열혈강의

자료구조

모범 답안

내용

1 장 3

2 장 5

3 장 7

4 장 9

5 장 15

6 장 17

7 장 19

8 장 23

9 장 30

10 장 35

1장

01_

1) 선형구조에 속하는 자료구조: 리스트, 스택, 큐, 덱

2) 비선형구조에 속하는 자료구조: 트리, 그래프

02_

1. 자료

   1) MP3 파일 이름 리스트

   2) 파일 이름 리스트에서 현재 연주되는 파일의 위치

2. 명령(연산)

이름	입력	출력	설명
Play()	N/A (없음)	성공 여부 (TRUE / FALSE)	음악을 연주한다
Stop()	N/A	성공 여부 (TRUE / FALSE)	음악 연주를 중지한다
Pause()	N/A	성공 여부 (TRUE / FALSE)	음악 연주를 일시 중지한다
Add_MP3()	파일 경로	성공 여부 (TRUE / FALSE)	MP3 파일을 목록에 추가한다
Delete_MP3()	삭제될 파일의 리스트에서의 위치	성공 여부 (TRUE / FALSE)	MP3 파일을 목록에서 제거한다
Get_MP3_Count()	N/A	현재 목록의 MP3 파일의 개수	현재 목록에서 파일의 개수를 반환한다

03_

알고리즘: 주어진 문제를 해결하기 위한 절차

알고리즘의 5가지 특성:

1) 입력: 외부에서 제공되는 자료가 0개 이상 있어야 한다

2) 출력: 적어도 1개 이상의 결과를 만들어야 한다

3) 명백성: 각 명령어는 의미가 모호하지 않고 명확해야 한다

4) 유한성: 한정된 수의 단계 뒤에는 반드시 종료된다. 무한히 동작해서는 안된다

5) 유효성: 모든 명령은 실행 가능한 연산이어야 한다.

예를 들어 0으로 나누는 연산과 같이 실행할 수 없는 연산을 포함해서는 안 된다.

04_

(1) 시간 복잡도 함수

$n * (1 + 1 + 1 + n * 3) = 3n^2 + 3n$

설명: (바깥 루프가 n번) * (비교 연산 1번 + 증가 연산 1 번 + 대입 연산 1 번

+ 내부 루프가 n번 * (비교 연산 1번 + 증가 연산 1번 + 더하기 연산 1번))

(2) 빅-오 표기법

$O(3n^2 + 3n) = O(n^2)$

05_

(1) $O(6n^3 + n! + 5n + 4) = O(n!)$

(2) $O(4n\log n + 5n^2 + 2) = O(5n^2)$

(3) $O(2^n + n^3 + 5) = O(2^n)$

06_

$O(n) < O(\log n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$

2장

03_

1) 10

2) 12

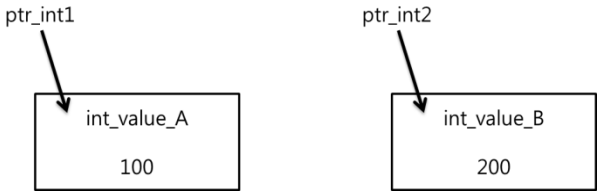
Page 105 문제 03. 2)

(기존) int *ptr_int = &int_value;

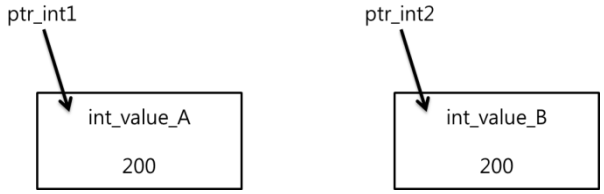
(수정) int *ptr_int = int_value;

04_

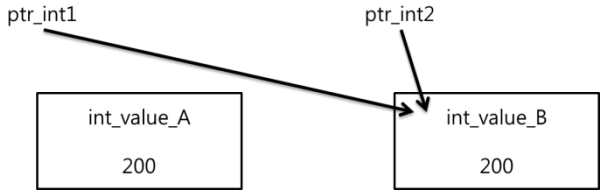
(1) 초기 설정



(2) *ptr_int1 = *ptr_int2 의 수행 결과

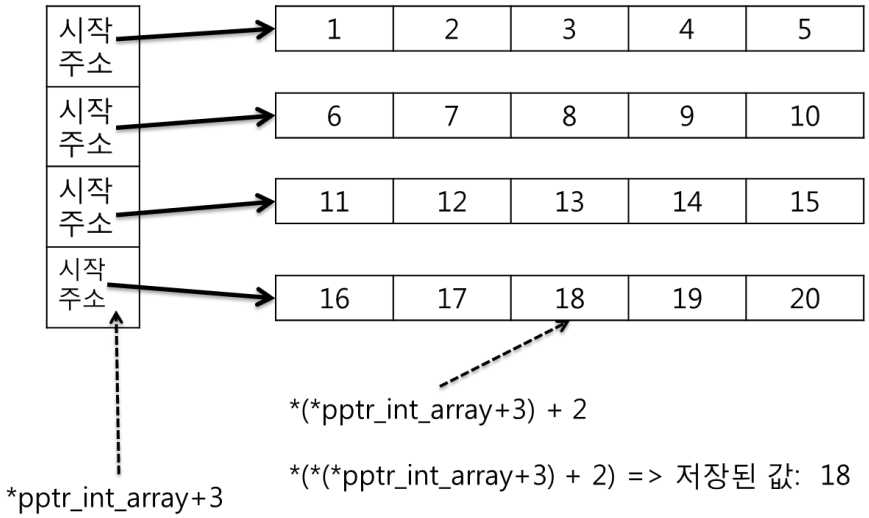


(3) ptr_int1 = ptr_int2 의 수행 결과



05_

(*pptr_int_array+3) + 2 의 값: 18



*(*pptr_int_array+3) => 저장된 '시작 주소' 값

3장

01_

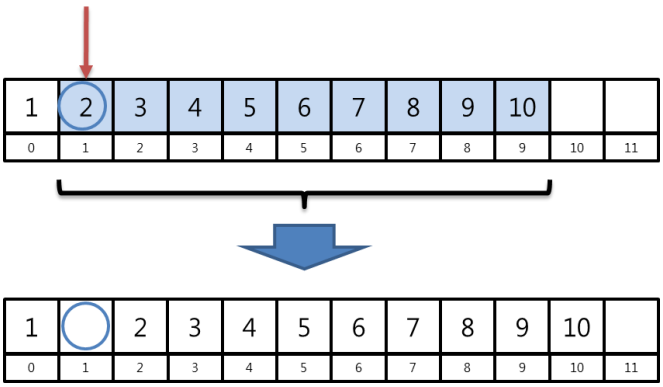
	장점	단점
연결 리스트	1) 원소 추가 혹은 삭제 시 배열 리스트 보다 성능이 우수함: 추가적인 원소 이동이 연산 불필요 2) 최대 원소 개수를 미리 지정해야 하는 제약사항이 없음: 효율적인 메모리 사용이 가능함	1) 탐색 비용이 높음 : 사용하려는 노드를 찾을 때 까지 포인터로 노드들을 탐색해야 함 2) 구현의 어려움: 동적 메모리 할당 및 포인터 연산 등으로 인해 구현 비용이 높음
배열 리스트	1) 탐색 비용이 낮음: 사용하려는 노드를 위치 인덱스를 이용하여 직접 접근이 가능함	1) 원소 추가 혹은 삭제 연산 수행 시 연결 리스트보다 성능이 낮음: 추가적인 원소의 이동이 필요함 2) 최대 원소 개수를 미리 지정해야 하는 제약사항이 있음: 메모리 사용의 낭비가 발생할 가능성이 높음

02_

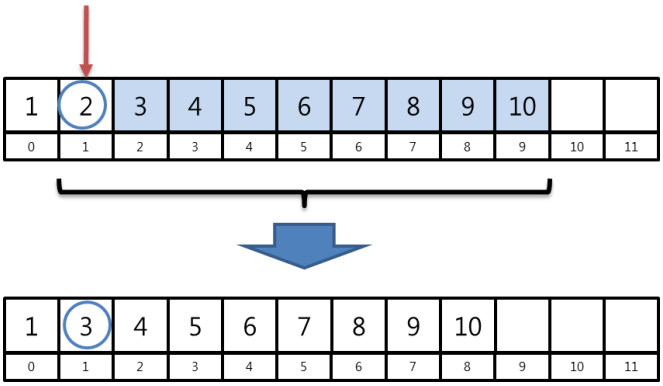
	공통점	차이점
단순 연결 리스트	포인터를 이용하여 구현함	1) 연결 리스트의 가장 기본이 되는 간단한 구조를 가짐
원형 연결 리스트		2) 연결 리스트의 가장 마지막 노드가 첫 번째 노드와 연결되어 원형을 이루는 구조임
이중 연결 리스트		3) 양방향 링크를 사용하기 때문에, 특정 노드의 이전 노드에 대한 직접 접근이 가능함

03_

1) 9개



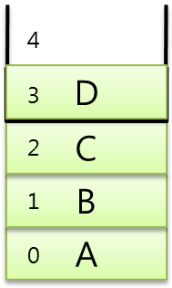
2) 8개



4장

01_

(1)



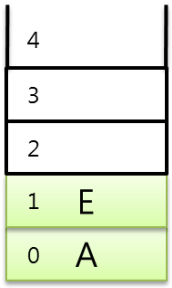
(2)

D → C → B → A

02_

LIFO는 Last-In-First-Out의 줄임말로 가장 나중에 들어간 자료가 가장 먼저 나온다라는 의미임

03_



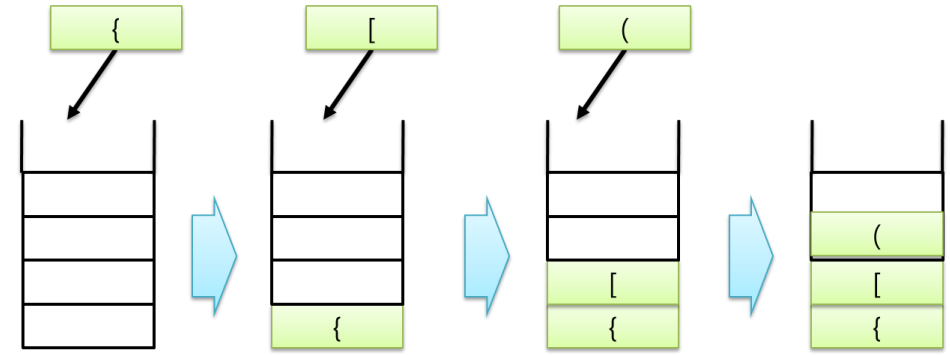
04_

(1)

