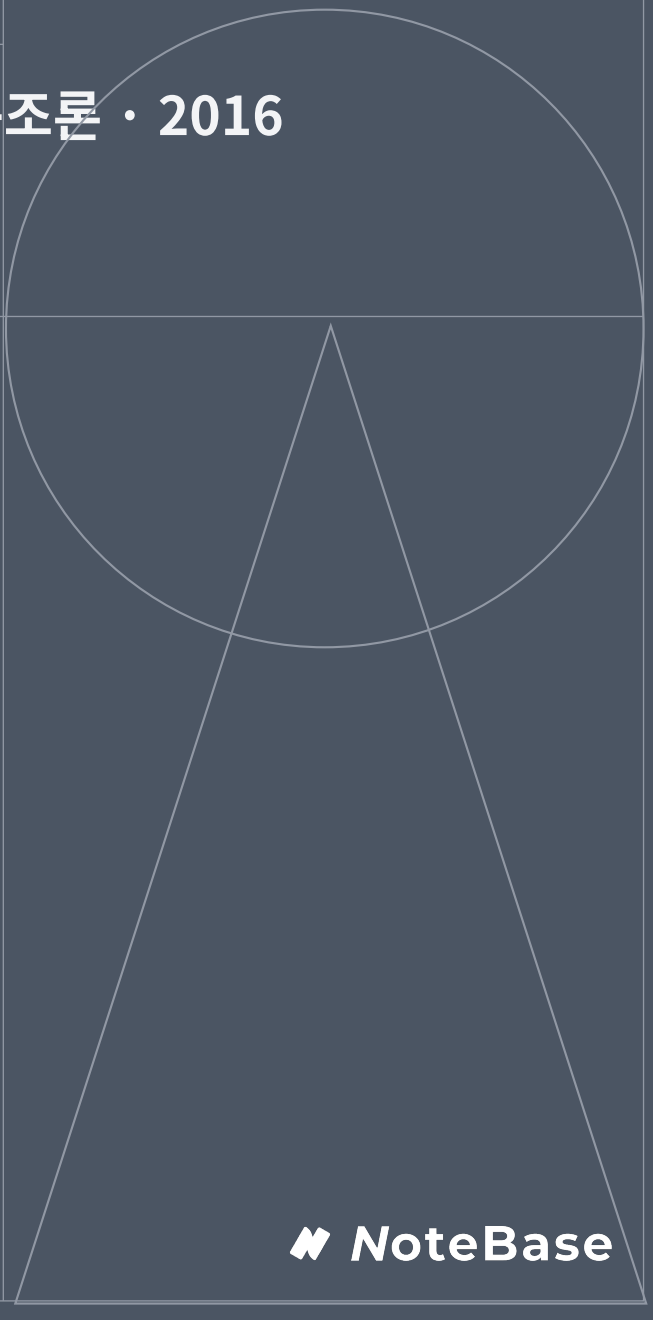
기출OX 해설집

공무원/국가직/7급 · 자료구조론 · 2016



NoteBase

기출OX 해설집

공무원/국가직/7급 · 자료구조론 · 2016

공무원/국가직/7급 · 자료구조론 · 2016

문 1. 후위 표기법(postfix notation)으로 표현된 다음 수식의 계산 결과는?

2 3 + 2 * 2 1 + 3 / -

① 7
② 8
③ 9
④ 10

정답 3번

출제 포인트

이 문제는 후위 표기식을 스택 방식으로 왼쪽부터 읽으면서 연산자마다 직전 두 피연산자에 계산을 적용해 최종값을 구하는 문제이다.

조건 정리

주어진 식: 2 3 + 2 * 2 1 + 3 / -
구해야 할 값: 식 전체의 최종 계산 결과
주의할 조건: 후위 표기식은 연산자가 나올 때마다 가장 최근의 두 값을 꺼내 계산하며, 마지막 '-'는 앞에서 만든 두 중간값 10과 1에 적용된다.

사용할 공식

후위 표기식 계산 원리는 스택의 top 두 원소를 꺼내 연산한 뒤 그 결과를 다시 넣는 방식이다.
이 공식을 쓰는 이유: 주어진 식은 괄호가 없는 후위 표기식이므로 연산자 우선순위를 따로 정하지 않고 스택 처리 순서만 따르면 된다.

NoteBase

계산 과정

1. 2와 3 다음에 '+'가 나오므로 2 + 3 = 5가 된다.
2. 그다음 2와 '*'를 적용하므로 5 × 2 = 10이 된다.
3. 다음 부분 2 1 + 는 2 + 1 = 3이 된다.
4. 이어서 3 / 는 직전 값 3과 새로 읽은 3에 적용되므로 3/3 = 1이 된다.
5. 마지막 '-'는 앞의 10과 1에 적용되므로 10 − 1 = 9가 된다.

선지별 해설

① X: 7은 최종 결과가 아니다. 7이 되려면 10 − 3처럼 '/' 결과를 1이 아니라 3으로 잘못 보아야 한다. 계산 결과를 9로 고치면 옳다.
② X: 8은 최종 결과가 아니다. 8이 되려면 중간 계산을 10 − 2처럼 처리하는 오류가 필요하다. '/'를 마친 값은 2가 아니라 1이므로 계산 결과를 9로 고치면 옳다.
③ O: 앞부분은 2 3 + 2 * = 10, 뒷부분은 2 1 + 3 / = 1이므로 최종값은 10 − 1 = 9이다.
④ X: 10은 앞부분 2 3 + 2 *까지만 계산한 값이다. 뒤의 2 1 + 3 / −를 반영해야 완성된 식이 되므로 계산 결과를 9로 고치면 옳다.

오답 포인트

출제자가 노린 함정은 연산자를 만날 때마다 바로 직전 두 값을 처리한다는 원리를 놓치게 만드는 데 있다. 특히 마지막 '-'를 10과 3에 적용하거나, 10이 나온 시점에서 계산을 끝낸다고 착각하면 오답이 된다.

다시 나오면 이렇게 판단

후위 표기식은 왼쪽부터 한 칸씩 읽으면서 숫자는 쌓고 연산자는 직전 두 값을 꺼내 계산한 뒤 다시 넣는 순서로 판단한다. 마지막에 스택에 남는 하나의 값이 정답이다.

암기 포인트

후위 표기식은 숫자는 push, 연산자는 두 값 pop 후 계산, 결과는 다시 push라는 흐름으로 정리된다.

NoteBase

문 2. 노드의 개수가 177인 완전 이진 트리(complete binary tree)에서 마지막 레벨(level)에 위치한 노드의 개수는? (단, 루트 노드는 첫 번째 레벨에 위치한다)

① 49

② 50

③ 78

④ 127

정답 2번

출제 포인트

이 문제는 완전 이진 트리에서 마지막 레벨 직전까지의 누적 노드 수를 구한 뒤 전체 노드 수에서 빼서 마지막 레벨의 노드 수를 계산하는 문제이다.

조건 정리

주어진 값: 전체 노드 수는 177이다.
구해야 할 값: 마지막 레벨에 있는 노드 수이다.
주의할 조건: 루트가 첫 번째 레벨이므로 레벨 1부터 레벨 7까지가 모두 찬 경우의 누적 노드 수는 $2^7 - 1 = 127$이다.

사용할 공식

$1 + 2 + 4 + \cdots + 2^{h-1} = 2^h - 1$
이 공식을 쓰는 이유: 완전 이진 트리는 마지막 레벨을 제외한 위 레벨들이 모두 채워져 있으므로, 마지막 레벨 직전까지의 누적 노드 수를 먼저 구하면 마지막 레벨의 노드 수를 바로 계산할 수 있다.

계산 과정

1. 레벨 1부터 레벨 7까지 모두 찬 노드 수를 구하면 $2^7 - 1 = 127$이다.
2. 전체 노드 수 177개에서 127개를 빼면 마지막 레벨의 노드 수가 된다.
3. $177 - 127 = 50$이므로 마지막 레벨의 노드 수는 50개이다.

NoteBase

선지별 해설

① X: 49는 마지막 레벨 직전까지의 누적 노드 수를 127이 아니라 128로 잘못 잡았을 때 나오는 값이다. 틀린 표현은 49라는 값 자체이며, 올바르게는 마지막 레벨 노드 수가 50이다.
② O: 레벨 7까지의 누적 노드 수가 127이고 전체가 177이므로 마지막 레벨 노드 수는 50이다. 계산 결과와 일치하므로 옳다.
③ X: 78은 완전 이진 트리의 누적 구조를 적용하지 않고 임의로 남은 수를 잘못 계산한 값이다. 틀린 표현은 78이라는 값 자체이며, 올바르게는 177에서 127을 뺀 50이다.
④ X: 127은 마지막 레벨의 노드 수가 아니라 마지막 레벨 직전까지 모두 찼을 때의 누적 노드 수이다. 틀린 표현은 마지막 레벨 노드 수를 127로 본 점이며, 올바르게는 마지막 레벨 이전 누적이 127이고 마지막 레벨은 50이다.

오답 포인트

완전 이진 트리에서 마지막 레벨 이전까지의 누적 노드 수와 마지막 레벨의 노드 수를 혼동하기 쉽다. 특히 2^h와 $2^h - 1$을 혼동하면 49나 127 같은 오답이 나온다.

다시 나오면 이렇게 판단

완전 이진 트리의 노드 수 문제는 먼저 마지막 레벨 이전까지가 몇 레벨인지 잡고, 그 누적을 $2^h - 1$로 계산한 뒤 전체 노드 수에서 빼서 마지막 레벨 노드 수를 판단한다.

암기 포인트

완전 이진 트리에서 마지막 레벨 이전까지 모두 찬 노드 수는 $2^h - 1$이고, 마지막 레벨 노드 수는 전체에서 그 값을 뺀 나머지이다.

NoteBase

문 3. 다음 C 언어 함수를 이용하여 asterisk(9)를 수행할 때 출력되는 '*'의 개수는?

```
#include <stdio.h>

void asterisk(int i)
{
    if(i > 1){
        asterisk(i/3);
        asterisk(i/3);
    }
    printf("*");
}
```

① 6
② 7
③ 8
④ 9

정답 2번

출제 포인트

이 문제는 재귀 호출에서 출력문이 실행되는 총 횟수를 점화식으로 구하는 문제이다.

조건 정리

주어진 값: $i = 9$에서 함수 `asterisk(9)`를 호출한다.
구해야 할 값: `printf("*")`가 실행되는 횟수, 즉 `*`의 총 출력 개수이다.
주의할 조건: C 언어 정수 나눗셈이므로 9/3 = 3, 3/3 = 1이고, `printf`는 모든 호출의 마지막에 한 번씩 실행된다.

NoteBase

사용할 공식

출력 개수를 $T(i)$라 두면, $i \le 1$일 때 $T(i) = 1$, $i > 1$일 때 $T(i) = 2T(i/3) + 1$이다.
이 공식을 쓰는 이유: `i>1`이면 같은 함수가 `i/3`으로 두 번 호출되고, 두 하위 호출이 끝난 뒤 현재 호출에서도 `printf("*")`가 한 번 실행되기 때문이다.

