

알고리즘

1. 다음 중 레드-블랙 트리(red-black tree)의 특성으로 옳은 것만을 모두 고르면?

ㄱ. 모든 노드는 레드 또는 블랙이다.
ㄴ. 루트 노드는 레드이다.
ㄷ. 레드 노드의 자식 노드는 항상 블랙이다.
ㄹ. 모든 리프 노드(NIL)는 레드이다.

- ① ㄱ, ㄴ
② ㄱ, ㄷ
③ ㄴ, ㄷ
④ ㄷ, ㄹ

2. 동적 계획법(dynamic programming)에 대한 설명으로 옳지 않은 것은?

- ① 부분 문제들 사이에는 의존적 관계가 없다.
② 동일한 부분 문제들을 다시 풀지 않도록 하기 위해 부분 문제들의 해를 저장한다.
③ 주어진 문제를 작은 문제로 나누어 해결하고 이를 이용하여 원래 문제의 해를 도출한다.
④ 이 기법을 적용할 수 있으려면 최적성의 원리(principle of optimality)가 성립하여야 한다.

3. 스택(stack)의 특징에 대한 설명으로 옳지 않은 것은?

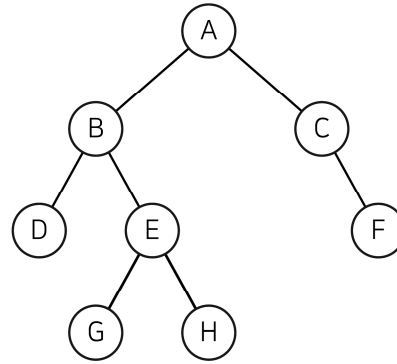
- ① 나중에 들어온 데이터가 먼저 나가는 후입 선출 구조(LIFO, Last-In First-Out)로 입출력이 일어난다.
② 제일 먼저 입력된 데이터가 맨 아래에 쌓이고 가장 최근에 입력된 데이터가 가장 위에 쌓이는 구조로 되어 있다.
③ 함수 호출에서 복귀주소를 기억하는 데 스택을 사용하고, 함수는 수행이 끝나면 최근에 자신을 호출한 함수로 돌아간다.
④ push 연산은 스택에서 하나의 요소를 제거하는 연산으로 top이 가리키는 요소를 스택에서 꺼내서 외부로 건네주는 연산이다.

4. 다음 중 안정적 정렬(stable sort) 알고리즘에 해당하는 것만을 모두 고르면?

ㄱ. 버블 정렬(bubble sort)
ㄴ. 힙 정렬(heap sort)
ㄷ. 삽입 정렬(insertion sort)
ㄹ. 계수 정렬(counting sort)

- ① ㄴ, ㄷ
② ㄴ, ㄹ
③ ㄱ, ㄴ, ㄷ
④ ㄱ, ㄷ, ㄹ

5. 다음 이진 트리(binary tree)의 단말 노드(leaf node) 개수와 간선(edge) 개수는?



	단말 노드 개수	간선 개수
①	3	7
②	3	8
③	4	7
④	4	8

6. 다음 설명에 해당하는 알고리즘 기법은?

○ 깊이 우선 탐색(DFS, Depth First Search)이 기본 골격이다.
○ 해를 찾는 도중에 해가 아니면 되돌아가서 다시 해를 탐색한다.
○ 최적화(optimization) 문제와 결정(decision) 문제를 해결할 수 있다.

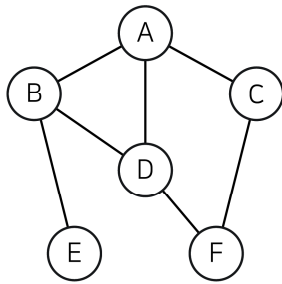
- ① 백 트래킹(back tracking)
② k-NN(k-Nearest Neighbors)
③ 그리디(greedy) 알고리즘
④ 분할 정복(divide-and-conquer) 알고리즘

7. 다음 중 문자열 매칭 알고리즘에 해당하는 것만을 모두 고르면?

