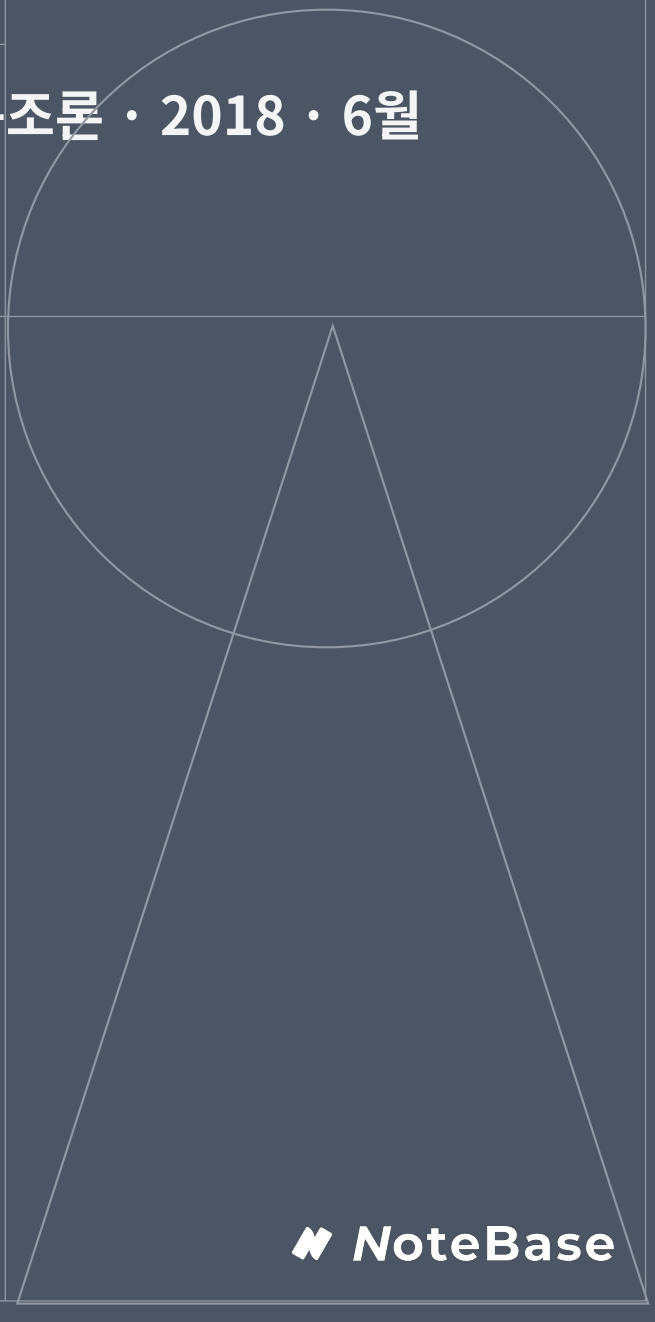
기출OX 해설집

공무원/서울시/7급 · 자료구조론 · 2018 · 6월



NoteBase

기출OX 해설집

공무원/서울시/7급 · 자료구조론 · 2018 · 6월

공무원/서울시/7급 · 자료구조론 · 2018 · 6월

문 1. C언어에서 정수형 2차원 배열 A[3][4]가 <보기>와 같이 초기화 선언된 경우, 원소 A[2][3]의 초기 값은?

<보기>
int A[3][4] = { 1, 2, 3, 4, 5, 6 };

① 0
② 4
③ 5
④ 6

정답 1번

출제 포인트

이 문제는 2차원 배열의 행 우선 저장 순서와 부분 초기화 시 남는 원소가 0으로 초기화되는 규칙을 적용할 수 있는지를 묻는 문제이다.

사례 정리

핵심 사실 1: 배열은 `int A[3][4]`이므로 전체 원소 수는 12개이다.
핵심 사실 2: 초기값은 `1, 2, 3, 4, 5, 6`까지만 주어졌다.
판단에 중요한 조건: C언어에서는 배열 초기화가 앞에서부터 행 우선으로 채워지고, 남는 원소는 0으로 초기화된다.

핵심 개념

2차원 배열 초기화 = 초기값 목록이 앞에서부터 `A[0][0]`, `A[0][1]` 순서로 채워지는 방식이다.
적용 요건: 선언된 전체 원소 수보다 초기값 개수가 적으면 나머지 원소는 0이 된다.

적용 기준

이 사례에서는 초기값 6개를 2차원 배열에 앞에서부터 대입한 뒤, 남는 6개 원소가 0이 되는 규칙을 적용해야 한다. 실제 배치는 `A[0][0]=1`, `A[0][1]=2`, `A[0][2]=3`, `A[0][3]=4`, `A[1][0]=5`, `A[1][1]=6`이고 그 뒤 원소들은 모두 0이다. 따라서 `A[2][3]`은 0이다.

NoteBase

선지별 해설

① O
0은 `A[2][3]`의 값이 맞다. 초기값 6개만 채운 뒤 남는 원소는 모두 0으로 초기화되므로 마지막 원소인 `A[2][3]`도 0이 된다.

② X
4는 `A[2][3]`의 값이 아니다. 4는 앞에서 네 번째로 채워져 `A[0][3]`에 들어간다. 올바르게는 `A[0][3]=4`이다.

③ X
5는 `A[2][3]`의 값이 아니다. 5는 둘째 행의 첫 원소인 `A[1][0]`에 들어간다. 올바르게는 `A[1][0]=5`이다.

④ X
6은 `A[2][3]`의 값이 아니다. 6은 둘째 행의 둘째 원소인 `A[1][1]`에 들어간다. 올바르게는 `A[1][1]=6`이다.

오답 포인트

초기값이 6까지만 보인다는 이유로 마지막 원소에도 6이 들어간다고 착각하거나, 2차원 배열을 행 단위로 끝까지 채운 뒤 남는 칸을 별도로 생각하지 못하면 틀리기 쉽다. 이 문제의 함정은 `A[2][3]`이라는 인덱스를 보고 단순히 여섯 번째 값과 연결하는 오해에 있다.

다시 나오면 이렇게 판단

배열 초기화 문제는 먼저 전체 크기를 계산하고, 초기값을 `A[0][0]`부터 행 우선으로 순서대로 적어 본 뒤, 남는 칸이 있으면 0으로 처리하는 방식으로 판단한다.

