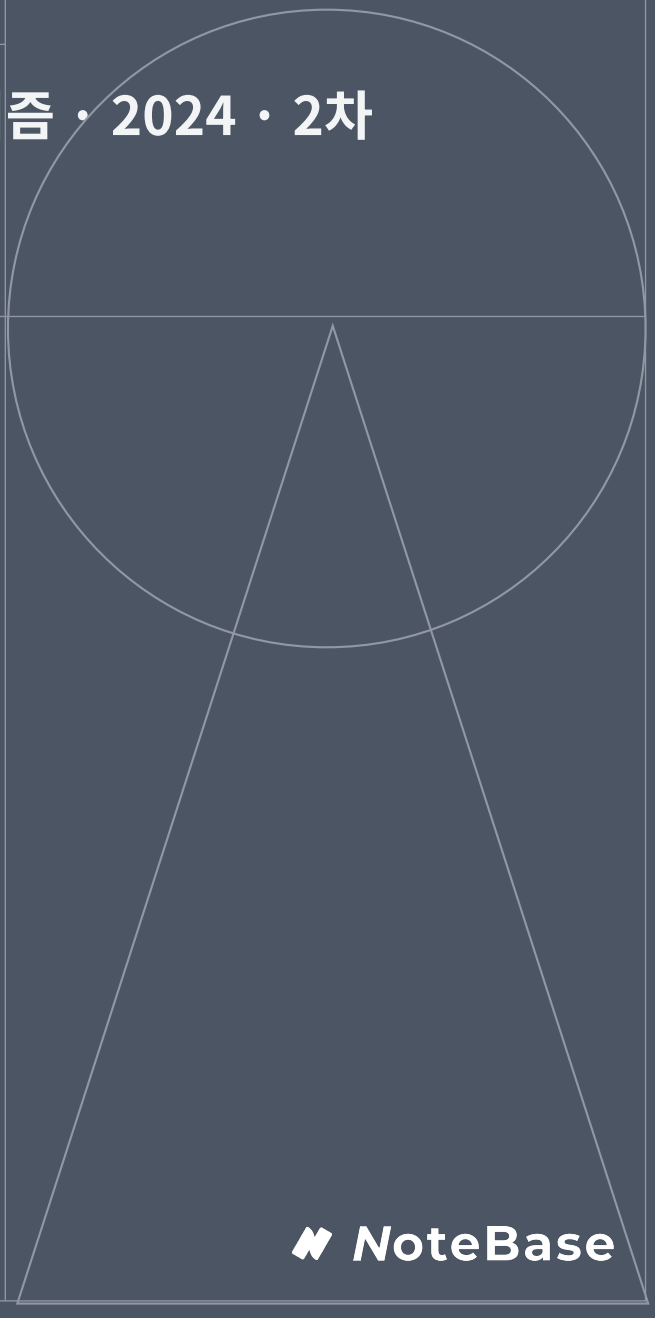
기출OX 해설집

공무원/국가직/7급 · 알고리즘 · 2024 · 2차



NoteBase

기출OX 해설집

공무원/국가직/7급 · 알고리즘 · 2024 · 2차

공무원/국가직/7급 · 알고리즘 · 2024 · 2차

문 1. 정점이 6개인 무방향 완전 그래프의 간선의 개수는?

① 6

② 12

③ 15

④ 21

정답 3번

출제 포인트

이 문제는 정점이 n개인 무방향 완전 그래프의 간선 수 공식을 적용해 값을 구하는 문제이다.

조건 정리

주어진 값: 정점 수는 6개이다.
구해야 할 값: 무방향 완전 그래프의 간선 수이다.
주의할 조건: 무방향 그래프이므로 정점 A-B와 B-A를 같은 간선 1개로 본다.

사용할 공식

무방향 완전 그래프 K_n의 간선 수는 $\frac{n(n-1)}{2}$이다.
이 공식을 쓰는 이유: 완전 그래프에서는 모든 서로 다른 두 정점이 정확히 1개의 간선으로 연결되므로, 간선 수는 정점 2개를 고르는 경우의 수와 같다.

계산 과정

1. 정점 수 $n = 6$을 공식에 대입한다.
2. 간선 수는 $\frac{6 \times 5}{2}$이다.
3. 계산하면 $\frac{30}{2} = 15$이므로 간선 수는 15개이다.

NoteBase

선지 판단

① X: 6은 정답이 아니다. 6은 정점 수만 그대로 본 값이며, 간선 수는 15개이다. 틀린 표현은 '간선의 개수는 6개이다'이고, 올바르게는 '간선의 개수는 15개이다'이다.
② X: 12는 정답이 아니다. 이는 일부 연결만 세거나 조합 계산을 잘못한 값이다. 틀린 표현은 '간선의 개수는 12개이다'이고, 올바르게는 '간선의 개수는 15개이다'이다.
③ O: $\frac{6 \times 5}{2} = 15$이므로 무방향 완전 그래프의 간선 수와 일치한다.
④ X: 21은 정답이 아니다. 이는 6까지의 합을 잘못 적용한 값에 가깝고, 완전 그래프의 간선 수 공식과 맞지 않는다. 틀린 표현은 '간선의 개수는 21개이다'이고, 올바르게는 '간선의 개수는 15개이다'이다.

오답 포인트

무방향 그래프인데 방향을 있는 것처럼 중복 계산하거나, 정점 수 자체를 간선 수로 착각하면 오답이 된다. 특히 $\frac{n(n-1)}{2}$에서 2로 나누는 이유를 놓치면 같은 정점 쌍을 두 번 세게 된다.

다시 나오면 이렇게 판단

완전 그래프 문제가 나오면 먼저 방향성 유무를 확인한 뒤, 무방향이면 $\frac{n(n-1)}{2}$, 방향이 있으면 $n(n-1)$을 적용해 판단한다.

암기 포인트

무방향 완전 그래프의 간선 수는 '정점 2개를 고르는 수'이므로 $\frac{n(n-1)}{2}$이다.

NoteBase

문 2. 큐에 다음 함수를 적용한 결과로 (가) ~ (다)에 들어갈 내용을 바르게 연결한 것은?

enqueue(e): 큐의 제 위치에 객체 e를 삽입한다.
dequeue(): 큐의 제 위치에서 객체를 삭제한다. 빈 큐일 경우에는 “error”를 출력한다.

연산순서	함수	출력	큐 저장공간
1	dequeue()	“error”	()
2	enqueue(1)		(1)
3	enqueue(2)		(1, 2)
4	enqueue(5)		(1, 2, 5)
5	dequeue()		(가)
6	enqueue(7)		(나)
7	dequeue()		(다)

① (가): (1, 2), (나): (1, 2, 7), (다): (1, 2)
② (가): (2, 5), (나): (2, 5, 7), (다): (5, 7)
③ (가): (2, 5), (나): (5, 7), (다): (7)
④ (가): (2, 5, 1), (나): (2, 5, 1, 7), (다): (5, 1, 7, 2)

정답 2번

출제 포인트

이 문제는 큐의 FIFO 규칙에 따라 enqueue와 dequeue를 순서대로 적용했을 때 저장공간이 어떻게 바뀌는지를 묻는 문제이다.

NoteBase

올바른 순서

1. dequeue() -> 빈 큐이므로 "error"를 출력하고 큐는 그대로 ()이다.
2. enqueue(1) -> 큐 뒤에 1이 들어가서 (1)이 된다.
3. enqueue(2) -> 큐 뒤에 2가 들어가서 (1, 2)가 된다.
4. enqueue(5) -> 큐 뒤에 5가 들어가서 (1, 2, 5)가 된다.
5. dequeue() -> 맨 앞의 1이 삭제되어 (가)는 (2, 5)가 된다.
6. enqueue(7) -> 큐 뒤에 7이 들어가서 (나)는 (2, 5, 7)이 된다.
7. dequeue() -> 맨 앞의 2가 삭제되어 (다)는 (5, 7)이 된다.

왜 이 순서: 큐는 먼저 들어간 데이터가 먼저 나오는 FIFO 구조이므로 dequeue는 항상 맨 앞 원소를 제거하고 enqueue는 항상 맨 뒤에 원소를 추가한다.

선지별 해설

1번: X. (가)를 (1, 2)로 둔 것이 틀리다. 5번째 연산의 dequeue()는 맨 앞 원소 1을 삭제하므로 (가)는 (2, 5)여야 한다. 틀린 표현은 (가)=(1, 2)이고, 올바르게는 (가)=(2, 5), (나)=(2, 5, 7), (다)=(5, 7)이다.

2번: O. 5번째 연산 뒤 (가)는 (2, 5), 6번째 연산 뒤 (나)는 (2, 5, 7), 7번째 연산 뒤 (다)는 (5, 7)으로 FIFO 규칙과 정확히 일치한다.

3번: X. (나)를 (5, 7)로 둔 것이 틀리다. 6번째 연산은 enqueue(7)이므로 기존 (2, 5) 뒤에 7만 추가되어 (나)는 (2, 5, 7)이어야 한다. 틀린 표현은 (나)=(5, 7)이고, 올바르게는 (나)=(2, 5, 7), (다)=(5, 7)이다.

4번: X. dequeue를 맨 앞 삭제가 아니라 회전처럼 처리한 것이 틀리다. 큐에서 삭제되는 값은 항상 제일 먼저 들어온 값이므로 5번째 연산 뒤에 (가)가 (2, 5, 1)이 될 수 없다. 틀린 표현은 (가)=(2, 5, 1), (나)=(2, 5, 1, 7), (다)=(5, 1, 7, 2)이고, 올바르게는 (가)=(2, 5), (나)=(2, 5, 7), (다)=(5, 7)이다.

오답 포인트

dequeue를 실행했을 때 맨 앞 원소가 삭제된다는 점을 놓치면 (가)에서 바로 틀린다. 또 enqueue는 기존 원소의 순서를 바꾸지 않고 맨 뒤에만 추가된다는 점을 잊으면 (나)와 (다)를 잘못 적게 된다.

다시 나오면 이렇게 판단

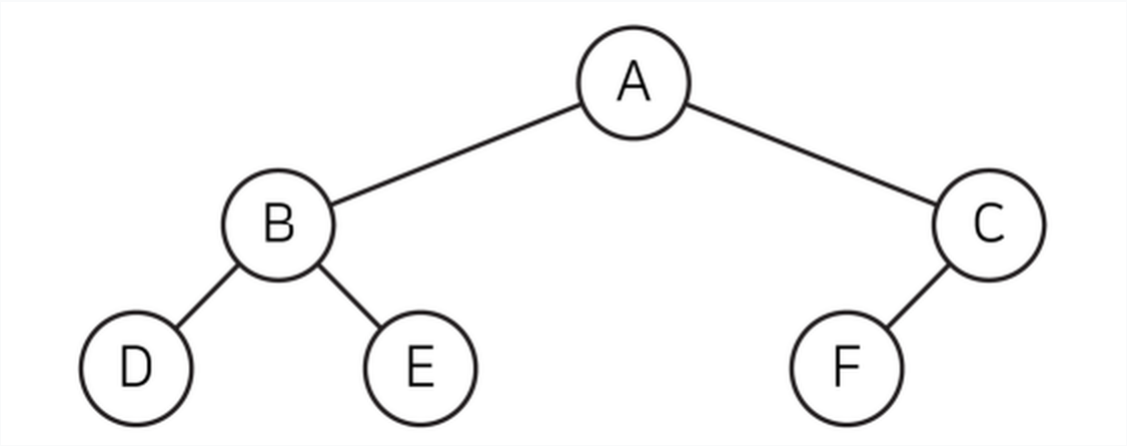
큐 문제는 현재 상태를 한 줄로 적어 두고, enqueue는 맨 뒤 추가, dequeue는 맨 앞 삭제라는 규칙만 기계적으로 적용하면 된다. 특히 dequeue가 나온 시점마다 제거되는 값이 가장 먼저 들어온 값인지 먼저 확인하면 된다.

암기 포인트

큐는 FIFO이므로 앞에서 삭제하고 뒤에 삽입하는 구조이다.

NoteBase

문 3. 다음 최대 힙에 대한 설명으로 옳지 않은 것은?



① 완전 이진 트리이어야 한다.

② 노드 A는 가장 큰 키 값을 가져야 한다.

③ 노드 B는 노드 F보다 큰 키 값을 가져야 한다.

④ 노드 B와 노드 D가 같은 키 값을 가질 수 있다.

정답 3번

출제 포인트

최대 힙의 성질이 모든 노드 사이의 전역 정렬이 아니라 부모와 자식 사이의 국소적 대소 관계만 보장한다는 점을 묻는 문제이다.

핵심 개념

최대 힙은 완전 이진 트리이면서 각 부모 노드의 키 값이 자식 노드의 키 값보다 크거나 같은 이진 트리이다.
루트는 전체에서 가장 큰 값을 가지지만, 서로 다른 서브트리에 있는 두 노드 사이의 대소관계는 일반적으로 정해지지 않는다.
또한 힙은 중복 키를 허용하므로 부모와 자식의 값이 같을 수도 있다.

NoteBase

선지별 해설

① O
최대 힙은 배열 기반 저장과 높이 균형을 위해 완전 이진 트리 형태를 가져야 한다. 따라서 이 설명은 맞다.

② O
최대 힙에서는 루트가 전체 노드 중 가장 큰 키 값을 가진다. 그림에서 루트는 A이므로 A가 가장 큰 값을 가져야 한다.

③ X
최대 힙은 부모와 자식 사이에서만 대소관계를 보장한다. B와 F는 조상-자식 관계가 아니므로 B가 F보다 커야 할 필요는 없다.
틀린 표현은 "노드 B는 노드 F보다 큰 키 값을 가져야 한다"이다.
올바르게는 "노드 B와 노드 F 사이의 대소관계는 최대 힙 조건만으로는 정해지지 않는다"이다.

④ O
최대 힙의 조건은 부모가 자식보다 크거나 같다는 것이다. 따라서 B와 D의 키 값이 같아도 힙 조건을 위반하지 않는다.

