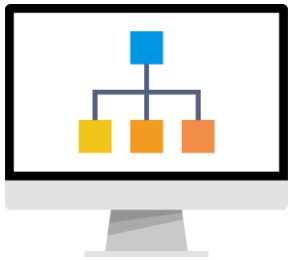


Data Structure & Algorithm
자료구조 및 알고리즘

26. 그래프 (Graph, Part 4)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef _MSC_VER
#pragma warning(disable: 4996)
#endif

typedef struct _node_t {
    int data;
    struct _node_t* left;
    struct _node_t* right;
}node_t;

node_t* create_node(int data) {
    node_t* node = (node_t*)malloc(sizeof(node_t));
    node->data = data;
    node->left = node->right = NULL;

    return node;
}

node_t* root;

void initialize() {
    root = create_node(0);
}
```

```c
int add(char pos_string[], int data) {
    node_t* p = root;
    int i;

    for (i = 0; i < strlen(pos_string) - 1; i++) {
        if (pos_string[i] == 'L') {
            if (p->left == NULL) return -1;
            p = p->left;
        }
        else {
            if (p->right == NULL) return -1;
            p = p->right;
        }
    }

    if (pos_string[i] == 'L') {
        if (p->left != NULL) return -1;
        node_t* node = create_node(data);
        p->left = node;
    }
    else {
        if (p->right != NULL) return -1;
        node_t* node = create_node(data);
        p->right = node;
    }

    return 1;
}
```

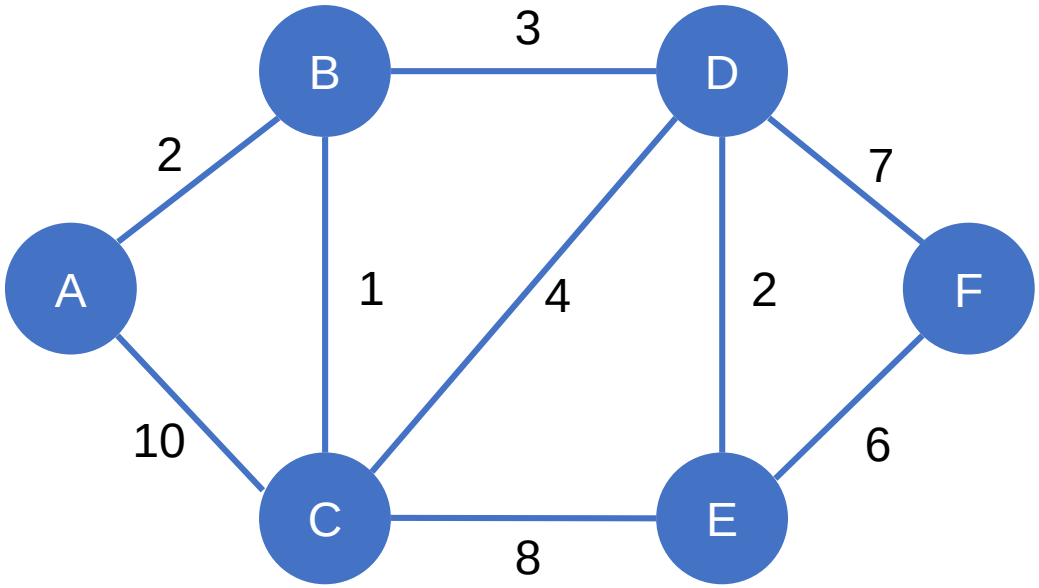
```c
int add(char pos_string[], int data) {
    node_t* p = root;
    int i;

    for (i = 0; i < strlen(pos_string) - 1; i++) {
        if (pos_string[i] == 'L') {
            if (p->left == NULL) return -1;
            p = p->left;
        }
        else {
            if (p->right == NULL) return -1;
            p = p->right;
        }
    }

    if (pos_string[i] == 'L') {
        if (p->left != NULL) return -1;
        node_t* node = create_node(data);
        p->left = node;
    }
    else {
        if (p->right != NULL) return -1;
        node_t* node = create_node(data);
        p->right = node;
    }

    return 1;
}
```

```c
int delete(char pos_string[]) {
    node_t* p = root;
    int i;

    for (i = 0; i < strlen(pos_string) - 1; i++) {
        if (pos_string[i] == 'L') {
            if (p->left == NULL) return -1;
            p = p->left;
        }
        else {
            if (p->right == NULL) return -1;
            p = p->right;
        }
    }

    if (pos_string[i] == 'L') {
        if (p->left == NULL) return -1;
        if (p->left->left || p->left->right) return -1;
        free(p->left);
        p->left = NULL;
    }
    else {
        if (p->right == NULL) return -1;
        if (p->right->left || p->right->right) return -1;
        free(p->right);
        p->right = NULL;
    }

    return 1;
}
```

```c
void preorder(node_t *node) {
    if (node == NULL) return;
    printf("%d ", node->data);
    preorder(node->left);
    preorder(node->right);
}

void inorder(node_t* node) {
    if (node == NULL) return;
    inorder(node->left);
    printf("%d ", node->data);
    inorder(node->right);
}

