: table, weight, root 테이블을 출력한다**

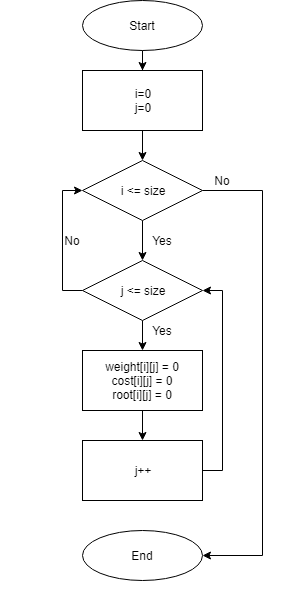
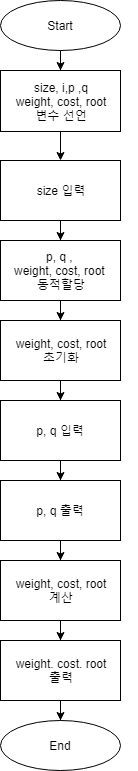
**void bst()**

**반환 형:void**

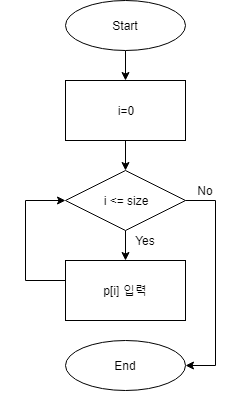
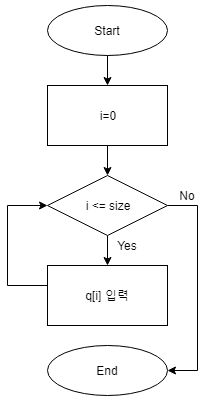
**역할: 입력받은 p,q 값을 이용해 table, weight, root 테이블을 계산한다.**

순서도

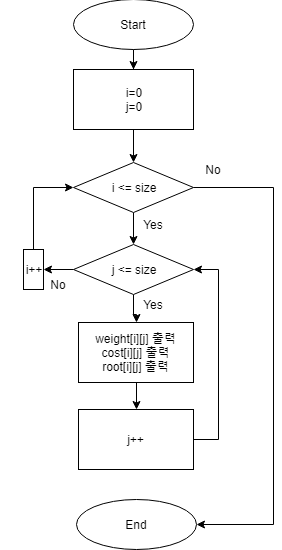
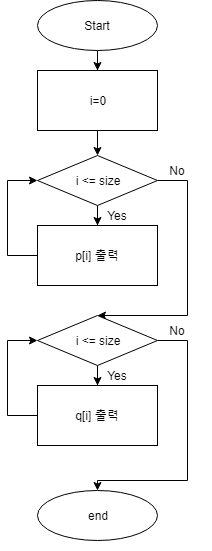
main() init\_zero\_table()

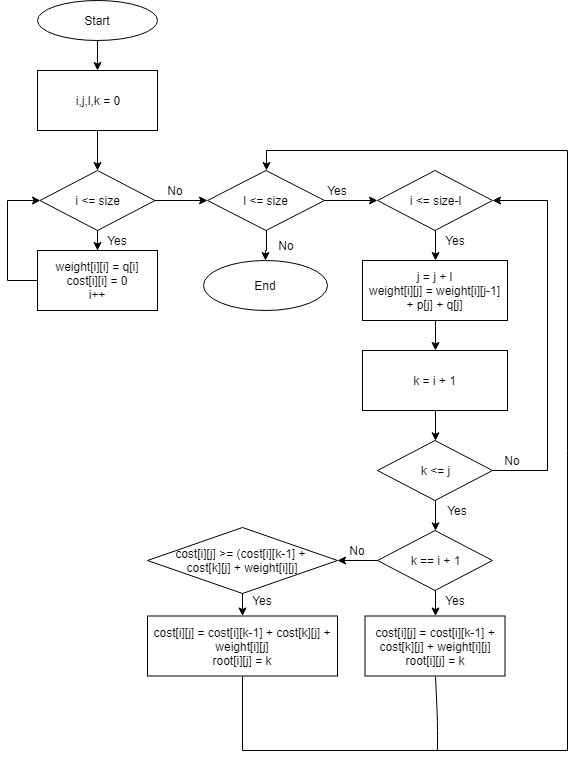


input\_q() Input\_p()

