The background features a dark blue gradient with faint, light blue concentric circles and degree markings (140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260) on the left side, suggesting a circular or rotational theme.

HEAP SORTING, HEAP STRUCTURE, PRIORITY QUEUE BINARY INDEX TREE DIJKSTRA ALGORITHM, SSSP에 대한 레포트

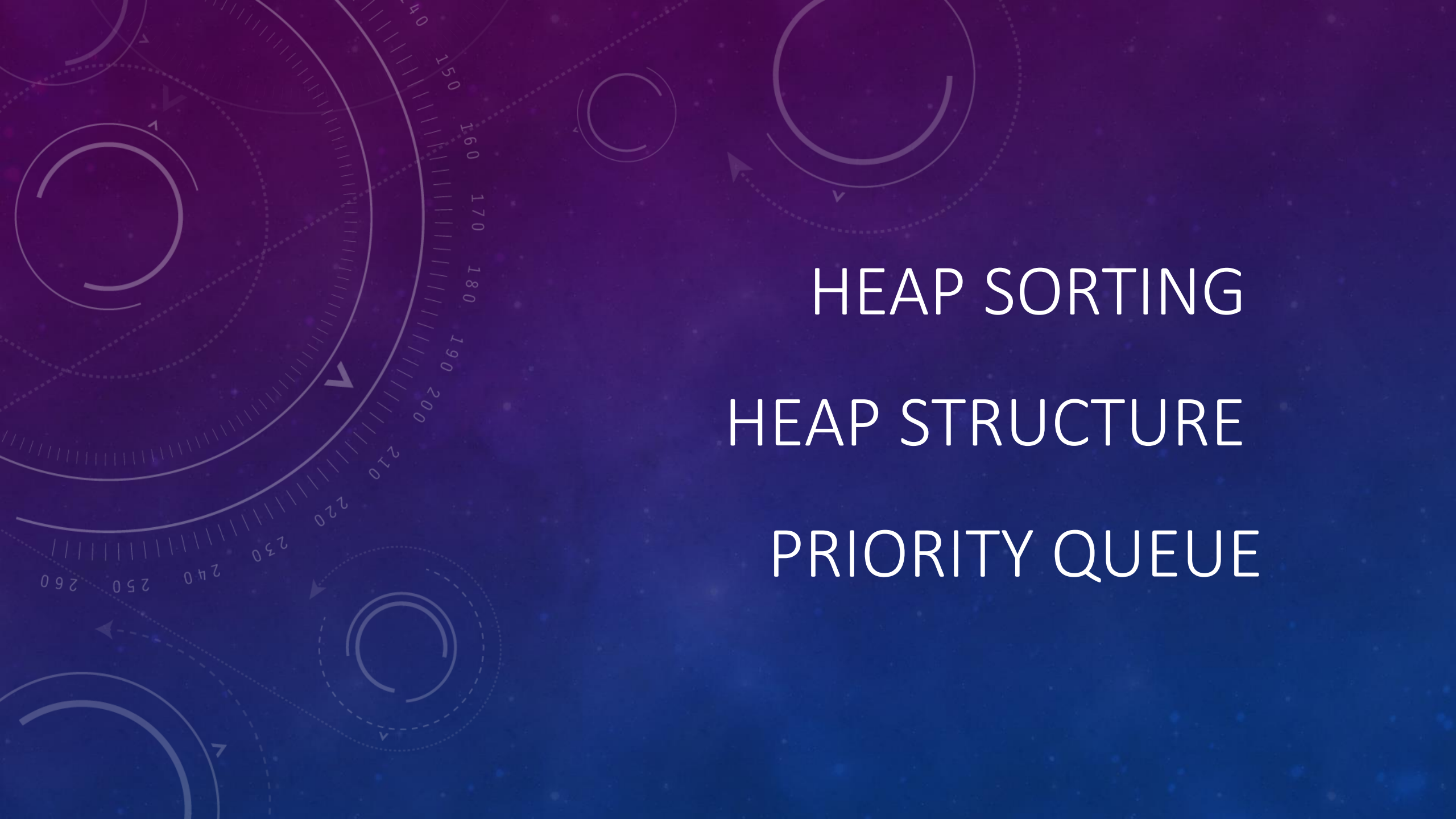
V2018113 권민섭

KMU9842@NAVER.COM

서강대학교 영상대학원

주제

- HEAP Sorting, Heap Structure, Priority Queue
- Binary Index Tree
- Dijkstra Algorithm, SSSP



HEAP SORTING

HEAP STRUCTURE

PRIORITY QUEUE

HEAP

- 힙 (Heap) - 무엇인가를 차곡차곡 쌓아올린 더미
- 여러 개의 값 중에서 가장 크거나 작은 값을 빠르게 찾기 위해 만든 이진 트리이다.
- 큰 키(높은 우선순위)에 자주 액세스 하거나 키 중심으로 정렬된 시퀀스를 써야 할 때 유용함
- 보통 완전이진트리를 기본으로 한다.

HEAP의 데이터 삽입

1. 가장 끝의 자리에 노드를 삽입한다.
2. 그 노드와 부모 노드를 서로 비교한다.
3. 규칙에 맞으면 그대로 두고, 그렇지 않으면 부모와 교환한다.
4. 규칙에 맞을 때까지 3번 과정을 반복한다.



HEAP의 데이터 삭제

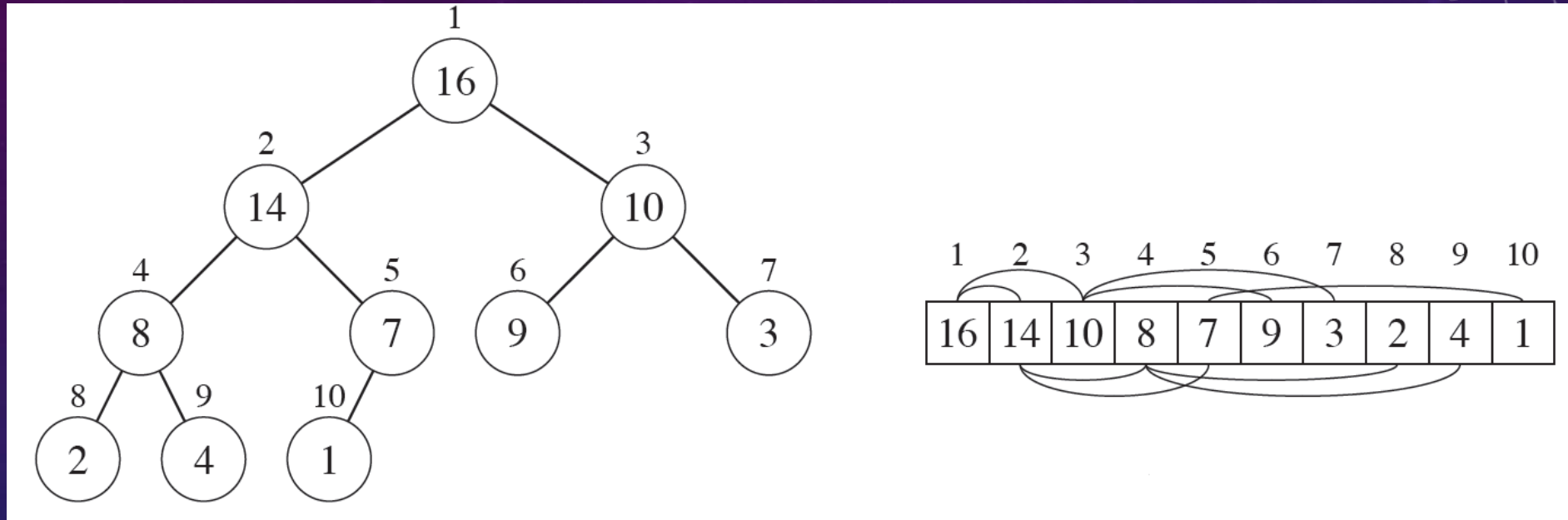
삭제는 루트노드만 제거 할 수 있다.

1. 루트 노드를 제거한다.
 2. 루트 자리에 가장 마지막 노드를 삽입한다.
 3. 올라간 노드와 그의 자식 노드(들)와 비교한다.
 4. 조건에 만족하면 그대로 두고, 그렇지 않으면 자식과 교환한다.
- 최대 힙
 1. 부모보다 더 큰 자식이 없으면 교환하지 않고 끝낸다.
 2. 부모보다 더 큰 자식이 하나만 있으면 그 자식하고 교환하면 된다.
 3. 부모보다 더 큰 자식이 둘 있으면 자식들 중 큰 값과 교환한다.
 - 최소 힙
 1. 부모보다 더 작은 자식이 없으면 교환하지 않고 끝낸다.
 2. 부모보다 더 작은 자식이 하나만 있으면 그 자식과 교환한다.
 3. 부모보다 더 작은 자식이 둘 있으면 자식들 중 작은 값과 교환한다.