계산 과정

1. $T(1) = 1$이다. `i`가 1이면 재귀 호출 없이 `printf`만 한 번 실행된다.
2. $T(3) = 2T(1) + 1 = 2 \times 1 + 1 = 3$이다.
3. $T(9) = 2T(3) + 1 = 2 \times 3 + 1 = 7$이다.
4. 호출 트리로 보면 $9 \to 3, 3 \to 1, 1, 1, 1$이므로 전체 호출 수는 $1 + 2 + 4 = 7$개이다.

선지별 해설

① X: 6개라는 뜻인데 오답이다. $T(9) = 2T(3) + 1$에서 현재 호출의 $+1$을 빼먹으면 6이 되지만, 실제로는 최상위 호출도 마지막에 `*`를 한 번 출력한다. 틀린 표현은 '6개'이고, 올바르게는 '7개'이다.
② O: 7개가 맞다. $T(3) = 3$, $T(9) = 2 \times 3 + 1 = 7$이므로 정답과 일치한다.
③ X: 8개라는 뜻인데 오답이다. 재귀가 두 배로만 늘어난다고 단순하게 보면 이런 값에 끌릴 수 있지만, 실제 출력 횟수는 완전이진트리의 노드 수처럼 $1 + 2 + 4 = 7$로 계산된다. 틀린 표현은 '8개'이고, 올바르게는 '7개'이다.
④ X: 9개라는 뜻인데 오답이다. 인수로 9가 전달되었다고 해서 `*`가 9번 출력되는 구조가 아니다. 출력 횟수는 숫자 자체가 아니라 재귀 호출 구조로 결정된다. 틀린 표현은 '9개'이고, 올바르게는 '7개'이다.

오답 포인트

`printf("*")`가 재귀 호출 안이 아니라 함수의 마지막에 있으므로 모든 호출이 `*`를 한 번씩 출력한다는 점을 놓치기 쉽다. 특히 $2T(i/3)$까지만 세고 현재 호출의 $+1$을 빠뜨리면 6으로 오답 처리된다.

다시 나오면 이렇게 판단

재귀 출력 개수 문제는 먼저 '한 번 호출될 때 자기 자신이 몇 번 출력하는지'를 확인한 뒤, 기저조건과 점화식 $T(i)$를 세운다. 그다음 작은 값부터 계산하거나 호출 트리의 전체 노드 수를 세면 된다.

암기 포인트

재귀 함수에서 출력문이 함수 바깥쪽에 있으면 '출력 횟수 = 전체 호출 수'로 보는 경우가 많다.

NoteBase

문 4. 가장 낮은 자릿수(LSD: Least Significant Digit) 우선 방식의 기수 정렬(radix sort)을 이용하여, 다음과 같이 배열에 저장된 10진수 데이터를 오름차순으로 정렬하고자 한다. 오른쪽에서 두 번째 자릿수(digit) 기준의 정렬 단계 후에 위치 [3]에 있는 데이터로 옳은 것은? (단, 각 단계별 정렬의 기준이 되는 자릿수는 0에서 9까지의 값을 가진다)

위치	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
데이터	307	424	132	216	241	835	546	630

① 630

② 546

③ 132

④ 216

정답 1번

출제 포인트

LSD 기수정렬에서 안정성을 유지하며 일의 자리 정렬 결과를 바탕으로 십의 자리 정렬을 수행했을 때 특정 위치의 값을 찾는 문제이다.

올바른 순서

1. 일의 자리 기준으로 각 수를 0부터 9까지의 버킷에 안정적으로 넣는다.
2. 버킷을 0부터 9까지 꺼내면 배열은 [630, 241, 132, 424, 835, 216, 546, 307]이 된다.
3. 이 배열을 입력으로 십의 자리 기준 버킷에 다시 안정적으로 넣는다.
4. 버킷을 0부터 9까지 꺼내면 최종 배열은 [307, 216, 424, 630, 132, 835, 241, 546]이 된다.

왜 이 순서: LSD 기수정렬은 낮은 자리에서 만든 상대적 순서를 유지한 채 다음 자리 정렬을 해야 하므로, 십의 자리 정렬은 반드시 일의 자리 정렬 결과를 그대로 입력으로 사용한다.

NoteBase

선지별 해설

① O
- 일의 자리 정렬 뒤 배열은 [630, 241, 132, 424, 835, 216, 546, 307]이다. 이를 십의 자리로 다시 정렬하면 0버킷 307, 1버킷 216, 2버킷 424, 3버킷 630·132·835, 4버킷 241·546 순이므로 위치 [3]은 630이다.

② X
- 546은 십의 자리가 4이므로 241과 같은 4버킷에 들어간다. 안정 정렬이므로 241 뒤에 놓여 최종 위치는 [7]이다.
- 올바르게는 위치 [3]의 데이터는 546이 아니라 630이다.

③ X
- 132는 십의 자리가 3이어서 630, 132, 835와 같은 3버킷에 속하지만, 일의 자리 정렬 뒤 입력 순서가 630 다음이므로 최종 위치는 [4]이다.
- 올바르게는 위치 [3]의 데이터는 132가 아니라 630이다.

④ X
- 216은 십의 자리가 1이므로 1버킷에 들어가고, 0버킷의 307 다음에 바로 와서 최종 위치는 [1]이다.
- 올바르게는 위치 [3]의 데이터는 216이 아니라 630이다.

오답 포인트

십의 자리만 바로 보고 원래 배열에서 재배치하면 오답이 된다. 이 문제의 함정은 십의 자리 정렬이 원배열이 아니라 일의 자리 정렬이 끝난 배열을 입력으로 받는다는 점과, 같은 버킷 안에서는 기존 순서가 유지된다는 안정성에 있다.

다시 나오면 이렇게 판단

LSD 기수정렬 문제는 현재 자리수만 보지 말고 바로 이전 패스의 결과 배열을 먼저 적은 뒤, 그 순서를 유지한 채 다음 자리 버킷에 넣어 판단한다.

암기 포인트

LSD 기수정렬은 낮은 자리부터, 같은 자릿값끼리는 이전 패스의 순서를 유지한다.

NoteBase

문 5. 해시 테이블 HT의 크기는 11이고 0부터 10까지의 인덱스를 가진다. 해시 함수 h(x)는 제산 함수로서 h(x) = x mod 11로 표현된다. 충돌 해결책으로는 이차 조사법(quadratic probing)을 사용한다. 해싱을 통해 해시 테이블에 저장할 레코드들의 키 값 순서는 다음과 같다. 마지막 레코드가 저장되는 해시 테이블 위치는? (단, 첫 번째 레코드가 저장되기 전에 해시 테이블은 비어 있고, 해시 테이블 버킷(bucket)당 슬롯(slot) 수는 1개이다)

51, 27, 70, 55, 13, 2, 37, 23, 33

① HT[1]

② HT[4]

③ HT[6]

④ HT[9]

정답 4번

출제 포인트

이 문제는 해시 함수 h(x)=x 11과 이차 조사법 h_i(x)=(h(x)+i^2) 11을 적용하여 마지막 키 33의 실제 저장 위치를 구하는 문제이다.

조건 정리

테이블 크기는 11이므로 인덱스는 0부터 10까지이다.
삽입 순서는 51, 27, 70, 55, 13, 2, 37, 23, 33이다.
각 버킷의 슬롯 수는 1개이므로 충돌이 나면 같은 위치에 함께 저장할 수 없고, 1^2, 2^2, 3^2 순서로 더하며 다음 위치를 찾는다.

계산 과정

먼저 앞선 키들을 저장하면 51은 HT[7], 27은 HT[5], 70은 HT[4], 55는 HT[0], 13은 HT[2]에 저장된다.
키 2는 $h(2) = 2$에서 충돌하므로 (2+1^2) 11=3에 저장된다.
키 37은 $h(37) = 4$에서 충돌하고, (4+1^2) 11=5도 차 있으므로 (4+2^2) 11=8에 저장된다.
키 23은 $h(23) = 1$이므로 HT[1]에 바로 저장된다.
마지막 키 33은 $h(33) = 0$인데 HT[0]에는 55가 이미 있다. 따라서 (0+1^2) 11=1을 보지만 HT[1]에는 23이 있고, (0+2^2) 11=4에도 70이 있으며, (0+3^2) 11=9가 비어 있으므로 HT[9]에 저장된다.

NoteBase

선지별 해설

① X
HT[1]은 33의 첫 번째 조사 위치이지만, 그 자리는 앞에서 23이 이미 차지하고 있다. 틀린 표현은 '마지막 레코드가 HT[1]에 저장된다'이고, 올바르게는 '33은 HT[1]을 조사만 하고 다음 위치로 이동한다'이다.

② X
HT[4]는 33의 두 번째 조사 위치이지만, 그 자리는 70이 이미 저장된 상태이다. 틀린 표현은 '마지막 레코드가 HT[4]에 저장된다'이고, 올바르게는 '33은 HT[4]에서도 충돌하여 다음 조사 위치를 찾는다'이다.

③ X
HT[6]은 33이 실제로 멈추는 위치가 아니다. 이 문제의 이차 조사 순서는 0에서 시작해 1, 4, 9 순으로 진행되며, HT[9]가 비어 있어서 그 전에 삽입이 끝난다. 틀린 표현은 '마지막 레코드가 HT[6]에 저장된다'이고, 올바르게는 '33은 세 번째 조사 위치인 HT[9]에 저장된다'이다.

④ O
33은 HT[0]에서 충돌한 뒤 HT[1], HT[4]도 차 있는 것을 확인하고, 세 번째 조사 위치 HT[9]에서 처음 빈 슬롯을 만난다. 따라서 마지막 레코드의 저장 위치는 HT[9]가 맞다.

오답 포인트

선형 조사법처럼 0, 1, 2, 3 순서로 한 칸씩 이동한다고 착각하면 오답이 된다. 이 문제는 이차 조사법이므로 충돌 뒤 이동량이 1^2, 2^2, 3^2로 커진다는 점과, 이미 저장된 앞선 키들의 위치를 끝까지 반영해야 한다는 점이 함정이다.

