2016족보

출석	☑
열	@June 21, 2024

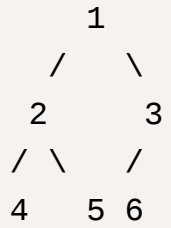
1. **완전이진트리 complete binary가 무엇인지 설명하고, n개의 원소를 갖는 완전이진트리의 높이를 구하는 식을 작성해라**

- 완전 이진 트리 : 높이가 k일때, level 1부터 k-1까지는 노드가 모드 채워져있고, 마지막 레벨 k에서는 왼쪽부터 오른쪽으로 노드가 순서대로 채워져 있는 트리
- [높이 구하는 식]

  트리의 높이 h = 루트 - 마지막레벨까지의 깊이

  최대 노드의 수 : 2^(h+1) -1 이다.

  노드의 개수를 n이라할 때 완전 이진트리의 높이는 최선의 경우 h=⌈log2(n)⌉, 최악의 경우 h=n이다.



💡 이진트리의 순회
VLR : preorder
LVR : inorder
LRV : postorder

2. **분할 정복 Dive and Conquer 방법은 알고리즘 설계에 사용되는 중요한 패러다임 중 하나임. 분할 정복이란 어떤 알고리즘 설계 패러다임인지 설명하고, 이 같은 패러다임으로 설계된 알고리즘의 예를 한 개 들어라**

- 분할정복 : 문제를 작은 2개의 문제로 분리하고 각각을 해결한 다음 결과를 모아서 원래의 문제를 해결
  - 분할 : 입력 배열을 같은 크기의 2개의 부분 배열로 나눔
  - 정복 : 부분배열을 정렬함. 부분 배열의 크기가 충분히 작지 않으면 재귀호출을 이용하여 다시 분할정복

    - 결합 : 정렬된 부분배열을 하나의 배열에 통ㅎ압
- 알고리즘 : 퀵정렬, 합병정렬

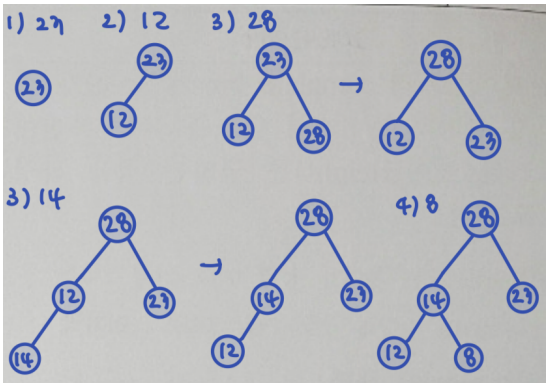
```
#include <stdio.h>
#include <stdlib.h>

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int* L = (int*)malloc(n1 * sizeof(int));
    int* R = (int*)malloc(n2 * sizeof(int));

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int i = 0; i < n2; i++)
        R[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
```