void postorder(node_t* node) {
    if (node == NULL) return;
    postorder(node->left);
    postorder(node->right);
    printf("%d ", node->data);
}
```

최단 경로 탐색

- 아래 그래프에서 A에서 F로 가는 최단 경로는?
  - A-B-D-F = 2 + 3 + 7 = 12

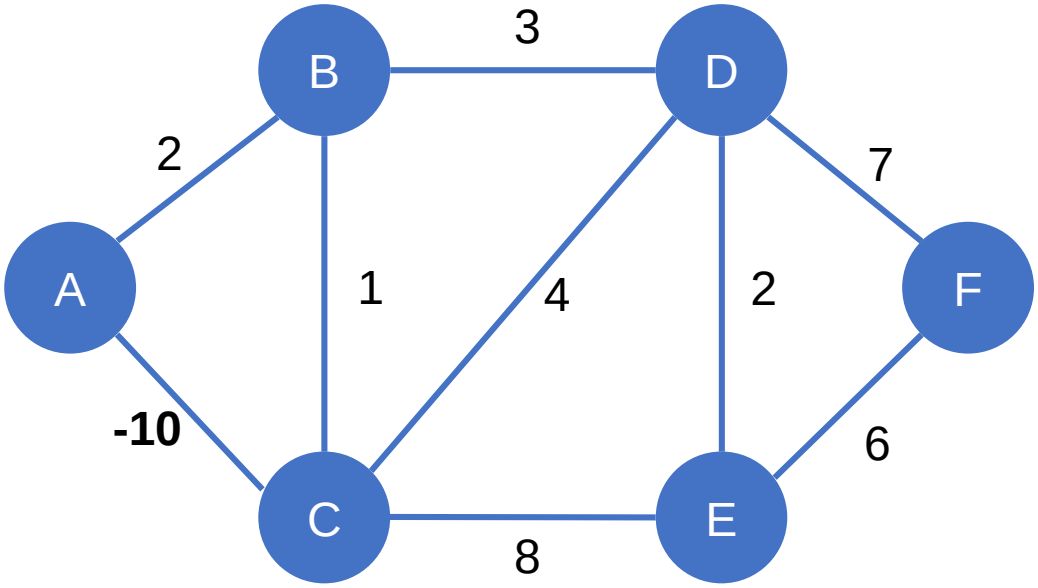


최단 경로 탐색 알고리즘

- 다익스트라 알고리즘 (Dijkstra Algorithm)
  - 주어진 시작 노드로부터 그래프의 다른 모든 노드로 가는 최단 경로를 탐색

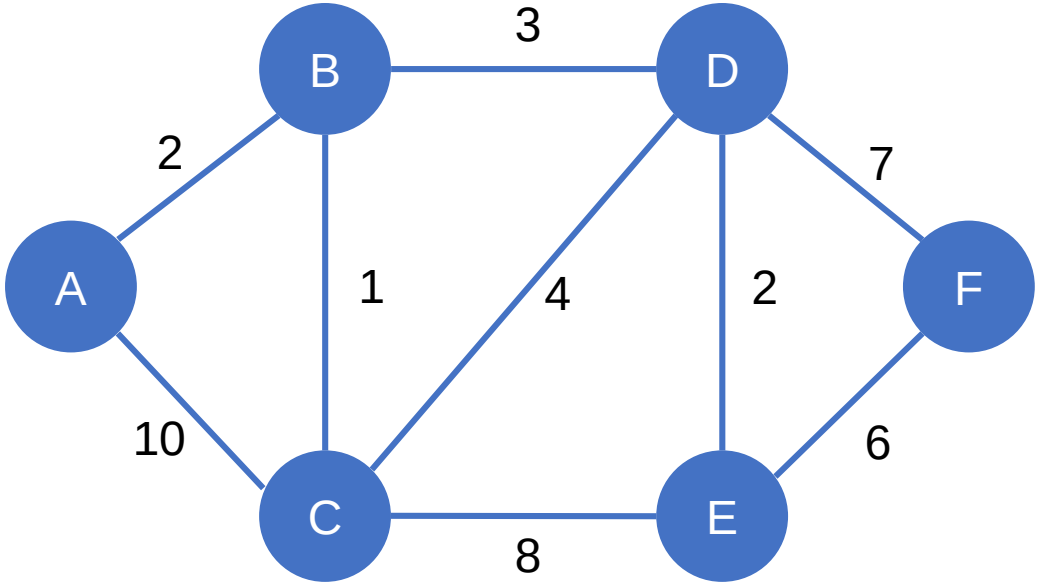
- 플로이드-워셜 알고리즘 (Floyd-Warshall Algorithm)
  - 그래프의 임의의 시작 노드에서 다른 모든 노드로 가는 최단경로를 탐색

음의 가중치를 가지는 사이클



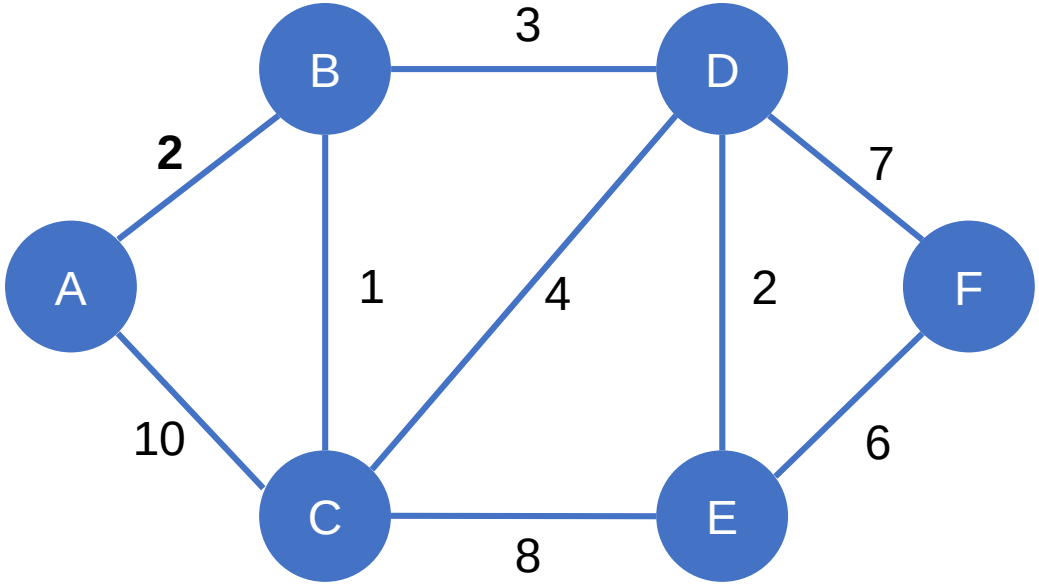
다익스트라 알고리즘

- A에서 시작해서 다른 모든 노드로 가는 최단 경로를 구해보자!