오답 포인트

최대 힙을 정렬된 트리처럼 오해하면 ③처럼 서로 다른 서브트리의 노드까지 모두 순서가 정해진다고 착각하기 쉽다. 힙은 전체 정렬 구조가 아니라 부모-자식 간 우선순위만 유지하는 구조이다.

다시 나오면 이렇게 판단

힙 문제에서는 먼저 완전 이진 트리 여부를 확인하고, 다음으로 부모와 자식 사이의 부등식만 적용한다. 비교 대상이 직접적인 부모-자식 관계가 아니면 힙 조건만으로는 대소를 단정할 수 없다.

암기 포인트

최대 힙은 "루트 최대, 부모 ≥ 자식, 형제·사촌끼리는 비교 불가"로 정리된다.

NoteBase

문 4. 그리디 알고리즘으로 최적의 해가 보장되지 않는 것은?

① 허프만(Huffman) 코딩

② 최소 신장 트리

③ 다익스트라(Dijkstra) 최단 경로

④ 0-1 배낭(knapsack) 문제

정답 4번

출제 포인트

이 문제는 그리디 알고리즘이 대표적으로 최적해를 보장하는 문제와 보장하지 않는 문제를 구분하는지를 묻는다.

핵심 개념

허프만 코딩: 문자 빈도를 기준으로 평균 코드 길이가 최소인 접두 부호를 만드는 그리디 알고리즘이다. 결정 차이축은 가장 작은 빈도 두 개를 반복 결합하는지 여부이다.
최소 신장 트리: 모든 정점을 최소 가중치로 연결하는 트리를 구하는 문제이다. 결정 차이축은 절단 속성을 만족하는 가장 유리한 간선을 선택해도 전역 최적이 유지되는지 여부이다.
다익스트라 최단 경로: 비음수 가중치 그래프에서 시작점으로부터의 최단 거리를 구하는 알고리즘이다. 결정 차이축은 현재 가장 짧은 거리의 정점을 확정해도 이후 더 짧아지지 않는지 여부이다.
0-1 배낭 문제: 물건을 쪼갤 수 없는 상태에서 제한 용량 안에 최대 가치를 담는 문제이다. 결정 차이축은 지역적으로 좋은 선택이 전체 최적을 항상 보장하는지 여부이다.

NoteBase

선지별 해설

① O
허프만 코딩은 매 단계에서 빈도가 가장 작은 두 노드를 합치는 그리디 선택으로 최적 접두 부호를 만든다. 따라서 평균 코드 길이 최소가 보장된다.

② O
최소 신장 트리는 Kruskal 알고리즘이나 Prim 알고리즘처럼 그리디 방식으로 구해도 최적해가 보장된다. 절단 속성 때문에 현재 선택한 최소 간선이 전체 최소 신장 트리의 일부가 될 수 있다.

③ O
다익스트라 알고리즘은 간선 가중치가 음수가 없을 때 현재 가장 가까운 정점을 확정하는 그리디 선택이 타당하다. 따라서 비음수 가중치 그래프에서는 최단 경로가 보장된다.

④ X
틀린 부분은 0-1 배낭 문제를 그리디로 풀면 최적해가 보장된다고 보는 점이다.
실제로는 물건을 분할할 수 없으므로 가치 밀도나 무게 기준의 지역 최적 선택이 전체 최적을 놓칠 수 있다.
올바르게는 분할 가능한 fractional knapsack 문제에서는 그리디가 최적이고, 0-1 배낭 문제는 동적 계획법으로 푸는 대표 문제이다.

오답 포인트

배낭 문제라는 이름만 보고 fractional knapsack과 0-1 knapsack을 같은 유형으로 묶으면 ④를 맞는 선택지처럼 보기 쉽다. 이 문항의 함정은 '물건을 쪼갤 수 있는가'라는 조건 차이를 숨긴 채 모두 그리디 대표 문제처럼 배열한 점이다.

다시 나오면 이렇게 판단

그리디 문항에서는 먼저 한 번 고른 선택을 되돌리지 않아도 되는 구조인지 확인한다. 허프만, 최소 신장 트리, 다익스트라는 그리디 선택 속성이 성립하지만, 0-1 배낭처럼 조합 전체를 비교해야 하는 문제는 그리디만으로 최적이 보장되지 않는다.

암기 포인트

허프만·MST·다익스트라는 그리디 최적, 0-1 배낭은 그리디 비보장이다.

NoteBase

문 5. 다음 C 코드에서 사용한 재귀 알고리즘의 시간 복잡도는? (단, n은 배열의 크기이다)

```
int S(int A[ ], int n){
   if( n <= 0 )  return 0;
   else return A[n-1] + S(A, n/2);
}
```

① $\Theta(\log n)$

② $\Theta(n)$

③ $\Theta(n\log n)$

④ $\Theta(n^2)$

정답 1번

출제 포인트

이 문제는 재귀 호출마다 입력 크기가 절반으로 줄어드는 점화식 $T(n) = T(n/2) + \Theta(1)$의 시간 복잡도를 판단하는 문제이다.

조건 정리

주어진 함수는 매 호출마다 두 번째 인자를 $n/2$로 줄여 다시 호출한다.
각 호출에서 수행하는 일은 종료 조건 비교, 배열 원소 1회 접근, 덧셈 1회이므로 $\Theta(1)$이다.
구해야 할 값은 이러한 상수 시간 호출이 몇 번 반복되는지에 따른 전체 수행 시간이다.

사용할 공식

점화식은 $T(n) = T(n/2) + \Theta(1)$이다.
이 공식을 쓰는 이유는 한 번 호출될 때마다 추가 작업은 상수 시간이고, 다음 재귀 호출의 문제 크기만 절반으로 감소하기 때문이다.

계산 과정

1. 처음 크기가 n이면 다음 호출의 크기는 $n/2$, 그다음은 $n/4$가 된다.
2. 이 과정은 $n/2^k \le 1$이 될 때까지 반복되므로 재귀 깊이는 $k = \Theta(\log n)$이다.
3. 각 단계 비용이 $\Theta(1)$이므로 전체 시간은 $\Theta(1) \times \Theta(\log n) = \Theta(\log n)$이다.

NoteBase

선지별 해설

① O
- 재귀 호출 깊이가 $\Theta(\log n)$이고 각 호출 비용이 상수이므로 전체 시간은 $\Theta(\log n)$이다.

② X
- $\Theta(n)$이 되려면 입력 크기가 1씩 줄어들거나 각 단계에서 선형 탐색을 해야 한다.
- 이 함수는 n을 절반으로 줄이므로 틀린 표현은 전체 시간이 $\Theta(n)$이라는 부분이고, 올바르게는 $\Theta(\log n)$이다.

③ X
- $\Theta(n \log n)$은 보통 깊이가 $\log n$이고 각 단계 비용이 $\Theta(n)$일 때 나온다.
- 이 함수는 각 단계 비용이 $\Theta(1)$이므로 틀린 표현은 단계당 선형 비용이 있다고 보는 부분이고, 올바르게는 $\Theta(\log n)$이다.

④ X
- $\Theta(n^2)$는 이중 반복이나 각 단계의 중첩 선형 작업이 있을 때 나타난다.
- 이 함수에는 그런 중첩이 없으므로 틀린 표현은 전체 시간이 제곱 시간이라는 부분이고, 올바르게는 $\Theta(\log n)$이다.

오답 포인트

배열 크기가 n이라는 이유만으로 전체 시간을 $\Theta(n)$으로 단정하는 실수가 자주 나온다. 이 문항의 핵심은 배열 전체를 도는지가 아니라 재귀마다 문제 크기가 얼마나 줄어드는지를 보는 데 있다.

다시 나오면 이렇게 판단

재귀 시간 복잡도는 먼저 한 호출에서 드는 비용을 본 뒤, 다음 호출에서 입력 크기가 얼마나 줄어드는지를 확인한다. 입력이 절반씩 줄고 단계당 비용이 상수이면 재귀 깊이만큼만 더해져 $\Theta(\log n)$으로 판단된다.

암기 포인트

$T(n) = T(n/b) + \Theta(1)$ 꼴은 재귀 깊이만 남으므로 시간 복잡도는 $\Theta(\log n)$이다.

NoteBase

문 6. 1부터 n까지 합을 구하는 다음 파이썬 프로그램의 빈칸에 들어갈 내용은? (단, n은 $n \geq 1$인 정수이다)

```
def sum(n):
    ret = 0
    for [          ]:
        ret += i
    return ret
```

① i in range(-1, n+1)

② i in range(0, n)

③ i in range(1, n)

④ i in range(1, n+1)

정답 4번

출제 포인트

이 문제는 파이썬의 range(start, stop)가 시작값은 포함하고 끝값은 포함하지 않는다는 규칙을 이용해 1부터 n까지 정확히 반복하는지를 묻는 문제이다.

핵심 개념

range(start, stop)는 start부터 stop 직전까지의 정수를 만든다.
1부터 n까지 모두 더하려면 반복값이 1, 2, ..., n이 되어야 하므로 시작값은 1이고 끝 경계는 n+1이어야 한다.

선지별 해설

1번 X: i in range(-1, n+1)은 -1, 0, 1, ..., n을 순회한다. 틀린 부분은 "-1"이다. 실제로는 1부터 시작해야 하므로 i in range(1, n+1)이어야 한다.
2번 X: i in range(0, n)은 0, 1, ..., n-1을 순회하므로 n이 빠진다. 틀린 부분은 끝값을 n으로 둔 점이다. 실제로는 n까지 포함되게 i in range(1, n+1)이어야 한다.
3번 X: i in range(1, n)은 1, 2, ..., n-1만 순회한다. 틀린 부분은 끝값을 n으로 둔 점이다. 실제로는 range의 끝값이 제외되므로 i in range(1, n+1)이어야 한다.
4번 O: i in range(1, n+1)은 1, 2, ..., n을 정확히 한 번씩 순회하므로 ret += i를 통해 1부터 n까지의 합을 올바르게 구한다.

NoteBase

오답 포인트

range의 끝값이 제외된다는 점을 놓치면 2번과 3번을 헷갈리기 쉽다. 또한 1번은 0이 더해져도 영향이 없다고 보고 지나치기 쉽지만, -1까지 포함되어 합이 1만큼 작아진다는 점이 함정이다.

다시 나오면 이렇게 판단

반복문 범위를 고르는 문제는 먼저 실제로 필요한 수열이 무엇인지 적은 뒤, range가 앞은 포함하고 뒤는 제외한다는 규칙에 맞춰 시작값과 마지막 값의 다음 수를 넣으면 된다.

암기 포인트

range는 앞을 포함하고 뒤를 제외하므로 1부터 n까지는 range(1, n+1)이다.

NoteBase

문 7. 연산 이론상의 문제 유형에 대한 설명으로 옳지 않은 것은?

① P는 다항식 시간에 해결할 수 있는 문제군이다.

② NP는 비결정론적 다항식 시간에 해결할 수 있는 문제군이다.

③ 어떤 NP-Complete 문제에 대해서 다항식 시간에 해결할 수 있는 알고리즘을 찾아내면, 다른 모든 NP-Complete 문제도 다항식 시간에 해를 구할 수 있다.

④ NP-Complete 문제들 중에는 NP에 속하지 않는 문제도 있다.

정답 4번

출제 포인트

이 문제는 P, NP, NP-Complete의 정의와 포함관계를 정확히 구별할 수 있는지를 묻는다.

핵심 개념

P는 결정론적 알고리즘으로 다항식 시간 안에 풀 수 있는 문제군이다. NP는 해가 주어졌을 때 그 해의 정당성을 다항식 시간 안에 검증할 수 있는 문제군이며, 비결정론적 튜링 기계로는 다항식 시간 안에 풀 수 있는 문제군으로도 본다. NP-Complete는 NP에 속하면서 동시에 NP-hard인 문제군이다.

선지별 해설

1번: O. P의 정의를 그대로 서술한 선택지이다. 다항식 시간에 해결 가능하다는 점이 핵심이다.
2번: O. NP는 비결정론적 다항식 시간에 해결 가능한 문제군이라는 설명이 맞다. 실무적으로는 다항식 시간 검증 가능 문제군으로 이해해도 같다.
3번: O. 어떤 NP-Complete 문제 하나라도 다항식 시간 알고리즘이 발견되면, 다른 모든 NP-Complete 문제는 그 문제로 다항식 시간 환원이 가능하므로 모두 다항식 시간에 풀 수 있다.
4번: X. 틀린 부분은 "NP에 속하지 않는"이다. NP-Complete는 정의상 반드시 NP에 속한다. 올바르게는 "NP-Complete 문제는 NP에 속하면서 NP-hard인 문제이다"라고 해야 한다.

오답 포인트

NP-Complete와 NP-hard를 혼동하면 4번에 속기 쉽다. NP-hard는 NP에 속하지 않을 수도 있지만, NP-Complete는 반드시 NP에 속한다. 또한 발문이 "옳지 않은 것"이므로 맞는 설명 셋을 먼저 지우고 남는 하나를 고르는 판단이 필요하다.

다시 나오면 이렇게 판단

복잡도 문제는 먼저 포함관계를 본다. P와 NP의 정의를 확인하고, NP-Complete는 "NP에 속함 + NP-hard"라는 두 조건을 동시에 만족하는지로 판단하면 된다.

NoteBase

암기 포인트

P는 빨리 푸는 문제, NP는 빨리 검증하는 문제, NP-Complete는 NP 안에 있으면서 가장 어려운 문제이다.

NoteBase

문 8. 다음 fib() 함수는 피보나치 수열을 계산하는 C 코드이다. fib(17)을 실행할 때 fib(8)과 fib(10)이 각각 호출되는 횟수를 바르게 연결한 것은?

```
int fib(int n){
      if(n==1 || n==2) return 1;
      else return (fib(n-1) + fib(n-2));
}
```

① fib(8): 52번, fib(10): 13번

② fib(8): 52번, fib(10): 21번

③ fib(8): 55번, fib(10): 13번

④ fib(8): 55번, fib(10): 21번

정답 4번

출제 포인트

이 문제는 단순 재귀 피보나치 호출에서 특정 함수가 몇 번 등장하는지 점화식으로 바꿔 계산할 수 있는지를 묻는 문제이다.