이후 조건을 만족할때까지 4의 과정을 반복한다.



HEAP의 구현



힙은 완전이진트리의 성질을 만족하므로, 각 노드에 순서대로 다음과 같이 번호를 지정하여 배열로 구현할 수 있다.

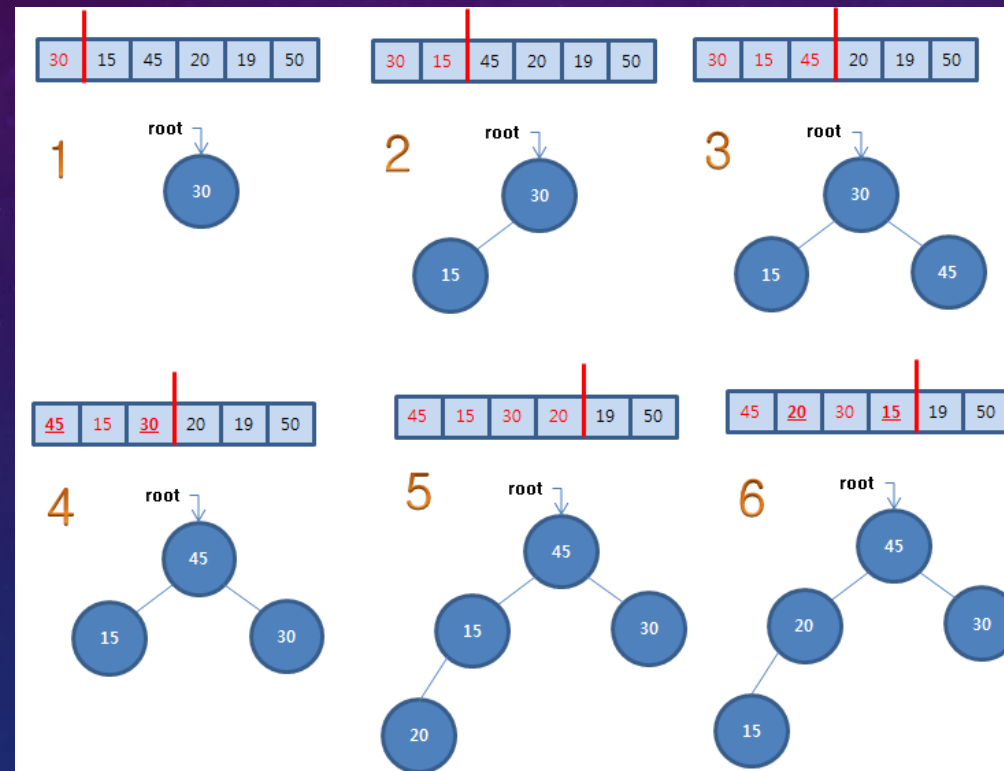
HEAP SORT

- 힙을 이용한 정렬
- 힙정렬은 추가적인 메모리를 요구하지 않는다.
- 최악의 경우를 피하려면 피벗을 잘 잡는 추가 규칙이 필요한 퀵정렬과는 달리, 알고리즘 자체만으로도 가장 안정적인 성능을 낸다.

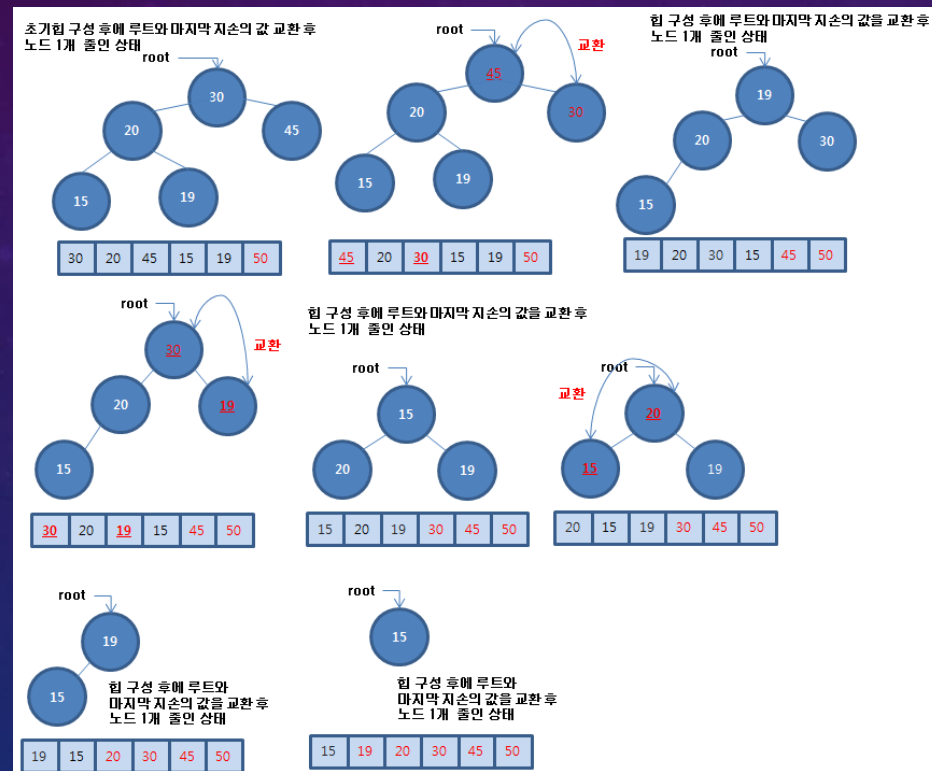
HEAP SORT의 수행

1. 주어진 원소들로 최대 힙을 구성합니다.
2. 최대 힙의 루트노드와 말단노드를 교환한다.
3. 새 루트 노드에 대해 최대 힙을 구성해준다.
4. 원소의 개수만큼 2와 3을 반복 수행한다.

HEAP SORT의 수행



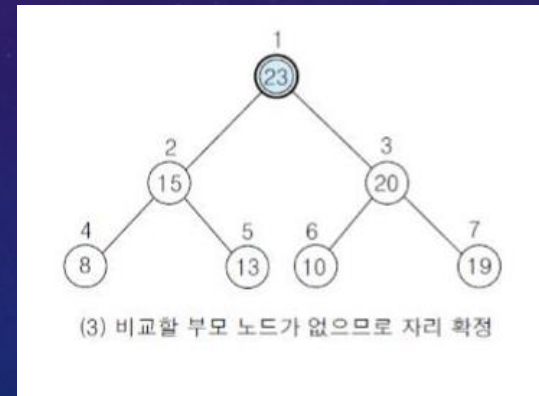
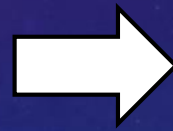
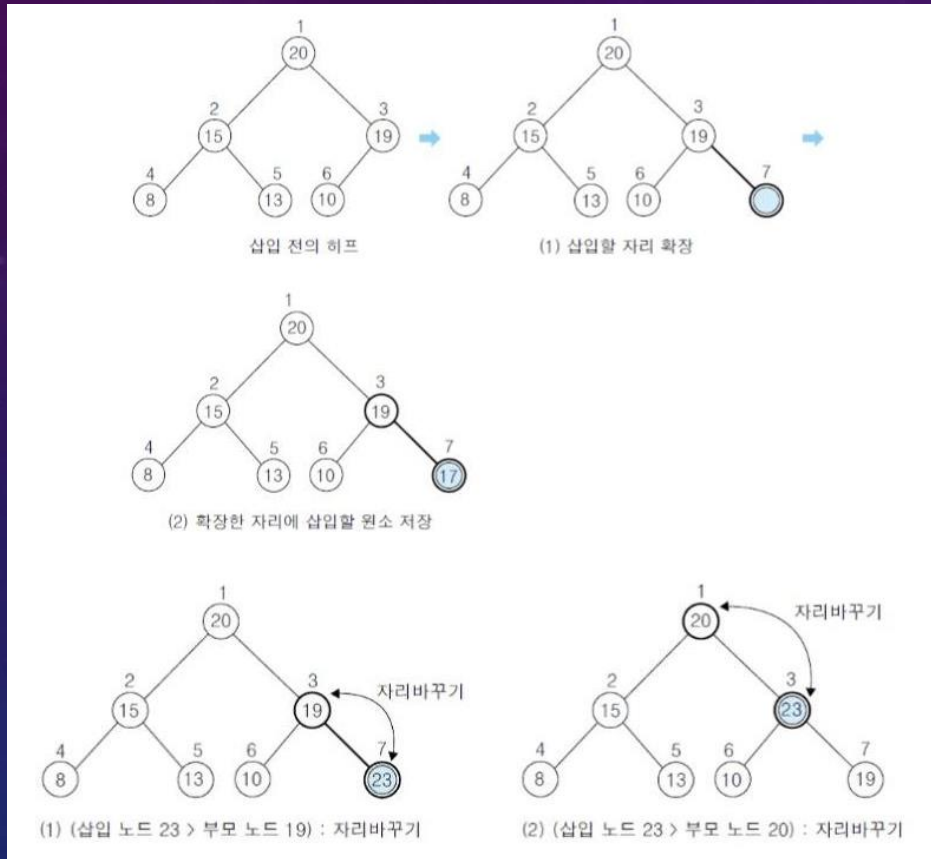
HEAP SORT의 수행