  - A에서 B로 가는 최단경로, A에서 C로 가는 최단경로, … A에서 F로 가는 최단경로



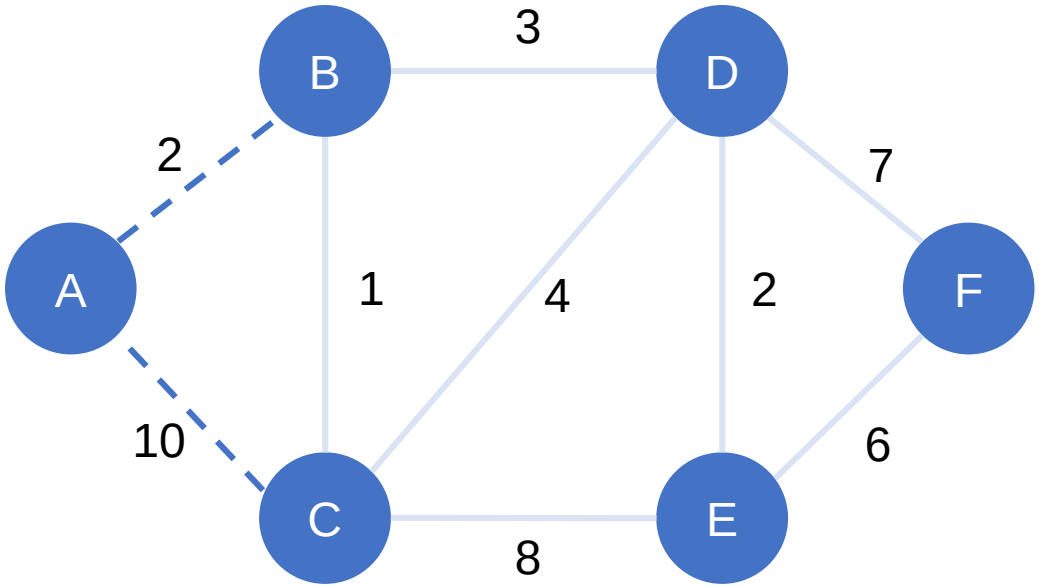
다익스트라 알고리즘

- 착안: A의 간선 중, 항상 최단경로라고 보장이 되는 간선은 무엇인가?
  - A와 연결된 간선 중 최소 가중치를 가지는 간선!
  - 이 간선과 연결된 B에 대한 최단경로를 찾았다.



다익스트라 알고리즘

- 각 노드로의 최단거리를 기록하자.
- A에서 나오는 간선들을 고려하여 최단 거리를 업데이트 한다.



각 노드로의 최단거리

A	B	C	D	E	F
0	?	?	?	?	?

최단 거리 확정 여부

A	B	C	D	E	F
T	F	F	F	F	F

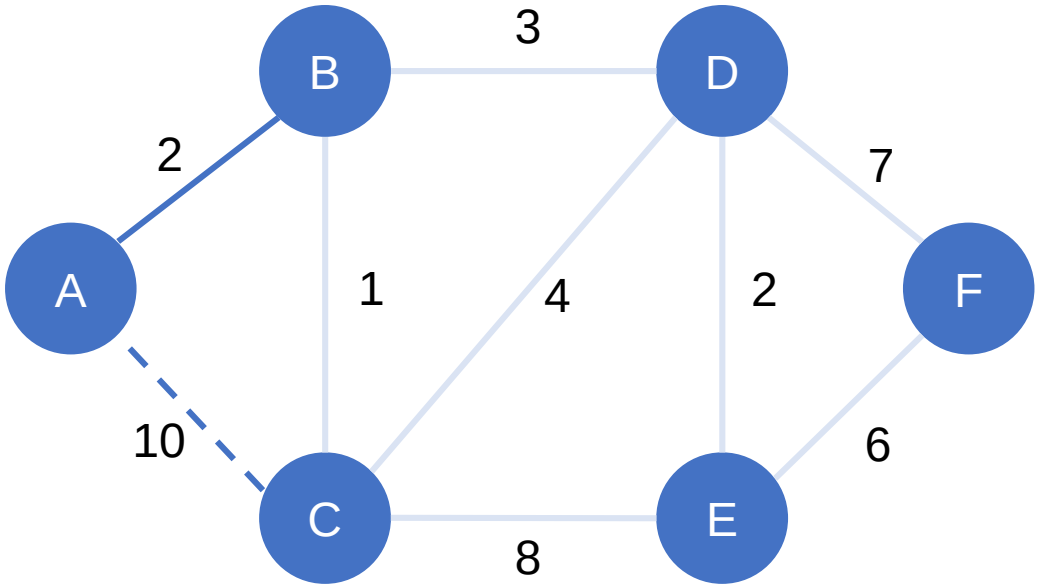
확정된 간선

고려중인 간선

고려되지 않은 간선

다익스트라 알고리즘

- B와 C에 대한 최단 거리가 무한대(?)에서 2와 10으로 업데이트되었다 = 더 좋은 경로를 찾음
- 이 중 더 짧은 B로의 최단 거리를 **확정**한다.



각 노드로의 최단거리

A	B	C	D	E	F
0	**2**	**10**	?	?	?

최단 거리 확정 여부

A	B	C	D	E	F
T	**T**	F	F	F	F