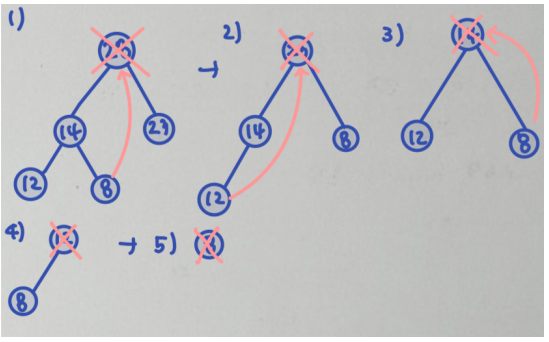
```
        j++;
        k++;
    }

    free(L);
    free(R);
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

void printArray(int A[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```

3. **다음의 순서로 [{23,12,28,14,8} - 23이 제일 먼저 입력 됨] 자료가 입력될 경우 최대 힙 Max Heap을 구성하고, 차례로 루트 노드를 제거해 나갈 경우에 남은 노드들로 최대 힙을 구성해나가는 과정을 그림으로 나타내라**

- 최대힙 : 부모 노드의 키값이 자식 노드의 키값보다 크거나 같은 완전 이진 트리

  히프의 높이 : n개의 노드를 가지고 있는 히프의 노드는 O(log n)





4. **{69 10 30 2 16 8 31 22}로 구성되는 자료들에 대해 1) 퀵 정렬의 동작 과정을 각 단계별로 기술하고, 2) 평균 시간 복잡도를 설명하고 Big O 표현으로 나타내라 3) 최악 시간 복잡도에 대해 Big O 표현을 나타내고 어떤 경우에 발생하는 지를 설명해라**

- 퀵 정렬 : 분할 정복 알고리즘을 사용한 정렬 방법으로, 평균적으로 매우 빠른 성능을 자랑함. 리스트를 2개의 부분리스트로 비균등 분할하고, 각각의 부분리스트를 다시 퀵정렬함(재귀호출)
- 동작 과정
  1. 피벗 선택 : 배열의 한 요소를 피벗으로 선택 → 단순화를 위해 첫 번째 요소를 피벗으로 선택
  2. 분할 : 피벗보다 작은 요소는 피벗의 왼쪽에, 큰 요소는 피벗의 오른쪽에 오도록 배열을 재정렬
  3. 재귀적 정렬 : 분할된 두 부분에 대해 재귀적으로 퀵 정렬을 수행

1) 피벗 선택 및 분할
피벗 69
→ {10,30,2,16,8,31,22}, 69, {없음}

2) 피벗 왼쪽 부분 정렬 **10,30,2,16,8,31,22**