{ A + [B * (C + D) }] }

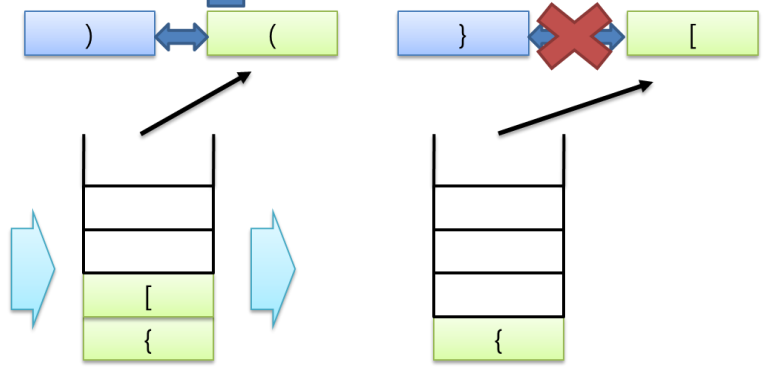
{ A + B [* (C + D) }] }

{ A + [B * (C + D) }] }



{ A + B [* (C + D) }]

{ A + B [* (C + D) }]

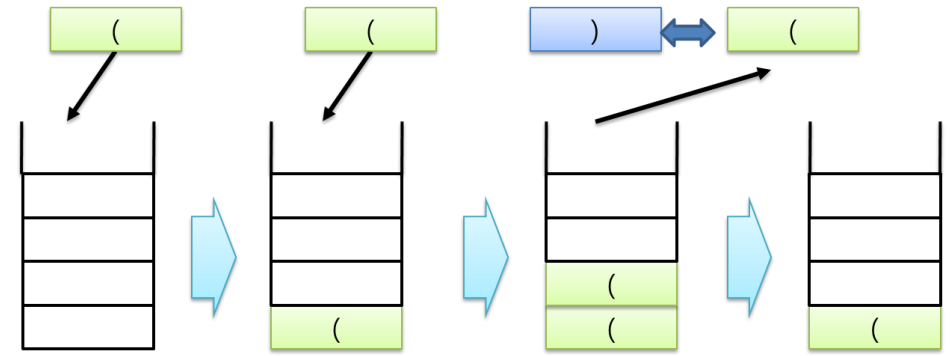


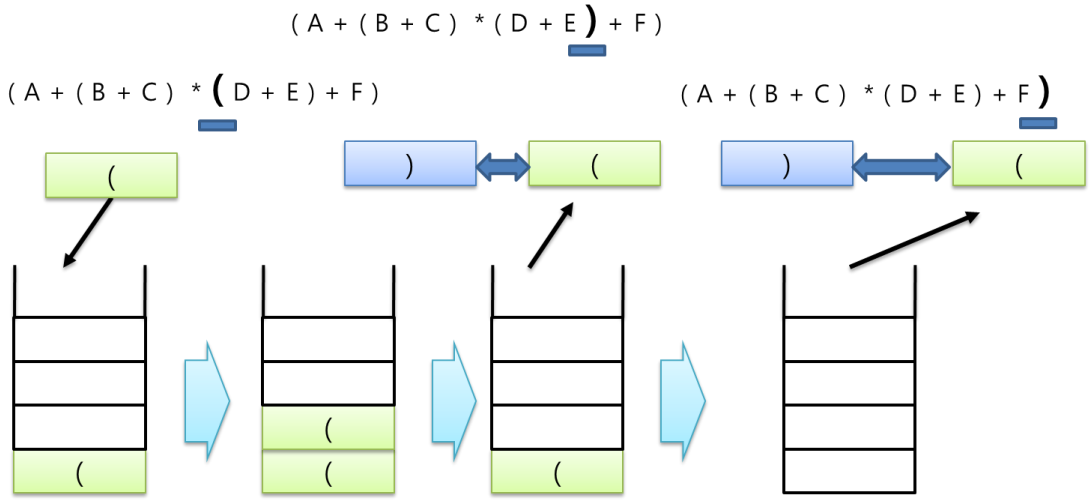
(2)

(A + (B + C) * (D + E) + F)

(A + (B + C) * (D + E) + F)

(A + (B + C) * (D + E) + F)

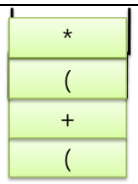
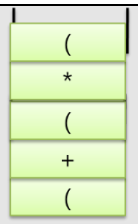
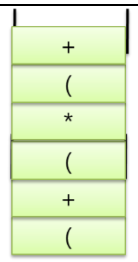
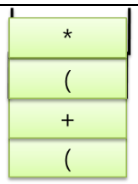
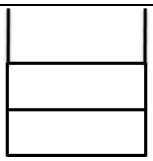


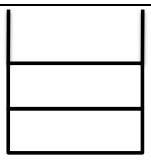


05_

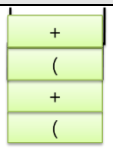
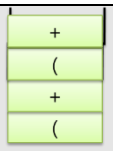
(1) (A + (B * (C + D)))

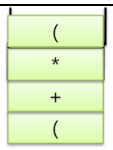
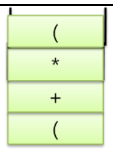
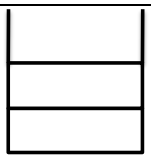
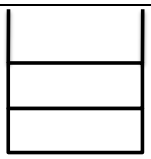
단계		처리	출력 내용	스택의 상태
1	**(** A + (B * (C + D)))	연산자 (를 스택에 push		(
2)	(**A** + (B * (C + D)))	피연산자 A를 출력	A	(
3	(A **+** (B * (C + D)))	연산자 +를 스택에 push	A	+ (
4	(A + **(** B * (C + D)))	연산자 (를 스택에 push	A	(+ (
5	(A + (**B** * (C + D)))	피연산자 B를 출력	A B	(+ (

6	(A + (B ***** (C + D)))	**연산자 ***를 스택에 push	A B	
7	(A + (B * **(** C + D)))	**연산자 (**를 스택에 push	A B	
8	(A + (B * (**C** + D)))	**피연산자 C**를 출력	A B C	
9	(A + (B * (C **+** D)))	**연산자 +**를 스택에 push	A B C	
10	(A + (B * (C + **D**)))	**피연산자 D**를 출력	A B C D	
11	(A + (B * (C + D **)**))	**피연산자 (**를 만날 때까지 pop	A B C D +	
12	(A + (B * (C + D) **)**)	**피연산자 (**를 만날 때까지 pop	A B C D + *	
13	(A + (B * (C + D)) **)**	**피연산자 (**를 만날 때까지 pop	A B C D + * +	

14	〈종료〉	스택에 남은 토큰들 pop	A B C D + * +	

(2) (A + (B + C) * (D + E) + F)

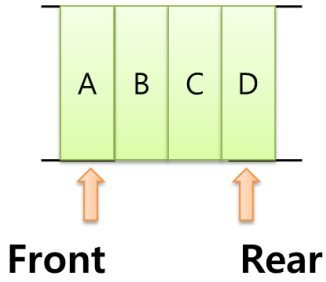
단계		처리	출력 내용	스택의 상태
1	**(** A + (B * C) * (D + E) + F)	연산자 (를 스택에 push		
2)	(**A** + (B + C) * (D + E) + F)	피연산자 A를 출력	A	
3	(A **+** (B + C) * (D + E) + F)	연산자 +를 스택에 push	A	
4	(A + **(** B + C) * (D + E) + F)	연산자 (를 스택에 push	A	
5	(A + (**B** + C) * (D + E) + F)	피연산자 B를 출력	A B	
6	(A + (B **+** C) * (D + E) + F)	연산자 +를 스택에 push	A B	
7	(A + (B * **C**) * (D + E) + F)	피연산자 C를 출력	A B C	
8	(A + (B + C **)** * (D + E) + F)	피연산자 (를 만날 때까지 pop	A B C +	

9	(A + (B + C) ***** (D + E) + F)	***** 를 스택에 push	A B C +	
10	(A + (B + C) * **(** D + E) + F)	(를 스택에 push	A B C +	
11	(A + (B + C) * (**D** + E) + F)	D 를 출력	A B C + D	
12	(A + (B + C) * (D **+** E) + F)	+ 를 스택에 push	A B C + D	
13	(A + (B * C) * (D + **E**) + F)	E 를 출력	A B C + D E	
14	(A + (B * C) * (D + E **)** + F)	(를 만날 때까지 pop	A B C + D E +	
15	(A + (B * C) * (D + E) **+** F)	***** 를 출력 + 를 출력	A B C + D E + ***** +	
16	(A + (B * C) * (D + E) + **F**)	F 를 출력	A B C + D E + ***** + F	
17	(A + (B * C) * (D + E) + F **)**	(를 만날 때까지 pop	A B C + D E + ***** + F +	
18	〈종료〉	스택에 남은 토큰들 pop	A B C + D E + ***** + F +	

5장

01_

(1)



(2)

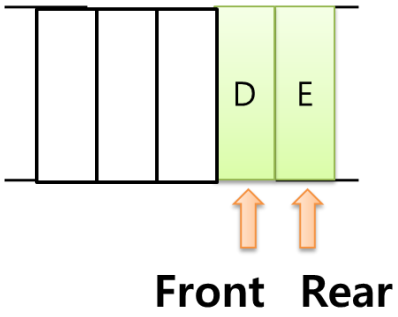
A → B → C → D

02_

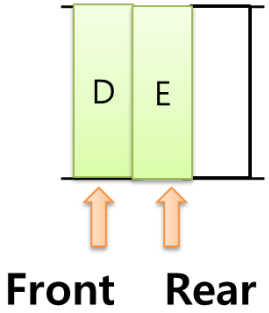
FIFO는 First-In-First-Out의 줄임말로 먼저 저장된 데이터가 나중 저장된 데이터 보다 항상 앞서 나오는 특성을 이야기 합니다. 이를 다른 말로 선입선출이라고도 합니다.