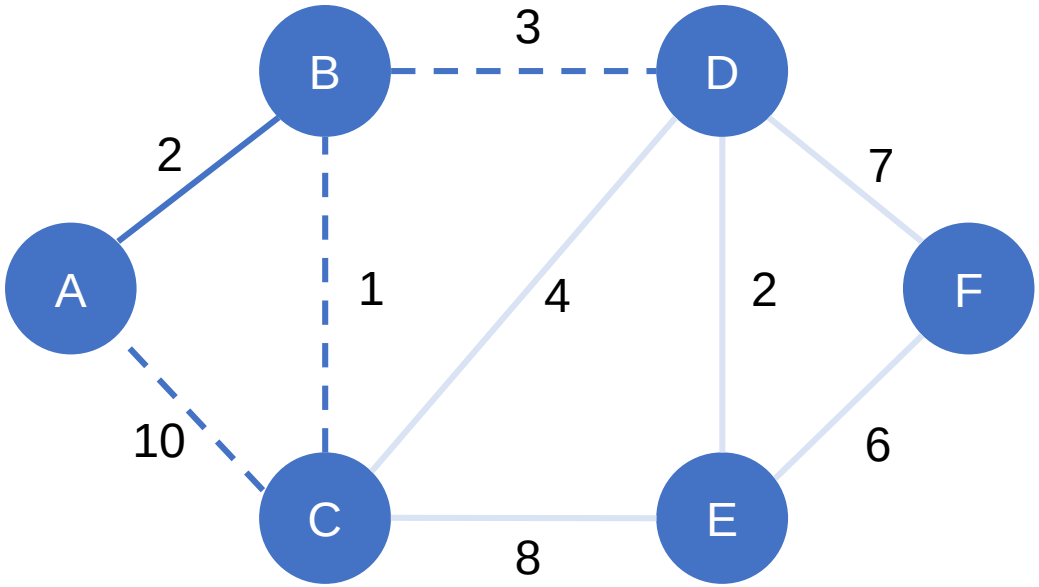
확정된 간선

고려중인 간선

고려되지 않은 간선

다익스트라 알고리즘

- B까지 거리 2인 최단 경로를 찾았다. 이제 B에서 다른 노드로 가는 간선을 고려하자.
- (X 거리) = (B까지의 최단 거리) + (B에서 X까지 거리)



각 노드로의 최단거리

A	B	C	D	E	F
0	2	**3**	**5**	?	?

최단 거리 확정 여부

A	B	C	D	E	F
T	T	F	F	F	F

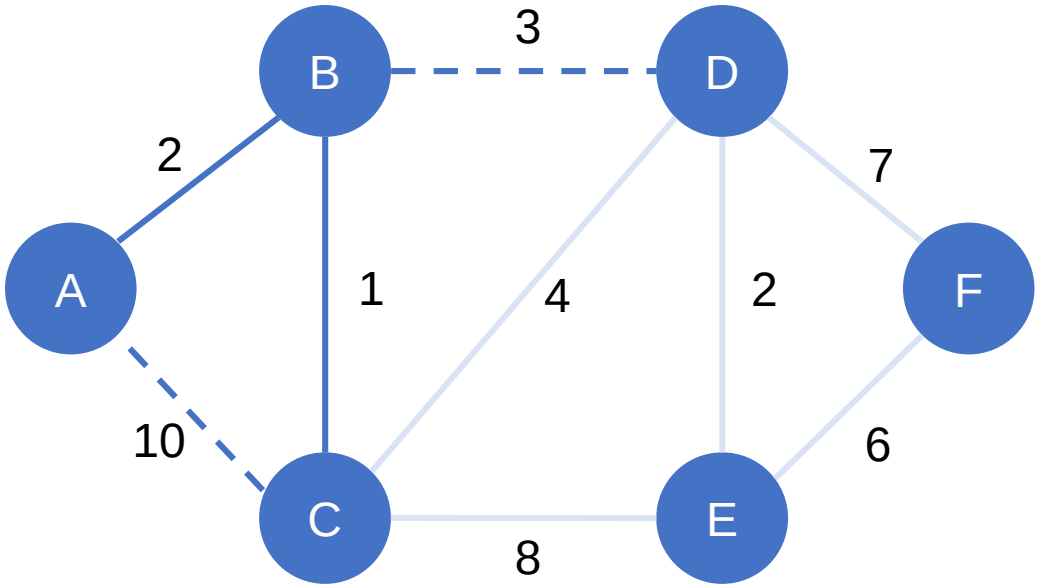
확정된 간선

고려중인 간선

고려되지 않은 간선

다익스트라 알고리즘

- 다시 확정되지 않은 노드 중 가장 짧은 거리를 가지는 C로의 경로를 확정한다.



각 노드로의 최단거리

A	B	C	D	E	F
0	2	3	5	?	?

최단 거리 확정 여부

A	B	C	D	E	F
T	T	**T**	F	F	F

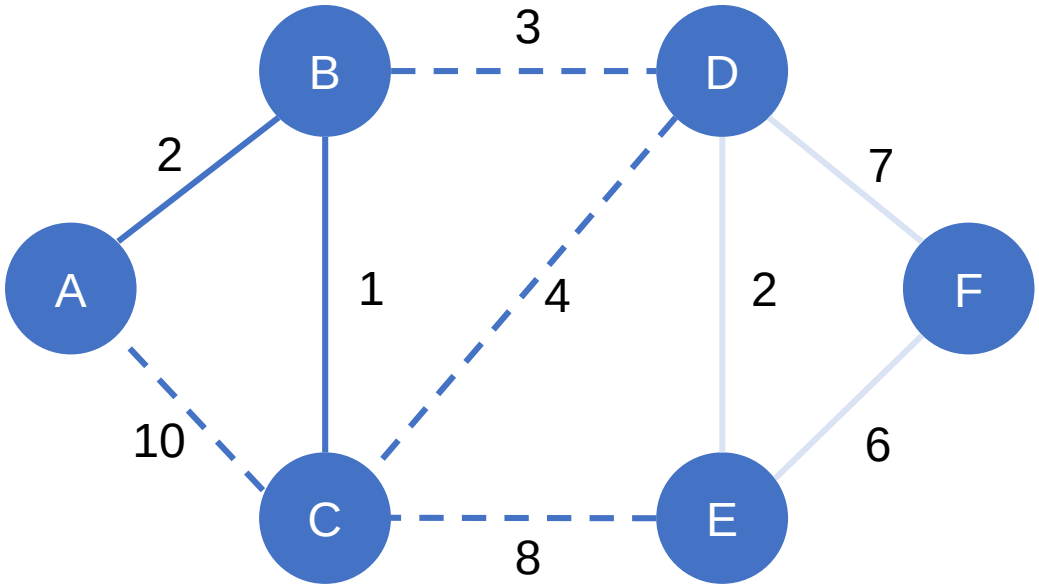
확정된 간선

고려중인 간선

고려되지 않은 간선

다익스트라 알고리즘

- C까지의 최단 경로를 확정했다. C 까지 최단 경로로 간 다음 한 간선 더 가는 D와 E로의 거리를 업데이트하자.