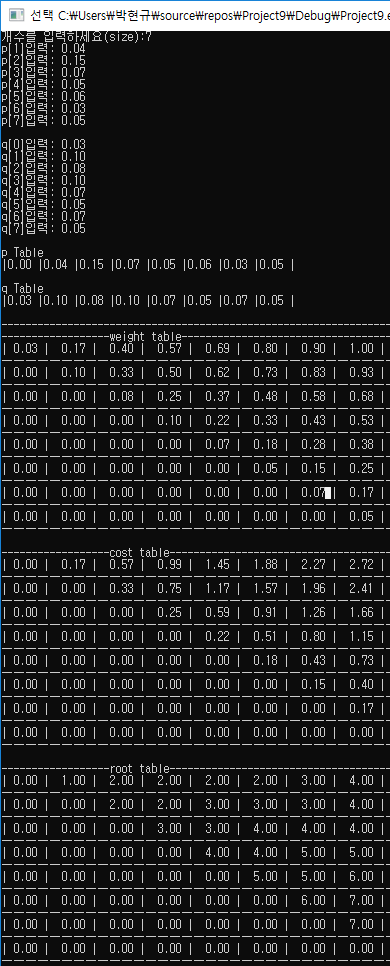


display\_pq() display()



bst()

실행 화면

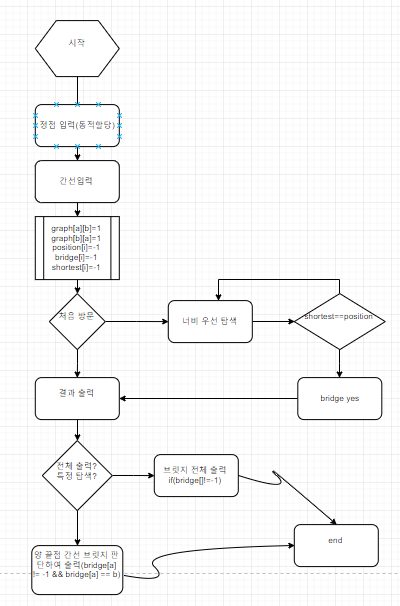


**Bridge Test**

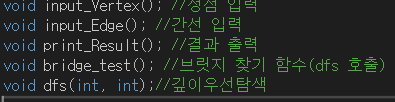
***접근 방법***: 초반에 정점의 개수를 입력 받을 수 있도록 input\_Vertex()함수를 호출 한다. 이 함수에서는 정점의 개수를 입력하고 Size변수에 넣는다. 그리고 개수만큼 현재 위치를 담을 postion과 브릿지를 판단 할 shortest 배열과 bridge의 정보를 담을 bridge 배열 그리고 그래프는 2차원 배열로 모두 동적 할당한다. 두 번째로 input\_Edge함수를 호출해서 간선의 개수와 간선들을 입력 받는다. 양 끝점을 입력하면 2차원 배열로 그래프를 만들어 나간다. Bridge\_test함수를 호출 하여 초기 카운트(방문횟수)는 0으로 초기화 각 만든 배열 모두 -1로 초기화한다. 만약 position이 -1이면 첫 방문으로 깊이우선탐색을 한다(dfs호출). dfs에서 방문 횟수를 증가시켜주고 그래프가 있고 아직 방문 하지 않았다면 dfs를 재귀호출해서 목적지 정점w로 깊이우선탐색을 한다. shortest에는 계속 위치를 담지만 첫 방문 일 때 정점 v와 w의 최소값으로 갱신한다. 만약 shortest와 현재 위치가 같으면 브릿지이다. 첫 방문이 아니면 정점 v의 shortest와 정점 w의 위치의 최소값을 shortest에 갱신한다. 즉 이들은 브릿지가 아니다. 출력시 전체 브릿지 출력은 bridge의 담긴 정보로 인덱스는 앞 정점 값은 뒤 정점이다. 2 버전 판단 시에는 **(bridge[a] != -1 && bridge[a] == b)** 조건을 통해 양 끝점 a와 b를 입력 받았을 때 bridge의 인덱스 a 의 값이 b와 같으면 브릿지이다.