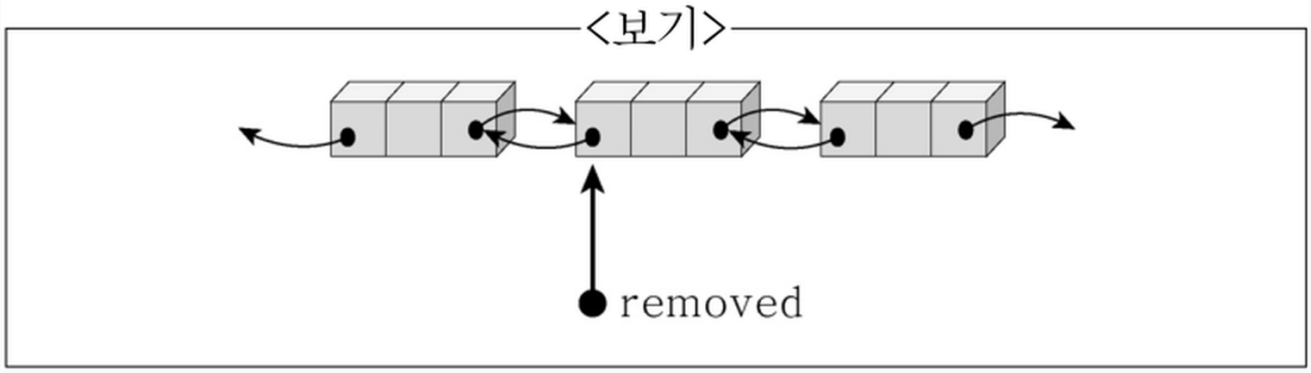
암기 포인트

2차원 배열 부분 초기화는 앞에서부터 채우고 남는 칸은 0이다.

NoteBase

문 2. <보기>와 같은 이중 연결 리스트(doubly linked list)에서 removed가 가리키는 노드를 삭제하려고 한다. 이를 수행하기 위한 2줄의 C/C++ 문장으로 가장 옳은 것은? (단, 노드에서 좌/우 노드를 가리키는 포인터 항목은 각각 llink, rlink이며, removed는 null이 아니다.)

<보기>



① removed → llink → rlink = removed → rlink;
removed → rlink → llink = removed → llink;

② removed → llink → llink = removed → llink;
removed → rlink → rlink = removed → rlink;

③ removed → llink → rlink = removed → llink;
removed → rlink → llink = removed → rlink;

④ removed → llink → rlink = removed → rlink → llink;
removed → rlink → llink = removed → llink → rlink;

정답 1번

출제 포인트

이 문제는 도식에서 removed의 좌우 인접 노드를 확인한 뒤, 이중 연결 리스트에서 노드 삭제가 왼쪽 노드의 rlink를 오른쪽 노드로, 오른쪽 노드의 llink를 왼쪽 노드로 다시 연결하는 연산임을 묻는다.

NoteBase

선지별 해설

① O: `removed->llink->rlink = removed->rlink;`는 삭제될 노드의 왼쪽 노드가 오른쪽 노드를 가리키게 한다. `removed->rlink->llink = removed->llink;`는 삭제될 노드의 오른쪽 노드가 왼쪽 노드를 가리키게 한다. removed를 건너뛰도록 양쪽을 서로 다시 잇는 정확한 삭제 코드이다.
② X: `removed->llink->llink = removed->llink;`는 왼쪽 이웃 노드의 llink를 자기 자신 쪽으로 바꾸는 잘못이다. 삭제에는 왼쪽 이웃의 `rlink`를 수정해야 한다. 올바르게는 `removed->llink->rlink = removed->rlink;`이다. `removed->rlink->rlink = removed->rlink;`도 오른쪽 이웃 노드의 rlink를 자기 자신 쪽으로 바꾸는 잘못이다. 올바르게는 `removed->rlink->llink = removed->llink;`이다.
③ X: `removed->llink->rlink = removed->llink;`는 왼쪽 이웃 노드의 rlink가 오른쪽 노드가 아니라 왼쪽 노드 자신을 가리키게 되어 자기 루프가 된다. 올바르게는 `removed->llink->rlink = removed->rlink;`이다. `removed->rlink->llink = removed->rlink;`도 오른쪽 이웃 노드의 llink가 왼쪽 노드가 아니라 자기 자신을 가리키게 만든다. 올바르게는 `removed->rlink->llink = removed->llink;`이다.
④ X: `removed->rlink->llink`와 `removed->llink->rlink`는 삭제 전에는 둘 다 removed를 가리키는 링크 값이다. 따라서 `removed->llink->rlink = removed->rlink->llink;`와 `removed->rlink->llink = removed->llink->rlink;`는 양쪽 링크를 다시 removed로 두는 셈이어서 삭제가 수행되지 않는다. 올바르게는 링크 필드값이 아니라 인접 노드 자체인 `removed->rlink`, `removed->llink`를 대입해야 한다.

오답 포인트

llink와 rlink의 이름만 따라 같은 방향 포인터를 고치거나, 포인터 필드의 값과 실제 인접 노드를 혼동하면 ②부터 ④까지의 오답으로 흐른다. 특히 ④는 `right->llink`와 `left->rlink`가 삭제 전에는 removed라는 기존 값을 가지고 있다는 점을 놓치게 만든다.

다시 나오면 이렇게 판단

삭제 대상 노드를 중심으로 수정할 곳은 두 군데뿐이다. 왼쪽 이웃의 오른쪽 포인터와 오른쪽 이웃의 왼쪽 포인터를 본 뒤, 코드가 `left->rlink = right`, `right->llink = left` 형태인지 확인하면 된다.

암기 포인트

이중 연결 리스트의 중간 노드 삭제는 `left.rlink = right`, `right.llink = left`로 요약된다.

NoteBase

문 3. 연결 리스트에 여러 개의 노드가 존재할 때, 다음 노드를 가리키는 포인터 값이 NULL인 경우가 전혀 없는 성질을 가진 연결 리스트는?

