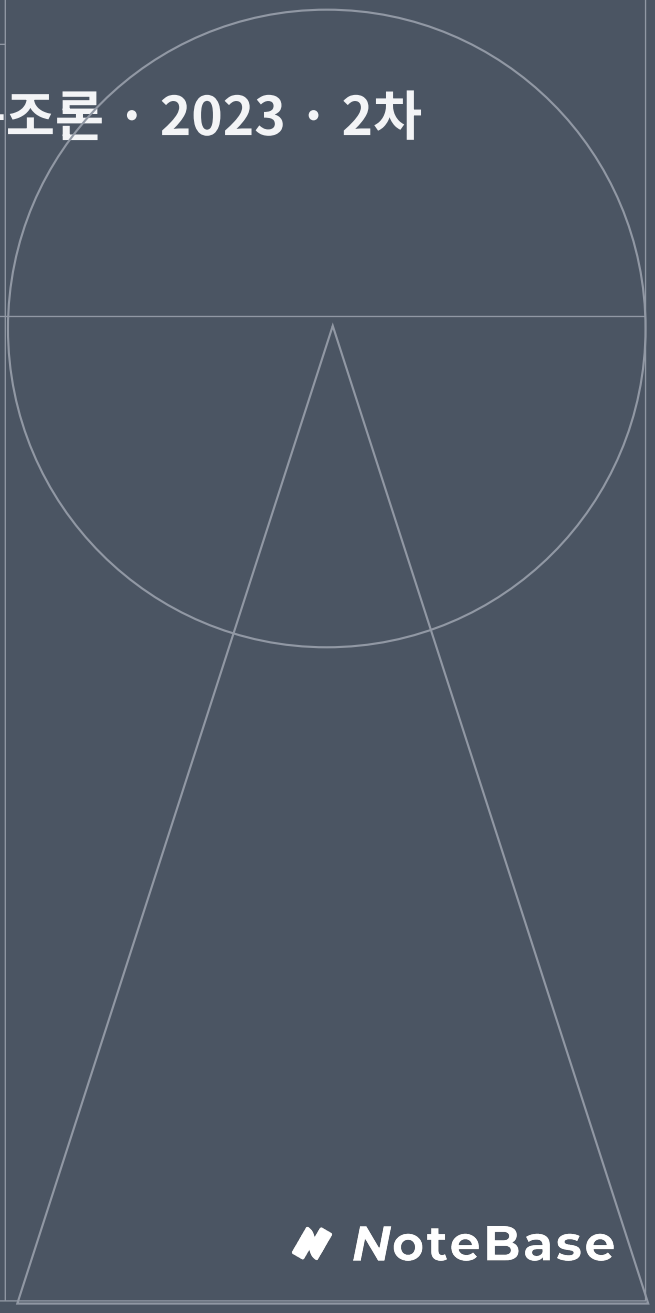
기출OX 해설집

공무원/국가직/7급 · 자료구조론 · 2023 · 2차



NoteBase

기출OX 해설집

공무원/국가직/7급 · 자료구조론 · 2023 · 2차

공무원/국가직/7급 · 자료구조론 · 2023 · 2차

문 1. 다음 C 코드의 시간 복잡도(time complexity)는?

```
void func(int n)
{
  int i, j, k = 0;
  for(i = 0; i < (n + 200); i++)
    for(j = 0; j < 300; j++)
      printf("k = %d\n", k++);
}
```

① $O(n)$

② $O(n^2)$

③ $O(\log_2 n)$

④ $O(n\log_2 n)$

정답 1번

출제 포인트

이 문제는 중첩 반복문의 실제 반복 횟수를 계산한 뒤 상수항과 상수배를 제거하여 점근적 시간 복잡도를 판단하는 문제이다.

NoteBase

조건 정리

주어진 값:
- 바깥 반복문은 $i = 0$부터 $i < n + 200$ 전까지 실행되므로 반복 횟수는 $n + 200$회이다.
- 안쪽 반복문은 $j = 0$부터 $j < 300$까지 실행되므로 반복 횟수는 300회이다.
- 안쪽에서 수행하는 `printf`는 한 번 실행될 때마다 상수 시간으로 본다.

구해야 할 값:
- 전체 `printf` 실행 횟수와 그에 따른 시간 복잡도이다.

주의할 조건:
- 200과 300은 모두 상수이므로 빅오 표기에서는 지배항만 남긴다.

사용할 공식

전체 수행 횟수는 바깥 반복 횟수와 안쪽 반복 횟수의 곱으로 계산한다.
$T(n) = (n + 200) \times 300 = 300n + 60000$
이 식을 쓰는 이유는 안쪽 반복문이 바깥 반복문이 한 번 돌 때마다 항상 300번씩 고정 실행되기 때문이다.

계산 과정

1. 바깥 반복문의 횟수는 $n + 200$회이다.
2. 각 바깥 반복마다 안쪽 반복문이 300회 실행된다.
3. 따라서 전체 실행 횟수는 $300(n + 200) = 300n + 60000$이다.
4. 빅오 표기에서는 상수배 300과 상수항 60000을 제거하므로 $T(n) = O(n)$이다.

NoteBase

선지별 해설

① O
- 계산 결과가 $300n + 60000$이므로 지배항은 n이다. 따라서 시간 복잡도는 $O(n)$이다.

② X
- $O(n^2)$가 되려면 두 반복문이 모두 n에 비례하여 함께 증가해야 한다. 이 코드의 안쪽 반복문은 300번으로 고정이므로 제곱 시간이 아니다.
- 틀린 표현은 "중첩 반복문이므로 무조건 $O(n^2)$이다"라는 판단이다. 올바르게는 "안쪽 반복 횟수가 상수이면 전체는 바깥 반복문 차수와 같은 $O(n)$이다"이다.

③ X
- $O(\log_2 n)$은 입력이 절반씩 줄어드는 형태에서 주로 나온다. 이 코드는 $i++$, $j++$로 1씩 증가할 뿐이며 로그형 감소나 분할이 없다.
- 틀린 표현은 "반복문이 하나만 n을 포함하므로 로그 시간일 수 있다"라는 판단이다. 올바르게는 "증가 폭이 일정한 선형 반복은 로그 시간이 아니라 선형 시간이다"이다.

④ X
- $O(n\log_2 n)$이 되려면 보통 한 축은 n, 다른 축은 $\log n$으로 증가해야 한다. 이 코드의 안쪽 반복문은 $\log n$이 아니라 상수 300이므로 해당하지 않는다.
- 틀린 표현은 "중첩 구조이므로 선형과 로그가 곱해진다"라는 판단이다. 올바르게는 "상수 반복은 $\log n$ 요인을 만들지 못하므로 $O(n)$으로 본다"이다.

오답 포인트

중첩 반복문이라는 모양만 보고 곧바로 $O(n^2)$로 가는 실수가 가장 흔하다. 이 문제의 함정은 바깥 반복문에만 n이 있고 안쪽은 300이라는 상수라는 점이다. 또한 $n + 200$처럼 상수가 더해져 있어도 차수는 바뀌지 않는다는 점을 놓치기 쉽다.

다시 나오면 이렇게 판단

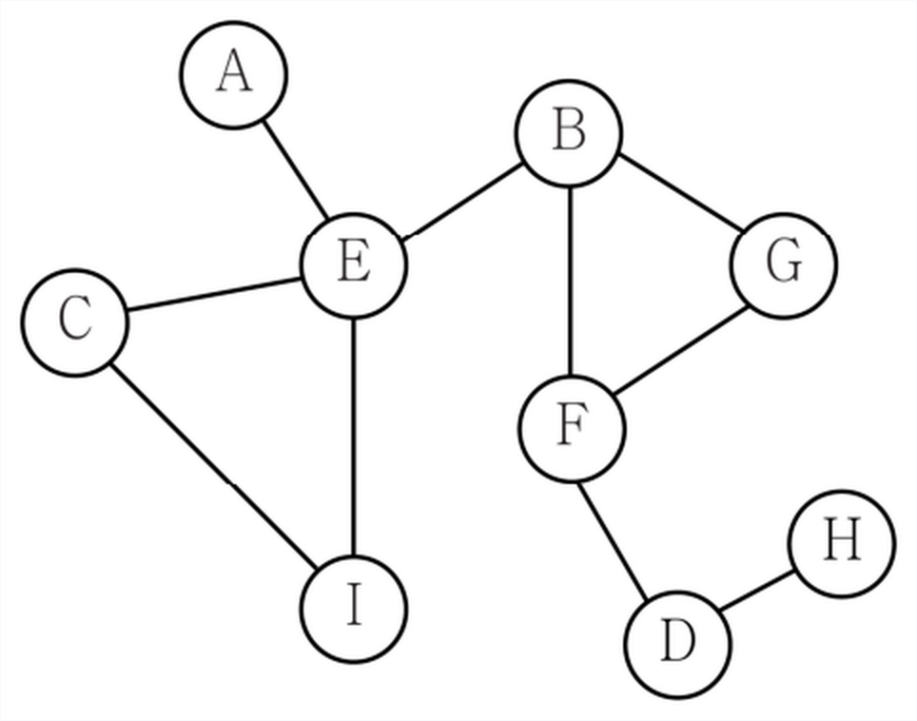
반복문 복잡도는 먼저 각 반복문의 실제 반복 횟수가 n에 의존하는지, 상수인지부터 분리해서 본다. 그다음 전체 횟수를 곱하거나 더한 뒤 상수배와 상수항을 제거하면 차수가 바로 드러난다.

암기 포인트

중첩 반복문이라도 안쪽이 상수회이면 전체 차수는 바깥 반복문 차수와 같아서 $O(n)$이다.

NoteBase

문 2. 다음 그래프(graph)에 대하여 깊이 우선 탐색(DFS, depth first search)을 하려고 한다. 시작 정점(vertex)이 B일 때, 정점 I는 몇 번째에 방문하는가? (단, B가 첫 번째 방문 정점이고, 탐색 시 인접한 정점들은 알파벳 오름차순으로 방문한다)



① 4

② 5

③ 8

④ 9

정답 2번

출제 포인트

이 문제는 그래프 그림을 보고 시작 정점 B에서 DFS를 수행할 때, 인접 정점을 알파벳 오름차순으로 따라가며 방문 순서를 정확히 추적하는지를 묻는 문제이다.

NoteBase

선지별 해설

① X
B에서 시작하면 방문 순서는 B→E→A→C→I→F→D→H→G가 된다. 정점 I는 4번째가 아니라 5번째에 방문한다.
틀린 표현은 "4번째"이고, 올바르게는 "5번째"이다.

② O
B의 인접 정점은 E, F, G이므로 먼저 E로 간다. E의 인접 정점 A, B, C, I 중 방문하지 않은 정점을 알파벳순으로 따라가면 A 다음 C, 그다음 I를 방문하므로 I는 전체에서 5번째가 된다.

③ X
정점 I를 8번째라고 보면 F-D-H 쪽 탐색까지 끝난 뒤에 I를 만난 것으로 잘못 계산한 것이다. 실제 DFS에서는 B→E로 내려간 뒤 C를 거쳐 I를 먼저 방문한다.
틀린 표현은 "8번째"이고, 올바르게는 "5번째"이다.

④ X
정점 I를 9번째라고 보면 마지막 정점으로 착각한 것이다. 실제 마지막 방문 정점은 G이고, I는 E 하위 가지에서 이미 방문된다.
틀린 표현은 "9번째"이고, 올바르게는 "5번째"이다.

오답 포인트

B의 인접 정점만 보고 E, F, G를 차례로 세면 오답이 된다. DFS는 한 정점에서 갈 수 있는 곳을 끝까지 내려간 뒤 되돌아오므로, E에 들어간 순간 A-C-I를 모두 처리한 다음에야 F 쪽으로 넘어간다는 점이 함정이다.

다시 나오면 이렇게 판단

DFS 순서 문제는 현재 정점의 미방문 인접 정점을 정렬 기준대로 하나 선택한 뒤, 더 내려갈 곳이 없을 때만 되돌아온다고 놓고 방문 번호를 직접 적으면 된다. 이 문제에서는 B에서 E를 먼저 선택한 뒤 E-A-C-I를 끝내고 돌아온다고 판단하면 I의 순서를 바로 구할 수 있다.

암기 포인트

DFS는 옆으로 넓게 가지 않고 한 가지를 끝까지 내려간 뒤 되돌아오는 탐색이다.

NoteBase

문 3. 이중 연결 리스트(doubly linked list)에 대한 설명으로 옳지 않은 것은?

① 이전 노드와 이후 노드에 대한 링크(link)를 가지고 있어 양방향 탐색이 가능하다.

② 단순 연결 리스트(singly linked list)에 비해 기억 장소가 추가로 소요된다.

③ 이진 트리(binary tree)에 활용이 가능하다.

④ 다항식(polynomial), 희소 행렬(sparse matrix)을 표현할 수 없다.

정답 4번

출제 포인트

이 문제는 이중 연결 리스트의 구조적 특징, 단순 연결 리스트와의 차이, 그리고 활용 가능 표현 대상을 구별하는 문제이다.

핵심 개념

단순 연결 리스트는 각 노드가 다음 노드만 가리키는 선형 구조이다.
이중 연결 리스트는 각 노드가 이전 노드와 다음 노드를 모두 가리키는 선형 구조이다.
결정 차이축은 링크 수와 탐색 방향이다. 단순 연결 리스트는 한 방향 탐색이 기본이고, 이중 연결 리스트는 양방향 탐색이 가능하다.
연결 리스트는 노드와 링크로 데이터를 연결하므로 다항식, 희소 행렬처럼 0이 아닌 항만 연결해 표현하는 데 활용할 수 있다.

NoteBase

선지별 해설

① O
이중 연결 리스트의 각 노드는 이전 노드를 가리키는 링크와 다음 노드를 가리키는 링크를 함께 가진다. 따라서 앞쪽과 뒤쪽으로 모두 이동할 수 있어 양방향 탐색이 가능하다.

