DAY3
과제 2번 보고서

2023741024
로봇학부 박건후

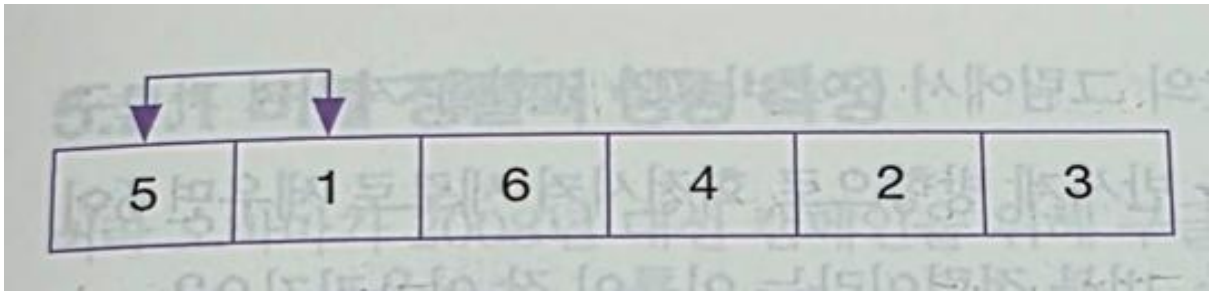
목차

(1) 정렬

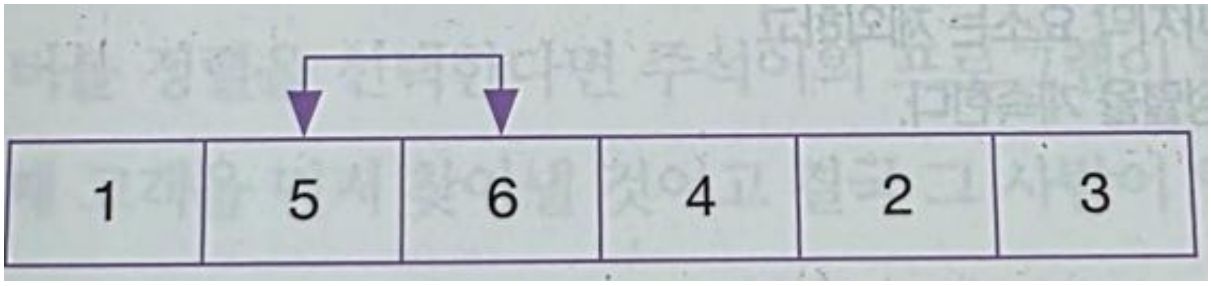
(1-1) 버블 정렬

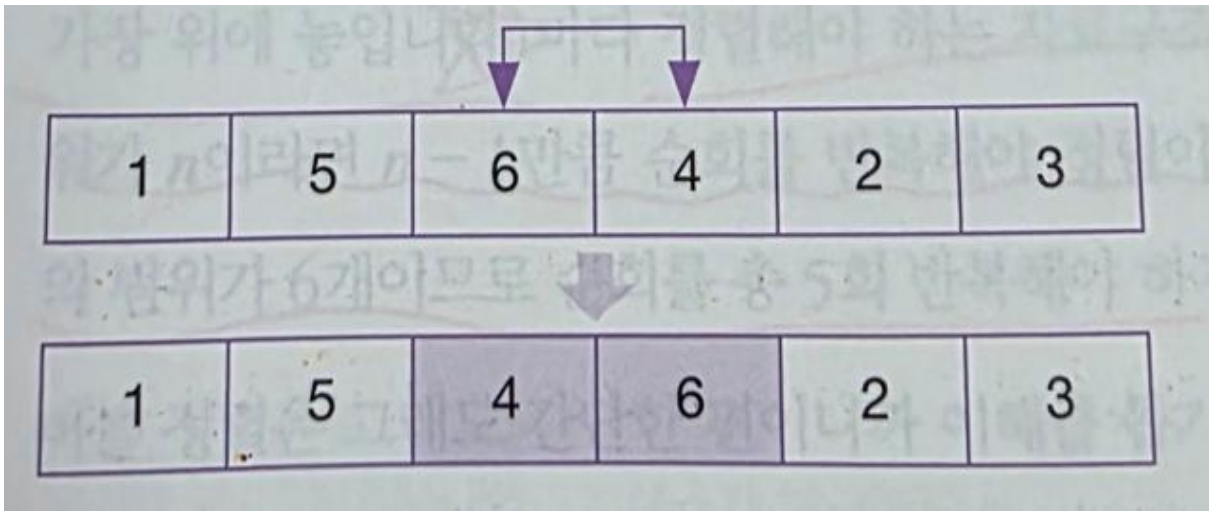
작동 방식:

자료구조를 순회하며 이웃한 요소들끼리 데이터를 교환하며 정렬 수행.



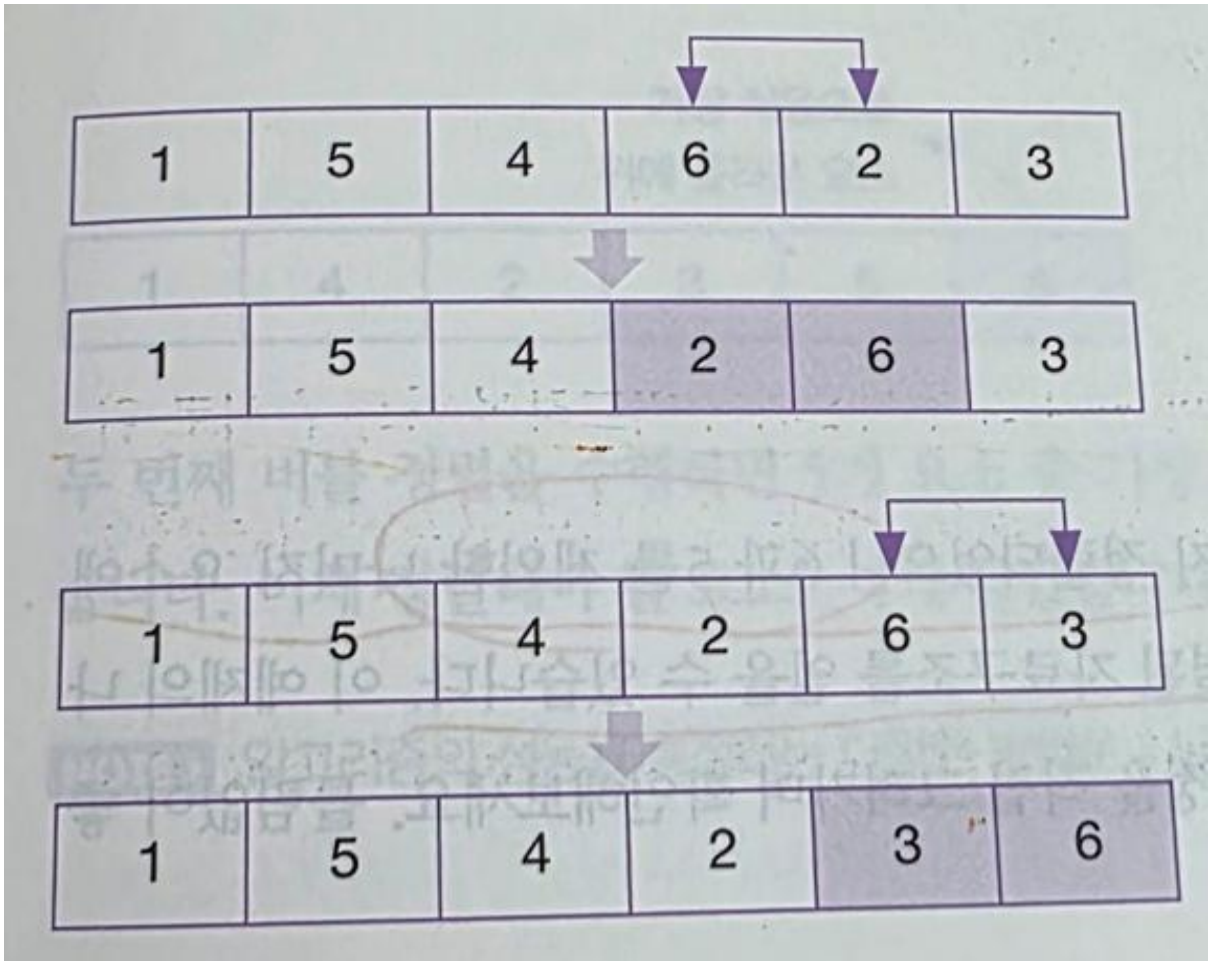
[1]: 맨 왼쪽부터 시작하여 이웃한 요소와 swap.