피벗 10
→ {2, 8}, 10, {16, 30, 31, 22}

5) 피벗 오른쪽 부분 정렬(재귀적으로)

30, 31, 22

피벗 30
→ {22}, 30, {31}

2-1) 피벗 왼쪽 부분 정렬(재귀적으로) **2, 8**

피벗 2
→ 2, 8(이미 정렬됨)

2-2) 피벗 오른쪽 부분 정렬 **16, 30, 31, 22**

피벗 16
→ 16, {30, 31, 22}(변화 없음)

6) 피벗 왼쪽 부분 정렬(재귀적으로) **22**

→ 22(변화 없음)

7) 피벗 오른쪽 부분 정렬(재귀적으로) **31**

→ 31(변화 없음)

최종 배열 : 2, 8, 10, 16, 22, 30, 31, 69

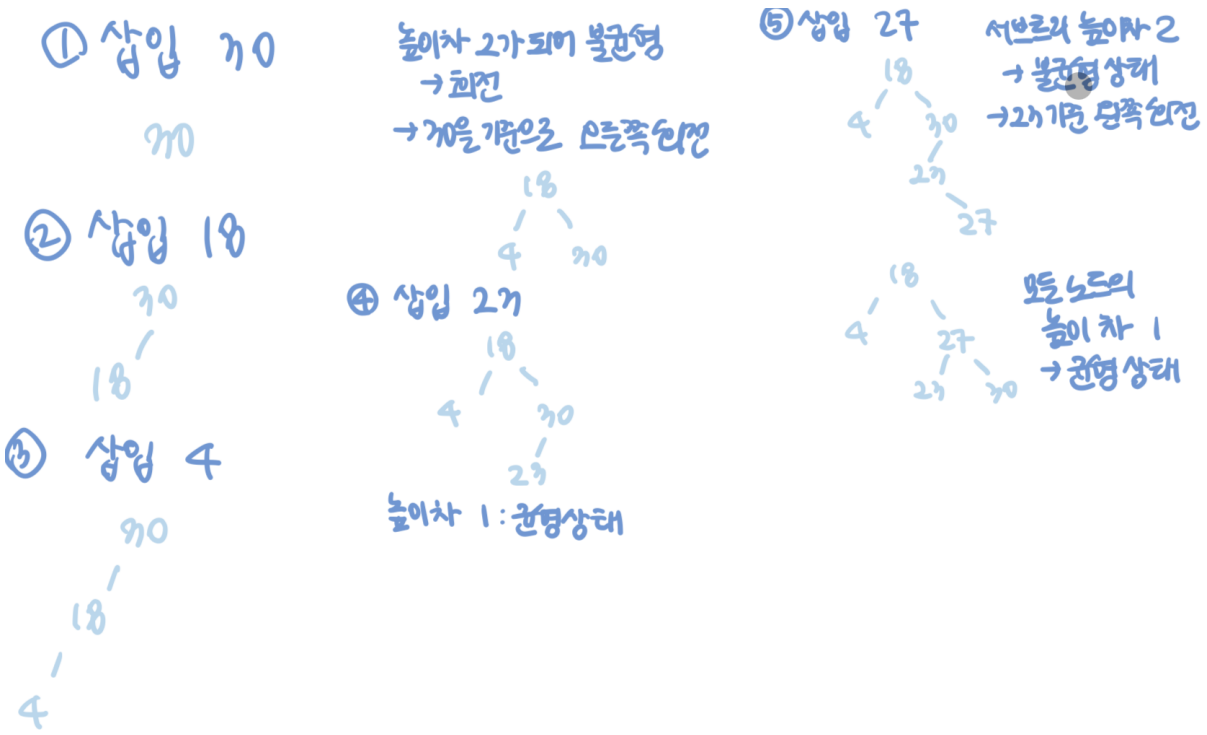
- 최선의 경우 복잡도
  - O(*n log n*). 이는 피벗이 배열을 반으로 나누는 경우
  - 분할이 항상 리스트의 가운데에서 이루어진다는 가정에서 n이 2의 거듭제곱이라고 가정하면 n개의 레코드를 가지는 리스트는 n/(2^k)의 크기로 나누어지고, 이 값이 1이 될때까지 나누어지므로
    k=log n개의 패스가 필요함. 이 과정이 n번 반복됨
- 최악의 경우 복잡도
  - 퀵 정렬 최악의 시간 복잡도는 O(*n^2*)임. 이는 피벗이 배열의 최소값이나 최대값으로 계속 선택되어 배열이 비균형하게 분할되는 경우
  - 예를 들어 이미 정렬된 배열이나 역순으로 정렬된 배열에서 첫번째 요소를 피벗으로 선택하는 경우에는 매 분할 단계마다 하나의 요소만 분할되고 나머지는 모두 한 쪽으로 몰리게 됨
  - 레코드의 수만큼 총 N번의 패스가 실행되고, 각 패스에서 n번 비교가 이루어지므로 n*n번 실행됨

5. **n개의 올림차순으로 정렬된 정수 배열에서 특정 정수값을 찾는 이진탐색 알고리즘을 구현하는 C언어 재귀 함수 binary_search의 바디를 작성해라. 탐색 성공의 경우 해당 정수값을 갖는 배열의 인덱스를, 실패할 경우 -1을 리턴함**