② O
단순 연결 리스트는 보통 링크가 1개이지만, 이중 연결 리스트는 이전 링크가 추가된다. 따라서 노드마다 저장해야 할 포인터가 더 많아져 기억 장소가 추가로 소요된다.

③ O
이중 연결 리스트 노드는 링크를 두 개 가지므로, 두 링크를 좌·우 자식 포인터처럼 대응시키는 방식으로 이진 트리 표현에 응용할 수 있다. 핵심은 링크가 두 개라는 구조적 유사성이다.

④ X
틀린 부분은 "표현할 수 없다"이다. 연결 리스트는 다항식의 각 항이나 희소 행렬의 0이 아닌 원소만 노드로 연결하여 표현할 수 있다. 올바르게는 "다항식, 희소 행렬을 표현할 수 있다"이다.

오답 포인트

④는 연결 리스트의 대표 활용 예를 부정문으로 뒤집어 놓은 함정이다. 다항식과 희소 행렬은 배열보다 연결 리스트와 자주 연결되는 전형적 응용인데, "표현할 수 없다"라는 단정 표현을 그대로 믿으면 오답이 된다. ③도 엄밀한 자료구조 명칭 차이만 보고 틀렸다고 판단하기 쉬우나, 문제는 응용 가능성을 묻고 있으므로 구조적 활용 관점에서 보아야 한다.

다시 나오면 이렇게 판단

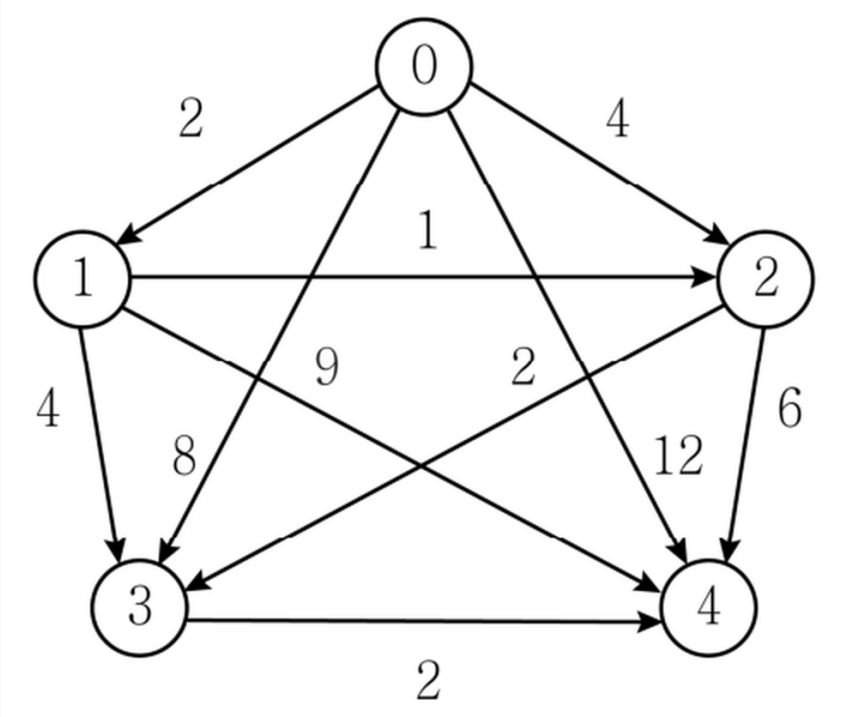
연결 리스트 문제에서는 먼저 링크 개수와 방향을 본다. 그다음 대표 응용 예가 다항식, 희소 행렬 같은 연결 표현인지 확인한다. 선지에 "할 수 없다" 같은 강한 부정 표현이 나오면 전형적 활용 예와 충돌하는지부터 점검하면 된다.

암기 포인트

이중 연결 리스트는 이전·다음 2링크, 양방향 탐색, 메모리 추가, 다항식·희소 행렬 표현 가능이다.

NoteBase

문 4. 다음은 정점 간의 거리를 표시한 방향 그래프(directed graph)이다. 시작 정점이 0일 때, 정점 3과 정점 4까지 최단 경로의 거리를 바르게 연결한 것은?



① 정점 3: 5, 정점 4: 7

② 정점 3: 5, 정점 4: 8

③ 정점 3: 8, 정점 4: 11

④ 정점 3: 8, 정점 4: 12

정답 1번

출제 포인트

가중치가 있는 방향 그래프에서 시작 정점 0으로부터 각 정점까지의 최단경로 거리를 누적 비교하는 문제이다.

조건 정리

주어진 값: 0→1=2, 0→2=4, 0→3=8, 0→4=12, 1→2=1, 1→3=4, 1→4=9, 2→3=2, 2→4=6, 3→4=2이다.
구해야 할 값: 시작 정점 0에서 정점 3과 정점 4까지의 최단경로 거리이다.
주의할 조건: 방향 그래프이므로 화살표 방향대로만 이동하며, 직행 간선보다 여러 간선을 거친 경로가 더 짧을 수 있다.

NoteBase

사용할 공식

최단경로는 각 후보 경로의 가중치 합 중 최솟값으로 정한다.
정점 3까지의 거리 = min(0→3, 0→1→3, 0→2→3, 0→1→2→3)이다.
정점 4까지의 거리 = min(0→4, 0→1→4, 0→2→4, 0→3→4, 0→1→2→4, 0→1→3→4, 0→2→3→4, 0→1→2→3→4)이다.
이 공식을 쓰는 이유: 문제는 시작 정점에서 각 도착 정점까지의 최단 거리 자체를 묻기 때문이다.

계산 과정

1. 정점 3의 후보 거리를 계산하면 직행은 8, 0→1→3은 2+4=6, 0→2→3은 4+2=6, 0→1→2→3은 2+1+2=5이다.
2. 따라서 정점 3의 최단경로 거리는 5이다.
3. 정점 4의 후보 거리를 계산하면 직행은 12, 0→3→4는 8+2=10, 0→2→4는 4+6=10, 0→1→4는 2+9=11, 0→1→3→4는 2+4+2=8, 0→2→3→4는 4+2+2=8, 0→1→2→4는 2+1+6=9, 0→1→2→3→4는 2+1+2+2=7이다.
4. 따라서 정점 4의 최단경로 거리는 7이다.

선지별 해설

① O: 정점 3의 최단경로 거리는 0→1→2→3으로 5이고, 정점 4의 최단경로 거리는 0→1→2→3→4로 7이므로 제시한 쌍이 맞다.
② X: 정점 3의 값 5는 맞지만 정점 4를 8로 본 것은 0→1→3→4 또는 0→2→3→4까지만 비교한 경우이다. 틀린 표현은 '정점 4 = 8'이며, 올바르게는 0→1→2→3→4의 합이 7이므로 '정점 4 = 7'이다.
③ X: 정점 3을 8로 본 것은 0→3 직행 간선만 보고 더 짧은 우회 경로를 놓친 경우이다. 틀린 표현은 '정점 3 = 8, 정점 4 = 11'이며, 올바르게는 정점 3은 0→1→2→3으로 5이고 정점 4는 0→1→2→3→4로 7이다.
④ X: 정점 3을 8로 둔 점과 정점 4를 12로 둔 점이 모두 직행 간선만 택한 오답이다. 틀린 표현은 '정점 3 = 8, 정점 4 = 12'이며, 올바르게는 정점 3은 5, 정점 4는 7이다.

오답 포인트

직행 간선의 가중치만 보고 바로 답을 정하면 오답이 된다. 이 문항은 ③과 ④처럼 한 번의 이동만 최단경로로 착각하게 만들고, ②처럼 중간 정점을 더 거치면 총합이 더 작아진다는 점을 놓치게 만든다.

다시 나오면 이렇게 판단

가중치 방향 그래프의 최단경로 문제는 직행 경로를 먼저 적은 뒤, 중간 정점을 1개 거치는 경로와 2개 이상 거치는 경로까지 차례로 더해 보고 최솟값을 확정하는 방식으로 판단한다.

NoteBase

암기 포인트

최단경로는 선분 모양이 아니라 가중치 합의 최솟값이며, 0→1→2→3→4처럼 경유가 늘어도 총합이 더 작으면 그 경로가 정답이다.

NoteBase

문 5. 어떤 트리 3개에 대한 모든 노드 개수의 합이 100이면 간선(edge) 개수의 합은?

① 94

② 95

③ 96

④ 97

정답 4번

출제 포인트

트리의 간선 수는 노드 수보다 항상 1개 적다는 성질을 여러 트리에 합산해 적용하는 문제이다.

조건 정리

주어진 값: 트리 3개의 전체 노드 수 합은 100이다.
구해야 할 값: 세 트리의 간선 수 합이다.
주의할 조건: 트리가 3개이므로 각 트리마다 간선은 노드보다 1개씩 적다.

사용할 공식

트리 1개의 간선 수는 $n-1$이다.
따라서 트리 3개의 간선 수 합은 $(n_1-1)+(n_2-1)+(n_3-1)=(n_1+n_2+n_3)-3$이다.
이 공식을 쓰는 이유는 각 트리가 연결되고 사이클이 없는 구조이므로, 트리마다 간선 수가 노드 수보다 정확히 1개 적기 때문이다.

계산 과정

1. 세 트리의 노드 수를 각각 n_1, n_2, n_3라 두면 $n_1+n_2+n_3=100$이다.
2. 각 트리의 간선 수는 각각 n_1-1, n_2-1, n_3-1이다.
3. 전체 간선 수는 $(n_1-1)+(n_2-1)+(n_3-1)=100-3=97$이다.

선지별 해설

① X: 94는 트리 3개라는 조건을 올바르게 반영한 값이 아니다. 각 트리마다 1개씩만 빼야 하므로 전체 간선 수는 94가 아니라 97이다.
② X: 95는 노드 수 합 100에서 5를 뺀 값인데, 트리 성질상 빼야 하는 개수는 트리 수 3개뿐이다. 올바른 값은 95가 아니라 97이다.
③ X: 96은 간선 수가 노드 수보다 1개 적다는 성질을 트리 1개에만 적용한 값처럼 보인다. 이 문제는 트리 3개이므로 96이 아니라 97이 된다.
④ O: 트리마다 간선 수는 노드 수보다 1개 적으므로 세 트리의 전체 간선 수는 100에서 3을 뺀 97이다.

NoteBase

오답 포인트

수험생이 전체를 하나의 트리처럼 보고 1만 빼거나, 반대로 근거 없이 더 많이 빼면 오답이 된다. 이 문항의 핵심 함정은 '트리 3개'라는 조건을 놓치고 트리의 개수만큼 1을 빼야 한다는 점을 잊는 데 있다.

다시 나오면 이렇게 판단

여러 개의 트리가 주어지면 먼저 트리 개수를 센 뒤, 전체 노드 수 합에서 트리 개수만큼 1을 뺀다. 즉 전체 간선 수는 노드 수 합에서 트리 수를 뺀 값으로 판단할 수 있다.

암기 포인트

트리 k개의 전체 간선 수는 전체 노드 수 합에서 k를 뺀 값이다.

NoteBase

문 6. 다음 하노이 탑 문제를 해결하는 파이썬(Python) 코드의 시간 복잡도는? (단, n은 원판의 개수이고 first, second, third는 기둥 번호이다)

```
def hanoiT(n, first, second, third):
  if n == 1:
    print("원판1 : %s 에서 %s"%(first, third))
  else:
    hanoiT(n - 1, first, third, second)
    print("원판%d : %s 에서 %s"%(n, first, third))
    hanoiT(n - 1, second, first, third)
```

① $O(n)$

② $O(n^2)$

③ $O(2^n)$

④ $O(\log_2 n)$

정답 3번

출제 포인트

이 문제는 하노이 탑 재귀 함수의 호출 횟수를 점화식으로 세워 시간 복잡도를 판단하는 문제이다.

조건 정리

주어진 값: 원판 수는 n이고, 함수는 $n = 1$일 때 출력 1번으로 끝난다.
구해야 할 값: 함수 전체의 시간 복잡도이다.
주의할 조건: $n > 1$이면 $n - 1$에 대한 재귀 호출이 두 번 일어나고, 그 사이의 출력문은 상수 시간이다.

사용할 공식

점화식은 $T(1) = 1$, $T(n) = 2T(n - 1) + 1$이다.
이 공식을 쓰는 이유는 매 단계에서 크기 $n - 1$ 문제를 두 번 풀고, 현재 원판 이동 출력 1번이 추가되기 때문이다.

NoteBase

계산 과정

1. $n = 1$일 때는 출력 1번이므로 $T(1) = 1$이다.
2. $n > 1$일 때는 첫 번째 재귀 호출 1번, 현재 원판 출력 1번, 두 번째 재귀 호출 1번이므로 $T(n) = 2T(n-1) + 1$이다.
3. 이 점화식을 전개하면 $T(n) = 2^n - 1$이므로 시간 복잡도는 $O(2^n)$이다.

선지별 해설

① X: $O(n)$은 재귀가 한 갈래로만 진행되거나 단계마다 상수 작업만 누적될 때의 형태이다. 이 코드는 매 단계에서 $n-1$ 크기 문제를 두 번 호출하므로 선형 시간이 아니다. 틀린 표현은 $O(n)$이고, 올바르게는 $O(2^n)$이다.
② X: $O(n^2)$은 보통 이중 반복이나 $T(n) = T(n-1) + n$ 같은 점화식에서 나온다. 이 코드는 단계마다 선형량이 더해지는 구조가 아니라 호출 수 자체가 두 배씩 늘어난다. 틀린 표현은 $O(n^2)$이고, 올바르게는 $O(2^n)$이다.
③ O: 점화식이 $T(n) = 2T(n-1) + 1$이므로 해는 $2^n - 1$이고, 상수항을 제외하면 시간 복잡도는 $O(2^n)$이다.
④ X: $O(\log_2 n)$은 문제 크기가 절반씩 줄어드는 이진 탐색류에서 나타난다. 이 코드는 문제 크기가 $n-1$로 1씩만 줄고, 동시에 두 갈래 재귀가 생기므로 로그 시간이 아니다. 틀린 표현은 $O(\log_2 n)$이고, 올바르게는 $O(2^n)$이다.

