11.1 최소 비용 신장 트리

신장 트리 그래프 내의 모든 정점을 포함하는 트리

모든 정점 연결, 사이클 포함 X 🡪 n개의 정점, n-1개의 간선

탐색 도중 사용된 간선만 모으면 만들 수 있음, 사용한 간선만 표시 알고리즘 11.1 depth\_first\_search(v):

v를 방문햇다고 표시

for(all u in (v에 인접한 정접))

if(u가 아직 방문 안됬다면)

(v,u)를 신장트리간선이라고 표시

depth\_first\_search(u)

신장트리 = 그래프의 최소 연결 부분 그래프

최소 비용 신장 트리 신장트리 중에서 사용된 간선들의 가중치 합이 최소인 신장트리

ex) 도로, 전기회로, 통신, 배관

조건 간선의 가중치의 합이 최소이어야 함

n-1개의 간선만 이용해야 함

사이클이 포함되서는 안됨

11.2 Kruskal의 MST알고리즘

그리디 알고리즘 선택할 때마다, 그 순간 가장 좋다고 생각되는 것 선택

선택 시에는 최적의 선택이나, 최종적인 해답이 최적이라는 보장은 없음 🡪 최종해답 검증 필요

Kruskal 알고리즘 각 단계에서 사이클을 이루지 않는 최소 비용 간선 선택

그래프의 간선을 가중치의 오름차순으로 정렬 🡪 정렬된 간선 리스트에서 사이클을 형성하지 않는 간선 선택

알고리즘 11.2 kruskal(g):

E를 오름차순으로 정렬

k=0

i=0

while(i<n-1)

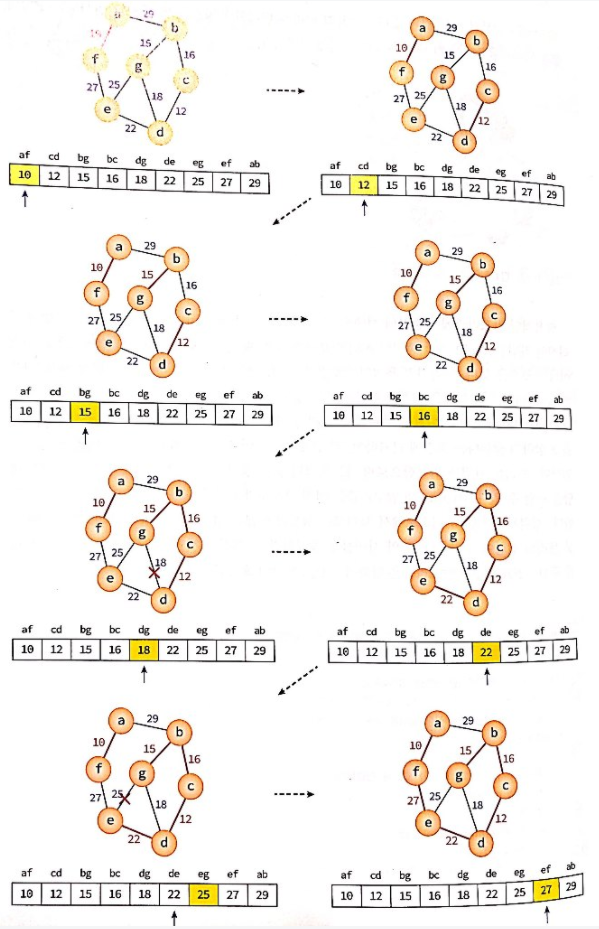
k=k+1

E[k] 선택

간선 선택 --> 사이클 형성 여부 확인

사이클을 형성하지 않으면 간선선택

i++



간선을 추가했을 때, 사이클이 형성되는지 검사해야함 = 추가하는 간선의 양끝 정점이 이미 집합에 속해있는지 검사 🡪 union-find 알고리즘 사용

union-find 연산 일반적으로 많이 사용되는 연산

union(x, y) 원소 x와 y가 속해 있는 집합을 입력으로 받아 합집합을 만듬

ex) {1}{2}{3}{4}{5}{6} – union(4,5), union(5,2) 🡪 {1,4}{5,2},{3},{6} – union(4,5), union(3,6) 🡪{1,4,5,2},{3,6}

find(x) 원소 x가 속해있는 집합 반환

union-find 연산의 구현 집합 구현 방법 비트벡터, 배열, 연결리스트, 트리

1차원 배열로 부모포인터의 위치만 저장 부모포인터가 없다면, -1

알고리즘11.3 UNION(a, b) :

root1 = FIND(a);

root2 = FIND(b);

if(root1 != root2)

parent[root1] = root2

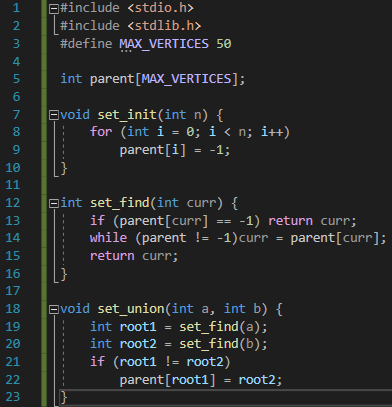
FIND(curr):

