10.1 그래프?

그래프의 소개 객체 사이의 연결관계를 표현할 수 있는 자료구조

전기소자 표현, 프로세스와 자원들이 어떻게 연관 표현에 사용

그래프의 역사 오일러의 문제 모든 다리를 한번만 건너 출발점으로 되돌아올수 있는가?

노드 정점

특정지역

간선 다리

오일러 경로 존재하는 모든 간선을 한번만 한번만 통과하면서 처음 정점으로 되돌아오는 경로

그래프로 표현할 수 있는 것들 도로 도로의 교차점, 일방통행 길

미로

선수과목

10.2 그래프의 정의와 용어

그래프의 정의 정점과 간선들의 유한집합

수학적 표현 G = (V, E)

V(G) 그래프 G의 정점들의 집합

E(G) 그래프 G의 간선들의 집합

ex) V(G1) = { 0,1,2,3 }

E(G1) = { (0,1), (0,2), (0,3), (1,2) }

무방향 그래프와 방향 그래프 무방향 그래프 양방향 간선

ex) (B, A) = (A, B)

방향 그래프 단방향 간선

ex) <B, A> != <A, B>

네트워크 간선에 가중치 할당 🡪 간선이 연결강도까지 표현

가중치 그래프 간선에 비용이나 가중치가 할당된 그래프

= 네트워크

표현할 수 있는 것들 도로의 길이, 회로소자의 용량, 통신망의 사용료

부분 그래프 어떤 그래프의 정점의 일부와 간선의 일부로 이루어진 그래프

그래프의 부분집합

정점의 차수 인접 정점 간선에 의해 직접 연결된 정점

차수 그 정점의 인접 정점의 수

ex) 0 – 3

| ＼

1 2

0의 인접정접 1,2,3

0의 차수 = 3

무방향 그래프에서 모든 정점의 차수를 합하면 간선 수의 2배가 됨

방향 그래프 진입차수 외부에서 오는 간선의 개수

진출차수 외부로 향하는 간선의 개수

경로 어떤 정점부터 어떤 정점까지의 정점의 나열

나열된 정점들 간에는 간선이 존재해야 함

단순 경로 경로 중 반복되는 간선이 없는 경우

사이클 단순경로의 시작 정점과 종료 정점이 같은 경우

연결 그래프 무방향 그래프 G에 있는 모든 정점쌍에 대해 항상 경로가 존재하는 그래프

ex) v1 – v2 – v3

비연결 그래프 연결그래프가 아닌 그래프

완전 그래프 그래프에 속해 있는 모든 정점들이 서로 연결되어 있는 그래프

정점의 수 = n 하나의 정점은 n-1개의 다른 정점으로 연결됨 🡪 간선의 수 n(n-1)/2

그래프 추상 데이터타입 ADT 10.1 객체: 정점의 집합과 간선의 집합

연산

create\_graph() ::= 그래프를 생성한다

init(g) ::= 그래프 g를 초기화한다

insert\_vertex(g,v) ::= 그래프 g에 정점 v를 삽입한다

insert\_edge(g, u, v) ::= 그래프 g에 간선 (u,v)를 삽입한다

delete\_vertex(g,v) ::= 그래프 g에 정점 v를 삭제한다

delete\_edge(g, u, v) ::= 그래프 g에 간선 (u, v)를 삭제한다

is\_empty(g) ::= 그래프 g가 공백상태인지 확인한다

adgacent(v) ::= 정점 v에 인접한 정점들의 리스트를 반환한다

destroy\_graph(g) ::= 그래프 g를 제거한다

Quiz 1. V(G) = { 0,1,2,3,4 }

E(G) = { (0,1),(0,2),(0,3),(1,2),(1,4),(2,3),(3,4) }

0–3, 1-3, 2-3, 3-3, 4-2

2. V(G) = { 0,2,3,4 }

E(G) = { <0,4><4,2><2,3><2,4><2,0> }

0 진입차수 1

진출차수 1

2 진입차수 1

진출차수 2

3 진입차수 1

진출차수 0

4 진입차수 2

진출차수 1

10.3 그래프의 표현 방법

그래프 표현 방법 인접행렬 2차원 배열을 사용해 그래프 표현

인접리스트 연결리스트를 이용해 그래프 표현

인접행렬 그래프의 정점 수가 n이라면 nxn의 2차원 배열 선언

항상 n2개의 메모리공간 필요

간선이 많은 밀집그래프 표현에 적합 but 간선이 적은 희소그래프 표현에 부적합

규칙 if(간선(i, j)가 그래프에 존재) M[i][j] = 1

else M[i][j] = 0

무방향 그래프의 인접행렬 대칭형태

(i, j)=(j, i)

간선의 존재 유무을 알아보는 시간복잡도 O(1)

정점의 차수를 알아보는 시간복잡도 O(n)

모든 간선의 수를 알아보는 시간복잡도 O(n2)