조건 정리

주어진 함수는 $n = 1$ 또는 $n = 2$이면 바로 종료되고, $n \geq 3$이면 `fib(n-1)`과 `fib(n-2)`를 각각 한 번씩 호출한다.
구해야 할 값은 `fib(17)`을 실행할 때 `fib(8)`과 `fib(10)`이 호출되는 횟수이다.
주의할 점은 피보나치 값 자체가 아니라 특정 호출이 재귀 트리에서 몇 번 나타나는지를 세어야 한다는 점이다.

사용할 공식

`fib(n)` 실행 중 `fib(k)`가 호출되는 횟수를 $C(n, k)$라 두면, $n > k$에서
$C(n, k) = C(n-1, k) + C(n-2, k), C(k, k) = 1$이다.
이 점화식이 피보나치와 같으므로 호출 횟수도 피보나치 수열을 따른다.
따라서 $C(n, k) = F_{n-k+1}$로 계산할 수 있다.

NoteBase

계산 과정

1. `fib(8)`의 호출 횟수는 $C(17, 8) = F_{17-8+1} = F_{10}$이다.
2. 피보나치 수열이 1,1,2,3,5,8,13,21,34,55이므로 $F_{10} = 55$이다.
3. `fib(10)`의 호출 횟수는 $C(17, 10) = F_{17-10+1} = F_8 = 21$이다.
4. 따라서 연결은 `fib(8)=55번`, `fib(10)=21번`이다.

선지 판단

1번: X. `fib(8)`을 52번으로 본 부분이 틀렸다. `fib(8)`은 $F_{10} = 55$번 호출되고, `fib(10)`은 13번이 아니라 $F_8 = 21$번 호출된다. 틀린 표현은 `fib(8)=52번, fib(10)=13번`이고, 올바르게는 `fib(8)=55번, fib(10)=21번`이다.
2번: X. `fib(10)=21번`은 맞지만 `fib(8)=52번`이 틀렸다. `fib(8)`은 $F_{10} = 55$번이다. 틀린 표현은 `fib(8)=52번`이고, 올바르게는 `fib(8)=55번`이다.
3번: X. `fib(8)=55번`은 맞지만 `fib(10)=13번`이 틀렸다. 13은 F_7에 해당하는 값이라 한 단계 적게 센 결과이다. 틀린 표현은 `fib(10)=13번`이고, 올바르게는 `fib(10)=21번`이다.
4번: O. `fib(8)=55번`, `fib(10)=21번`으로 두 값이 모두 맞다.

오답 포인트

호출 횟수를 피보나치 값과 같은 것으로 막연히 외우면 인덱스를 한 단계씩 밀려서 13이나 52 같은 오답을 고르기 쉽다. 이 문제의 함정은 `fib(n)`의 반환값이 아니라 재귀 트리 안에서 특정 `fib(k)` 노드가 몇 번 나타나는지를 세는 데 있다.

다시 나오면 이렇게 판단

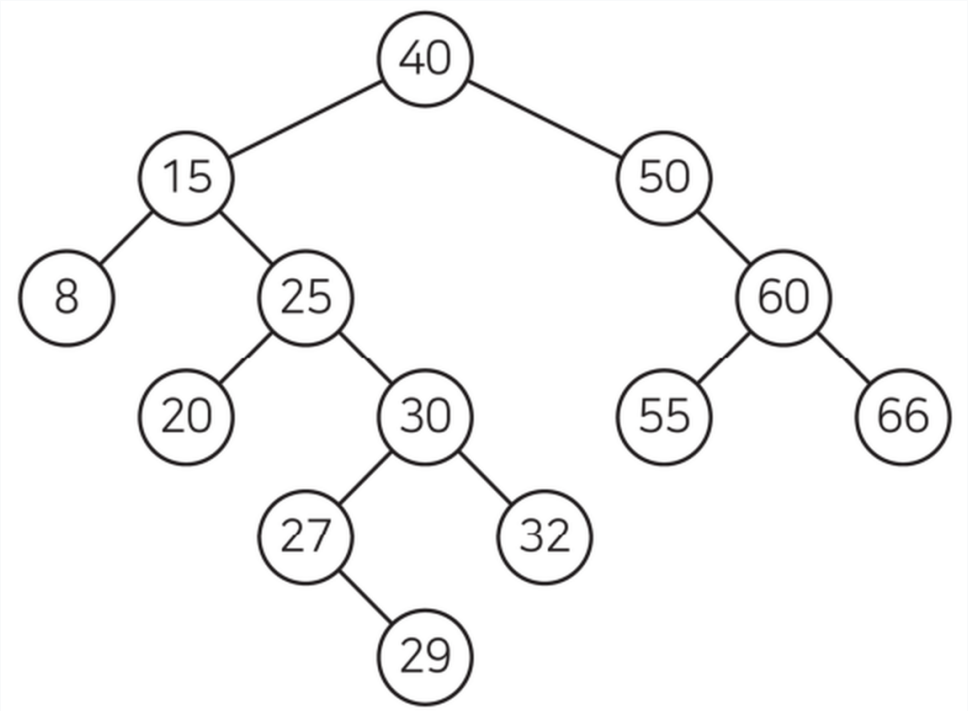
재귀 호출 횟수 문제는 `특정 호출의 횟수`를 새로운 변수로 두고, 부모 호출이 `n-1`, `n-2`로 갈라지는 구조를 그대로 점화식으로 옮겨 판단한다. 그다음 기준항 $C(k, k) = 1$을 세워 몇 번째 피보나치인지까지 맞춘다.

암기 포인트

단순 재귀 피보나치에서 특정 `fib(k)`의 호출 횟수는 `fib(n-k+1)` 꼴의 피보나치 인덱스로 바뀐다고 기억해 두면 된다.

NoteBase

문 9. 다음 이진 검색 트리에서 노드 이동 또는 포인터 변경의 횟수를 최소화하면서 노드를 삭제하는 과정의 설명으로 옳지 않은 것은?



① 노드 8을 삭제하려면 노드 15에서 노드 8을 가리키는 포인터를 NIL(또는 NULL)로 변경한다.

② 노드 25를 삭제하려면 그 자리에 노드 30을 놓는다.

③ 노드 50을 삭제하려면 노드 40이 노드 60을 직접 가리키도록 포인터를 변경한다.

④ 노드 60을 삭제하려면 그 자리에 노드 55 또는 노드 66을 놓는다.

정답 2번

출제 포인트

이 문제는 이진 검색 트리에서 삭제 대상 노드의 자식 수에 따라 가장 적은 포인터 변경으로 삭제하는 규칙을 도식에서 판단하는 문제이다.

NoteBase

선지별 해설

① O
노드 8은 리프 노드이므로 부모 15의 왼쪽 포인터만 NIL로 바꾸면 삭제가 끝난다.
틀린 표현은 없다.

② X
노드 25는 두 자식을 가지므로 그 자리에 바로 30을 두는 방식은 최소 변경이 아니다. 30을 올리면 30의 왼쪽 부분트리인 27, 29와 25의 왼쪽 자식 20의 연결을 다시 조정해야 한다.
틀린 표현은 "그 자리에 노드 30을 놓는다"이다.
올바르게는 중위 전임자 20 또는 중위 후속자 27을 올린 뒤, 그 노드를 원래 자리에서 삭제하는 방식이 최소 변경이다.

③ O
노드 50은 자식이 하나뿐이고 그 자식이 60이므로 부모 40이 60을 직접 가리키게 하면 된다.
틀린 표현은 없다.

④ O
노드 60은 두 자식 55와 66을 가지므로 삭제 시 중위 전임자 55 또는 중위 후속자 66으로 대체할 수 있다.
틀린 표현은 없다.

오답 포인트

자식이 둘인 노드 삭제에서 "오른쪽 자식"과 "중위 후속자"를 같은 것으로 착각하기 쉽다. 이 트리에서 25의 오른쪽 자식은 30이지만, 중위 후속자는 오른쪽 부분트리의 최솟값인 27이므로 30을 바로 올리는 설명은 최소 변경 규칙과 맞지 않는다.

다시 나오면 이렇게 판단

삭제할 노드의 자식 수를 먼저 센 뒤 판단한다. 자식이 0개이면 부모 포인터만 끊고, 1개이면 그 자식을 바로 승격시키며, 2개이면 중위 전임자 또는 중위 후속자처럼 실제 삭제가 쉬운 노드로 치환한다.

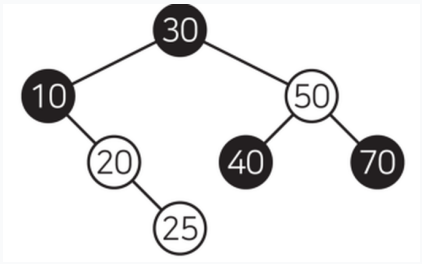
암기 포인트

BST 삭제는 자식 수로 나뉜다. 0개는 끊기, 1개는 승격, 2개는 전임자·후속자 치환이다.

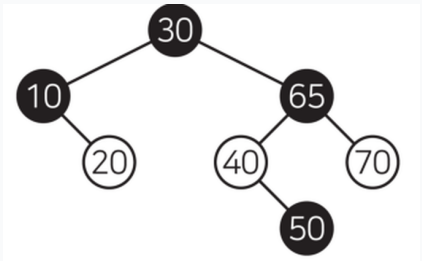
NoteBase

문 10. 레드-블랙 트리로 옳은 것은? (단, 외부 노드(leaf)는 표시하지 않고, 블랙 노드는 검정색으로, 레드 노드는 흰색으로 나타낸 것이다)

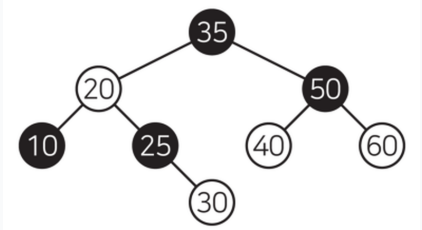
①



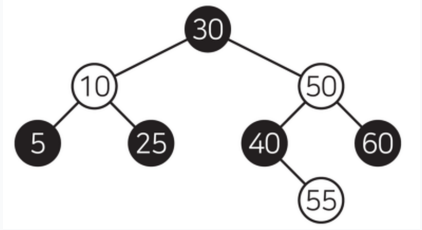
②



③



④



정답 3번

출제 포인트

이 문제는 각 도식이 이진 탐색 트리 순서를 유지하는지, 레드 노드의 자식 색 규칙을 지키는지, 외부 노드까지의 블랙 높이가 같은지를 함께 판별하는 문제이다.

NoteBase

선지별 해설

① X
- 틀린 부분은 레드인 20의 자식 25도 레드라는 점이다. 레드-블랙 트리에서는 레드 노드의 자식이 모두 블랙이어야 하므로 연속된 레드가 나오면 안 된다.
- 올바르게는 20 또는 25 중 적어도 하나가 블랙이어야 한다.

② X
- 틀린 부분은 30→65→40→50 경로만 블랙 노드 수가 하나 더 많다는 점이다. 30→65→70 경로와 비교하면 외부 노드까지의 블랙 높이가 같지 않다.
- 올바르게는 어떤 내부 노드에서 외부 노드로 내려가더라도 모든 경로의 블랙 노드 수가 같아야 한다.

③ O
- 루트 35는 블랙이고, 레드인 20·30·40·60의 자식은 모두 블랙 외부 노드 또는 블랙 내부 노드이다.
- 또한 35에서 외부 노드까지 가는 모든 경로의 블랙 노드 수가 같고, 키의 대소 관계도 이진 탐색 트리 순서를 만족하므로 레드-블랙 트리 조건을 만족한다.

④ X
- 색 규칙과 블랙 높이만 보면 맞아 보이지만, 틀린 부분은 55가 50의 왼쪽 서브트리에 들어가 있다는 점이다. 55는 50보다 크므로 50의 왼쪽에 올 수 없다.
- 올바르게는 55가 50의 오른쪽 서브트리에 있어야 이진 탐색 트리 순서를 만족한다.

오답 포인트

레드-블랙 트리 문제에서 레드-레드 금지만 보고 끝내면 ②와 ④를 놓치기 쉽다. ②는 블랙 높이 불일치가 함정이고, ④는 색 조건은 맞아도 이진 탐색 트리 순서 자체가 깨진 경우라는 점이 함정이다.

다시 나오면 이렇게 판단

루트가 블랙인지 확인한 뒤 레드 노드의 자식 색을 먼저 본다. 그다음 각 루트-외부 노드 경로의 블랙 수를 세고, 마지막으로 모든 노드가 이진 탐색 트리의 대소 관계를 지키는지 확인하면 된다.

암기 포인트

레드-블랙 트리는 레드 연속 금지, 블랙 높이 동일, BST 순서 유지의 세 축으로 판별한다.

NoteBase

문 11. 해시 테이블에 대한 설명으로 옳지 않은 것은?

① 충돌이 발생하지 않는다면 상수 시간 $O(1)$에 검색과 저장이 가능하다.

② 키 값을 입력으로 받아 해시 테이블상의 주소를 리턴하는 해시 함수는 나누기 방법뿐만 아니라 곱하기 방법으로도 만들 수 있다.

③ 저장 원소의 총수를 해시 테이블의 크기로 나눈 적재율(load factor)은 해시 테이블의 성능에 중요한 영향을 미친다.

④ 충돌을 해결하기 위한 별도 체이닝과 오픈 주소 방법은 모두 추가적인 저장 공간을 필요로 하지 않는다.

정답 4번

출제 포인트

이 문제는 해시 테이블의 기본 성질, 해시 함수의 방식, 적재율, 충돌 해결 방식의 차이를 구분할 수 있는지를 묻는다.