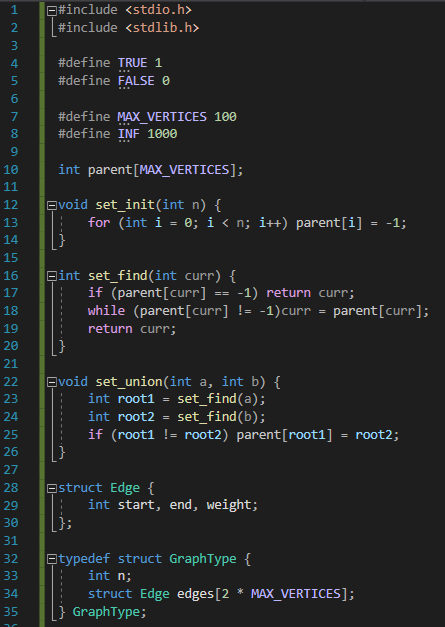
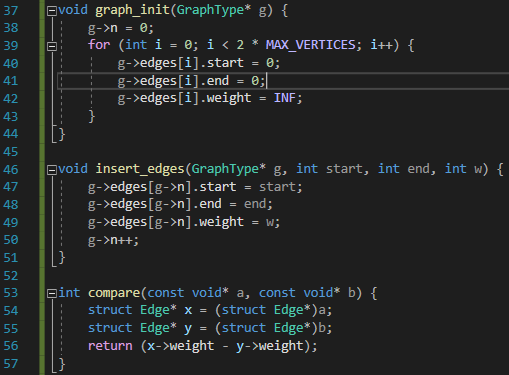
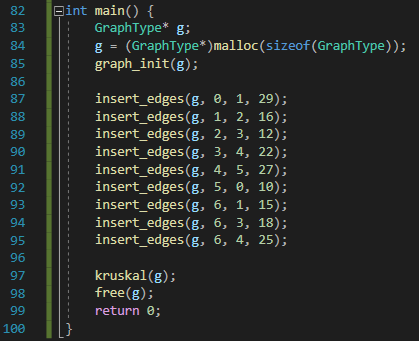
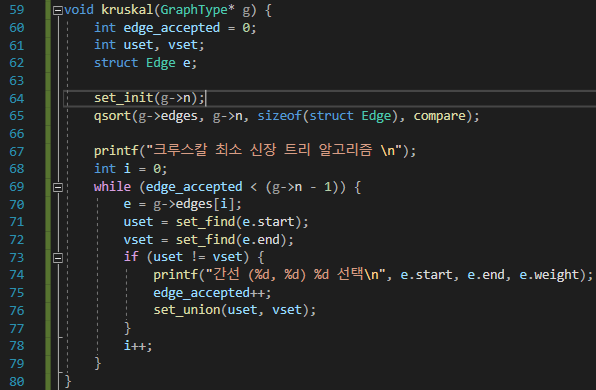
if(parent[curr] == 1)

return curr

while(parent[curr] != -1) curr = parent;

return curr

프로그램11.1 

kruskal의 알고리즘 구현 프로그램 11.2   

시간 복잡도 분석 |e|log2|e|

Quiz 1. (0,1) – (7,3) – (7,4) – (4,5) – (5,6) – (5,1) – (1,2)

11.3 Prim의 MST알고리즘

시작 정점으로부터 출발해 신장트리를 단계적으로 확장하는 방법

앞 단계에서 만들어진 신장 트리 집합에 인접한 정점들중 최저 간선으로 연결된 정점 선택

비교 Kruskal 간선 기반 알고리즘

Prim 정점 기반 알고리즘

알고리즘11.4

Prim 알고리즘의 구현 알고리즘11.5 //입력: 네트워크 G = (v, e), s는 시작 정점

//출력: 최소 비용 신장 트리를 이루는 정점들의 집합

Prim(G, s):

for(u in v)