03_

선형 큐 기준



04_



05_

pDeque->pRearNode->pRLink = pNode;

pNode->pLLink = pDeque->pRearNode;

pDeque->pRearNode = pNode;

6장

01_

재귀 호출을 할 때에, 문제의 범위가 줄어들지 않기 때문에 무한 루프에 빠지게 된다.

```
ret = n * factorial( n );
```

02_

재귀 호출에 종료 조건(기본 경우 혹은 최소 한계)이 없기 때문에 무한 루프에 빠지게 된다.

```
if (n >= 1) { .... 과 같은 조건이 없음
```

03_

1

1

2

1

1

2

5

04_

```
int sum(int n) {
    if (n > 1) {
        return sum(n - 1) + n;
    }
    return n;
}
```

05_

```
double sum(double n) {
        if (n > 1) {
                return sum(n – 1) + (1.0 / n);
        }
        return 1;
}
```

06_

```
int sum(int n) {
        int result = 0;
        for(int i = 1; i <= n; i++) {
                result = result + i;
        }
        return result;
}
```

7장

01_

(1) 루트 노드 A

(2) 단말 노드 F, G, H, K, J

(3) 내부 노드 A, B, C, D, E, I

(4) F의 후손 노드 없음

B의 후손 노드 D, F, G

(5) I의 선조 노드 E, C, A

(6) H의 형제 노드 I, J

(7) 노드 F의 차수 0

(8) 노드 G의 높이 1

노드 A의 높이 5

(9) 노드 A의 레벨 1

노드 G의 레벨 4

02_

(1) 전위 순회 과정

+ * A - * B C D * E + F G

(2) 중위 순회 과정

A * B * C - D + E * F + G

(3) 후위 순회 과정

A B C * D - * E F G + * +

03_

노드 개수 n = 40개

(1) 간선의 개수: 39개

(2) 최대 높이(h) = 40 (h ≤ n)

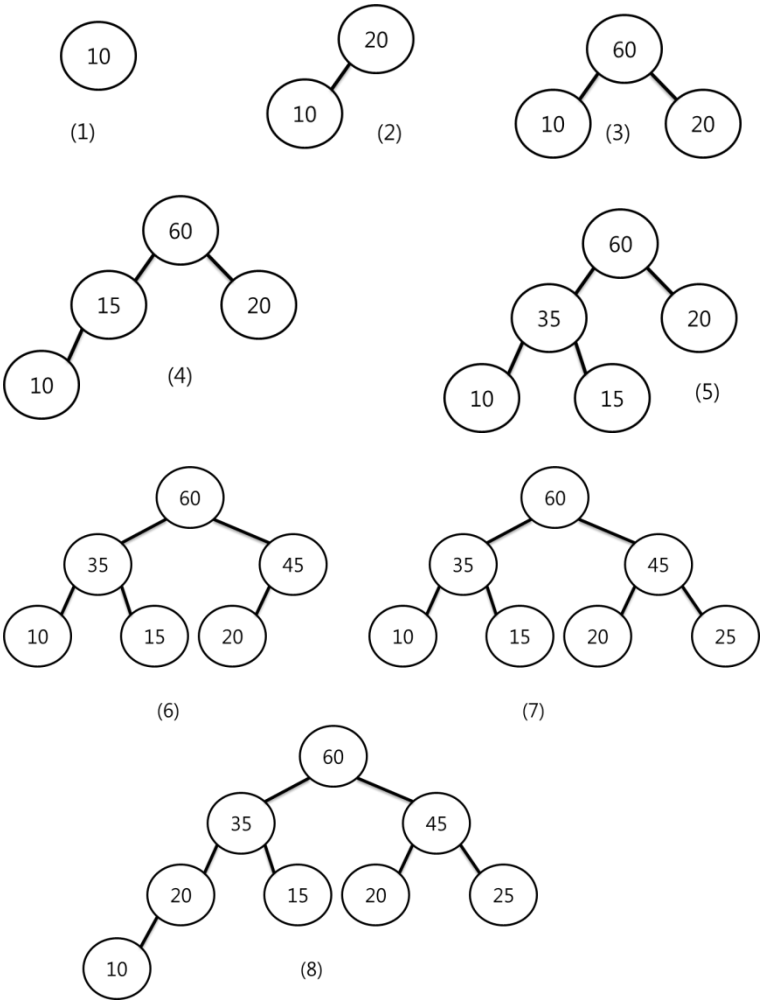
(3) 최소 높이(h) = 6 ($n \le 2^h - 1$)

$2^5 - 1 = 31 < 40$

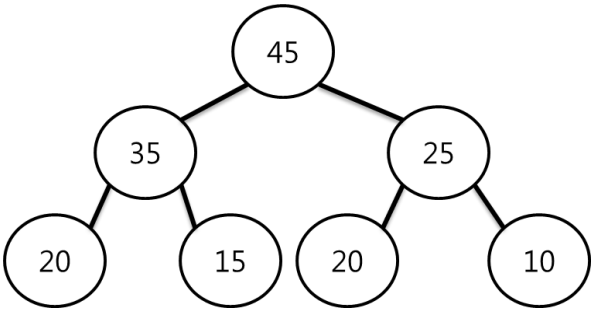
$2^6 - 1 = 63 > 40$

04_

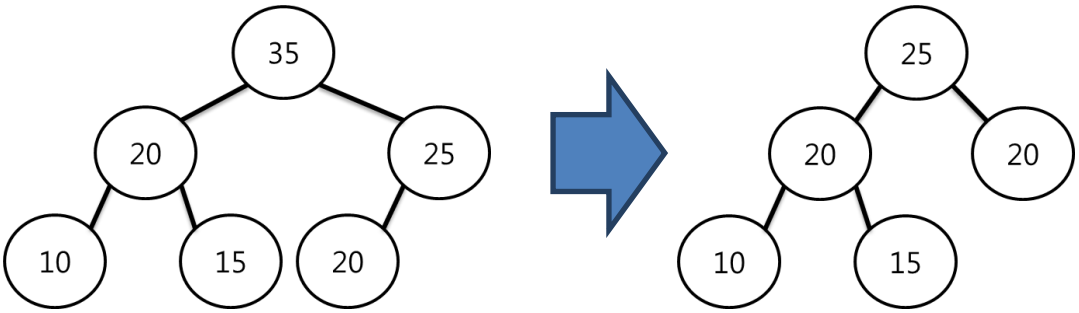
(1)



(2)



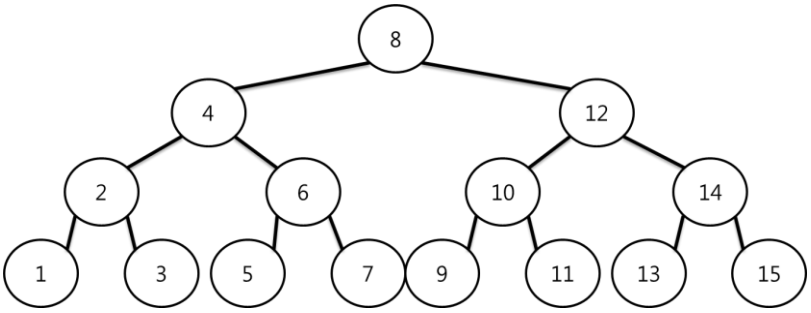
(3) 앞의 (2)에서 2번 더 삭제 연산이 발생한 경우



05_

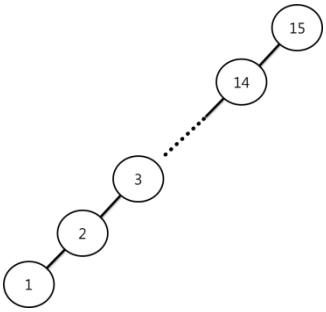
(1)

최소 높이: 4



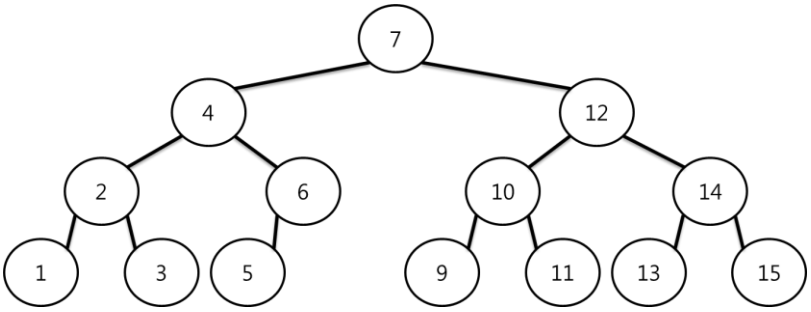
(2)