- ㄱ. 벨만 포드(Bellman Ford)
 ㄴ. 보이어-무어(Boyer-Moore)
 ㄷ. 다익스트라(Dijkstra)
 ㄹ. KMP(Knuth, Morris, Pratt)

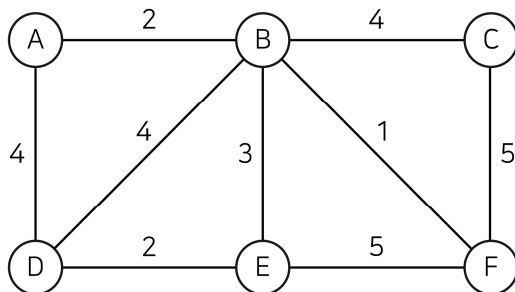
- ① ㄱ, ㄴ
 ② ㄱ, ㄷ
 ③ ㄴ, ㄷ
 ④ ㄴ, ㄹ

8. 다음 그래프에서 정점 A부터 깊이 우선 탐색을 수행할 때 가능한 정점들의 방문 순서로 옳은 것은?



- ① $A \rightarrow B \rightarrow E \rightarrow F \rightarrow D \rightarrow C$
 ② $A \rightarrow B \rightarrow D \rightarrow C \rightarrow E \rightarrow F$
 ③ $A \rightarrow D \rightarrow B \rightarrow E \rightarrow F \rightarrow C$
 ④ $A \rightarrow D \rightarrow F \rightarrow E \rightarrow B \rightarrow C$

9. 다음 그래프에 대해 정점 A를 시작으로 프림(Prim) 알고리즘을 수행했을 때, 생성된 최소 신장 트리의 간선 개수는?



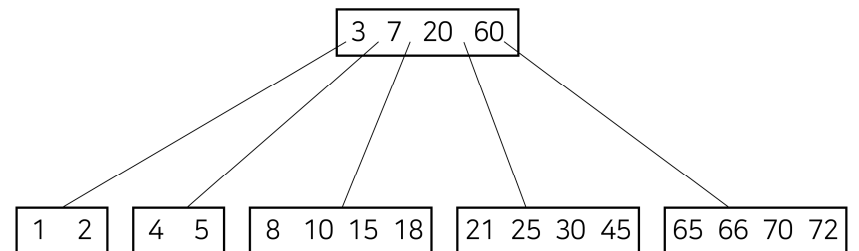
- ① 4개
 ② 5개
 ③ 6개
 ④ 7개

10. 다음 설명에 해당하는 정렬 알고리즘은?

- 배열을 정렬된 앞부분과 정렬되지 않은 뒷부분으로 나누어 처리한다.
 ○ 최선의 경우 시간 복잡도는 $O(n)$ 이며 최악의 경우 시간 복잡도는 $O(n^2)$ 이다.

- ① 삽입 정렬
 ② 버블 정렬
 ③ 병합(merge) 정렬
 ④ 선택(selection) 정렬

11. 다음 B-트리에 “50”을 삽입할 때의 설명으로 옳지 않은 것은? (단, 각 노드는 최대 4개의 키를 가질 수 있다)



- ① 50이 루트 노드가 된다.
 ② 모든 리프 노드의 레벨은 동일하다.
 ③ 50을 삽입하면 오버플로우가 발생한다.
 ④ 50을 삽입할 때 분할(split) 연산이 필요하다.

12. 다음 C언어 함수를 이용하여 $\text{rec}(5)$ 를 수행한 결과값은?

```
int rec(int n) {
    if (n <= 0)
        return 0;
    else if (n <= 3)
        return n;
    else
        return rec(n-4) + rec(n-3) + rec(n-2) + rec(n-1);
}
```

- ① 9
 ② 10
 ③ 12
 ④ 15

13. 배열 $A = [5, 2, 4, 6, 1, 3]$ 를 삽입 정렬로 오름차순 정렬할 때, 전체 과정에서 수행되는 key 값의 총 비교 횟수는? (단, 알고리즘은 다음과 같이 동작한다고 가정하며, 배열의 인덱스는 1부터 시작하고, n 값은 배열 A 의 원소의 개수이다)

```
for i ← 2 to n do
    key ← A[i]
    j ← i - 1
    while j ≥ 1 and A[j] > key do
        A[j+1] ← A[j]
        j ← j - 1
    A[j+1] ← key
```

- ① 12
- ② 13
- ③ 14
- ④ 15

14. 위상 정렬(topological sort)에 대한 설명으로 옳은 것은?