핵심 개념

해시 테이블은 키를 해시 함수로 배열 인덱스에 대응시켜 빠르게 탐색·삽입하는 자료구조이다.
해시 함수는 키를 테이블 주소로 바꾸는 함수이며 나누기 방법, 곱하기 방법 등이 있다.
적재율은 저장 원소 수를 테이블 크기로 나눈 값이며, 값이 커질수록 충돌 가능성이 커져 성능에 영향을 준다.
별도 체이닝은 같은 주소에 충돌한 원소를 연결 리스트 등의 추가 구조에 저장하는 방식이고, 오픈 주소는 테이블 내부의 빈 칸을 찾아 저장하는 방식이다.

선지별 해설

1번: O. 충돌이 전혀 없다면 원하는 주소로 바로 접근하므로 검색과 저장이 상수 시간 $O(1)$에 가능하다.
2번: O. 해시 함수는 나누기 방법만 있는 것이 아니고, 곱하기 방법처럼 다른 설계 방법으로도 만들 수 있다.
3번: O. 적재율은 원소 수 대비 테이블 크기를 나타내며, 적재율이 높아질수록 충돌이 늘어나 탐색·삽입 성능이 나빠질 수 있다.
4번: X. 틀린 표현은 "별도 체이닝과 오픈 주소 방법은 모두 추가적인 저장 공간을 필요로 하지 않는다"이다.
실제로는 오픈 주소는 테이블 내부에서 충돌을 해결하지만, 별도 체이닝은 연결 리스트나 버킷 같은 추가 저장 구조가 필요하다.

오답 포인트

4번은 충돌 해결 기법 두 개를 한 문장에 묶어 둘 다 같은 성질을 가진 것처럼 제시한 점이 함정이다. 오픈 주소는 외부 연결 구조가 없지만, 별도 체이닝은 충돌 원소를 따로 매달아 저장하므로 추가 공간 사용 여부에서 두 방식이 다르다.

NoteBase

다시 나오면 이렇게 판단

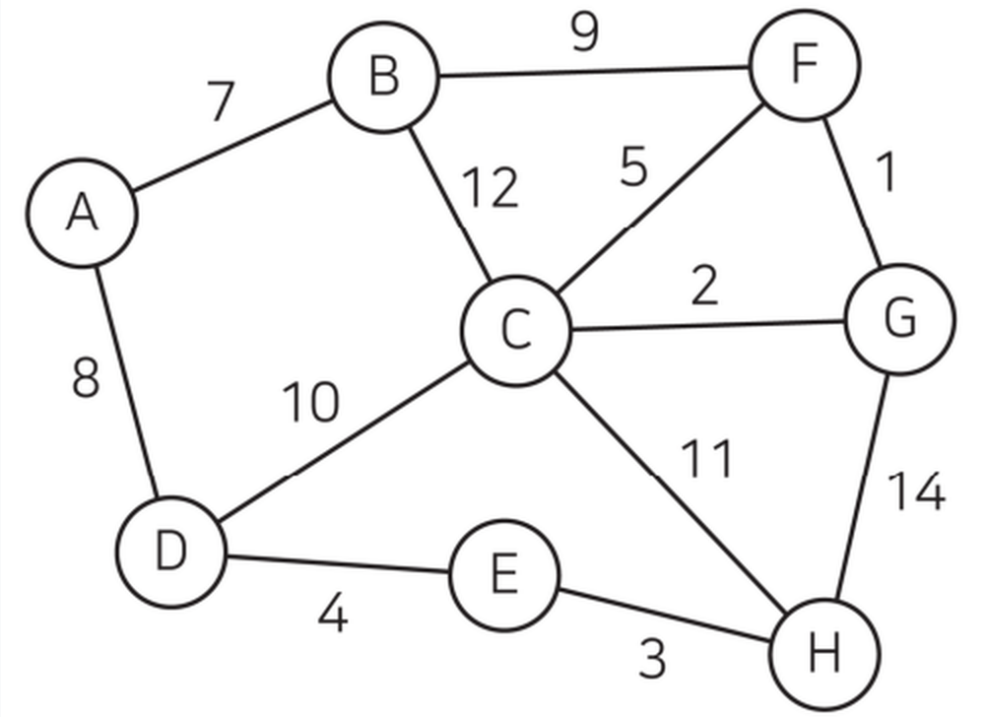
해시 문제에서는 먼저 해시 함수의 역할, 적재율의 의미, 충돌 해결 방식이 테이블 내부 처리인지 외부 연결 구조 사용인지로 나누어 판단하면 된다. 충돌 해결 방식이 둘 이상 함께 제시되면 공통점인지 차이점인지부터 확인하는 것이 안전하다.

암기 포인트

오픈 주소는 테이블 안에서 해결하고, 별도 체이닝은 테이블 밖 연결 구조를 붙여 해결한다.

NoteBase

문 12. 프림(Prim) 알고리즘을 적용하여 다음 그래프의 정점 A에서 시작하는 최소 신장 트리를 구할 때, 마지막으로 선택되는 정점은?



① C

② F

③ G

④ H

정답 1번

출제 포인트

이 문제는 가중치 그래프에서 현재 트리와 바깥 정점을 잇는 최소 간선을 반복 선택하는 프림 알고리즘의 편입 순서를 도식으로 추적하는 문제이다.

NoteBase

선지별 해설

① O
A에서 시작하면 선택 순서는 A-B(7), A-D(8), D-E(4), E-H(3), B-F(9), F-G(1), G-C(2)이다. 따라서 A 이후 마지막으로 트리에 편입되는 정점은 C이다.

② X
F는 B와 D, E, H가 이미 편입된 뒤 B-F(9)로 연결될 때 선택된다. 틀린 표현은 "마지막으로 선택되는 정점"이다. 올바르게는 "F는 A 이후 다섯 번째로 편입되는 정점"이다.

③ X
G는 F가 편입된 직후 frontier 간선 중 최소인 F-G(1)로 선택된다. 틀린 표현은 "마지막으로 선택되는 정점"이다. 올바르게는 "G는 A 이후 여섯 번째로 편입되는 정점"이다.

④ X
H는 D-E(4) 다음 단계에서 E-H(3)가 가장 작아 일찍 편입된다. 틀린 표현은 "마지막으로 선택되는 정점"이다. 올바르게는 "H는 A 이후 네 번째로 편입되는 정점"이다.

오답 포인트

프림 알고리즘을 전체 그래프의 최소 간선 순서로 오해하면 F-G(1), C-G(2), E-H(3)만 보고 G나 H를 먼저 고르기 쉽다. 프림은 항상 현재 트리에 이미 포함된 정점에서 밖으로 나가는 간선만 비교하므로, 각 단계의 frontier를 따로 봐야 한다.

다시 나오면 이렇게 판단

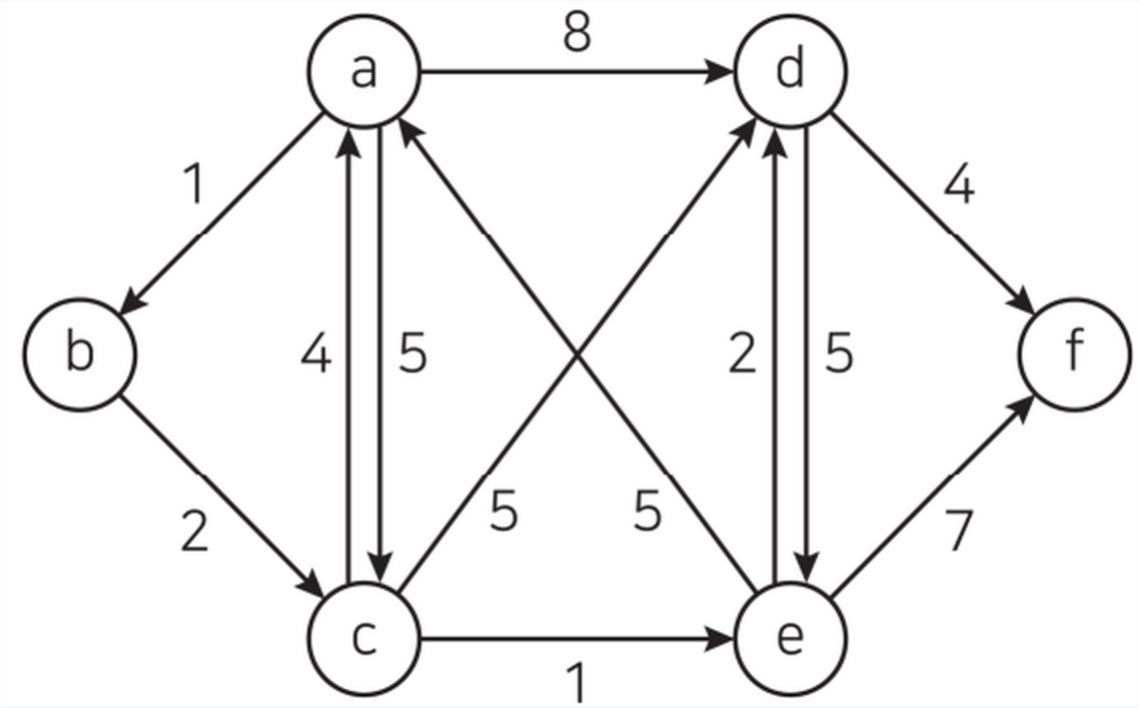
현재까지 포함된 정점 집합을 먼저 표시하고, 그 집합에서 바깥 정점으로 나가는 간선만 남긴 뒤 최소 가중치 간선을 한 단계씩 선택하면 편입 순서가 안정적으로 정리된다.

암기 포인트

프림은 전체 최소 간선이 아니라 현재 트리에서 뻗는 최소 간선을 고르는 알고리즘이다.

NoteBase

문 13. 다음 그래프에 다익스트라 알고리즘을 적용하여 구한 정점 a에서 정점 f까지 최단 경로의 거리는?



① 10

② 11

③ 12

④ 13

정답 1번

출제 포인트

이 문제는 방향 그래프에 다익스트라 알고리즘을 적용하여 시작 정점 *a*에서 정점 *f*까지의 최단 거리 값을 구하는 문제이다.

조건 정리

주어진 값: 각 간선의 방향과 가중치가 주어져 있다.
구해야 할 값: 정점 *a*에서 정점 *f*까지의 최단 경로 길이이다.
주의할 조건: 방향 그래프이므로 화살표 방향을 따라가야 하고, *d*와 *e* 사이처럼 양방향 가중치가 서로 다를 수 있다.

NoteBase

사용할 원리

모든 가중치가 양수이므로 다익스트라 알고리즘을 사용할 수 있다.
이 원리를 쓰는 이유: 시작 정점에서 가장 가까운 정점을 하나씩 확정하면서 인접 정점의 거리를 완화하면 최단 거리를 안정적으로 구할 수 있기 때문이다.

계산 과정

1. 초기값은 $d(a) = 0$이고, a에서 직접 가는 값으로 $d(b) = 1$, $d(c) = 5$, $d(d) = 8$로 둔다.
2. 가장 작은 b를 확정하면 $a \to b \to c$로 $d(c) = 1 + 2 = 3$이 되어 기존 5보다 작아진다.
3. 다음으로 c를 확정하면 $d(e) = 3 + 1 = 4$, $d(d) = \min(8, 3 + 5) = 8$이 된다.
4. 다음으로 e를 확정하면 $d(d) = \min(8, 4 + 2) = 6$, $d(f) = 4 + 7 = 11$이 된다.
5. 다음으로 d를 확정하면 $d(f) = \min(11, 6 + 4) = 10$이 된다.
6. 따라서 최단 경로는 $a \to b \to c \to e \to d \to f$이고 최단 거리는 10이다.

선지별 해설

① O
- 다익스트라 갱신 결과 $d(f) = 10$이다.
- 최단 경로는 $a \to b \to c \to e \to d \to f$이고 길이는 $1 + 2 + 1 + 2 + 4 = 10$이다.

② X
- 11은 $a \to b \to c \to e \to f$로 바로 계산할 때 나오는 값이다.
- 틀린 표현은 "최단 거리가 11"이라는 값이다. 올바르게는 $e \to f$의 7보다 $e \to d \to f$의 $2 + 4 = 6$이 더 짧으므로 최단 거리는 10이다.

③ X
- 12는 처음의 $a \to c = 5$를 그대로 고정하고 $a \to c \to e \to d \to f$로 계산할 때 나오는 값이다.
- 틀린 표현은 c까지의 거리를 5로 확정하는 판단이다. 올바르게는 $a \to b \to c$로 완화하면 $d(c) = 3$이 되어 전체 최단 거리는 10이다.

④ X
- 13은 $e \to d$의 가중치 2를 반대로 읽어 5로 쓰면 $1 + 2 + 1 + 5 + 4 = 13$이 되어 나오는 값이다.
- 틀린 표현은 $e \to d$를 5로 보는 것이다. 올바르게는 위쪽 화살표 $e \to d$의 가중치는 2이고, 5는 $d \to e$의 가중치이다.

오답 포인트

$a \to c = 5$를 먼저 보고 그대로 확정하거나, d와 e 사이의 방향별 가중치가 다르다는 점을 놓치면 11, 12, 13 같은 오답이 나온다. 다익스트라는 더 짧은 경로가 발견될 때마다 거리값을 다시 완화한다는 점이 핵심이다.

NoteBase

다시 나오면 이렇게 판단

시작 정점의 거리를 0으로 두고 인접 정점을 초기화한 뒤, 가장 작은 임시 거리 정점을 하나씩 확정하면서 그 정점에서 나가는 간선을 완화하면 된다. 각 단계에서 화살표 방향과 양방향 가중치의 차이를 함께 확인하면 실수를 줄일 수 있다.

암기 포인트

다익스트라는 "가장 작은 임시 거리 확정 -> 인접 정점 완화"를 반복하는 알고리즘이며, 방향 그래프에서는 같은 두 정점 사이여도 방향에 따라 가중치가 달라질 수 있다.

NoteBase

문 14. 다음 파이썬 프로그램의 실행 결과는?

```
def test(n):
    if(n<0):
        return 0
    return n+test(n-2)

print(test(5))
```

① 5
② 8
③ 9
④ 10

정답 3번

출제 포인트

이 문제는 재귀 함수의 종료 조건과 반환값 누적 과정을 따라가며 최종 출력값을 구하는 문제이다.