```
#include <stdio.h>
#define MAX_LENGTH 300

int Array[MAX_LENGTH] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 2

// 이진 탐색 함수 (재귀적 구현)
int binary_search(int key, int low, int high) {
```

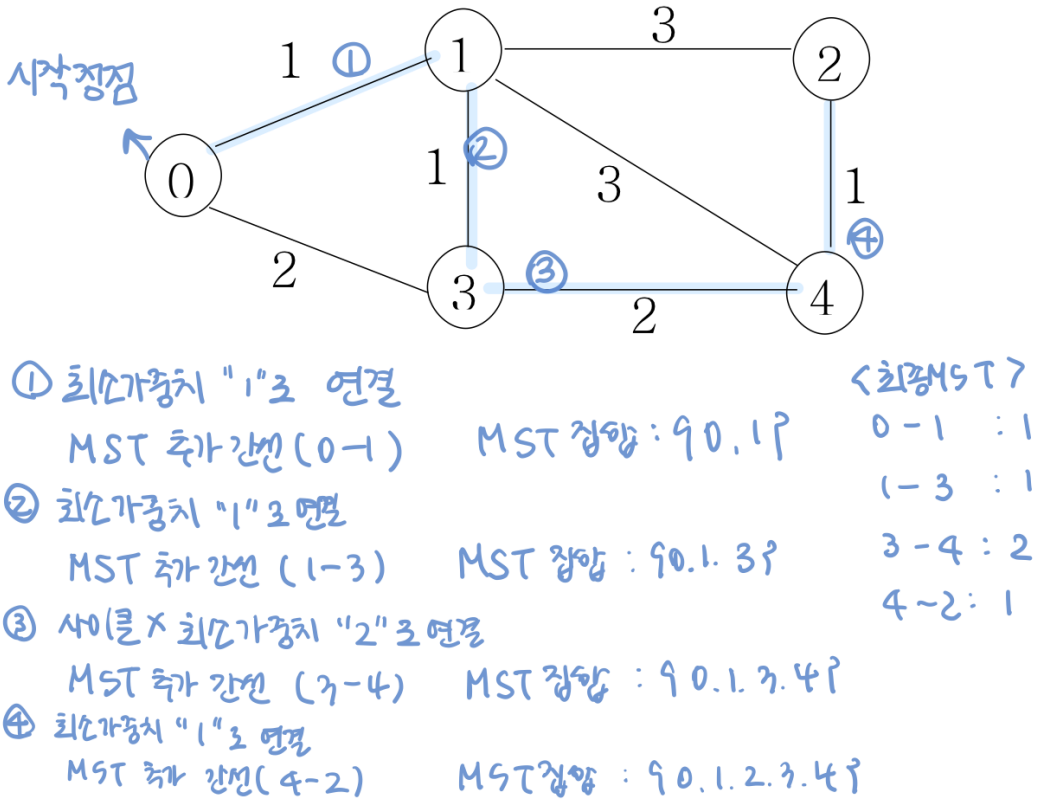
```
    if (low > high) {
        return -1; // 탐색 실패
    }

    int mid = low + (high - low) / 2; // 중간 인덱스 계산

    if (Array[mid] == key) {
        return mid; // 키 값을 찾은 경우
    } else if (Array[mid] > key) {
        return binary_search(key, low, mid - 1); // 왼쪽 반을 탐
    } else {
        return binary_search(key, mid + 1, high); // 오른쪽 반을
    }
}

int main() {
    int n = 15; // 배열의 실제 길이

    int key = 15; // 찾고자 하는 값
    int index = binary_search(key, 0, n - 1);

    if (index != -1) {
        printf("%d의 인덱스 : %d\n", key, index);
    } else {
        printf("%d를 찾을 수 없습니다.\n", key);
    }