최대 높이: 15

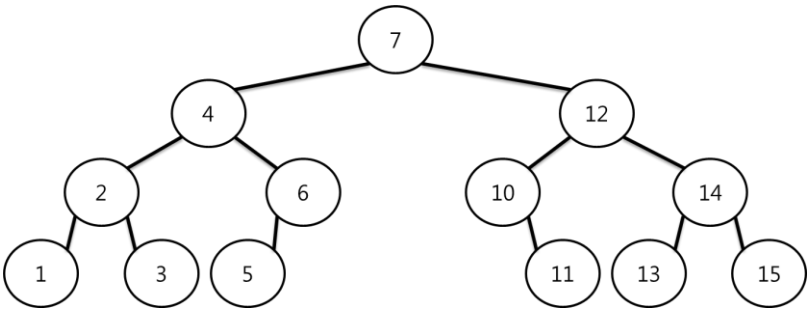


06_

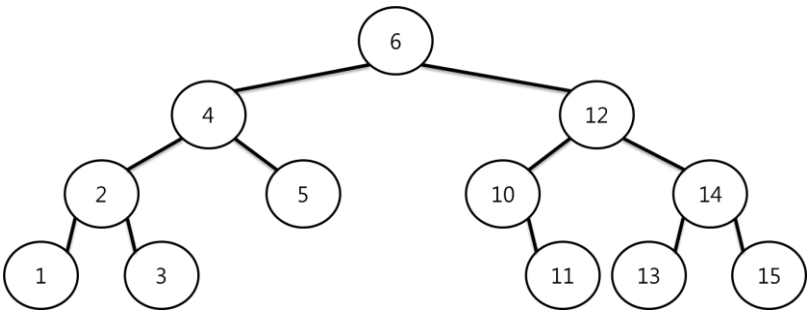
1) 8삭제 후



2) 9삭제후



3) 7삭제후



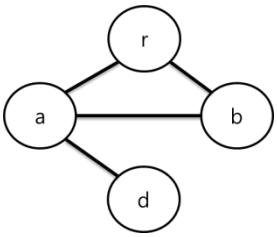
10_

pLeftChildNode != NULL && pLeftChildNode ->visited == FALSE

pushLSBinTreeNode(pStack, pRightChildNode);

8장

01_



02_

G = (V , E)

V(G) = { 0, 1, 2, 3, 4, 5, 6, 7 }

E(G) = {<0,3>, <1,0>, <7,1>, <6,1>, <6,4>, <6,7>, <2,0>, <2,3>, <2,6>, <2,4>,

<4,3>, <5,7>, <5,4>}

03_

(1) 각 노드에서의 진입차수/진출차수

노드(정점)	진입 차수	진출 차수
0	2	1
1	2	1
2	0	4
3	3	0
4	3	1
5	0	2
6	1	3
7	2	1

(2) 각 노드(정점)에서의 인접 노드들

노드(정점)	인접 노드들
0	1, 2, 3
1	0, 6, 7
2	0, 3, 4, 6
3	0, 2, 4

4	2, 3, 5, 6
5	4, 7
6	1, 2, 4, 7
7	1, 5, 6

(3) 인접 행렬 표현

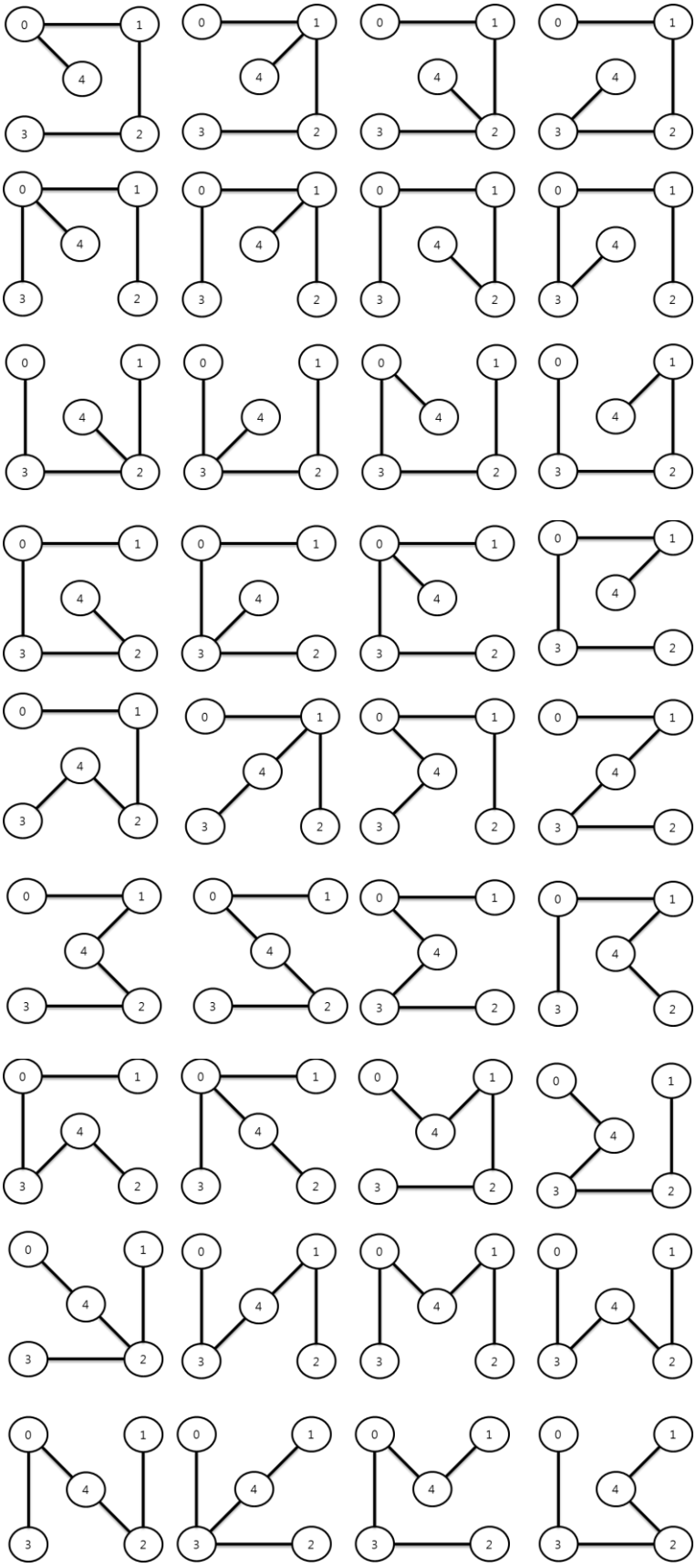
	0	1	2	3	4	5	6	7
0	0	0	0	**1**	0	0	0	0
1	**1**	0	0	0	0	0	0	0
2	**5**	0	0	**7**	**3**	0	**1**	0
3	0	0	0	0	0	0	0	0
4	0	0	0	**2**	0	0	0	0
5	0	0	0	0	**1**	0	0	**7**
6	0	**3**	0	0	**1**	0	0	**1**
7	0	**1**	0	0	0	0	0	0

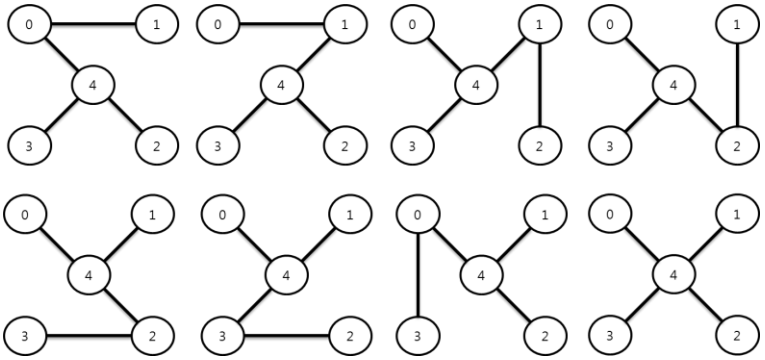
(4) 인접 리스트 표현 (인접 및 가중치 표현)

0	→ [3|1]
1	→ [0|1]
2	→ [0|5] → [3|7] → [4|3] → [6|1]
3	
4	→ [3|2]
5	→ [4|1] → [7|7]
6	→ [1|3] → [4|1] → [7|1]
7	→ [1|1]

04_

신장 트리로 모두 44가지가 가능함

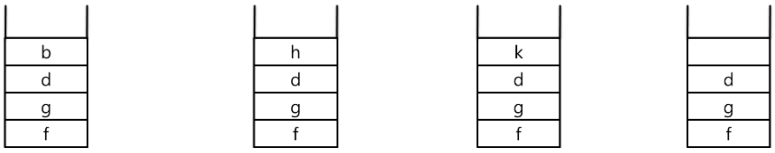

0 1 4 3 2



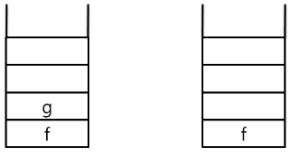
05_

(1) 깊이 우선 탐색

a → b → h → k →

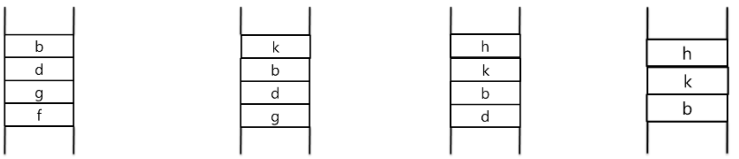


d → g → f



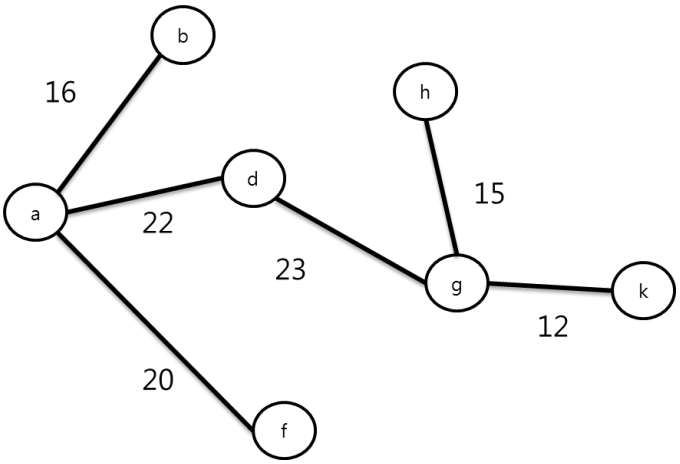
(2) 넓이 우선 탐색

a → f → g → d →



b → k → h

(3)~(4) 최소 비용 신장 트리



(3) Kruskal 알고리즘

1) g-k, 최소 가중치 12

2) g-h, 최소 가중치 15

3) a-b, 최소 가중치 16

4) a-f, 최소 가중치 20

5) a-d, 최소 가중치 22

6) d-g, 최소 가중치 23

(4) Prim 알고리즘

0) 시작 노드를 b로 설정

1) b-a, 최소 가중치 12

2) a-f, 최소 가중치 20

3) a-d, 최소 가중치 22

4) d-g, 최소 가중치 23

5) g-k, 최소 가중치 12

6) g-h, 최소 가중치 15

06_

(1) 노드(정점) 2에서 최단 경로: Dijkstra 알고리즘

1) 초기화

<2,0>: 5

<2,1>: ∞

<2,3>: 7

<2,4>: 3

<2,5>: ∞

<2,6>: 1

<2,7>: ∞

2) 최단 경로

	1 번째		**2 번째**		**3 번째**		**4 번째**		**5 번째**		**6,7번째**
<2,0>	5		5		5		5		4		4
<2,1>	∞		4		4		3		3		3
<2,3>	7	→	7	→	4	→	4	→	4	→	4
<2,4>	3		2		2		2		2		2
<2,5>	∞		∞		∞		∞		∞		∞
<2,6>	1		1		1		1		1		1
<2,7>	∞		2		2		2		2		2
		노드6 선택		노드4 선택		노드7 선택		노드1 선택		노드 0과 3 선택	