[2]: 오름차순 정렬이라면 이웃한 요소가 자기 자신보다 작다면 swap 하고 크다면 swap X.

그 이웃한 요소를 자기 자신으로 취하고. 내림차순이라면 그 반대.



[3]: 이를 반복하면 제일 큰 수가 제일 뒤로 감. 이를 반복

성능:

-n(n-1)/2만큼의 비교를 수행하기에 -> $O(n^2)$.

-이런 속도는 상용적으로 쓰기에 문제가 큼.

- 하지만 구현이 간단.

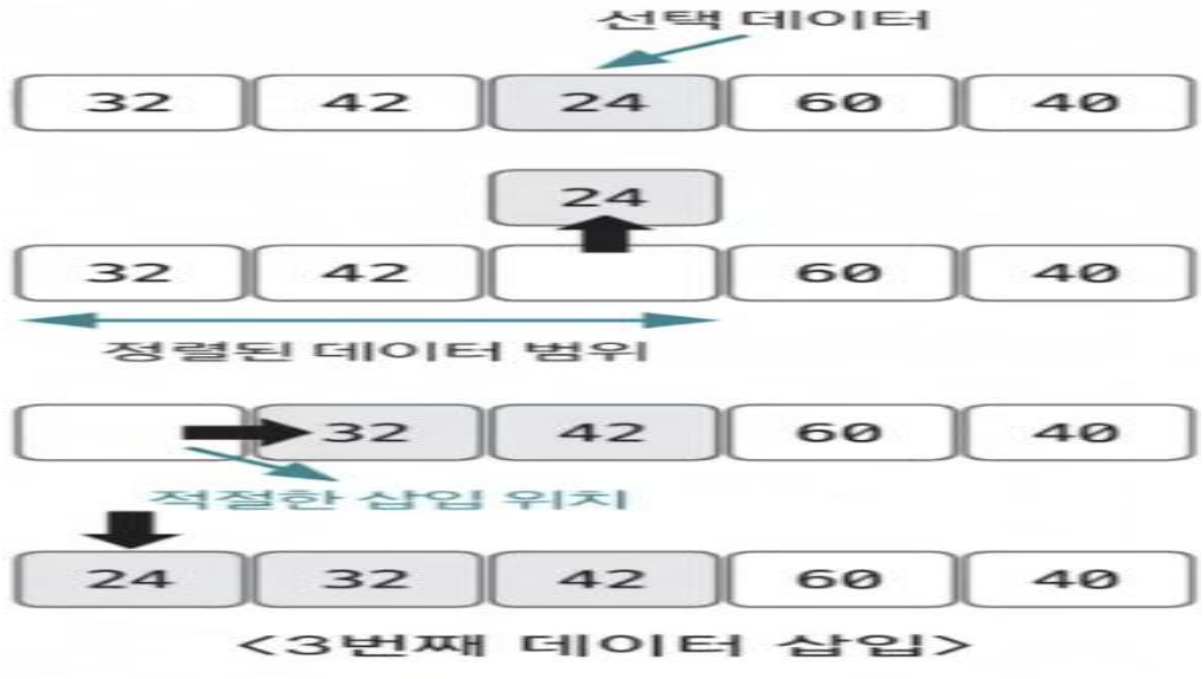
(1-2) 삽입 정렬

작동 방식:

- 자료구조를 순회하며 순서에 어긋나는 요소를 찾고, 올바른 위치에 다시 삽입.
- 버블 정렬만큼 구현이 간단하므로 많이 활용되며, 성능도 버블 정렬과 비슷.
- 버블 정렬과 반대로 정렬 범위를 1씩 늘림.

<2번째 데이터 삽입>

- 처음에는 정렬 범위가 2개이고 그 범위 내에서만 정렬



- 다음에 정렬 범위가 1증가하고 범위의 마지막 요소를 정렬. 이를 반복

성능:

- $O(n^2)$으로 버블 정렬과 성능 비슷.
- 최악의 경우에는 성능이 비슷함.
- 최선의 경우에는 비교연산을 한 번도 하지 않기에 효율적.
- 비교적 크기가 작은 자료구조 정렬할 때는 삽입 정렬이 더 유리.

(1-3) 퀵 정렬

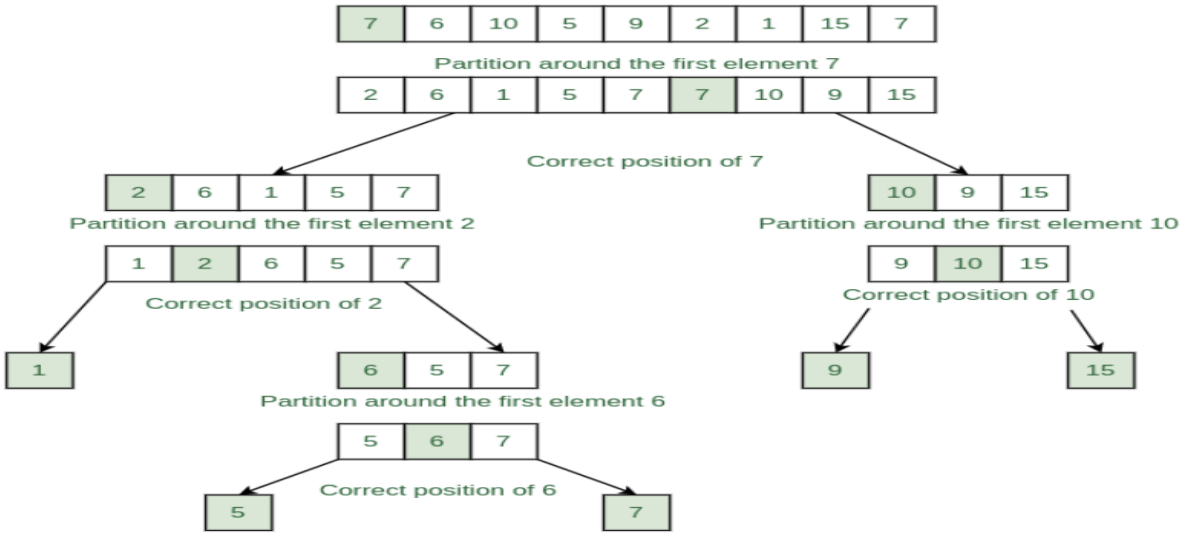
작동 방식:

- 퀵 정렬은 분할 정복을 바탕에 둔 알고리즘.
- 분할 정복은 전체를 공략하는 대신 전체를 잘게 나누어 공략하는 기법

[1]: 자료 내에서 나눌 기준 요소 선정 및 정렬 대상 분류

[2]: 그 기준 요소보다 작은 건 왼쪽에 큰 것을 오른쪽에 배치

[3]: 그 후 왼쪽 오른쪽 내에서도 각각 기준 요소를 정하고 또 각각의 기준 요소의 값과 비교하여 왼쪽, 오른쪽으로 나눔, 이를 반복. 아래 사진과 같음



성능:

- 최선의 경우는 쪼갤 때마다 같은 개수로 쪼개지는 경우. 이 때는 log2n의 복잡도를 가짐.
- 최악의 경우에는 쪼갤 때마다 그 요소보다 모두 크거나 모두 작은 경우. 이 때는 버블 정렬과 같은 복잡도를 가짐.

- 위의 2개의 경우는 모두 드물게 발생하며 평균적으로는 nlog2n의 복잡도를 가진다.