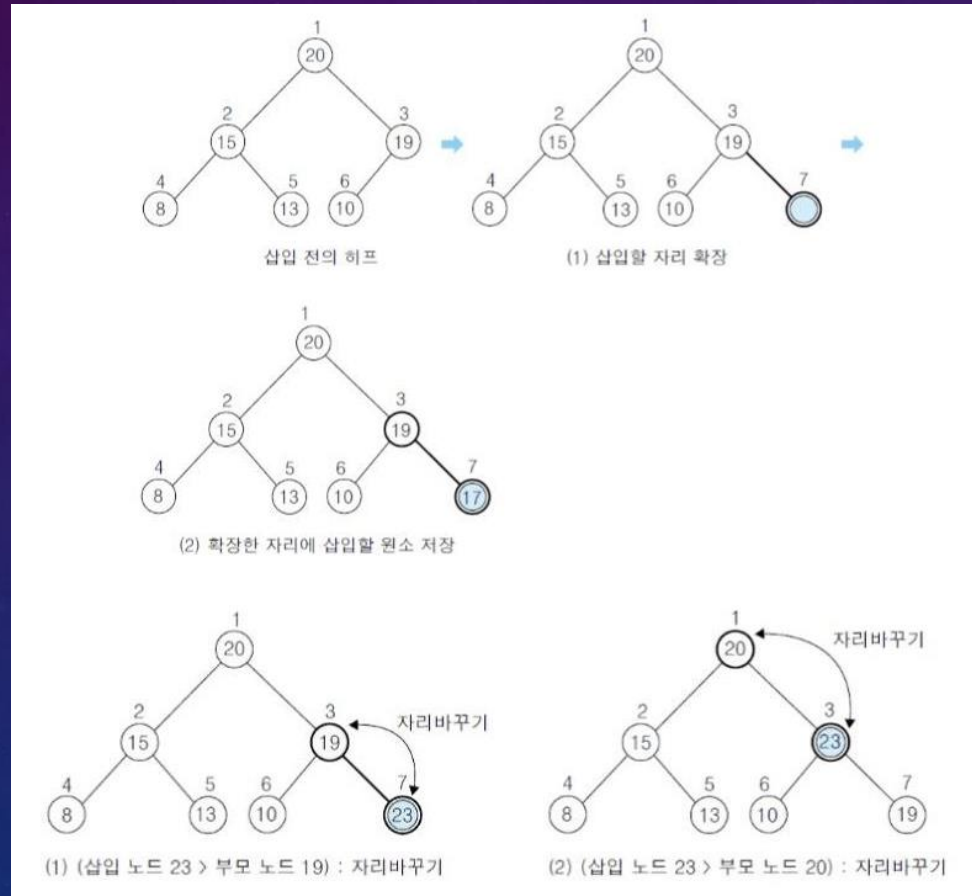
PRIORITY QUEUE

- 우선순위 큐
- 큐에 우선순위를 부여하여, 넣을때의 순서와 상관없이 우선순위가 높은 원소부터 빼낸다.
- 기존의 큐처럼 순서대로 나오는것이 아닌 우선순위(최대값 또는 최소값)을 사용해 빼내기 때문에 힙 구조를 사용한다.
- 기존의 힙구조와 동일하게 구현하면서 우선순위를 추가해 만든다.

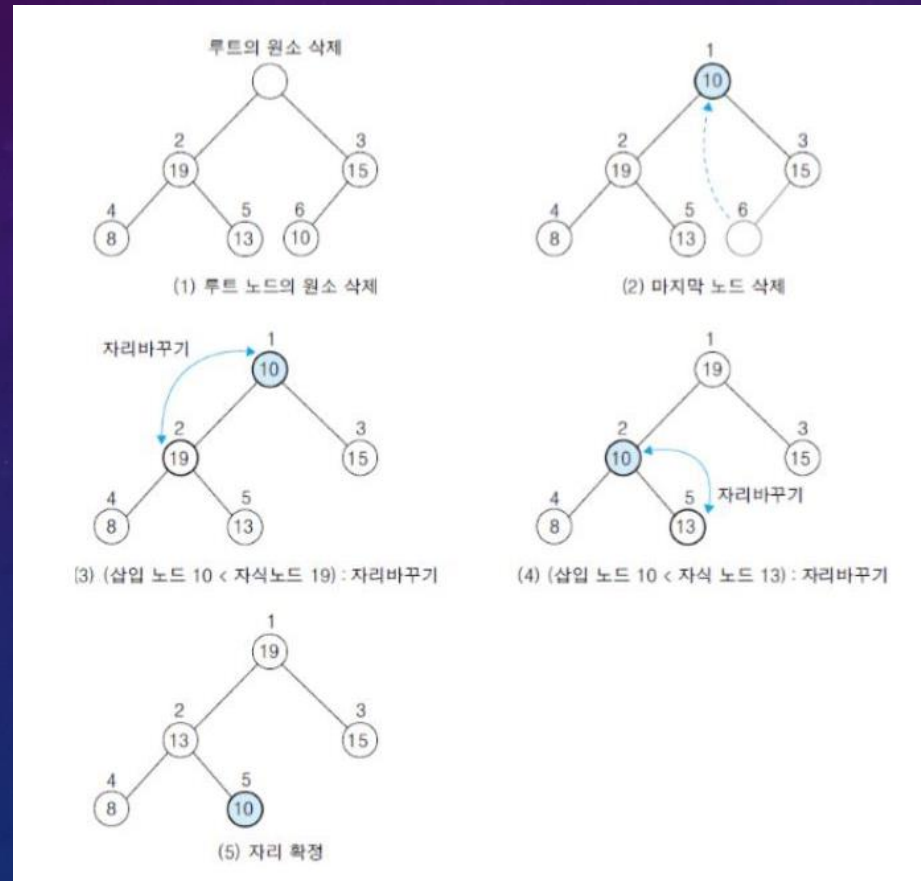
PRIORITY QUEUE의 수행



PRIORITY QUEUE의 수행



PRIORITY QUEUE의 수행



The background is a dark blue gradient with a subtle pattern of white dots. On the left side, there are several concentric circles and a large arc with a scale. The scale has numbers ranging from 140 to 260 in increments of 10. There are also some curved arrows and dashed lines, giving it a technical or scientific feel.