    return 0;
}
```

6. **다음의 차례로 {30, 18, 4, 23, 27} 자료가 삽입이 될 경우 각 단계별로 AVL이 구성되는 각 과정을 그림으로 나타내시오**



3번에서 LL회전, 5번에서 RR회전하고 LL회전

7. **Minimum Spanning Tree MST란 무엇인지 설명하고, 다음의 그래프에 대해 1) MST가 구성되는 과정을 보여라. `Prim의 알고리즘` 을 사용할 경우 vertex 0을 시작점으로 해라 2) 적용한 알고리즘의 시간 복잡도를 Big O 표기식으로 나타내라**
    - Minimum Spanning Tree 정의
        - 가중 그래프에서 모든 정점을 연결하는 부분 그래프 중 가중치의 합이 최소인 트리
        - 모든 정점을 포함하며, 정점 간의 모든 간선이 최소 가중치를 가지고 사이클이 없음. 그래프의 정점의수를 v라 하면 정확히 v-1개의 간선을 가짐
    - Prim's Algorithm
        - 임의의 정점을 시작점으로 선택하여 MST 집합에 포함되지 않은 정점 중 현재 MST 집합과 최소 간선으로 연결된 정점을 선택
        - 선택된 정점 MST 집합에 추가하고, 해당 간선을 MST의 일부로 만듦.
        - 모든 정점이 MST 집합에 포함될 때까지 반복함



- 시간 복잡도 : O(n^2)

8. Minimum spanning Tree의 Krystal 알고리즘은 cycle 탐지를 위해 Union-Find알고리즘을 사용함. 다음과 같은 edge들로 구성된 그래프에서 Kruskal알고리즘을 사용할 때 array를 이용한 Union-Find 알고리즘 동작 과정을 array 값의 변화로 나타내라

edge 구성 노드	길이
(0,1)	1
(0,2)	2
(0,3)	2
(2,3)	3
(2,4)	6
(1,3)	7

array						
parent	−1	−1	−1	−1	−1
num	1	1	1	1	1

- Union-Find Algorithm
  - find : 특정 원소가 속한 집합을 찾는 연산
  - union : 두 집합을 합치는 연산
- 간선들의 가중치 오름차순 정렬과 초기 상태의 Union-Find 배열을 참고한다.

-1	0	-1	-1	-1
1	2	1	1	1

-1	0	0	-1	-1
1	3	1	1	1

-1	0	0	0	-1
1	4	1	1	1

2와 3은 이미 사이클로 연결됨

-1	0	~~0~~	~~0~~	-1
1	4	1	1	1

-1	0	0	0	0
1	5	1	1	1

1과 3은 이미 사이클로 연결됨

-1	~~0~~	0	~~0~~	0
1	5	1	1	1

- (0, 1), 가중치 1
- (0, 2), 가중치 2
- (0, 3), 가중치 2
- (2, 4), 가중치 6

Union-Find 배열의 최종 상태

- `parent = [1, -1, 1, 1, 1]`
- `num = [1, 5, 1, 1, 1]`

Kruskal's Algorithm 시간 복잡도

- 간선 정렬: O(ElogE)
- Union-Find 연산: 거의 O(1)