(1-4) 병합 정렬

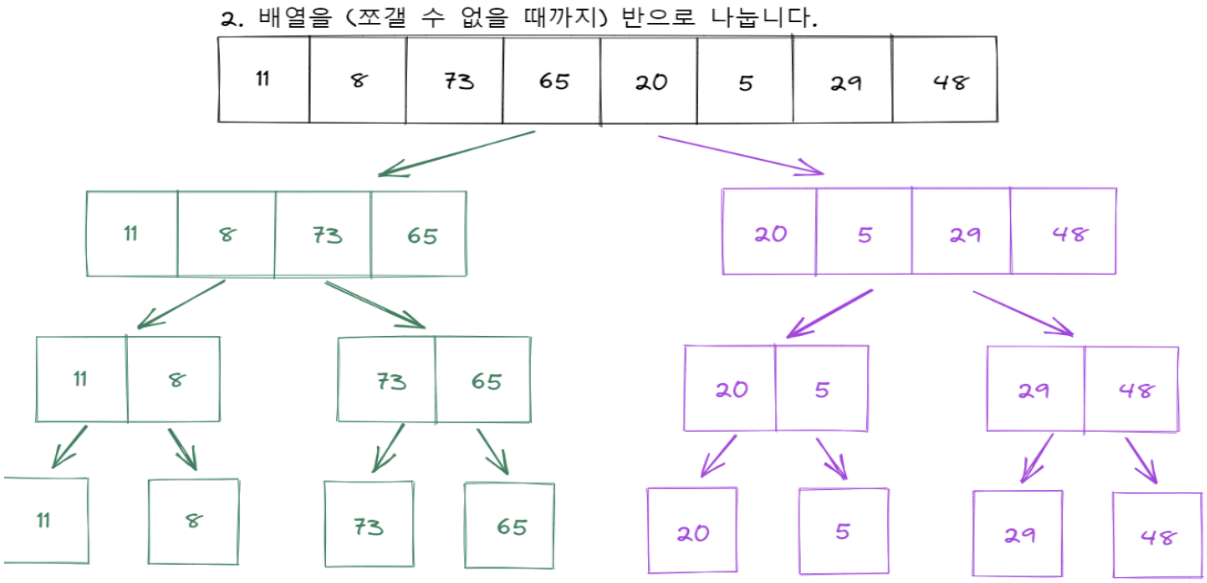
작동 방식:

[1] 정렬할 데이터를 반으로 나누기.

[2] 나뉜 하위 데이터의 크기가 2이상이면 이 하위 데이터에 대해 단계 1을 반복.

[3] 하위 데이터 둘을 정렬하고 병합하여 원래대로 하나의 데이터로 만들기.

[4] 데이터가 원래대로 모두 하나가 될 때까지 [3]을 반복한다.



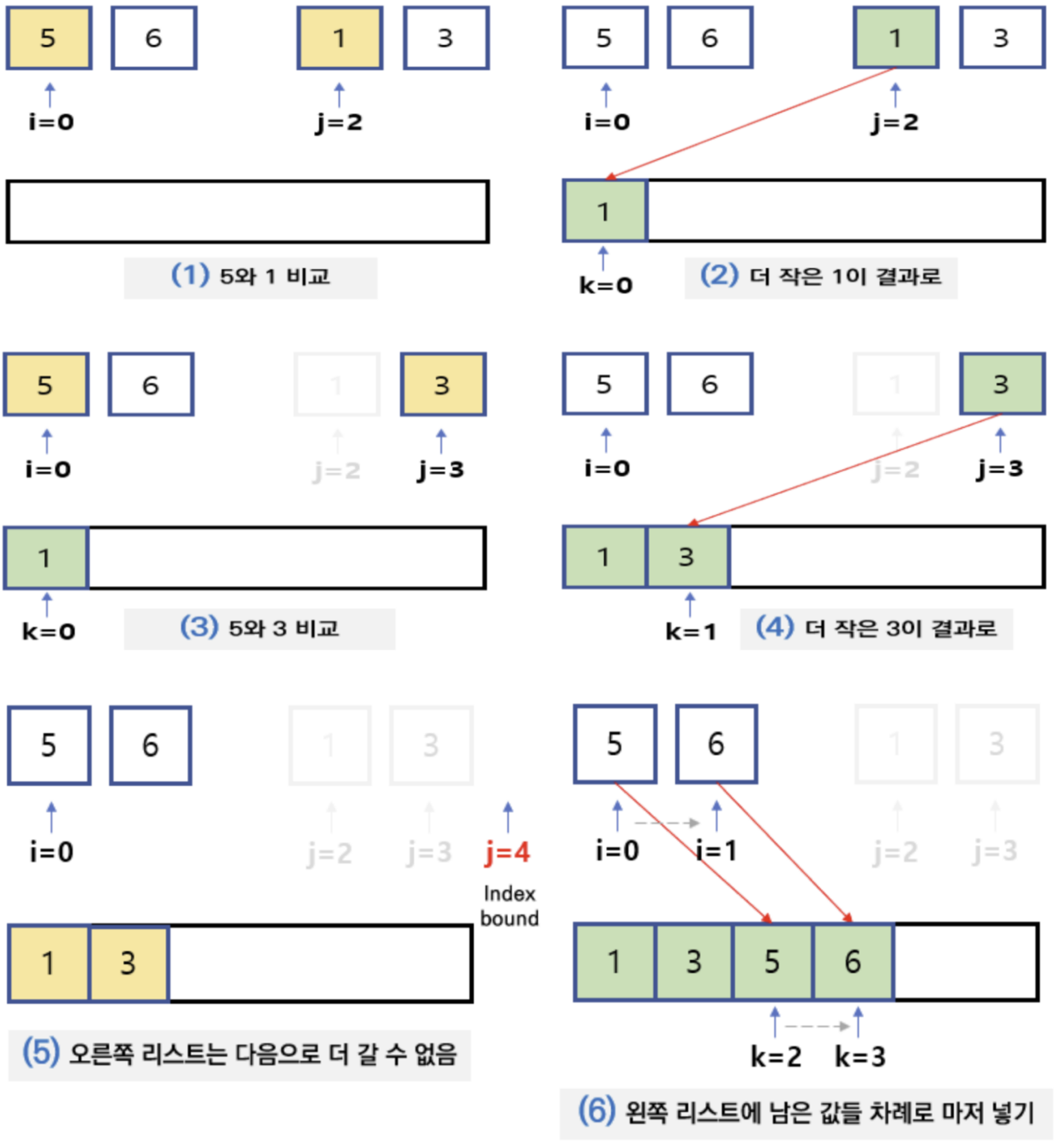
- 11, 8을 정렬하여 8, 11로 합치고, 73, 65를 정렬하여 65, 73으로 합침.
- 나중에는 8, 11, 65, 73과 5, 20, 29, 48을 정렬하여 최종 정렬된 데이터가 되게 하는 방식임.

- 정렬+병합 구현법

[1] 두 데이터를 합한 것만큼 비어 있는 공간을 마련.

[2] 두 데이터의 첫 번째 요소들을 비교 후 작은 요소를 새 데이터에 추가. 해당 추가된 요소 삭제 후 바로 뒷 요소는 첫 요소로 이동.

[3] 양쪽 데이터가 빌 때까지 [2]를 반복.



(2) 탐색

(2-1) 이진 탐색

- 이진 탐색은 정렬된 데이터에서 사용할 수 있는 고속 탐색 알고리즘
- 이 알고리즘의 핵심이 탐색 범위를 1/2씩 줄여나가는 방식
- log2n의 시간으로 탐색 소요 시간은 미미하게 증가한다는 의미.