조건 정리

주어진 값: test(5)를 호출한다.
구해야 할 값: print(test(5))의 출력값이다.
주의할 조건: $n < 0$이면 0을 반환하고, 그렇지 않으면 $n + test(n - 2)$를 반환한다.

사용할 공식

재귀 호출 전개 원리는 n이 2씩 줄어 종료 조건에 도달할 때까지 현재 값을 누적하는 것이다.
이 원리를 쓰는 이유: 함수 본문이 $n + test(n - 2)$이므로 각 호출의 n이 그대로 합에 더해지기 때문이다.

NoteBase

계산 과정

1. test(5)는 $5 + test(3)$으로 전개된다.
2. test(3)은 $3 + test(1)$으로 전개된다.
3. test(1)은 $1 + test(-1)$으로 전개된다.
4. test(-1)은 $-1 < 0$이므로 0을 반환한다.
5. 따라서 test(1)=1, test(3)=4, test(5)=9가 된다.

선지별 해설

1번 5: X. 첫 호출의 값 5만 보고 재귀로 더해지는 3과 1을 누락한 값이다. 틀린 표현은 '출력값이 5이다'이고, 올바르게는 '출력값은 9이다'이다.
2번 8: X. 5 + 3까지만 더하고 test(1)=1을 빠뜨린 값이다. 틀린 표현은 '출력값이 8이다'이고, 올바르게는 '출력값은 5+3+1=9이다'이다.
3번 9: O. $test(5) = 5 + test(3) = 5 + 3 + test(1) = 5 + 3 + 1 + test(-1) = 5 + 3 + 1 + 0 = 9$이므로 맞는 선택지이다.
4번 10: X. 종료 조건에 도달하기 전의 누적 구조를 잘못 따라가 1을 한 번 더 더한 값으로 볼 수 있다. 틀린 표현은 '출력값이 10이다'이고, 올바르게는 '종료 조건에서 0을 반환하므로 최종 출력값은 9이다'이다.

오답 포인트

재귀 문제에서 가장 흔한 실수는 종료 조건에서 무엇을 반환하는지 확인하지 않고, 호출이 줄어드는 과정만 보고 중간 합을 임의로 정하는 것이다. 이 문항은 test(-1)이 0을 반환한다는 점과 각 단계에서 현재 n이 한 번씩만 더해진다는 점을 동시에 확인해야 한다.

다시 나오면 이렇게 판단

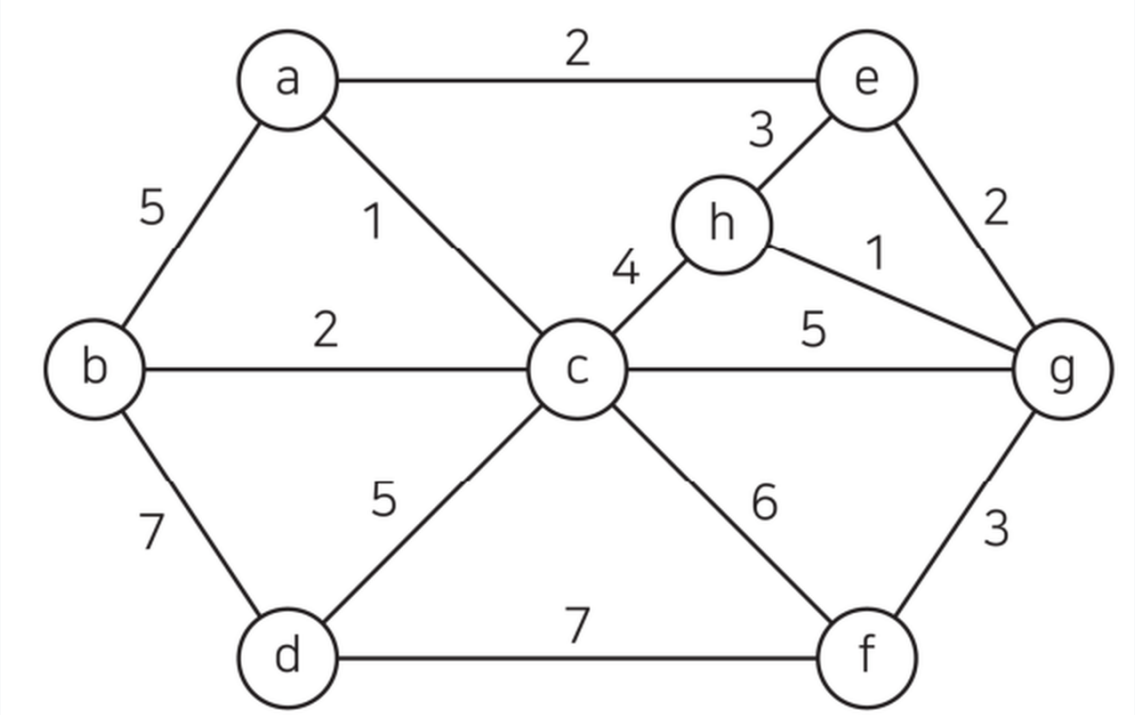
재귀 실행 결과 문제는 종료 조건을 먼저 확인한 뒤, 호출식을 한 단계씩 전개하여 바닥값에서부터 거꾸로 올라오며 합을 복원하면 안정적으로 판단할 수 있다.

암기 포인트

재귀값 계산은 종료 조건 확인 후 전개, 바닥값 확정 후 역순 누적이라는 흐름으로 정리된다.

NoteBase

문 15. 다음 그래프에 크루스칼(Kruskal) 알고리즘을 적용하여 최소 신장 트리를 만들 때, 마지막으로 최소 신장 트리에 포함되는 간선은? (단, (i, j)는 정점 i와 정점 j를 잇는 간선이다)



① (a, b)

② (a, c)

③ (b, c)

④ (c, d)

정답 4번

출제 포인트

이 문제는 가중치가 작은 간선부터 사이클이 생기지 않게 선택하는 크루스칼 알고리즘으로 최소 신장 트리를 구성할 때, 어느 간선이 마지막으로 채택되는지를 판단하는 문제이다.

NoteBase

선지별 해설

① X. (a, b)는 가중치 5이지만, 그보다 앞서 (a, c), (b, c), (a, e), (e, g), (h, g), (g, f)가 선택되면 a와 b는 이미 같은 연결 요소에 들어가므로 사이클을 만든다. 틀린 표현은 "마지막으로 포함된다"이고, 올바르게는 "사이클 때문에 포함되지 않는다"이다.
② X. (a, c)는 가중치 1이므로 가장 먼저 선택되는 간선 묶음에 속한다. 틀린 표현은 "마지막으로 포함된다"이고, 올바르게는 "초기에 바로 포함된다"이다.
③ X. (b, c)는 가중치 2이므로 (a, c) 다음 단계에서 b를 기존 트리에 붙이는 간선이다. 틀린 표현은 "마지막으로 포함된다"이고, 올바르게는 "초반에 포함되어 b를 연결한다"이다.
④ O. (c, d)는 가중치 5이며, 앞선 선택으로 d만 남은 상태에서 d를 기존 연결 요소에 붙이는 가장 작은 간선이다. 따라서 이 간선이 최소 신장 트리를 완성하는 마지막 포함 간선이다.

오답 포인트

가중치가 큰 간선이 항상 마지막에 들어간다고 보면 틀리기 쉽다. 크루스칼은 현재 남아 있는 연결 요소를 가장 작은 비용으로 이어 주는 간선을 택하므로, 마지막 간선은 남은 정점을 연결하는 최소 간선인지로 판단해야 한다. 이 문제에서는 d가 끝까지 분리되어 있고 d에 연결된 간선들 중 (c, d)=5, (b, d)=7, (d, f)=7이므로 (c, d)가 마지막 선택이 된다.

다시 나오면 이렇게 판단

가중치를 오름차순으로 나열한 뒤, 선택된 간선으로 연결 요소를 갱신하면서 사이클이 생기면 버린다. 거의 완성된 시점에 남아 있는 정점이나 연결 요소가 무엇인지 확인하면 마지막 포함 간선을 빠르게 찾을 수 있다.

암기 포인트

크루스칼의 핵심은 가중치 오름차순으로 보되 사이클이면 제외한다는 점이며, 마지막 간선은 남은 연결 요소를 잇는 최소 간선이다.

NoteBase

문 16. 주어진 동전의 집합에 대하여 가장 적은 개수의 동전으로 거스름돈을 만드는 문제를 그리디 알고리즘을 이용해서 해결할 수 있는 경우는? (단, 거스름돈은 10의 배수이다)

① 10, 50, 60, 200

② 10, 40, 50, 100

③ 10, 50, 100, 130

④ 10, 50, 100, 200

정답 4번

출제 포인트

거스름돈 문제에서 그리디가 성립하려면 큰 동전 하나를 작은 동전들로 바꾸었을 때 동전 수가 더 줄어드는 반례가 없어야 한다는 점을 묻는 문제이다.

핵심 개념

그리디 거스름돈 알고리즘은 남은 금액에서 가능한 가장 큰 동전을 먼저 선택하는 방식이다.
이 방식이 항상 맞으려면 큰 동전 1개가 작은 동전 여러 개보다 동전 수 측면에서 결코 불리하지 않아야 한다.
즉 특정 금액에서 큰 동전을 먼저 고르면 오히려 동전 수가 늘어나는 반례가 생기면 그 동전 체계에는 그리디를 그대로 적용할 수 없다.

NoteBase

선지별 해설

① X
10, 50, 60, 200 체계에서는 100원을 만들 때 그리디는 60원을 먼저 고른 뒤 10원 4개를 더해 총 5개를 사용한다.
하지만 최적해는 50원 2개로 총 2개이다.
틀린 부분은 "그리디로 해결할 수 있다"는 판단이다.
올바르게는 이 동전 집합에서는 100원 같은 반례가 있으므로 그리디가 항상 최적해를 보장하지 않는다.

② X
10, 40, 50, 100 체계에서는 80원을 만들 때 그리디는 50원을 먼저 고른 뒤 10원 3개를 더해 총 4개를 사용한다.
하지만 최적해는 40원 2개로 총 2개이다.
틀린 부분은 "그리디로 해결할 수 있다"는 판단이다.
올바르게는 이 동전 집합에서는 80원에서 반례가 생기므로 그리디가 깨진다.

③ X
10, 50, 100, 130 체계에서는 150원을 만들 때 그리디는 130원을 먼저 고른 뒤 10원 2개를 더해 총 3개를 사용한다.
하지만 최적해는 100원 1개와 50원 1개로 총 2개이다.
틀린 부분은 "그리디로 해결할 수 있다"는 판단이다.
올바르게는 이 동전 집합에서는 150원에서 반례가 생기므로 그리디가 항상 맞지 않는다.

④ O
10, 50, 100, 200 체계에서는 큰 동전 하나를 작은 동전들로 바꾸어도 동전 수가 줄지 않는다.
50원은 10원 5개보다 적은 개수이고, 100원은 50원 2개와 같거나 그보다 유리하며, 200원은 100원 2개와 같거나 그보다 유리하다.
따라서 가능한 한 200원, 100원, 50원을 먼저 쓰는 선택이 동전 수를 손해 보지 않으므로 모든 10의 배수 거스름돈에서 그리디가 최적해와 일치한다.

오답 포인트

동전 값이 단순히 오름차순으로 정렬되어 있거나 큰 단위가 존재한다는 이유만으로 그리디가 성립한다고 보면 오답이 된다. 이 문제는 100원, 80원, 150원처럼 작은 반례 금액 하나만 찾아도 해당 선지가 바로 탈락한다는 점이 함정이다.

다시 나오면 이렇게 판단

각 선지마다 큰 동전을 먼저 골랐을 때 애매해지는 가장 작은 금액을 찾아 본다. 그 금액에서 큰 동전 1개를 쓰는 경우와 그보다 작은 동전 조합을 비교하여 더 적은 개수의 조합이 나오면 그 동전 체계에서는 그리디가 성립하지 않는다.

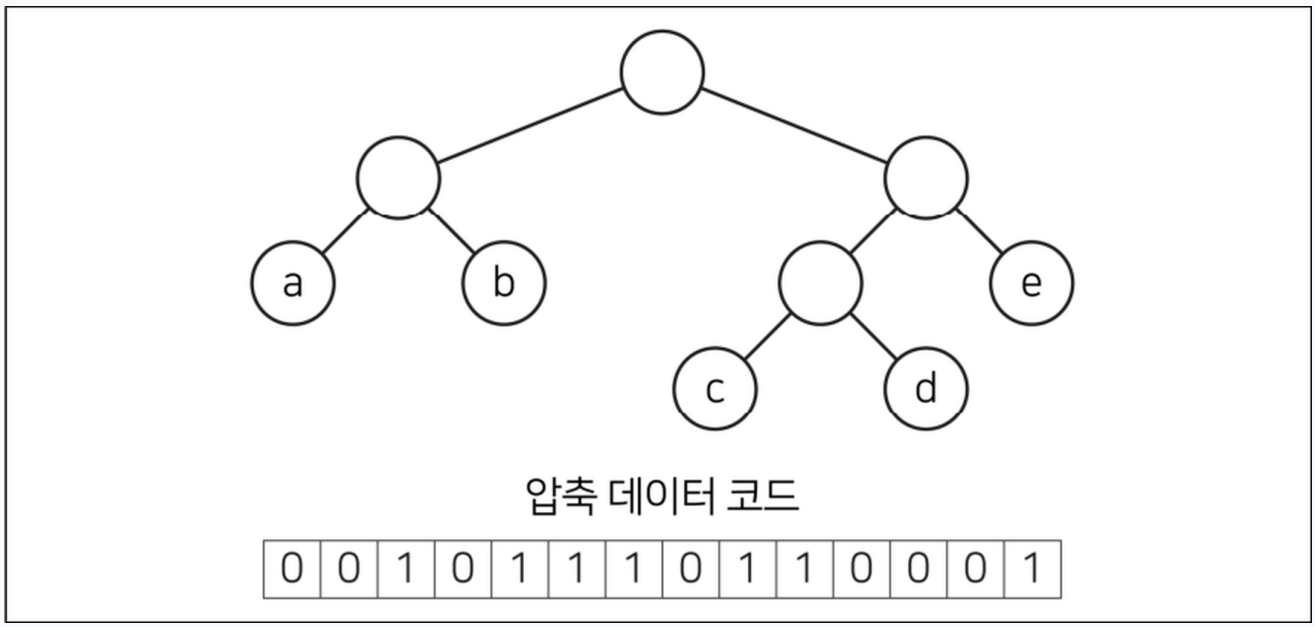
NoteBase

암기 포인트

거스름돈 그리디는 큰 동전 1개를 작은 동전들로 바꿨을 때 동전 수가 줄어드는 반례가 없을 때만 안전하다.