  - D로의 경로는 기존 경로가 더 짧으므로 무시한다.



각 노드로의 최단거리

A	B	C	D	E	F
0	2	3	5	**11**	?

최단 거리 확정 여부

A	B	C	D	E	F
T	T	T	F	F	F

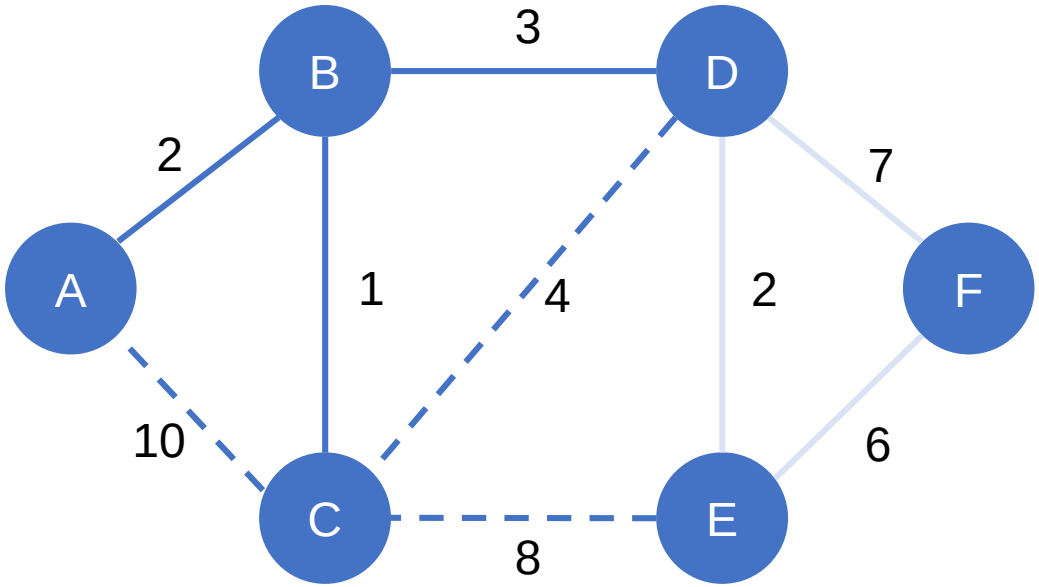
확정된 간선

고려중인 간선

고려되지 않은 간선

다익스트라 알고리즘

- 확정되지 않은 노드 중 최단 거리를 가지는 D를 확정하자.



각 노드로의 최단거리

A	B	C	D	E	F
0	2	3	5	11	?

최단 거리 확정 여부

A	B	C	D	E	F
T	T	T	**T**	F	F

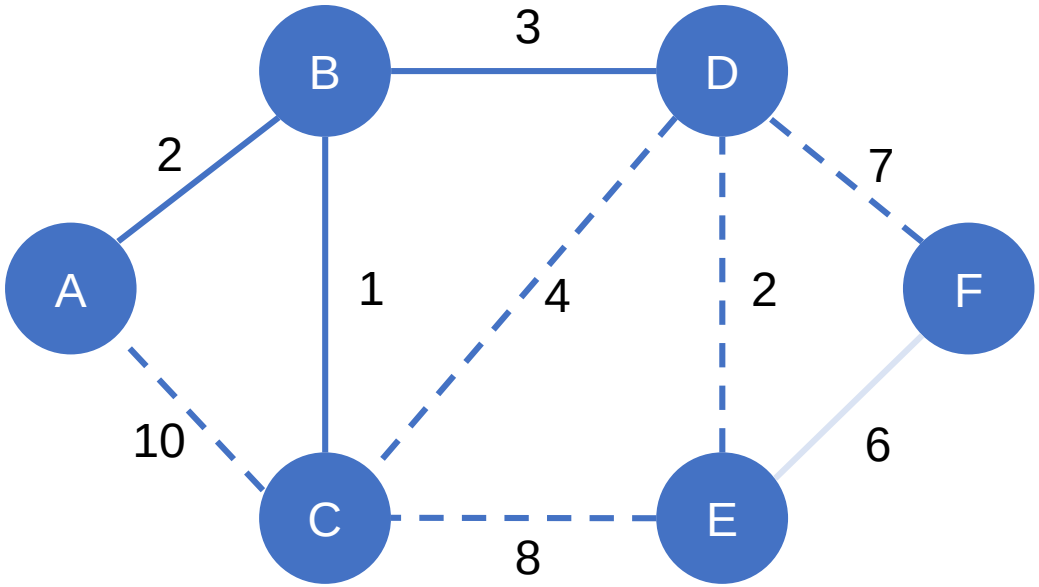
확정된 간선

고려중인 간선

고려되지 않은 간선

다익스트라 알고리즘

- 그 다음, D에서 다른 노드로 가는 경로를 업데이트 하자.
  - C로 가는 경로는 업데이트 되지 않는다.



각 노드로의 최단거리

A	B	C	D	E	F
0	2	3	5	**7**	**12**

최단 거리 확정 여부

A	B	C	D	E	F
T	T	T	T	F	F

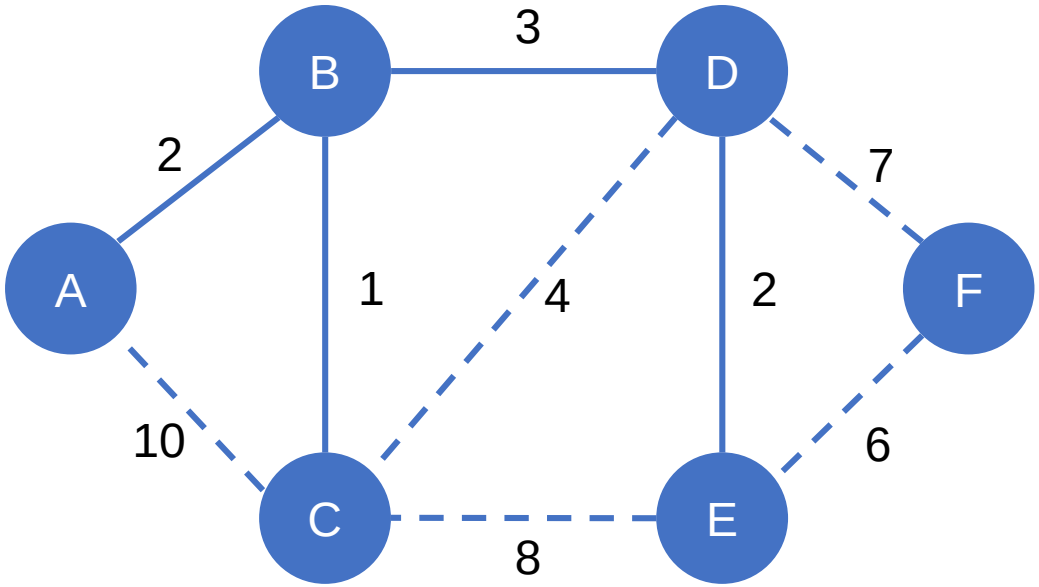