**정점 간선 입력 받고 깊이을 통해서 브릿지를 판단하여 브릿지 전체 버전과 브릿지 판단 버전을 두개로 접근 했다.**

***순서도***



***함수***



**Void input\_Vertex()**

**반환 형: void**

**역할: 정점을 입력한다.**

**Void input\_Edge()**

**반환 형: void**

**역할: 간선 개수와 간선을 입력한다.**

**void print\_result()**

**반환 형:void**

**역할: 버전 1과 버전 2를 나누어서 출력하게 한다.(select)**

**Void bridge\_test()**

**반환 형:void**

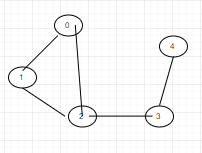
**역할: 브릿지를 본격 찾는 함수(dfs를 호출)**

**Void dfs(int, int)**

**반환 형:void**

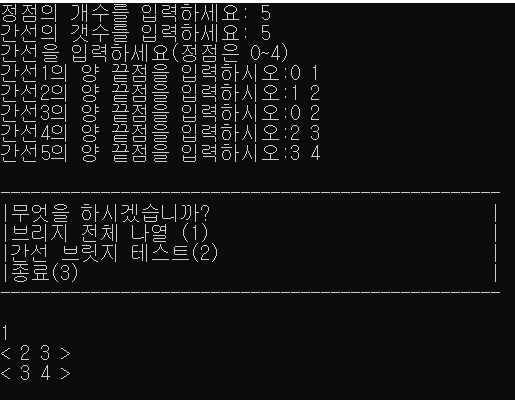
**인자:int int(정점)**

**역할: 깊이 우선 탐색을 하여 브릿지를 찾는다.**

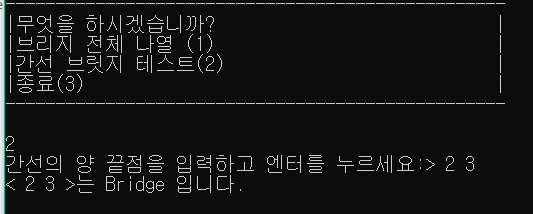


***출력 화면***

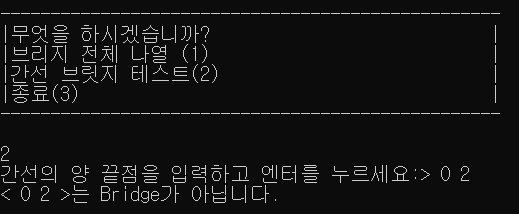
**<정점 간선 입력 후 브릿지 전체 출력>**



**<정점 간선 입력 후 간선 브릿지 테스트>**



**<간선 브릿지 테스트 브릿지 아닌 경우>**



**어려웠던 점과 극복 방법**

과제를 하면서 OBST는 실제로 수업시간에 과제를 내줬기 때문에 어렵지 않게 짤 수 있었지만 테이블을 출력하는 부분에서 오랜만에 2차원 배열을 써서 그런지 많이 헷갈렸다. 하지만 대학교 1학년 때 테이블 출력 했던 부분을 참고를 하여 테이블 출력하게 했다.

BridgeTest는 처음에 어떻게 접근 할지를 몰랐다. 브릿지라는 용어도 몰랐지만 구글링을 통해 브릿지가 단절선이라는 것을 알고 접근 법도 깊이우선탐색을 통해 하면 된다는 것을 알았다.

그래프와 실제 깊이우선탐색을 어떻게 매핑 시킬지 어려웠지만 팀원과 함께 머리를 맞대고 짜고 구글링을 통해 참고를 해서 쉽게 풀릴 수 있었다. 또한 간선 브릿지 테스를 할 때 bridge가 배열이여서 판단하는데 인덱스와 값을 어떻게 적절히 비교할지 고민 됐었지만 일일이 다 그려가면서 해보니까 잘 풀렸다.

**소감**

이 번 과제를 통해 알고리즘을 이해 할 수 있었고 스스로 알고리즘을 짜는 시간이 였기 때문에 어려워도 많은 것을 얻을 수 있었다.

그리고 팀워크가 이렇게 좋은지 깨닫는 시간이었다. 안풀리는데 서로 맞대고 풀다 보면 풀리는 순간 힘들었지만 기분이 좋았다.