① 단순 연결 리스트(singly linked list)

② 이중 연결 리스트(doubly linked list)

③ 원형 연결 리스트(circular linked list)

④ 헤드노드를 가진 단순 연결 리스트(singly linked list with head node)

정답 3번

출제 포인트

이 문제는 연결 리스트의 마지막 노드가 다음 노드를 어떻게 가리키는지에 따라 선형 리스트와 원형 리스트를 구별할 수 있는지를 묻는다.

핵심 개념

단순 연결 리스트는 각 노드가 다음 노드 하나만 가리키는 선형 구조이며 마지막 노드의 next는 보통 NULL이다.
이중 연결 리스트는 각 노드가 이전 노드와 다음 노드를 모두 가리키는 선형 구조이며 마지막 노드의 next는 보통 NULL이다.
원형 연결 리스트는 마지막 노드가 처음 노드로 다시 연결되는 구조이므로 next 포인터가 NULL이 되지 않는다.
헤드노드를 가진 단순 연결 리스트는 맨 앞에 관리용 노드가 추가된 단순 연결 리스트일 뿐이며, 마지막 노드의 next가 NULL인 선형 구조라는 점은 그대로이다.

선지별 해설

1번 X: 단순 연결 리스트는 선형 구조이므로 마지막 노드의 next가 NULL이 된다. 틀린 표현은 단순 연결 리스트가 NULL 없는 구조라는 점이다. 올바르게는 마지막 노드의 다음 포인터가 NULL인 구조이다.
2번 X: 이중 연결 리스트도 마지막 노드의 next는 NULL이 되는 경우가 일반적이므로 조건에 맞지 않는다. 틀린 표현은 이중 연결 리스트가 다음 포인터 기준으로 NULL이 전혀 없다는 점이다. 올바르게는 이전 포인터와 다음 포인터를 모두 가지지만 끝 노드의 next는 NULL인 선형 구조이다.
3번 O: 원형 연결 리스트는 마지막 노드의 next가 첫 노드를 가리키므로 여러 노드가 있어도 다음 노드를 가리키는 포인터 값이 NULL인 경우가 없다. 이 성질이 원형 구조의 핵심이다.
4번 X: 헤드노드를 가진 단순 연결 리스트도 본질은 선형 단순 연결 리스트이므로 마지막 실제 데이터 노드의 next는 NULL이다. 틀린 표현은 헤드노드가 있으면 NULL이 사라진다는 점이다. 올바르게는 시작 처리를 쉽게 할 뿐 끝 노드의 next는 여전히 NULL이다.

NoteBase

오답 포인트

헤드노드가 있으면 리스트 전체가 닫힌 구조가 된다고 착각하면 4번으로 흔들린다. 헤드노드는 시작 처리를 편하게 하는 보조 노드일 뿐이고, 마지막 노드가 처음으로 되돌아가는 구조는 원형 연결 리스트에서만 성립한다.

다시 나오면 이렇게 판단

발문에서 마지막 노드의 다음 포인터가 NULL인지 아닌지를 먼저 본다. 마지막 노드가 첫 노드나 헤드노드로 다시 연결되면 원형 리스트이고, 끝에서 연결이 끊기면 선형 리스트이다.

암기 포인트

선형 리스트의 끝은 NULL이고, 원형 리스트의 끝은 다시 처음으로 연결된다.

NoteBase

문 4. 수식 ((A+B)-(C/D))을 후위 표기법(postfix notation)으로 가장 바르게 변환한 것은?

① ABCD+-/

② -+AB/CD

③ AB+-CD/

④ AB+CD/-

정답 4번

출제 포인트

중위식 $((A + B) - (C/D))$를 후위식으로 바꿀 때 괄호 안의 부분식부터 처리하고, 각 부분식의 연산자를 피연산자 뒤에 붙이는 순서를 묻는 문제이다.

올바른 순서

1. $(A + B)$를 먼저 후위식으로 바꾸면 `AB+`가 된다.
2. (C/D)를 후위식으로 바꾸면 `CD/`가 된다.
3. 두 부분식을 왼쪽부터 이어 붙이면 `AB+CD/`가 된다.
4. 마지막 바깥 연산자 `-`를 맨 뒤에 붙이면 `AB+CD/-`가 된다.

왜 이 순서: 후위 표기법은 각 부분식이 먼저 완성된 뒤 그 결과에 적용되는 연산자를 뒤에 붙이므로, 바깥의 `-`는 두 부분식 `AB+`와 `CD/`가 만들어진 다음에만 올 수 있다.

선지별 해설

① X: `ABCD+-/`는 `+`가 `C`와 `D`에 적용되고, 그 뒤 `-`, `/`가 이어지는 형태가 되어 원래 식의 구조와 다르다. 틀린 표현은 부분식 경계를 무시하고 연산자를 뒤섞은 점이다. 올바르게는 `(A+B)`를 `AB+`, `(C/D)`를 `CD/`로 만든 뒤 `-`를 마지막에 붙인 `AB+CD/-`가 된다.

② X: `-+AB/CD`는 연산자가 피연산자 앞에 오는 전위 표기법 형태이므로 후위 표기법이 아니다. 틀린 표현은 `-`, `+`가 맨 앞에 배치된 점이다. 올바르게는 모든 연산자가 해당 피연산자 뒤에 오는 `AB+CD/-`가 된다.

③ X: `AB+-CD/`는 `AB+`까지는 맞지만 그 다음 `-`가 너무 일찍 나와 오른쪽 부분식 `CD/`가 완성되기 전에 바깥 연산을 시도한다. 틀린 표현은 바깥 연산자 `-`의 선행 배치이다. 올바르게는 `CD/`를 만든 다음 마지막에 `-`를 붙여 `AB+CD/-`가 된다.

④ O: `AB+CD/-`는 `(A+B)`를 `AB+`, `(C/D)`를 `CD/`로 바꾼 뒤 두 결과에 대해 마지막으로 `-`를 적용한 형태이므로 원래 식 $((A + B) - (C/D))$와 정확히 대응한다.

오답 포인트

후위 표기법이라는 말은 알지만 부분식 단위를 끊지 않고 기호를 순서대로만 옮기면 ①이나 ③처럼 바깥 연산자 `-`를 너무 빨리 배치하기 쉽다. ②는 후위와 전위를 혼동하는 전형적 함정이다.

NoteBase

다시 나오면 이렇게 판단

괄호가 있는 식은 가장 안쪽 부분식부터 각각 후위식으로 완성한 뒤, 마지막에 그 두 결과를 묶는 바깥 연산자를 맨 뒤에 붙인다고 판단하면 된다.

