Chapter 14.
그래프 (Graph)

14주차

최소비용 신장트리

사이클의 이해

- 트리 : 그래프의 한 유형
  - 사이클을 형성하지 않는 그래프
- 경로 : 두 개의 정점을 잇는 간선이 있을 때 정점을 순서대로 나열할 것
- 단순 경로 : 동일한 간선을 중복하여 포함하지 않는 경로
- 사이클 : 시작과 끝이 같은 단순경로
  - 사이클도 단순경로이다
- 신장 트리(spanning tree)
  - 그래프의 모든 정점이 간선에 의해 하나로 연결됨.
  - 그래프 내에 사이클이 존재하지 않음.

최소 비용 신장 트리 (minimum cost spanning tree)

- 가중치 : 비용, 거리, 시간 등을 의미하는 값
- 가중치 그래프 : 연결선에 가중치를 부여한 그래프

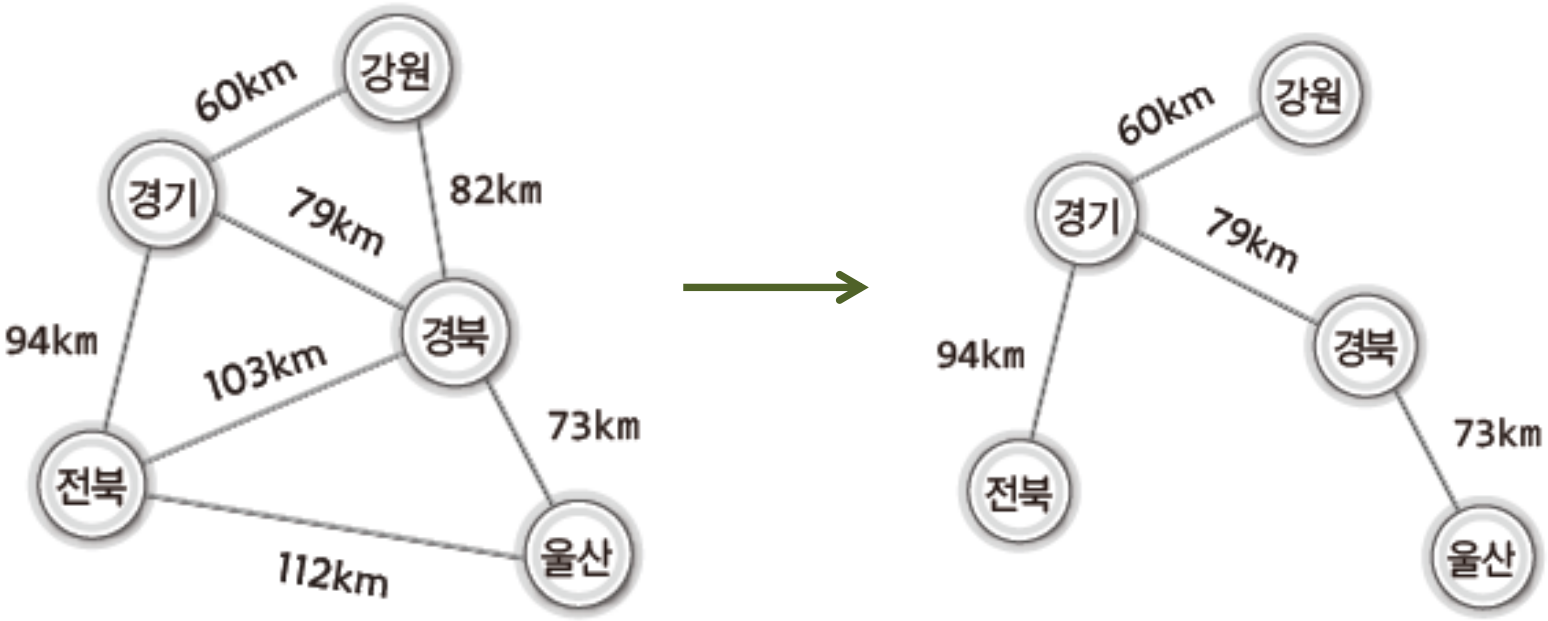
- 최소비용신장트리 : 무방향 가중치 그래프에서 신장 트리를 구성하는 연결선들의 가중치 합이 최소인 신장 트리

- 용도 : 각종 망을 최소 비용으로써 구성하는 데 사용.

최소비용 신장트리

◆ 최소비용 신장트리
- 신장 트리의 모든 간선의 가중치의 합이 최소인 그래프

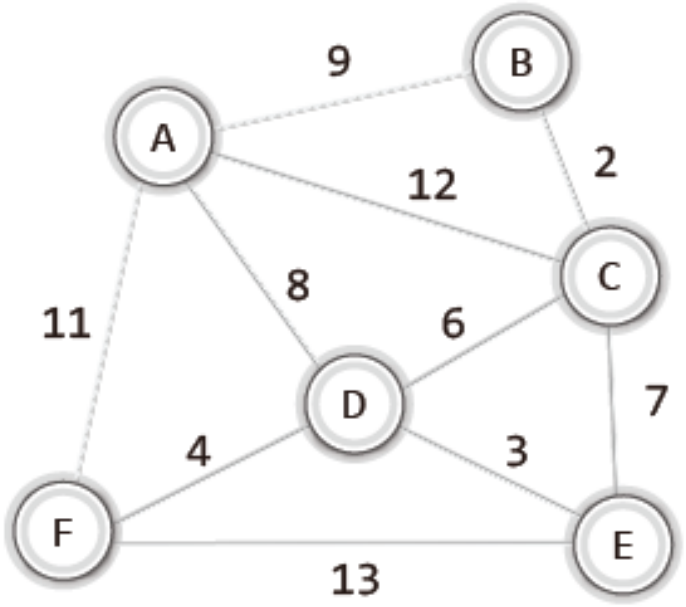
◆ 최소비용 신장트리 구성 예



581-587p

최소비용 신장트리 만들기
(크루스칼 알고리즘)