오답 포인트

재귀 깊이가 n이므로 $O(n)$이라고 바로 단정하거나, 깊이만 보고 $O(\log n)$으로 착각하는 함정이 있다. 이 문제는 깊이만 볼 것이 아니라 각 단계에서 하위 문제가 몇 개 생기는지를 함께 봐야 한다.

다시 나오면 이렇게 판단

재귀 시간 복잡도는 먼저 한 번 호출될 때 하위 호출이 몇 개 생기는지 세고, 다음으로 문제 크기가 얼마나 줄어드는지 본다. 같은 크기의 하위 문제가 두 번 생기면 $T(n) = 2T(n-1)$ +상수 형태를 먼저 의심하면 된다.

암기 포인트

하노이 탑 이동 횟수와 재귀 호출 수는 $2^n - 1$ 꼴이므로 시간 복잡도는 $O(2^n)$이다.

NoteBase

문 7. 어떤 이진 트리의 전위 순회(preorder traversal)와 중위 순회(inorder traversal)한 순서가 다음과 같을 때, 이 트리의 단말 노드만을 모두 나열한 것은?

◯ 전위 순회: A, B, D, E, C, F, G, H, I
◯ 중위 순회: E, D, B, A, G, F, H, C, I

① A, D, E
② B, C, D, F
③ D, E, F, I
④ E, G, H, I

정답 4번

출제 포인트

전위 순회의 첫 원소를 루트로 잡고, 중위 순회에서 루트의 위치를 기준으로 좌우 서브트리를 분할하여 이진 트리를 복원한 뒤 말단 노드를 찾는 문제이다.

올바른 순서

1. 전위 순회의 첫 원소 A를 전체 루트로 확정한다.
2. 중위 순회에서 A를 기준으로 왼쪽 E, D, B와 오른쪽 G, F, H, C, I로 나누어 좌우 서브트리 범위를 정한다.
3. 왼쪽 서브트리는 B가 루트이고, 그 왼쪽에 D, 다시 그 왼쪽에 E가 온다. 오른쪽 서브트리는 C가 루트이고, 그 왼쪽에 F, 오른쪽에 I가 오며 F의 왼쪽과 오른쪽에 각각 G와 H가 온다.
4. 자식이 없는 노드만 고르면 E, G, H, I가 된다.

왜 이 순서: 전위 순회는 루트를 먼저 알려 주고 중위 순회는 그 루트의 좌우 경계를 정해 주므로, 루트 확정 후 좌우 분할 순서로 복원해야 자식 관계가 정확히 나온다.

NoteBase

선지별 해설

① X
A는 전체 루트이므로 말단 노드가 아니다. D도 왼쪽 자식 E를 가지므로 내부 노드이다. 틀린 표현은 A, D를 말단 노드에 포함한 부분이다. 올바르게는 말단 노드는 E, G, H, I이다.

② X
B는 왼쪽 자식 D를 가지고, C는 자식 F와 I를 가지며, D는 자식 E를 가지고, F는 자식 G와 H를 가진다. 따라서 B, C, D, F는 모두 내부 노드이다. 틀린 표현은 내부 노드만 말단 노드로 나열한 부분이다. 올바르게는 자식이 없는 E, G, H, I가 말단 노드이다.

③ X
E와 I는 말단 노드가 맞지만 D와 F는 각각 자식을 가진다. D는 E의 부모이고 F는 G와 H의 부모이므로 말단 노드가 아니다. 틀린 표현은 D, F를 포함한 부분이다. 올바르게는 D, F 대신 G, H가 들어가야 한다.

④ O
복원된 트리에서 E는 D의 왼쪽 자식으로 끝나고, G와 H는 F의 좌우 자식으로 더 내려갈 자식이 없으며, I는 C의 오른쪽 자식으로 끝난다. 따라서 E, G, H, I가 말단 노드 전체이다.

오답 포인트

루트와 내부 노드를 말단 노드와 혼동하기 쉽다. 특히 D와 F처럼 한쪽 또는 양쪽 자식을 가진 노드를 끝에 가까워 보인다는 이유로 말단으로 착각하면 ③을 고르기 쉽다.

다시 나오면 이렇게 판단

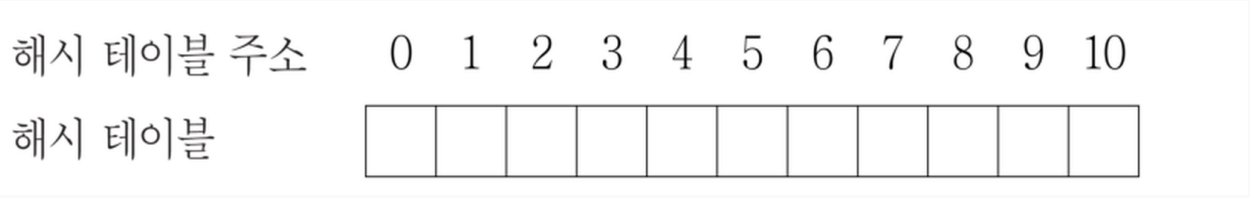
전위 순회에서는 첫 원소로 루트를 먼저 잡고, 중위 순회에서는 그 루트의 위치로 좌우 서브트리를 나눈다. 이 과정을 재귀적으로 반복해 자식 관계를 복원한 뒤 자식이 하나도 없는 노드만 추리면 말단 노드를 안정적으로 찾을 수 있다.

암기 포인트

전위는 루트 먼저, 중위는 좌우 경계, 자식이 없으면 말단 노드이다.

NoteBase

문 8. 다음 비어 있는 크기가 11인 해시 테이블(hash table)에 입력키가 4, 6, 15, 26, 17 순서로 저장되었다. 입력키 26이 저장된 해시 테이블의 주소는? (단, 해시 함수는 h(k) = k 11이고, 충돌 해결은 선형 조사법(linear probing)을 사용한다)



① 1

② 4

③ 6

④ 7

정답 4번

출제 포인트

이 문제는 해시 함수 h(k)=k 11로 초기 주소를 구한 뒤, 충돌이 나면 선형 조사법으로 다음 빈 칸을 찾아 최종 저장 주소를 계산하는 문제이다.

조건 정리

주어진 값: 해시 테이블 크기 11, 입력 순서 4, 6, 15, 26, 17
구해야 할 값: 입력키 26이 실제로 저장되는 주소
주의할 조건: 충돌이 발생하면 원래 해시값에 머무르지 않고, 한 칸씩 뒤로 이동하며 빈 칸을 찾는다.

사용할 공식

해시 함수는 h(k)=k 11이다.
이 공식을 쓰는 이유: 각 키의 첫 후보 주소를 먼저 구해야 하고, 그 주소가 이미 차 있으면 선형 조사법으로 다음 주소를 순서대로 확인해야 하기 때문이다.

NoteBase

계산 과정

1. 키 4는 4 11=4이므로 주소 4에 저장된다.
2. 키 6은 6 11=6이므로 주소 6에 저장된다.
3. 키 15는 15 11=4인데 주소 4가 이미 차 있으므로 다음 칸인 주소 5에 저장된다.
4. 키 26은 26 11=4이다. 주소 4는 키 4, 주소 5는 키 15, 주소 6은 키 6이 차지하고 있으므로 다음 빈 칸인 주소 7에 저장된다.
5. 따라서 키 26의 최종 저장 주소는 7이다.

선지별 해설

① X: 1은 26 11의 결과도 아니고, 선형 조사 결과로도 나오지 않는다. 틀린 표현은 최종 저장 주소를 1로 본 점이며, 올바르게는 주소 7이다.
② X: 4는 키 26의 최초 해시 주소일 뿐 실제 저장 주소가 아니다. 주소 4에는 이미 키 4가 들어 있으므로 충돌이 발생한다. 틀린 표현은 충돌 후 이동을 반영하지 않은 점이며, 올바르게는 주소 7이다.
③ X: 6은 선형 조사 도중 지나가는 칸이지만 이미 키 6이 저장되어 있어 사용할 수 없다. 틀린 표현은 점유된 칸을 최종 저장 위치로 본 점이며, 올바르게는 주소 7이다.
④ O: 키 26의 초기 주소는 4이고, 4→5→6이 모두 차 있으므로 선형 조사법에 따라 다음 빈 칸 7에 저장된다.

오답 포인트

이 문항의 함정은 해시값과 최종 저장 주소를 혼동하는 점이다. 특히 ②처럼 26 11=4만 계산하고 끝내면 충돌 처리 과정을 놓치게 된다. ③도 선형 조사 중간에 만나는 주소가 비어 있는지 확인하지 않으면 고르기 쉬운 오답이다.

다시 나오면 이렇게 판단

해시 충돌 문제는 각 키의 초기 해시 주소를 먼저 적고, 충돌이 나면 한 칸씩 이동하면서 이미 저장된 값이 있는지 순서대로 확인해 최종 빈 칸을 찾는 방식으로 판단한다.

암기 포인트

선형 조사법은 해시값이 아니라 충돌 뒤 처음 만나는 빈 칸이 최종 저장 주소이다.

NoteBase

문 9. 다음은 유클리드 알고리즘(euclidean algorithm)을 사용하여 두 정수의 최대공약수(greatest common divisor)를 구하는 파이썬 프로그램이다. gcd(12, 21)를 실행한 후 count 값과 (가)에 들어갈 내용을 바르게 연결한 것은?

```
def gcd(m, n):
    global count
    count = count + 1
    if m < n:
        temp = m
        m = n
        n = temp
    if m % n == 0:
        return n
    else:
        return gcd(  (가)  )

count = 0
gcd(12, 21)
```

① count: 2, (가): m, m % n

② count: 2, (가): n, n % m

③ count: 3, (가): m, n % m

④ count: 3, (가): n, m % n

정답 4번

출제 포인트

이 문제는 유클리드 알고리즘의 재귀 호출 인자와 호출 횟수를 추적해 최대공약수를 구하는 과정을 묻는 문제이다.

조건 정리

주어진 호출은 gcd(12, 21)이다.
함수에 들어올 때마다 count가 1씩 증가한다.
먼저 m<n이면 두 값을 교환한 뒤, 나누어떨어지지 않으면 gcd(n, m % n)으로 재귀 호출해야 한다.

NoteBase

사용할 공식

유클리드 알고리즘의 재귀식은 gcd(m, n)=gcd(n, m % n)이다.
이 공식을 쓰는 이유는 큰 수를 작은 수로 나눈 나머지와 작은 수의 최대공약수가 원래 두 수의 최대공약수와 같기 때문이다.

계산 과정

첫 호출 gcd(12, 21)에서 m<n이므로 값을 바꾸면 gcd(21, 12) 상태가 된다. 이때 count는 1이다.
21 % 12 = 9이므로 다음 호출은 gcd(12, 9)이다. 이때 count는 2이다.
12 % 9 = 3이므로 다음 호출은 gcd(9, 3)이다. 이때 count는 3이다.
9 % 3 = 0이므로 3을 반환하고 종료된다.

선지별 해설

① X
count가 2라는 부분이 틀리다. 실제로는 gcd(21, 12) → gcd(12, 9) → gcd(9, 3)까지 세 번 호출되므로 count는 3이다.
또한 (가)를 m, m % n으로 두면 첫 재귀 호출이 gcd(21, 9)가 되어 작은 수를 첫 번째 인자로 넘기는 유클리드 알고리즘의 표준 재귀식과 다르다. 틀린 표현은 "m, m % n"이고, 올바르게는 "n, m % n"이다.

② X
count가 2라는 부분이 틀리다. 실제 호출 횟수는 3회이다.
또한 (가)를 n, n % m으로 두면 m>=n으로 정리한 뒤에도 작은 수를 큰 수로 나눈 나머지를 쓰게 된다. 첫 단계에서 12 % 21 = 12이므로 감소가 일어나지 않아 알고리즘 진행 방식이 틀어진다. 틀린 표현은 "n % m"이고, 올바르게는 "m % n"이다.

③ X
count가 3이라는 점은 맞지만 (가)가 틀리다.
(가)를 m, n % m으로 두면 첫 단계에서 gcd(21, 12) 다음이 다시 gcd(21, 12)가 되어 같은 상태가 반복된다. 12 % 21 = 12이므로 나머지가 줄지 않아 종료로 진행하지 못한다. 틀린 표현은 "m, n % m"이고, 올바르게는 "n, m % n"이다.

④ O
유클리드 알고리즘은 큰 수와 작은 수를 정리한 뒤 gcd(n, m % n)으로 재귀 호출한다.
gcd(12, 21)은 gcd(21, 12) → gcd(12, 9) → gcd(9, 3)으로 진행되므로 count는 3이고, (가)는 n, m % n이 맞다.

NoteBase

오답 포인트

이 문항의 함정은 재귀 호출 직전에 m<n이면 값을 바꾼다는 점을 놓치거나, 나머지를 n % m으로 잘못 쓰는 데 있다. 유클리드 알고리즘은 항상 큰 수를 작은 수로 나눈 나머지로 줄여야 하므로 m % n의 방향이 핵심이다.

다시 나오면 이렇게 판단

재귀형 최대공약수 문제는 먼저 m과 n의 대소를 정리하고, 다음 호출이 gcd(작은 수, 큰 수를 작은 수로 나눈 나머지) 꼴인지 확인한다. 그다음 호출 횟수를 한 단계씩 적어 count를 계산하면 된다.

암기 포인트

유클리드 알고리즘의 핵심은 "앞에는 작은 수, 뒤에는 큰 수를 작은 수로 나눈 나머지"이다. 재귀식은 gcd(n, m % n)으로 기억하면 된다.

NoteBase

문 10. 배열(array)로 표현한 이진 트리 중에서 최소 힙(minimum heap)에 해당하지 않는 것은?