BINARY INDEX TREE (FENWICK TREE)

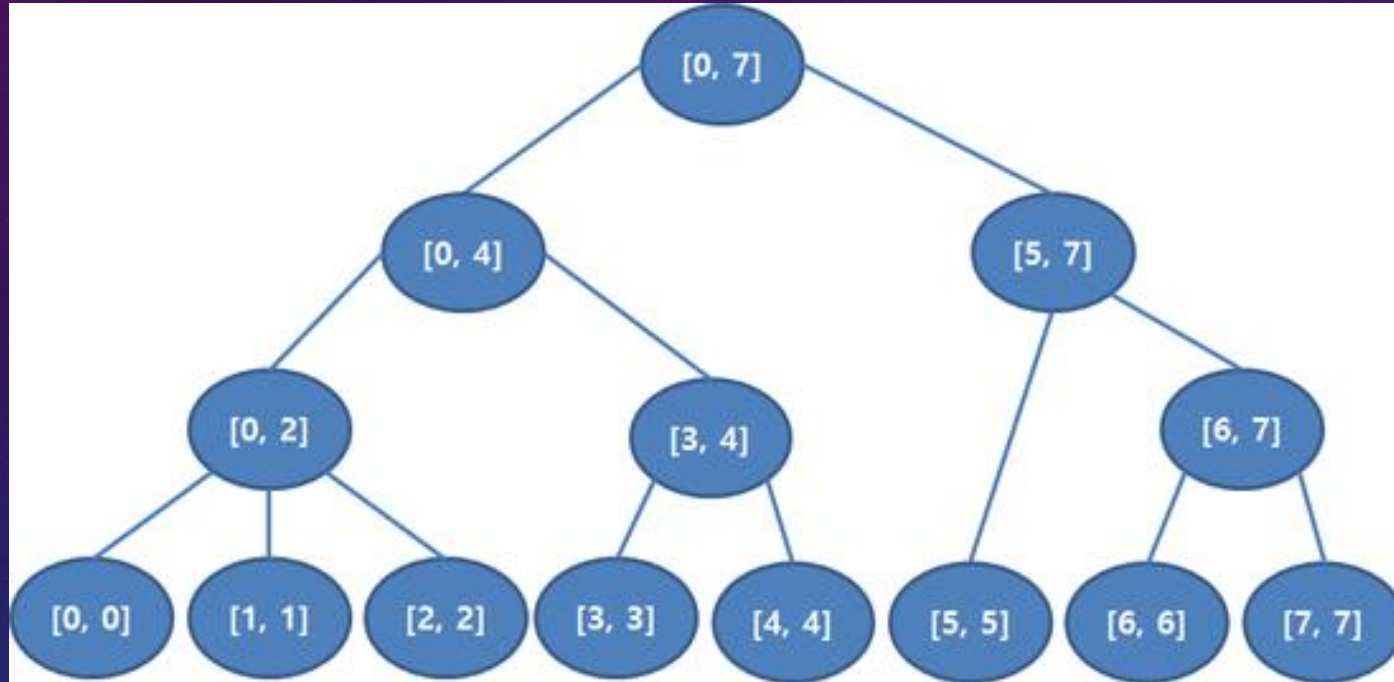
BINARY INDEX TREE

- Binary - 이진, Indexed - 인덱스가 매겨진, Tree - 트리
- 인덱스가 매겨진 이진트리. 펜윅 트리라고도 한다.
- 세그먼트 트리의 메모리를 절약하기 위해 만들어진 자료구조이다.
- 정수는 거듭제곱들의 합으로 표현되며, 누적빈도수는 하위 빈도수들의 집합들의 합으로 표현된다.
- 구현이 쉬운편

SEGMENT TREE?

- 배열의 부분 합을 구할 때 배열이 계속해서 바뀔 수 있다고 할 경우, 부분 합을 트리구조에 저장함으로서 $O(\log N)$ 의 속도로 배열의 부분 합을 빠르게 구할 수 있게 하는 배열.

SEGMENT TREE?



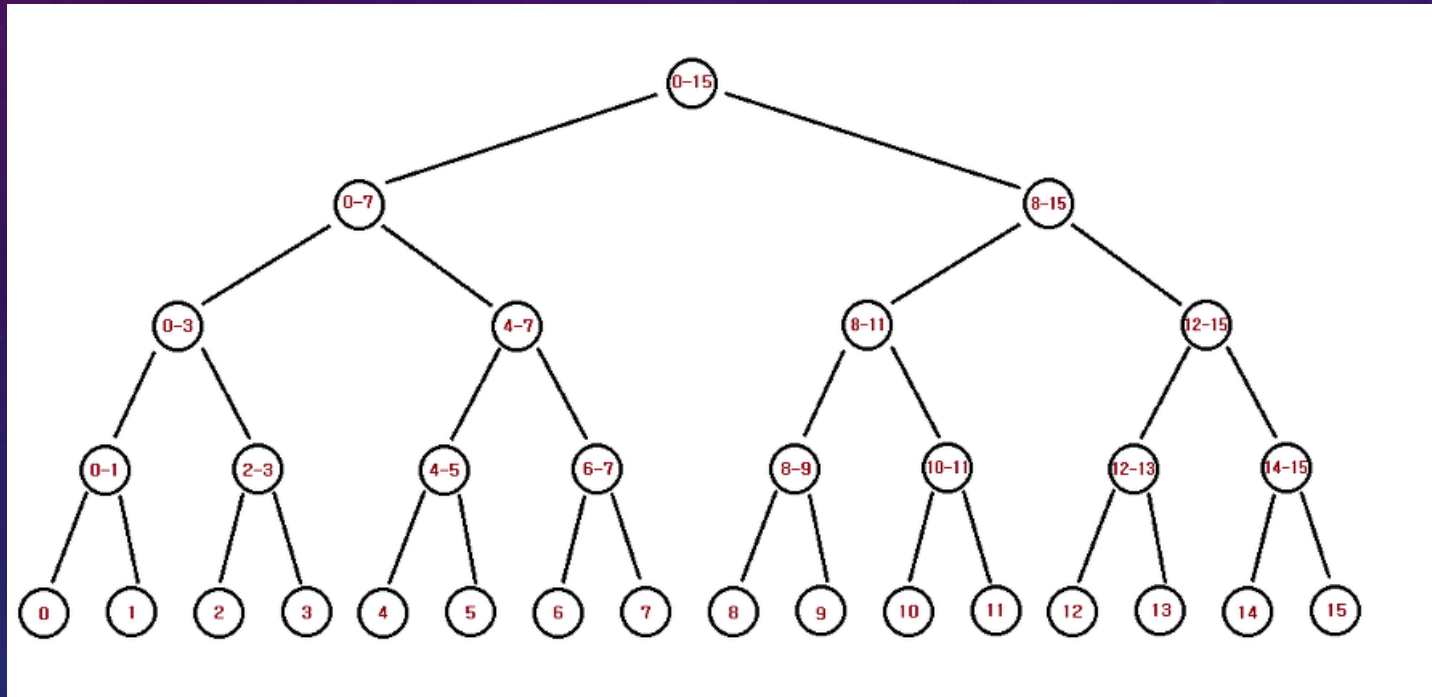
텍스트를 입전체 구간이 $[0, N-1]$ 이라 할 때, 즉 N 개의 공간이 있을 때,

리프 노드들은 길이가 1인 각각의 구간을 갖고 있고,
부모 노드는 자신의 자식들의 구간의 합을 갖고 있으며 모든 구간은 연속적이어야만 한다.
이렇게 각 노드가 구간, 혹은 그 구간의 정보를 저장하고 있는 자료구조를 말한다.

BINARY INDEX TREE와 SEGMENT TREE의 차이

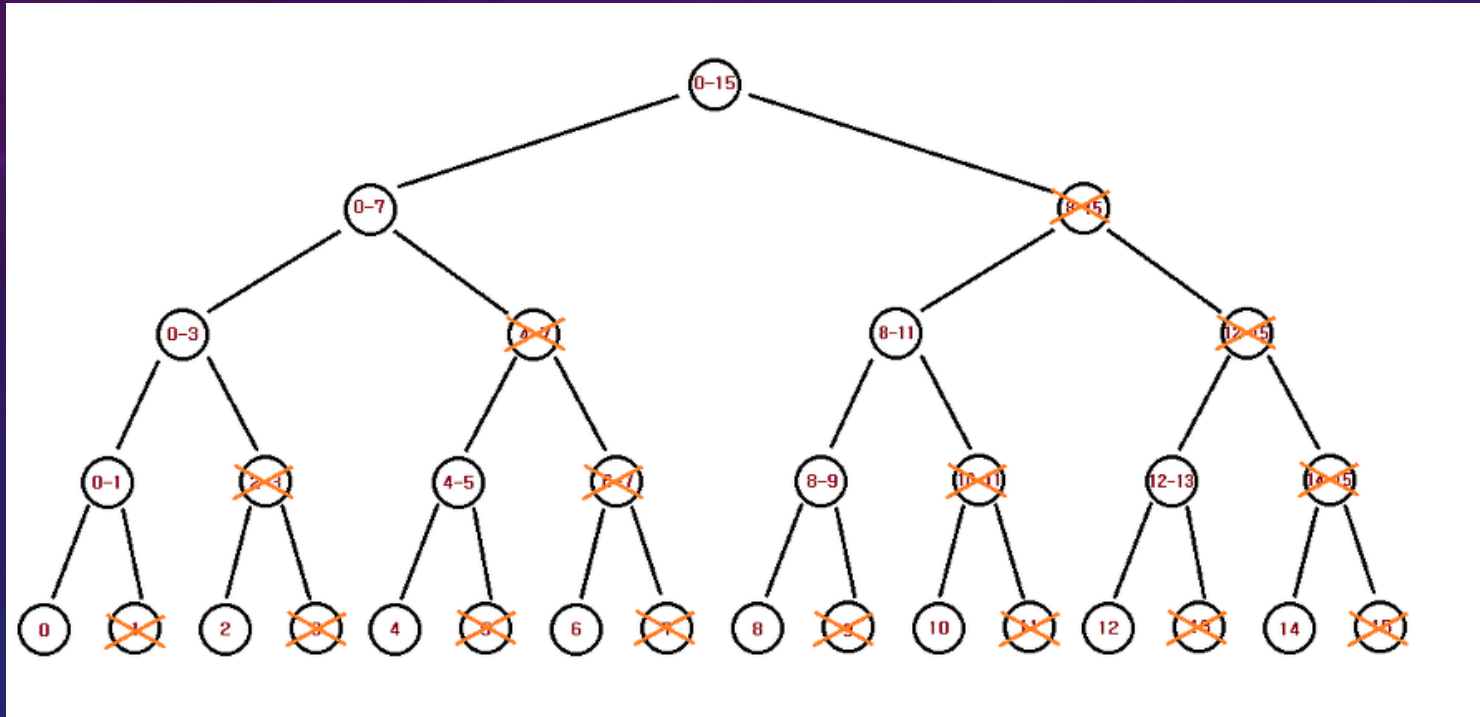
- 세그먼트 트리는 구간의 누적 합 뿐만 아니라, 특정 구간의 최대 값과 최소 값을 구할 수 있다.
- 하지만, 펜윅 트리는 구간의 누적 합은 구할 수 있지만, 특정 구간의 최대 값과 최소 값을 구하기에 제한적이다.