다시 나오면 이렇게 판단

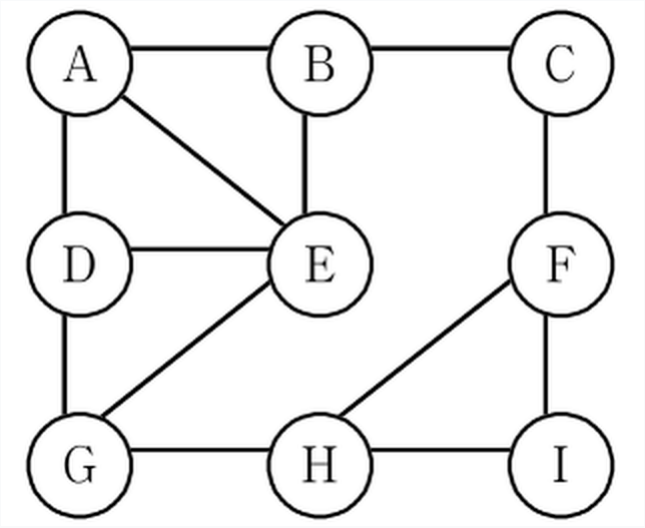
해시 충돌 문제는 각 키를 주어진 순서대로 실제로 배치해 본 뒤, 마지막 키에 대해서만 홈 위치와 조사 순서를 따라가면 된다. 조사법이 선형인지 이차인지 먼저 확인하고, 각 조사 위치가 비어 있는지 차례로 점검하면 된다.

암기 포인트

이차 조사법은 충돌 뒤 $h(x) + 1^2$, $h(x) + 2^2$, $h(x) + 3^2$ 순서로 본다는 점이 핵심이다.

NoteBase

문 6. 시작 정점이 A일 때, 다음 그래프에 대한 깊이 우선 탐색(DFS: Depth First Search) 및 너비 우선 탐색(BFS: Breadth First Search)의 방문 순서로 옳은 것은? (단, 인접한 정점들은 알파벳 순서로 방문한다)



① DFS: A, B, C, F, H, G, D, E, I
BFS: A, B, C, D, E, G, F, H, I

② DFS: A, B, C, F, H, G, D, E, I
BFS: A, B, D, E, C, G, F, H, I

③ DFS: A, B, C, F, H, G, E, D, I
BFS: A, B, D, E, C, G, F, H, I

④ DFS: A, B, C, F, H, G, E, D, I
BFS: A, B, C, D, E, G, F, H, I

정답 2번

출제 포인트

이 문제는 그래프의 인접 정점을 알파벳 순으로 선택하면서 DFS와 BFS의 방문 순서를 정확히 전개할 수 있는지를 묻는 문제이다.

NoteBase

올바른 순서

DFS는 A, B, C, F, H, G, D, E, I이다.
A에서 B, B에서 C, C에서 F로 가장 깊게 내려간 뒤 H에서 아직 방문하지 않은 G를 먼저 가고, G에서 D를 거쳐 E를 방문한 다음 마지막으로 I를 방문하기 때문이다.
BFS는 A, B, D, E, C, G, F, H, I이다.
BFS는 큐를 사용하므로 A의 인접 정점 B, D, E를 먼저 모두 방문 대상으로 확정한 뒤, 그 다음 레벨의 C, G, F, H, I 순으로 넓게 퍼져 나가기 때문이다.

선지별 해설

① X
DFS는 맞지만 BFS가 틀리다. BFS에서 A의 인접 정점은 알파벳 순으로 B, D, E이므로 C가 D와 E보다 먼저 방문될 수 없다.
틀린 표현: BFS가 A, B, C, D, E, G, F, H, I이다.
올바르게는: BFS는 A, B, D, E, C, G, F, H, I이다.

② O
DFS는 A, B, C, F, H, G, D, E, I이고 BFS는 A, B, D, E, C, G, F, H, I이므로 두 순서가 모두 맞다.
DFS는 깊이 우선으로 막힐 때까지 내려가고, BFS는 같은 레벨의 정점을 큐 순서대로 확장한다는 규칙과 일치한다.

③ X
BFS는 맞지만 DFS가 틀리다. H에서 G를 방문한 뒤 G의 미방문 인접 정점은 D가 먼저이고, D에서 E를 방문한 다음에야 되돌아와 I를 방문한다.
틀린 표현: DFS가 A, B, C, F, H, G, E, D, I이다.
올바르게는: DFS는 A, B, C, F, H, G, D, E, I이다.

④ X
DFS와 BFS가 모두 틀리다. DFS는 ③과 같은 이유로 G 다음에 E가 아니라 D가 와야 하고, BFS는 ①과 같은 이유로 C가 D와 E보다 먼저 올 수 없다.
틀린 표현: DFS가 A, B, C, F, H, G, E, D, I이고 BFS가 A, B, C, D, E, G, F, H, I이다.
올바르게는: DFS는 A, B, C, F, H, G, D, E, I이고 BFS는 A, B, D, E, C, G, F, H, I이다.

NoteBase

오답 포인트

DFS와 BFS를 모두 알파벳이 빠른 정점부터 간다고만 외우면 틀리기 쉽다. DFS는 현재 정점에서 가장 깊게 내려가는 규칙이 우선이고, BFS는 큐에 먼저 들어간 같은 레벨 정점이 먼저 나온다는 점이 핵심이다. 이 문제에서는 BFS에서 C를 너무 빨리 배치하거나, DFS에서 G 다음에 D보다 E를 먼저 두는 실수가 대표적 함정이다.

다시 나오면 이렇게 판단

그래프 순회 문제는 각 정점의 인접 정점을 먼저 알파벳 순으로 정리한 뒤 판단한다. DFS는 한 갈래를 끝까지 추적하고 막히면 되돌아가는 순서인지 확인하고, BFS는 시작 정점의 1레벨, 2레벨, 3레벨 순으로 큐에 들어간 순서가 유지되는지 확인하면 된다.

암기 포인트

DFS는 깊게 갔다가 되돌아오고, BFS는 가까운 레벨부터 넓게 간다.

NoteBase

문 7. 다음 C 프로그램의 실행 결과로 첫 번째 줄에 출력되는 값과 두 번째 줄에 출력되는 값의 차이 값은? (단, int로 선언된 정수형 변수는 4바이트를 차지하고 char로 선언된 문자형 변수는 1바이트를 차지한다)

```
#include <stdio.h>

void main()
{
    struct node {
        int id;
        char name[8];
        int score;
    } A[10];

    printf("%d\n", &A[0]);
    printf("%d\n", &A[4]);
}
```

① 56

② 64

③ 72

④ 80

정답 2번

출제 포인트

이 문제는 구조체의 크기와 배열 원소 간 주소 차이를 이용해 두 출력값의 차이를 구하는 문제이다.

NoteBase

조건 정리

주어진 값:
- int는 4바이트이다.
- char는 1바이트이다.
- 구조체는 int id, char name[8], int score로 구성된다.
- A는 이 구조체의 배열 A[10]이다.

구해야 할 값:
- &A[0]와 &A[4]의 주소 차이이다.

주의할 조건:
- 배열 원소 간 주소 차이는 원소 개수 차이 × 구조체 1개의 크기이다.

사용할 공식

구조체 1개의 크기 = 4 + 8 + 4 = 16바이트이다.
주소 차이 = (인덱스 차이) × (원소 크기) = (4 - 0) × 16 = 64바이트이다.
이 공식을 쓰는 이유:
A는 구조체 배열이므로 &A[4]는 &A[0]에서 구조체 4개만큼 떨어진 주소가 된다.

계산 과정

1. 구조체 node의 크기를 구하면 int id가 4바이트, char name[8]이 8바이트, int score가 4바이트이므로 총 16바이트이다.
2. A[0]에서 A[4]까지의 인덱스 차이는 4이다.
3. 따라서 주소 차이는 4 × 16 = 64바이트이다.

NoteBase

선지 판단

① X
- 56은 구조체 크기를 14바이트로 잘못 본 값이다.
- 틀린 표현은 구조체 1개의 크기를 14바이트로 본 점이다. 올바르게는 4 + 8 + 4 = 16바이트이다.

② O
- 구조체 1개의 크기가 16바이트이고 A[0]와 A[4]의 인덱스 차이가 4이므로 주소 차이는 64바이트이다.

③ X
- 72는 구조체 크기에 근거 없이 여분의 바이트를 더한 계산이다.
- 틀린 표현은 구조체 1개의 크기를 18바이트처럼 계산한 점이다. 올바르게는 이 문제의 조건에서 구조체 크기는 16바이트이다.

④ X
- 80은 A[0]부터 A[4]까지 원소가 5개라는 점만 보고 5 × 16으로 계산한 값이다.
- 틀린 표현은 주소 차이를 원소 수 5개로 계산한 점이다. 올바르게는 시작점과 끝점의 간격이므로 인덱스 차이 4를 곱해야 한다.

오답 포인트

A[0]와 A[4] 사이에 원소가 다섯 개 보인다는 이유로 5를 곱하는 실수가 가장 흔하다. 이 문제는 원소 개수가 아니라 인덱스 차이를 곱해야 하며, 먼저 구조체 1개의 크기를 정확히 구해야 한다.

다시 나오면 이렇게 판단

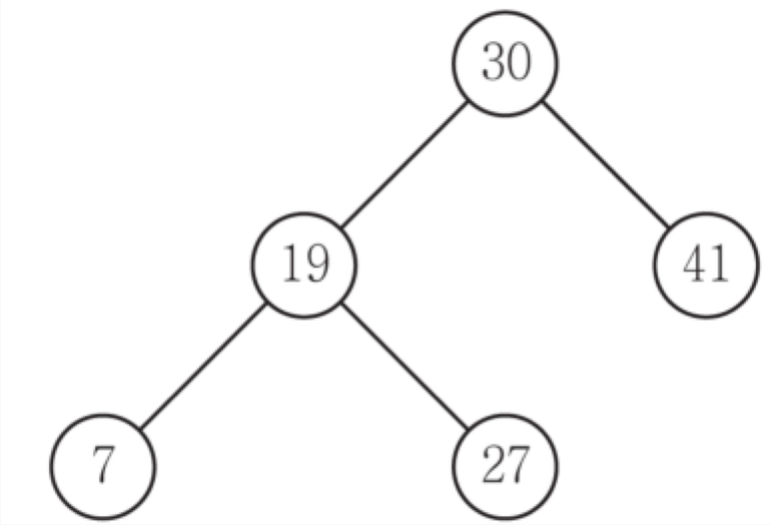
구조체 배열의 주소 차이 문제는 먼저 구조체 1개의 크기를 합산한 뒤, 배열 첨자 차이만큼 곱해 판단한다. 시작 원소와 끝 원소를 모두 세지 않고 간격만 계산하는 방식이다.

암기 포인트

구조체 배열 주소 차이 = 구조체 크기 × 첨자 차이이다.

NoteBase

문 8. 다음 AVL 트리에 정수 23을 삽입하면 AVL 트리로서의 균형이 깨진다. 회전을 통해 이 트리를 AVL 트리로 변환한 후에 전위 순회(preorder traversal)를 수행하였을 때 노드 방문 순서는?



① 30, 19, 7, 23, 41, 27

② 30, 7, 19, 23, 27, 41

③ 27, 19, 7, 23, 30, 41

④ 27, 19, 7, 30, 23, 41

정답 3번

출제 포인트

이 문제는 AVL 트리에 23을 삽입한 뒤 루트 30에서 발생하는 LR(Left-Right) 불균형을 회전으로 복구하고, 복구된 트리의 전위순회 결과를 판단하는 문제이다.