① | 1 | 3 | 13 | 7 | 9 | 10 | 20 | 17 | 25 |

② | 1 | 3 | 13 | 9 | 7 | 20 | 17 | 25 | 10 |

③ | 1 | 3 | 17 | 7 | 13 | 25 | 20 | 9 | 10 |

④ | 1 | 9 | 3 | 17 | 10 | 13 | 7 | 20 | 25 |

정답 1번

출제 포인트

배열로 저장된 완전이진트리에서 부모 인덱스 i와 자식 인덱스 $2i$, $2i+1$를 대응시켜 최소 힙의 부모 $\leq$ 자식 조건을 판별하는 문제이다.

핵심 개념

최소 힙은 완전이진트리이면서 모든 부모 노드의 키가 각 자식 노드의 키보다 작거나 같은 구조이다.
배열 표현에서는 인덱스 i의 자식이 $2i$, $2i+1$이므로 각 부모-자식 쌍만 확인하면 된다.

NoteBase

선지별 해설

① O
배열 [1, 3, 13, 7, 9, 10, 20, 17, 25]에서 인덱스 3의 값 13의 자식은 인덱스 6의 10과 인덱스 7의 20이다. 최소 힙은 부모가 자식보다 작거나 같아야 하는데 13 > 10이므로 최소 힙 조건을 위반한다. 따라서 이 배열은 최소 힙에 해당하지 않는다.
틀린 표현은 없다. 이 선택지는 발문의 조건을 정확히 만족한다.

② X
인덱스 1의 1은 3, 13보다 작고, 인덱스 2의 3은 9, 7보다 작으며, 인덱스 3의 13은 20, 17보다 작다. 인덱스 4의 9도 25, 10보다 작으므로 모든 부모-자식 쌍이 최소 힙 조건을 만족한다.
틀린 판단은 '최소 힙에 해당하지 않는다'이다. 올바르게는 '최소 힙에 해당한다'이다.

③ X
인덱스 1의 1, 인덱스 2의 3, 인덱스 3의 17, 인덱스 4의 7이 각각 자신의 자식보다 작거나 같으므로 최소 힙 조건을 모두 만족한다. 형제 노드 사이의 크기 순서는 최소 힙 판정 기준이 아니다.
틀린 판단은 '최소 힙에 해당하지 않는다'이다. 올바르게는 '최소 힙에 해당한다'이다.

④ X
루트 1은 9와 3보다 작고, 9는 17과 10보다 작으며, 3은 13과 7보다 작다. 17도 20과 25보다 작으므로 부모가 자식보다 큰 경우가 없다. 따라서 최소 힙 조건을 만족한다.
틀린 판단은 '최소 힙에 해당하지 않는다'이다. 올바르게는 '최소 힙에 해당한다'이다.

오답 포인트

③과 ④에서 같은 레벨의 값이 정렬되어 있지 않다는 이유로 힙이 아니라고 오해하기 쉽다. 최소 힙은 형제 간 대소관계나 레벨 전체의 정렬을 요구하지 않고, 각 부모와 자식 사이의 대소관계만 본다.

다시 나오면 이렇게 판단

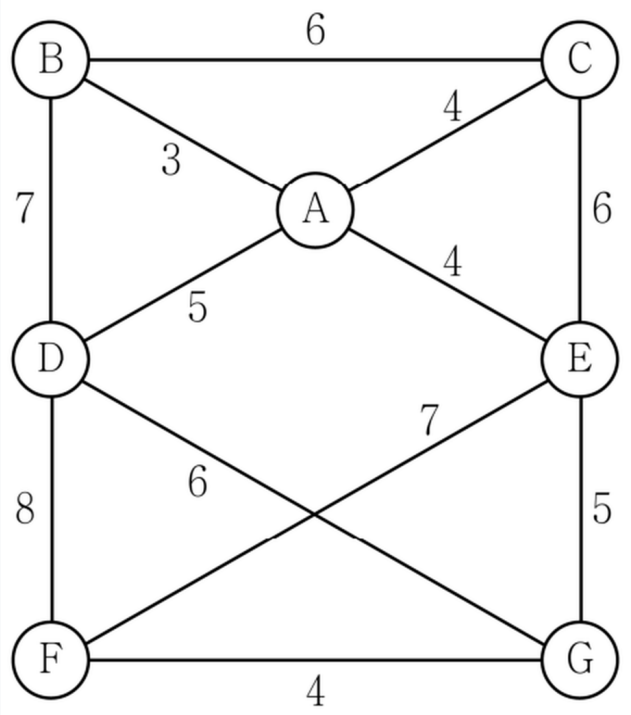
배열 문제는 트리 모양을 다시 그리기보다 인덱스 i마다 자식 $2i$, $2i+1$를 대입해 부모 $\leq$ 자식인지 순서대로 확인하면 된다. 한 쌍이라도 부모 > 자식이면 최소 힙이 아니다.

암기 포인트

최소 힙은 '부모만 본다'가 핵심이다. 형제끼리의 순서는 상관없고 부모 $\leq$ 자식만 지키면 된다.

NoteBase

문 11. 다음 그래프에서 크루스칼 알고리즘(Kruskal algorithm)을 사용하여 만든 최소 비용 신장 트리(minimum cost spanning tree)에 대한 모든 간선의 가중치 합은?



① 21

② 25

③ 31

④ 33

정답 2번

출제 포인트

이 문제는 크루스칼 알고리즘으로 사이클을 만들지 않는 가장 작은 간선부터 선택하여 최소 신장 트리의 총 가중치를 구할 수 있는지를 묻는다.

조건 정리

정점은 A, B, C, D, E, F, G의 7개이므로 최소 신장 트리에는 간선이 정확히 6개 필요하다.
가중치가 작은 순서로 보면 AB=3, AC=4, AE=4, FG=4, AD=5, EG=5, BC=6, CE=6, DG=6, BD=7, FE=7, DF=8이다.
크루스칼 알고리즘은 이 순서대로 보면서 사이클이 생기지 않는 간선만 채택한다.

NoteBase

사용할 공식

사용 원리는 '가중치가 가장 작은 간선부터 확인하되, 이미 선택한 간선들과 합쳐 사이클이 생기면 버리고, 모든 정점이 연결될 때까지 반복한다'는 크루스칼 알고리즘이다.
이 문제는 수치를 직접 계산하는 공식보다 간선 선택 규칙으로 최소 합을 만드는 경우이므로 이 원리를 그대로 적용한다.

계산 과정

1. 가장 작은 간선 AB(3)을 선택한다.
2. 다음으로 작은 AC(4), AE(4), FG(4)는 모두 사이클을 만들지 않으므로 선택한다.
3. 가중치 5인 AD(5)를 넣으면 D가 연결되고, EG(5)를 넣으면 {A,B,C,D,E}와 {F,G}가 하나로 연결된다.
4. 이 시점에 선택한 간선은 AB, AC, AE, FG, AD, EG의 6개이며 7개 정점이 모두 연결된다.
5. 총합은 3+4+4+4+5+5=25이다.

선지 판단

1번 X: 21는 7개 정점을 잇는 최소 신장 트리의 6개 간선을 모두 채운 합이 될 수 없다. 작은 간선만 보고 연결 완료 전에 합산을 멈춘 값에 가깝다. 최소 신장 트리의 총합은 25이므로 '21'이 아니라 '25'가 옳다.
2번 O: 크루스칼 알고리즘으로 AB(3), AC(4), AE(4), FG(4), AD(5), EG(5)를 선택하면 모든 정점이 연결되고 총합이 25가 된다. 따라서 정답에 해당한다.
3번 X: 31은 가중치 6 또는 7인 큰 간선을 불필요하게 포함한 경우 나오는 과대 계산값이다. 이미 가중치 5의 AD와 EG로 연결이 끝나므로 더 큰 간선을 넣을 필요가 없다. 옳은 값은 25이다.
4번 X: 33은 사이클 검사 없이 큰 간선을 추가하거나 최소 신장 트리 완성 뒤에도 간선을 더한 경우의 값이다. 크루스칼 알고리즘은 6개 간선에서 멈추므로 총합은 33이 아니라 25이다.

오답 포인트

가장 흔한 실수는 작은 간선을 고른 뒤에도 '사이클 여부'와 '모든 정점 연결 완료 여부'를 끝까지 점검하지 않는 것이다. 특히 이 문제는 가중치 4의 간선 3개를 고른 뒤, 5인 AD와 EG만 추가하면 이미 신장 트리가 완성되는데 6이나 7짜리 간선을 더 넣기 쉽다.

다시 나오면 이렇게 판단

크루스칼 문제는 가중치 오름차순 정렬 -> 사이클이 없으면 선택 -> 간선 수가 정점 수보다 1개 적어지면 종료의 순서로 판단한다. 선택한 간선이 7개 정점 기준 6개가 되는 순간 더 큰 간선은 보지 않아도 된다.

암기 포인트

크루스칼은 '작은 것부터, 사이클은 제외, 간선 수는 $n-1$에서 종료'로 정리된다.

NoteBase

문 12. 다음 treetest() 함수로 만들어지는 트리의 유형은?

```
typedef struct treeNode {
    int data;
    struct treeNode *left, *right;
} treeNode;

void treetest() {
    treeNode n1 = { 12, NULL, NULL };
    treeNode n2 = { 33, NULL, NULL };
    treeNode n3 = { 21, &n1, &n2 };
    treeNode n4 = { 74, NULL, NULL };
    treeNode n5 = { 67, NULL, &n4 };
    treeNode n6 = { 59, NULL, &n5 };
    treeNode n7 = { 46, &n3, &n6 };
    treeNode* root = &n7;
}
```

① 스레드 이진 트리(threaded binary tree)

② 완전 이진 트리(complete binary tree)

③ 이진 탐색 트리(binary search tree)

④ 포화 이진 트리(full binary tree)

정답 3번

출제 포인트

이 문제는 주어진 연결 구조가 어떤 이진트리 성질을 만족하는지 판별하는 문제이다.

핵심 개념

스레드 이진 트리는 NULL 포인터를 순회용 링크로 활용한 트리이다.
완전 이진 트리는 마지막 레벨을 제외한 모든 레벨이 가득 차고 마지막 레벨도 왼쪽부터 채워진 트리이다.
이진 탐색 트리는 각 노드에서 왼쪽 서브트리의 모든 값이 작고 오른쪽 서브트리의 모든 값이 큰 트리이다.
포화 이진 트리는 각 내부 노드가 두 자식을 가지며 각 레벨이 빈칸 없이 채워진 형태로 이해하는 트리이다.

NoteBase

선지별 해설

1번 X: 틀린 부분은 "스레드 이진 트리"이다. 코드에는 left와 right가 일반 자식 포인터로만 쓰였고 NULL을 중위순회 전후 노드로 연결한 흔적이 없다. 올바르게는 스레드 처리가 없는 일반 이진 연결 구조이다.
2번 X: 틀린 부분은 "완전 이진 트리"이다. 완전 이진 트리라면 마지막 레벨의 노드가 왼쪽부터 연속해서 채워져야 하는데, 이 트리는 59와 67이 왼쪽 자식 없이 오른쪽으로만 이어져 중간 빈칸이 생긴다. 올바르게는 완전 이진 트리가 아니다.
3번 O: 루트 46을 기준으로 왼쪽 서브트리는 21, 12, 33으로 모두 46보다 작고, 오른쪽 서브트리는 59, 67, 74로 모두 46보다 크다. 또한 21의 왼쪽 12는 21보다 작고 오른쪽 33은 21보다 크며, 59의 오른쪽 67, 67의 오른쪽 74도 부모보다 커서 이진 탐색 트리 조건을 재귀적으로 만족한다.
4번 X: 틀린 부분은 "포화 이진 트리"이다. 포화 이진 트리라면 내부 노드가 두 자식을 갖고 각 레벨이 가득 차야 하는데, 59와 67은 자식이 하나뿐이고 전체 레벨도 꽉 차 있지 않다. 올바르게는 이 트리는 포화 이진 트리가 아니라 이진 탐색 트리이다.

오답 포인트

오른쪽으로 59-67-74가 정렬되어 있다는 점만 보고 편향된 모양이라서 탐색 트리가 아니라고 오해하기 쉽다. 이진 탐색 트리는 모양이 아니라 각 노드의 대소 관계로 판단하며, 완전 이진 트리와 포화 이진 트리는 모양 조건까지 함께 본다.

다시 나오면 이렇게 판단

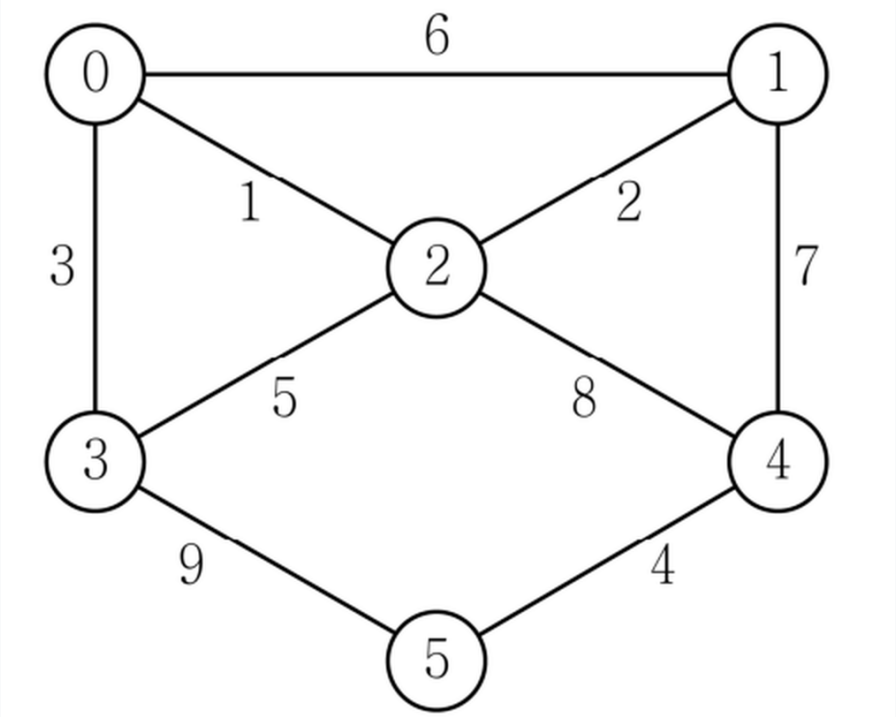
트리 유형 문제는 먼저 값의 대소 관계로 이진 탐색 트리 여부를 보고, 그다음 레벨이 왼쪽부터 채워지는지로 완전 여부를, NULL 링크를 순회용으로 바꿨는지로 스레드 여부를 구분하면 된다.