distance[u] = INF

distance[s] = 0 //정점들이 트리집합에 추가되면서, distance값 변경

우선순위 큐에 모든 정점 삽입

for(i=0; i<n-1; i++) //루프로 가장 작은 distance값을 가지는 정점 꺼냄

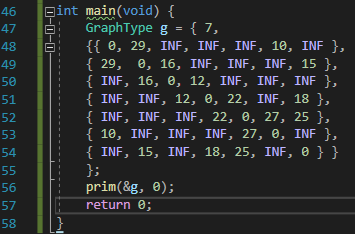
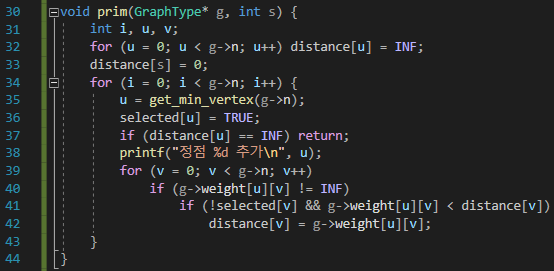
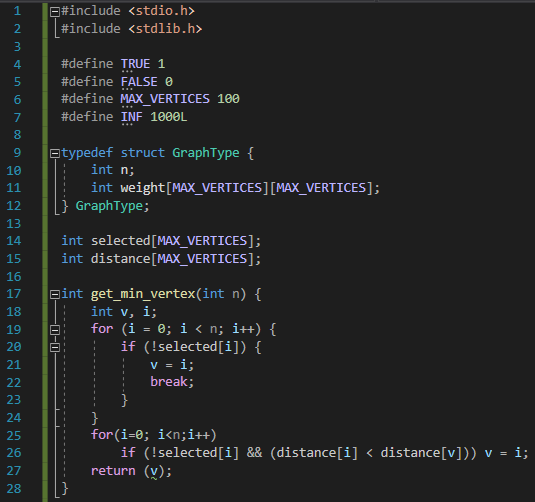
u=delete\_min(Q) //꺼냈으니 큐에서 삭제

printf(u)

for(each in (u의 인접정점))

if(v in Q and weight[u][v]<dist[v])

dist[v] = weight[u][v] //정점을 꺼냈으니 distance값 변경

프로그램11.3 

Prim 알고리즘의 분석 주 반복문이 정점의 수 n번 반복, 내부 반복문이 n번 반복 🡪 O(n2)

희소 그래프 대상 Kruskal의 알고리즘 적합

밀집 그래프 대상 Prim의 알고리즘 적합

Quiz 1. 0 🡪 7 🡪 3 🡪 4 🡪 5 🡪 1 🡪 2 🡪 6

11.4 최단 경로

최단 경로 네트워크에서 정점 i와 정점 j를 연결하는 경로 중에서 간선의 가중치 합이 최소가 되는 경로

간선의 가중치 비용, 거리, 시간…

최단 경로를 찾아내는 알고리즘 1. Dijikstra 알고리즘 하나의 시작 정점에서 다른 정점까지의 최단 경로

2. Floyd 알고리즘 모든 정점에서 다른 모든 정점까지의 최단 경로를 구함

11.5 Dijikstra의 최단 경로 알고리즘

하나의 시작 정점으로부터 모든 다른 정점까지의 최단 경로를 찾는 알고리즘

최단 경로 경로의 길이 순

요소 시작정점 v

집합 S 시작정점v로부터의 최단 경로가 이미 발견된 정점들의 집합

알고리즘 각 단계에서 가장 distance값이 가장 작은 정점을 추가 🡪 모든 정점까지의 최단거리를 구할 수 있음

배열 distance 시작정점에서 집합S에 있는 정점만을 거쳐서 다른 정점으로 가는 최단 거리를 기록하는 배열

새로운 정점이 S에 추가되면 distance값 수정 새로 추가된 정점을 거쳐서 정점까지 가는 거리와 기존의 거리를 비교해 더 작은 거리로 distance값을 수정

새로 추가된 정점 u

목표정점 w

distance[w] = min(distance[w], distance[u]+weight[u][w])

2차원배열 weight 가중치 인접 행렬

distance[w] = weight[v][w]

직접간선이 없을 경우 INF저장

알고리즘11.6 //입력: 가중치 그래프 g, 가중치는 음수가 아님

//출력: distance 배열, distance[u]는 v에서 u까지의 거리

shortest\_path(g, v) :

s = { v }

for(정점 w in g):

distance[w] = weight[v][w];

while 모든 정점이 S에 포함되지 않으면:

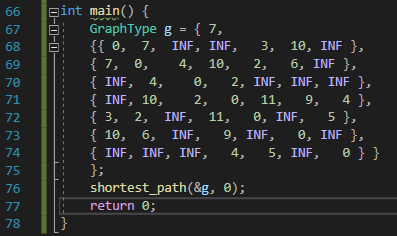
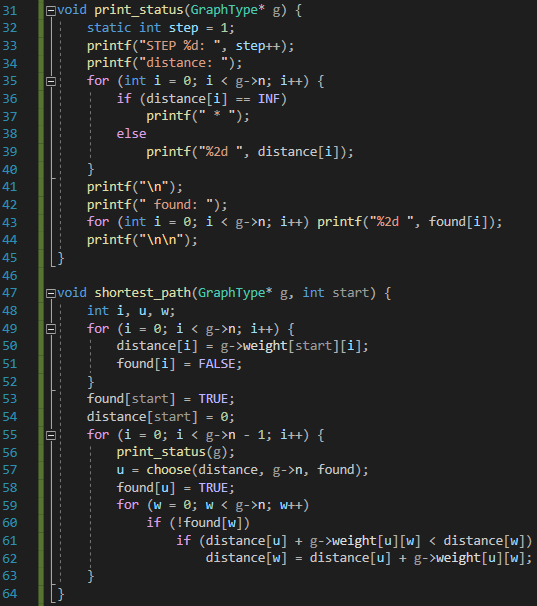
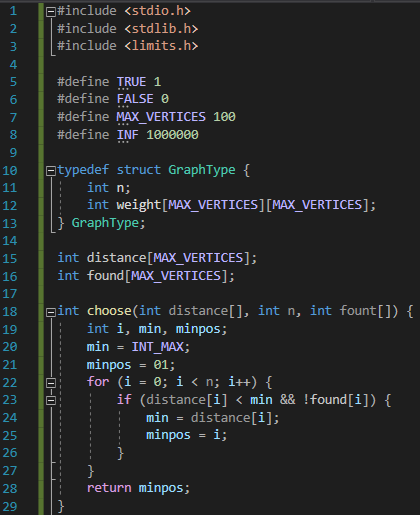
u = 집합 s에 속하지 않는 정점 중에서 최소 distance 정점

append(s, u)

for(u에 인접하고 S에 있는 각 정점)

if(distance[u]+weight[u][z] < distance[z])

distance[z] = distance[u]+weight[u][z]

Dijikstra의 알고리즘 구현 프로그램11.4 

Dijikstra의 알고리즘 분석 n개 정점 주 반복문 n번 반복, 내부반복문 2n번 반복 🡪 O(n2)

Quiz 1. 0 7 \* 5 \* 2 \* \* 🡪 0 7 \* 5 2 2 6 \* 🡪 0 6 \* 5 4 2 6 8 🡪 0 6 7 5 4 2 6 8 🡪 0 6 7 5 4 2 6 7 🡪 0 6 7 5 4 2 6 7 🡪 0 6 7 5 4 2 6 7

11.6 Floyd의 최단 경로 알고리즘

그래프에 존재하는 모든 정점 사이의 최단 경로를 한꺼번에 찾아줌

요소 2차원배열 weight 가중치 인접 행렬

직접간선이 없을 경우 INF저장

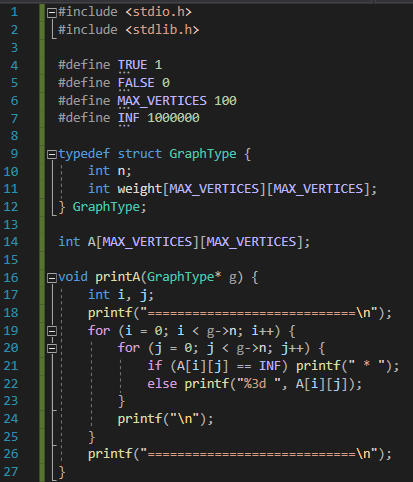
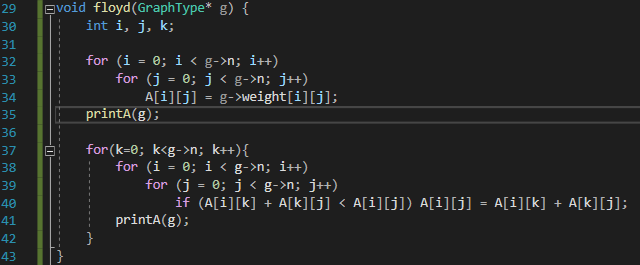
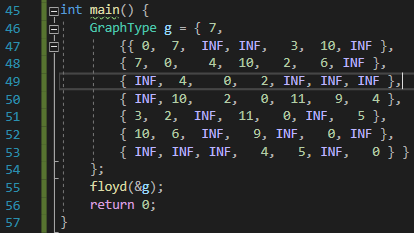
알고리즘11.8 floyd(g):

for(k=0 to n-1)

for(i=0 to n-1)

for(j=0 to n-1)

weight[i][j] = min(weight[i][j], weight[i][k]+weight[k][j])

Floyd의 알고리즘 구현 프로그램11.5   

Floyd의 알고리즘 분석 시간복잡도 O(n3)

간결한 알고리즘 🡪 Dijikstra의 알고리즘보다 빠름

Quiz 1. 0 3 1 🡪 0 3 1 🡪 0 3 1 🡪 0 1 1

2 0 4 2 0 3 2 0 3 2 0 3

5 0 1 5 0 1 2 0 1 2 0 1

11.7 위상 정렬

선행 방향 그래프에서 간선 <u, v>가 있다면 정점 u는 정점 v를 선행함

위상정렬 그래프에 존재하는 각 정점들의 선행순서를 위배하지 않으면서 모든 정점을 나열하는 것

알고리즘 진입차수가 0인 정점 선택 🡪 선택된 정점과 그 정점에 부착된 간선을 삭제 🡪 진입차수가 0인 정점 선택 🡪 선택된 정점과 그 정점에 부착된 간선 삭제 🡪 …