- 가중치를 기준으로 간선을 정렬한 후에 신장 트리가 될 때까지 간선을 하나씩 선택 또는 삭제해 나가는 방식
- 크루스칼 알고리즘 1 :간선을 하나씩 선택
  - 간선을 오름차순으로 정렬 -> 가중치가 가장 작은 간선을 포함시키기 (단, 사이클이 형성된다면 skip)
- 크루스칼 알고리즘 2 :간선을 하나씩 삭제
  - 간선을 내림차순으로 정렬 -> 가중치가 가장 큰 간선을 제외시키기 (단, 신장 트리가 아니게 될 때는 skip)



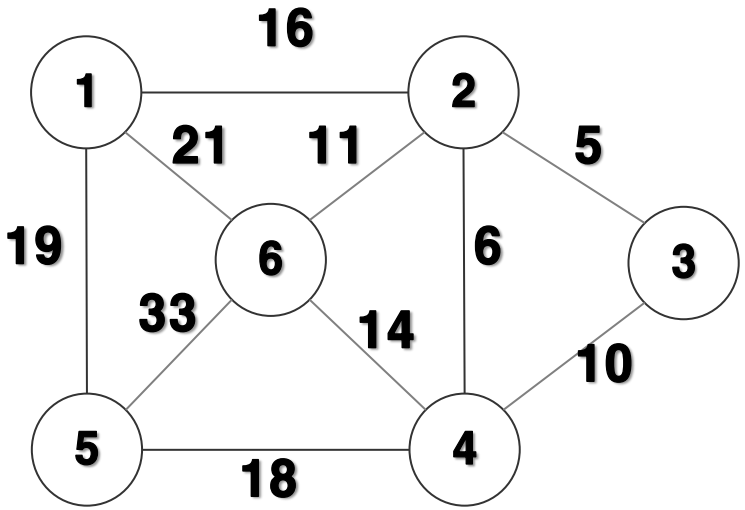
▶ [그림 14-49: 크루스칼 알고리즘 1의 1/4]

2, 3, 4, 6, 7, 8, 9, 11, 12, 13

가중치의 오름차순 정렬

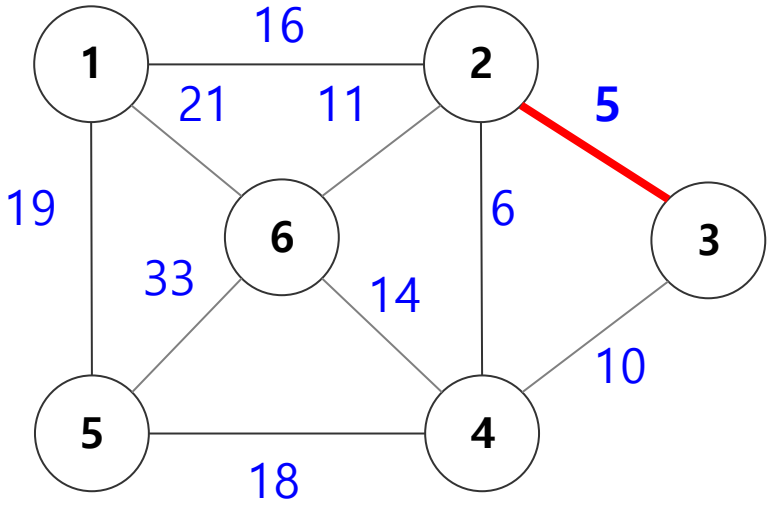
Kruscal 알고리즘으로 최소 비용 신장 트리 만들기

- 연결선중 가장 가중치가 적은 연결선의 순으로 선택
- 사이클이 발생하면 제외.
- 위 과정 반복 후, 모든 정점이 연결되면 알고리즘 끝낸다.

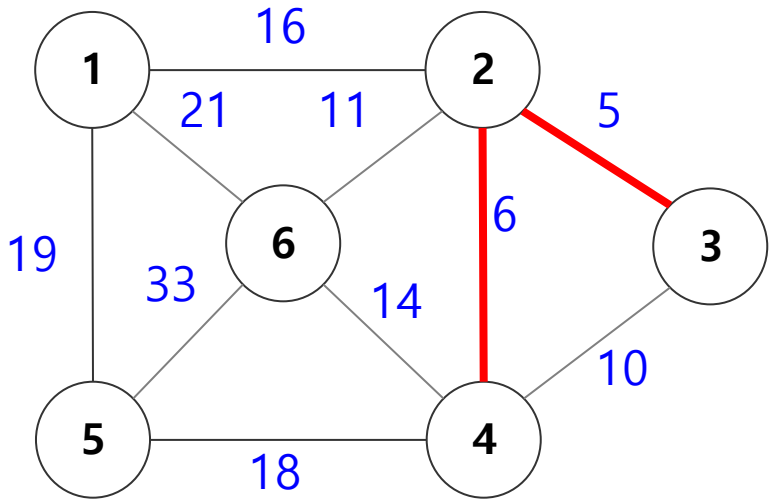


[가중치가 있는 그래프]