NoteBase

문 17. 다음 그림의 허프만 트리를 이용하여 인코딩한 일부 데이터의 코드를 문자열로 디코딩한 것은? (단, 노드의 왼쪽 자식 연결 간선에는 0, 오른쪽 자식 연결 간선에는 1이 부여된다)



① abbedcb

② abcde

③ adebcb

④ adebd

정답 3번

출제 포인트

이 문제는 허프만 트리에서 왼쪽 간선을 0, 오른쪽 간선을 1로 읽어 각 문자 코드를 만든 뒤, 압축 비트를 앞에서부터 끊김 없이 복호화할 수 있는지를 묻는다.

NoteBase

선지별 해설

1번: X. 허프만 코드는 접두어가 겹치지 않으므로 앞에서부터 유일하게 끊어 읽어야 한다. 이 트리의 코드는 a=00, b=01, c=100, d=101, e=11이다. 압축 데이터 00101110110001은 00|101|11|01|100|01로 끊겨 adebcb가 된다. 따라서 abbedcb는 두 번째 코드가 bb로 이어져 비트열과 맞지 않는다. 틀린 표현은 "abbedcb"이고, 올바르게는 "adebcb"이다.
2번: X. abcde가 되려면 비트열이 00|01|100|101|11 순서여야 하므로 0001100101111이 되어야 한다. 제시된 비트열 00101110110001과 길이와 순서가 모두 다르다. 틀린 표현은 "abcde"이고, 올바르게는 "adebcb"이다.
3번: O. a=00, d=101, e=11, b=01, c=100, b=01로 차례대로 읽으면 정확히 001011101101001이 된다. 각 문자가 트리의 리프에서 끝나므로 끊는 위치도 유일하다.
4번: X. adebd가 되려면 00|101|11|01|101이어서 비트열이 0010111011101이 된다. 제시된 비트열 끝부분은 10001이므로 c=100과 b=01이 이어진 구조이지 d=101로 끝나지 않는다. 틀린 표현은 "adebd"이고, 올바르게는 "adebcb"이다.

오답 포인트

허프만 문제에서 가장 흔한 실수는 문자를 먼저 추측하고 비트를 억지로 맞추는 것이다. 이 문항은 오른쪽 부분트리에서 e=11, c=100, d=101을 헷갈리거나, 마지막 10001을 c+b가 아니라 d로 잘못 읽는 함정을 두고 있다.

다시 나오면 이렇게 판단

루트에서 리프까지 내려가며 각 문자의 코드를 먼저 표로 정리한 뒤, 압축 비트를 왼쪽부터 읽다가 리프에 도달하는 순간마다 바로 한 문자를 확정하면 된다. 허프만 코드는 접두어가 겹치지 않으므로 이렇게 자르면 복호화 결과가 하나로 정해진다.

암기 포인트

허프만 복호화는 코드표를 먼저 만들고, 비트열은 리프에서 끊는 방식으로 읽는다.

NoteBase

문 18. 알파벳 순으로 정렬된 다음 배열에 대하여 이진 탐색으로 원소 P를 찾으려고 할 때, 비교해야 할 원소의 개수는?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
C	E	G	H	J	L	M	N	P	Q	S	T	V	W	Z

① 2

② 3

③ 4

④ 5

정답 3번

출제 포인트

이 문제는 정렬된 배열에서 이진 탐색을 수행할 때 가운데 원소와 실제로 몇 번 비교하는지를 단계적으로 세는 문제이다.

조건 정리

주어진 배열은 인덱스 0부터 14까지의 15개 원소로 정렬되어 있고 찾을 값은 P이다.
이진 탐색은 현재 구간의 가운데 인덱스를 잡아 그 원소와 P를 비교한 뒤, 결과에 따라 왼쪽이나 오른쪽 절반만 남긴다.
비교 횟수는 구간을 줄인 횟수가 아니라 가운데 원소와 P를 실제로 대조한 횟수이다.

사용할 원리

각 단계에서 $mid = \lfloor (low + high)/2 \rfloor$를 잡아 A[mid]와 P를 비교한다.
이 원리를 쓰는 이유는 배열이 알파벳 순으로 정렬되어 있어 비교 결과만으로 절반의 구간을 버릴 수 있기 때문이다.

계산 과정

1. 처음 구간은 0~14이므로 가운데 인덱스는 7이고 원소 N과 P를 1번 비교한다. P가 더 뒤에 있으므로 구간은 8~14가 된다.
2. 구간 8~14의 가운데 인덱스는 11이고 원소 T와 P를 2번 비교한다. P가 더 앞에 있으므로 구간은 8~10이 된다.
3. 구간 8~10의 가운데 인덱스는 9이고 원소 Q와 P를 3번 비교한다. P가 더 앞에 있으므로 구간은 8~8이 된다.
4. 구간 8~8의 가운데 인덱스는 8이고 원소 P와 4번 비교하여 탐색이 끝난다.

NoteBase

선지별 해설

① X: 2번 비교하면 N, T까지만 확인한 상태이므로 P를 특정할 수 없다. 틀린 표현은 "2번"이고, 올바르게는 N, T, Q, P의 4번 비교이다.
② X: 3번 비교 뒤에는 후보가 인덱스 8 하나로 줄어들 뿐 아직 A[8]이 P인지 확인하지 않았다. 틀린 표현은 "3번"이고, 올바르게는 마지막 확인 비교까지 포함한 4번 비교이다.
③ O: 비교 순서가 N → T → Q → P이므로 실제 비교 횟수는 4번이다.
④ X: 5번은 마지막 원소 P를 이미 비교해 놓고 같은 상황을 한 번 더 센 경우이다. 틀린 표현은 "5번"이고, 올바르게는 단일 원소 구간에서도 비교는 1번만 일어나므로 총 4번이다.

오답 포인트

후보 구간이 원소 1개로 줄어든 시점을 곧바로 탐색 완료로 착각하면 ②처럼 3번으로 오답 처리하기 쉽다. 반대로 마지막 원소 확인을 중복 계산하면 ④처럼 5번으로 세기 쉽다.

다시 나오면 이렇게 판단

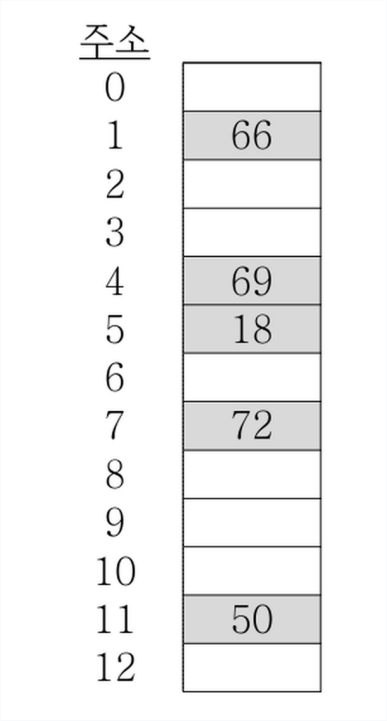
이진 탐색 횟수 문제는 구간 길이를 감으로 세지 않고, 각 단계의 가운데 인덱스와 비교한 원소를 직접 적어 내려가며 횟수를 센다. 후보가 1개로 줄어든 뒤에는 그 원소와의 최종 비교가 실제로 수행되는지도 함께 확인한다.

암기 포인트

이진 탐색의 비교 횟수는 절반으로 줄인 횟수가 아니라 가운데 원소와 실제로 맞대조한 횟수이다.

NoteBase

문 19. 그림과 같은 상태의 크기 13인 해시 테이블에 이중(double) 해싱 방법으로 원소 14를 삽입하고 이어서 원소 20을 삽입할 때, 원소 20의 저장 주소는? (단, 해시 함수 h(k,i) = (h_1(k) + i · h_2(k)) 13, 충돌 횟수 $i = 0, 1, 2, \ldots$이고, h_1(k) = k 13, h_2(k) = 1 + (k 11)이다)



① 6

② 8

③ 9

④ 10

정답 2번

출제 포인트

이 문제는 이중 해싱에서 앞선 삽입 결과까지 반영하여 탐사 수열을 계산하고 최종 저장 주소를 구하는 문제이다.

조건 정리

테이블 크기는 13이다.

기존 저장 주소는 1:66, 4:69, 5:18, 7:72, 11:50이다.

원소 14를 먼저 삽입한 뒤 원소 20을 삽입한다.

해시 함수는 h(k,i)=(h_1(k)+i· h_2(k)) 13, h_1(k)=k 13, h_2(k)=1+(k 11)이다.

NoteBase

기출OX 해설집 · 공무원/국가직/7급 · 알고리즘 · 2024 · 2차

사용할 공식

h(k,i)=(h_1(k)+i· h_2(k)) 13이다.
이 문제는 이중 해싱이므로 첫 주소는 $h_1(k)$, 충돌 뒤의 이동 간격은 $h_2(k)$로 계산한다.

계산 과정

1. 원소 14에 대해 $h_1(14) = 1$, h_2(14)=1+(14 11)=4이다.
2. 14의 탐사 순서는 1, 5, 9이고 주소 1과 5는 차 있으므로 첫 빈 칸 9에 14를 저장한다.
3. 원소 20에 대해 $h_1(20) = 7$, h_2(20)=1+(20 11)=10이다.
4. 20의 탐사 순서는 7, 4, 1, 11, 8, ...이다.
5. 주소 7, 4, 1, 11은 이미 차 있고 주소 8이 첫 빈 칸이므로 20은 8에 저장된다.

선지별 해설

① X
주소 6은 20의 탐사 수열에 뒤늦게 나타나는 값이다. 주소 8이 먼저 비어 있으므로 6까지 갈 수 없다. 틀린 표현은 '20의 저장 주소가 6이다'이고, 올바르게는 '20은 첫 빈 칸 8에 저장된다'이다.

② O
20의 탐사 수열은 7, 4, 1, 11, 8 순서이다. 앞의 네 칸은 기존 원소 72, 69, 66, 50이 차지하고 있으므로 첫 빈 칸 8에 저장된다.

③ X
주소 9는 14를 삽입할 때 확보되는 위치이다. 20은 9에 도달하기 전에 주소 8에서 이미 삽입이 끝난다. 틀린 표현은 '20의 저장 주소가 9이다'이고, 올바르게는 '9는 14의 저장 주소이고 20의 저장 주소는 8이다'이다.

④ X
주소 10은 $h_2(20) = 10$에서 나온 이동 간격일 뿐 저장 주소가 아니다. 실제 저장 주소는 (7+i· 10) 13으로 순서대로 계산해야 하며 첫 빈 칸은 8이다. 틀린 표현은 '보조 해시값 10이 곧 저장 주소이다'이고, 올바르게는 '10은 보폭이고 저장 주소는 8이다'이다.

오답 포인트

원소 14를 먼저 넣는 조건을 빼먹으면 테이블 상태가 달라진다.
또한 $h_2(k)$를 최종 주소로 오해하거나, 첫 번째 충돌 뒤 한 번만 이동하고 멈추면 오답이 된다.

다시 나오면 이렇게 판단

이중 해싱 문제는 먼저 앞선 삽입으로 테이블 상태를 갱신한 뒤, 각 원소마다 $h_1(k)$와 $h_2(k)$를 따로 구해 탐사 수열을 적어 본다. 그 수열에서 처음 만나는 빈 칸이 최종 저장 주소가 된다.

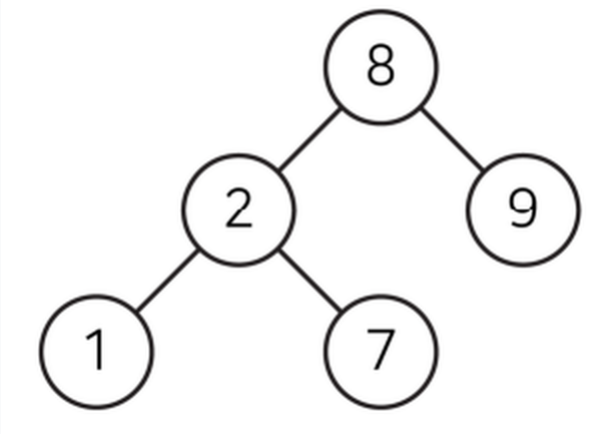
NoteBase

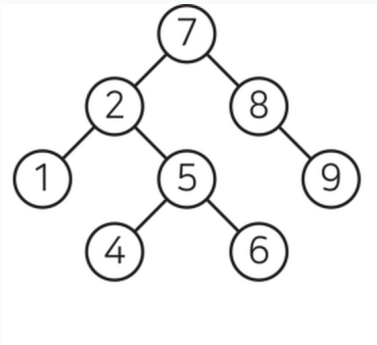
암기 포인트

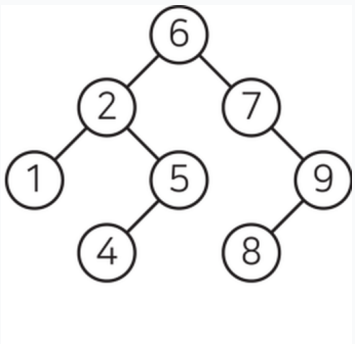
이중 해싱은 '시작점은 h_1, 이동 간격은 h_2, 정답은 첫 빈 칸'이라는 구조로 기억한다.