선지별 해설

① X: 23은 19의 오른쪽 자식 27의 왼쪽에 삽입되므로 30의 왼쪽 서브트리에서 LR 불균형이 발생한다. 따라서 회전 후 루트가 30으로 남을 수 없다는 점이 틀렸다. 올바르게는 27이 전체 루트가 되고 전위순회도 30으로 시작하지 않는다.
② X: 이 순서는 30이 루트로 남고 7이 19보다 먼저 부모처럼 방문되는 형태인데, AVL 재조정 결과와 맞지 않는다. 올바르게는 LR 회전 후 27이 루트, 19가 그 왼쪽 자식이 되므로 전위순회는 27, 19로 시작한다.
③ O: 23을 삽입하면 30에서 LR 불균형이 생긴다. 먼저 19를 기준으로 좌회전하면 27이 올라오고, 이어 30을 기준으로 우회전하면 최종 트리는 루트 27, 왼쪽 19(왼쪽 7, 오른쪽 23), 오른쪽 30(오른쪽 41)이 된다. 따라서 전위순회 결과는 27, 19, 7, 23, 30, 41이다.
④ X: 27이 루트가 되는 점은 맞지만 23의 위치가 틀렸다. 이 순서는 30의 왼쪽 자식이 23인 형태를 뜻하는데, 실제 LR 회전 후 23은 19의 오른쪽 자식이 된다. 올바르게는 27, 19, 7, 23, 30, 41이다.

NoteBase

오답 포인트

삽입 직후의 BST 모양만 보고 전위순회를 구하거나, LR 회전을 한 번의 막연한 재배치로 처리하면 30이 루트로 남는다고 착각하기 쉽다. 이 문항의 함정은 불균형이 19가 아니라 30에서 발생한다는 점과, 23이 최종적으로 30의 왼쪽이 아니라 19의 오른쪽으로 간다는 점이다.

다시 나오면 이렇게 판단

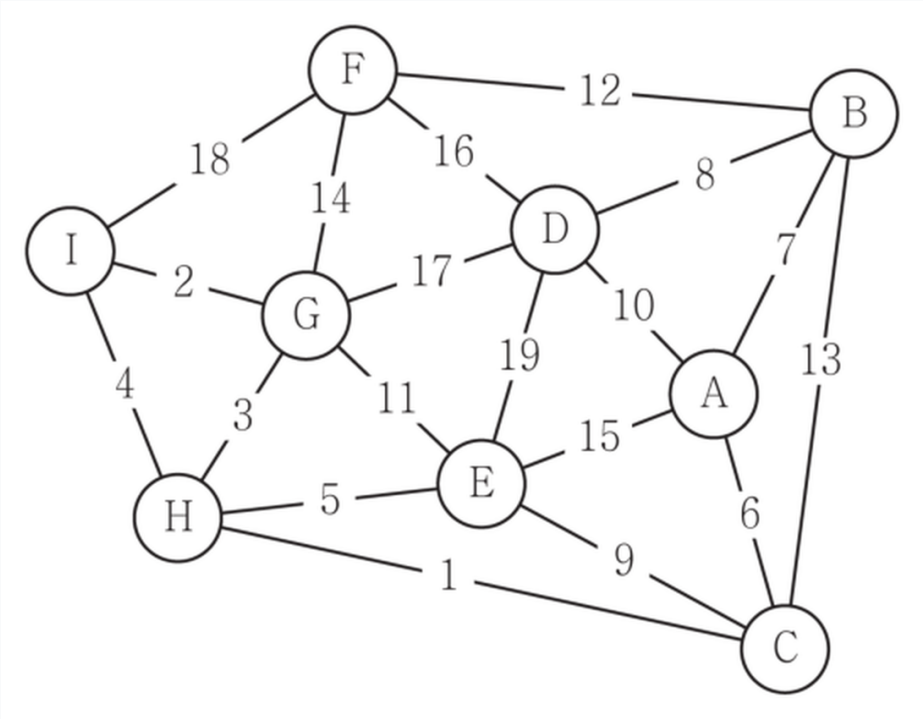
삽입 위치를 먼저 찾고, 가장 아래에서부터 균형이 처음 깨지는 조상을 찾으면 된다. 그 조상에서 삽입 경로가 왼쪽-오른쪽이면 LR, 오른쪽-왼쪽이면 RL로 판단하고, 이중 회전 후 새 서브트리 루트를 기준으로 전위순회를 읽으면 된다.

암기 포인트

AVL의 LR형은 왼쪽 자식 좌회전 후 부모 우회전이며, 가운데 노드가 새 루트가 된다.

NoteBase

문 9. 다음 가중 그래프(weighted graph)는 9개의 정점과 1부터 19까지의 가중치(비용)를 갖는 19개의 간선으로 구성되어 있다. Kruskal 알고리즘을 사용하여 최소 비용 신장 트리(minimum cost spanning tree)를 구성할 때, 최소 비용과 마지막으로 선택되는 간선을 순서대로 나열한 것은?



① 43, (G, E)

② 43, (B, F)

③ 44, (G, E)

④ 44, (B, F)

정답 4번

출제 포인트

이 문제는 Kruskal 알고리즘에서 간선을 가중치 오름차순으로 보되, 사이클을 만드는 간선은 버리면서 최소 신장 트리를 완성할 때의 총비용과 마지막 선택 간선을 찾는 문제이다.

조건 정리

정점은 9개이므로 최소 신장 트리에는 간선이 8개 선택되어야 한다.
가중치는 1부터 19까지 모두 다르므로 Kruskal의 선택 순서는 가중치 오름차순으로 유일하게 정해진다.
선택 여부는 가중치가 작은 순서대로 보면서 현재까지 선택한 간선들과 사이클을 만드는지만 확인하면 된다.

NoteBase

사용할 공식

Kruskal 알고리즘은 가중치가 작은 간선부터 검사하여 사이클을 만들지 않으면 선택하고, 사이클을 만들면 버린다.
이 원리를 쓰는 이유는 최소 신장 트리가 모든 정점을 연결하면서 총비용을 최소로 해야 하므로, 각 단계에서 연결 성분을 합치는 가장 작은 간선을 고르는 방식이기 때문이다.

계산 과정

가중치 순서대로 보면 1의 (H,C), 2의 (I,G), 3의 (H,G)는 선택된다.
4의 (I,H)는 이미 I-G-H로 연결되어 사이클이 생기므로 버린다.
5의 (H,E), 6의 (A,C), 7의 (A,B), 8의 (D,B)는 모두 서로 다른 연결 성분을 잇기 때문에 선택된다.
9의 (E,C), 10의 (D,A), 11의 (G,E)는 모두 기존 연결 안에서 사이클을 만들므로 버린다.
이 시점에서 연결되지 않은 정점은 F 하나뿐이므로 F를 가장 싸게 잇는 간선 12의 (B,F)를 마지막으로 선택한다.
따라서 선택 간선은 (H,C), (I,G), (H,G), (H,E), (A,C), (A,B), (D,B), (B,F)이고 총비용은 1+2+3+5+6+7+8+12=44이다.

선지별 해설

① X
43이 될 수 없고 마지막 간선도 (G,E)가 아니다. (G,E)는 가중치 11이지만 그 시점에는 G-H-E가 이미 연결되어 있어 사이클을 만들므로 선택되지 않는다. 틀린 표현은 "43, (G,E)"이고, 올바르게는 "44, (B,F)"이다.

② X
마지막 간선이 (B,F)인 점은 맞지만 총비용이 43이라는 부분이 틀리다. 마지막으로 선택되는 간선의 가중치가 12이므로 전체 합은 44가 된다. 틀린 표현은 "43"이고, 올바르게는 "44"이다.

③ X
총비용 44는 맞지만 마지막 간선이 (G,E)라는 부분이 틀리다. (G,E)는 11이더라도 이미 연결된 G와 E를 다시 잇는 간선이라 Kruskal에서 제외된다. 틀린 표현은 "(G,E)"이고, 올바르게는 "(B,F)"이다.

④ O
총비용이 44이고 마지막으로 선택되는 간선이 (B,F)이다. F는 가장 마지막까지 미연결로 남고, F를 기존 트리에 붙이는 가장 작은 간선이 12의 (B,F)이므로 이 선지가 옳다.

NoteBase

오답 포인트

작은 가중치부터 단순히 더해서 43이나 44를 만드는 식으로 접근하면 틀리기 쉽다. 특히 4의 (I,H), 9의 (E,C), 10의 (D,A), 11의 (G,E)는 수치가 작아 보여도 이미 연결된 정점 사이를 다시 이어 사이클을 만들기 때문에 선택되지 않는다. 또한 마지막 단계에서 아직 연결되지 않은 정점이 누구인지 확인하지 않으면 (G,E)를 마지막 간선으로 착각하기 쉽다.

다시 나오면 이렇게 판단

Kruskal 문제는 가중치 오름차순 정렬, 사이클 여부 확인, 현재 남은 미연결 정점 확인의 순서로 판단한다. 특히 마지막 간선은 아직 트리에 들어오지 않은 정점을 기존 트리에 붙이는 최소 간선인지부터 보면 빠르게 결정된다.

암기 포인트

Kruskal은 작은 간선부터 보되 사이클이면 버리고, 끝까지 남은 연결 성분을 잇는 최소 간선을 고른다.

NoteBase

문 10. 허프만 코딩(Huffman coding) 알고리즘을 이용하여 다음 19개 문자로 구성된 문자열을 압축하였다. 압축된 문자열에 사용된 비트 수는?

a b a b a d c d a b a c d a b c d d e

① 39

② 42

③ 44

④ 46

정답 2번

출제 포인트

이 문제는 문자 빈도로 허프만 트리를 만들고, 병합 가중치의 총합으로 전체 압축 비트 수를 구하는 문제이다.

조건 정리

주어진 문자열의 빈도는 a=6, b=4, c=3, d=5, e=1이다.
구해야 할 값은 허프만 코딩 후 전체 문자열을 표현하는 총 비트 수이다.
허프만 코딩에서는 가장 작은 두 빈도를 반복해서 합치며, 각 병합 가중치의 합이 전체 비트 수와 같다.

사용할 공식

전체 비트 수는 허프만 트리의 가중 경로 길이 *WPL*과 같고, 이는 각 단계의 병합 가중치 합과 같다.
즉 WPL = Σ (각 문자 빈도 × 코드 길이) = Σ (각 병합 노드의 가중치)이다.
이 공식을 쓰는 이유는 허프만 코딩이 두 최소 빈도를 반복 결합해 최적 접두 부호를 만들기 때문이다.