BINARY INDEX TREE와 SEGMENT TREE



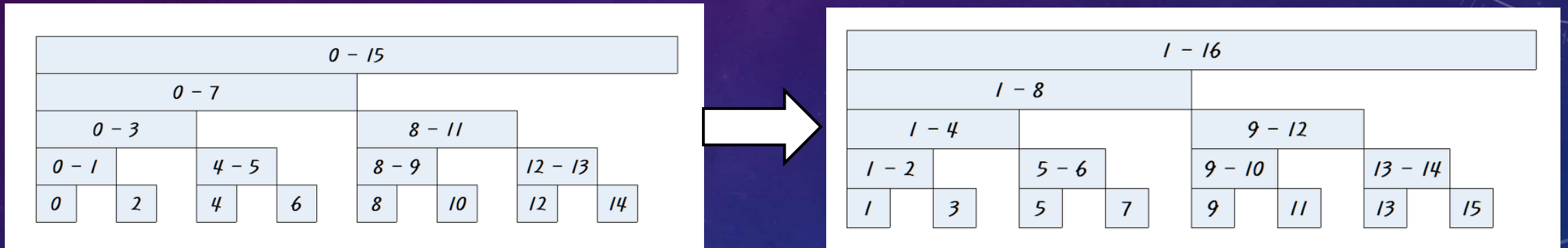
다음과 같은 새그먼트 트리가 있을 경우,
팬웍 트리로 바꾸면

BINARY INDEX TREE와 SEGMENT TREE

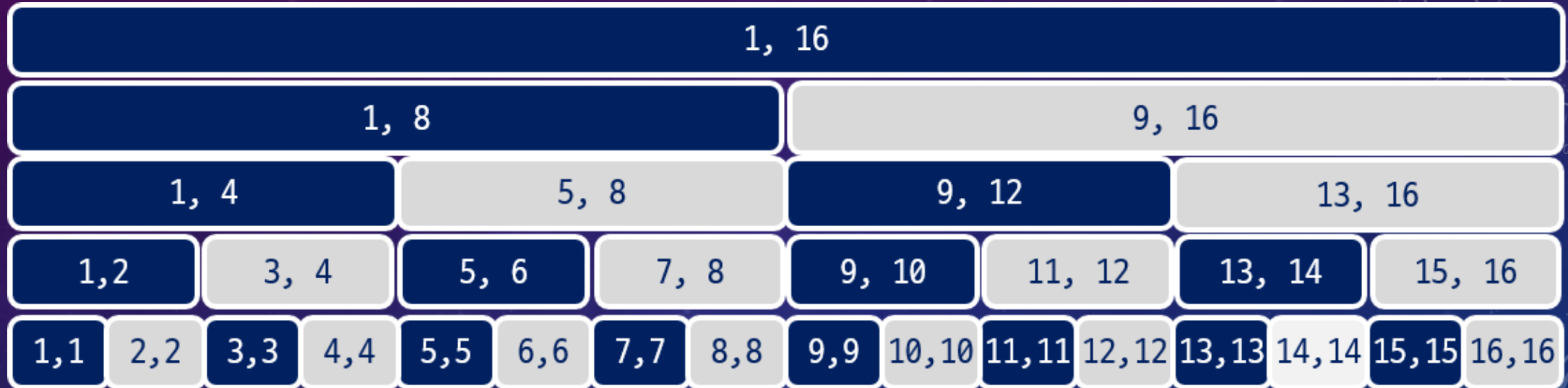


다음과 같이 변경된다.

BINARY INDEX TREE와 SEGMENT TREE

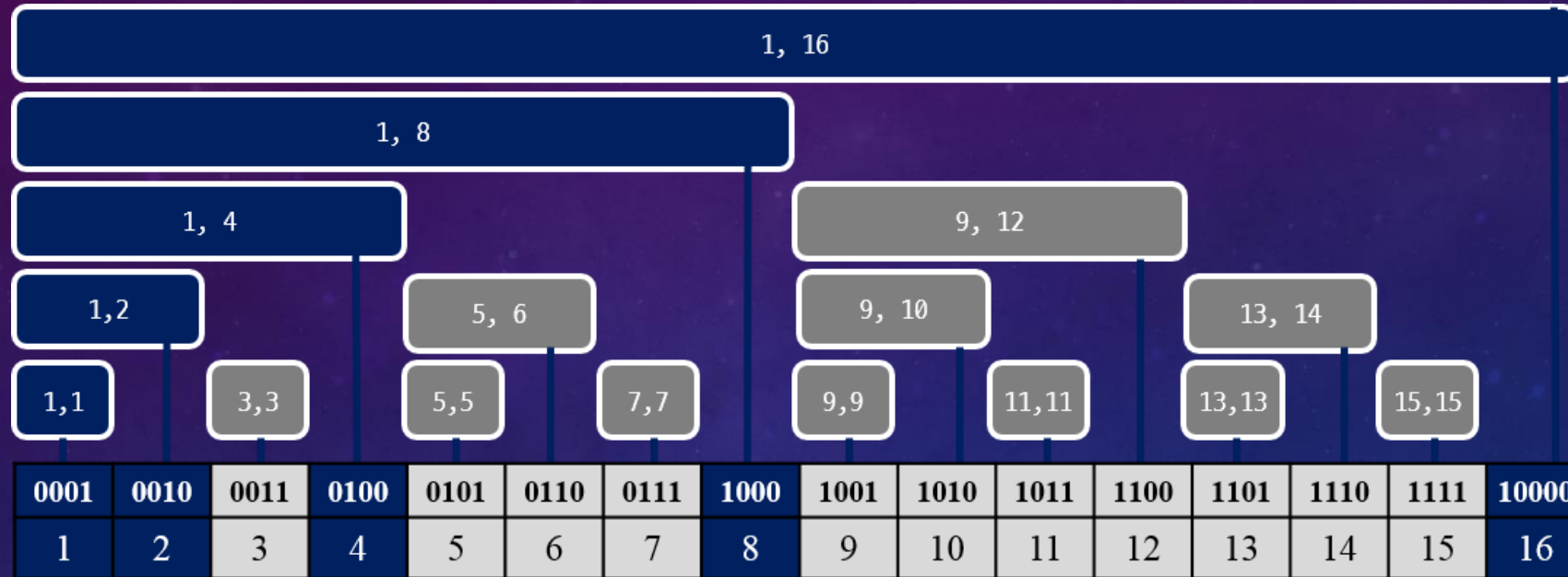


BINARY INDEX TREE의 표현



다음과 같은 세그먼트 트리가 있을 경우,
펜윅트리로 바꾸면서 다음과 같이 색칠되어 있는 부분만 저장한다.

BINARY INDEX TREE의 표현



해당 그림은 [1, 16]의 하위구간들을 1~16까지의 값으로 대응시켜서 보여준다.
각 구간에 해당하는 정보를 배열에 저장하기 때문에, 데이터의 개수만큼 공간이 필요하다.