Kruscal 알고리즘으로 최소 비용 신장 트리 만들기

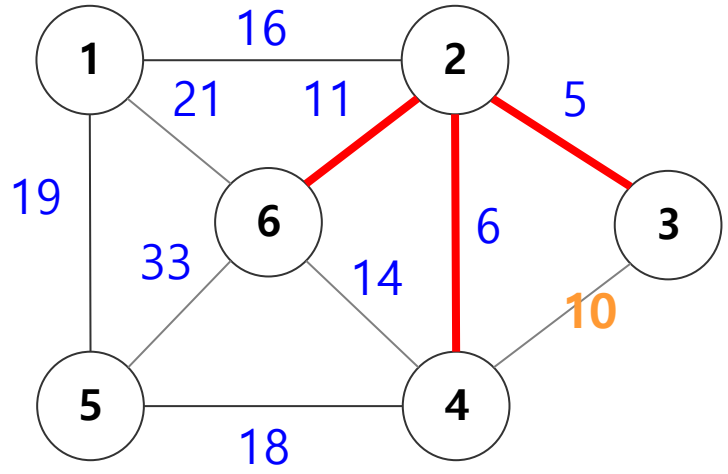


① 가장 적은 가중치 5를 선택

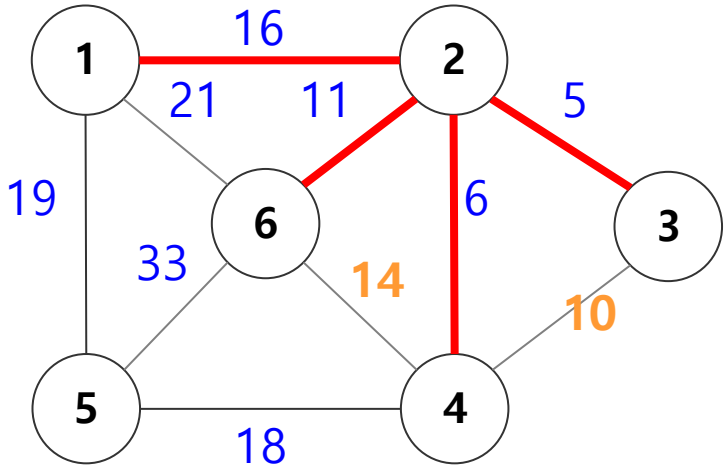


② 두번째 적은 가중치 6을 선택

Kruscal 알고리즘으로 최소 비용 신장 트리 만들기

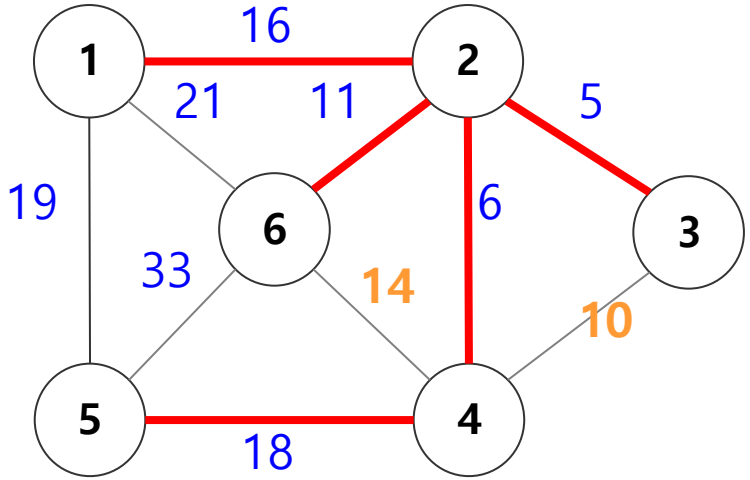


③ 세번째 적은 가중치 10을 선택하면 Cycle이 발생하여, 제외 네번째 적은 11을 선택

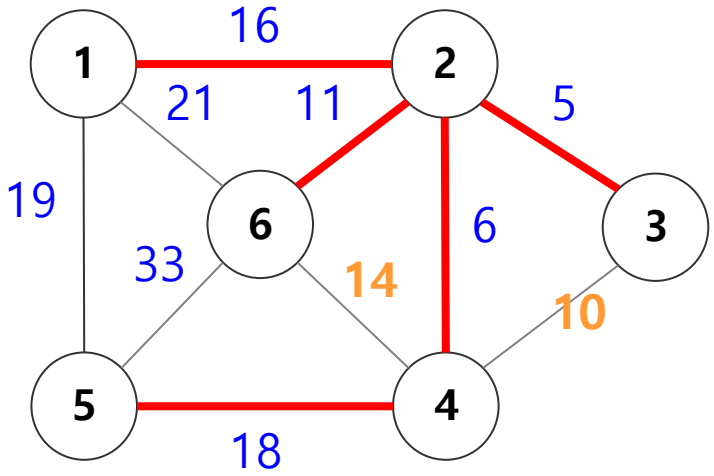


④ 다섯번째 적은 14를 선택하면 Cycle이 발생하여 제외, 여섯번째 적은 16을 선택

Kruscal 알고리즘으로 최소 비용 신장 트리 만들기



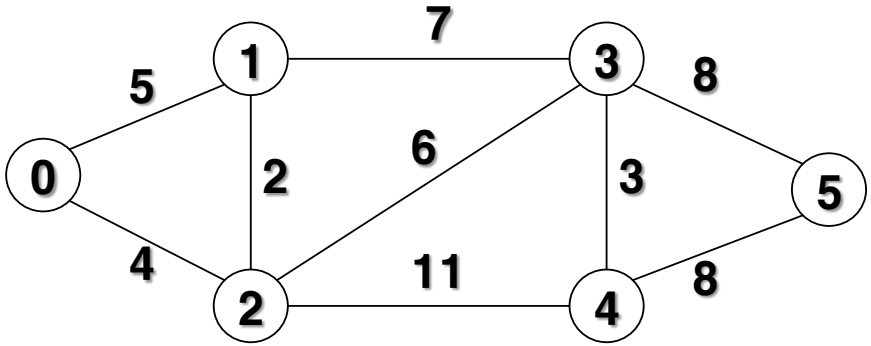
⑤ 일곱번째 적은 18을 선택



⑥ 모든 정점이 연결된 신장 트리가 완성

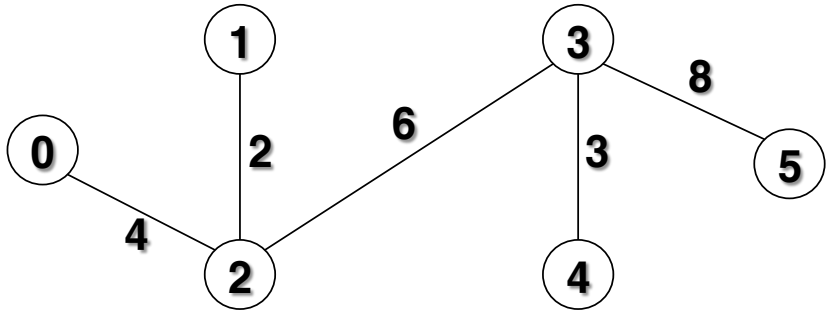
총비용 = 5+6+11+16+18 = 56

Kruscal 알고리즘으로 최소 비용 신장 트리 만들기



Kruscal 알고리즘에 의한 최소비용 신장트리 작성

선택되는 연결선의 순서를 쓰고, 최소 비용을 계산하시오.