- ① 깊이 우선 탐색 방식을 이용해서는 구현할 수 없다.
- ② 하나의 그래프에 한 개의 위상 정렬만 존재한다.
- ③ 사이클이 없는 모든 종류의 그래프에 적용할 수 있다.
- ④ 간선 $\langle i, j \rangle$ 를 가질 때 정점 i 는 정점 j 를 선행한다.

15. 다음 파이썬 프로그램의 실행 결과는?

```
def search(arr, target):
    low, high = 0, len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target: return mid
        elif arr[mid] < target: low = mid + 1
        else: high = mid - 1
    return -1

if __name__ == "__main__":
    a = [2, 5, 8, 12, 16, 23, 38, 56, 91]
    print(search(a, 23))
```

- ① -1
- ② 4
- ③ 5
- ④ 6

16. 다음은 어떤 문자열에 포함된 문자들의 빈도수를 나타낸 표이다. 허프만(Huffman) 코드를 적용하여 인코딩한 결과의 비트 수는?

문자	a	b	c	d	e	f
빈도수	7	20	10	5	50	6

- ① 98
- ② 202
- ③ 300
- ④ 331

17. 기수 정렬(radix sort)에 대한 설명으로 옳은 것만을 모두 고르면?

- ㄱ. 비교 기반의 정렬 알고리즘이다.
- ㄴ. 저장공간이 추가로 필요하다.
- ㄷ. 안정적 정렬(stable sort) 알고리즘이다.
- ㄹ. 시간 복잡도는 $O(n\log n)$ 이다.

- ① ㄱ, ㄷ
- ② ㄴ, ㄷ
- ③ ㄴ, ㄹ
- ④ ㄷ, ㄹ

18. 다음은 0 이상의 모든 정수 n 에 대하여 $n!$ 을 구하는 파이썬 코드이다. (가)에 들어갈 코드로 옳지 않은 것은?

```
def factorial(n):
    if (가):
        return 1
    return n * factorial(n - 1)
```

- ① $n == 0$
- ② $n <= 0$
- ③ $n == 1$
- ④ $n <= 1$

19. 다음 파이썬 프로그램의 실행 결과는?

```
cnt = 0
class Node:
    def __init__(self, key, h, lf = None, rg = None):
        self.key = key
        self.h = h
        self.lf = lf
        self.rg = rg
class TREE:
    def __init__(self):
        self.r = None
    def h(self, n):
        if n == None:
            return 0
        return n.h
    def put(self, key):
        self.r = self.putitem(self.r, key)
    def putitem(self, n, key):
        if n == None:
            return Node(key, 1)
        if n.key > key:
            n.lf = self.putitem(n.lf, key)
        elif n.key < key:
            n.rg = self.putitem(n.rg, key)
        else:
            return n
        n.h = max(self.h(n.lf), self.h(n.rg)) + 1
        return self.bal(n)
    def bal(self, n):
        global cnt
        if self.bf(n) > 1:
            if self.bf(n.lf) < 0:
                n.lf = self.lf(n.lf)
                cnt += 1
            n = self.rg(n)
            cnt += 1
        elif self.bf(n) < -1:
            if self.bf(n.rg) > 0:
                n.rg = self.rg(n.rg)
                cnt += 1
            n = self.lf(n)
            cnt += 1
        return n
    def rg(self, j):
        i = j.lf
        j.lf = i.rg
        i.rg = j
        j.h = max(self.h(j.lf), self.h(j.rg)) + 1
        i.h = max(self.h(i.lf), self.h(i.rg)) + 1
        return i
```

```
def lf(self, j):
    i = j.rg
    j.rg = i.lf
    i.lf = j
    j.h = max(self.h(j.lf), self.h(j.rg)) + 1
    i.h = max(self.h(i.lf), self.h(i.rg)) + 1
    return i
def bf(self, n):
    return self.h(n.lf) - self.h(n.rg)
if __name__ == '__main__':
    t = TREE()
    value = [5, 22, 3, 14, 2, 19]
    for i in value:
        t.put(i)
    print(cnt)
```

- ① 1
- ② 2
- ③ 3
- ④ 4

20. 그래프의 임의의 한 점에서 출발하여 다른 모든 점을 한 번씩만 방문하고, 다시 시작점으로 돌아오는 경로 중 최단 경로를 찾는 문제는?