계산 과정

빈도를 오름차순으로 정리하면 e=1, c=3, b=4, d=5, a=6이다.
1단계: e(1)와 c(3)를 합쳐 4를 만든다.
2단계: 새로 만든 4와 b(4)를 합쳐 8을 만든다.
3단계: d(5)와 a(6)를 합쳐 11을 만든다.
4단계: 8과 11을 합쳐 19를 만든다.
따라서 전체 비트 수는 $4 + 8 + 11 + 19 = 42$이다.
같은 내용을 코드 길이로 보면 a와 d는 2비트, b는 2비트, c와 e는 3비트가 되어
$6 \times 2 + 5 \times 2 + 4 \times 2 + 3 \times 3 + 1 \times 3 = 42$이다.

NoteBase

선지별 해설

① X
39비트는 계산 결과와 다르다. 허프만 병합 가중치 합은 42이므로 39비트가 아니라 42비트가 된다.

② O
병합 가중치 합이 $4 + 8 + 11 + 19 = 42$이므로 42비트가 맞다.

③ X
44비트는 계산 결과와 다르다. 잘못된 병합 순서를 잡거나 중간 가중치를 잘못 더하면 나올 수 있으나, 올바른 총 비트 수는 42비트이다.

④ X
46비트는 계산 결과와 다르다. 빈도는 맞게 세고도 병합 가중치 총합을 틀리게 더하면 나올 수 있으나, 올바르게는 42비트이다.

오답 포인트

이 문항의 함정은 문자 개수만 세고 끝내거나, 허프만 병합 순서를 정확히 잡지 않은 채 임의로 코드 길이를 배정하는 데 있다. 특히 같은 빈도 4가 두 번 나오므로 중간 노드 4와 문자 b를 다시 결합하는 과정을 빼먹으면 오답이 된다.

다시 나오면 이렇게 판단

허프만 코딩 문제는 먼저 각 문자 빈도를 센 뒤, 가장 작은 두 값을 반복해서 결합한다. 그다음 각 병합 가중치를 모두 더하면 전체 비트 수를 빠르게 구할 수 있다.

암기 포인트

허프만 총 비트 수는 병합 가중치 총합과 같다.

NoteBase

문 11. C 언어에서 행 우선(row major) 순서로 저장되는 2차원 배열 a[4][5]가 있을 때, 나타내는 원소가 다른 것은?

① *(*(a+1)+2)

② *(a[1]+2)

③ **((a+1)+2)

④ a[1][2]

정답 3번

출제 포인트

2차원 배열에서 `a`, `a[i]`, `*(a+i)`의 의미와 행 단위 포인터 연산이 어떤 원소를 가리키는지 묻는 문제이다.

핵심 개념

`a`는 첫 번째 행을 가리키는 포인터처럼 동작하므로 `a+1`은 다음 행으로 이동한다.
`a[1]`과 `*(a+1)`은 둘 다 두 번째 행을 뜻하고, 식에서 사용되면 그 행의 첫 원소 주소로 변환된다.
따라서 `*(*(a+1)+2)`, `*(a[1]+2)`, `a[1][2]`는 모두 `a[1][2]`를 뜻하지만, `**((a+1)+2)`는 `a+3`의 첫 원소인 `a[3][0]`를 뜻한다.

선지별 해설

① X
`*( *(a+1) + 2 )`는 `a+1`로 두 번째 행으로 간 뒤, 그 행에서 2칸 이동한 원소를 읽으므로 `a[1][2]`이다. 다른 원소가 아니라 `a[1][2]`를 나타내는 식이다.

② X
`*(a[1]+2)`에서 `a[1]`은 두 번째 행의 시작 주소로 변환되고, 여기에 2를 더한 뒤 역참조하므로 `a[1][2]`이다. 다른 원소가 아니라 1, 4번과 같은 `a[1][2]`를 나타낸다.

③ O
`**((a+1)+2)`는 `**(a+3)`와 같고, 이는 네 번째 행의 첫 원소를 읽는 식이므로 `a[3][0]`이다. 1, 2, 4번이 가리키는 `a[1][2]`와 다른 원소이다.

④ X
`a[1][2]`는 두 번째 행 세 번째 열의 원소 자체이다. 다른 원소가 아니라 1, 2번과 동일한 `a[1][2]`를 직접 표기한 식이다.

NoteBase

오답 포인트

`a+1`과 `a[1]+2`를 모두 단순히 숫자 1, 2를 더하는 것으로 보면 행 이동과 열 이동이 섞여 오답이 된다. 이 문항의 함정은 `a`는 행 단위로 이동하고, `a[1]`은 이미 특정 행 안에서 열 단위로 이동한다는 차이를 놓치게 만드는 데 있다.

다시 나오면 이렇게 판단

먼저 `a`가 가리키는 대상이 행 전체인지, `a[i]`가 가리키는 대상이 특정 행의 첫 원소인지부터 구분한다. 그다음 `+k`가 행 이동인지 열 이동인지 따져서 마지막 역참조 결과를 인덱스 형태로 바꾸면 된다.

암기 포인트

`a+n`은 n개 행 이동이고, `a[i]+j`는 i행에서 j개 열 이동이므로 `*(*(a+i)+j)=a[i][j]`이다.

NoteBase

문 12. 다음 알고리즘의 시간 복잡도를 빅세타(Θ) 표기법으로 표현한 것은? (단, $n > 0$)

```
int iterate(int n)
{
    int i, count = 0;
    for (i = 1; i < n; i *= 2)
        count++;
    return count;
}
```

① $\Theta(\log n)$

② $\Theta(n)$

③ $\Theta(n \log n)$

④ $\Theta(n^2)$

정답 1번

출제 포인트

이 문제는 반복문에서 제어변수가 2배씩 증가할 때 반복 횟수가 $\log n$ 수준이 된다는 점을 묻는 문제이다.

조건 정리

주어진 값: `` `for (i = 1; i < n; i *= 2)` ``
구해야 할 값: 반복문의 시간 복잡도
주의할 조건: `` `i` ``가 1씩 증가하는 것이 아니라 매 반복마다 2배가 된다.

사용할 공식

반복 횟수를 k라 두면 반복 중의 `` `i` `` 값은 $1, 2, 4, \dots, 2^{k-1}$ 꼴이다.
반복이 끝나는 시점은 $2^k \geq n$가 되는 순간이므로 k는 $\log_2 n$에 비례한다.
이 공식을 쓰는 이유: 제어변수가 덧셈으로 증가하지 않고 곱셈으로 증가하므로, 몇 번 곱해야 n에 도달하는지를 세면 되기 때문이다.

NoteBase

계산 과정

1. 처음 `i = 1`에서 시작한다.
2. 한 번 반복할 때마다 `i`는 `2`배가 되어 1, 2, 4, 8, …로 증가한다.
3. k번 반복 후의 값은 2^k 수준이다.
4. 반복 조건은 `i < n`이므로 $2^k < n$인 동안만 반복한다.
5. 따라서 반복 횟수는 $k \approx \log_2 n$이고 시간 복잡도는 $\Theta(\log n)$이다.

선지별 해설

① O: `i`가 매번 2배가 되므로 반복 횟수는 $\log_2 n$에 비례한다. 빅세타 표기에서는 로그의 밑은 상수 차이이므로 $\Theta(\log n)$이 맞다.
② X: $\Theta(n)$은 `i++`처럼 1씩 증가하여 반복 횟수가 `n`에 비례할 때의 형태이다. 이 코드의 틀린 표현은 `선형 반복`이라고 본 점이며, 올바르게는 `2배씩 증가하는 로그 반복`이다.
③ X: $\Theta(n \log n)$은 보통 선형 반복 안에 로그 반복이 있거나 그 반대처럼 두 단계의 반복 구조가 있을 때 나온다. 이 코드에는 반복문이 하나뿐이다. 올바르게는 `단일 로그 반복이므로 $\Theta(\log n)$`이다.
④ X: $\Theta(n^2)$은 대체로 이중 반복문처럼 `n`이 두 번 곱해질 때 나온다. 이 코드에는 중첩 반복이 없다. 올바르게는 `제어변수가 기하급수적으로 증가하므로 $\Theta(\log n)$`이다.

오답 포인트

많이 틀리는 지점은 `for`문이 보이면 곧바로 n번 반복한다고 생각하는 점이다. 이 문제의 핵심은 반복문의 개수가 아니라 증가 방식이다. `i *= 2`를 `i++`처럼 읽으면 ②로 오답 처리되기 쉽다.

다시 나오면 이렇게 판단

반복문의 시간 복잡도는 먼저 제어변수가 `+c`로 증가하는지, `*c`로 증가하는지를 본다. `+c`이면 선형 $\Theta(n)$을, `*c`이면 로그 $\Theta(\log n)$을 우선 판단 기준으로 잡는다.

암기 포인트

`i += c`는 선형, `i *= c`는 로그라는 구분이 시간 복잡도 판단의 기본이다.

NoteBase

문 13. 다음은 동적 메모리 할당을 통해 구조체 공간을 확보하고 이름과 학번을 입력받아 구조체에 저장하는 C 언어 함수이다. ㉠, ㉡에 들어갈 내용으로 옳게 짝지어진 것은?

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

struct student{
    char name[20];
    int id;
};