위상순서 선택되는 정점의 순서

알고리즘11.9 //입력: 그래프(V, E)

//출력: 위상 정렬 순서

topo\_sort(G):

for (int i = 0; i < n - 1; i++) {

if(모든 정점이 선행 정점을 가지면) 사이클이 존재하고 위상 정렬 불가

선행정점을 가지지 않는 정점 v를 선택

v를 출력

v와 v에서 나온 모든 간선을 그래프에서 삭제

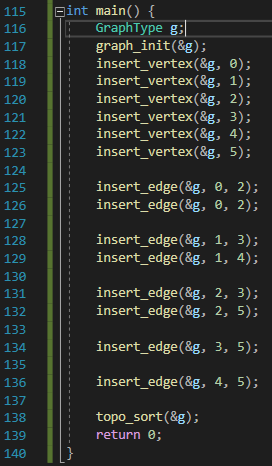
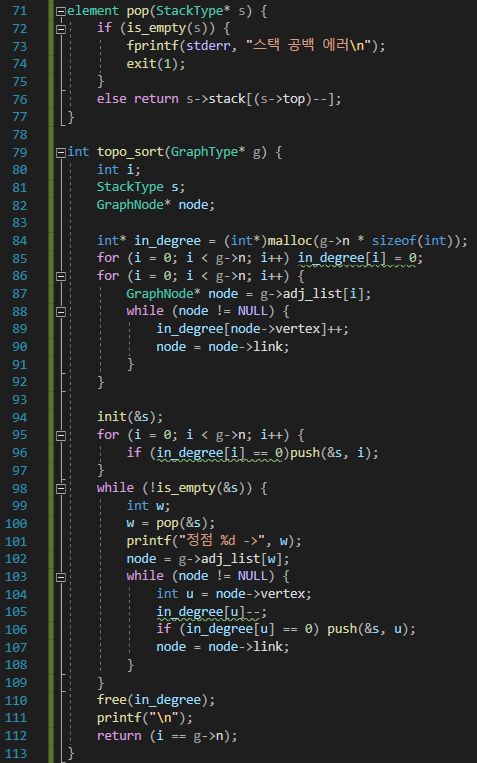
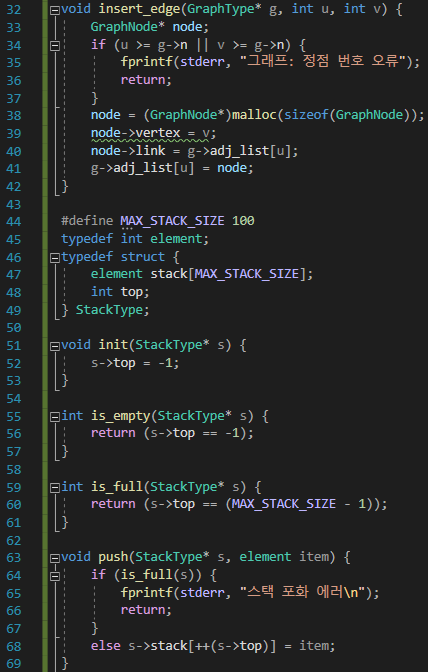
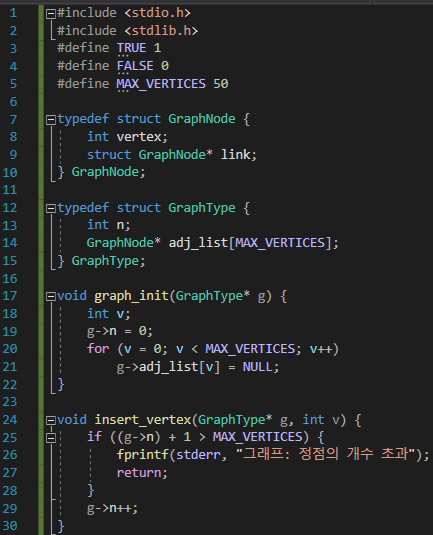
}