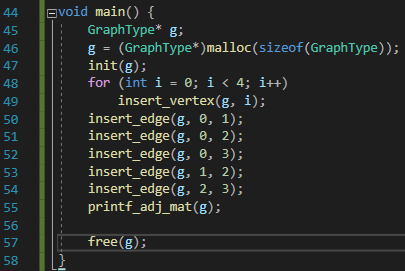
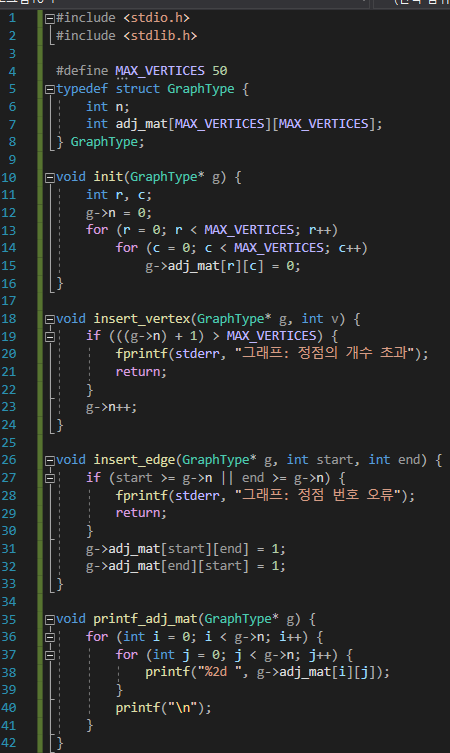
인접행렬을 이용한 그래프 추상 데이터 타입의 구현 구조체로 그래프 관련 변수 선언 그래프에 존재하는 정점의 개수 n개 필요

2차원배열 선언

정점 삽입연산 구조체의 n++

간선 삽입연산 무방향 그래프 m[i][j] = m[j][i] =1

방향 그래프 m[i][j] = 1

프로그램10.1 

인접리스트 각각의 정점에 인접한 정점들을 연결리스트로 표시

각 연결리스트의 노드들은 인접정점을 저장

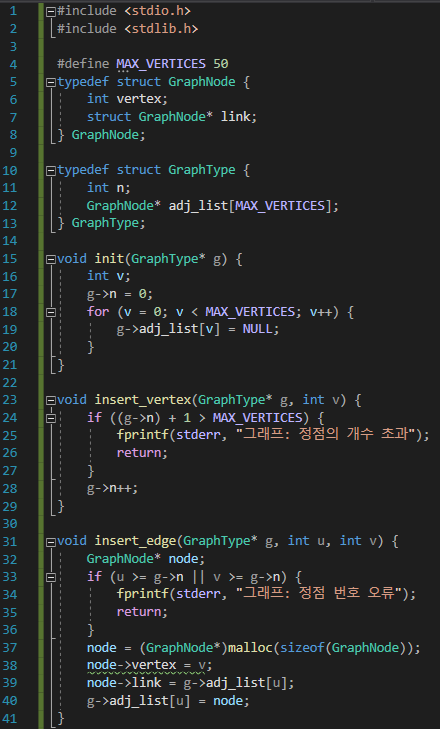
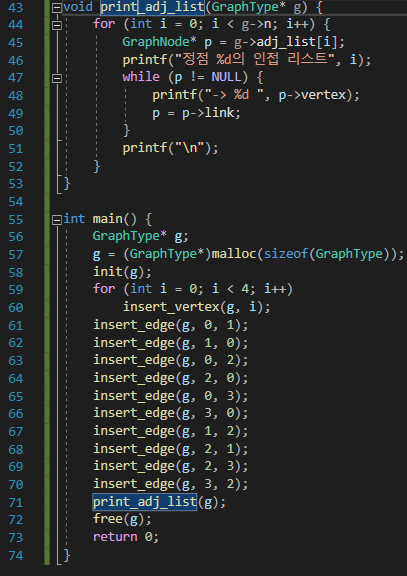
각 연결리스트들은 헤더노드를 가지고 있음 🡪 헤더노드는 하나의 배열로 구성됨 🡪 정점의 번호만 알면 번호를 인덱스로 해 접근가능

무방향 그래프 (i, j) i🡪j, j🡪i로 두 번 표현

정점의 수는 n개, 간선의 수는 e개인 무방향 그래프 표현 n개의 연결리스트, n개의 헤더노드, 2e개의 노드 필요

간선의 수가 적은 희소그래프 표현에 적합

전체 간선의 수를 알아보는 시간복잡도 O(n+e)

인접리스트를 이용한 그래프 추상 데이터 타입의 구현 프로그램10.2  

Quiz 1. 인접행렬 for (int i = 0; i < 4; i++)

insert\_vertex(g, i);

insert\_edge(g, 0, 1);

insert\_edge(g, 0, 2);

insert\_edge(g, 0, 3);

insert\_edge(g, 1, 2);

insert\_edge(g, 2, 3);

insert\_edge(g, 1, 4);

insert\_edge(g, 3, 4);