암기 포인트

후위 표기법은 부분식 완성 후 연산자 후치, 바깥 연산자는 항상 가장 나중이다.

NoteBase

문 5. 이진 탐색 트리(binary search tree)에 대한 설명으로 가장 옳지 않은 것은?

① 모든 노드는 2개 이내의 자식노드(child node)를 가질 수 있다.

② 왼쪽 자식노드(child node) 없이 오른쪽 자식노드만 있는 노드는 허용되지 않는다.

③ 왼쪽 서브트리(subtree)에 있는 노드들의 키 값들은 루트노드(root node)의 키 값보다 작다.

④ 루트노드(root node)의 경우, 반드시 2개의 자식노드(child node)를 모두 가질 필요는 없다.

정답 2번

출제 포인트

이 문제는 이진 탐색 트리의 핵심 조건이 키의 대소 관계인지, 트리의 모양 제약인지 구별할 수 있는지를 묻는다.

핵심 개념

이진 트리는 각 노드가 최대 2개의 자식을 갖는 트리이다.
이진 탐색 트리는 이진 트리 중에서 왼쪽 서브트리의 키 < 부모 키 < 오른쪽 서브트리의 키를 만족하는 트리이다.
핵심은 키의 정렬 규칙이며, 자식이 반드시 둘이어야 하거나 왼쪽부터 채워져야 한다는 모양 조건은 아니다.

선지별 해설

1번 O: 이진 탐색 트리는 이진 트리의 한 종류이므로 각 노드는 자식을 최대 2개까지 가질 수 있다.
2번 X: 틀린 부분은 '허용되지 않는다'이다. 이진 탐색 트리는 키의 대소 관계만 만족하면 되므로 왼쪽 자식 없이 오른쪽 자식만 있는 노드도 있을 수 있다. 올바르게는 '허용될 수 있다'이다.
3번 O: 왼쪽 서브트리에 있는 노드들의 키 값이 루트 노드의 키 값보다 작아야 한다는 것은 이진 탐색 트리의 기본 성질이다.
4번 O: 이진 탐색 트리는 완전 이진 트리나 정이진트리처럼 모양을 강제하지 않으므로 루트 노드가 반드시 두 자식을 모두 가질 필요는 없다.

오답 포인트

이진 탐색 트리를 완전 이진 트리나 정이진트리와 혼동하면 2번 같은 선지에 속기 쉽다. 이진 탐색 트리는 '정렬 규칙'을 정하는 구조이고, 자식 배치나 개수를 강제하는 구조가 아니다.

다시 나오면 이렇게 판단

트리 문제가 나오면 먼저 조건이 키의 대소 관계인지, 모양 제약인지 구분한다. 왼쪽 < 부모 < 오른쪽처럼 순서를 말하면 이진 탐색 트리의 조건이고, 오른쪽 자식만 불가 또는 자식이 반드시 둘이어야 한다는 말은 다른 형태의 이진 트리 조건이다.

NoteBase

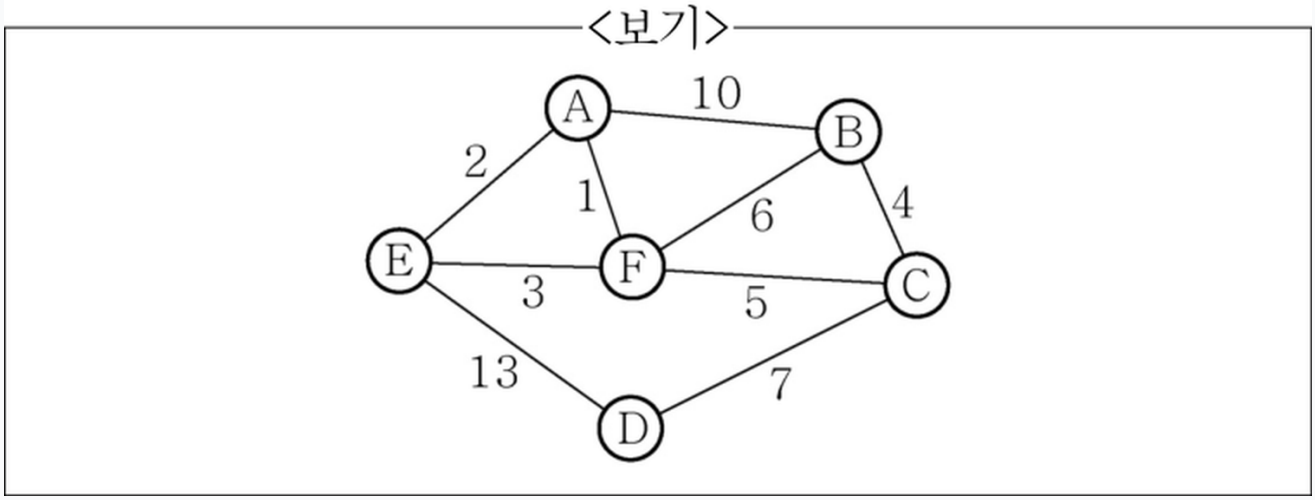
암기 포인트

BST는 키 순서 규칙이고, 완전 이진 트리·정이진트리는 모양 규칙이다.

NoteBase

문 6. <보기>의 그래프에서 크루스칼(Kruskal) 알고리즘을 사용하여 최소신장트리(minimum spanning tree)를 만들 때, 세 번째 삽입하게 되는 선분(edge)은?

<보기>



① (A, E)

② (E, F)

③ (B, C)

④ (C, F)

정답 3번

출제 포인트

이 문제는 그래프의 간선을 가중치 오름차순으로 보되, 사이클이 생기는 간선은 제외하는 크루스칼 알고리즘의 선택 순서를 판단하는 문제이다.