작동 방식:

[1] 데이터 중앙에 있는 요소를 고르기

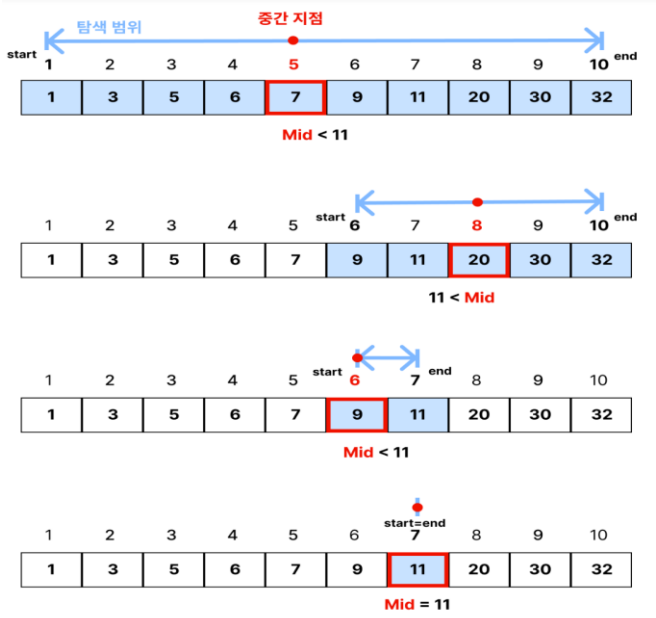
[2] 중앙 요소값과 찾고자 하는 목표값을 비교함

[3] 목표 값이 중앙 요소값보다 작다면 중앙을 기준으로 데이터 왼편에 대해, 크다면 오른편에 대해 이진 탐색을 수행함.

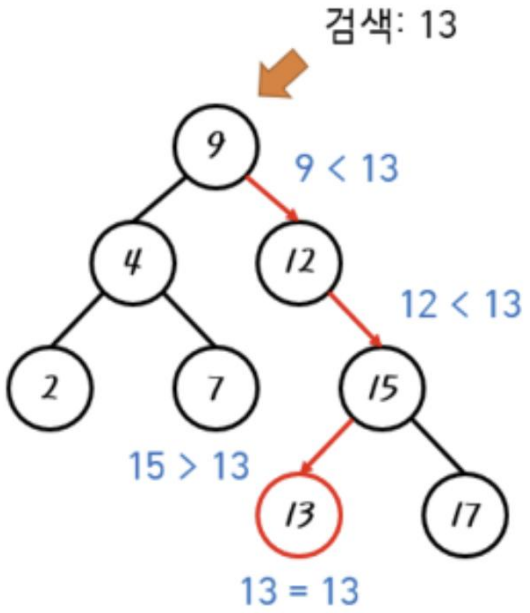
[4] 찾고자 하는 값을 찾을 때까지 이를 반복함

이진 탐색 트리

- 이진 탐색을 위한 이진 트리인 알고리즘이 아닌 자료구조
- 이진 탐색 트리는 왼쪽 자식 노드는 나보다 작고, 오른쪽 자식 노드는 나보다 큼.
- 각 노드가 그 노드의 왼쪽 하위 트리, 오른쪽 하위 트리의 중앙값
- 이진 탐색 트리가 한쪽으로 치우친 경우에는 비교를 많이 하게 되어 비효율적
- 이를 해결하기 위해 레드 블랙 트리를 사용.



(배열의 이진 탐색)



(이진 탐색 트리의 이진 탐색)

(2-2) 깊이 우선 탐색

더 나아갈 길이 보이지 않을 때까지 깊이 들어간다는 마인드

작동 방식:

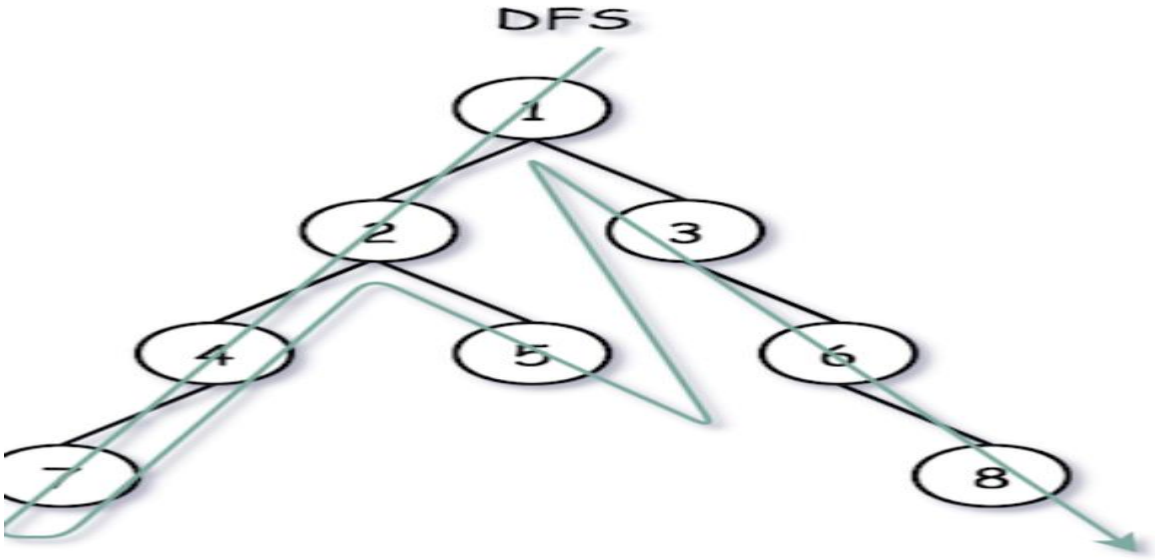
[1] 시작 정점을 밟은 후 이 정점을 방문했음으로 표시.

[2] 이 정점과 이웃 정점 중에 아직 방문하지 않은 곳을 선택하여 이를 시작 정점으로 삼고 즉 [1]을 다시 수행. (재귀함수)

[3] 더 이상 방문하지 않은 이웃 정점이 없으면 이전 정점으로 돌아가 단계 [2]를 수행.

[4] 이전 정점으로 돌아가도 더 이상 방문할 이웃 정점이 없다면 그래프의 모든 정점을 방문했다는 뜻이므로 탐색을 종료.

- 마치 미로 찾기를 하는 것과 같은데, 실제 미로 찾기 문제를 푸는 데에도 사용함.



- 이 깊이 우선 탐색 알고리즘은 그래프 정렬 알고리즘인 위상 정렬이라는 알고리즘의 기반이 됨.
- 이 외에도 그래프를 이용한 다른 알고리즘에서 초석으로 사용됨.

(2-3) 너비 우선 탐색

꼼꼼하게 좌우를 살피며 다니자는 마인드

- 시작 정점을 지난 후 깊이가 1인 모든 정점을 방문하고 그 다음에는 깊이가 2인 모든 정점 방문함.
- 이런 식으로 한 단계씩 깊이를 더해감-
- 더 이상 방문할 정점이 없을 때 탐색 종료.

동작 방식:

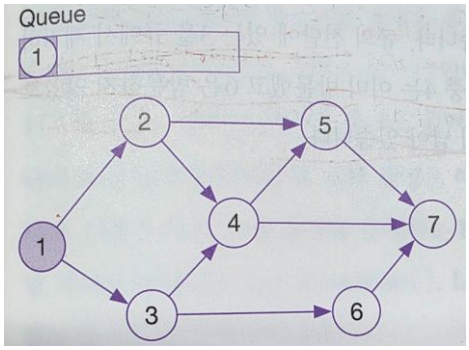
[0] 탐색을 도와줄 큐 필요

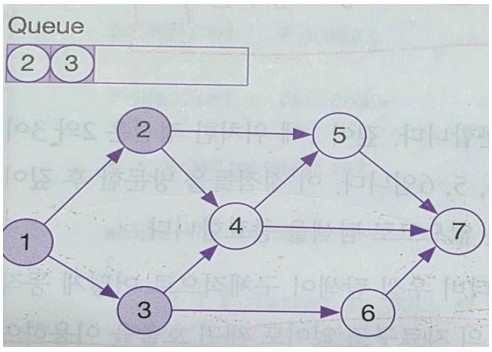
[1] 시작 정점을 방문했음으로 표시하고 큐에 삽입.