9. **해싱hashing에서 1) 해싱함수란 무엇인지, 2) 충돌 collision이 무엇이며 왜 발생하는지, 3) 충돌을 해결하는 방법들 중 체이닝 chaining이 무엇인지 기술해라**

   1) 해싱함수

   키 값에 대한 산술적 연산에 의해 테이블의 주소를 계산하여 항목에 접근함. 여기서 해시 테이블이 키 값의 연산에 의해 직접 접근이 가능한 구조를 말함

요약하자면, 탐색 키를 입력받아 해시 주소를 생성하고, 이 해시 주소가 배열로 구현된 해시 테이블의 인덱스가 된다.

2) 충돌이 무엇이고, 왜 발생하는가

충돌이랑 서로 다른 두 개 이상의 입력이 동일한 해시값을 생성할 때 발생하는 현상

- 해시 함수의 출력 크기 제한 : 해시 함수의 출력 크기가 제한되어 있으므로, 무한한 입력 공간에서 유한한 출력 공간으로 매핑할 때 반드시 일부 입력이 동일한 해시 값을 가질 수 밖에 없음 → 비둘기집 원리
- 해시 함수의 설계 문제 : 해시 함수가 입력의 다양성을 잘 반영하지 못하고 특정 패턴의 입력에서 동일한 해시값을 생성할 수 있음

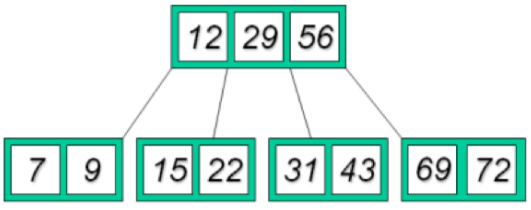
3) 체이닝이란

해시 테이블에서 충돌을 해결하는 한 가지 방법. 각 버켓에 삽입과 삭제가 용이한 연결리스트 할당

오버플로우 문제를 연결리스트로 해결함. 각 버켓에 고정된 슬롯이 할당되어 있지 않고, 각 버켓에 삽입과 삭제가 용이한 연결리스트를 할당하여 버켓 내에서는 연결리스트를 순차 탐색함

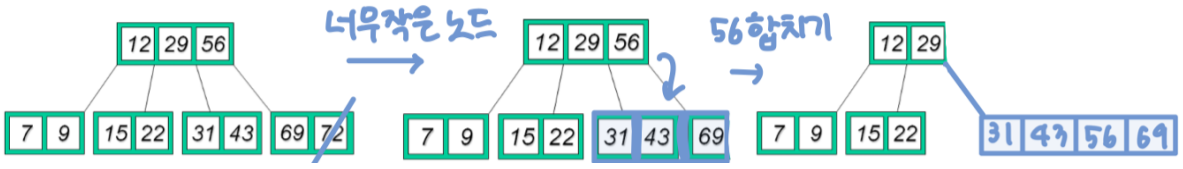
체이닝은 충돌이 발생할 때마다 연결리스트에 데이터를 추가하기 때문에, 이론적으로는 충돌이 많이 발생해도 데이터 검색이 가능함.

10. **다음의 5-ary B-tree에서 72를 제거했을 때의 결과 트리를 작성하고, 이유를 설명해라**

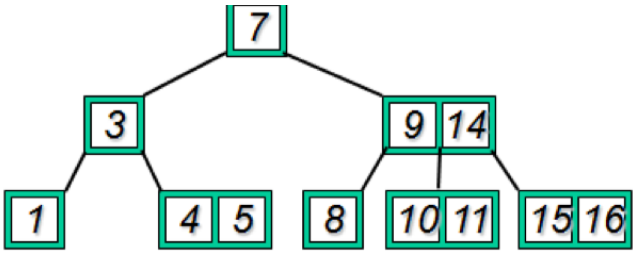


5-ary B-tree의 기본 개념

- 각 노드가 최대 4개의 키와 5개의 자식 노드를 가질 수 있는 트리
- 각 노드의 키는 정렬되어 있으며, 자식 노드는 각 키 사이의 범위를 나타냄
- 노드는 적어도⌈m/2⌉ - 1개의 키를 가져야 하며, 여기서 m은 노드의 차수임. 즉 이 경우는 각 노드가 적어도 2개의 키를 가져야함



11. **다음의 3-ary B-Tree에서 13이 입력될 경우 결과 B-Tree를 그리고 이유를 설명해라**



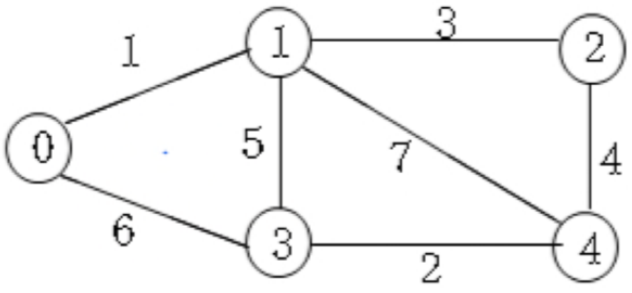
3-ary B-Tree는 각 노드가 최대 2개의 키와 3개의 자식 노드를 가질 수 있는 키. 적어도 1개의 키를 가져야함을 의미

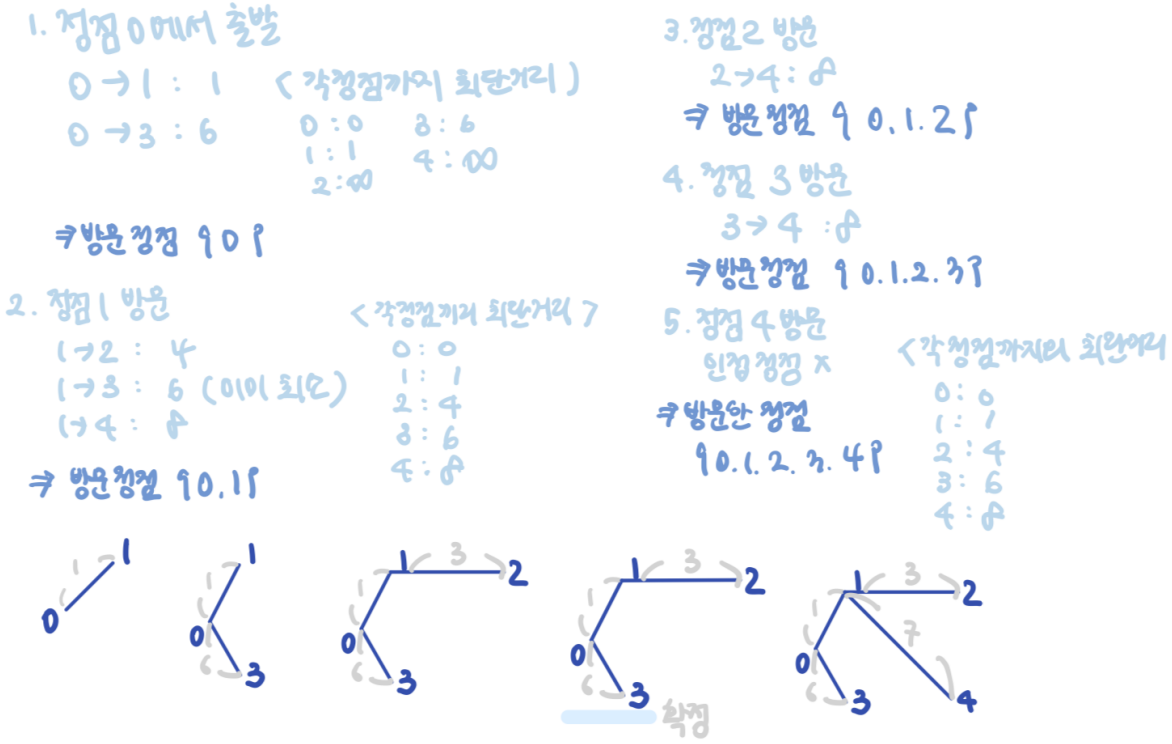
1. 13은 루트노드보다 크기때문에 오른쪽 서브트리로 이동
2. 13은 [9 | 14]노드의 중간이므로 그 중간인 [10 | 11]로 들어감
3. 근데 여기서 최대 키 수를 초과하므로 11을 부모 노드로 올림
4. 올리면 [9 | 11 | 14]노드로 재구성 되고, 트리의 균형이 유지됨