위상 정렬 알고리즘의 구현 구성 in\_degree 1차원 배열

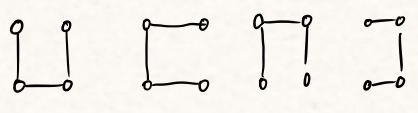
각 정점의 진입차수 기록

요소가 0인 인덱스 선택

스택 후보정점들이 저장됨

프로그램11.6 

연습문제

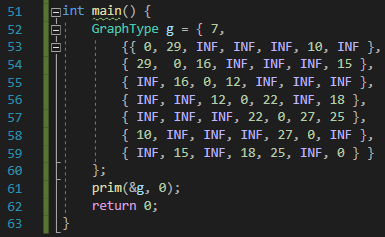
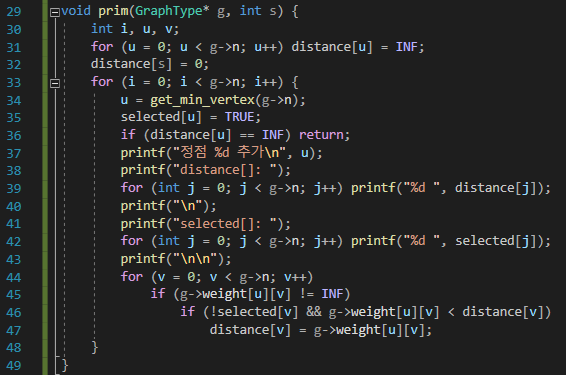
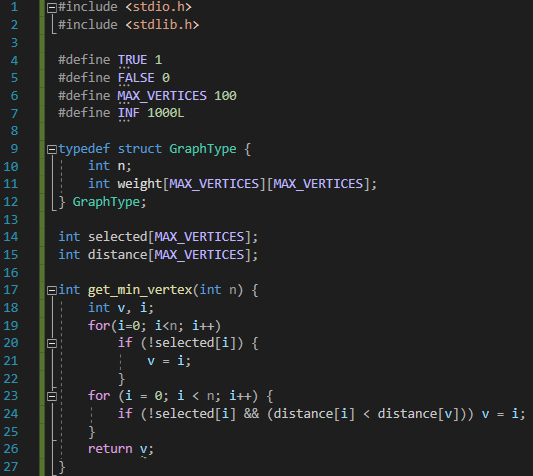
1. 

2. [ 5,5,6,7,7,8,8,9,9,11,15 ] 🡪 (A, D)[ X,5,6,7,7,8,8,9,9,11,15 ] 🡪 (A,D)(C,E)[ X,X,6,7,7,8,8,9,9,11,15 ] 🡪 (A,D)(C,E)(D,F)[ X,X,X,7,7,8,8,9,9,11,15 ] 🡪 (A,D)(C,E)(D,F)(A,B)[ X,X,X,X,7,8,8,9,9,11,15 ] 🡪 (A,D)(C,E)(D,F)(A,B)(B,E)[ X,X,X,X,X,8,8,9,9,11,15 ]

🡪(A,D)(C,E)(D,F)(A,B)(B,E)(E,G)[ X,X,X,X,X,8,8,X,9,11,15 ]

3. A 🡪 D 🡪 F 🡪 B 🡪 E 🡪 C 🡪 G

4.



distance[] 정점간의 거리가 저장되어 있는 배열

selected[] 정점의 선택 여부를 나타내는 배열

5.

단계 선택된 정점 found배열 distance배열

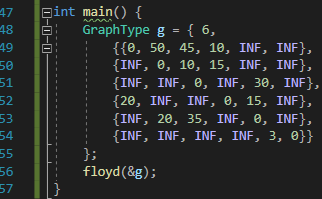
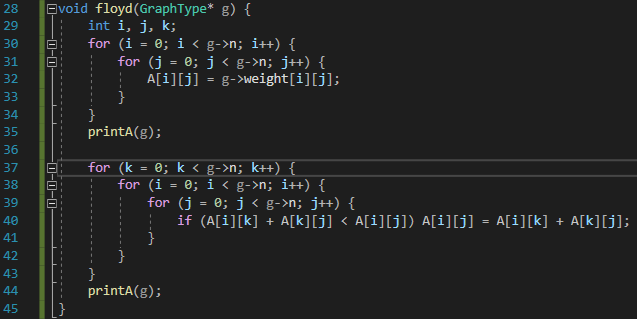
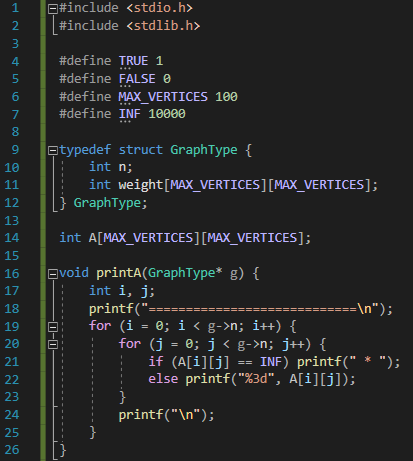
1 3 [0] [0, 50, 45, 10, INF, INF]