(2) 그래프 상의 모든 최단 경로: Floyd 알고리즘

1) 초기화

	0	1	2	3	4	5	6	7
0	0	∞	∞	1	∞	∞	∞	∞
1	1	0	∞	∞	∞	∞	∞	∞
2	5	∞	0	7	3	∞	1	∞
3	∞	∞	∞	0	∞	∞	∞	∞
4	∞	∞	∞	2	0	∞	∞	∞
5	∞	∞	∞	∞	1	0	∞	7
6	∞	3	∞	∞	1	∞	0	1
7	∞	1	∞	∞	∞	∞	∞	0

2) 최단 경로

	0	1	2	3	4	5	6	7
0	0	∞	∞	1	∞	∞	∞	∞
1	1	0	∞	2	∞	∞	∞	∞
2	4	3	0	4	2	∞	1	2
3	∞	∞	∞	0	∞	∞	∞	∞
4	∞	∞	∞	2	0	∞	∞	∞
5	9	8	∞	3	1	0	∞	7
6	3	2	∞	3	1	∞	0	1
7	2	1	∞	3	∞	∞	∞	0

9장

01_

70 50 80 60(a) 30 90 60(b) 40

(1) 삽입 정렬

Step-1	50	70	80	60(a)	30	90	60(b)	40
Step-2	50	70	80	60(a)	30	90	60(b)	40
Step-3	50	60(a)	70	80	30	90	60(b)	40
Step-4	30	50	60(a)	70	80	90	60(b)	40
Step-5	30	50	60(a)	70	80	90	60(b)	40
Step-6	30	50	60(a)	60(b)	70	80	90	40
Step-7	30	40	50	60(a)	60(b)	70	80	90

(2) 버블 정렬

Step-1	50	70	60(a)	30	80	60(b)	40	90
Step-2	50	60(a)	30	70	60(b)	40	80	90
Step-3	50	30	60(a)	60(b)	40	70	80	90
Step-4	30	50	60(a)	40	60(b)	70	80	90
Step-5	30	50	40	60(a)	60(b)	70	80	90
Step-6	30	40	50	60(a)	60(b)	70	80	90
Step-7	30	40	50	60(a)	60(b)	70	80	90

(3) 선택 정렬

Step-1	30	50	80	60(a)	70	90	60(b)	40
Step-2	30	40	80	60(a)	70	90	60(b)	50
Step-3	30	40	50	60(a)	70	90	60(b)	80
Step-4	30	40	50	60(a)	70	90	60(b)	80
Step-5	30	40	50	60(a)	60(b)	90	70	80
Step-6	30	40	50	60(a)	60(b)	70	90	80
Step-7	30	40	50	60(a)	60(b)	70	80	90

(4) 셸 정렬

70 50 80 60(a) 30 90 60(b) 40

간격 4: {70, 30}, {50, 90|, {80, 60(b)}, {60(a),40}

간격 4	30	50	60(b)	40	70	90	80	60(a)

간격 2: {30, 60(b), 70, 80}, {50, 40, 90, 60(a)}

간격 2	30	40	60(b)	50	70	60(a)	80	90

간격 1: {30, 40, 60(b), 50, 70, 60(a), 80, 90}

간격 1	30	40	50	60(a)	60(b)	70	80	90

02_

(1) 퀵 정렬

62, 48, 93, 78, 24, 65, 23, 9

1) 피봇 9

62	48	93	78	24	65	23	9 *

2) 피봇 23

62	48	93	78	24	65	23*	9

3) 피봇 65

78	93	65*	62	24	48	23	9

4) 피봇 93

93*	78	65	62	24	48	23	9

5) 피봇 48

93	78	65	62	48*	24	23	9

정렬 결과: 93 78 65 62 48 24 23 9

(2) 병합 정렬

62, 48, 93, 78, 24, 65, 23, 9

1) 분할

62, 48, 93, 78, 24, 65, 23, 9

2) 병합 (1)

{62, 48} {93, 78}, {65, 24}, {23, 9}

3) 병합 (2)

{93, 78, 62, 48}, {65, 24, 23, 9}

4) 병합 (3)

{93, 78, 65, 62, 48, 24, 23, 9}

(3) 기수 정렬

62, 48, 93, 78, 24, 65, 23, 9

1) 1의 자릿수

9 48 78 65 24 93 23 62

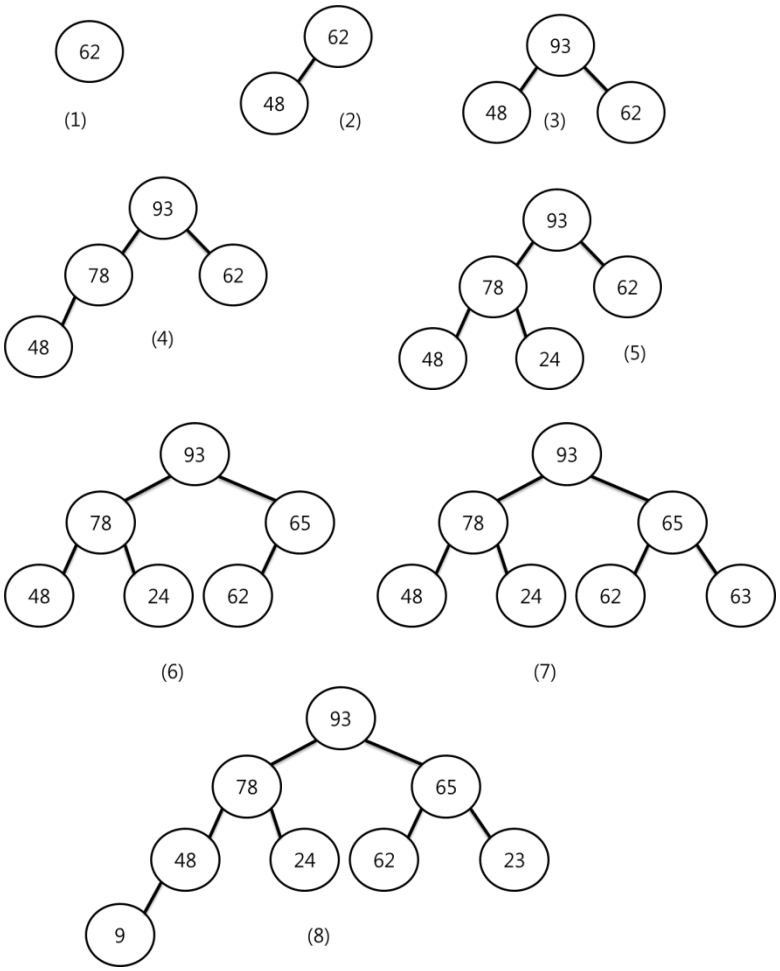
2) 10의 자릿수

93 78 65 62 48 24 23 62

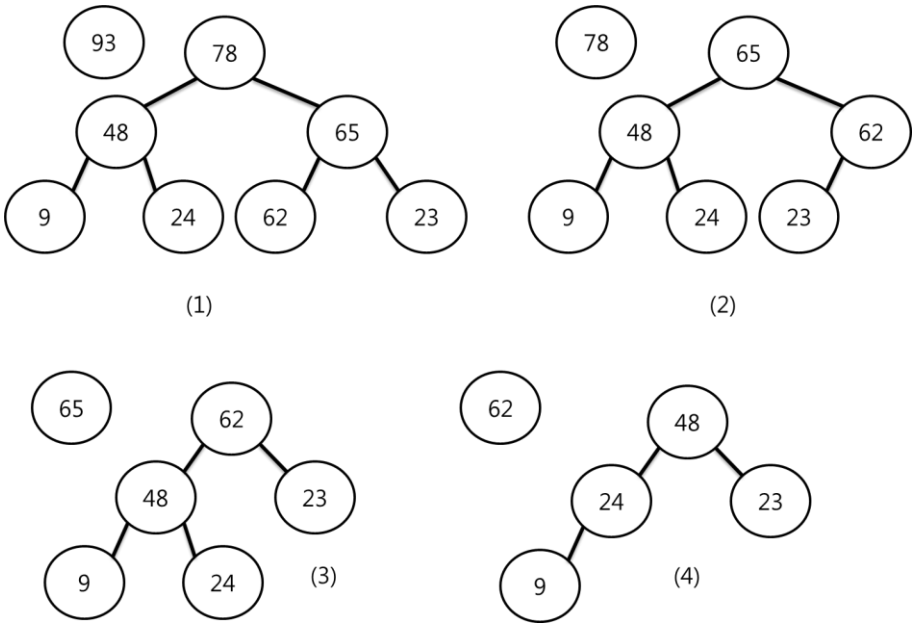
(4) 히프 정렬

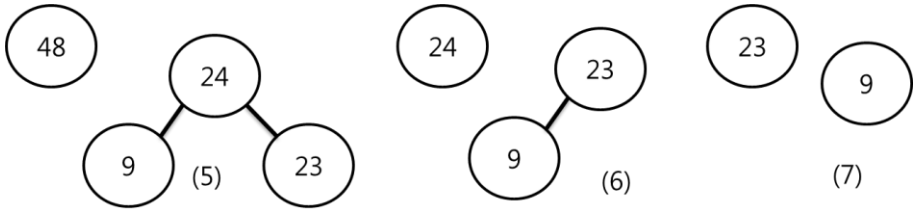
62, 48, 93, 78, 24, 65, 23, 9

1) 삽입 단계



2) 삭제 단계





03_

퀵 정렬의 특성

퀵 정렬은 인접한 피봇을 기준으로 두 자료의 키 값을 비교하여 위치를 교환하는 알고리즘으로,자료의 이동 연산이 비교 연산보다 상대적으로 적게 발생한다. 단, 피봇을 기준으로 나누어지는 2개의 부분 집합에 불균형이 발생하면, 즉 기존 자료의 정렬 정도에 따라 효율성이 최악의 경우 $O(n^2)$이 될 수 있다

퀵 정렬의 효율성

최선 혹은 평균: $O(n\log_2 n)$, 최악: $O(n^2)$

퀵 정렬의 안정성

안정성이 유지되지 않는 불안정 알고리즘이다.