struct student *alloc()
{
    struct student *p;
    p = _____㉠_____;
    if(p==NULL){
        printf("Memory allocation error.");
        return p;
    }
    printf("Student name?\n");
    gets(p->name);
    printf("Student id?\n");
    scanf("%d", ___㉡___);
    return p;
}
```

	㉠	㉡
①	(struct student)malloc(sizeof(struct student))	&((*p).id)
②	(struct student *)malloc(sizeof(struct student))	p->id
③	(struct student)malloc(sizeof(struct student))	(*p).id
④	(struct student *)malloc(sizeof(struct student))	&(p->id)

정답 4번

출제 포인트

이 문제는 동적 메모리 할당에서 포인터형을 맞게 선언하는 방법과 구조체 포인터의 멤버를 입력할 때 주소 연산자를 붙이는 규칙을 묻는 문제이다.

핵심 개념

malloc은 확보한 메모리의 시작 주소를 반환하므로 구조체 변수 자체가 아니라 구조체 포인터에 대입해야 한다.
구조체 포인터 `p`의 멤버 접근은 `p->id`와 `(*p).id`가 같은 뜻이다.
`scanf("%d", ...)`는 정수값이 아니라 정수를 저장할 메모리 주소를 인자로 받으므로 `p->id`가 아니라 `&(p->id)` 또는 `&((*p).id)`가 와야 한다.

NoteBase

선지별 해설

① X
㉠의 `(struct student)malloc(sizeof(struct student))`가 틀리다. malloc의 반환값은 주소이므로 `struct student` 자체가 아니라 `struct student *`로 받아야 한다. 올바르게는 `(struct student *)malloc(sizeof(struct student))`이다. ㉡의 `&((*p).id)`는 주소를 넘기는 올바른 표현이다.

② X
㉠의 `(struct student *)malloc(sizeof(struct student))`는 맞다. 그러나 ㉡의 `p->id`는 정수값 자체이므로 `scanf`의 두 번째 인자로 쓸 수 없다. 올바르게는 `&(p->id)`이다.

③ X
㉠의 `(struct student)malloc(sizeof(struct student))`는 ①과 같은 이유로 틀리다. 또한 ㉡의 `(*p).id`도 멤버값 자체일 뿐 주소가 아니다. 올바르게는 `&(p->id)` 또는 `&((*p).id)`이다.

④ O
㉠의 `(struct student *)malloc(sizeof(struct student))`는 동적으로 확보한 메모리의 시작 주소를 구조체 포인터에 저장하는 올바른 표현이다. ㉡의 `&(p->id)`도 `id` 필드의 주소를 넘기므로 `scanf("%d", ...)`와 맞는다.

오답 포인트

구조체 멤버 접근식 `p->id`와 주소식 `&(p->id)`를 같은 것으로 착각하면 ②와 ③을 고르기 쉽다. 또 캐스팅에서 `*` 하나가 빠진 `(struct student)`를 보고도 모양만 비슷하다고 넘기면 ①이나 ③으로 흔들리기 쉽다.

다시 나오면 이렇게 판단

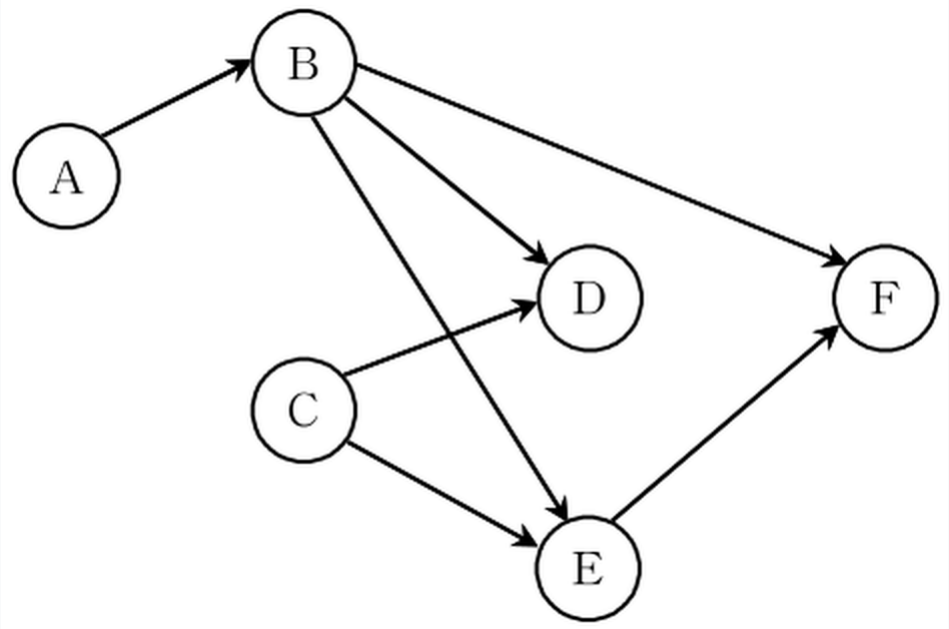
대입문의 오른쪽이 `malloc`이면 반환형이 주소라는 점부터 확인한다. 따라서 왼쪽 변수형과 캐스팅은 포인터형이어야 한다. `scanf`가 오면 입력받을 값이 아니라 저장 위치의 주소가 필요한지 먼저 판단하면 `p->id`와 `&(p->id)`를 바로 구분할 수 있다.

암기 포인트

`malloc`은 주소를 준다, `scanf`는 주소를 받는다.

NoteBase

문 14. 다음 그래프의 위상 정렬(topological sorting) 결과로 나올 수 있는 위상 순서(topological order)로 옳지 않은 것은?



① A, B, C, D, E, F

② C, A, B, E, D, F

③ A, C, B, E, D, F

④ C, A, D, B, E, F

정답 4번

출제 포인트

이 문제는 방향 비순환 그래프의 위상 정렬에서 각 간선의 시작 정점이 도착 정점보다 반드시 먼저 나와야 한다는 선후 제약을 판정하는 문제이다.

올바른 순서

1. A와 C는 진입차수가 0이므로 시작 부분에 올 수 있다.
2. B는 A에서 들어오는 간선이 있으므로 A 뒤에 와야 한다.
3. D와 E는 각각 B와 C의 선행을 필요로 하므로 B와 C가 나온 뒤에 와야 한다.
4. F는 B와 E가 모두 나온 뒤에 와야 한다.

왜 이 순서: 위상 정렬은 모든 간선 $u \rightarrow v$에 대해 u가 v보다 먼저 나와야 하므로, 선행 정점이 뒤로 밀리면 그 순서는 위상 순서가 될 수 없다.

NoteBase

선지별 해설

① O
- A가 B보다 먼저 나오고, C가 D와 E보다 먼저 나오며, E가 F보다 먼저 나오므로 모든 간선 방향을 거스르지 않는다.

② O
- C와 A가 먼저 나온 뒤 B가 오고, E와 D도 각각 B와 C 뒤에 배치되어 있으며, F는 E 뒤에 있으므로 위상 순서로 가능하다.

③ O
- A와 C가 선행 정점으로 먼저 나오고, B 뒤에 E와 D가 배치되며, F가 E 뒤에 있어 모든 선후 제약을 만족한다.

④ X
- 틀린 부분: D가 B보다 먼저 배치되어 있다.
- 올바르게는: 간선 B→D가 있으므로 B가 반드시 D보다 먼저 와야 한다. 따라서 C, A, B, D, E, F 또는 C, A, B, E, D, F처럼 고쳐야 한다.

오답 포인트

위상 정렬은 하나의 정답 순서를 외우는 문제가 아니라 모든 간선의 선후관계를 지키는지만 보면 된다. 이 문항에서는 D와 E가 B, C 뒤에 와야 하고 F는 E 뒤에 와야 하는데, ④처럼 B보다 D를 앞세우면 바로 탈락한다.

다시 나오면 이렇게 판단

위상 정렬 문제는 각 간선을 선행 제약으로 바꾸어 본다. A→B이면 A가 먼저, B→D이면 B가 먼저라는 식으로 확인한 뒤, 선지에서 한 번이라도 뒤집힌 제약이 있으면 그 선지는 옳지 않다.

암기 포인트

위상 정렬은 간선의 화살표를 거슬러 올라가면 탈락이며, 시작점은 앞에 오고 도착점은 뒤에 온다.

NoteBase

문 15. 공백 상태인 이진 탐색 트리(binary search tree)에 1부터 5까지의 정수를 삽입하고자 한다. 삽입 결과, 이진 탐색 트리의 높이가 가장 낮은 삽입 순서는?

① 1, 2, 3, 4, 5

② 1, 3, 2, 4, 5

③ 3, 1, 2, 4, 5

④ 4, 3, 5, 2, 1

정답 3번

출제 포인트

이 문제는 이진 탐색 트리에서 삽입 순서에 따라 트리 높이가 어떻게 달라지는지를 묻는 절차형 문제이다.

올바른 순서

3, 1, 2, 4, 5 순서로 삽입하면 루트가 3이 되어 좌우로 값이 분산된다. 이후 1과 2가 왼쪽 서브트리를, 4와 5가 오른쪽 서브트리를 이루어 가장 긴 경로가 3단계로 유지된다. 정답을 가르는 핵심은 첫 삽입값이 중앙값 3이냐는 점이다.

선지별 해설 ①

X. 1, 2, 3, 4, 5를 순서대로 넣으면 1이 루트가 되고 나머지가 모두 오른쪽으로만 붙어 한쪽으로 치우친 사슬형 트리가 된다. 틀린 부분은 첫 삽입값이 1이라 균형이 전혀 잡히지 않는다는 점이다. 올바르게는 중앙값 3을 먼저 넣는 3, 1, 2, 4, 5가 가장 낮은 높이를 만든다.

선지별 해설 ②

X. 1, 3, 2, 4, 5는 1이 루트가 되고 3이 오른쪽 자식이 되므로 처음부터 오른쪽으로 치우친다. 2가 3의 왼쪽에 붙어 일부 완화되지만 가장 긴 경로는 1-3-4-5로 더 길다. 틀린 부분은 첫 삽입값이 1인 상태에서 높이 최소화를 기대한 점이다. 올바르게는 3을 루트로 두어 좌우를 먼저 나누는 3, 1, 2, 4, 5가 된다.

선지별 해설 ③