```
          [7 | 11]
        /      \
   [3]          [9 | 14]
  /   \        /   |   |   \
[1] [4 | 5] [8] [10] [13] [
```

노드는 3개의 키를 가지고 있지만 이는 분할 규칙에 의해 허용될 수 있음. 분할되지 않은 상태에서 최대 허용 키는 2이지만 분할 후 자식 노드가 균형을 유지하고 있어서 구조적으로 올바를까..?

12. **다음의 그래프에서 정점 0에서 각 정점으로 가는 최단거리를 구하는 Dijkstra알고리즘을 수행 단계 별로 나타내라**





2023 데이터구조 기말 시험 간략 내용

1. **Greedy 방법에 대해 공부해라**
    a. 각 단계에서 최선의 답을 선택하는 과정을 반복함으로써 최종적인 해답에 도달하는 주요 알고리즘 설계 기법
2. **MST의 크루스칼 알고리즘을 이해해라**
    a. kruskal : O(e*log(e))
    b. prim : O(V^2)
    c. dijkstra : O(n^2)
3. **AVL에 대해 공부해라**
    a. 각 노드에서 왼쪽 서브트리와 오른쪽 서브트리의 높이 차이가 1 이하인 이진 탐색 트리
    b. 시간 복잡도 : O(log n)
4. **우선순위큐에 대해 공부해라**

표현 방법	삽입	삭제
순서없는 배열	O(1)	O(n)
순서없는 연결 리스트	O(1)	O(n)
정렬된 배열	O(n)	O(1)
정렬된 연결 리스트	O(n)	O(1)
히프	O(logn)	O(logn)

5. **퀵정렬에 대해 공부해라**
6. **이진탐색트리의 특성에 대해 공부해라**
7. **B-Tree의 입력과 삭제에 대해 공부해라**
8. 최대 힙
    a. 시간 복잡도 : O(log n)→O(n log n)