04_

평균 기준

$O(n^2)$: 선택 정렬, 버블 정렬, 삽입 정렬

$O(n\log_2 n)$: 퀵 정렬, 병합 정렬, 히프 정렬

$O(n)$: 기수 정렬

10장

01_

2번

9 → 14

02_

abc의 경우

호너의 방법 (자릿수 31 사용)

((97 * 31 + 98) * 31 + 99) = 96354

def의 경우

호너의 방법 (자릿수 31 사용)

((100 * 31 + 101) * 31 + 102) = 99333

(1) 이동 접기 함수

1) **문자열 abc의 키 값: 17**

해시 테이블 크기 M = 127 (3자리)

963 54 : 3자리 2개로 분할

963
+ 54
1,017

초과 자릿수인 천의 자리 1을 버리면, 최종적으로 17이 남음

2) **문자열 def의 키 값 26**

해시 테이블 크기 M = 127 (3자리)

993 33 : 3자리 2개로 분할

993
+ 33
1,026

초과 자릿수인 천의 자리 1을 버리면, 최종적으로 26이 남음

(2) 중간 제곱 함수

1) **문자열 abc의 키 값: 93**

96354 * 96354 = 9,28**4,093**,316

중간 4자리 4,093에서 초과 자릿수 4 버림

2) **문자열 def의 키 값 44**

99333 * 99333 = 9,86**7,044**,889

중간 4자리 7,044에서 초과 자릿수 7 버림

03_

(1) 선형 조사법

89 15 12 13 4 30 1 2 25 27

h(k)=k mod 11

h(85) = 85 mod 11 = 8

h(15) = 15 mod 11 = 4

h(12) = 12 mod 11 = 1

h(13) = 13 mod 11 = 2

h(4) = 4 mod 11 = 4 (충돌)

h(4) = (4 + 1) mod 11 = 5

h(30) = 30 mod 11 = 8 (충돌)

h(30) = (30 + 1) mod 11 = 9

h(1) = 1 mod 11 = 1 (충돌)

h(1) = (1 + 1) mod 11 = 2 (충돌)

h(1) = (1 + 2) mod 11 = 3

h(2) = 2 mod 11 = 2 (충돌)

h(2) = (2 + 1) mod 11 = 3 (충돌)

h(2) = (2 + 2) mod 11 = 4 (충돌)

h(2) = (2 + 3) mod 11 = 5 (충돌)

h(2) = (2 + 4) mod 11 = 6

h(25) = 25 mod 11 = 3 (충돌)

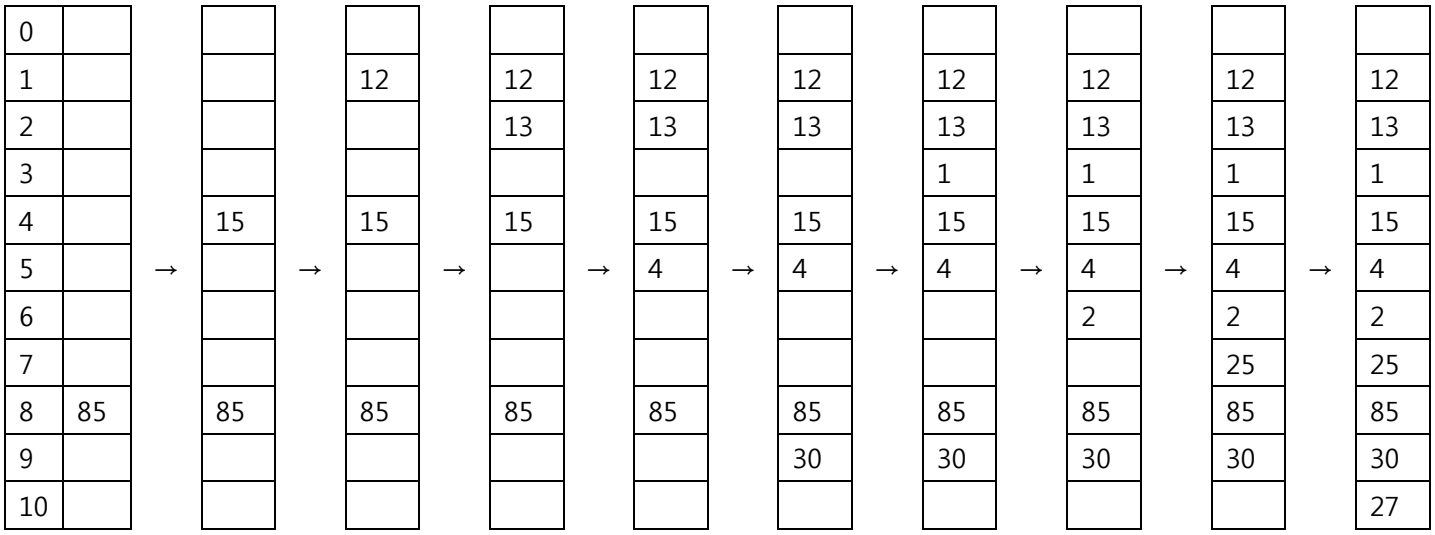
.....

h(25) = (25 + 4) mod 11 = 7

h(27) = 27 mod 11 = 5 (충돌)

....

h(27) = (27 + 5) mod 11 = 10



(2) 제곱 조사법

89 15 12 13 4 30 1 2 25 27

h(85) = 85 mod 11 = 8

h(15) = 15 mod 11 = 4

h(12) = 12 mod 11 = 1

h(13) = 13 mod 11 = 2

h(4) = 4 mod 11 = 4 (충돌)

h(4) = (4 + 1) mod 11 = 5

h(30) = 30 mod 11 = 8 (충돌)

h(30) = (30 + 1) mod 11 = 9

h(1) = 1 mod 11 = 1 (충돌)

h(1) = (1 + 1) mod 11 = 2 (충돌)

h(1) = (1 + 2) mod 11 = 3

h(2) = 2 mod 11 = 2 (충돌)

h(2) = (2 + 1) mod 11 = 3 (충돌)

h(2) = (2 + 4) mod 11 = 6

h(25) = 25 mod 11 = 3 (충돌)

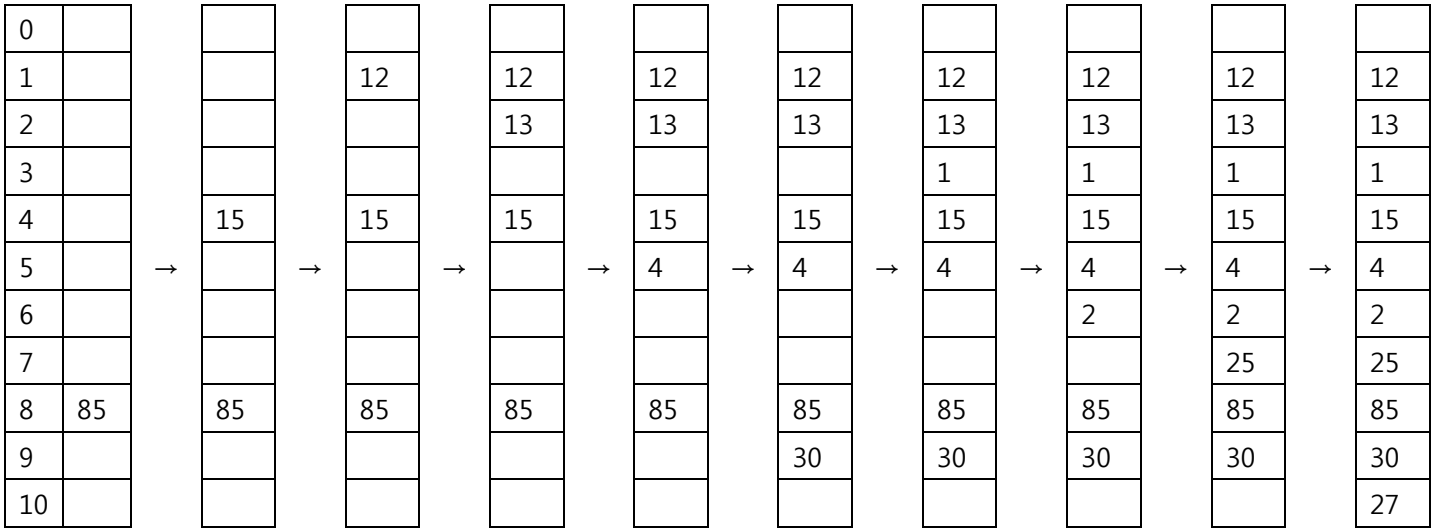
.....

h(25) = (25 + 4) mod 11 = 7

h(27) = 27 mod 11 = 5 (충돌)

....

h(27) = (27 + 16) mod 11 = 43 mod 11 = 10



(3) 이중 해싱 기법 (조사 간격 M = 5 – (k mod 5))

89 15 12 13 4 30 1 2 25 27

h(85) = 85 mod 11 = 8

h(15) = 15 mod 11 = 4

h(12) = 12 mod 11 = 1

h(13) = 13 mod 11 = 2

h(4) = 4 mod 11 = 4 (충돌)