NoteBase

선지별 해설

① X
(A, E)의 가중치는 2이므로 (A, F)의 가중치 1 다음에 바로 채택된다. 따라서 (A, E)는 세 번째가 아니라 두 번째로 삽입된다.
② X
(E, F)의 가중치는 3이지만, 이미 (A, F)와 (A, E)가 선택된 뒤에는 E-A-F가 연결되어 있으므로 (E, F)를 넣으면 사이클이 생긴다. 따라서 세 번째로 삽입되는 간선이 아니라 제외되는 간선이다.
③ O
(B, C)의 가중치는 4이고, 앞에서 선택된 (A, F), (A, E)와는 다른 연결요소를 이루므로 사이클을 만들지 않는다. 따라서 세 번째로 삽입되는 간선은 (B, C)이다.
④ X
(C, F)의 가중치는 5이고, (B, C)가 세 번째로 채택된 다음에야 A-E-F 집합과 B-C 집합을 연결하는 간선으로 선택된다. 따라서 (C, F)는 세 번째가 아니라 네 번째로 삽입된다.

오답 포인트

가중치가 세 번째로 작은 간선이 항상 세 번째로 선택된다고 보면 ②를 고르기 쉽다. 크루스칼 알고리즘은 가중치 순서대로 검토하되, 이미 연결된 정점들 사이에 사이클을 만드는 간선은 건너뛴다는 점이 함정이다.

다시 나오면 이렇게 판단

간선을 가중치 오름차순으로 정렬한 뒤, 각 간선마다 현재까지 선택된 간선들과 함께 사이클이 생기는지만 확인하면 된다. 선택 순서는 가중치 순서와 같지 않을 수 있고, 사이클을 만드는 간선이 끼면 그다음 간선이 실제 채택 순서로 올라온다.

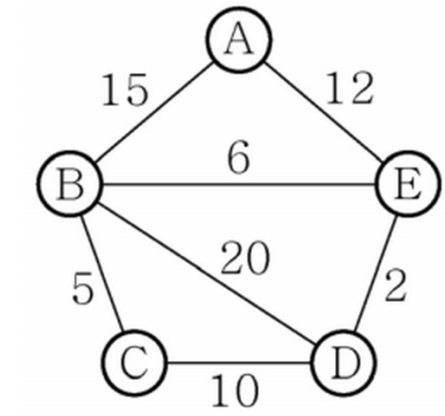
암기 포인트

크루스칼은 작은 가중치 우선이 아니라 작은 가중치 중 사이클이 없는 간선 우선이다.

NoteBase

문 7. <보기>의 그래프를 정점 D에서 시작하여 가중치가 작은 것부터 큰 것 순서대로 깊이 우선 탐색(depth first search)을 할 때 방문하는 순서대로 정점들을 가장 바르게 나열한 것은?

<보기>



① D → E → B → C → A

② D → E → C → B → A

③ D → B → C → E → A

④ D → E → A → C → B

정답 1번

출제 포인트

이 문제는 가중치가 작은 간선부터 선택하는 깊이 우선 탐색에서 현재 정점의 미방문 인접 정점을 작은 가중치 순으로 내려가고, 더 갈 곳이 없을 때 되돌아오는 방문 순서를 묻는 문제이다.

올바른 순서

1. D -> E: D의 인접 정점은 E(2), C(10), B(20)이므로 E를 먼저 방문한다.
2. E -> B: E의 미방문 인접 정점은 B(6), A(12)이므로 B를 먼저 방문한다.
3. B -> C: B의 미방문 인접 정점은 C(5), A(15)이고 C가 더 작으므로 C를 먼저 방문한다.
4. C에서는 더 갈 미방문 정점이 없으므로 B로 되돌아간 뒤 A를 방문한다.

왜 이 순서인가: DFS는 한 경로를 끝까지 따라간 뒤 막히면 되돌아오는 방식이므로, E에서 A가 보여도 먼저 B와 C를 처리한 뒤 A로 간다.

따라서 방문 순서는 D -> E -> B -> C -> A이다.

NoteBase

선지별 해설

① O: D에서 E, E에서 B, B에서 C를 차례로 선택하고 C에서 막히면 되돌아가 A를 방문하므로 D -> E -> B -> C -> A가 맞다.
② X: 틀린 부분은 E 다음에 C를 방문한 점이다. C는 E의 인접 정점이 아니라 B에 도달한 뒤 B의 최소 미방문 인접 정점으로 방문된다. 올바르게는 D -> E -> B -> C -> A이다.
③ X: 틀린 부분은 D 다음에 B를 방문한 점이다. D의 인접 간선 중 최소 가중치는 D-E(2)이므로 B(20)보다 E를 먼저 방문한다. 올바르게는 D -> E -> B -> C -> A이다.
④ X: 틀린 부분은 E 다음에 A를 방문한 점이다. E의 미방문 인접 정점은 B(6), A(12)이고 작은 값은 B이므로 A를 먼저 방문할 수 없다. 올바르게는 D -> E -> B -> C -> A이다.

오답 포인트

가중치가 작은 것부터라는 조건을 현재 정점 기준이 아니라 그래프 전체의 전역 최소 선택으로 오해하면 ③ 같은 오답이 나온다. 또한 DFS의 핵심인 되돌아감 규칙을 놓치고 E에서 바로 A로 가면 ④를 고르기 쉽고, E와 C의 직접 연결이 없다는 점을 놓치면 ②를 고르기 쉽다.

다시 나오면 이렇게 판단

시작 정점에서 현재 정점의 미방문 인접 정점만 가중치 오름차순으로 고르고, 더 내려갈 곳이 없을 때만 직전 정점으로 되돌아간다고 판단하면 된다. 전역 최소 간선을 뽑는 문제가 아니라 현재 위치 기준의 DFS라는 점을 먼저 고정하면 된다.

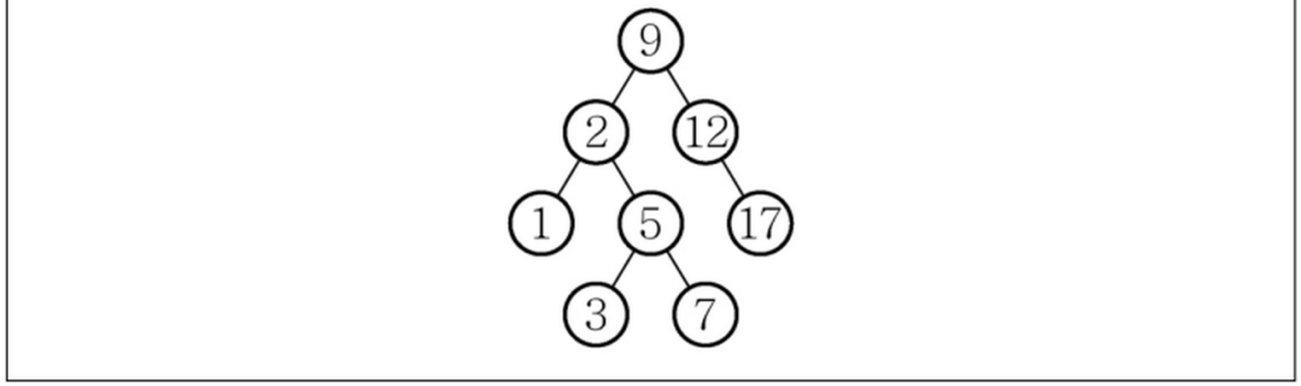
암기 포인트

DFS는 현재 정점에서 가장 작은 간선을 따라 끝까지 내려가고, 막히면 되돌아온다고 기억하면 된다.