인접리스트 for (int i = 0; i < 4; i++)

insert\_vertex(g, i);

insert\_edge(g, 0, 1);

insert\_edge(g, 1, 0);

insert\_edge(g, 0, 2);

insert\_edge(g, 2, 0);

insert\_edge(g, 0, 3);

insert\_edge(g, 3, 0);

insert\_edge(g, 1, 2);

insert\_edge(g, 2, 1);

insert\_edge(g, 2, 3);

insert\_edge(g, 3, 2);

insert\_edge(g, 1, 4);

insert\_edge(g, 4, 1);

insert\_edge(g, 3, 4);

insert\_edge(g, 4, 3);

2. 인접행렬 insert\_edge(g, 0, 4);

insert\_edge(g, 2, 0);

insert\_edge(g, 2, 4);

insert\_edge(g, 4, 2);

insert\_edge(g, 2, 3);

인접리스트 insert\_edge(g, 0, 4);

insert\_edge(g, 2, 0);

insert\_edge(g, 2, 4);

insert\_edge(g, 4, 2);

insert\_edge(g, 2, 3);

3. 인접행렬 int get\_degree(GraphType\* g, int v) {

int res = 0;

for (int i = 0; i < g->n; i++) {

if (g->adj\_mat[v][i] == 1) res++;

}

return res;

}

인접리스트 int get\_degree(GraphType\* g, int v) {

if (v >= g->n)

return -1;

if (g->adj\_list[v] == NULL) return 0;

GraphNode\* n = g->adj\_list[v];

int res = 1;

for (; n->link != NULL; n = n->link) {

res++;

}

return res;

}

10.4 그래프의 탐색

그래프 탐색 하나의 정점으로부터 시작해 차례대로 모든 정점들을 한 번씩 방문하는 것

특정정점에서 다른 정점으로 갈 수 있는지 여부 확인

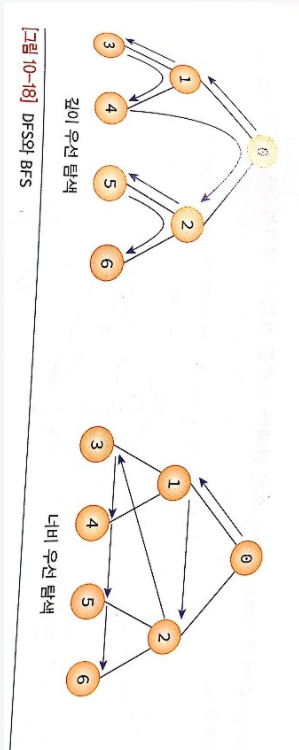
ex) 전자회로 검사

깊이 우선 탐색 DFS, Depth First Search

시작 정점에서 한 방향으로 계속 가다가 더 이상 진행할 수 없을 시, 가장 가까운 갈림길로 되돌아와 탐색 진행

너비 우선 탐색 BFS, Breath First Search

시작 정점으로부터 가까운 정점을 먼저 방문, 멀리 떨어져 있는 정점을 나중에 방문



10.5 깊이 우선 탐색

그래프의 시작정점에서 출발, 시작정점 v를 방문했다고 표시 🡪 v에 인접한 정점들 중 아직 방문하지 않은 정점 u 선택 🡪 u를 시작정점으로 다시 탐색 시작

🡪 방문하지 않은 정점이 없다면 탐색 종료

순환 알고리즘

알고리즘10.1 depth\_first\_search(v) :

v가 방문되었다고 표시;

for(all u in (v에 인접한 정점))

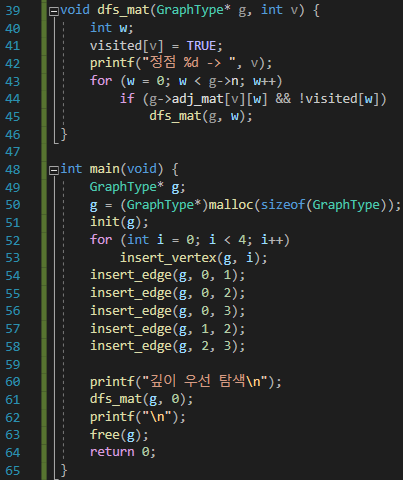
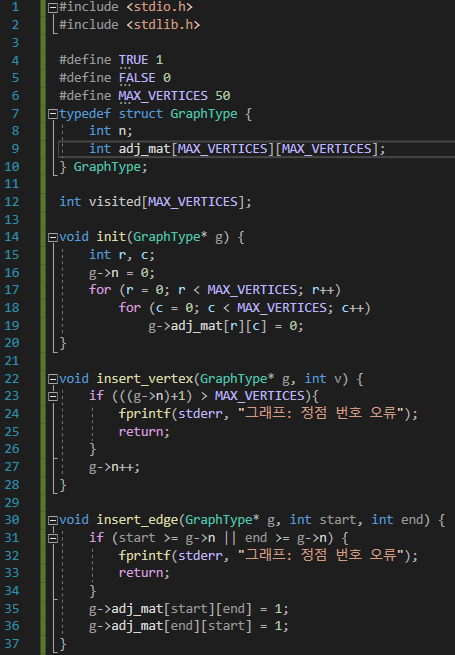
if(u가 아직 방문되지 않음)