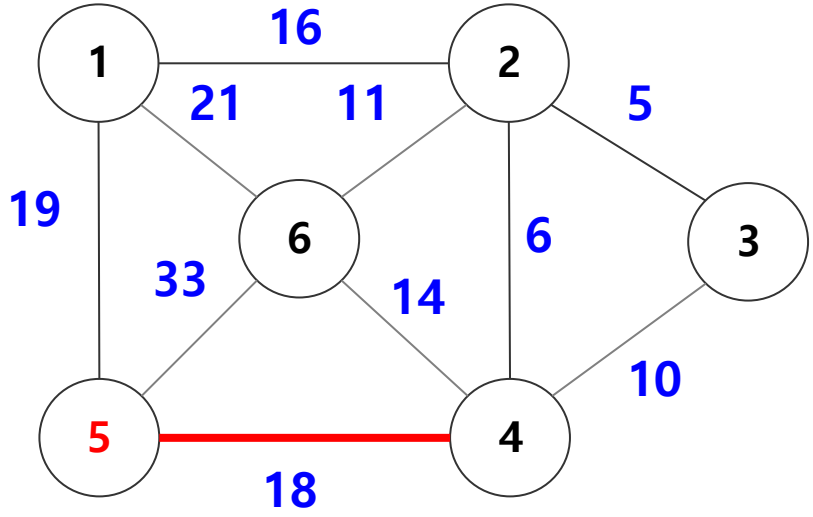


연결선 2 -> 3 -> 4 -> 6 -> 8 : 최소비용=23

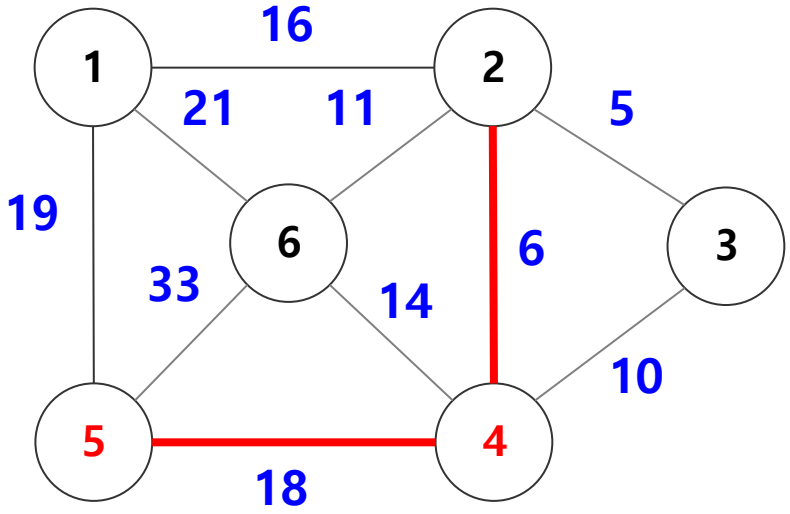
Prim 알고리즘으로 최소 비용 신장 트리 만들기

- (1) 그래프 G에서 임의로 시작 정점을 선택하여 '결과 노드집합'에 포함시키기.
- (2) '결과 노드집합'의 정점들에 부속된 연결선 중에서 가중치가 가장 작은 연결선(사이클을 발생시키는 연결선은 제외)을 연결하여 '결과 노드집합'에 노드 추가.
- (3) (2)를 반복.
- (4) 그래프 G의 연결선이 n-1개가 되면 최소 비용 신장 트리 완성.