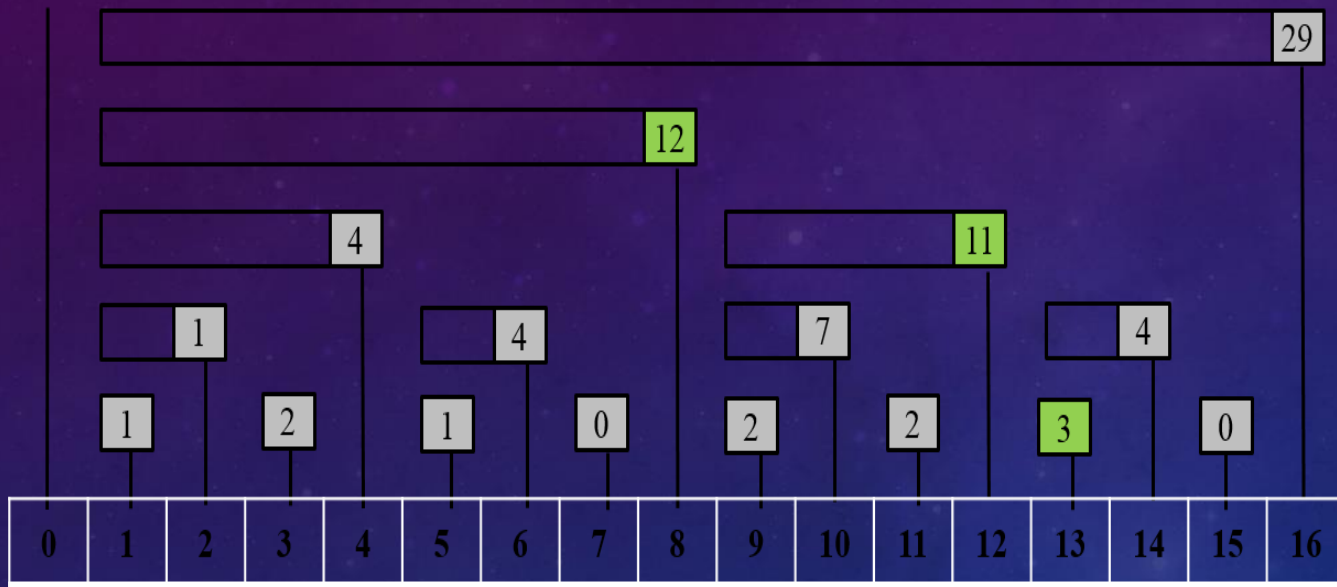
BINARY INDEX TREE의 기본 연산

- BIT의 기본 연산에는 다음과 같이 있다.
 - $\text{Update}(p, v)$: p 의 인덱스 위치값에 v 를 더한다.
 - $\text{Query}(l, r)$: $[l, r]$ 구간에 대한 질의

BINARY INDEX TREE의 구간 질의

- BIT는 누적된 구간의 합을 저장하는 형태이다.
- 따라서, $[l, r]$ 구간은 $[1, r]$ 구간합에서 $[1, l-1]$ 까지의 구간합을 빼서 구할 수 있다.

BINARY INDEX TREE의 구간 질의

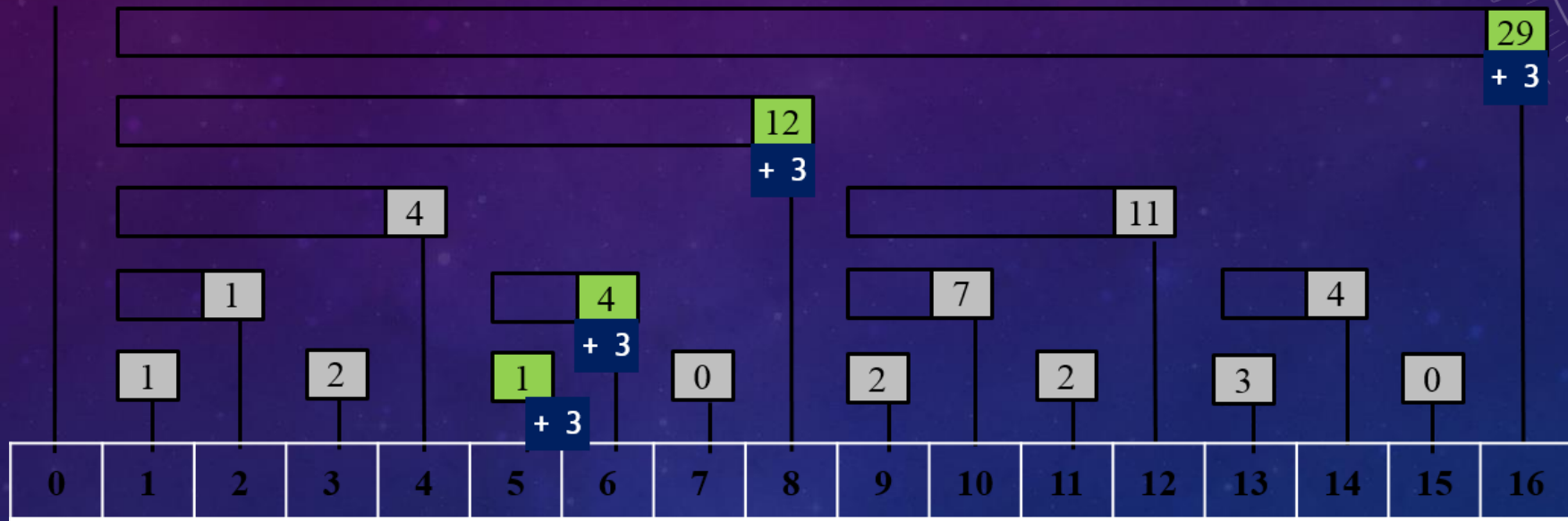


1. 13은 2진수로 1101 이다.
2. 13(1101)번 값 가져오기
3. $13(1001) - (0001) \Rightarrow 12(1100)$
4. 12(1100)번 값 가져오기
5. $12(1100) - (0100) \Rightarrow 8(1000)$
6. 8(1000)번 값 가져오기
7. $8(1000) - (1000) \Rightarrow 0$, 중단
8. 13, 12, 8에 해당하는 값들을 모두 더하면 26 이 된다.

BINARY INDEX TREE의 단일값 갱신

- 특정 위치의 인덱스 값에 포함된 최하위 비트를 더하고, 다시 변경된 인덱스 값의 최하위 비트를 더하면서 변경해 나가면 특정위치의 자료값을 변경할 수 있으며, 수정되어야 할 곳 들을 찾을 수 있다.

BINARY INDEX TREE의 단일값 갱신



- 다음은 $\text{update}(5, 3)$, 즉 5번 값에 3을 더하는 경우 BIT의 변경되는 구간을 보여준다.



DIJKSTRA ALGORITHM, SSST

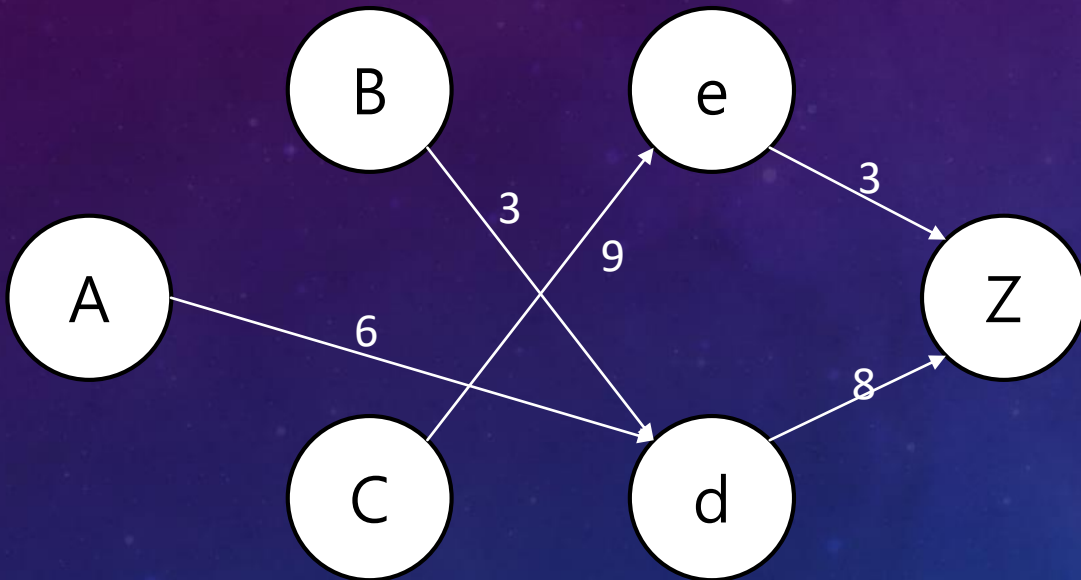
DIJKSTRA ALGORITHM

- 다익스트라 알고리즘은, 그래프에서 노드 간의 최단 경로를 찾는 알고리즘이다.