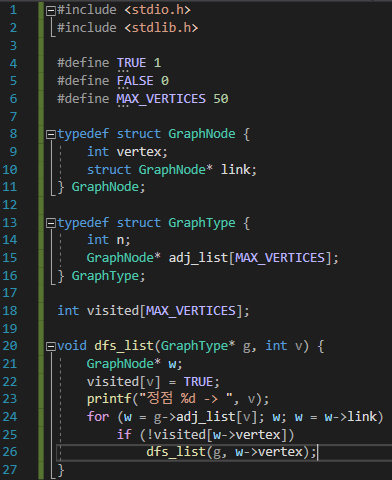
then depth\_first\_search(u);

깊이 우선 탐색의 구현(인접행렬 버전) 깊이 우선 탐색의 구현 방법 1. 순환호출

2. 스택

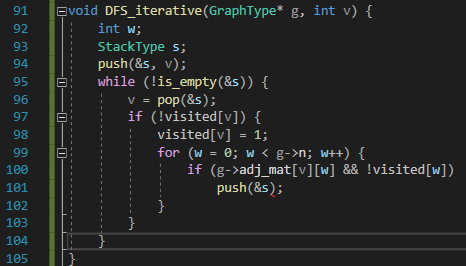
방문여부를 확인하기 위한 배열 선언

프로그램10.3 

깊이 우선 탐색의 구현(인접리스트 버전) 프로그램10.4 

명시적인 스택을 이용한 깊이 우선 탐색의 구현 명시적인 스택을 이용해 구현 가능

스택에서 1개의 정점을 꺼내 탐색 시작 🡪 정점을 방문한 후에 인접 정점들을 스택에 추가 🡪 스택에 요소가 안남을 때까지 반복

도전문제 

깊이 우선 탐색의 분석 정점의 수가 n이고 간선의 수가 e인 그래프인 경우 그래프가 인접리스트로 표현되어 있다면 O(n+e)

그래프가 인접행렬로 표현되어 있다면 O(n2)

Quiz 1. 0 🡪 1 🡪 2 🡪 1 🡪 4 🡪 3 🡪 4 🡪 1 🡪 0

10.6 너비 우선 탐색

시작정점으로부터 가까운 정점을 먼저 방문 🡪 멀리 떨어져 있는 정점을 나중에 방문

가까운 거리에 있는 정점들을 차례로 저장 후, 꺼낼 수 있는 자료 사용 = 큐

알고리즘10.2 breadth\_first\_search(v):