확정된 간선

고려중인 간선

고려되지 않은 간선

다익스트라 알고리즘

- 반복적으로 E를 확정하고 E의 인접한 간선들을 고려한다.



각 노드로의 최단거리

A	B	C	D	E	F
0	2	3	5	7	12

최단 거리 확정 여부

A	B	C	D	E	F
T	T	T	T	**T**	F

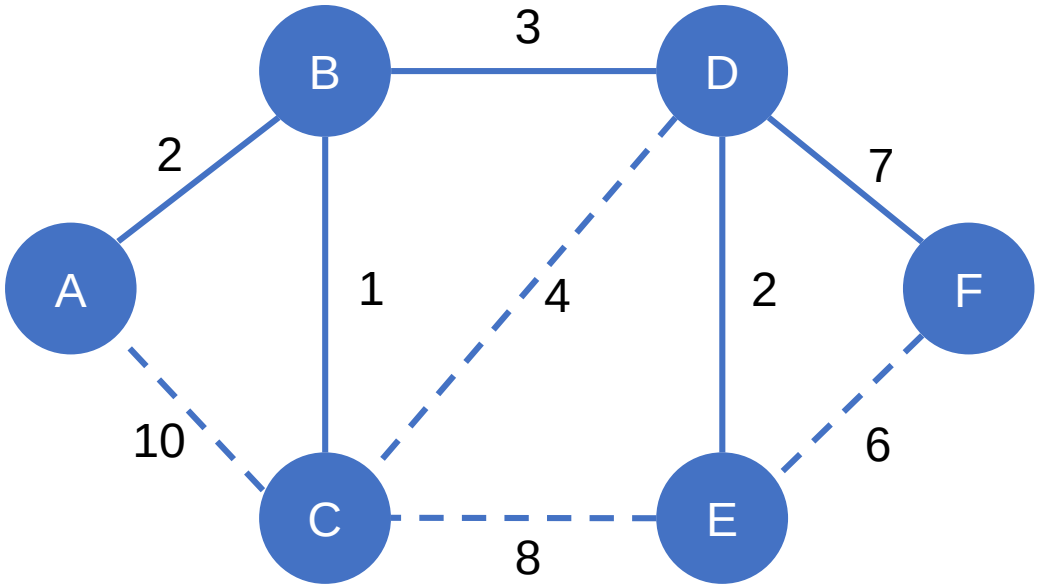
확정된 간선

고려중인 간선

고려되지 않은 간선

다익스트라 알고리즘

- 마지막으로 F를 확정한다.



각 노드로의 최단거리

A	B	C	D	E	F
0	2	3	5	7	12

최단 거리 확정 여부

A	B	C	D	E	F
T	T	T	T	T	**T**

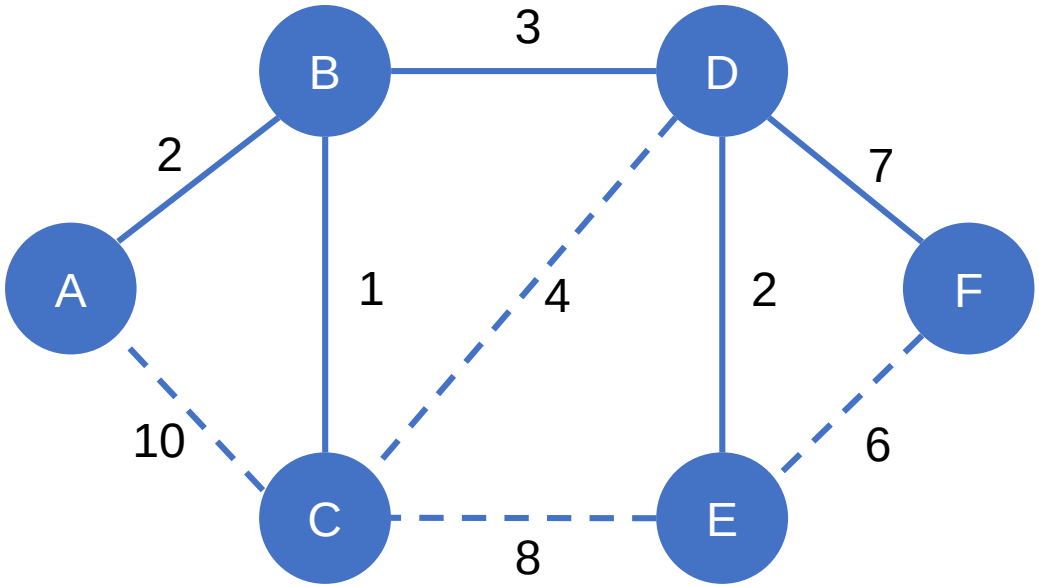
확정된 간선

고려중인 간선

고려되지 않은 간선

다익스트라 알고리즘

- A에서 시작해서 모든 노드로의 최단 거리를 찾았다!



각 노드로의 최단거리

A	B	C	D	E	F
0	2	3	5	7	12

최단 거리 확정 여부

A	B	C	D	E	F
T	T	T	T	T	**T**

확정된 간선

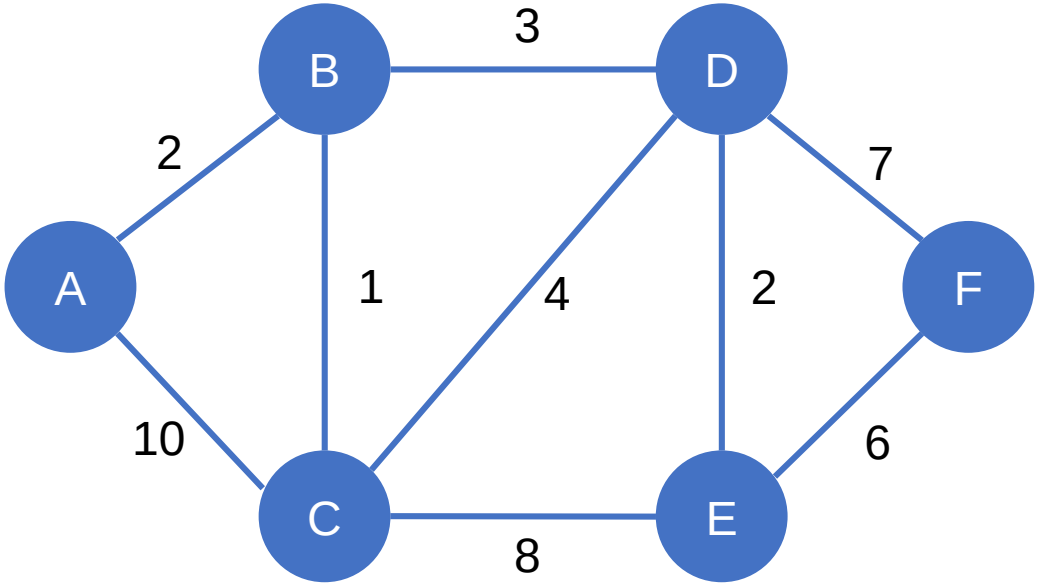
고려중인 간선

고려되지 않은 간선