Prim 알고리즘으로 최소 비용 신장 트리 만들기

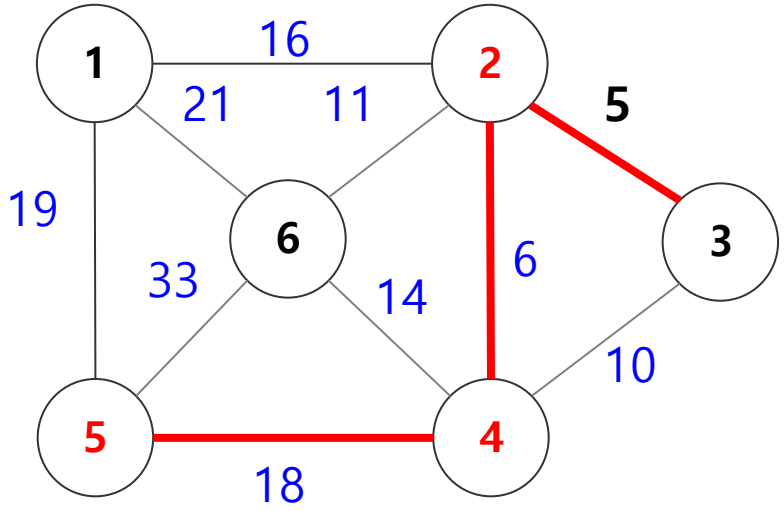


① 임의의 정점 V를 선택,
여기서는 5를 선택
정점 5에 부속된 연결선중
가중치가 적은 18을 선택

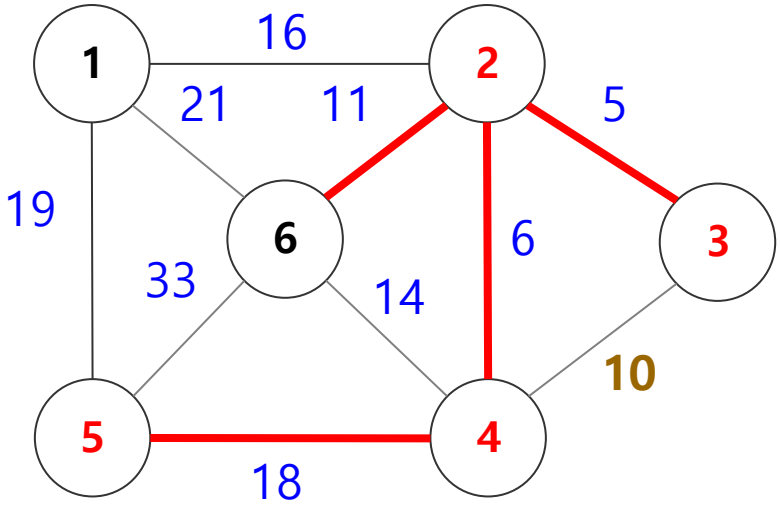


② 정점 5, 4에 대해
가장 적은 연결선 6을 선택

Prim 알고리즘으로 최소 비용 신장 트리 만들기

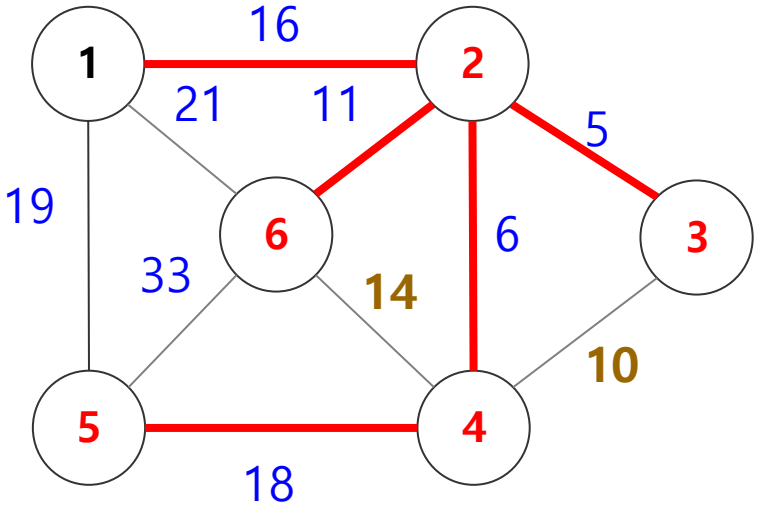


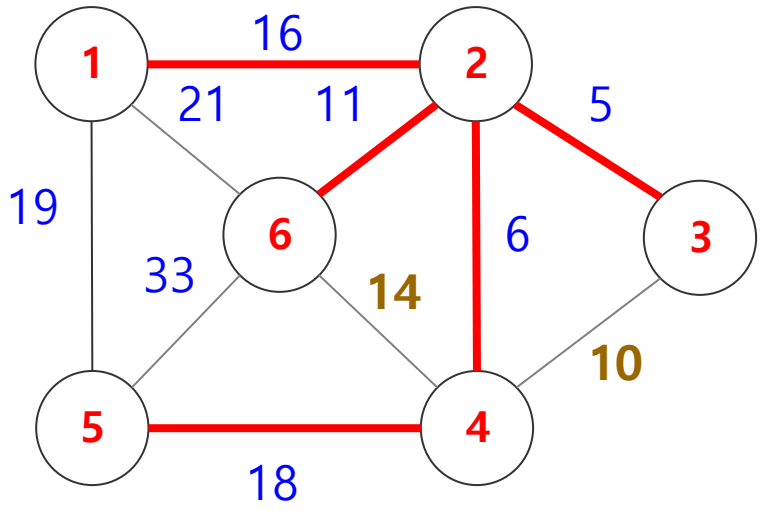
③ 정점 5, 4, 2에 대해
가장 적은 연결선 5을 선택



④ 정점 5, 4, 2, 3에 대해
가장 적은 연결선 10을 선택하면
cycle이 발생되므로 제외하고
두번째로 적은 연결선 11을 선택

Prim 알고리즘으로
최소 비용 신장 트리 만들기



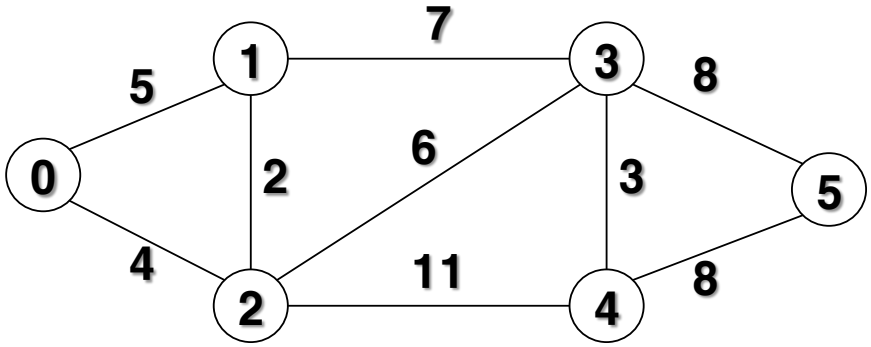


⑤ 정점 5,4, 2, 3, 6에 부속된 연결선 중 가장 적은 값인 14를 선택 하면 cycle이 발생하므로 제외하고 두번째로 적은 연결선 16을 선택.

⑥ 최종 선택된 최소비용 SpanningTree

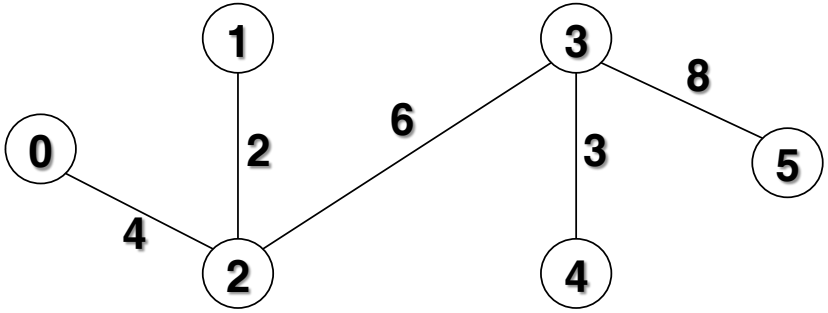
총비용 = 56

Prim 알고리즘으로 최소 비용 신장 트리 만들기



Prime 알고리즘에 의한 최소비용 신장트리 작성 (3부터 시작)

정점 확장되는 순서를 쓰고, 최소 비용을 계산하시오.