O. 3을 먼저 넣으면 루트가 중앙값이 되어 1, 2는 왼쪽으로, 4, 5는 오른쪽으로 분산된다. 결과적으로 왼쪽은 3-1-2, 오른쪽은 3-4-5 형태가 되어 가장 긴 경로가 가장 짧다.

선지별 해설 ④

X. 4, 3, 5, 2, 1은 처음에는 좌우가 나뉘는 듯하지만 3 밑으로 2와 1이 연속해서 붙어 왼쪽 가지가 길어진다. 틀린 부분은 4를 루트로 잡아 1, 2, 3이 한쪽에 몰리게 만든 점이다. 올바르게는 1부터 5까지의 중앙값 3을 루트로 두는 3, 1, 2, 4, 5가 가장 낮은 높이를 만든다.

NoteBase

오답 포인트

삽입 결과를 끝까지 그리지 않고 첫 두 단계만 보고 균형이라고 판단하면 ④에 끌리기 쉽다. 이 문항의 함정은 중간에 좌우가 한 번 나뉘어 보여도 이후 삽입이 같은 쪽에 연속되면 높이가 다시 커진다는 점이다.

다시 나오면 이렇게 판단

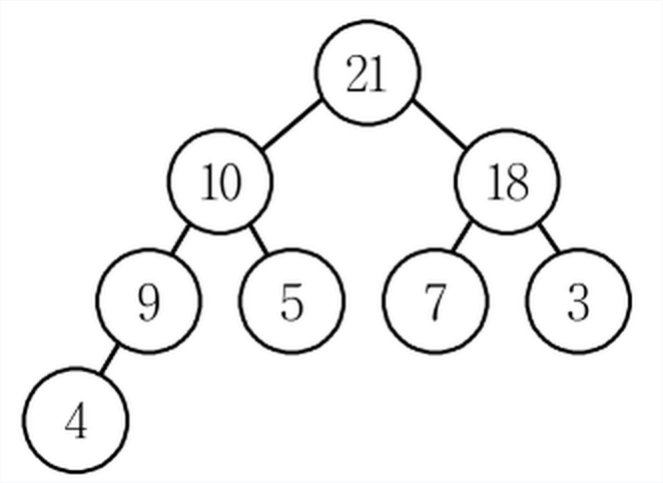
BST 높이 최소 문제는 먼저 루트가 중앙값에 가까운지 본다. 그다음 남은 값들이 루트의 왼쪽과 오른쪽에 고르게 분산되는지 확인하면 가장 낮은 높이를 빠르게 찾을 수 있다.

암기 포인트

BST 높이를 낮추려면 먼저 중앙값을 루트로 둔다는 기준이 가장 중요하다.

NoteBase

문 16. 다음 최대 힙(max heap)에 13을 삽입하는 연산을 수행한 후에 루트 노드를 삭제하는 연산을 수행하였다. 그 결과, 루트 노드의 오른쪽 자식 노드의 값으로 옳은 것은? (단, 삽입 및 삭제 연산은 최대 힙으로의 복원 과정을 포함한다)



① 9

② 10

③ 13

④ 18

정답 1번

출제 포인트

이 문제는 최대 힙 도식에서 13의 삽입에 따른 상향 이동과 루트 삭제에 따른 하향 이동을 순서대로 적용하여 최종 부모-자식 관계를 판단하는 문제이다.

NoteBase

선지별 해설

① O

- 초기 힙의 레벨 순서는 21, 10, 18, 9, 5, 7, 3, 4이다. 13을 다음 위치에 삽입하면 9의 오른쪽 자식으로 들어가고, 13>9이므로 교환한 뒤 다시 13>10이므로 한 번 더 교환된다. 삽입 후 힙은 21, 13, 18, 10, 5, 7, 3, 4, 9가 된다.
- 이제 루트 21을 삭제하고 마지막 원소 9를 루트로 올리면 9, 13, 18, 10, 5, 7, 3, 4가 된다. 루트 9는 더 큰 자식 18과 교환되고, 그 뒤에는 자식 7과 3보다 크므로 멈춘다. 최종 힙은 18, 13, 9, 10, 5, 7, 3, 4이므로 루트의 오른쪽 자식은 9이다.

② X

- 10은 최종 힙에서 루트의 왼쪽 자식 13의 왼쪽 자식으로 남는다.
- 틀린 표현은 "루트의 오른쪽 자식이 10이다"이다. 올바르게는 "10은 세 번째 레벨로 내려가고 루트의 오른쪽 자식은 9이다"이다.

③ X

- 13은 삽입 과정에서 9, 10과 교환되어 루트의 왼쪽 자식까지 올라간다. 삭제 후에도 13은 그대로 루트의 왼쪽 자식이다.
- 틀린 표현은 "루트의 오른쪽 자식이 13이다"이다. 올바르게는 "13은 최종 힙에서 루트의 왼쪽 자식이다"이다.

④ X

- 18은 루트 삭제 후 9와 비교될 때 더 큰 자식이므로 루트 자리로 올라간다. 따라서 18은 오른쪽 자식이 아니라 최종 루트가 된다.
- 틀린 표현은 "루트의 오른쪽 자식이 18이다"이다. 올바르게는 "18은 최종 루트이고, 루트의 오른쪽 자식은 9이다"이다.

오답 포인트

삽입 후 상태만 보고 13이나 10을 고르거나, 삭제 연산에서 마지막 원소를 루트로 올린 뒤 한 번만 비교하고 멈추는 경우가 많다. 이 문항은 삽입의 상향 이동과 삭제의 하향 이동을 모두 끝까지 수행해야 정답이 나온다.

다시 나오면 이렇게 판단

힙 연산 문항은 배열의 레벨 순서를 먼저 적고, 삽입은 부모와 비교하며 위로 올리고, 삭제는 마지막 원소를 루트로 보낸 뒤 더 큰 자식과 비교하며 아래로 내리는 순서로 추적하면 된다.

암기 포인트

최대 힙은 삽입은 위로, 루트 삭제 복원은 더 큰 자식 쪽으로 아래로 이동한다.

NoteBase

문 17. 다음은 구현하고자 하는 원형 큐(circular queue)에 대한 <조건>과 <설명>이다. <설명>의 괄호 안에 들어갈 수로 옳게 짝지어진 것은?

<조건>
크기가 6이고 인덱스가 0부터 5까지인 배열로 구현된다.
공백 상태에서만 front와 rear의 값이 서로 같으며, 그 외에는 값이 서로 다르다.
(front + 1) 6은 다음에 삭제할 원소 위치를 가리킨다.
(rear + 1) 6은 다음에 삽입할 원소 위치를 가리킨다.
<설명>
최대 (㉠)개의 원소를 저장할 수 있다.
front=5이고 rear=3인 경우 저장된 원소 수는 (㉡)개이다.
front=(㉢)이고 rear=2인 경우는 포화 상태를 의미한다.

① ㉠: 6, ㉡: 5, ㉢: 1
② ㉠: 6, ㉡: 4, ㉢: 1
③ ㉠: 5, ㉡: 5, ㉢: 3
④ ㉠: 5, ㉡: 4, ㉢: 3

정답 4번

출제 포인트

이 문제는 배열 크기 6인 원형 큐에서 한 칸을 비워 두는 구현 규칙을 바탕으로 최대 저장 수, 현재 저장 원소 수, 포화 조건을 계산하는 문제이다.

조건 정리

배열 크기는 6이고 인덱스는 0부터 5까지이다.
공백 상태에서만 front와 rear가 같고, 그 외에는 서로 다르다.
다음 삭제 위치는 (front+1) 6, 다음 삽입 위치는 (rear+1) 6이므로 이 구현은 공백과 포화를 구분하려고 한 칸을 비워 두는 원형 큐이다.

NoteBase

사용할 공식

최대 저장 원소 수는 $N-1$이다.
저장 원소 수는 (rear-front+N) N이다.
포화 상태는 (rear+1) N = front일 때이다.

이 공식을 쓰는 이유:
이 원형 큐는 front와 rear가 공백 상태에서만 같아야 하므로 배열 한 칸을 항상 비워 두어야 한다. 따라서 배열 크기 전체를 저장 공간으로 쓰지 않고, 원소 수와 포화 조건도 그 규칙에 맞춰 계산한다.

계산 과정

1. 배열 크기가 6이므로 최대 저장 원소 수는 $6-1=5$이다. 따라서 ㄱ은 5이다.
2. $front=5$, $rear=3$이면 저장 원소 수는 (3-5+6) 6 = 4이다. 따라서 ㄴ은 4이다.
3. 포화 상태는 (rear+1) 6 = front이다. $rear=2$이면 front=(2+1) 6 = 3이어야 하므로 ㄷ은 3이다.

선지별 해설

① X: ㄱ이 6으로 틀렸다. 이 구현은 한 칸을 비워 두므로 최대 저장 수는 6이 아니라 5이다. ㄴ과 ㄷ도 각각 5, 1로 적혀 있는데 올바르게는 4, 3이다.
② X: ㄱ이 6으로 틀렸다. 배열 크기와 최대 저장 수를 같게 본 값이다. ㄴ은 맞지만 ㄷ이 1로 틀렸고, 올바르게는 포화 조건 (rear+1) 6 = front에 따라 3이다.
③ X: ㄱ과 ㄷ은 맞지만 ㄴ이 5로 틀렸다. $front=5$, $rear=3$에서 저장 원소 수를 직선 배열처럼 세면 안 되고 (rear-front+6) 6으로 계산해야 하므로 올바르게는 4이다.
④ O: ㄱ=5, ㄴ=4, ㄷ=3으로 세 값이 모두 맞다. 최대 저장 수, 저장 원소 수, 포화 조건을 모두 정확히 적용한 선택지이다.

오답 포인트

이 문항의 함정은 배열 크기 6을 그대로 최대 저장 수 6으로 착각하는 점, 원형으로 감긴 구간의 원소 수를 직선 배열처럼 세는 점, 포화 조건을 공백 조건과 혼동하는 점이다. 특히 front와 rear가 같은 상태는 공백이고, 포화는 다음 삽입 위치가 front와 겹치는 상태라는 구분이 핵심이다.

다시 나오면 이렇게 판단

원형 큐 문항은 먼저 한 칸 비움 구현인지 확인한 뒤 최대 저장 수는 $N-1$로 판단하고, 현재 원소 수는 (rear-front+N) N, 포화 여부는 (rear+1) N = front로 판단한다.

암기 포인트

한 칸 비우는 원형 큐는 공백이면 $front=rear$, 포화이면 (rear+1) N = front, 최대 저장 수는 $N-1$이다.

NoteBase

문 18. 다음 C 언어 함수를 이용하여 do_something(660, 735)를 수행한 결과 값은?