암기 포인트

BST는 값의 순서, 완전·포화는 모양, 스레드는 NULL 링크 활용이 핵심이다.

NoteBase

문 13. 다음 그래프에서 프림 알고리즘(Prim algorithm)을 사용하여 최소 비용 신장 트리를 만들 때, 트리에 네 번째로 포함되는 간선은? (단, 시작 정점은 0이다)



① (0, 1)

② (1, 4)

③ (2, 3)

④ (4, 5)

정답 2번

출제 포인트

이 문제는 시작 정점 0에서 Prim 알고리즘을 적용할 때 현재 트리와 트리 밖 정점을 잇는 간선들 중 최솟값만 순서대로 고르는 규칙을 그래프에서 읽어내는 문제이다.

NoteBase

선지별 해설

① X
(0,1)은 가중치가 6이지만 1은 두 번째 단계에서 더 작은 간선 (2,1)로 이미 트리에 포함된다. 그 뒤의 (0,1)은 트리 내부 정점끼리 잇는 간선이어서 선택되지 않는다. 틀린 표현은 네 번째 포함 간선이라는 점이며, 올바르게는 (0,1)은 MST에 포함되지 않는다.

② O
선택 순서는 (0,2)→(2,1)→(0,3)이다. 이 상태에서 트리 밖 정점으로 나가는 후보는 (1,4)=7, (2,4)=8, (3,5)=9이고 최솟값은 (1,4)이므로 네 번째 간선은 (1,4)이다.

③ X
정점 3이 트리에 들어오는 단계에서 후보인 (0,3)=3과 (2,3)=5를 비교하면 더 작은 것은 (0,3)이다. 틀린 표현은 3이 2를 통해 연결된다는 점이며, 올바르게는 3은 세 번째 단계에서 (0,3)으로 포함된다.

④ X
(4,5)는 가중치가 4로 작아 보이지만 초반에는 두 끝점 4와 5가 모두 트리 밖에 있으므로 Prim 알고리즘의 후보 간선이 아니다. 4가 (1,4)로 들어온 다음에야 선택 가능해지므로 (4,5)는 다섯 번째 간선이다. 틀린 표현은 네 번째 포함 간선이라는 점이며, 올바르게는 (4,5)는 마지막에 포함된다.

오답 포인트

가중치 4인 (4,5)만 보고 바로 고르면 Prim 규칙을 놓치게 된다. Prim은 전체 그래프의 최소 간선을 고르는 알고리즘이 아니라 현재 트리에서 바깥으로 뻗는 간선 중 최소를 고르는 알고리즘이므로, 후보 집합이 단계마다 달라진다는 점이 함정이다.

다시 나오면 이렇게 판단

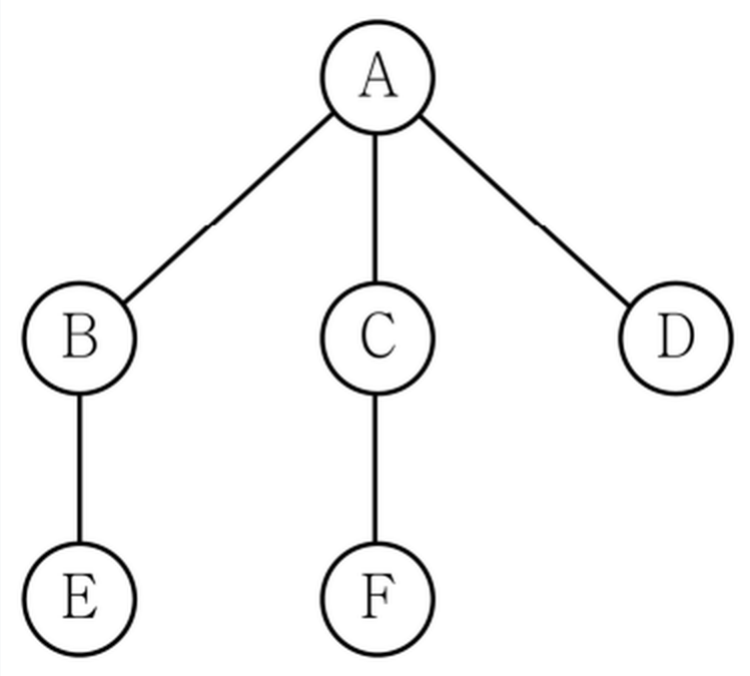
매 단계마다 이미 포함된 정점 집합을 먼저 표시한 뒤, 그 집합과 바깥 정점을 연결하는 간선만 후보로 남겨 가장 작은 가중치를 고르면 된다. 트리 안쪽 간선과 트리 밖끼리만 잇는 간선은 바로 제외된다.

암기 포인트

Prim은 현재 트리 경계선에서만 최소 간선을 고르는 확장형 알고리즘이다.

NoteBase

문 14. 다음 그래프에서 정점 A부터 너비 우선 탐색(BFS, breadth first search)을 사용하여 나올 수 있는 결과로 옳지 않은 것은?



① A B C D E F

② A C B D F E

③ A C F B E D

④ A D C B F E

정답 3번

출제 포인트

이 문제는 그래프 도식에서 BFS가 시작 정점으로부터 같은 깊이의 정점을 먼저 모두 방문하고, 그 다음 깊이의 정점으로 이동한다는 레벨 순서를 판단하는 문제이다.

NoteBase

선지별 해설

① O
A의 인접 정점을 B, C, D 순서로 큐에 넣으면 방문 순서는 A B C D가 되고, 이후 B의 자식 E, C의 자식 F가 차례로 처리되어 A B C D E F가 될 수 있다.

② O
A의 인접 정점을 C, B, D 순서로 넣으면 깊이 1 정점 방문 순서가 A C B D가 된다. 그 다음 C에서 F, B에서 E가 추가되어 A C B D F E가 될 수 있다.

③ X
F는 C의 자식이므로 깊이 2 정점이다. BFS에서는 깊이 1 정점인 B, C, D를 모두 방문한 뒤에야 깊이 2 정점 E, F를 방문하므로 A C F B E D는 나올 수 없다. 틀린 표현은 F가 B, D보다 먼저 나온 부분이며, 올바르게는 A C B D F E처럼 깊이 1 정점을 모두 처리한 다음 F가 나와야 한다.

④ O
A의 인접 정점을 D, C, B 순서로 넣으면 깊이 1 정점 방문 순서는 A D C B가 된다. 이후 C의 자식 F, B의 자식 E가 순서대로 큐에 들어가 A D C B F E가 될 수 있다.

오답 포인트

BFS에서 시작 정점의 인접 정점 순서는 바뀔 수 있다는 점만 보고 ③도 가능하다고 착각하기 쉽다. 그러나 인접 정점의 순서는 바뀌어도 깊이 2 정점이 깊이 1 정점보다 먼저 방문되는 일은 없다.

다시 나오면 이렇게 판단

BFS 순서를 판단할 때는 먼저 정점을 깊이별로 묶는다. 시작 정점 다음에는 깊이 1 정점만 나와야 하고, 그 집합이 모두 끝난 뒤에야 깊이 2 정점이 나올 수 있다.

암기 포인트

BFS는 큐 기반 레벨 탐색이므로 같은 깊이를 먼저, 더 깊은 정점은 나중에 방문한다.

NoteBase

문 15. 다음 C 프로그램의 실행 결과는?

```
#include <stdio.h>

void func(int *array, int size) {
  int *p = array;
  *p = 100;
  *(p + 2) = 100;
  p = p + 1;
}

int main(int argc, char *argv[]) {
  int array[5] = { 2, 4, 6, 8, 10 };
  func(array, 5);
  for(int i = 0; i < 5; i++)
    printf("%d ", array[i]);
  printf("\n");
  return 0;
 }
```

① 100 4 6 8 100

② 100 4 6 100 10

③ 100 4 100 8 10

④ 100 100 68 8 10

정답 3번

출제 포인트

이 문제는 배열이 함수에 전달될 때 원본 첫 원소의 주소가 넘어가며, 포인터 연산으로 어느 인덱스가 실제로 바뀌는지를 추적해 출력 결과를 구하는 문제이다.

조건 정리

초기 배열은 [2, 4, 6, 8, 10]이다.
p는 처음에 array의 첫 번째 원소를 가리킨다.
*p = 100은 첫 번째 원소를 100으로 바꾼다.
*(p + 2) = 100은 세 번째 원소를 100으로 바꾼다.
p = p + 1은 포인터 위치만 다음 원소로 이동시키며 배열 값은 더 바꾸지 않는다.

NoteBase

계산 과정

1. 시작 배열은 [2, 4, 6, 8, 10]이다.
2. *p = 100 실행 후 배열은 [100, 4, 6, 8, 10]이 된다.
3. *(p + 2) = 100에서 p는 첫 번째 원소를 가리키므로 p + 2는 세 번째 원소를 가리킨다. 따라서 배열은 [100, 4, 100, 8, 10]이 된다.
4. p = p + 1은 포인터 자체만 두 번째 원소 쪽으로 옮길 뿐 출력 전에 추가 대입이 없으므로 배열은 그대로 [100, 4, 100, 8, 10]이다.

선지별 해설

① X: 마지막 값 10은 어떤 대입문으로도 바뀌지 않는다. 틀린 표현은 마지막 값 100이며, 올바르게는 10이다.
② X: *(p + 2)는 네 번째 원소가 아니라 세 번째 원소를 바꾼다. 틀린 표현은 네 번째 값 100이며, 올바르게는 세 번째 값이 100이고 네 번째 값은 8이다.
③ O: 첫 번째 원소는 *p = 100으로, 세 번째 원소는 *(p + 2) = 100으로 바뀌고 나머지 원소는 그대로이므로 결과는 100 4 100 8 10이다.
④ X: p = p + 1은 값을 바꾸지 않고 가리키는 위치만 이동시킨다. 따라서 두 번째 값이 100이 될 수 없고 68이라는 값도 이 코드에서 만들어지지 않는다. 올바르게는 100 4 100 8 10이다.

오답 포인트

많이 틀리는 지점은 p + 2를 현재 줄에서 두 칸 이동한 결과가 아니라 처음 p가 가리키던 첫 원소 기준의 세 번째 원소로 보아야 한다는 점이다. 또 p = p + 1을 값 변경으로 착각하면 두 번째 원소가 바뀐다고 오해하게 된다.

다시 나오면 이렇게 판단

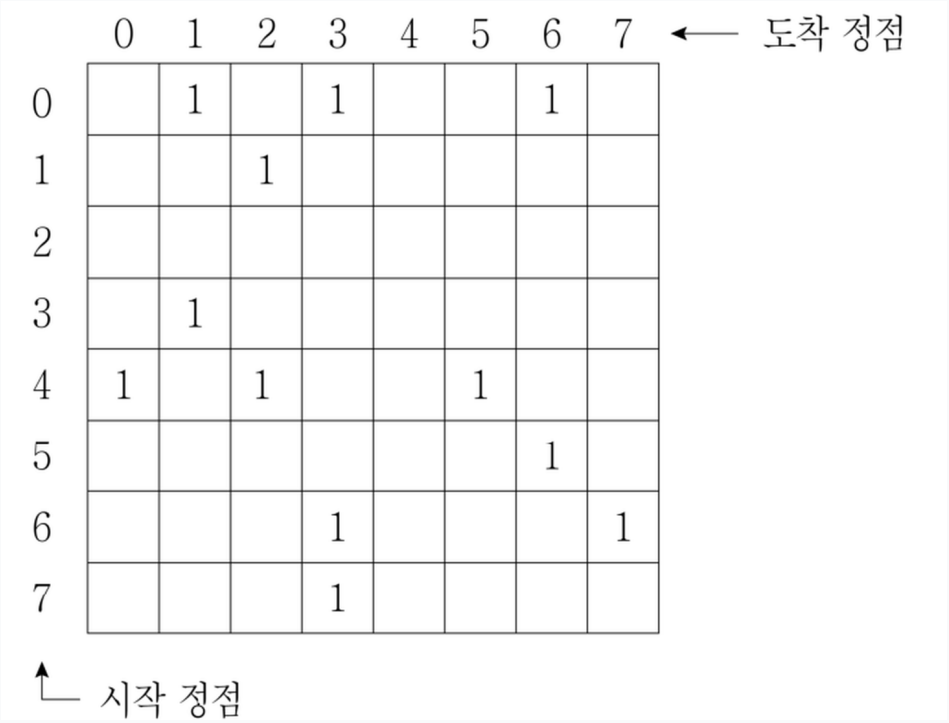
포인터 추적 문제는 각 문장마다 가리키는 위치와 실제 대입문인지 여부를 분리해서 본다. 대입문이면 해당 인덱스의 값만 갱신하고, 포인터 이동문이면 배열 값은 그대로 둔 채 다음 줄 해석 기준만 바꾼다.

암기 포인트

배열을 함수에 넘기면 원본이 바뀌고, 포인터를 이동해도 대입이 없으면 배열 값은 바뀌지 않는다.

NoteBase

문 16. 다음 인접 행렬(adjacency matrix)로 표현된 그래프에 대해 위상 정렬(topological sort)을 수행한 후, 위상 순서(topological order)에서 마지막에 위치하는 정점은? (단, 위상 정렬 과정에서 선택 가능한 정점이 여러 개인 경우 가장 작은 숫자의 정점을 선택한다)



① 1
② 2
③ 3
④ 6

정답 2번

출제 포인트

이 문제는 인접 행렬에서 각 정점의 진입 차수를 계산한 뒤, 진입 차수가 0인 정점을 반복 제거하는 위상 정렬의 선택 순서를 묻는 문제이다.