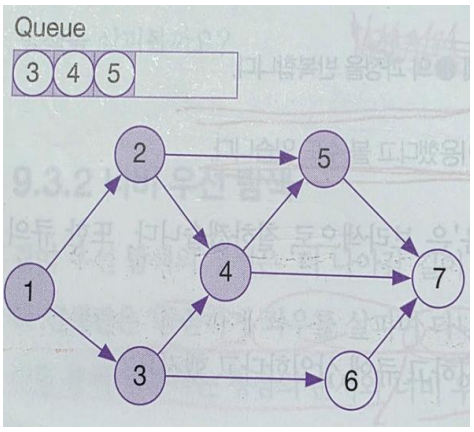
[2] 큐로부터 정점을 제거. 제거한 정점의 인접 정점 중에서 아직 방

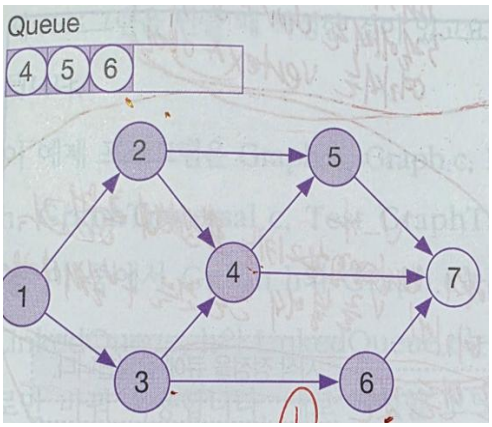
문하지 않은 곳을 방문했음으로 표시 후 큐에 삽입.

[3] 큐가 비면 탐색이 끝난 것. 큐가 빌 때까지 [2] 반복









- 너비 우선 탐색은 그래프에서 최단 경로를 찾는 알고리즘의 기반이 됨.

(3) 그 외 그래프 관련 알고리즘

(3-1) 프림 알고리즘

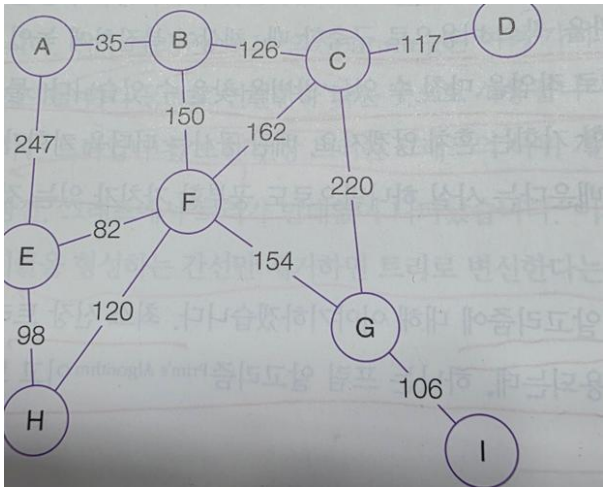
- 그래프로부터 최소 신장 트리를 만드는 알고리즘이다.
- 최소 신장 트리를 만들기 위해서는 간선에 가중치가 실려 있어야 함.

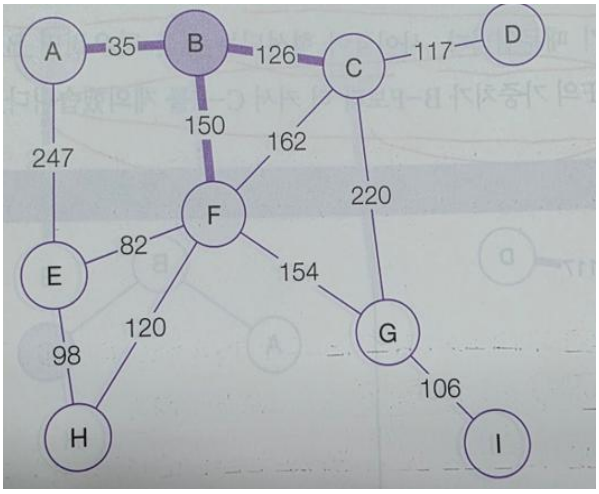
작동 방식:

[1] 그래프에서 임의의 정점을 시작 정점으로 선택하여 최소 신장 트리의 뿌리 노드로 삽입.

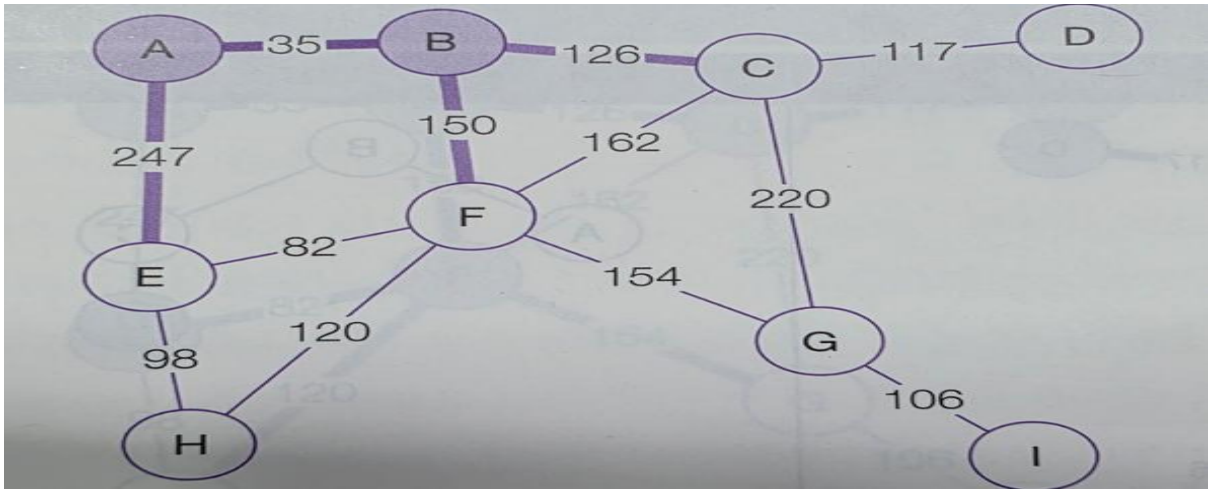
[2] 최소 신장 트리에 삽입된 정점들과 이 정점들의 모든 인접 정점 사이의 간선의 가중치를 조사 간선 중에 가장 가중치가 작은 것을 골라 간선과 연결된 정점을 최소 신장 트리에 삽입. 단, 사이클을 형성해서는 안 됨.

[3] [2]의 과정을 반복하다가 최소 신장 트리가 그래프의 모든 정점을 연결하게 되면 알고리즘을 종료.