```c
#include <stdio.h>

#define N 6
#define INF 9999

int distance[N][N], shortest[N], visited[N];

void set_edge(int a, int b, int w) {
    distance[a][b] = distance[b][a] = w;
}

// 방문하지 않은 정점 중 최단 거리를 가진 노드의 번호를 돌려준다.
int get_shortest_node() {
    int min = INF, min_v;
    for (int i = 0; i < N; i++) {
        if (!visited[i] && min > shortest[i]) {
            min = shortest[i];
            min_v = i;
        }
    }
    return min_v;
}
```

```c
int main() {
    int i, j, v;

    for (i = 0; i < N; i++) for (j = 0; j < N; j++) distance[i][j] = INF;
    for (i = 0; i < N; i++) shortest[i] = INF;

    set_edge(0, 1, 2); set_edge(0, 2, 10); set_edge(1, 2, 1); set_edge(1, 3, 3);
    set_edge(2, 3, 4); set_edge(2, 4, 8); set_edge(3, 4, 2); set_edge(3, 5, 7);
    set_edge(4, 5, 6);

    int start = 0;
    shortest[start] = 0;

    for (i = 0; i < N; i++) {
        v = get_shortest_node();
        visited[v] = 1;

        for (j = 0; j < N; j++) {
            if (shortest[j] > shortest[v] + distance[v][j])
                shortest[j] = shortest[v] + distance[v][j];
        }
    }

    for (i = 0; i < N; i++) {
        printf("From %d, To %d = %d\n", start, i, shortest[i]);
    }
    return 0;
}
```

실행 결과