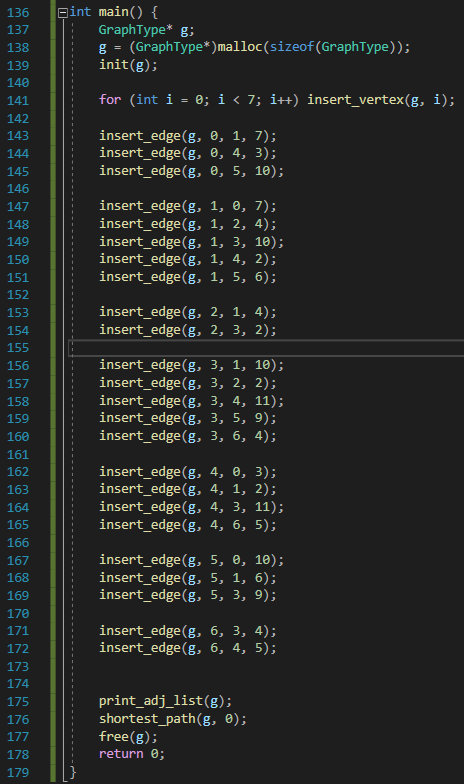
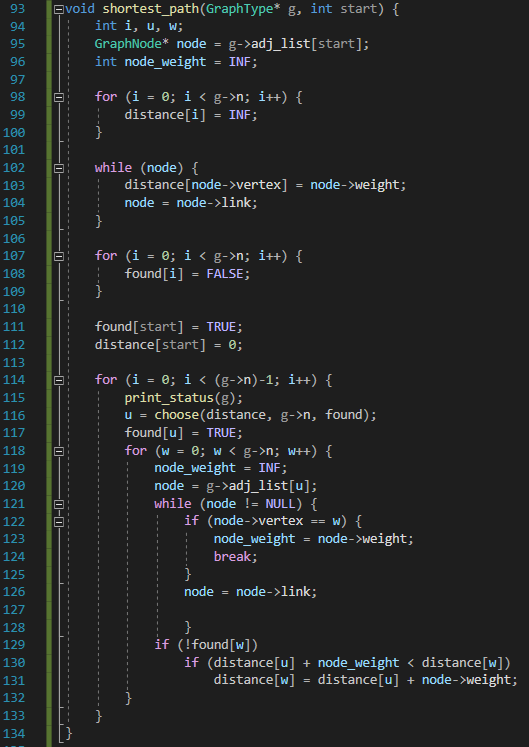
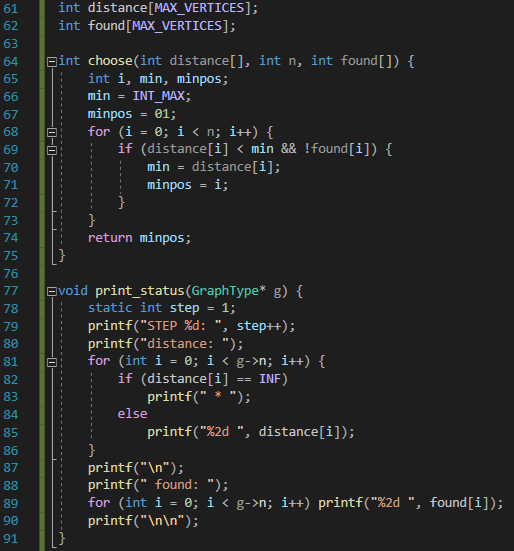
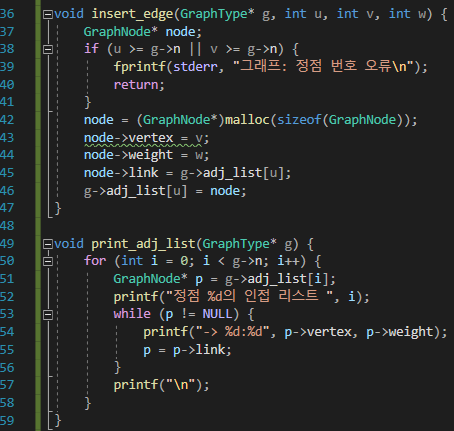
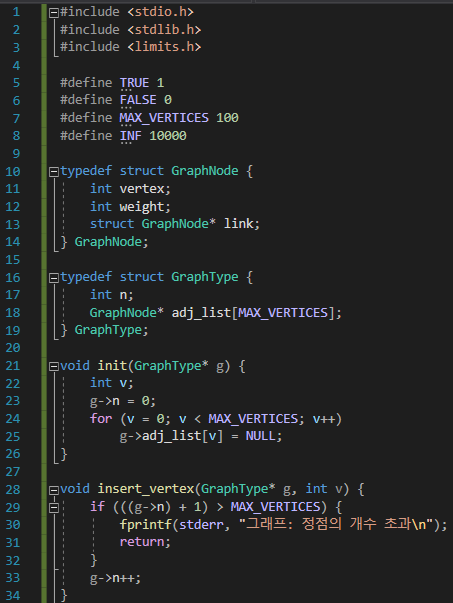
2 4 [0, 3] [0, 50, 45, 10, 25, INF]