조사간격 M = 5 – (4 mod 5) = 1

h(4) = (4 + 1) mod 11 = 5

h(30) = 30 mod 11 = 8 (충돌)

조사간격 M = 5 – (8 mod 5) = 2

h(30) = (30 + 2) mod 11 = 10

h(1) = 1 mod 11 = 1 (충돌)

조사간격 M = 5 – (1 mod 5) = 4

h(1) = (1 + 4) mod 11 = 5 (충돌)

h(1) = (5 + 4) mod 11 = 9

h(2) = 2 mod 11 = 2 (충돌)

조사간격 M = 5 – (2 mod 5) = 3

h(2) = (2 + 3) mod 11 = 5 (충돌)

h(2) = (5 + 3) mod 11 = 8 (충돌)

h(2) = (8 + 3) mod 11 = 0

h(25) = 25 mod 11 = 3 (충돌)

조사간격 M = 5 – (25 mod 5) = 5

h(25) = (25 + 5) mod 11 = 8 (충돌)

h(25) = (8 + 5) mod 11 = 2 (충돌)

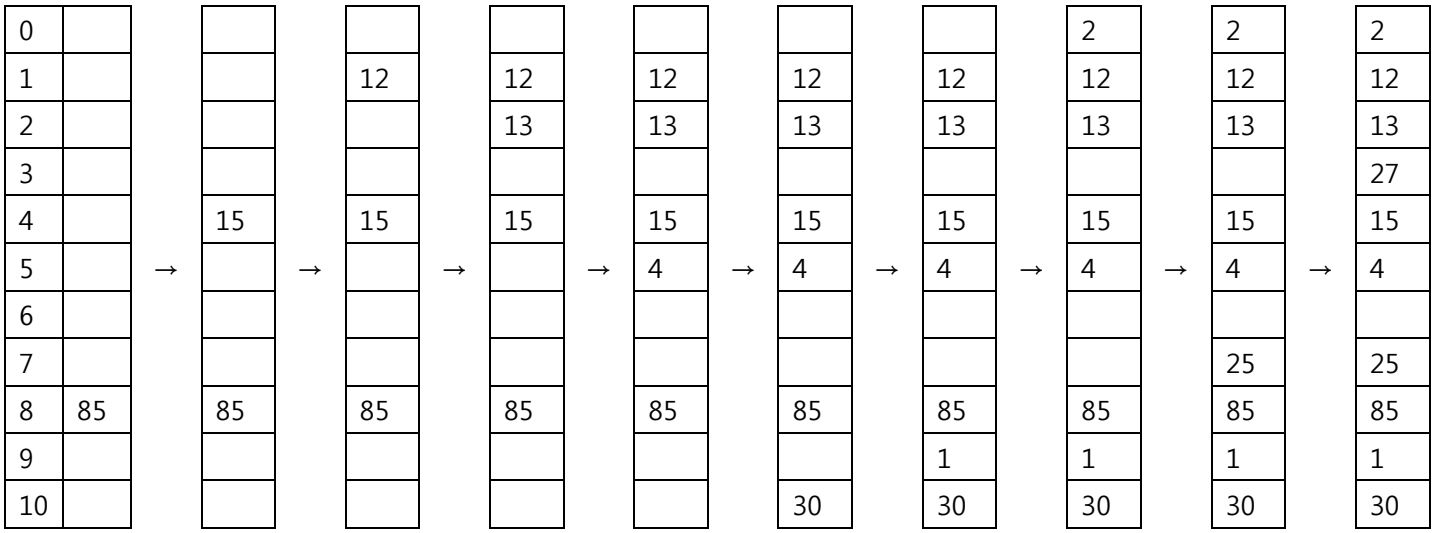
h(25) = (2 + 5) mod 11 = 7

h(27) = 27 mod 11 = 5 (충돌)

조사간격 M = 5 – (27 mod 5) = 3

h(27) = (27 + 3) mod 11 = 8 (충돌)

h(27) = (0 + 3) mod 11 = 3

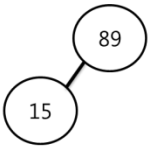


05_

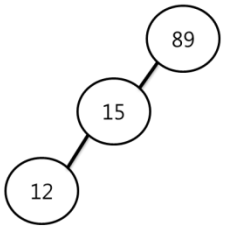
89 → 15 → 12 → 13 → 4 → 30 → 1 →2 → 25 → 27

1) 이진탐색 트리의 경우

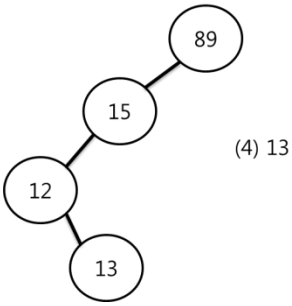
(1) 89



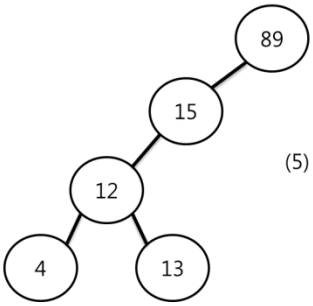
(2) 15



(3) 12

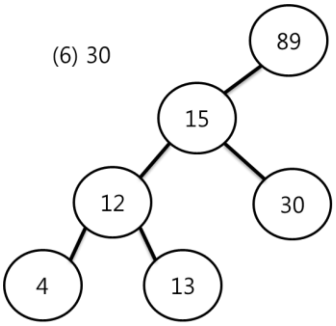


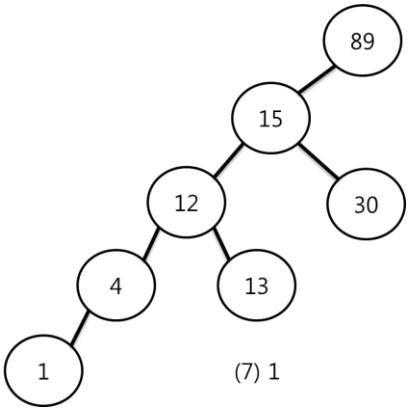
(4) 13



(5) 4

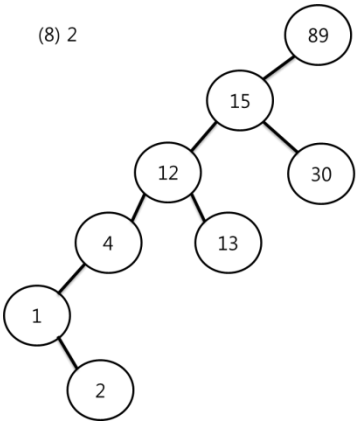
(6) 30



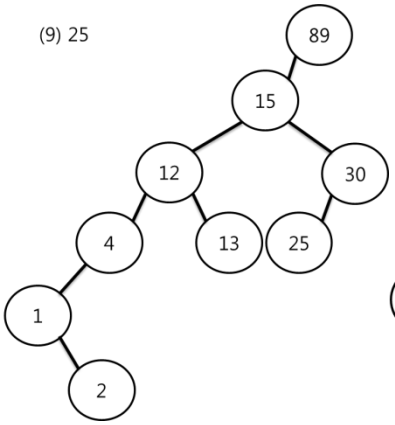


(7) 1

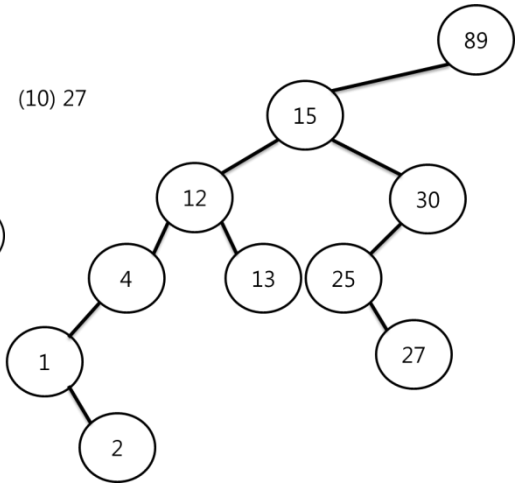
(8) 2



(9) 25

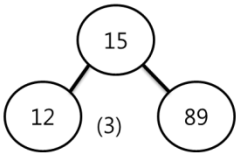


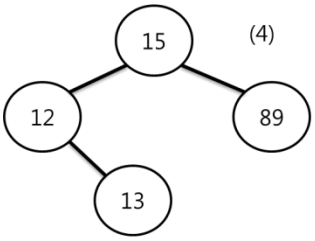
(10) 27

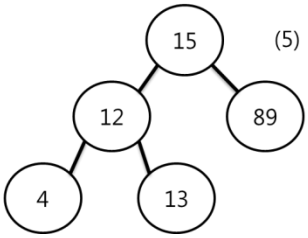


2) AVL트리의 경우

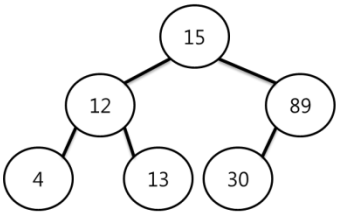
(1) 89 (2) 15 (3) 12

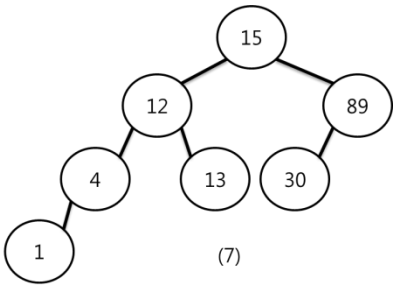


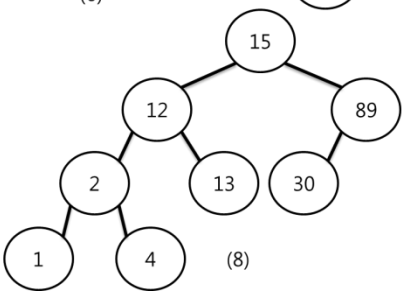




(4) 13 (5) 4

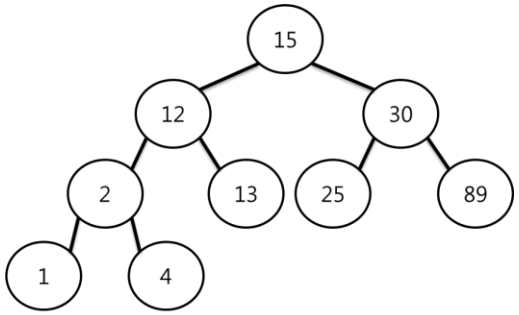




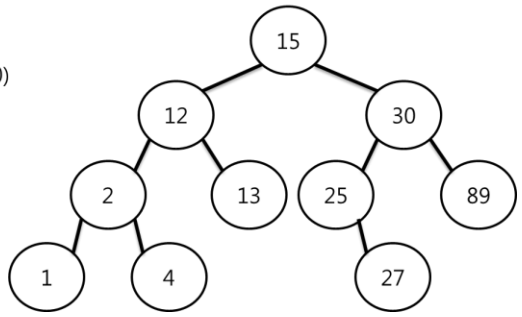


(6) 30 (7) 1 (8) 2

(9)