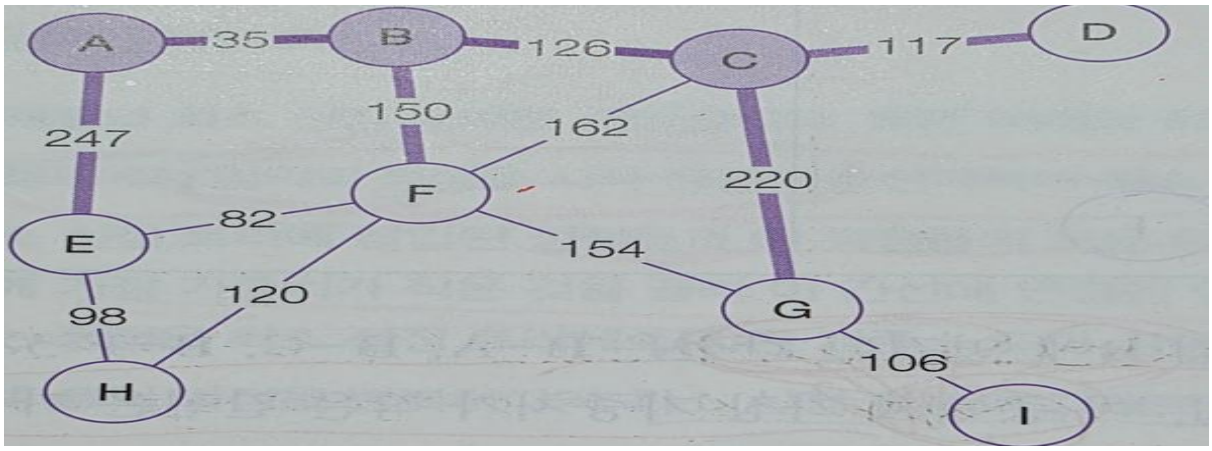




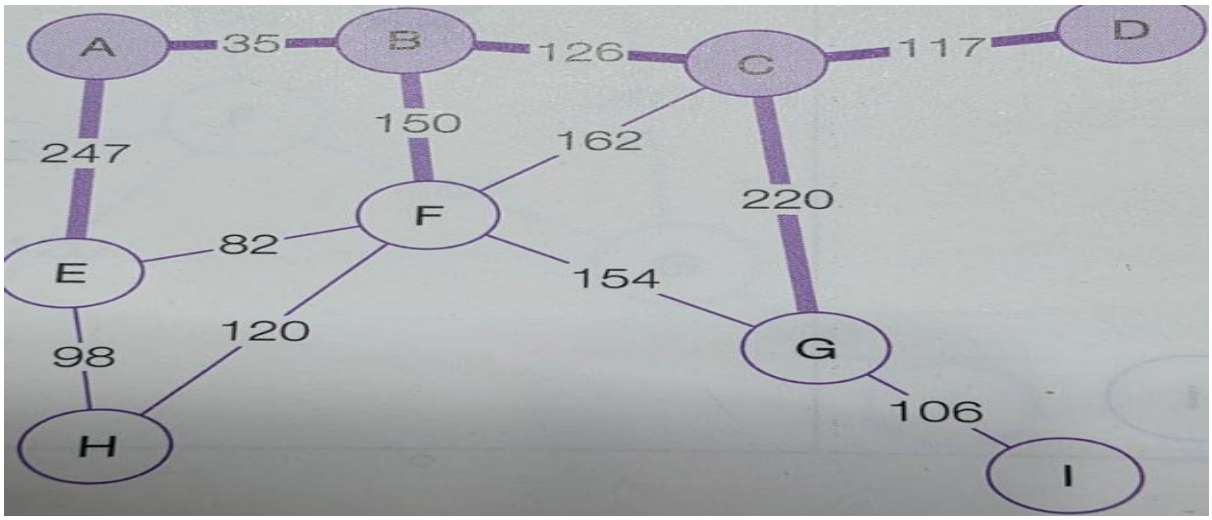
임의의 정점으로 B를 잡음. B노드가 최소 신장 트리의 뿌리 노드가 됨.



- B와 연결된 간선은 B-A, B-C, B-F 3개.
- 가장 가중치가 작은 간선이 35인 B-A이므로 A를 최소 신장 트리에 추가. (우선순위 큐를 통해 가중치 값이 제일 낮은 것부터 정점 조사함)

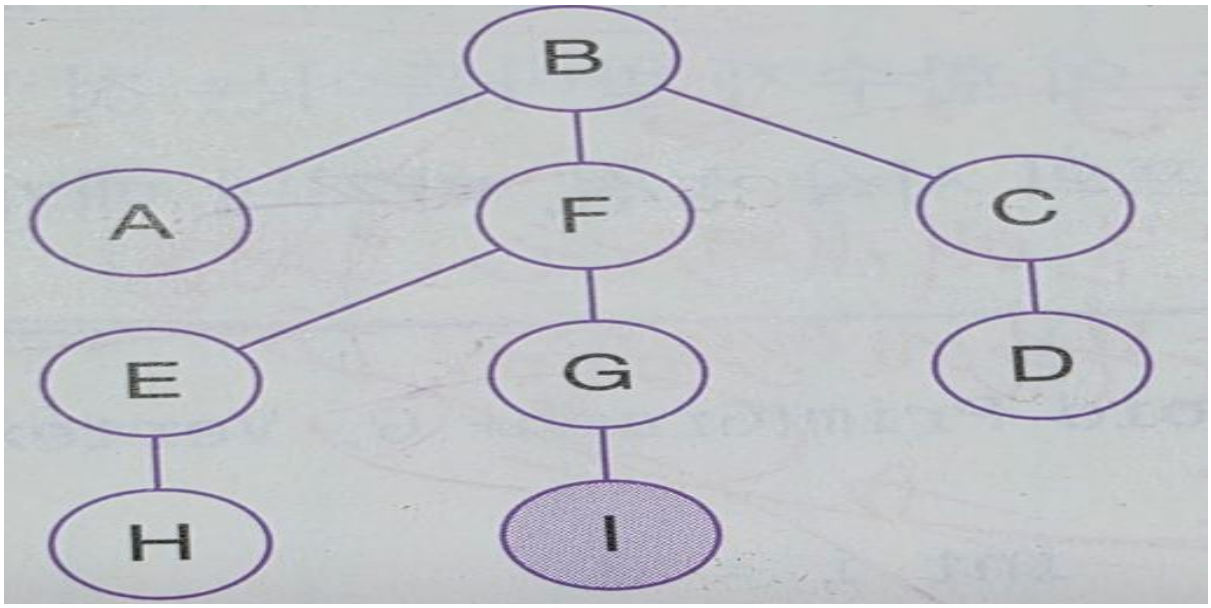


- 이들 노드에 연결된 간선은 B-C, B-F, A-E인데 이 중 최소 가중치인 간선이 B-C이므로 C를 최소 신장 트리에 추가.



- 이제 확인할 간선은 A-E, B-F, C-G, C-D. 이 때 C-F가 빠진 이유는 사이클이 생기기 때문.
- 그 간선을 제외한 간선들 중에 제일 가중치가 작은 간선이 C-D 간선이므로 D를 추가.

이런 식으로 반복하다보면 아래 사진과 같은 최소 신장트리가 완성됨



- 최소 신장 트리에서 신장 트리는 그래프의 모든 정점을 연결하는 트리라는 의미.
- 최소 신장 트리라는 것은 여러 간선 중 가중치의 합이 최소가 되는 간선만 남긴 신장 트리를 의미.

이것의 쓸모:

[1] 최소한의 비용으로 모든 도시를 연결하는 도로를 건설할 방법을 찾을 때

[2] 새로 건설할 호텔의 배관을 최소 비용으로 구축할 때 등등

(3-2) 데이크스트라 알고리즘

프림 알고리즘과 비슷하지만 다른 예로 최단 경로를 "탐색"한다는 것에 대한 알고리즘.

이에 대한 차이:

[1] 프림 알고리즘은 각각의 간선의 가중치가 최소, 데이크스트라 알고리즘은 이동할때 지나는 간선들의 가중치 합이 최소

[2] 프림 알고리즘은 간선의 길이로 어떤 간선을 먼저 연결할지 결정. 데이크스트라 알고리즘은 경로의 길이로 간선을 연결.

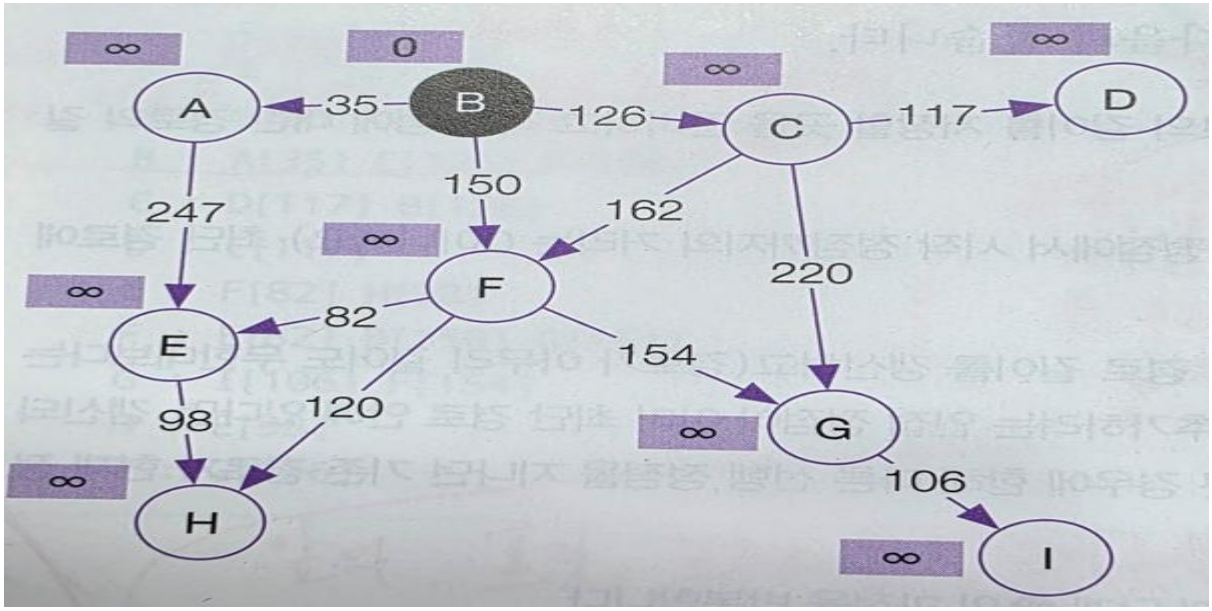
동작 방법:

[1] 각 정점에는 시작점으로부터 자신에게 이르는 경로의 길이를 저장. 각 정점에 대한 경로의 길이를 큰 수(오버플로우 되기 전의 가장 큰 값)로 초기화

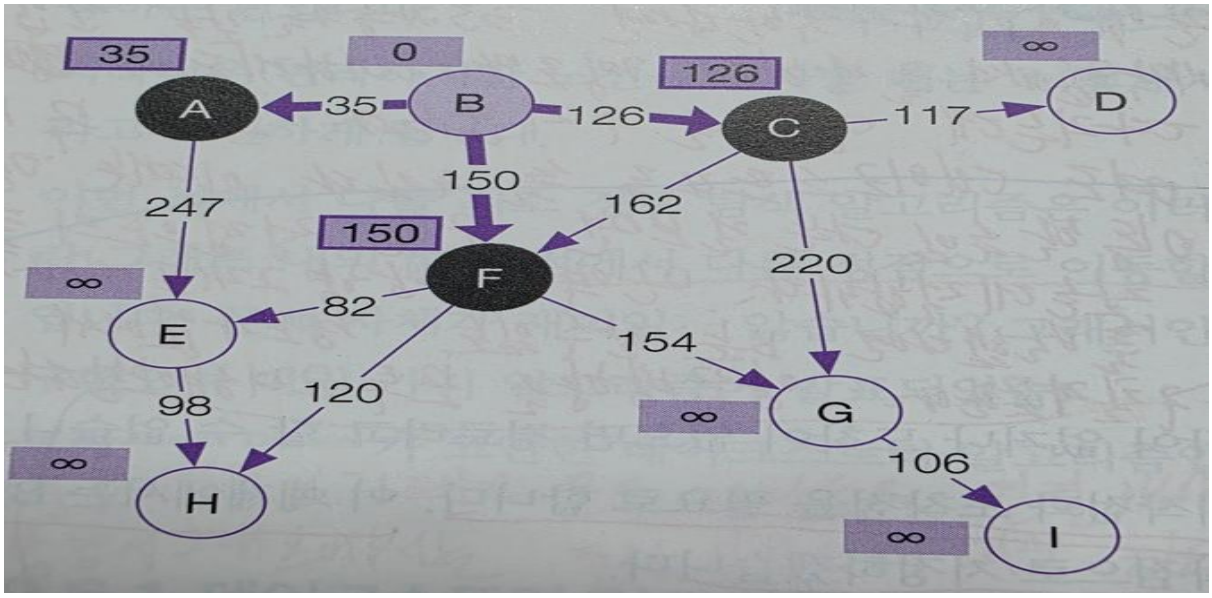
[2] 시작 정점의 경로 길이를 0으로 초기화하고 최단 경로에 추가

[3] 최단 경로에 새로 추가된 정점의 인접 정점에 대해 경로 길이를 갱신하고 이들을 최단 경로에 추가. 만약 갱신되기 이전의 경로 길이가 새로운 경로 길이보다 더 큰 경우에 한해 길이 갱신 및 경로 수정.

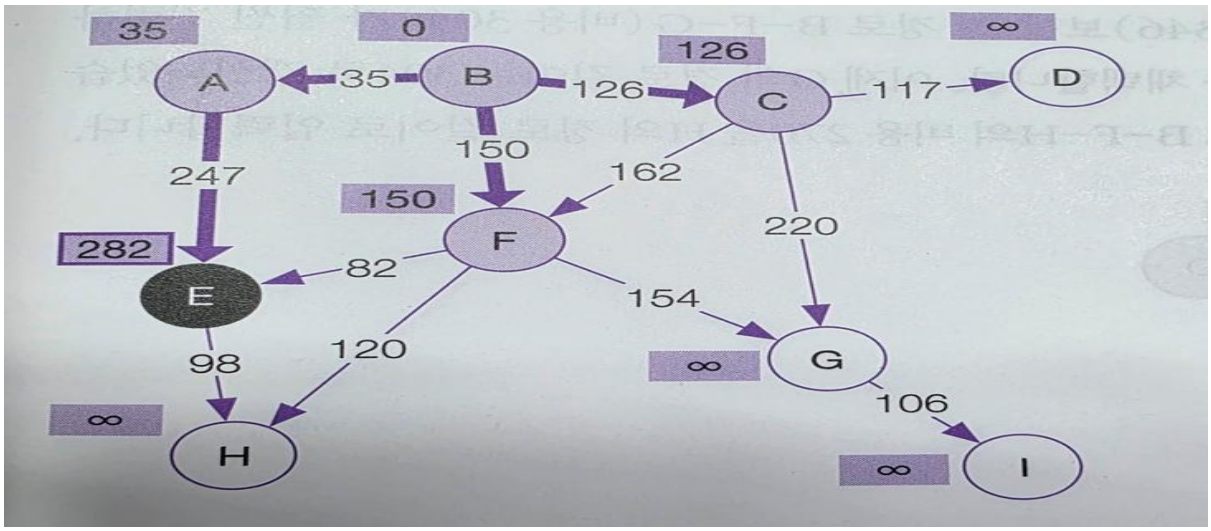
[4] 그래프 내의 모든 정점이 최단 경로에 소속될 때까지 [3]을 반복.



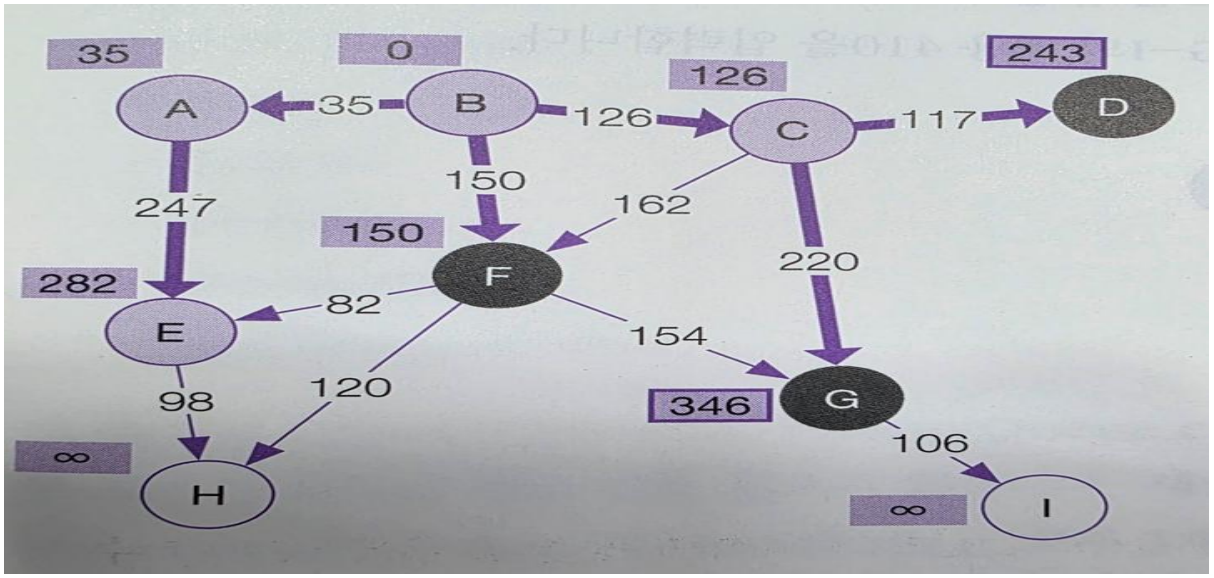
- 시작 정점인 B에 인접한 정점들을 찾고 간선의 가중치를 조사.
- 현재 조사된 간선들의 가중치 값이 큰 수보다 작으므로 A, C, F 정점들과의 거리를 가중치값으로 수정. (이 값들은 각 노드에 대응되는 배열 요소에 저장됨. 각 정점에는 인덱스 정보가 저장됨.)