NoteBase

문 8. <보기>의 AVL 트리에 4를 삽입할 경우 균형이 깨진 트리를 균형 트리로 다시 만들기 위해 필요한 회전은?

<보기>



① LL 회전

② LR 회전

③ RR 회전

④ RL 회전

정답 4번

출제 포인트

이 문제는 AVL 트리에서 삽입 경로를 따라 최초로 균형이 깨지는 노드를 찾고, 그 불균형이 LL·LR·RR·RL 중 어느 회전에 해당하는지를 판단하는 문제이다.

선지별 해설

1번(X): 4는 9→2→5→3 경로로 내려가 3의 오른쪽 자식으로 삽입된다. 이때 균형이 깨지는 노드는 2이고, 2의 오른쪽 자식 5의 왼쪽 부분트리에서 삽입이 일어났으므로 LL 회전이 아니다. 틀린 표현은 'LL 회전'이다. 올바르게는 'RL 회전'이다.

2번(X): LR 회전은 불균형 노드의 왼쪽 자식의 오른쪽 부분트리에 삽입될 때 발생한다. 이 문제는 2의 오른쪽 자식 5의 왼쪽 부분트리 쪽으로 삽입된 경우이므로 LR 회전이 아니다. 틀린 표현은 'LR 회전'이다. 올바르게는 'RL 회전'이다.

3번(X): 2는 삽입 후 오른쪽으로 무거워지기는 하지만, 새 노드 4는 5의 오른쪽이 아니라 5의 왼쪽 부분트리인 3의 오른쪽에 들어간다. 따라서 오른쪽-오른쪽형인 RR 회전이 아니다. 틀린 표현은 'RR 회전'이다. 올바르게는 'RL 회전'이다.

4번(O): 4를 삽입하면 최초 불균형 노드는 2이고, 불균형 형태는 오른쪽 자식 5의 왼쪽 부분트리에서 발생한 오른쪽-왼쪽형이다. 따라서 5를 기준으로 우회전한 뒤 2를 기준으로 좌회전하는 RL 회전이 필요하다.

NoteBase

오답 포인트

삽입된 값 4가 5보다 작고 3보다 크다는 점을 놓치면 RR로 오인하기 쉽다. 회전 종류는 루트 9 기준이 아니라 최초 불균형 노드 2 기준으로 판단해야 한다는 점이 함정이다.

다시 나오면 이렇게 판단

삽입 위치를 끝까지 찾은 뒤 아래에서 위로 올라오며 최초 불균형 노드를 잡으면 된다. 그 다음 불균형 노드 기준으로 새 노드가 왼쪽의 왼쪽인지, 왼쪽의 오른쪽인지, 오른쪽의 오른쪽인지, 오른쪽의 왼쪽인지 확인하면 회전 종류가 정해진다.

암기 포인트

AVL 회전은 최초 불균형 노드 기준으로 LL·LR·RR·RL을 정하며, 오른쪽 자식의 왼쪽에서 무너지면 RL이다.

NoteBase

문 9. <보기>의 프로시저는 재귀호출(recursive call)을 이용해 정의되었다. <보기>의 프로시저에서 호출 f(3)이 최종적으로 돌려주는 정수값은?

<보기>

```
int f(int n) { if (n < 0) then return 1;
               else return ( n + f(n-1));
             }
```

① 1

② 3

③ 6

④ 7

정답 4번

출제 포인트

이 문제는 재귀 함수의 종료 조건과 누적 합 계산을 따라가며 $f(3)$의 반환값을 구하는 문제이다.

조건 정리

주어진 값: $n < 0$이면 1을 반환하고, 그 외에는 $n + f(n-1)$을 반환한다.
구해야 할 값: $f(3)$
주의할 조건: 종료 조건이 $n = 0$이 아니라 $n < 0$이므로 $f(0)$도 재귀식을 적용한다.

사용할 공식

재귀식은 $f(n) = n + f(n-1)$이고, 종료 조건은 $n < 0$일 때 $f(n) = 1$이다.
이 식을 쓰는 이유: 문제에서 함수가 자신의 이전 값 $f(n-1)$에 현재 값 n을 더하도록 정의되어 있기 때문이다.

계산 과정

1. $f(-1) = 1$이다.
2. $f(0) = 0 + f(-1) = 1$이다.
3. $f(1) = 1 + f(0) = 2$이다.
4. $f(2) = 2 + f(1) = 4$이다.
5. $f(3) = 3 + f(2) = 3 + 4 = 7$이다.

NoteBase

선지별 해설

1번 X: 1은 종료 조건에 걸리는 $f(-1)$이나 그 값을 이용한 $f(0)$의 결과이다. $f(3)$은 재귀를 끝까지 전개하면 7이므로, 1이 아니라 7이다.
2번 X: 3은 중간 단계만 보고 멈춘 값에 가깝다. $f(3)$은 $3 + f(2)$이므로 뒤의 $f(2) = 4$를 더해야 하며, 올바른 값은 3이 아니라 7이다.
3번 X: 6은 $3 + 2 + 1 + 0$만 더하고 종료 조건의 반환값 1을 빠뜨렸을 때 나오는 값이다. 이 함수는 마지막에 $f(-1) = 1$까지 포함하므로, 6이 아니라 7이다.
4번 O: $f(3) = 3 + f(2) = 3 + 4 = 7$이므로 맞는 선택지이다.