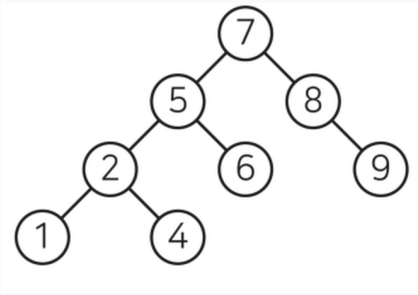
NoteBase

문 20. 다음 AVL 트리에 대해 노드 4, 5, 6을 차례대로 삽입한 결과는? (단, 리밸런싱을 위한 회전 횟수는 최소로 한다)



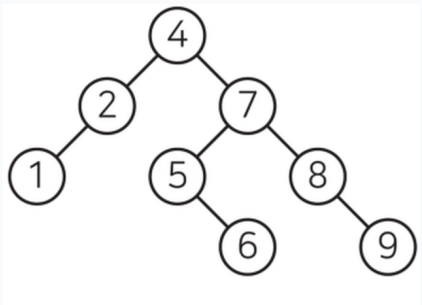
① 

② 

③ 

NoteBase

④



정답 1번

출제 포인트

이 문제는 AVL 트리에 노드 4, 5, 6을 순서대로 삽입할 때 각 단계의 불균형 지점을 찾아 최소 회전으로 최종 트리 모양을 판단하는 문제이다.

올바른 순서

1. 노드 4 삽입 — 4는 7의 왼쪽 자식으로 들어가고, 루트 8에서 좌우 높이 차가 2가 되어 LR 불균형이 발생한다.
2. 첫 리밸런싱 수행 — 노드 2를 기준으로 좌회전하고 노드 8을 기준으로 우회전하면 루트가 7이 되고 왼쪽은 2-1-4, 오른쪽은 8-9 형태가 된다.
3. 노드 5 삽입 — 5는 4의 오른쪽 자식으로 들어가며 아직 모든 노드의 균형 인수는 허용 범위 안이다.
4. 노드 6 삽입 — 6은 5의 오른쪽 자식으로 들어가고 노드 4에서 RR 불균형이 생기므로 노드 4를 기준으로 한 번 좌회전하면 5가 서브트리의 루트가 된다.

왜 이 순서: AVL 삽입은 삽입 후 아래에서 위로 올라가며 처음 만나는 불균형 노드만 최소 회전으로 바로잡으면 되므로, 6 삽입 뒤에는 노드 4만 조정하면 충분하다.

NoteBase

선지별 해설

① O
최종 트리는 루트 7, 왼쪽 서브트리 2-1-5(왼쪽 4, 오른쪽 6), 오른쪽 서브트리 8-9가 된다. 이는 4 삽입 시 LR 회전으로 루트가 7이 되고, 6 삽입 시 노드 4에서만 RR 회전을 한 결과와 일치한다.

② X
틀린 부분: 루트를 6으로 바꾸고 9의 왼쪽에 8을 둔 구조는 6 삽입 후 상위 노드까지 추가 회전을 했다고 본 형태이다.
올바르게는: 6 삽입 뒤 첫 불균형 노드는 4이므로 4에서 한 번 좌회전하면 끝나며, 루트 7과 오른쪽의 8-9 구조는 유지된다.

③ X
틀린 부분: 왼쪽 서브트리의 루트를 5로 만든 것은 노드 2에서 한 번 더 회전한 형태이다.
올바르게는: 노드 4에서 좌회전한 뒤 노드 2의 균형 인수는 허용 범위로 돌아오므로 노드 2를 추가로 회전시키지 않는다. 따라서 왼쪽 서브트리의 루트는 2이다.

④ X
틀린 부분: 루트를 4로 만든 것은 4 삽입 또는 6 삽입 후 필요 이상의 상위 회전을 가정한 형태이다.
올바르게는: 4는 일부 서브트리 내부에서만 이동하는 노드이며 전체 트리의 루트가 될 정도의 회전 중심이 아니다. 최소 회전을 적용하면 전체 루트는 7로 유지된다.

오답 포인트

함정은 삽입이 끝날 때마다 트리 전체를 다시 크게 돌린다고 생각하는 데 있다. AVL은 삽입 경로를 따라 올라가며 처음 만나는 불균형 노드에서 필요한 최소 회전만 수행한다. 특히 ③처럼 노드 2까지 한 번 더 돌리거나 ④처럼 4를 루트로 올리는 판단이 대표적인 과잉 회전 오답이다.

다시 나오면 이렇게 판단

AVL 삽입 문제는 삽입 위치를 먼저 찍고, 삽입 경로를 아래에서 위로 거슬러 올라가며 처음 균형이 깨진 노드를 찾으면 된다. 그 노드에서 LL, LR, RR, RL 중 어떤 경우인지 판정해 한 번 또는 두 번만 회전한 뒤, 상위 노드가 이미 균형이면 추가 회전을 하지 않는 방식으로 판단한다.

암기 포인트

AVL 삽입은 첫 불균형 노드에서 최소 회전만 한다는 점이 핵심이다.

NoteBase

문 21. 다음 파이썬 프로그램의 실행 결과는?

```
a=[29,88,60,26,40,74,43,22,61,40]
def partition():
    i = -1
    n = len(a)
    for j in range(0, n-2):
        if a[j] <= a[n-1]:
            i += 1
            b = a[i]
            a[i] = a[j]
            a[j] = b
    b = a[i+1]
    a[i+1] = a[n-1]
    a[n-1] = b

partition()
print(a)
```

① 29 22 26 40 40 74 43 88 61 60

② 29 22 40 26 40 74 43 88 61 60

③ 29 26 22 40 40 74 43 60 61 88

④ 29 26 40 22 40 74 43 88 61 60

정답 4번

출제 포인트

이 문제는 마지막 원소를 피벗으로 사용하는 partition 코드를 한 줄씩 추적하여 배열이 어떻게 재배치되는지를 묻는 문제이다.

조건 정리

- 초기 배열: [29, 88, 60, 26, 40, 74, 43, 22, 61, 40]
- 피벗: 마지막 원소 40
- 반복 범위: j는 0부터 7까지이므로 값 61이 있는 인덱스 8은 비교하지 않는다.
- 비교 조건: a[j] <= 40 이므로 40도 왼쪽 구간으로 이동 대상이 된다.

NoteBase

사용할 공식

이 문항은 수치 공식이 아니라 Lomuto 방식 partition 규칙을 적용하는 문제이다.
- a[j] <= pivot 이면 i를 1 증가시키고 a[i]와 a[j]를 교환한다.
- 반복이 끝나면 a[i+1]와 pivot을 교환한다.
이 규칙을 쓰는 이유는 제시된 코드가 마지막 원소를 피벗으로 삼아 pivot 이하 원소를 앞쪽으로 모으는 구현이기 때문이다.

계산 과정

1. j=0에서 29 <= 40 이므로 i=0이 되고 자기 자신과 교환하여 배열은 그대로 [29, 88, 60, 26, 40, 74, 43, 22, 61, 40]이다.
2. j=1의 88, j=2의 60은 조건을 만족하지 않는다.
3. j=3에서 26 <= 40 이므로 i=1이 되고 a[1]과 a[3]을 교환하여 [29, 26, 60, 88, 40, 74, 43, 22, 61, 40]이 된다.
4. j=4에서 40 <= 40 이므로 i=2가 되고 a[2]와 a[4]를 교환하여 [29, 26, 40, 88, 60, 74, 43, 22, 61, 40]이 된다.
5. j=5의 74, j=6의 43은 조건을 만족하지 않는다.
6. j=7에서 22 <= 40 이므로 i=3이 되고 a[3]과 a[7]을 교환하여 [29, 26, 40, 22, 60, 74, 43, 88, 61, 40]이 된다.
7. 반복 종료 후 a[i+1]=a[4]=60 과 pivot 40을 교환하므로 최종 배열은 [29, 26, 40, 22, 40, 74, 43, 88, 61, 60]이 된다.

NoteBase

선지별 해설

① X
- 앞부분을 29, 22, 26, 40으로 본 점이 틀렸다.
- 22는 j=7에서 나중에 이동하므로 26과 40보다 앞에 올 수 없다.
- 틀린 표현은 앞 네 원소 29, 22, 26, 40이며, 올바르게는 29, 26, 40, 22이다.

② X
- 22와 26의 이동 순서를 뒤바꾼 결과이다.
- 실제로는 j=3에서 26이 먼저 index 1로 이동하고, 22는 마지막 비교 구간인 j=7에서 index 3으로 이동한다.
- 틀린 표현은 29, 22, 40, 26의 배치이며, 올바르게는 29, 26, 40, 22이다.

③ X
- 마지막 교환 대상을 잘못 본 결과이다.
- 반복이 끝난 뒤 바뀌는 값은 a[4]의 60과 마지막 피벗 40뿐이므로 88은 그대로 index 7에 남아야 한다.
- 틀린 표현은 뒷부분 43, 60, 61, 88이며, 올바르게는 43, 88, 61, 60이다.

④ O
- 29, 26, 40, 22가 차례대로 앞 구간에 모이고, 마지막에 a[4]의 60과 pivot 40이 교환된다.
- 최종 배열은 [29, 26, 40, 22, 40, 74, 43, 88, 61, 60]이므로 4번이 맞다.

오답 포인트

이 문항의 함정은 세 가지이다.
- range(0, n-2)를 range(0, n-1)로 잘못 읽어 61까지 비교하는 실수가 자주 나온다.
- 비교식이 < 가 아니라 <= 이므로 값 40도 앞 구간에 들어간다는 점을 놓치기 쉽다.
- partition은 앞부분을 정렬하는 과정이 아니므로 22, 26, 40이 오름차순으로 자동 정리된다고 보면 오답이 된다.

다시 나오면 이렇게 판단

partition 추적 문제는 피벗, 반복 범위, 비교 연산자, 마지막 교환 대상을 먼저 적어 둔 뒤 조건을 만족하는 순간마다 i와 swap 위치를 한 단계씩 갱신하면 안정적으로 판단할 수 있다.

암기 포인트

Lomuto partition은 마지막 원소를 피벗으로 잡고, pivot 이하를 앞에 모은 뒤 마지막에 a[i+1]와 pivot을 바꾼다.

NoteBase

문 22. 점근적 표기의 정의로 옳지 않은 것은?

① $O(g(n)) = \{f(n) \mid$ 모든 $n \geq n_0$에 대하여 $f(n) \leq c \cdot g(n)$인 양의 상수 c와 n_0가 존재한다$\}$

② $\Omega(g(n)) = \{f(n) \mid$ 모든 $n \geq n_0$에 대하여 $c \cdot g(n) \leq f(n)$인 양의 상수 c와 n_0가 존재한다$\}$

③ $\Theta(g(n)) = \{f(n) \mid \lim_{n\to\infty}\frac{f(n)}{g(n)}=0\}$

④ $\omega(g(n)) = \{f(n) \mid \lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty\}$

정답 3번

출제 포인트

이 문제는 O, Ω, Θ, ω의 정의를 서로 구별할 수 있는지를 묻는 문제이다.

핵심 개념

$O(g(n))$는 $f(n)$의 점근적 상한이고, 충분히 큰 n에서 $f(n) \leq cg(n)$으로 눌릴 수 있음을 뜻한다.
$\Omega(g(n))$는 $f(n)$의 점근적 하한이고, 충분히 큰 n에서 $cg(n) \leq f(n)$이 성립함을 뜻한다.
$\Theta(g(n))$는 점근적으로 같은 차수라는 뜻으로, 상한과 하한을 동시에 만족해야 한다.
$\omega(g(n))$는 $f(n)$가 $g(n)$보다 점근적으로 엄격히 더 크게 증가함을 뜻한다.

선지별 해설 ①

O. $O(g(n)) = \{f(n) \mid \exists c > 0, \exists n_0, \forall n \geq n_0,\ f(n) \leq cg(n)\}$는 점근적 상한의 표준 정의이다. 선지의 핵심 표현인 "$f(n) \leq c \cdot g(n)$인 양의 상수 c와 n_0가 존재한다"가 맞다.

선지별 해설 ②

O. $\Omega(g(n)) = \{f(n) \mid \exists c > 0, \exists n_0, \forall n \geq n_0,\ cg(n) \leq f(n)\}$는 점근적 하한의 정의이다. 선지의 부등호 방향이 O와 반대라는 점이 핵심이며, 제시된 문장은 옳다.

선지별 해설 ③

X. 틀린 부분은 "$\Theta(g(n)) = \{f(n) \mid \lim_{n\to\infty} f(n)/g(n)=0\}$"라는 표현이다. $\lim_{n\to\infty} f(n)/g(n)=0$은 $f(n) = o(g(n))$를 뜻하므로 Θ가 아니라 little-o의 정의에 해당한다. 올바르게는 $\Theta(g(n))$가 되려면 충분히 큰 n에서 $c_1 g(n) \leq f(n) \leq c_2 g(n)$를 만족하는 양의 상수 c_1, c_2가 존재해야 한다.

선지별 해설 ④

O. $\omega(g(n))$는 $f(n)$가 $g(n)$보다 점근적으로 훨씬 크게 증가하는 경우를 뜻한다. 보통 $\lim_{n\to\infty} f(n)/g(n)=\infty$로 표현하며, 선지의 진술은 ω의 의미와 맞다.

NoteBase

오답 포인트

Θ와 o를 혼동하면 ③처럼 비슷해 보이는 식에 속기 쉽다. Θ는 "같은 차수"이고, 극한값이 0이면 오히려 $f(n)$이 $g(n)$보다 한 단계 작은 함수라는 뜻이다.

다시 나오면 이렇게 판단

점근적 표기 문제는 먼저 부등호 방향과 극한값의 의미를 본다. 상한이면 O, 하한이면 Ω, 양쪽을 동시에 묶으면 Θ, 비율이 0이면 o, 비율이 ∞이면 ω로 판단하면 된다.