(10)



(9)25 (10) 27

06_ 최대 자료 저장 개수

(1) 높이가 3, 4-원 탐색 트리

(최대 자료 저장 개수) = 3 + 12 + 96 = 111 개

m=4, 각 노드는 최대 3개(=4 – 1)개의 자료를 가짐

높이 1: 루트 노드가 4개의 서브트리 가짐, 루트 노드는 자료 3개 가짐

높이 2: 높이 2인 노드는 모두 4개이며, 각각 자료 3개씩 가짐, 따라서, 모두 12개의 자료 저장

아울러, 높이 2인 4개의 서브트리가 각각 4개의 서브트리 가짐,

높이 3: 높이 3인 노드는 모두 16개(=4 * 4)이며, 각각 자료 3개씩 가짐, 따라서, 모두 96개 자료 저장

높이가 3이므로 서브트리는 0개임

(2) 높이가 4, 5-원 탐색 트리

(최대 자료 저장 개수) = 4 + 20 + 100 + 500 = 624 개

m=5, 각 노드는 최대 4개(=5 – 1)개의 자료를 가짐

높이 1: 루트 노드가 5개의 서브트리 가짐, 루트 노드는 자료 4개 가짐

높이 2: 높이 2인 노드는 모두 5개이며, 각각 자료 4개씩 가짐, 따라서, 모두 20개의 자료 저장

아울러, 높이 2인 5개의 서브트리가 각각 5개의 서브트리 가짐,

높이 3: 높이 3인 노드는 모두 25개(=5 * 5)이며, 각각 자료 4개씩 가짐,

따라서, 모두 100(=25*4)개 자료 저장

높이가 3인 25개의 서브트리가 각각 5개의 서브트리 가짐

높이 4: 높이 4인 노드는 모두 125개(=25 * 5)이며, 각각 자료 4개씩 가짐,

따라서, 모두 500개 자료 저장

높이가 4이므로 서브트리는 0개임

(3) 높이가 h, m-원 탐색 트리

(최대 자료 저장 개수) = (m-1)

+ (m - 1) * m

+ (m - 1) * m * m

+ ....

+ (m - 1) * $m^{(h - 1)}$

= m^h – 1

07_ 최대 자료 저장 개수

(1) 높이가 3, 차수가 5인 B-트리

124

(2) 높이가 5, 차수가 5인 B-트리

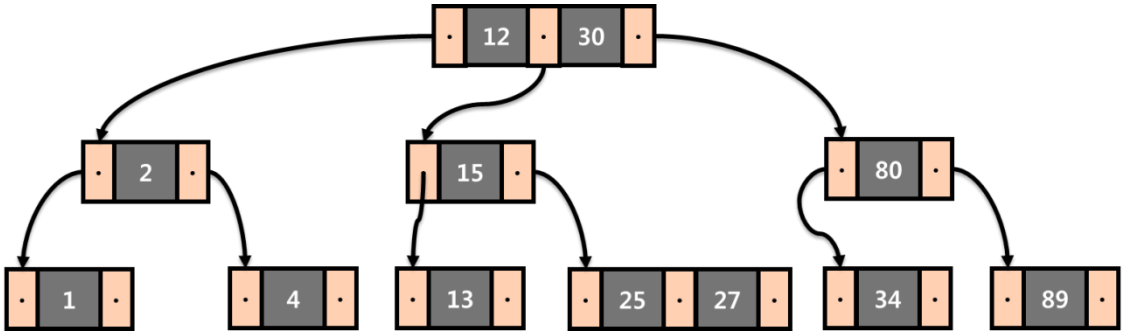
3124

(3) 높이가 h, 차수가 5인 B-트리

5^h - 1

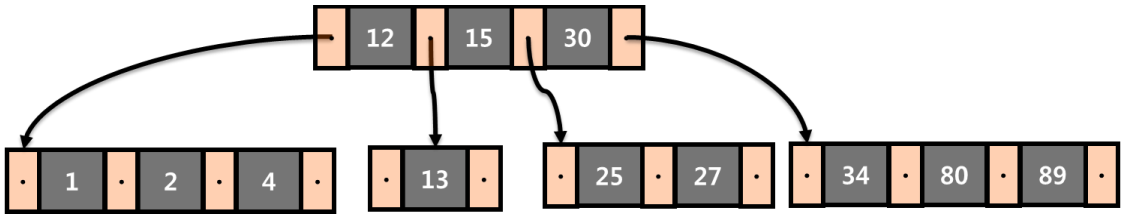
08_ m=3인 B-트리

85 → 15 → 12 → 13 → 4 → 30 → 1 → 2 → 25 → 27 → 34 → 80



09_ m=4인 B-트리

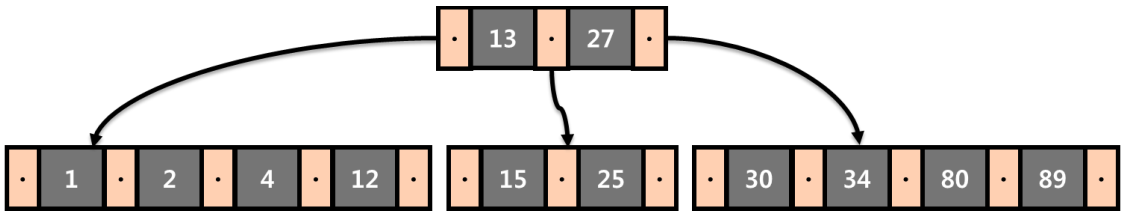
85 → 15 → 12 → 13 → 4 → 30 → 1 → 2 → 25 → 27 → 34 → 80



10_ ~ 11_

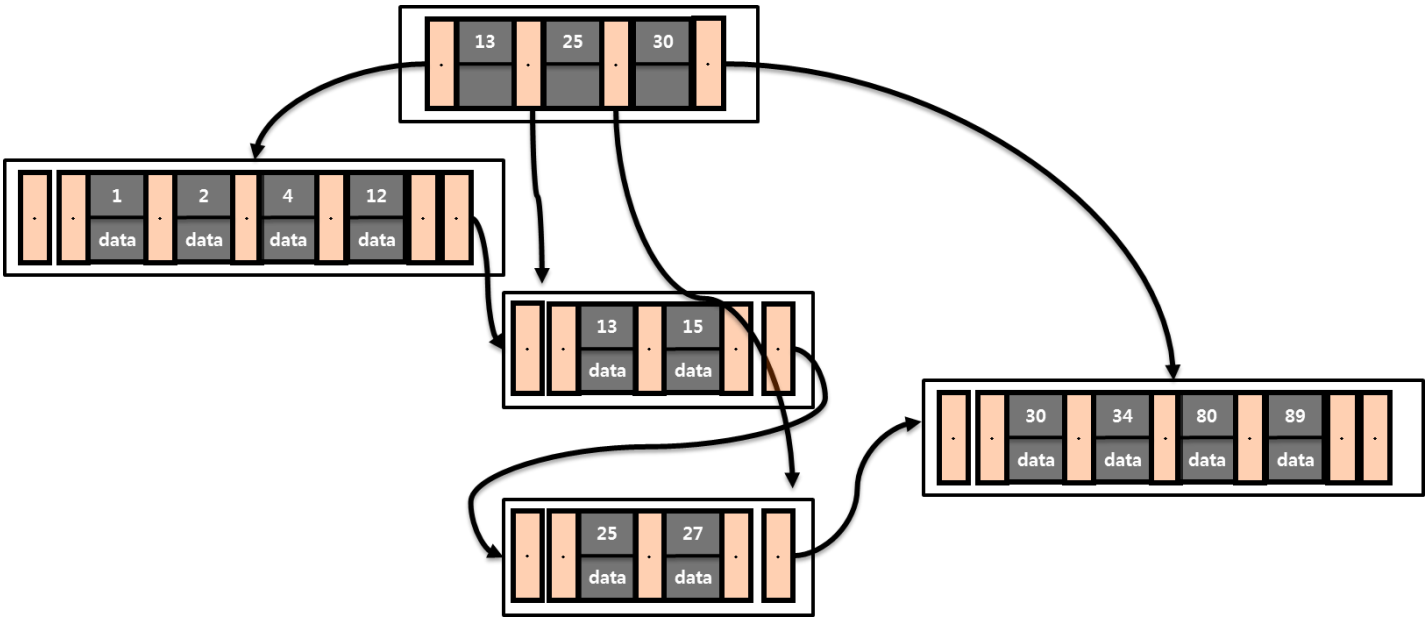
85 → 15 → 12 → 13 → 4 → 30 → 1 → 2 → 25 → 27 → 34 → 80

m=5인 B-트리



85 → 15 → 12 → 13 → 4 → 30 → 1 → 2 → 25 → 27 → 34 → 80

m=5인 B+트리



85 → 15 → 12 → 13 → 4 → 30 → 1 → 2 → 25 → 27 → 34 → 80

m=5인 B*트리

이 경우 앞의 m=5, B-트리와 동일함