- ① 통 채우기(bin packing) 문제
- ② 정점 커버(vertex cover) 문제
- ③ 그래프 색칠하기(graph coloring) 문제
- ④ 여행자 문제(TSP, Traveling Salesman Problem)

21. 수행 시간 분석(run time analysis)에 관한 설명으로 옳지 않은 것은?

- ① 최악 경우(worst case) 분석은 상한의 의미가 있다.
- ② 최선 경우(best case) 분석은 수행 시간이 가장 적은 경우를 의미한다.
- ③ 평균 경우(average case) 분석은 모든 입력에 대한 수행 시간을 알고 계산하기 때문에 시간 복잡도의 척도로 많이 사용한다.
- ④ 상각(amortized) 분석은 일련의 연산을 수행하는 데 걸리는 총 시간을 연산 수로 나누어 계산한다.

22. 문제를 해결하기 위한 알고리즘 개발 단계를 순서대로 바르게 나열한 것은?

- (가) 알고리즘 구현
- (나) 알고리즘 설계
- (다) 알고리즘 분석
- (라) 알고리즘 정확성 검증

- ① (나) → (다) → (라) → (가)
- ② (나) → (라) → (다) → (가)
- ③ (다) → (나) → (라) → (가)
- ④ (다) → (라) → (나) → (가)

23. 그리디 알고리즘에 해당하는 것은?

- ① 크루스칼(Kruskal) 알고리즘
- ② 0/1 배낭(knapsack) 문제 풀이
- ③ 플로이드-워셜(Floyd-Warshall) 알고리즘
- ④ 최장 공통 부분 수열(LCS, Longest Common Subsequence) 구하기

24. 정렬 알고리즘에 대한 설명으로 옳은 것은?

- ① 힙 정렬의 시간 복잡도는 $O(n^2)$ 이다.
- ② 퀵 정렬은 정렬할 전체 원소에 대해 정렬을 수행한 후 기준값을 중심으로 왼쪽 부분집합과 오른쪽 부분집합으로 분할한다.
- ③ 삽입 정렬은 입력에 민감한 알고리즘으로, 입력이 거의 정렬되어 있을 때 처리시간이 느리고, 입력이 역으로 정렬되어 있을 때는 최선의 경우로 빠르다.
- ④ 병합 정렬은 크기가 n 인 입력을 $n/2$ 크기로 분할하고, 각각에 대해 같은 방식으로 정렬을 수행한 후, 2개의 각각 정렬된 부분을 합병한다.

25. 다음 파이썬 프로그램의 실행 결과는?

```
class NodeBS(object):
    def __init__(self, vl = None):
        self.vl = vl
        self.root = None
        self.lf = None
        self.rg = None
    def _ad(self, vl):
        nd = NodeBS(vl)
        if vl > self.vl:
            self.rg = self.rg and self.rg._ad(vl) or nd
        elif vl < self.vl:
            self.lf = self.lf and self.lf._ad(vl) or nd
        else:
            print("node duplicated")
        return self
class Bst(NodeBS):
    def add_node(self, vl):
        if not self.root:
            self.root = NodeBS(vl)
        else:
            self.root._ad(vl)
    def order(self):
        current = self.root
        nodes, stack = [], []
        while stack or current:
            if current:
                nodes.append(current.vl)
                stack.append(current)
                current = current.lf
            else:
                current = stack.pop()
                current = current.rg
        return nodes
if __name__ == "__main__":
    bt = Bst()
    in_list = [6, 4, 8, 2, 5, 7, 9, 1, 3, 10]
    for i in in_list:
        bt.add_node(i)
    print(bt.order())
```

- ① [6, 4, 2, 1, 3, 8, 5, 7, 9, 10]
- ② [6, 4, 2, 1, 5, 8, 3, 7, 9, 10]
- ③ [6, 4, 2, 1, 3, 5, 8, 7, 9, 10]
- ④ [6, 4, 2, 1, 5, 3, 8, 7, 9, 10]