v가 방문되었다고 표시

큐 q에 정점 v를 삽입

while (!q) {

q에서 정점w를 삭제

for(all u in w에 인접한 정점){

if (u가 아직 방문되지 않았으면) {

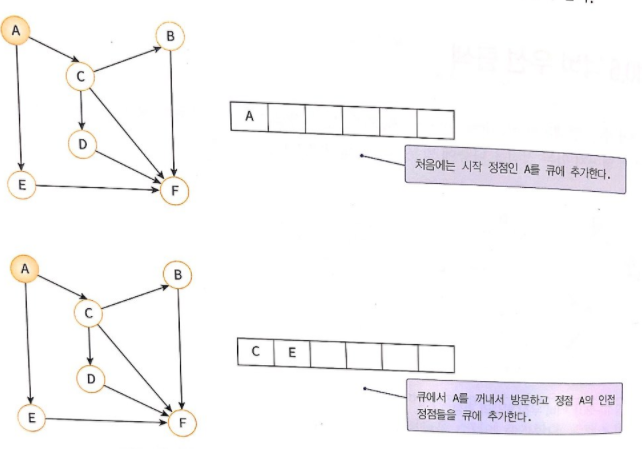
u를 큐에 삽입

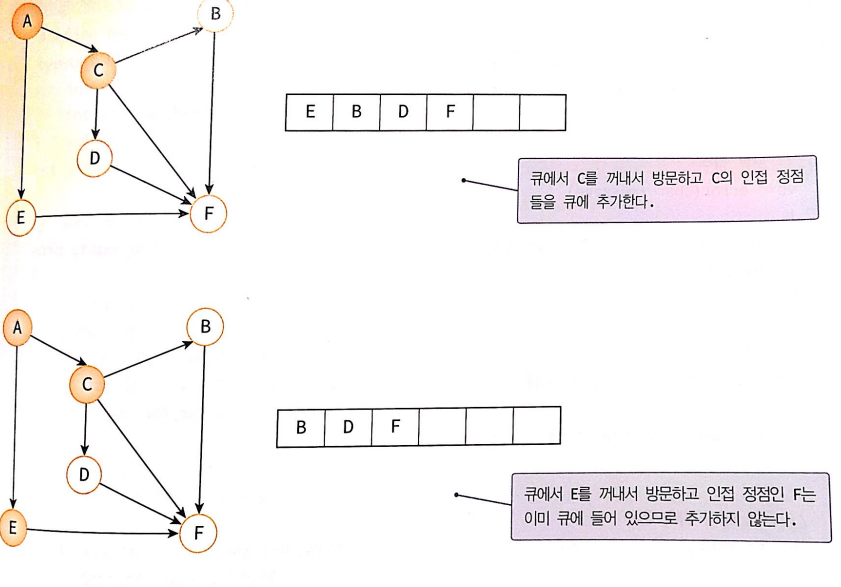
u를 방문되었다고 표시

}

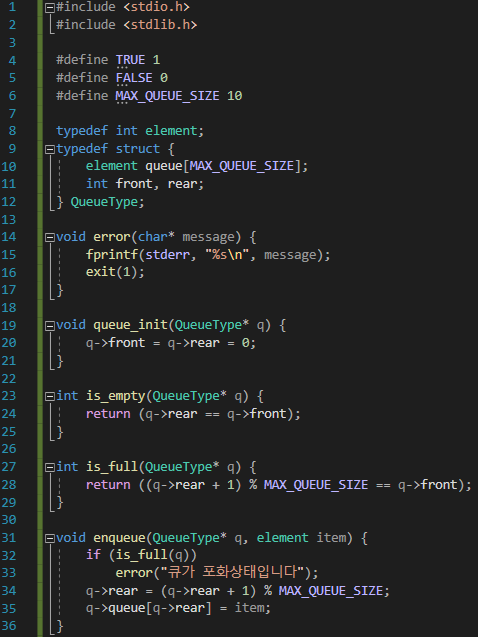
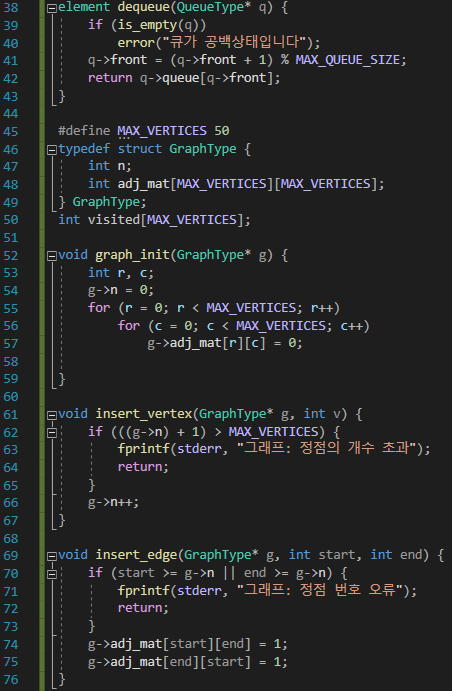
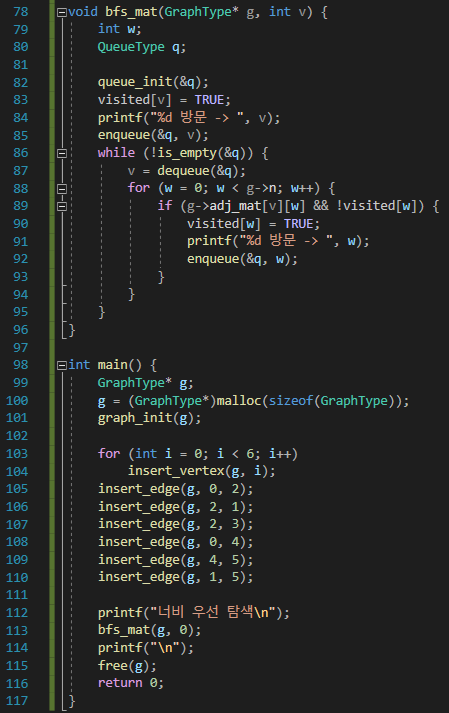
}

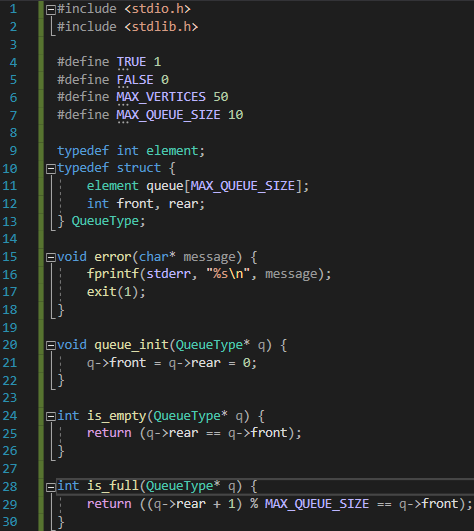
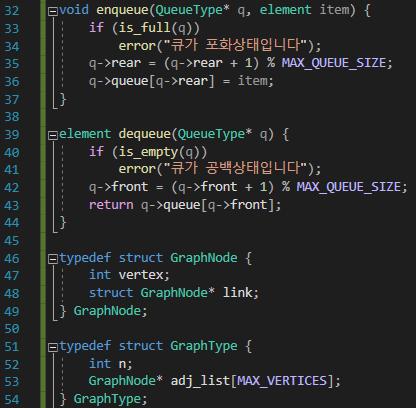
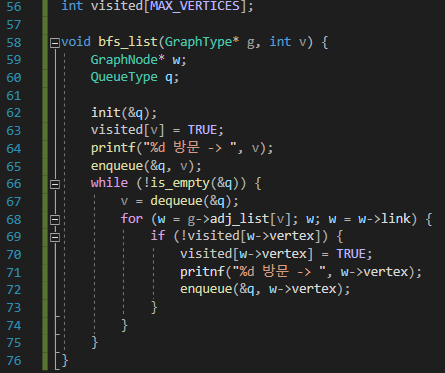
}