DIJKSTRA ALGORITHM

- 예) Jungol 귀가(1208번 문제)
http://jungol.co.kr/bbs/board.php?bo_table=pbank&wr_id=491&sca=3070
- 경로는 5개, 출발점은 A, B, C중 하나가 되며, 목적지는 z일 경우,

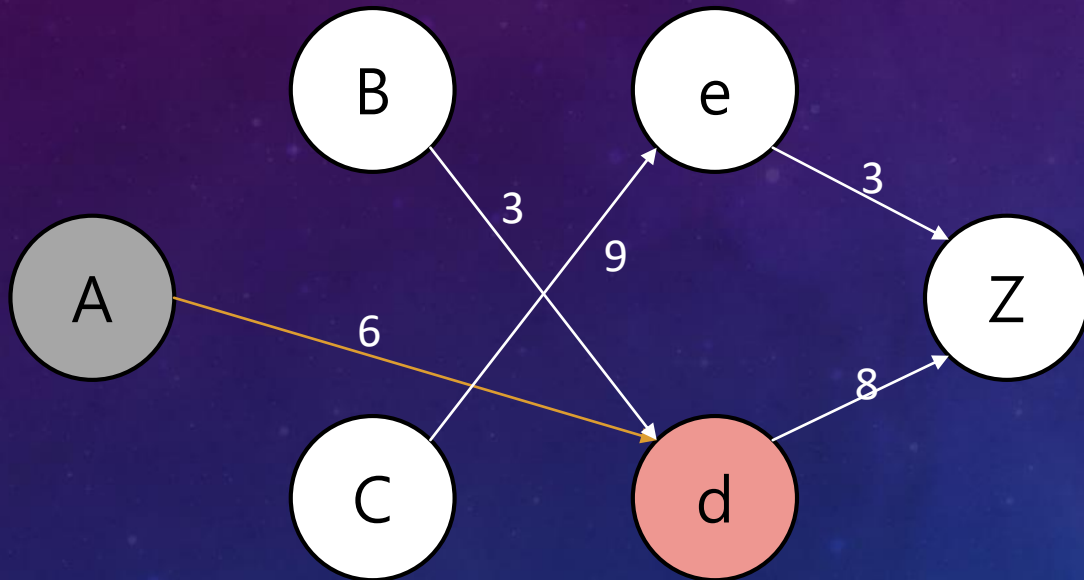
DIJKSTRA ALGORITHM



- $S = \{ \}$
 - $d[A] = 0$
 - $d[B] = -$
 - $d[C] = -$
 - $d[d] = -$
 - $d[e] = -$
 - $d[Z] = -$
- $Q = \{A, B, C, d, e, Z\}$

1. 모든 경로에 대하여 초기화 실행한다.
s는 방문한 정점, Q는 방문하지 않은 정점,
 $d[N]$ 는 출발점 부터 N까지의 거리를 나타낸다.

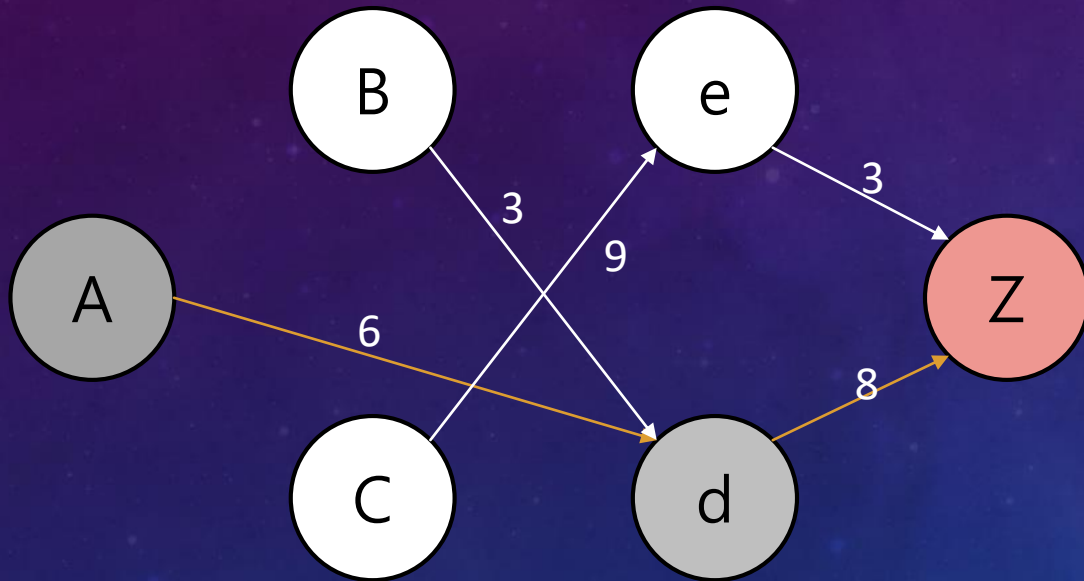
DIJKSTRA ALGORITHM



- $S = \{ A, d \}$
 - $d[A] = 0$
 - $d[B] = -$
 - $d[C] = -$
 - $d[d] = 6$
 - $d[e] = -$
 - $d[Z] = -$
- $Q = \{ A, B, C, d, e, Z \}$

2. 루프를 돌려 갈 수 있는 노드를 방문하고
거리를 계산한다.

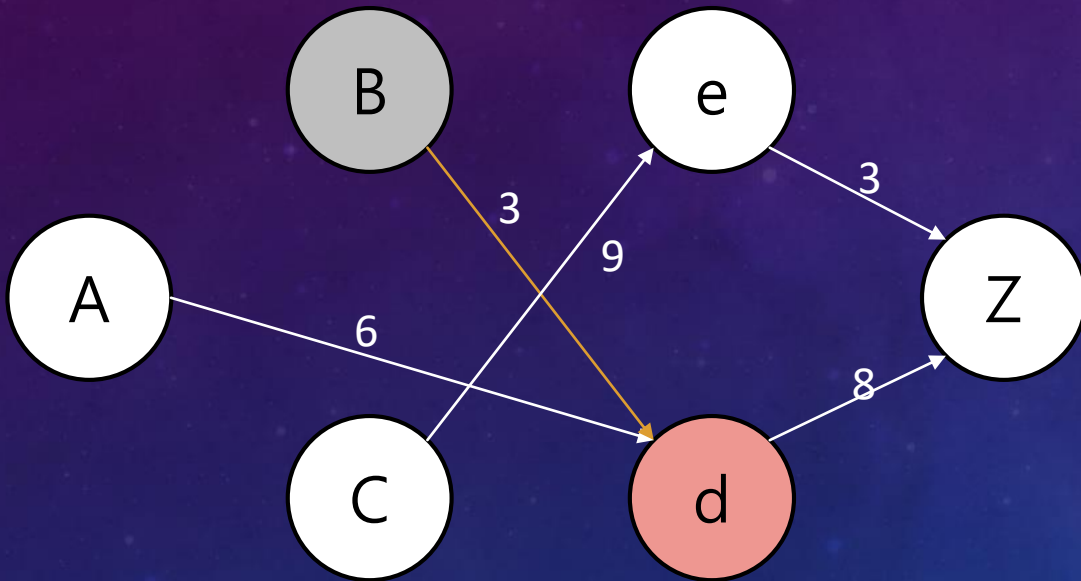
DIJKSTRA ALGORITHM



- $S = \{ A, d, Z \}$
 - $d[A] = 0$
 - $d[B] = -$
 - $d[C] = -$
 - $d[d] = 6$
 - $d[e] = -$
 - $d[Z] = 14$
- $Q = \{ A, B, C, d, e, Z \}$
- $A \rightarrow Z = 14$

3. 목적지 z에 도달했을 경우, 거리를 저장한다.
A → Z 까지의 거리는 14이다. 중복되는 경로가 있을 경우,
가장 작은 경로가 있으면 그것을 저장한다.