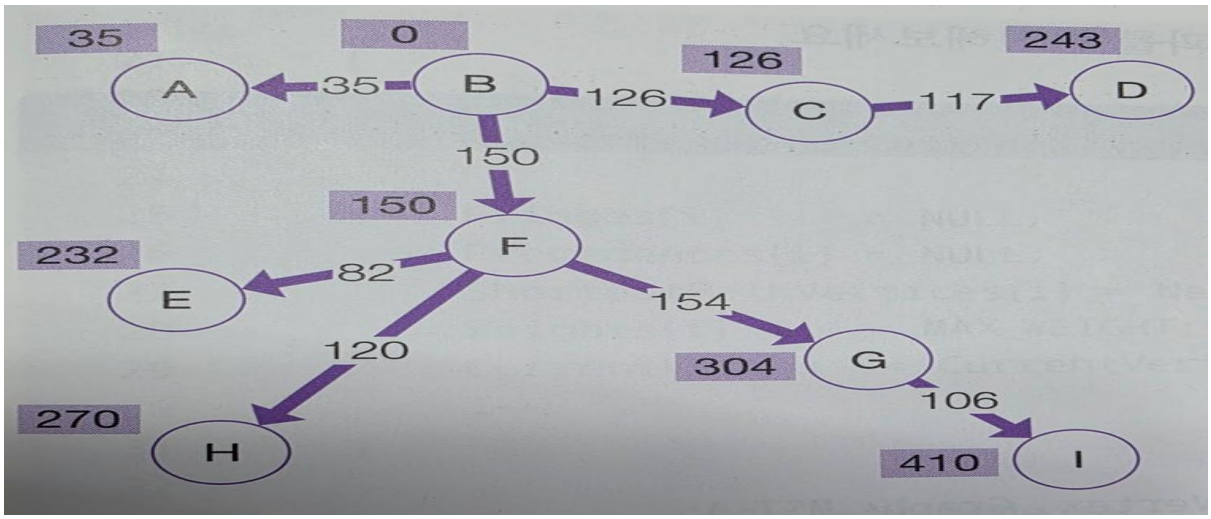
- A와 인접한 노드 -> E 노드
- E까지의 거리값을 갱신 = A에 저장된 값 + A-E간선 길이 값 (인접된 노드를 보는 순서는 각 정점이 가리키는 연결리스트의 노드 순서에 따름)



- C의 인접 정점 = D, G, F 노드
- D, G는 큰 수보다 작음 -> C의 126이라는 값 + 해당 간선 가중치
- F에서는 126+162인 288이 150보다 작아 채택 X



- F 인접 정점 = E, G, H
- E = 282라는 값, B-F-E경로의 합은 232, B-A-E 경로 폐기, B-F-E를 채택.
- G 경우에도 B-F-G 가중치 합이 작아 B-F-G경로로 채택.
- H의 값 갱신.



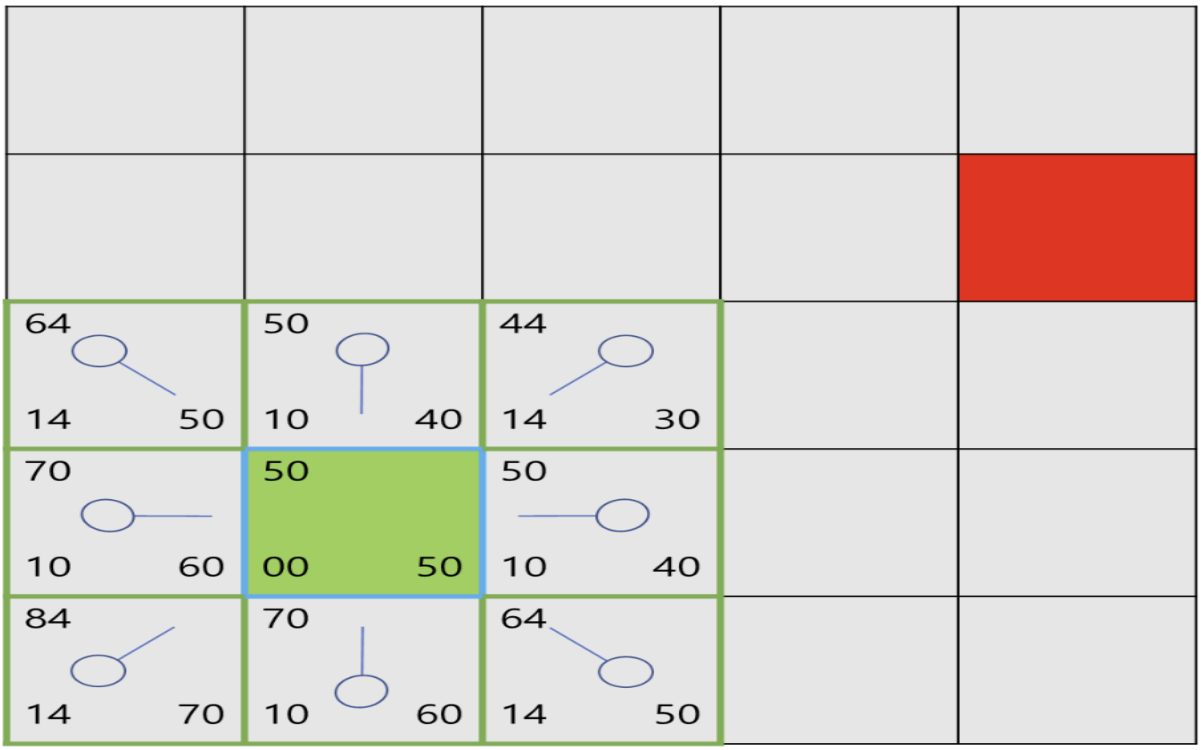
위의 방식대로 진행하면 왼쪽 사진과 같은 경로가 만들어짐.

(3-3) A* 알고리즘

- A* 알고리즘에서는 데이크스트라 알고리즘을 변형
- 시작 지점과 목표 지점을 명확히 중점을 두어 만듦.
- 휴리스틱으로 이를 가능하게 함.

휴리스틱이란: 현재 지점~ 목표 지점까지의 예상 거리

- 이 휴리스틱으로 계산된 거리 + 시작 지점~ 현재 지점까지의 소요된 비용 = 이동 거리의 값이 최소가 됨.
- 이런 상태의 노드를 다음 노드로 잡아 이동함



동작 방식:

[1] 위의 사진에서 초록색으로 색칠된 부분이 시작점, 빨간색으로 색칠된 부분이 목표 지점. 시작 지점으로부터의 인접 노드 조사 시작.

[2] 시작 지점~ 인접 노드까지의 소요되는 비용 혹은 거리 + 인접 노드~ 목표 지점까지 걸리는 거리가 작은 것을 다음 노드로 지

정 (이 때 시작 지점~ 인접 노드까지의 소요되는 비용은 데이크스트라의 알고리즘의 원리와 같음)

[3] 인접 노드로 목표 지점이 나올 때까지 계속 [2]를 반복함

A*와 데이크스트라 알고리즘 간의 차이:

[1] 데이크스트라 알고리즘은 시작점으로부터 모든 목적지까지의 최단 경로를 모두 구함 => 사용자가 원하는 목표 지점까지의 최단 경로를 구하는 것에 적합 X.

[2] 데이크스트라 알고리즘은 목표 지점에 대한 사전 정보 없이 시작됨. 반면 A* 알고리즘은 목표 지점까지의 추정 비용을 계산하고 그 목표 지점으로 직관적으로 이동함. 휴리스틱 거리를 반영하여 거리가 제일 짧게 나오는 쪽으로 이동함.