거리가 0인 시작 정점 방문 🡪 거리가 1인 정점 🡪 거리가 2인 정점 🡪 …

너비 우선 탐색의 구현(인접 행렬 버전) 프로그램10.5   

너비 우선 탐색의 구현(인접 리스트 버전) 프로그램10.6   

너비 우선 탐색의 분석 인접리스트 표현 O(n+e)

인접행렬 표현 O(n2)

Quiz 1. 2 🡪 01 🡪 13 🡪 34 🡪 4 🡪

연습문제

1. 1

2. 0 🡪 1

1🡪 0 🡪 2 🡪 3

2 🡪 1

3 🡪 1

3. 2

4. 방향그래프라면 1, 무방향그래프라면 2

5. 2

6. 인접행렬 { 01001 }, { 10110 }, { 01001 }, { 01001 }, { 10110 }

인접리스트 0 🡪 1 🡪 4

1 🡪 0 🡪 2 🡪 3

2 🡪 1 🡪 4

3 🡪 1 🡪 4

4 🡪 0 🡪 2 🡪 3

7. (1) 0 1/3

1 2/2

2 3/1

3 2/2

4 3/2

5 0/1

(2) 0 { 1,3 }

1 { 2 }

2 { 4 }

3 { 0,4 }

4 { 1 }

5

(3) { 0,50,45,10,0,0 }, { 0,0,10,15,0,0 }, { 0,0,0,0,30,0 }, { 20,0,0,0,15,0 }, { 0,20,35,0,0,0 }, { 0,0,0,0,3,0 }

(4) 0 🡪 1,50 🡪 2,45 🡪 3, 10

1 🡪 2,10 🡪 3,15

2 🡪 4,30

3 🡪 0,20 🡪 4,15

4 🡪 1, 20 🡪 2,35

5 🡪 4,3

(5) 0130 85

024130 130

030 30

1301 85

1241 60

130241 130

242 65

2412 60

241302 130

3412 50

3013 85

302413 130

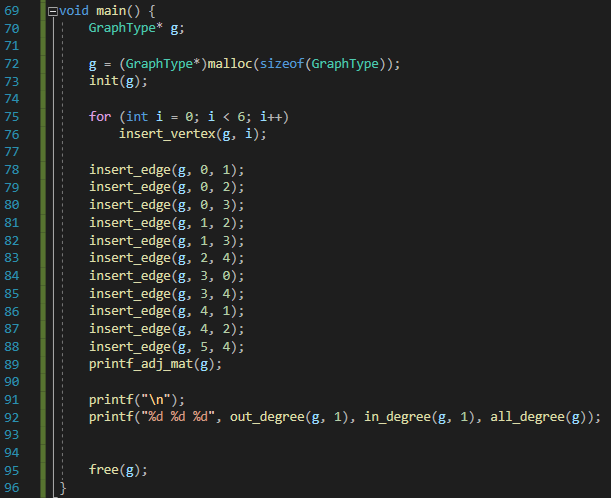
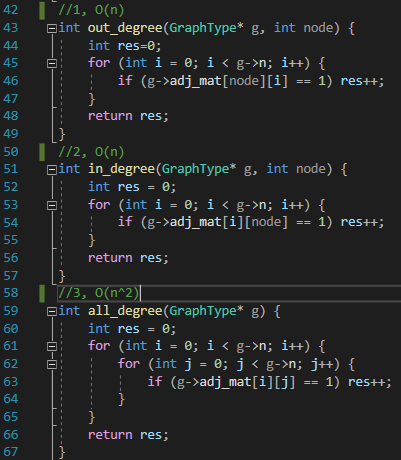
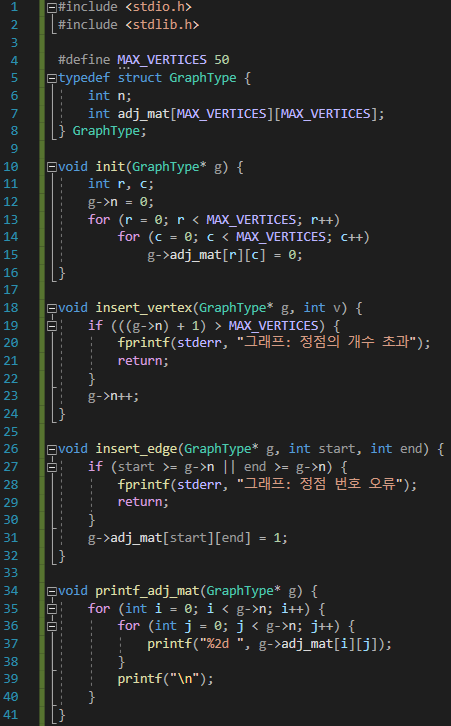
4124 60