```
From 0, To 0 = 0
From 0, To 1 = 2
From 0, To 2 = 3
From 0, To 3 = 5
From 0, To 4 = 7
From 0, To 5 = 12
```

핵심 부분

- shortest[j] > shortest[v] + distance[v][j]

- v는 최단 경로가 방금 확정된 정점이다.

- 만약, (이제까지 알려진) j로의 최단 거리가 (새로 확정한) v까지 간 다음 여기서 간선 하나만 더 가서 j로 이르는 경로보다 길다면? 더 짧은 경로를 찾았으므로 업데이트

핵심 부분

- shortest[j] > shortest[v] + distance[v][j]
- 만약, shortest[j] = INF 였다면?
- 만약, v와 j가 연결되어 있지 않다면?
- 만약, 음의 가중치를 가진 간선이 있다면?

플로이드-워셜 알고리즘

- 모든 노드 쌍에 대해 최단 경로를 구하는 알고리즘

- D[i][j] = 정점 i로부터 정점 j로 가는 최단 거리

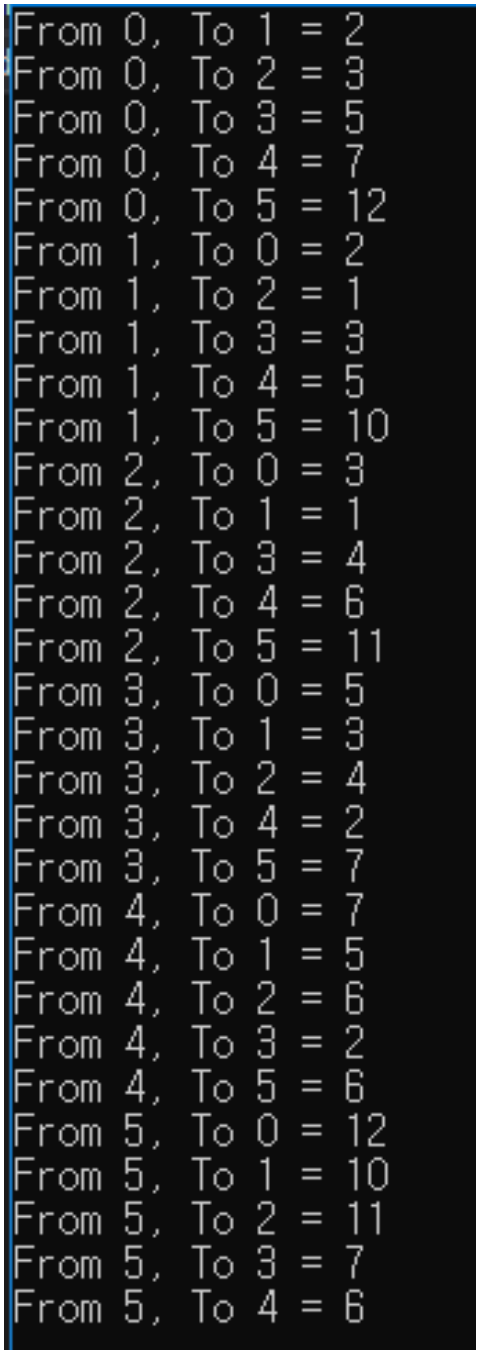
- 그래프를 인접 행렬 W[i][j]로 나타내었다면 초기 D[i][j] = W[i][j]가 된다.
  - 단, 연결되지 않은 노드의 경우 무한대의 가중치로 가정

플로이드-워셜 알고리즘

- i에서 j로 가려고 한다. 이제까지 알려진 최단 거리는 D[i][j]이다.
  - D[i][j]

- 만약 i에서 k로 간 다음, k에서 j로 가는 경로가 더 짧다면? 업데이트한다.
  - D[i][j] > D[i][k] + D[k][j]

```c
#include <stdio.h>

#define N 6
#define INF 9999

int w[N][N], d[N][N];

void set_edge(int a, int b, int w2) {
    w[a][b] = w[b][a] = w2;
}

int main() {
    int i, j, k;

    for (i = 0; i < N; i++) for (j = 0; j < N; j++) w[i][j] = INF;

    set_edge(0, 1, 2); set_edge(0, 2, 10); set_edge(1, 2, 1); set_edge(1, 3, 3);
    set_edge(2, 3, 4); set_edge(2, 4, 8); set_edge(3, 4, 2); set_edge(3, 5, 7); set_edge(4, 5, 6);

    for (i = 0; i < N; i++) for (j = 0; j < N; j++) d[i][j] = w[i][j];

    for (k = 0; k < N; k++)
        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++)
                if (d[i][j] > d[i][k] + d[k][j])
                    d[i][j] = d[i][k] + d[k][j];

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            if (i != j) printf("From %d, To %d = %d\n", i, j, d[i][j]);
        }
    }
    return 0;
}
```


From 0, To 1 = 2
From 0, To 2 = 3
From 0, To 3 = 5
From 0, To 4 = 7
From 0, To 5 = 12
From 1, To 0 = 2
From 1, To 2 = 1
From 1, To 3 = 3
From 1, To 4 = 5
From 1, To 5 = 10
From 2, To 0 = 3
From 2, To 1 = 1
From 2, To 3 = 4
From 2, To 4 = 6
From 2, To 5 = 11
From 3, To 0 = 5
From 3, To 1 = 3
From 3, To 2 = 4
From 3, To 4 = 2
From 3, To 5 = 7
From 4, To 0 = 7
From 4, To 1 = 5
From 4, To 2 = 6
From 4, To 3 = 2
From 4, To 5 = 6
From 5, To 0 = 12
From 5, To 1 = 10
From 5, To 2 = 11
From 5, To 3 = 7
From 5, To 4 = 6

플로이드-워셜 알고리즘의 이해

```
for (k = 0; k < N; k++)
        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++)
                if (d[i][j] > d[i][k] + d[k][j])
                    d[i][j] = d[i][k] + d[k][j];
```

- k=0, 노드 쌍 (i, j)에 대해 0을 지나는 경로 고려, i -> 0 -> j
- k=1, 노드 쌍 (i, j)에 대해 1을 지나는 경로 고려, i -> 1 -> j

- 3에서 4로 갈 때, 0과 1을 모두 지나는 경로는 문제 없나요?
  - 3 -> 0 -> 1 -> 4가 최소 경로라면,
  - k=0, i=3, j=1일 때, 3 -> 0 -> 1 경로가 고려되고,
  - k=1, i=3, j=4일 때, 3 -> 0 -> 1 -> 4 경로가 계산된다.

요약

- 최단 경로: 그래프에서 두 개의 노드 간의 경로 중 가중치의 합이 가장 적은 경로

- 다익스트라 알고리즘 (Dijkstra Algorithm)
  - 시작 노드로부터 다른 모든 노드로의 최단 거리
  - 음의 가중치를 가진 간선이 없어야

- 플로이드-워셜 알고리즘 (Floyd-Warshall Algorithm)
  - 모든 노드 쌍에 대한 최단 거리
  - 음의 가중치를 가진 사이클이 없어야

출석 인정을 위한 보고서 제출

- 아래 질문에 대한 답을 포털에 제출하시오. 양식은 자유.

- 어떤 그래프 G의 정점의 개수를 V 간선의 개수를 E라고 하고 인접행렬을 사용하여 표현했을 때,

- 다익스트라 알고리즘의 시간복잡도는?
- 플로이드-워셜 알고리즘의 시간복잡도는?
- (선택 질문) 다익스트라 알고리즘을 우리가 배운 자료구조를 활용하여 더 빨리 만들 수 있을까?