NoteBase

올바른 순서

1. 4 - 처음에 진입 차수가 0인 유일한 정점이다.
2. 0 - 4를 제거하면 0과 5의 진입 차수가 0이 되는데, 더 작은 0을 먼저 선택한다.
3. 5 - 남은 진입 차수 0 정점 중 가장 작다.
4. 6 - 5를 제거하면 6의 진입 차수가 0이 된다.
5. 7 - 6을 제거하면 7의 진입 차수가 0이 된다.
6. 3 - 7을 제거하면 3의 진입 차수가 0이 된다.
7. 1 - 3을 제거하면 1의 진입 차수가 0이 된다.
8. 2 - 1을 제거한 뒤 마지막으로 진입 차수가 0이 된다.

왜 이 순서: 앞 정점의 제거가 뒤 정점의 진입 차수를 0으로 만들어야 다음 선택이 가능하므로, 2는 1과 4가 모두 빠진 뒤에야 선택할 수 있다.

선지별 해설

① X: 정점 1은 정점 0과 3에서 들어오는 간선을 받으므로 초반에 선택될 수 없다. 실제 순서에서는 3이 제거된 뒤에야 진입 차수가 0이 되므로 마지막 정점이 아니다. 틀린 표현은 '마지막 정점이 1'이라는 판단이며, 올바르게는 1은 끝에서 두 번째에 위치한다.

② O: 정점 2는 정점 1과 4에서 들어오는 간선을 받는다. 4가 먼저 제거된 뒤에도 1이 남아 있으므로 바로 선택되지 못하고, 1이 제거된 다음에야 진입 차수가 0이 되어 최종적으로 마지막에 놓인다.

③ X: 정점 3은 정점 0, 6, 7에서 들어오는 간선을 받지만, 3에서 1로 나가는 간선이 있으므로 3이 마지막이면 1을 그 뒤에 둘 수 없다. 틀린 표현은 '마지막 정점이 3'이라는 판단이며, 올바르게는 3은 7 다음에 오고 1보다 먼저 제거된다.

④ X: 정점 6은 정점 0과 5에서 들어오는 간선을 받는다. 0과 5가 제거되면 비교적 이른 시점에 진입 차수가 0이 되며, 이후 7과 3의 선행 정점 역할을 한다. 틀린 표현은 '마지막 정점이 6'이라는 판단이며, 올바르게는 6은 중간 단계에서 제거된다.

오답 포인트

행이 시작 정점이고 열이 도착 정점이라는 방향을 반대로 읽으면 진입 차수 계산이 뒤집힌다. 또 진입 차수가 0인 정점이 여러 개 생길 때 가장 작은 번호를 먼저 고른다는 조건을 놓치면 0과 5의 순서를 바꾸어 잘못 판단하기 쉽다.

다시 나오면 이렇게 판단

위상 정렬 문제는 먼저 각 열의 진입 차수를 세고, 진입 차수 0 정점을 하나 제거한 뒤 새로 0이 되는 정점을 갱신하는 방식으로 끝까지 추적하면 된다. 동률 조건이 있으면 그 조건까지 매 단계 함께 적용해야 마지막 정점을 정확히 찾을 수 있다.

NoteBase

암기 포인트

위상 정렬은 진입 차수 0 제거의 반복이며, 동률이면 지정된 우선순위를 끝까지 유지한다.

NoteBase

문 17. 다음 C 코드에 10개의 노드로 이루어진 이중 연결 리스트를 입력하여 수행한 후, 이중 연결 리스트에 남아 있는 노드의 개수는? (단, 10개 노드 입력 후 수행 시 start는 첫 번째 노드를 가리킨다)

```
typedef struct node {
  int key;
  struct node *prev, *next;
} node;

void foobar(node *start) {
    node *current = start;
    int iseven = 0;
    while(current != NULL) {
      if(iseven == 1) {
        current->prev->next = current->next;
        if(current->next != NULL)
          current->next->prev = current->prev;
      }
      current = current->next;
      iseven = 1 - iseven;
    }
}
```

① 5

② 6

③ 8

④ 10

정답 1번

출제 포인트

이 문제는 이중 연결 리스트를 순회하면서 `iseven` 값이 1일 때 현재 노드를 연결에서 제외하는 과정과, 그 결과 남는 노드 수를 묻는 문제이다.

NoteBase

올바른 순서

1. `current`는 첫 번째 노드에서 시작하고 `iseven`은 0이므로 첫 노드는 삭제되지 않는다.
2. 한 칸 이동할 때마다 `iseven`이 0과 1로 번갈아 바뀌므로 두 번째, 네 번째, 여섯 번째처럼 짝수 번째 방문에서 삭제 조건이 성립한다.
3. 삭제가 일어나면 `current->prev->next = current->next`와 `current->next->prev = current->prev`로 현재 노드를 리스트 연결에서 뺀다.
4. 그 뒤 `current = current->next`로 다음 노드로 진행하므로 삭제된 노드 다음 노드부터 계속 순회한다.

왜 이 순서인가: 삭제는 현재 노드의 앞뒤 포인터만 다시 연결하고 순회 포인터는 기존 `current->next`로 이동하므로, 결과적으로 2, 4, 6, 8, 10번째 노드가 차례로 빠진다.

선지별 해설

① O
- 10개 노드에서 짝수 번째 노드 5개가 제거되므로 1, 3, 5, 7, 9만 남아 총 5개가 된다.

② X
- 틀린 부분: 삭제가 4번만 일어난다고 본 판단이 틀리다.
- 올바르게는: 마지막 10번째 노드도 `iseven == 1` 상태에서 삭제되므로 남는 노드는 6개가 아니라 5개이다.

③ X
- 틀린 부분: 일부 노드만 건너뛴다고 보고 2개만 삭제된 것처럼 계산한 점이 틀리다.
- 올바르게는: 순회는 끝까지 진행되고 짝수 번째 방문마다 삭제가 반복되므로 8개가 아니라 5개가 남는다.

④ X
- 틀린 부분: 코드가 순회만 하고 삭제하지 않는다고 본 해석이 틀리다.
- 올바르게는: `prev->next`와 `next->prev`를 갱신해 실제로 현재 노드를 리스트에서 제거하므로 10개가 그대로 남지 않는다.

오답 포인트

삭제 뒤에 `current = current->next`를 수행하므로 노드를 하나 더 건너뛴다고 착각하기 쉽다. 그러나 삭제된 노드의 `next`는 원래 다음 노드를 그대로 가리키므로 순회는 1, 2, 3, 4, ... 순서대로 방문하고, 그중 짝수 번째 방문 노드만 리스트에서 빠진다.

다시 나오면 이렇게 판단

연결 리스트 삭제 코드는 `누가 방문되는가`와 `누가 연결에서 빠지는가`를 분리해서 본다. 시작 노드, 토글 변수 변화, 삭제 시 앞뒤 포인터 재연결, 삭제 뒤 이동 대상 순서로 추적하면 남는 노드 수를 안정적으로 판단할 수 있다.

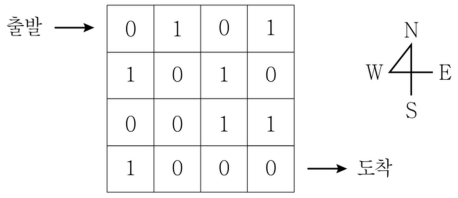
NoteBase

암기 포인트

현재 노드를 지우고도 순회는 `current->next`로 이어지므로 이 코드는 짝수 번째 노드를 차례로 제거해 절반인 5개를 남긴다.

NoteBase

문 18. 다음 미로 찾기 문제를 스택(stack)을 이용하여 해결하고자 한다. 출발점에서 시작하여 도착점까지 가기 위해 필요한 push 연산 횟수와 pop 연산 횟수는? (단, 출발점, 도착점 좌표에 대해서는 push 연산 횟수에서 제외한다)



○ 출발점은 (0, 0), 도착점은 (3, 3)이다.
○ 출발점에서 현재 좌표까지 이동 경로는 push 연산으로 현재 좌표를 스택에 저장하고, 직전 좌표로 돌아가는 경로는 pop 연산으로 현재 좌표를 스택에서 삭제하는 방법으로 만들어진다.
○ 미로의 크기는 4×4 2차원 배열로 구성되었고, 1은 장벽, 0은 통과할 수 있는 길이다. 4×4 2차원 배열 외부는 모두 장벽으로 간주하여 이 배열을 벗어날 수 없다고 가정한다.
○ 각 좌표에서 이전에 이동했던 좌표를 제외하고, N(북)에서 NE(북동), E(동), SE(남동), S(남), SW(남서), W(서), NW(북서) 순서로 검색하여 가장 먼저 찾은 이동 가능한 좌표로 이동한다. 주위 좌표가 모두 벽이거나 방문했던 곳일 경우 직전 좌표로 돌아간다.

① push: 3, pop: 0

② push: 4, pop: 0

③ push: 5, pop: 2

④ push: 6, pop: 2

정답 3번

출제 포인트

이 문제는 4×4 미로를 8방향 우선순위 N→NE→E→SE→S→SW→W→NW로 깊이 우선 탐색하면서, 출발점과 도착점을 제외한 중간 좌표의 push 횟수와 막다른 길에서의 pop 횟수를 정확히 세는 문제이다.

NoteBase

선지별 해설

① X
실제 탐색은 (0,0)→(1,1)→(0,2)→(1,3)까지 갔다가 막혀 두 번 되돌아온 뒤, (1,1)→(2,1)→(3,2)→(3,3)으로 도착한다. 출발점과 도착점을 제외하면 push 대상은 (1,1), (0,2), (1,3), (2,1), (3,2)로 5회이고, pop은 (1,3), (0,2)에서 2회이다. 틀린 표현은 "push 3, pop 0"이고, 올바르게는 "push 5, pop 2"이다.

② X
처음 경로인 (1,1)→(0,2)→(1,3)까지는 맞지만, (1,3)에서 더 진행할 수 없어 반드시 백트래킹이 발생한다. 따라서 pop이 0회일 수 없고, 이후 (2,1), (3,2)를 추가로 방문하므로 push도 4회가 아니라 5회이다. 틀린 표현은 "push 4, pop 0"이고, 올바르게는 "push 5, pop 2"이다.

③ O
우선순위대로 탐색하면 SE 방향의 (1,1)을 먼저 방문하고, 다시 NE의 (0,2), SE의 (1,3)으로 진행한다. (1,3)은 사방과 대각선이 모두 벽이거나 방문 좌표라 pop 1회가 발생하고, (0,2)도 남은 후보가 모두 벽 또는 방문 좌표라 pop 1회가 추가된다. 이후 (1,1)에서 S 방향의 (2,1), 다시 SE의 (3,2)를 거쳐 E 방향의 도착점 (3,3)에 도달하므로 push는 5회, pop은 2회가 된다.

④ X
도착점 (3,3)은 문제 조건상 push 횟수에서 제외한다. 따라서 중간 좌표를 모두 세어도 push는 6회가 아니라 5회이다. pop 2회는 맞지만 push 6회가 틀리다. 틀린 표현은 "push 6"이고, 올바르게는 "push 5"이다.

오답 포인트

출발점과 도착점을 push에서 제외한다는 조건을 놓치면 ④처럼 6회로 잘못 세기 쉽다. 또한 (1,3)에서 막힌 뒤 바로 도착 경로로 이어진다고 착각하면 백트래킹 2회를 놓쳐 ①, ②처럼 pop 0회로 오답을 고르기 쉽다.

다시 나오면 이렇게 판단

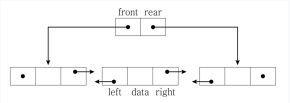
미로 스택 문제는 현재 칸에서 우선순위 방향을 하나씩 대입해 다음 좌표를 적고, 더 갈 곳이 없으면 직전 좌표로 돌아오는 pop을 별도로 표시하면서 추적한다. 마지막에 출발점과 도착점의 제외 조건을 다시 반영하면 push와 pop을 안정적으로 셀 수 있다.

암기 포인트

DFS 스택 추적은 "이동하면 push, 막히면 pop, 출발점·도착점은 예외"로 정리된다.

NoteBase

문 19. 다음은 이중 연결 리스트를 이용하여 덱(deque)을 구현한 파이썬 코드이다. (가) ~ (라)에 들어갈 내용을 A ~ F에서 바르게 연결한 것은?




```
class DQNode:
    def __init__(self, data, left, right):
        self.data = data
        self.left = left
        self.right = right

class DoublyLinkedDQ:
    def __init__(self):
        self.front = None
        self.rear = None

    def insertFront(self, data):
        node = DQNode(data, None, self.front)
        if self.front == None:
            self.front = node
            self.rear = self.front
        else:
            (가)
            (나)

    def deleteRear(self):
        if self.front != None:
            data = self.rear.data
            (다)
            if self.rear == None:
                (라)
            else:
                self.rear.right = None
            return data
```

A. self.front = node
B. self.front = self.rear
C. self.front.left = node
D. self.front.right = node
E. self.rear = self.rear.left
F. self.rear = self.rear.right

① (가): A, (나): C, (다): F, (라): A

② (가): C, (나): A, (다): E, (라): B

③ (가): C, (나): A, (다): F, (라): D

④ (가): D, (나): C, (다): E, (라): C

정답 2번

출제 포인트

이 문제는 이중 연결 리스트에서 앞쪽 삽입과 뒤쪽 삭제가 일어날 때 `front`, `rear`, `left`, `right` 포인터가 어느 방향으로 갱신되는지를 코드와 도식으로 함께 판단하는 문제이다.