```
unsigned int do_something(unsigned int x, unsigned int y)
{
    while (y!=0) {
        int temp;
        temp = x % y;
        x = y;
        y = temp;
    }
    return x;
}
```

① 5
② 9
③ 15
④ 25

정답 3번

출제 포인트

이 문제는 C 코드가 유클리드 호제법으로 최대공약수를 구한다는 점을 파악하고, do_something(660, 735)의 반환값을 계산하는 문제이다.

조건 정리

주어진 값: $x = 660, y = 735$이다.
구해야 할 값: while 문이 끝난 뒤 반환되는 x의 값이다.
주의할 조건: 이 함수는 매 반복마다 $x \leftarrow y$, y ≤ftarrow x y로 갱신하므로 두 수의 최대공약수를 구한다.

NoteBase

사용할 공식

유클리드 호제법을 사용한다.

(x, y) = (y, x y)

이 공식을 쓰는 이유:

코드의 temp = x % y; x = y; y = temp;가 정확히 나머지 연산으로 두 수를 줄여 가는 유클리드 호제법의 갱신식이기 때문이다.

계산 과정

1. 처음 값은 $x = 660, y = 735$이다. 660 735 = 660이므로 다음 값은 $x = 735, y = 660$이다.
2. 735 660 = 75이므로 다음 값은 $x = 660, y = 75$이다.
3. 660 75 = 60이므로 다음 값은 $x = 75, y = 60$이다.
4. 75 60 = 15이므로 다음 값은 $x = 60, y = 15$이다.
5. 60 15 = 0이므로 다음 값은 $x = 15, y = 0$이다.
6. $y = 0$이 되어 while 문이 종료되고 $x = 15$를 반환한다.

선지 판단

① X

- 5는 반환값이 아니다. 계산 결과는 15이다.
- 틀린 표현 → 올바르게는 반환값이 5가 아니라 15이다.

② X

- 9는 중간 나머지나 최대공약수로 나오지 않는다. 계산 결과는 15이다.
- 틀린 표현 → 올바르게는 반환값이 9가 아니라 15이다.

③ O

- 반복 종료 시 $x = 15$이므로 반환값은 15이다. 두 수 660과 735의 최대공약수도 15이다.

④ X

- 25는 660과 735를 동시에 나누지 못하므로 최대공약수가 될 수 없다. 계산 결과도 15이다.
- 틀린 표현 → 올바르게는 반환값이 25가 아니라 15이다.

계산 결과 15이므로 정답은 3번이다.

NoteBase

오답 포인트

이 함수가 단순 나머지 계산이 아니라 최대공약수를 구하는 유클리드 호제법이라는 점을 놓치면 첫 나머지 660이나 중간 값 75, 60 같은 값에 끌릴 수 있다. 또한 종료 조건이 $y = 0$일 때 반환하는 값이 마지막 0이 아니라 그때의 x라는 점이 함정이다.

다시 나오면 이렇게 판단

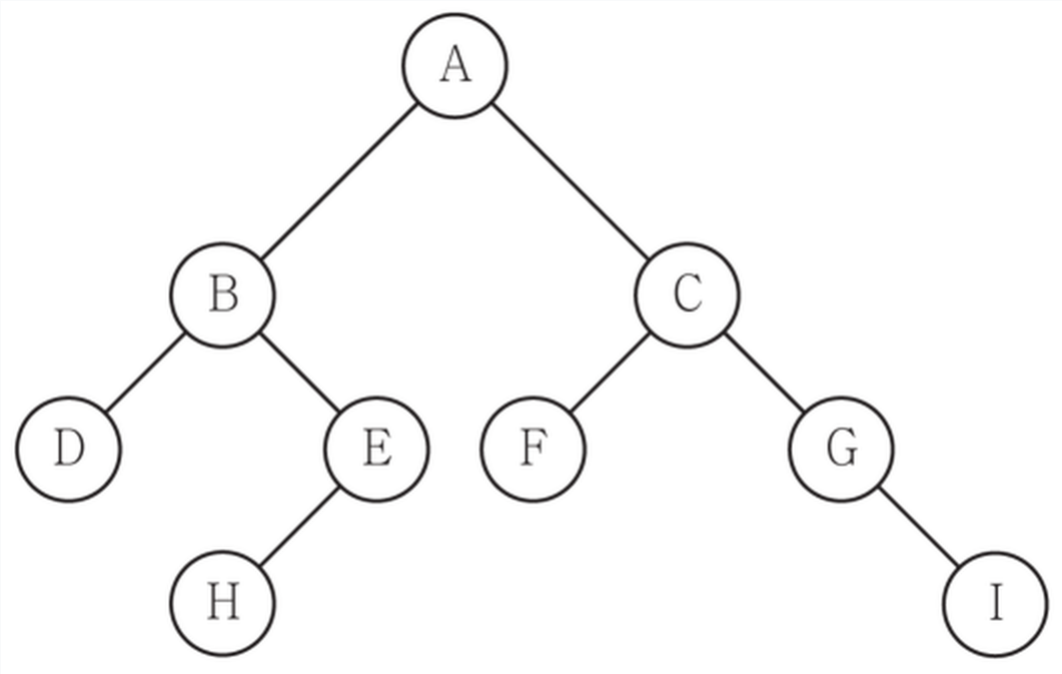
이런 코드 추적 문제는 반복문의 갱신식이 무엇을 계산하는지 먼저 확인한 뒤, 각 단계의 (x, y) 변화를 끝까지 적어 보고 $y = 0$이 되는 순간의 x를 답으로 판단한다.

암기 포인트

유클리드 호제법은 x y를 반복하다가 $y = 0$이 되면 그때의 x가 최대공약수이다.

NoteBase

문 19. 다음 이진 트리에 대하여 전위순회(preorder traversal), 중위순회(inorder traversal), 후위순회(postorder traversal)를 수행하였다. 3가지 순회 방법 중 2가지 이상에서 방문 순서가 동일한 노드 수는? (예를 들어, 3개의 노드 A, B, C를 갖는 이진 트리에서 전위순회 : B-A-C, 중위순회 : A-B-C, 후위순회 : A-C-B의 결과를 얻었다면, 중위순회 및 후위순회에서 노드 A에 대한 방문 순서가 동일하고 전위순회 및 중위순회에서 노드 C에 대한 방문 순서가 동일하므로, 2가지 이상에서 방문 순서가 동일한 노드 수는 2개이다)



① 2

② 3

③ 4

④ 5

정답 4번

출제 포인트

이 문제는 이진 트리 도식에서 전위순회, 중위순회, 후위순회의 방문 순서를 각각 구한 뒤, 같은 순번에 놓이는 노드를 세는 문제이다.

NoteBase

선지별 해설

① X
- 2개라는 판단은 일부 노드만 확인한 경우에 나온다.
- 전위순회는 A-B-D-E-H-C-F-G-I, 중위순회는 D-B-H-E-A-F-C-G-I, 후위순회는 D-H-E-B-F-I-G-C-A이다.
- 같은 방문 순서가 2가지 이상에서 일치하는 노드는 B(전위 2, 중위 2), D(중위 1, 후위 1), E(전위 4, 중위 4), G(전위 8, 중위 8), I(전위 9, 중위 9)로 5개이다.
- 2개이다 → 올바르게는 5개이다.

② X
- 3개라고 보면 B, D, E 정도만 세고 뒤쪽 노드의 일치를 놓친 경우이다.
- G와 I도 각각 전위순회와 중위순회에서 같은 순번에 방문된다.
- 3개이다 → 올바르게는 5개이다.

③ X
- 4개라고 보면 B, D, E, G까지만 세고 I의 일치를 빠뜨린 경우이다.
- I는 전위순회 9번째, 중위순회 9번째로 방문되므로 포함된다.
- 4개이다 → 올바르게는 5개이다.

④ O
- 같은 방문 순서가 2가지 이상에서 일치하는 노드는 B, D, E, G, I의 5개이다.
- 따라서 5개를 제시한 ④가 옳다.

오답 포인트

순회 결과를 끝까지 적지 않고 중간에서 세기를 멈추면 G와 I 같은 뒤쪽 노드의 순번 일치를 놓치기 쉽다. 또한 같은 두 순회만 비교해도 된다는 조건을 놓치고 세 순회가 모두 같아야 한다고 오해하면 개수를 더 적게 세게 된다.

다시 나오면 이렇게 판단

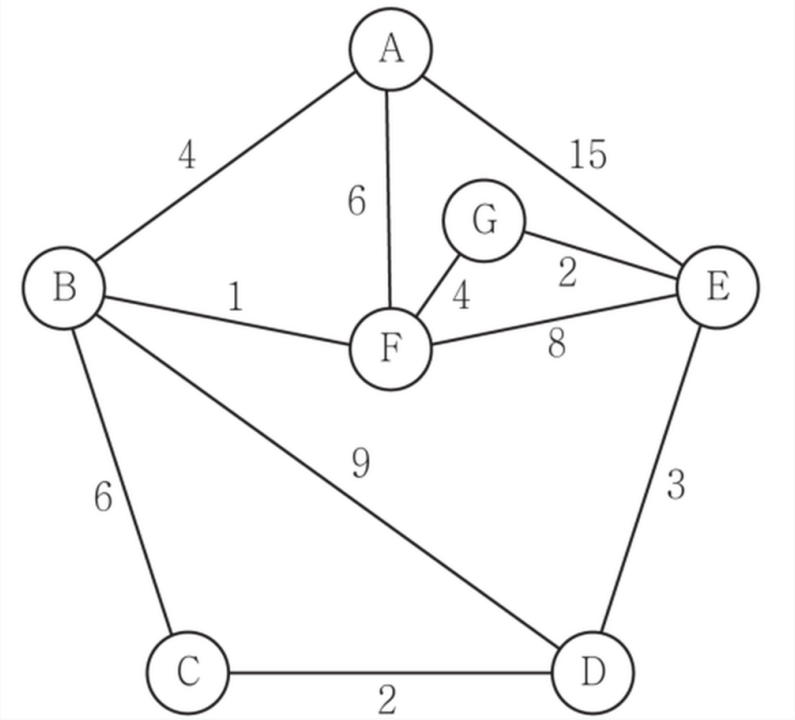
트리 순회 비교 문제는 먼저 세 순회 결과를 끝까지 한 줄씩 적은 뒤, 각 노드별로 전위-중위-후위의 순번을 표처럼 대응시켜 본다. 그다음 두 순회 이상에서 같은 숫자가 한 번이라도 나오면 해당 노드를 센다.

암기 포인트

전위는 루트-왼쪽-오른쪽, 중위는 왼쪽-루트-오른쪽, 후위는 왼쪽-오른쪽-루트이며, 비교형 문제는 순회명보다 각 노드의 방문 순번 대응이 핵심이다.

NoteBase

문 20. Dijkstra 알고리즘을 이용하여 다음 그래프의 정점 A에서 각 정점으로의 최단경로를 구할 때, 최단경로 발견 순서가 정점 G 바로 다음인 정점은? (단, 간선 옆의 수는 각 정점들 사이의 거리를 나타낸다)



① 정점 C

② 정점 D

③ 정점 E

④ 정점 F

정답 1번

출제 포인트

이 문제는 가중치 그래프에서 Dijkstra 알고리즘을 적용하여 각 단계의 임시 거리 중 최솟값을 고정하는 순서를 판단하는 문제이다.

NoteBase

선지별 해설

① O
- A에서 시작하면 초기 거리는 B=4, F=6, E=15이다. 먼저 B가 확정되고, B를 거치면 F=5, C=10, D=13으로 갱신된다. 다음으로 F가 확정되고, F를 거치면 G=9, E=13이 된다. 그다음 G가 확정되면 E=11로 줄어들지만 C=10이 더 작으므로 G 바로 다음에 확정되는 정점은 C이다.

② X
- D는 G 직후 정점이 아니다. G까지 확정한 뒤의 후보 거리는 C=10, E=11, D=13이므로 D보다 C가 먼저 선택된다.
- 틀린 표현은 "정점 D가 G 바로 다음에 확정된다"이다. 올바르게는 "정점 D는 C와 E가 처리된 뒤 최단거리 12로 확정된다"이다.

③ X
- E는 G를 거치며 거리 11로 갱신되지만, 같은 시점의 C 거리는 10이므로 E가 먼저 확정되지 않는다.
- 틀린 표현은 "정점 E가 G 바로 다음에 확정된다"이다. 올바르게는 "정점 E는 C 다음에 확정된다"이다.

④ X
- F는 G보다 먼저 확정된다. B를 거쳐 F의 거리가 5가 되므로 G의 거리 9보다 앞서 선택된다.
- 틀린 표현은 "정점 F가 G 바로 다음에 확정된다"이다. 올바르게는 "정점 F는 B 다음, G 이전에 확정된다"이다.

오답 포인트

G를 거친 뒤 E의 거리가 13에서 11로 줄어드는 점만 보고 E를 바로 선택하기 쉽다. 그러나 Dijkstra 알고리즘은 매 단계에서 갱신된 값들 중 최솟값을 고정하므로, 같은 시점의 C=10을 함께 비교해야 한다.

다시 나오면 이렇게 판단

시작 정점에서 임시 거리를 적은 뒤, 가장 작은 값을 하나씩 확정하면서 그 정점을 거칠 때마다 인접 정점 거리를 다시 계산하면 된다. "갱신되었다"와 "가장 작다"를 구분하면 확정 순서를 흔들리지 않고 판단할 수 있다.

암기 포인트

Dijkstra는 매 단계마다 현재 임시 거리의 최솟값을 확정하며, 갱신 여부보다 최솟값 비교가 우선이다.

NoteBase