정점 3 -> 4 -> 2 -> 1 -> 0 -> 5 : 최소비용=23

STL iterator
(반복자)

- 반복자로 인해 알고리즘은 특정 container에 종속되지 않고 순회/접근이 가능
- 저장된 원소를 순회하고 접근하는 일반화된 방법을 제공
  - 순회 : ++ != == 연산자
  - 접근 : * 연산자
- 순차열의 시작과 끝
  - begin() ~ end() : begin 포함, end 불포함

```
#include <iostream>
#include <list>
using namespace std;
int main() {
        vector<int> v;
        v.push_back(10);  v.push_back(20);  v.push_back(30);

        vector<int>::iterator iter = v.begin();
        while (iter != v.end()) {
                cout << *iter << endl;
                iter++;
        }
        return 0;
}
```

vector 를 list 로 바꿔도 iterator는 그대로 사용.

Iterator 사용 방법

C++ STL container
(priority_queue)

pop() 할 때 max 또는 min 값을 꺼내 줌.

- 기본 형태
  - priority_queue<T, Container, Compare>
- 우선순위 큐 인터페이스
  - push/pop/top 연산 제공(높은 우선순위 요소가 위에 존재)
  - 추가 삭제
- 힙을 사용하여 구현됨 (요소들은 비교를 지원해야 함)

C++ STL container (priority_queue)

pop() 할 때 max 또는 min 값을 꺼내 줌.

```
#include <iostream>
#include <queue>
using namespace std;
int main() {
        int values[] = { 5,2,8,9,1,14 };
        priority_queue< int, vector<int>, less<int> > pq;
        cout << "push: ";
        for (int i = 0; i < sizeof(values) / sizeof(values[0]); i++) {
                cout << values[i] << " ";
                pq.push(values[i]);
        }
        cout << endl;
        cout << "size: " << pq.size() << endl;
        cout << "pop: " << endl;
        while (!pq.empty()) {
                cout << "top: " << pq.top() << endl;
                pq.pop();
        }
        cout << endl;
        return 0;

}
```

비교연산 class 사용.
less : 작은것부터
greater : 큰 것부터

Push elements

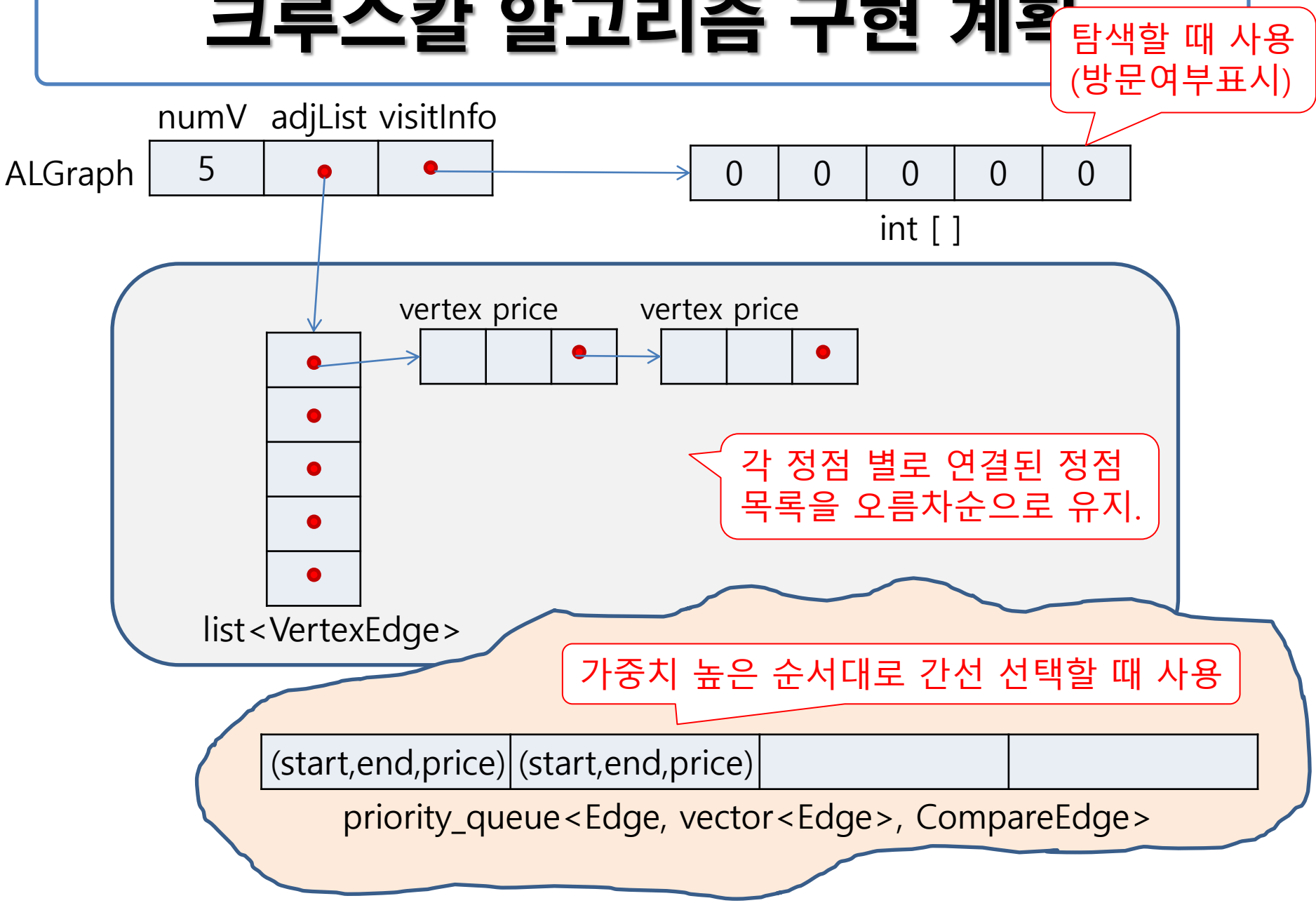
Pop elements

[실행결과]

```
push: 5 2 8 9 1 14
size: 6
pop:
top: 14
top: 9
top: 8
top: 5
top: 2
top: 1
```