암기 포인트

O는 위에서 누름, Ω는 아래서 받침, Θ는 같은 차수, 비율 0은 o, 비율 ∞는 ω이다.

NoteBase

문 23. 배열 A를 오름차순으로 정렬하기 위해 5, 3, 1의 간격을 사용하여 셸(Shell) 정렬을 수행할 때, 간격 3에 대한 처리까지 수행을 완료한 후의 배열 A의 상태로 옳은 것은?

i	0	1	2	3	4	5	6	7	8	9	10
A[i]	9	18	26	21	4	17	13	2	6	24	8

① 4 9 13 26 6 8 18 24 2 17 21

② 6 4 2 8 13 9 18 17 21 24 26

③ 9 2 6 13 4 17 21 8 26 24 18

④ 9 18 26 4 17 21 2 6 13 8 24

정답 2번

출제 포인트

이 문제는 셸 정렬에서 각 간격마다 같은 나머지 인덱스 집합만 삽입 정렬하고, 그 결과를 원래 인덱스 자리에 다시 놓는 과정을 추적하는지를 묻는다.

올바른 순서

1. 간격 5 처리: 인덱스 집합 (0,5,10), (1,6), (2,7), (3,8), (4,9)를 각각 삽입 정렬한다.
2. 간격 5 처리 결과: [8, 13, 2, 6, 4, 9, 18, 26, 21, 24, 17]이 된다.
3. 간격 3 처리: 위 결과에서 인덱스 집합 (0,3,6,9), (1,4,7,10), (2,5,8)를 각각 삽입 정렬한다.
4. 간격 3 처리 결과: [6, 4, 2, 8, 13, 9, 18, 17, 21, 24, 26]이 된다.

왜 이 순서인가: 간격 3 정렬은 원래 배열이 아니라 간격 5 정렬이 끝난 배열을 입력으로 사용하기 때문이다.

NoteBase

선지별 해설

1번: X. 이 배열은 간격 3으로 정렬된 부분수열을 원래 인덱스 자리에 되돌리지 않고 앞에서부터 이어 붙인 형태에 가깝다. 셸 정렬은 같은 나머지 인덱스 원소를 정렬한 뒤 다시 0,1,2,3,... 위치에 재배치해야 하므로 결과가 될 수 없다. 올바르게는 [6, 4, 2, 8, 13, 9, 18, 17, 21, 24, 26]이다.
2번: O. 간격 5 처리 뒤 배열 [8, 13, 2, 6, 4, 9, 18, 26, 21, 24, 17]에서 간격 3 부분수열을 정렬하면 (0,3,6,9) -> [6,8,18,24], (1,4,7,10) -> [4,13,17,26], (2,5,8) -> [2,9,21]가 된다. 이를 원래 위치에 넣으면 [6, 4, 2, 8, 13, 9, 18, 17, 21, 24, 26]이 되어 정답과 일치한다.
3번: X. 이 배열은 초기 배열에 바로 간격 3 정렬만 적용한 결과이다. 문제는 간격 5 처리까지 끝낸 뒤 그 다음 간격 3 처리를 묻고 있으므로 선행 단계가 빠져 있다. 올바르게는 [6, 4, 2, 8, 13, 9, 18, 17, 21, 24, 26]이다.
4번: X. 이 배열은 연속한 블록 단위로 정렬한 것처럼 보이지만, 간격 3 정렬은 연속 구간이 아니라 같은 나머지 인덱스 집합 {0,3,6,9}, {1,4,7,10}, {2,5,8}를 정렬해야 한다. 따라서 이 배열은 간격 3 처리 결과가 아니다. 올바르게는 [6, 4, 2, 8, 13, 9, 18, 17, 21, 24, 26]이다.

오답 포인트

간격 정렬을 할 때 비교 대상이 인접 원소가 아니라 같은 간격만큼 떨어진 원소라는 점을 놓치기 쉽다. 특히 3번처럼 간격 3 결과만 보고 맞다고 판단하면, 그 전에 수행된 간격 5 결과를 입력으로 써야 한다는 순서를 놓치게 된다.

다시 나오면 이렇게 판단

셸 정렬 상태 문제는 현재 간격보다 앞선 간격의 결과를 먼저 확정한 뒤, 같은 나머지 인덱스끼리 부분수열을 만들고, 각 부분수열을 삽입 정렬한 결과를 원래 인덱스에 다시 꽂아 넣는 순서로 판단한다.

암기 포인트

셸 정렬은 연속 블록 정렬이 아니라 같은 나머지 인덱스끼리의 삽입 정렬 누적이다.

NoteBase

문 24. 다음 C 코드에 해당하는 문자열 탐색 방법은? (단, s1은 입력 문자열, s2는 s1에서 찾고자 하는 패턴 문자열이고, n1은 s1의 길이, n2는 s2의 길이를 나타낸다)

```
int match(char *s1, char *s2) {
      int pt = 0, p = 0;
      while(pt != n1 && p != n2) {
            if (s1[pt] == s2[p]) {
                  pt++;
                  p++;
            } else {
                  pt = pt - p + 1;
                  p = 0;
            }
      }
      if (p == n2) return pt-p;
      else return -1;
}
```

① 브루트-포스(brute-force) 알고리즘

② 라빈-카프(Rabin-Karp) 알고리즘

③ 오토마타를 이용한 매칭 알고리즘

④ 보이어-무어(Boyer-Moore) 알고리즘

정답 1번

출제 포인트

이 문제는 문자열 탐색 코드의 불일치 처리 방식으로 브루트-포스 탐색과 전처리 기반 탐색 알고리즘을 구분하는 문제이다.

핵심 개념

브루트-포스는 텍스트의 각 시작 위치에서 패턴을 앞에서부터 하나씩 비교하다가 불일치가 나면 시작 위치를 한 칸만 옮겨 다시 비교하는 방식이다.
라빈-카프는 문자열 자체를 직접 끝까지 대조하기 전에 해시값을 비교해 후보 위치를 거르는 방식이다.
오토마타 매칭은 패턴으로 상태 전이표를 미리 만들어 입력 문자를 읽을 때 상태를 이동시키는 방식이다.
보이어-무어는 패턴의 오른쪽 끝부터 비교하고, 불일치가 나면 나쁜 문자 규칙이나 좋은 접미사 규칙으로 여러 칸을 건너뛸 수 있는 방식이다.

NoteBase

선지별 해설

1번 O: 일치하면 `pt`와 `p`를 함께 증가시키고, 불일치하면 `pt = pt - p + 1`, `p = 0`으로 되돌린다. 이는 직전 비교를 시작했던 텍스트 위치의 다음 칸으로 이동해 처음부터 다시 비교하는 전형적인 브루트-포스 문자열 탐색이다.
2번 X: 틀린 부분은 라빈-카프라고 본 점이다. 라빈-카프라면 부분 문자열과 패턴의 해시값을 계산하고 갱신하는 코드가 있어야 한다. 이 코드는 해시 연산이 전혀 없고 문자를 직접 순차 비교하므로 올바르게는 브루트-포스 알고리즘이다.
3번 X: 틀린 부분은 오토마타를 이용한 매칭이라고 본 점이다. 오토마타 매칭이라면 상태 전이 함수나 전이표에 따라 현재 상태를 갱신하는 구조가 나타나야 한다. 이 코드는 상태 전이표 없이 포인터 두 개만 되돌리므로 올바르게는 브루트-포스 알고리즘이다.
4번 X: 틀린 부분은 보이어-무어라고 본 점이다. 보이어-무어라면 패턴의 오른쪽부터 비교하고 불일치 시 여러 칸을 건너뛰는 이동량 계산이 있어야 한다. 이 코드는 왼쪽부터 비교하고 실패 시 시작 위치를 한 칸만 옮기므로 올바르게는 브루트-포스 알고리즘이다.

오답 포인트

문자열 탐색 문제에서 알고리즘 이름만 외우고 전처리 흔적을 보지 않으면 헷갈리기 쉽다. 이 코드에는 해시, 상태 전이표, 큰 폭의 점프가 전혀 없는데도 문자열 탐색이라는 이유만으로 2번, 3번, 4번을 고르기 쉽다.

다시 나오면 이렇게 판단

문자열 탐색 코드를 보면 먼저 불일치가 났을 때 무엇을 이용해 다음 위치를 정하는지 본다. 시작 위치를 한 칸만 옮기면 브루트-포스, 해시를 쓰면 라빈-카프, 상태 전이를 쓰면 오토마타, 오른쪽부터 비교하며 큰 폭으로 건너뛰면 보이어-무어로 판단할 수 있다.

암기 포인트

한 칸씩 다시 시작하면 브루트-포스, 해시면 라빈-카프, 상태 전이면 오토마타, 오른쪽 비교와 점프면 보이어-무어이다.

NoteBase

문 25. 다음은 텍스트에서 패턴과 일치하는 문자열을 찾기 위한 KMP(Knuth Morris Pratt) 알고리즘에서의 전처리 과정을 구현한 파이썬 프로그램이다. 패턴이 abcbabca인 경우, 아래 표의 (가) ~ (라)에 들어갈 숫자를 바르게 연결한 것은?

```
def preprocess(pn: str):
    pt = 1
    k = 0
    pi = [0] * (len(pn) + 1)
    pi[pt-1] = -1
    pi[pt] = 0
    while pt != len(pn):
        if pn[pt] == pn[k]:
            k += 1
            pt += 1
            pi[pt] = k
        elif k == 0:
            pt += 1
            pi[pt] = k
        else:
            k = pi[k]
    print(pi)
pattern = 'abcbabca'
preprocess(pattern)
```

i	0	1	2	3	4	5	6	7	8
pn[i]	a	b	c	b	a	b	c	a	
pi[i]	-1	0	0	0	0	(가)	(나)	(다)	(라)

① (가): 1, (나): 1, (다): 1, (라): 1

② (가): 1, (나): 2, (다): 3, (라): 1

③ (가): 1, (나): 2, (다): 3, (라): 0

④ (가): 3, (나): 3, (다): 3, (라): 0

정답 2번

출제 포인트

이 문제는 KMP 전처리에서 실패 함수 배열 `pi`를 한 칸씩 갱신하는 과정을 정확히 따라갈 수 있는지를 묻는 문제이다.

NoteBase

조건 정리

패턴은 `abcbabca`이다.
초깃값은 `pt = 1`, `k = 0`, `pi[0] = -1`, `pi[1] = 0`이다.
표에서 구해야 할 값은 `pi[5]`, `pi[6]`, `pi[7]`, `pi[8]`이다.

계산 과정

`pt = 1, 2, 3`에서는 각각 `b`, `c`, `b`가 `a`와 다르므로 `k = 0` 상태에서 한 칸씩 넘겨 `pi[2] = 0`, `pi[3] = 0`, `pi[4] = 0`이 된다.
`pt = 4`에서는 `pn[4] = a`와 `pn[0] = a`가 같으므로 `k = 1`, `pt = 5`, `pi[5] = 1`이 된다.
`pt = 5`에서는 `pn[5] = b`와 `pn[1] = b`가 같으므로 `k = 2`, `pt = 6`, `pi[6] = 2`가 된다.
`pt = 6`에서는 `pn[6] = c`와 `pn[2] = c`가 같으므로 `k = 3`, `pt = 7`, `pi[7] = 3`이 된다.
`pt = 7`에서는 `pn[7] = a`와 `pn[3] = b`가 달라서 바로 0으로 두는 것이 아니라 `k = pi[3] = 0`으로 되돌아간다.
같은 `pt = 7`에서 다시 `pn[7] = a`와 `pn[0] = a`를 비교하면 같으므로 `k = 1`, `pt = 8`, `pi[8] = 1`이 된다.
따라서 `(가), (나), (다), (라)`는 차례로 `1, 2, 3, 1`이다.

선지 판단

1번 X: `(가)`는 1로 맞지만 `(나)`, `(다)`를 1로 본 부분이 틀리다. `ab` 다음에는 접두사 길이가 2, `abc` 다음에는 접두사 길이가 3으로 늘어나므로 올바르게는 `1, 2, 3, 1`이다.
2번 O: `pi[5] = 1`, `pi[6] = 2`, `pi[7] = 3`까지는 연속 일치로 증가하고, 마지막에는 한 번 실패 후 다시 `a`가 맞아 `pi[8] = 1`이 되므로 `1, 2, 3, 1`이 정확하다.
3번 X: 마지막 값을 0으로 둔 부분이 틀리다. `pn[7]`와 `pn[3]`가 다를 때 즉시 `pi[8] = 0`이 되는 것이 아니라 `k`를 `pi[3]`으로 되돌린 뒤 같은 위치를 다시 비교한다. 다시 비교하면 `a`와 `a`가 맞으므로 올바르게는 `1, 2, 3, 1`이다.
4번 X: `pi[5]`, `pi[6]`, `pi[7]`를 모두 3으로 본 부분이 틀리다. 첫 일치는 `a` 하나만 맞은 상태이므로 `pi[5] = 1`에서 시작해 차례로 2, 3이 된다. 올바르게는 `1, 2, 3, 1`이다.

오답 포인트

마지막 `a`에서 불일치가 나오면 곧바로 0을 적는 실수가 가장 흔하다. 이 코드는 불일치가 나면 `k = pi[k]`로 되돌아가 같은 `pt`를 다시 비교하므로, 재비교 후 다시 일치할 수 있다는 점이 함정이다.

다시 나오면 이렇게 판단

KMP 전처리 문제는 `현재 문자 비교 -> 일치 시 k 증가 -> 불일치 시 k를 pi[k]로 후퇴 -> 같은 위치 재비교` 순서로 추적하면 된다. 불일치가 한 번 나와도 바로 0으로 확정하지 않는지가 핵심 판단 기준이다.

NoteBase

암기 포인트

KMP 실패 함수는 불일치가 나와도 `k`를 먼저 되돌리고, 같은 문자에서 다시 접두사 일치를 확인하는 배열이다.

NoteBase