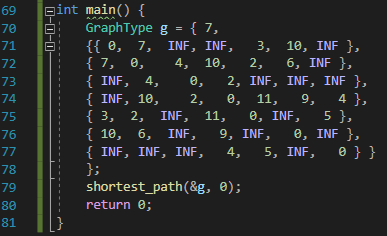
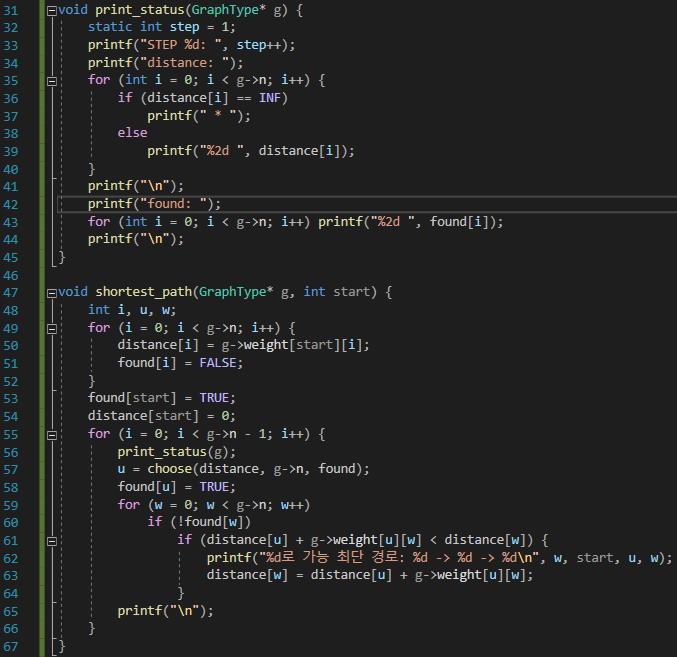
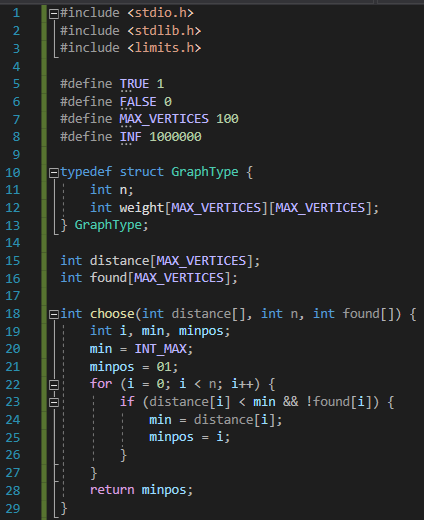
3 1 [0, 3, 4] [0, 45, 45, 10, 25, INF]

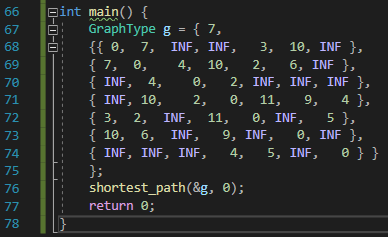
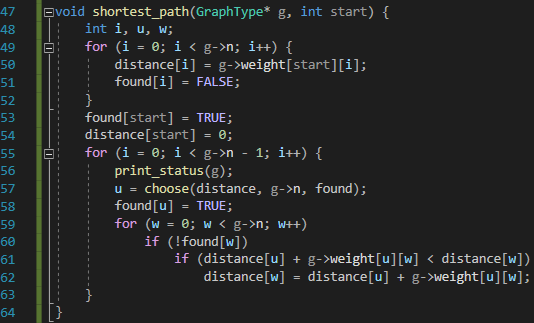
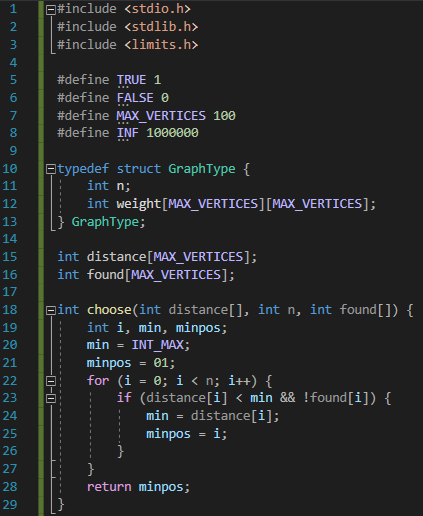
4 2 [0,3,4,1] [0,45,45,10,25,INF]

5 X [0,3,4,1,2] [0,45,45,10,25,INF]

6. 

7. 

8. 

9. 

10. 1 🡪 2 🡪 3 🡪 4 🡪 5 🡪 6 🡪 7 🡪 8