589-598p

크루스칼 알고리즘 구현 계획



크루스칼 알고리즘 구현
(전처리기 선언)

```
#include <stdio.h>
#include <cstdlib>
#include <memory>
#include <stack>
#include <list>
#include <queue>
#include <iostream>
using namespace std;

#define TRUE  1
#define FALSE 0

// 정점의 이름들을 상수화
enum { A, B, C, D, E, F, G, H, I, J };
```

크루스칼 알고리즘 구현
(타입 선언)

```
class Edge {
public:
	int start, end, price;
	Edge(int _start, int _end, int _price) {
		start = _start; end = _end; price = _price;
	}
};
class VertexEdge {
public:
	int vertex, price;
	VertexEdge(int _vertex, int _price) {
		vertex = _vertex; price = _price;
	}
};
class ALGraph {
public:
	int numV;   // 정점의 수
	list<VertexEdge> *adjList;   // 간선의 정보
	int* visitInfo;
};
```

크루스칼 알고리즘 구현
(그래프 초기화 / 해제)

```
void GraphInit(ALGraph* pg, int nv)
{
	int i;

	pg->adjList = new list<VertexEdge>[nv];
	pg->numV = nv;

	// 탐색 정보 등록을 위한 공간의 할당 및 초기화
	pg->visitInfo = new int[pg->numV];
	memset(pg->visitInfo, 0, sizeof(int) * pg->numV);
}
void GraphDestroy(ALGraph* pg)
{
	if (pg->adjList != NULL)
		delete [] pg->adjList;

	if (pg->visitInfo != NULL)
		delete [] pg->visitInfo;
}
```

589-598p

크루스칼 알고리즘 구현
(리스트에 간선 추가. 오름차순으로)

```
void AddListBySort(list<VertexEdge>& _list, int toV, int price)
{
	list<VertexEdge>::iterator iter = _list.begin();
	VertexEdge ve(toV, price);
	while (iter != _list.end()) {
		if (iter->vertex > toV) {
			_list.insert(iter, ve);
			return;
		}
		iter++;
	}
	_list.push_back(ve);
}
```

현 위치 바로 앞에 넣으면 될까?

제일 크니까 맨 뒤로.

AddEdge()

호출

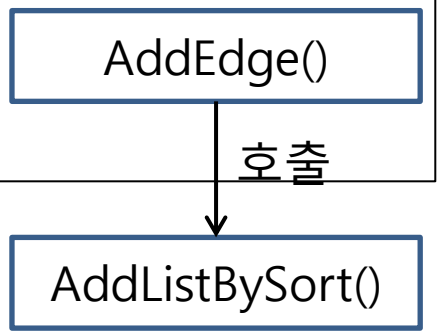
AddListBySort()

589-598p

크루스칼 알고리즘 구현
(리스트에 간선 추가. 오름차순으로)

```
struct CompareEdge {
        bool operator()(Edge& e1, Edge& e2) {
                return (e1.price < e2.price);
        }
};
void AddEdge(ALGraph* pg,
    priority_queue<Edge, vector<Edge>, CompareEdge> *pq,
    int fromV, int toV, int price)
{
        AddListBySort(pg->adjList[fromV], toV, price);
        AddListBySort(pg->adjList[toV], fromV, price);
        if (pq != NULL)
                pq->push(Edge(fromV, toV, price));
}
```

처음 그래프 생성할 때 :
  우선순위 큐에 넣어야 함.
"최소비용신장트리"로 만드는 과정일 때 :
  우선순위 큐에 다시 넣지 않아야 함.



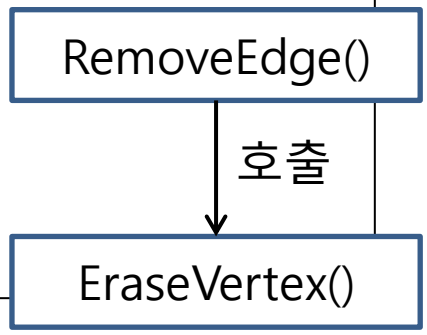
589-598p

크루스칼 알고리즘 구현
(간선 삭제)

```
void EraseVertex(list<VertexEdge>& _list, int vertex) {
        list<VertexEdge>::iterator iter = _list.begin();
        while (iter != _list.end()) {
                if (iter->vertex == vertex) {
                        _list.erase(iter);
                        break;
                }
                iter++;
        }
}

void RemoveEdge(ALGraph* pg, int fromV, int toV)
{
        EraseVertex(pg->adjList[fromV], toV);
        EraseVertex(pg->adjList[toV], fromV);
}
```

현 위치의 항목을 지우면 될까?