오답 포인트

종료 조건을 $n = 0$으로 착각하거나 마지막 반환값 1을 누락하면 6을 고르기 쉽다. 또 $f(3) = 3 + f(2)$에서 $f(2)$를 끝까지 계산하지 않고 멈추면 3 같은 중간값에 흔들리기 쉽다.

다시 나오면 이렇게 판단

재귀 함수 값 문제는 종료 조건을 먼저 적고, 가장 작은 입력부터 차례대로 값을 쌓아 올라가면 된다. 특히 종료 조건이 $n < 0$, $n = 0$, $n \le 1$ 중 무엇인지 먼저 확인하면 오답을 줄일 수 있다.

암기 포인트

재귀값 계산은 종료 조건 확인 후 바닥값부터 역순 누적이라는 흐름으로 정리된다.

NoteBase

문 10. 인접 행렬 A^-1=pmatrix0 & 4 & ∞ & 9 \ 4 & 0 & 8 & 11 \ ∞ & 8 & 0 & 2 \ 9 & 11 & 2 & 0pmatrix이 주어진 그래프에서 Floyd의 최단 경로 알고리즘을 이용하여 $A^{-1} \rightarrow A^0 \rightarrow A^1 \rightarrow A^2 \rightarrow A^3$순으로 행렬을 계산하면 모든 정점 간의 최단 경로를 구할 수 있다. 이 때, 행렬의 특정 원소의 값으로 가장 옳은 것은?

① $A^1[0][2] = 11$

② $A^2[1][3] = 11$

③ $A^3[0][2] = 11$

④ $A^1[1][3] = 10$

정답 3번

출제 포인트

Floyd 알고리즘에서 A^k가 의미하는 중간 정점 허용 범위를 이해하고, 갱신식으로 특정 원소가 언제 줄어드는지를 계산하는 문제이다.

조건 정리

주어진 것은 초기 거리 행렬 A^{-1}이다.
A^0는 정점 0까지, A^1은 정점 1까지, A^2는 정점 2까지, A^3은 정점 3까지를 중간 정점으로 허용한 최단거리 행렬이다.
확인해야 할 핵심 값은 $A^1[0][2]$, $A^1[1][3]$, $A^2[1][3]$, $A^3[0][2]$이다.

사용할 공식

$A^k[i][j] = \min(A^{k-1}[i][j], A^{k-1}[i][k] + A^{k-1}[k][j])$
이 식은 정점 k를 새 중간 정점으로 사용할 때 기존 거리와 $i \rightarrow k \rightarrow j$ 경로의 거리를 비교하는 Floyd 알고리즘의 기본 갱신식이다.

계산 과정

1. A^0에서는 정점 0을 거쳐도 더 짧아지는 값이 없어 $A^0 = A^{-1}$이다.
2. $A^1[0][2] = \min(\infty, A^0[0][1] + A^0[1][2]) = \min(\infty, 4 + 8) = 12$이다.
3. $A^1[1][3] = \min(11, A^0[1][1] + A^0[1][3]) = \min(11, 0 + 11) = 11$이다.
4. $A^2[1][3] = \min(11, A^1[1][2] + A^1[2][3]) = \min(11, 8 + 2) = 10$이다.
5. $A^3[0][2] = \min(12, A^2[0][3] + A^2[3][2]) = \min(12, 9 + 2) = 11$이다.

NoteBase

선지별 해설

① X: $A^1[0][2]$는 정점 1을 거치는 경로 $0 \to 1 \to 2$가 처음 열리므로 12이다. 틀린 표현은 "11"이고, 올바르게는 $A^1[0][2] = 12$이다.
② X: $A^2[1][3]$는 정점 2를 중간에 허용하면 $1 \to 2 \to 3$의 길이 10으로 갱신된다. 틀린 표현은 "11"이고, 올바르게는 $A^2[1][3] = 10$이다.
③ O: $A^3[0][2]$는 정점 3을 거치는 경로 $0 \to 3 \to 2$의 길이 11이 기존 12보다 짧으므로 11이 된다.
④ X: $A^1[1][3]$는 정점 0과 1을 허용해도 기존 직접 거리 11보다 짧아지지 않는다. 틀린 표현은 "10"이고, 올바르게는 $A^1[1][3] = 11$이다.

오답 포인트

A^k의 위첨자 k를 반복 횟수로만 보고 중간 정점 허용 범위를 놓치면 오답이 된다. 특히 $A^2[1][3]$가 정점 2를 거치며 10으로 줄어든다는 점과, $A^3[0][2]$가 정점 3을 거치며 11로 한 번 더 줄어든다는 점을 순서대로 보지 않으면 ②와 ③을 혼동하게 된다.

다시 나오면 이렇게 판단

Floyd 문제는 A^k가 "정점 0부터 k까지를 중간 정점으로 허용한 행렬"이라는 뜻부터 잡고, 필요한 원소만 갱신식 (기존값, 경유값)으로 단계별 계산하면 된다.

암기 포인트

Floyd의 A^k는 정점 k까지 허용한 최단거리 행렬이며, 값은 기존 거리와 새 경유 거리 중 작은 값으로 갱신된다.

NoteBase

문 11. <보기>의 두 가지 (A, B) 정의에 대한 참과 거짓으로 가장 옳은 것은?

<보기>
A) 이진검색알고리즘은 정렬(sort)된 데이터를 필요로 한다.
B) 이진검색알고리즘의 평균 시간복잡도는 $O(\log_2 N)$이다. (단, N은 데이터의 개수이다.)

① A ↔ 참; B ↔ 참

② A ↔ 거짓; B ↔ 참

③ A ↔ 거짓; B ↔ 거짓

④ A ↔ 참; B ↔ 거짓

정답 1번

출제 포인트

이 문제는 이진검색의 전제 조건과 평균 시간복잡도를 함께 묻는 문제이다.

핵심 개념

A: 이진검색은 가운데 값을 기준으로 탐색 구간을 절반씩 버리는 방법이므로 데이터가 정렬되어 있어야 한다.
B: 이진검색의 평균 시간복잡도는 비교할 때마다 후보 수가 절반으로 줄어들기 때문에 $O(\log_2 N)$이다.
구별 축은 전제조건과 성능이다. A는 정렬 필요 여부를, B는 탐색 횟수 증가 속도를 말한다.