NoteBase

선지별 해설

① X: `insertFront`에서 새 노드는 맨 앞에 붙으므로 기존 `front` 노드의 `left`가 새 노드를 가리켜야 한다. 따라서 `(가)=C`, `(나)=A`까지는 맞지만, `deleteRear`에서는 뒤 노드를 지울 때 먼저 `rear`를 한 칸 왼쪽으로 옮겨야 하므로 `(다)=E`이어야 한다. `①`의 `(다)=F`는 틀렸고, 올바른 조합은 `C-A-E-B`이다.
② O: `insertFront`의 비어 있지 않은 경우에는 기존 맨 앞 노드의 `left`를 새 노드로 연결한 뒤 `front`를 새 노드로 바꾸므로 `(가)=C`, `(나)=A`이다. `deleteRear`에서는 `rear=self.rear.left`로 한 칸 앞 노드로 이동하므로 `(다)=E`이고, 삭제 후 비게 되면 `front`도 `rear`와 같이 `None`이 되어야 하므로 `(라)=B`이다.
③ X: `(가)=C`, `(나)=A`는 맞지만 `(다)=F`가 틀리다. 뒤 노드를 삭제할 때 `rear`를 `rear.right`로 보내면 이미 끝 노드의 바깥쪽으로 이동하는 셈이어서 연결 구조가 맞지 않는다. 올바르게는 `(다)=E`이다.
④ X: 맨 앞 삽입에서는 기존 `front.right`가 아니라 기존 `front.left`를 새 노드로 연결해야 하므로 `(가)=D`는 틀리다. 또한 삭제 후 공백 deque 처리도 `front.left=node`가 아니라 `front=rear`를 통해 둘 다 `None`으로 맞춰야 하므로 `(라)=C`도 틀리다. 올바르게는 `(가)=C`, `(라)=B`이다.

오답 포인트

앞쪽 삽입이라고 해서 기존 `front`의 `right`를 바꾼다고 착각하기 쉽지만, 새 노드는 기존 `front`의 왼쪽에 붙으므로 바뀌는 것은 `front.left`이다. 또한 뒤쪽 삭제에서 마지막 노드를 지운 뒤에는 `rear`만 `None`으로 두면 안 되고 `front`도 함께 `None`이 되어야 한다는 점이 함정이다.

다시 나오면 이렇게 판단

끝점 삽입·삭제 문제는 먼저 새 노드가 어느 쪽 끝에 붙는지 또는 어느 끝에서 빠지는지를 정한 뒤, 이웃 노드의 반대편 링크를 연결한다. 그다음 끝 포인터를 한 칸 안쪽으로 이동시키고, 구조가 비면 `front`와 `rear`를 동시에 `None`으로 맞추면 된다.

암기 포인트

앞 삽입은 `기존 front.left 갱신 후 front 교체`, 뒤 삭제는 `rear를 left로 이동 후 비면 front도 함께 None`이다.

NoteBase

문 20. 다음 C 프로그램의 실행 결과는?

```c
#include <stdio.h>
typedef struct stack {
  int data[10]; int top;
} stack;
stack s1, s2;

void push(stack *ss, int i) {
  ss->top = ss->top + 1;
  ss->data[ss->top] = i;
}

int pop(stack *ss) {
  return ss->data[ss->top--];
}

int size(stack *ss) {
  return ((ss->top) + 1);
}

void push2(int n) {
  push(&s1, n);
}

int pop2() {
  if(size(&s2) == 0) {
    int s = size(&s1);
    for(int i = 0; i < s; i++)
      push(&s2, pop(&s1));
  }
  return pop(&s2);
}

int main(int argc, char **argv) {
  s1.top = s2.top = -1;
  push2(1);  push2(2);  push2(3);
  printf("%d  ", pop2());
  printf("%d  ", pop2());
  push2(4);  push2(5);
  printf("%d  ", pop2());
  printf("%d  ", pop2());
  printf("%d  ", pop2());
  return 0;
}
```

① 1 2 3 4 5

② 2 1 5 4 3

③ 3 2 5 4 1

④ 5 4 3 2 1

정답 1번

출제 포인트

이 문제는 두 개의 스택으로 큐를 구현할 때 `pop2()`가 `s2`가 비어 있을 때만 `s1`의 원소를 옮긴다는 흐름을 묻는 문제이다.

NoteBase

올바른 순서

1. `push2(1)`, `push2(2)`, `push2(3)`으로 `s1`에는 아래에서부터 1, 2, 3이 쌓인다.
2. 첫 번째 `pop2()`에서 `s2`가 비어 있으므로 `s1`의 3, 2, 1을 차례로 꺼내 `s2`로 옮긴다. 그러면 `s2`의 top은 1이 되어 첫 출력은 1이고, 다음 `pop2()`는 2를 출력한다.
3. 그다음 `push2(4)`, `push2(5)`는 `s1`에 새로 쌓이지만 `s2`에는 아직 3이 남아 있으므로 다음 `pop2()`는 먼저 3을 출력한다.
4. 이후 `s2`가 비면 그제서야 `s1`의 5, 4를 `s2`로 옮기고 4, 5 순서로 출력한다.

왜 이 순서인가: `s2`에 먼저 들어가 있던 기존 원소를 모두 소비한 뒤에만 `s1`의 나중 원소를 옮기므로 전체 출력은 큐의 FIFO 순서인 1, 2, 3, 4, 5가 된다.

선지별 해설

① O: 실제 출력은 1 2 3 4 5이다. 첫 이동에서 `s2`에 1이 top으로 올라오고, 남아 있던 3을 먼저 꺼낸 뒤에야 나중에 들어온 4와 5를 처리하므로 FIFO가 유지된다.

② X: `2 1 5 4 3`은 첫 두 출력의 순서와 뒤쪽 순서를 모두 잘못 보았다. 틀린 표현은 1보다 2가 먼저 나온다는 점과 3보다 5, 4가 먼저 나온다는 점이다. 올바르게는 1이 먼저 나오고, `s2`에 남아 있던 3이 4와 5보다 먼저 나와서 1 2 3 4 5가 된다.

③ X: `3 2 5 4 1`은 첫 번째 출력을 스택 top인 3으로 착각한 경우이다. 틀린 표현은 첫 출력이 3이라는 점이다. 올바르게는 `s1`의 원소를 `s2`로 모두 옮긴 뒤 `s2`에서 pop하므로 첫 출력은 1이고 전체는 1 2 3 4 5이다.

④ X: `5 4 3 2 1`은 전체 구조를 하나의 스택처럼 본 경우이다. 틀린 표현은 모든 값이 역순으로 나온다는 점이다. 올바르게는 두 스택을 이용해 큐를 구현했으므로 출력은 완전 역순이 아니라 FIFO 순서인 1 2 3 4 5이다.

오답 포인트

함정은 `push2(4)`, `push2(5)`가 실행된 뒤 곧바로 4와 5가 출력된다고 보는 데 있다. `pop2()`는 `s2`가 빌 때만 `s1`에서 옮기므로, 이미 `s2`에 남아 있는 3이 반드시 먼저 출력된다.

다시 나오면 이렇게 판단

두 스택 큐 구현 문제는 `입력 스택(s1)`과 `출력 스택(s2)`를 분리해서 본다. `dequeue` 시점마다 먼저 `s2`가 비어 있는지 확인하고, 비어 있지 않으면 `s2`에서 바로 꺼내며, 비어 있을 때만 `s1` 전체를 옮긴다고 판단하면 순서를 정확히 잡을 수 있다.

암기 포인트

`s2`가 비어야만 이동하고, 비어 있지 않으면 그대로 꺼낸다. 이 규칙이 두 스택 큐의 FIFO를 만든다.

NoteBase

문 21. 다음 데이터를 비어 있는 AVL(Adelson-Velskii, Landis) 트리에 순서대로 삽입할 때, 완전 이진 트리에 해당하는 것만을 모두 고른 것은?

ㄱ. 1, 2, 4, 3, 6, 5
ㄴ. 2, 1, 5, 4, 6, 3
ㄷ. 3, 1, 4, 2, 5, 6
ㄹ. 4, 1, 2, 3, 6, 5

① ㄱ, ㄴ
② ㄱ, ㄹ
③ ㄴ, ㄷ
④ ㄷ, ㄹ

정답 2번

출제 포인트

이 문제는 AVL 트리에 데이터를 순서대로 삽입한 뒤 회전 결과까지 반영하여 최종 트리가 완전 이진 트리인지 판단하는 문제이다.

올바른 순서

1. 주어진 수열을 이진 탐색 트리 규칙대로 차례로 삽입한다.
2. 삽입할 때마다 균형 인수의 절댓값이 2가 되면 LL, RR, LR, RL 회전을 적용한다.
3. 모든 삽입이 끝난 뒤 최종 트리의 레벨 순서를 확인한다.
4. 마지막 레벨을 제외한 모든 레벨이 꽉 차 있고, 마지막 레벨도 왼쪽부터 연속으로 채워져 있으면 완전 이진 트리이다.

왜 이 순서인가: AVL 삽입은 중간 회전이 최종 형태를 바꾸므로 단순 삽입 결과가 아니라 회전까지 끝난 최종 구조로 완전 이진 트리 여부를 판단해야 한다.

ㄱ: 최종 구조가 루트 4, 왼쪽 2, 오른쪽 6, 그 아래 1, 3, 5가 되어 완전 이진 트리이다.

ㄴ: 최종 구조가 루트 4, 왼쪽 2, 오른쪽 5, 그 아래 1, 3, 6이 되어 오른쪽 왼쪽 자리가 비어 있으므로 완전 이진 트리가 아니다.

ㄷ: 최종 구조가 루트 3, 왼쪽 1, 오른쪽 5, 그 아래 2, 4, 6이 되어 마지막 레벨이 왼쪽부터 채워지지 않으므로 완전 이진 트리가 아니다.

ㄹ: 최종 구조가 루트 4, 왼쪽 2, 오른쪽 6, 그 아래 1, 3, 5가 되어 완전 이진 트리이다.

NoteBase

선지별 해설

① X
ㄱ은 완전 이진 트리이고 ㄴ은 완전 이진 트리가 아니므로 오답이다. 틀린 표현은 'ㄱ, ㄴ'이다. 올바르게는 'ㄱ, ㄹ'이다.

② O
ㄱ과 ㄹ만 최종 AVL 트리가 루트 4를 중심으로 [4, 2, 6, 1, 3, 5] 형태가 되어 마지막 레벨이 왼쪽부터 채워진다. 따라서 정답이다.

③ X
ㄴ과 ㄷ은 모두 완전 이진 트리가 아니므로 오답이다. 틀린 표현은 'ㄴ, ㄷ'이다. 올바르게는 'ㄱ, ㄹ'이다.

④ X
ㄹ은 맞지만 ㄷ이 틀리므로 오답이다. 틀린 표현은 'ㄷ, ㄹ'이다. 올바르게는 'ㄱ, ㄹ'이다.

오답 포인트

AVL 문제에서 많이 틀리는 지점은 삽입 순서만 보고 중간 회전을 빼먹는 점이다. 특히 ㄴ과 ㄷ은 균형은 맞아 보이지만 마지막 레벨의 빈칸 위치가 왼쪽부터 연속이 아니어서 완전 이진 트리가 아니다.

다시 나오면 이렇게 판단

AVL 삽입 문제는 '삽입 -> 불균형 발생 위치 확인 -> 회전 적용 -> 최종 레벨 순서 확인'의 순서로 판단하면 된다. 완전 이진 트리는 마지막 레벨의 빈칸이 오른쪽 끝에만 있어야 한다는 기준으로 판별된다.

암기 포인트

AVL은 균형 여부만 보지 않고, 완전 이진 트리는 마지막 레벨이 왼쪽부터 채워지는지를 함께 본다.

NoteBase

문 22. 다음 데이터를 비어 있는 2-3 트리에 순서대로 삽입하였다. 8을 삭제한 후, 트리에서 레벨 1의 값을 나열하면? (단, 루트 노드의 레벨은 0이다)

20, 14, 35, 17, 23, 10, 5, 8, 27, 31

① 10, 20

② 14, 27

③ 8, 20, 27

④ 17, 20, 27

정답 2번

출제 포인트

이 문제는 2-3 트리의 삽입에 따른 분할과 삭제 후 재균형 결과를 끝까지 추적할 수 있는지를 묻는 문제이다.

올바른 순서

1. 20, 14, 35를 넣으면 루트가 분할되어 루트는 20, 자식은 14와 35가 된다.
2. 17, 23, 10, 5, 8을 차례로 넣으면 왼쪽 부분트리의 분할이 루트까지 전파되어 루트는 14, 레벨 1은 8과 20이 된다.
3. 27, 31을 넣으면 오른쪽 부분트리에서 23, 27, 35가 분할되고 27이 승격되어 레벨 1은 8과 20, 27이 된다.
4. 8을 삭제하면 8이 있는 내부 노드의 두 리프 5와 10이 먼저 병합되고, 비게 된 내부 노드는 형제 쪽의 여유 키와 부모 키를 재분배받아 루트가 20, 레벨 1이 14와 27이 된다.

왜 이 순서: 삭제 직후 왼쪽 내부 노드가 공백이 되므로 형제 노드의 여유 키 20을 부모로 올리고 부모의 14를 왼쪽으로 내려 보내는 재균형이 필요하다.

NoteBase

선지별 해설

① X
이 선택지는 삭제 후 재균형을 반영하지 못한 값이다. 10은 병합된 리프의 값일 뿐 레벨 1로 올라오지 않는다.
틀린 표현 → 올바르게는 레벨 1의 값은 14, 27이다.

② O
삽입을 모두 마친 뒤 8을 삭제하면, 왼쪽 내부 노드가 비면서 형제 노드 20, 27과 부모 14 사이에서 재분배가 일어난다. 그 결과 20은 루트로 승격되고 레벨 1에는 14와 27이 남는다.

③ X
이 선택지는 삭제된 8을 레벨 1에 그대로 남겨 둔 점이 틀리다. 또한 20은 최종적으로 루트로 올라가므로 레벨 1에 있지 않다.
틀린 표현 → 올바르게는 8은 삭제되고, 레벨 1의 값은 14와 27이다.

④ X
이 선택지는 20을 레벨 1 값으로 본 점이 틀리다. 삭제 후 재균형에서 20은 부모 자리로 올라가 레벨 0의 루트가 된다.
틀린 표현 → 올바르게는 20은 루트이고, 레벨 1의 값만 쓰면 14와 27이다.