크루스칼 알고리즘 구현
(그래프를 출력)

```
void ShowGraphEdgeInfo(ALGraph* pg) {
        int i;
        int vx;

        for (i = 0; i < pg->numV; i++)      {
                printf("%c와 연결된 정점: ", i + 65);

                list<VertexEdge>::iterator iter =
                    pg->adjList[i].begin();
                while (iter != pg->adjList[i].end()) {
                        VertexEdge ve = *iter;
                        printf("%c(%d) ",
                            ve.vertex + 65, ve.price);
                        iter++;
                }
                printf("₩n");
        }
}
```

589-598p

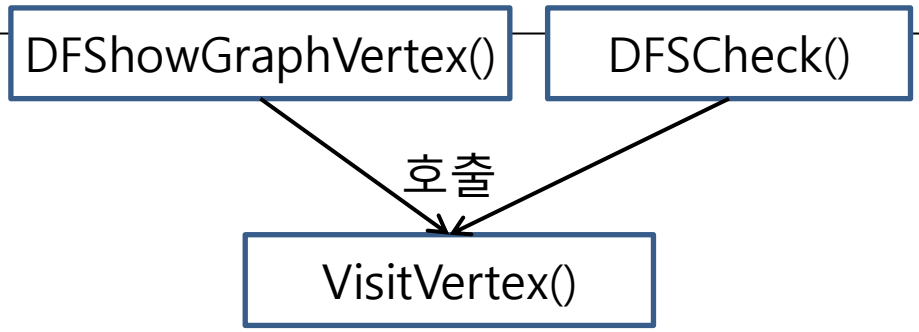
크루스칼 알고리즘 구현
(정점 방문 처리)

```
int VisitVertex(ALGraph* pg, int visitV, bool bPrint=true)
{
	if (pg->visitInfo[visitV] == 0)
	{
		pg->visitInfo[visitV] = 1;
		if (bPrint)
			printf("%c ", visitV + 65); // 방문 정점 출력
		return TRUE;
	}

	return FALSE;
}
```

방문했음을 표시

방문 성공/실패를 알려야 함.



크루스칼 알고리즘 구현
(Depth First Search: 정점의 정보 출력)

```
void DFShowGraphVertex(ALGraph* pg, int startV)
{
	stack<int> _stack;
	int visitV = startV;
	int nextV;

	// 탐색 정보 초기화
	memset(pg->visitInfo, 0, sizeof(int) * pg->numV);

	VisitVertex(pg, visitV);    // 시작 정점 방문

    .........
}
```

589-598p

```
void DFShowGraphVertex(ALGraph* pg, int startV) {
    .........
    while (TRUE) {
        list<VertexEdge>::iterator iter = pg->adjList[visitV].begin();
        int visitFlag = FALSE;
        while (iter != pg->adjList[visitV].end()) {
            if (VisitVertex(pg, iter->vertex, true) == TRUE) {
                _stack.push(visitV);
                visitV = iter->vertex;
                visitFlag = TRUE;
                break;
            }
            iter++;
        }
        if (visitFlag == FALSE) {
            if (_stack.empty())    // 스택이 비면 DFS종료
                break;
            else {
                visitV = _stack.top();
                _stack.pop();
            }
        }
    }
}
```

크루스칼 알고리즘 구현
(Depth First Search: 정점의 정보 출력)

크루스칼 알고리즘 구현
(두 정점이 연결 상태인지 체크)

```
bool DFSCheck(ALGraph* pg, int startV, int endV) {
	stack<int> _stack;
	int visitV = startV;

	// 탐색 정보 초기화
	memset(pg->visitInfo, 0, sizeof(int) * pg->numV);

	.........
	return false;
}
```

589-598p

크루스칼 알고리즘 구현
(두 정점이 연결 상태인지 체크)

```
while (TRUE) {
    list<VertexEdge>::iterator iter = pg->adjList[visitV].begin();
    int visitFlag = FALSE;
    while (iter != pg->adjList[visitV].end()) {
        VertexEdge ve = *iter;
        if (VisitVertex(pg, ve.vertex, false) == TRUE) {
            if (ve.vertex == endV)
                return true;
            _stack.push(visitV);
            visitV = ve.vertex;
            visitFlag = TRUE;
            break;
        }
        iter++;
    }
    if (visitFlag == FALSE) {
        if (_stack.empty())    // 스택이 비면 DFS종료
            break;
        else {
            visitV = _stack.top();
            _stack.pop();
        }
    }
}
```

visitV와 인접한 정점 중 해로 방문되는 정점 찾기.

방문 성공시,
현재 위치를 스택에 보관하고 새로 방문한 정점을 현재 위치로 삼기.

새로 방문할 노드를 못 찾았을 때:
스택에 보관했던 정점으로 돌아감.

크루스칼 알고리즘 구현
(main 함수 : 그래프 생성 후 dfs)

```
int main(void) {
	ALGraph graph;
	priority_queue<Edge, vector<Edge>, CompareEdge> pq;
	GraphInit(&graph, 7);// A, B, C, D, E, F, G의 정점 생성

	AddEdge(&graph, &pq, A, B, 10);
	AddEdge(&graph, &pq, A, D, 20);
	AddEdge(&graph, &pq, B, C, 30);
	AddEdge(&graph, &pq, D, C, 40);
	AddEdge(&graph, &pq, D, E, 50);
	AddEdge(&graph, &pq, E, F, 60);
	AddEdge(&graph, &pq, E, G, 70);

	ShowGraphEdgeInfo(&graph);

	DFShowGraphVertex(&graph, A); printf("₩n");
	DFShowGraphVertex(&graph, C); printf("₩n");
	DFShowGraphVertex(&graph, E); printf("₩n");
	DFShowGraphVertex(&graph, G); printf("₩n");
	.........
```

크루스칼 알고리즘 구현
(main 함수 : 그래프 생성 후 dfs)

```
        cout << "최소비용신장트리" << endl;
        while (!pq.empty()) {
                const Edge edge = pq.top();
                pq.pop();

                RemoveEdge(&graph, edge.start, edge.end);


                if (!DFSCheck(&graph, edge.start, edge.end)) {
                        AddEdge(&graph, NULL,
                                edge.start, edge.end, edge.price);
                }
        }
        ShowGraphEdgeInfo(&graph);

        GraphDestroy(&graph);
        return 0;
}
```

가장 가중치 큰 간선 꺼내기

간선 제외해 보기.

DFS 하여 두 정점이 아직 연결 상태인지 확인

두 정점이 연결 상태가 아니게 되었으면 간선을 다시 넣기