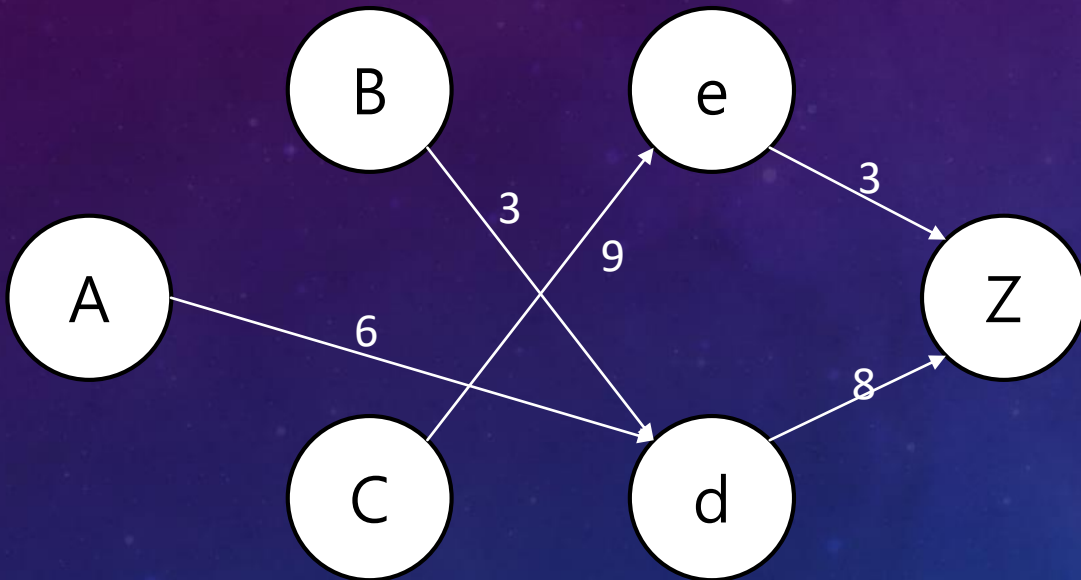
DIJKSTRA ALGORITHM



- $S = \{ B, d \}$
 - $d[A] = -$
 - $d[B] = 0$
 - $d[C] = -$
 - $d[d] = 3$
 - $d[e] = -$
 - $d[Z] = -$
- $Q = \{ A, B, C, e, Z \}$

4. 다른 출발점에서 위의 루프를 다시 실행한다.

DIJKSTRA ALGORITHM



- $A \rightarrow Z = 14$
- $B \rightarrow Z = 11$
- $C \rightarrow Z = 12$

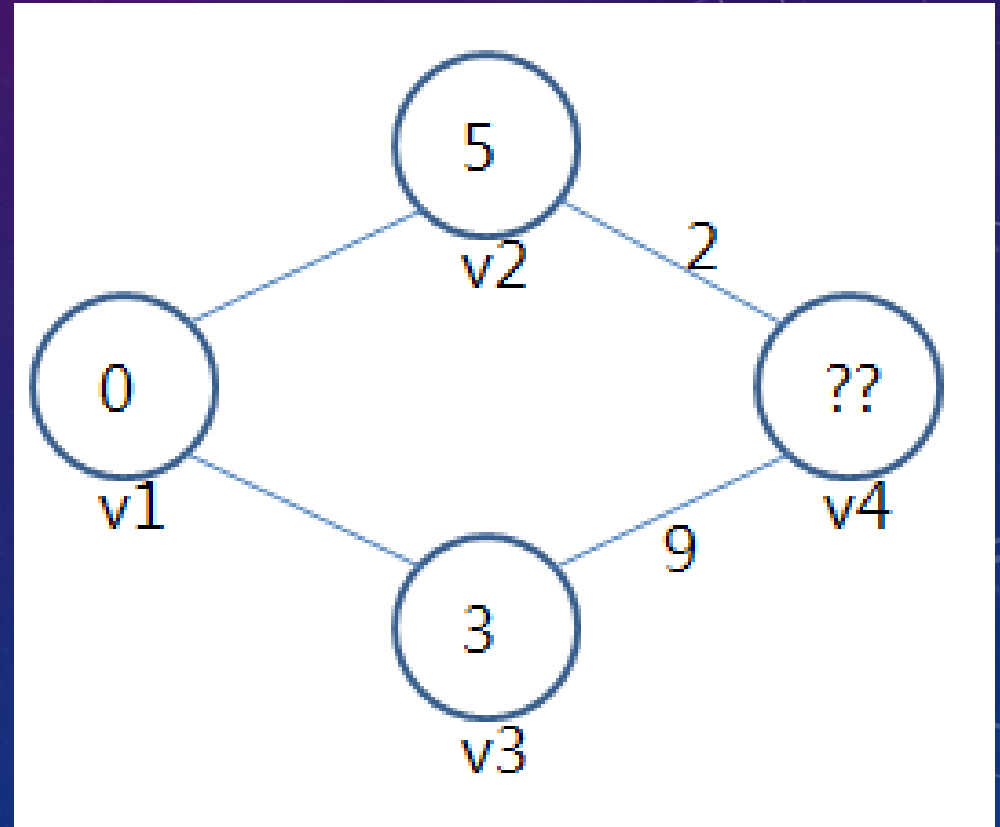
5. 출발점(A, B, C)의 루프를 모두 돌았을 경우, 각 출발점에서 z까지의 거리를 비교하여 가장 짧은 거리가 정답이 된다.

SSSP, SINGLE SOURCE SHORTEST PATH

- 단일 출발지 최다 경로
- 단일 출발지 최단경로란 하나의 출발점에서 각 정점까지 도달하는데 비용을 계산하여 최단경로를 구하는 것이다.

SSSP, SINGLE SOURCE SHORTEST PATH

- 다음 그래프에서 v_1 에서 v_4 로 가는 경로는 2가지가 존재한다.
- 시작정점을 v_1 이라 할 경우, v_2, v_3 까지 도달하는데 걸리는 비용을 알고 있다고 가정하면 최단 경로는 두 경로 중 비용이 적은 경로를 선택하는 것
- $v_1 \rightarrow v_2 \rightarrow v_4$ 는 $5 + 2 = 7$ 의 비용이 든다.
- $v_1 \rightarrow v_3 \rightarrow v_4$ 는 $3 + 9 = 12$ 의 비용이 든다.
- 따라서 적은 비용을 소모하는 $v_1 \rightarrow v_2 \rightarrow v_4$ 경로를 선택하는 것이 최단 경로이다.



SSSP, SINGLE SOURCE SHORTEST PATH

- SSSP는 단일 출발지에서 최단경로를 구하는 것
- 1차원 배열로 각 정점 까지의 경로를 표현 할 수 있음
- SSSP에는 음의 가중치를 설정 할 수 있는데, 이중 음의 가중치를 설정하지 않고 찾는것이 다익스트라 알고리즘이다.
- 여기서, 최단경로의 추정 값을 구할때 Relaxation을 이용하게 된다.

RELAXATION

- Relaxation란, $d[v]$ 를 정점 v 까지의 최단 경로 추정 값, $s(s,v)$ 를 정점 s 에서 v 까지의 실제 최단 경로 비용이라고 할 때, $d[v]$ 의 값이 $s(s,v)$ 의 상한이 되도록 유지하는 것을 말한다.
- 목적지 정점(v_2)의 추정값과 직전 정점(v_1)의 추정 값 + 간선의 가중치를 비교하여 목적지 정점의 추정 값을 조정하는 것으로 만들 수 있다.



BIT부분이 잘 이해가 되질 않습니다 수업 때 다른 분들의 설명 도움을...

참조

- 힙 - <https://ratsgo.github.io/data%20structure&algorithm/2017/09/27/heapsort/>
- 힙 정렬 - <http://ehpub.co.kr/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98-c%EC%96%B8%EC%96%B4-3-5-1-%ED%9E%99-%EC%A0%95%EB%A0%AC-%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98-%EC%86%8C%EA%B0%9C/>
- 우선순위 큐 - <http://robodream.tistory.com/183>
- 펜윅 트리 - <http://kimbregas.tistory.com/20>
- 펜윅 트리 - <http://www.crocus.co.kr/666>
- 펜윅 트리 - <http://www.algocoding.net/advanced/fenwick.html#id3>
- 세그먼트 트리 - <https://kks227.blog.me/220791986409?Redirect=Log&from=postView>
- 다익스트라 알고리즘 - <http://trowind.tistory.com/80>
- SSSP - <http://victorydntmd.tistory.com/103>