오답 포인트

삭제를 단순히 값 하나 제거로만 보면 8이나 20을 레벨 1에 잘못 남기기 쉽다. 이 문항의 함정은 삭제 뒤의 병합과 형제 노드로부터의 재분배까지 봐야 한다는 점이다.

다시 나오면 이렇게 판단

2-3 트리 문제는 삽입이 끝난 상태의 트리를 먼저 확정한 뒤, 삭제에서는 내부 키 삭제인지 리프 삭제인지 구분하고 병합 여부와 형제 차용 여부를 순서대로 추적하면 된다. 마지막에는 승격된 키가 어느 레벨로 이동했는지 다시 확인하면 된다.

암기 포인트

2-3 트리 삭제는 대체, 병합, 재분배의 흐름으로 본다. 형제 노드에 여유 키가 있으면 부모와 형제 사이 재분배가 일어나 레벨 값이 바뀔 수 있다.

NoteBase

문 23. 숫자 분석법(digit analysis)은 모든 키 값이 미리 알려져 있을 때 유용한 해시 함수이다. 키 집합이 다음과 같을 때, 숫자 분석법을 이용하여 십진수 두 자리의 주소를 생성하고자 한다. 두 자리 주소에 들어갈 자릿수로 가장 적절한 것은?

0001	1006	2011	3016	4021	5026	6031	7036	8041	9046
0002	1007	2012	3017	4022	5027	6032	7037	8042	9047
0003	1008	2013	3018	4023	5028	6033	7038	8043	9048
0004	1009	2014	3019	4024	5029	6034	7039	8044	9049
0005	1010	2015	3020	4025	5030	6035	7040	8045	9050

① 100의 자릿수, 1의 자릿수

② 100의 자릿수, 10의 자릿수

③ 1000의 자릿수, 1의 자릿수

④ 1000의 자릿수, 10의 자릿수

정답 3번

출제 포인트

이 문제는 숫자분석법에서 키 집합의 자릿수 분포를 보고 충돌이 가장 적은 두 자릿수를 고를 수 있는지를 묻는다.

자료 읽는 순서

1. 각 자릿수별로 값이 얼마나 다양하게 바뀌는지 확인한다.
2. 모든 키에서 고정된 자릿수는 주소 생성에 부적절하다고 판단한다.
3. 후보 자릿수 두 개를 묶었을 때 같은 조합이 반복되는지 확인한다.

NoteBase

선지별 해설

1번: X. 100의 자릿수는 표의 모든 키에서 0으로 고정되어 변별력이 없다. 1의 자릿수는 0부터 9까지 변하지만 앞자리가 항상 0이므로 주소가 00부터 09 정도로만 몰려 충돌이 많다. 틀린 표현은 '100의 자릿수'이며, 올바르게는 '1000의 자릿수'를 써야 한다.
2번: X. 100의 자릿수는 여전히 모두 0이고, 10의 자릿수도 0, 1, 2, 3, 4, 5 정도만 반복되어 조합 수가 매우 적다. 예를 들어 0001~0005와 1006~1010은 10의 자릿수가 모두 0이어서 주소가 크게 겹친다. 틀린 표현은 '100의 자릿수, 10의 자릿수'이며, 올바르게는 '1000의 자릿수, 1의 자릿수'이다.
3번: O. 1000의 자릿수는 0부터 9까지 고르게 바뀌고, 1의 자릿수도 함께 바뀌어 각 키의 두 자릿수 조합이 중복되지 않는다. 따라서 주어진 50개의 키를 두 자리 주소에 가장 고르게 분산시킬 수 있다.
4번: X. 1000의 자릿수는 유용하지만 10의 자릿수는 같은 묶음 안에서 반복된다. 예를 들어 2011~2015는 모두 1000의 자릿수가 2이고 10의 자릿수가 1이므로 다섯 키가 같은 주소로 충돌한다. 틀린 표현은 '10의 자릿수'이며, 올바르게는 '1의 자릿수'이다.

오답 포인트

숫자분석법에서는 숫자가 있다는 이유만으로 아무 자릿수나 고르면 안 된다. 이 문항에서는 100의 자릿수가 전부 0이라는 점을 놓치거나, 10의 자릿수가 여러 키에서 반복된다는 점을 보지 못하면 오답으로 흐른다.

다시 나오면 이렇게 판단

자릿수 선택 문제는 먼저 각 자릿수의 분산 정도를 본다. 고정된 자릿수는 버리고, 두 자릿수를 묶었을 때 중복 조합이 가장 적은 쌍을 정답으로 판단한다.

암기 포인트

숫자분석법은 값이 고정된 자릿수는 버리고, 가장 다양하게 흩어지는 자릿수를 고르는 방식이다.

NoteBase

문 24. 다음 파이썬 프로그램의 실행 결과는?

```
import queue

values=queue.Queue()
for i in range(20):
   if i % 3 == 0:
      values.put(i)
   elif i % 4 == 0:
      values.get()
   elif i % 5 == 0:
      values.get()
print(values.get(), values.get())
```

① 15
② 18
③ 15 18
④ 12 15 18

정답 3번

출제 포인트

이 문제는 `if-elif` 분기 우선순위와 큐의 FIFO 특성을 이용해 반복문 종료 뒤 남는 출력값을 추적하는 문제이다.

올바른 순서

1. `i % 3 == 0`이면 현재 `i`를 큐 뒤에 넣는다.
2. `elif` 구조이므로 여러 조건이 동시에 성립해도 앞에서 처음 참이 된 조건 하나만 실행된다.
3. `i % 4 == 0` 또는 `i % 5 == 0`은 앞 조건이 거짓일 때만 큐 앞 값을 하나 뺀다.
4. 실제 변화 시점만 적으면 `0:[0]`, `3:[0,3]`, `4:[3]`, `5:[]`, `6:[6]`, `8:[]`, `9:[9]`, `10:[]`, `12:[12]`, `15:[12,15]`, `16:[15]`, `18:[15,18]`이 된다.
5. 반복 종료 뒤 큐는 `[15, 18]`이므로 `print(values.get(), values.get())`의 출력은 `15 18`이 된다.

왜 이 순서인가: `12`와 `15`는 각각 `4`와 `5`의 배수이기도 하지만 `if-elif` 때문에 삽입만 수행되고, `16`에서 먼저 빠지는 값도 FIFO 때문에 `12`가 된다.

NoteBase

선지별 해설

① X
`15`만 출력되려면 마지막 `get()`이 한 번만 실행되거나 두 번째 값이 없어야 한다. 실제로 반복 종료 뒤 큐에는 `[15, 18]`이 남아 있으므로 두 값이 모두 출력된다.
틀린 표현: `15`
올바르게는: `15 18`

② X
`18`만 출력되려면 `15`가 먼저 제거되어야 한다. 하지만 큐는 FIFO이므로 `16`에서 제거되는 값은 먼저 들어온 `12`이고, `15`는 그대로 남는다.
틀린 표현: `18`
올바르게는: `15 18`

③ O
반복문이 끝난 뒤 큐 상태는 `[15, 18]`이다. `Queue`는 먼저 들어간 값을 먼저 꺼내므로 첫 번째 `get()`이 `15`, 두 번째 `get()`이 `18`을 반환한다.

④ X
`12`는 `15`보다 먼저 큐에 들어가지만 `16`에서 `i % 4 == 0`이 참이 되어 제거된다. 따라서 출력 시점에는 `12`가 남아 있지 않다.
틀린 표현: `12 15 18`
올바르게는: `15 18`

오답 포인트

`12`와 `15`에서 `%4`, `%5`도 성립하므로 삭제가 일어난다고 착각하거나, `if-elif`를 서로 독립적인 `if`로 읽으면 큐 상태를 잘못 계산한다. 또한 큐를 스택처럼 생각하면 마지막 출력을 `18 15`처럼 거꾸로 오해할 수 있다.

다시 나오면 이렇게 판단

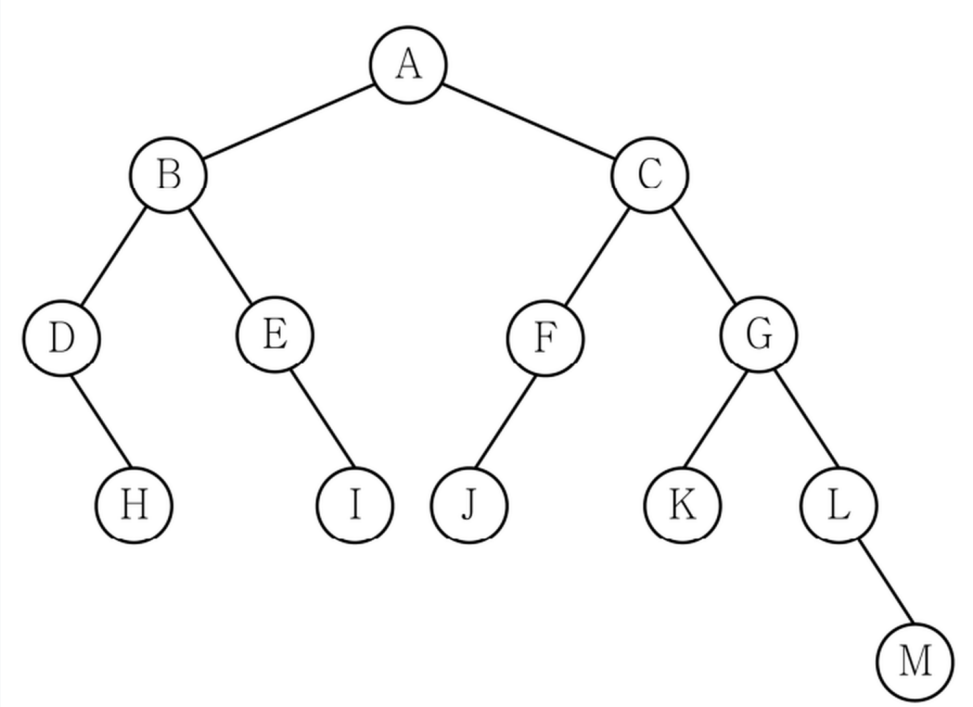
실행 결과 추적 문제는 `분기 우선순위 확인 -> 자료구조 성질 확인 -> 값이 변하는 시점만 기록` 순서로 판단한다. 이 문제에서는 `if-elif` 때문에 `12`와 `15`가 삭제되지 않고 들어가며, FIFO 때문에 마지막 두 값은 들어온 순서대로 `15`, `18`이 된다.

암기 포인트

`if-elif`는 첫 참 조건만 실행되고, 큐는 먼저 넣은 값이 먼저 나온다.

NoteBase

문 25. 다음 트리는 노드에서 키와 값이 무엇인지 알 수는 없지만 이진 탐색 트리라고 가정했을 때, 이에 대한 설명으로 옳지 않은 것은? (단, 루트 노드만 있는 경우에 트리 높이는 0이다)



① 트리의 높이는 4이다.

② AVL 트리에 해당한다.

③ C 노드의 중위 후속자(inorder successor)는 M 노드이다.

④ 높이를 증가시키지 않고 트리에 추가할 수 있는 최대 노드 수는 18개이다.

정답 3번

출제 포인트

이 문제는 이진 탐색 트리 도식에서 높이, AVL 균형 조건, 중위 후속자, 높이를 유지한 채 채울 수 있는 최대 노드 수를 함께 판단하는 문제이다.

NoteBase

선지별 해설

① O
트리 높이는 루트에서 가장 깊은 리프까지의 간선 수이다. 가장 긴 경로는 A-C-G-L-M이므로 간선 수는 4이고 높이는 4이다.

② O
AVL 트리는 각 노드의 왼쪽 부분트리와 오른쪽 부분트리의 높이 차가 1 이하인 이진 탐색 트리이다. B는 왼쪽 1, 오른쪽 1, C는 왼쪽 1, 오른쪽 2, A는 왼쪽 2, 오른쪽 3이므로 모든 노드에서 높이 차가 1 이하이다. 따라서 이 트리는 AVL 트리에 해당한다.

③ X
중위 후속자는 중위 순회에서 바로 다음에 방문되는 노드이다. 어떤 노드에 오른쪽 부분트리가 있으면 그 후속자는 오른쪽 부분트리에서 가장 왼쪽에 있는 노드가 된다. C의 오른쪽 부분트리는 G이고, 그 부분트리의 가장 왼쪽 노드는 K이므로 C의 중위 후속자는 K이다. 틀린 표현은 "M 노드"이고, 올바르게는 "C 노드의 중위 후속자는 K 노드이다"이다.

④ O
높이를 4로 유지하는 이진트리의 최대 노드 수는 $2^5 - 1 = 31$개이다. 현재 노드 수는 A부터 M까지 13개이므로, 높이를 증가시키지 않고 추가할 수 있는 최대 노드 수는 31-13=18개이다.

오답 포인트

이 문항의 함정은 중위 후속자를 오른쪽 부분트리의 가장 큰 노드로 착각하는 데 있다. 중위 후속자는 오른쪽 부분트리의 가장 왼쪽 노드이며, 이 문제에서는 M이 아니라 K이다. 또 높이를 노드 수로 세지 않고 간선 수로 세어야 한다는 조건도 함께 확인해야 한다.

다시 나오면 이렇게 판단

이진 탐색 트리 성질을 묻는 도식형 문항에서는 먼저 발문이 옳은 것인지 옳지 않은 것인지 확인한 뒤, 높이는 가장 긴 경로의 간선 수로 계산한다. 중위 후속자는 "오른쪽 부분트리로 한 번 이동한 뒤 끝까지 왼쪽"으로 찾고, 높이를 유지한 최대 노드 수는 $2^{h+1} - 1$에서 현재 노드 수를 빼서 판단한다.

암기 포인트

중위 후속자는 오른쪽이 있으면 오른쪽으로 한 번 간 뒤 가장 왼쪽 노드이다. 높이 h인 이진트리의 최대 노드 수는 $2^{h+1} - 1$이다.

NoteBase