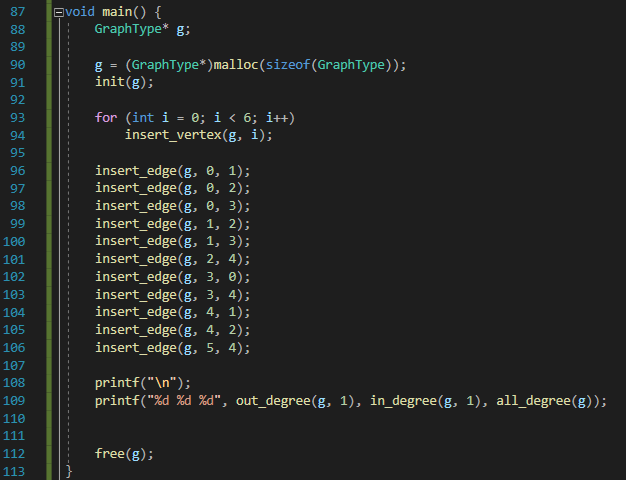
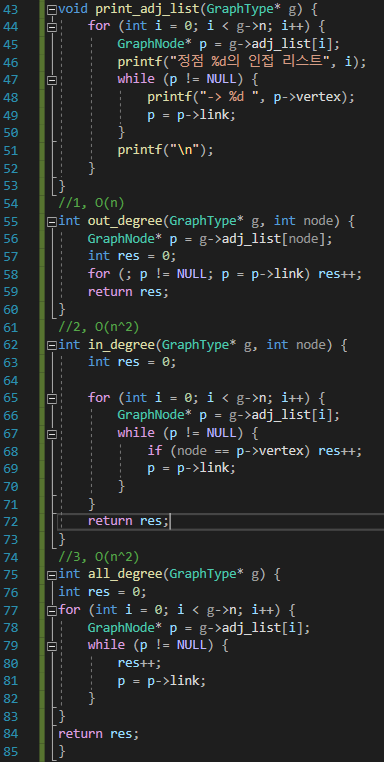
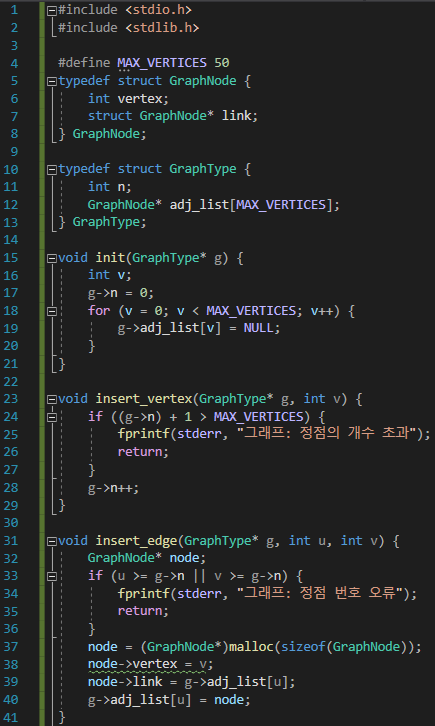
424 65

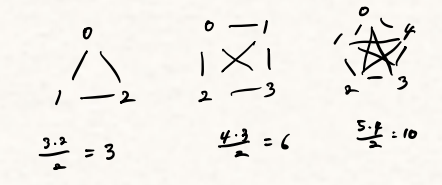
413024 130

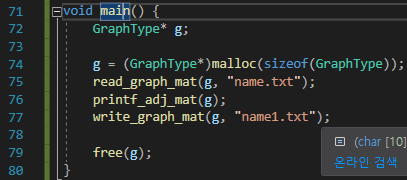
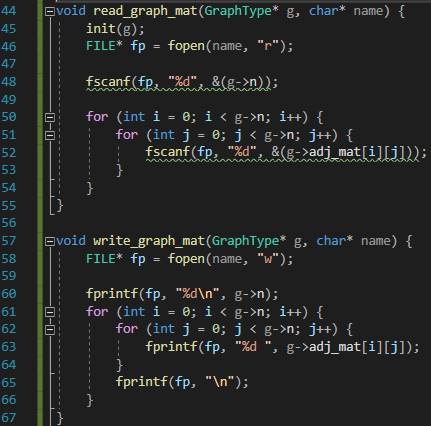
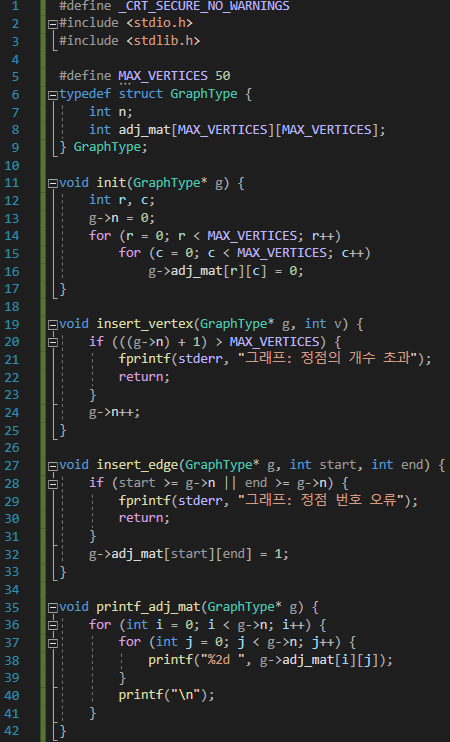
4134 50

8. 

9. 

10. 

11. 

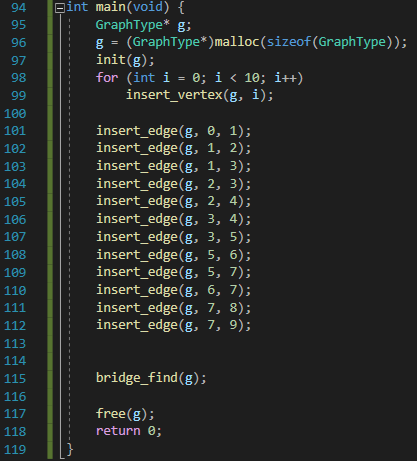
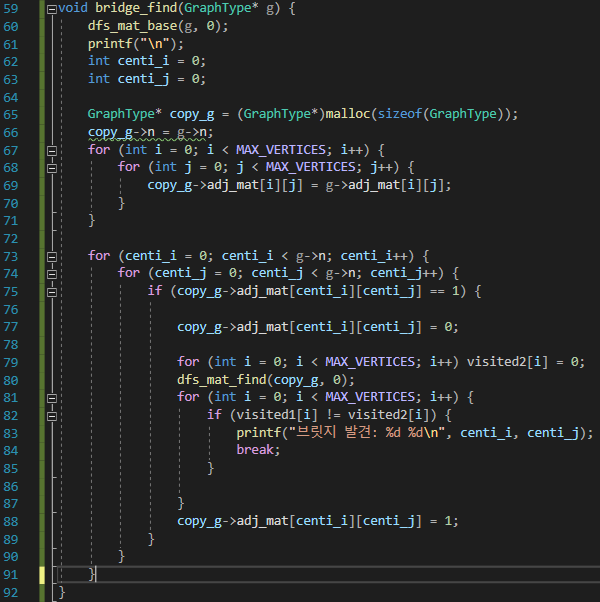
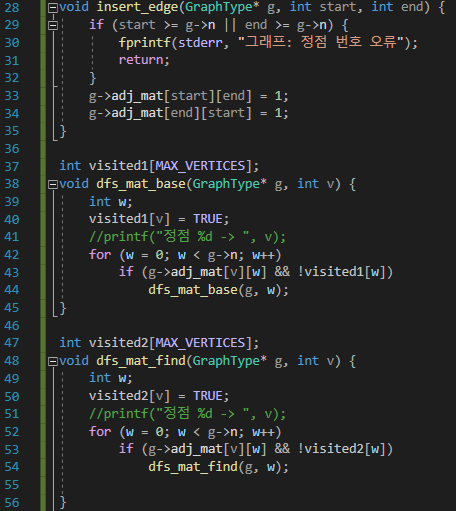
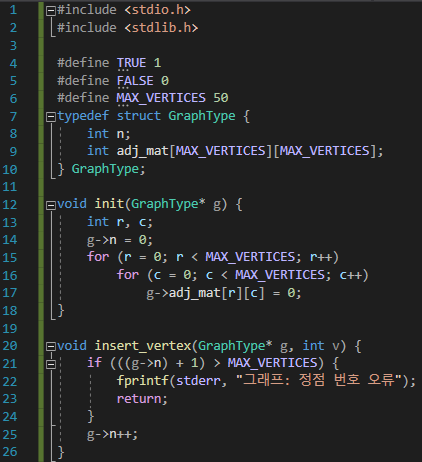
12. 

13. (1) 3 1 0 2 4 5 6 7 8 9

(2) 6 5 3 1 0 2 4 7 8 9

(3) 3 1 4 5 0 2 6 7 8 9

(4) 6 5 7 3 8 9 1 4 0 2

14. 

15. A B E G F C D