선지별 해설

① O
A는 참이고 B도 참이므로 옳다. 이진검색은 정렬된 상태에서만 중간값 비교로 왼쪽 또는 오른쪽 절반을 버릴 수 있고, 평균 비교 횟수도 로그 수준으로 증가한다.

② X
틀린 부분은 "A↔거짓"이다. A는 정렬된 데이터가 필요하므로 참이다. 올바르게는 "A↔참, B↔참"이다.

③ X
틀린 부분은 "A↔거짓; B↔거짓"이다. A는 정렬이 필요하므로 참이고, B도 평균 시간복잡도가 $O(\log_2 N)$이므로 참이다. 올바르게는 "A↔참; B↔참"이다.

④ X
틀린 부분은 "B↔거짓"이다. 이진검색은 한 번 비교할 때마다 탐색 범위가 절반으로 줄어 평균 시간복잡도가 $O(\log_2 N)$이다. 올바르게는 "A↔참; B↔참"이다.

NoteBase

오답 포인트

정렬 필요 여부를 선형검색과 혼동하거나, 시간복잡도에서 로그와 선형을 혼동하면 ②나 ④로 흔들리기 쉽다. 특히 이진검색이 빠르다는 사실만 기억하고 왜 빠른지 모르면 B를 거짓으로 오판하기 쉽다.

다시 나오면 이렇게 판단

이진검색 문제가 나오면 먼저 정렬이 전제인지 확인하고, 다음으로 비교할 때마다 탐색 후보가 절반으로 줄어드는지 떠올리면 된다. 정렬 필요와 절반 감소가 보이면 시간복잡도는 로그 계열로 판단한다.

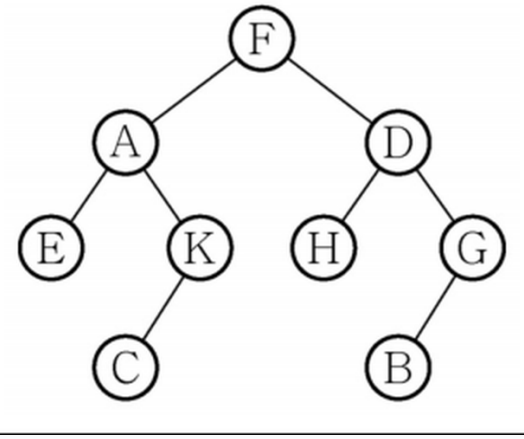
암기 포인트

이진검색은 정렬 후 절반씩 줄인다. 그래서 전제는 정렬, 복잡도는 로그이다.

NoteBase

문 12. <보기>의 이진트리(binary tree)를 후위순회(postorder traversing)한 결과는? (단, 루트노드(root node)는 F이다.)

<보기>



① F-A-E-K-C-D-B-H-G

② E-A-C-K-F-H-D-B-G

③ F-A-E-K-C-D-H-G-B

④ E-C-K-A-H-B-G-D-F

정답 4번

출제 포인트

이 문제는 이진트리의 후위순회가 왼쪽 서브트리, 오른쪽 서브트리, 루트 순서로 진행된다는 규칙을 실제 트리 구조에 적용하는 문제이다.

올바른 순서

1. A의 왼쪽 자식 E를 먼저 방문한다.
2. A의 오른쪽 서브트리 K에서는 왼쪽 자식 C를 방문한 뒤 K를 방문한다.
3. 왼쪽 서브트리의 루트 A를 마지막에 방문하여 왼쪽 전체는 E-C-K-A가 된다.
4. 오른쪽 서브트리 D에서는 왼쪽 자식 H를 먼저 방문하고, 오른쪽 서브트리 G에서는 B를 방문한 뒤 G를 방문하여 H-B-G-D가 된다.
5. 양쪽 서브트리가 모두 끝난 뒤 루트 F를 마지막에 방문하므로 전체 후위순회 결과는 E-C-K-A-H-B-G-D-F가 된다.

왜 이 순서: 후위순회는 부모를 반드시 자식들 뒤에 방문하므로 A는 E와 K 뒤에, K는 C 뒤에, D는 H와 G 뒤에, G는 B 뒤에, F는 모든 노드 뒤에 와야 한다.

NoteBase

선지별 해설

① X
- 틀린 이유: 맨 앞의 F는 루트인데, 후위순회에서는 루트가 항상 마지막에 방문된다. 또한 D의 서브트리도 H-B-G-D 순서여야 하는데 제시문은 D-B-H-G처럼 부모와 자식의 순서가 뒤섞여 있다.
- 올바르게는: E-C-K-A-H-B-G-D-F이다.

② X
- 틀린 이유: A가 C와 K보다 먼저 나와 왼쪽 서브트리의 후위순회 규칙을 어긴다. 또 F가 오른쪽 서브트리보다 먼저 나와 루트 후방문 원칙도 깨진다.
- 올바르게는: 왼쪽 서브트리는 E-C-K-A, 오른쪽 서브트리는 H-B-G-D, 마지막 루트는 F이다.

③ X
- 틀린 이유: F가 맨 앞에 있어 루트가 먼저 방문되었고, K와 C의 순서도 K-C가 아니라 C-K여야 한다. 오른쪽의 G와 B도 G-B가 아니라 B-G여야 한다.
- 올바르게는: E-C-K-A-H-B-G-D-F이다.

④ O
- 맞는 이유: 왼쪽 서브트리에서 E-C-K-A, 오른쪽 서브트리에서 H-B-G-D를 얻은 뒤 마지막에 루트 F를 방문한 결과와 정확히 일치한다.

오답 포인트

루트 F를 먼저 적거나, K와 C처럼 부모와 자식의 선후관계를 바꾸는 실수가 잦다. 후위순회는 각 서브트리 안에서도 같은 규칙이 재귀적으로 적용되므로, K 앞에 C가 와야 하고 G 앞에 B가 와야 한다는 점이 함정이다.

다시 나오면 이렇게 판단

후위순회 문제는 큰 트리를 바로 읽지 말고 루트 기준으로 왼쪽 서브트리와 오른쪽 서브트리로 먼저 분리한 뒤, 각 서브트리 안에서 다시 왼쪽-오른쪽-루트 순서를 적용하면 된다. 부모 노드가 자식보다 앞에 나오면 즉시 오답으로 판단할 수 있다.

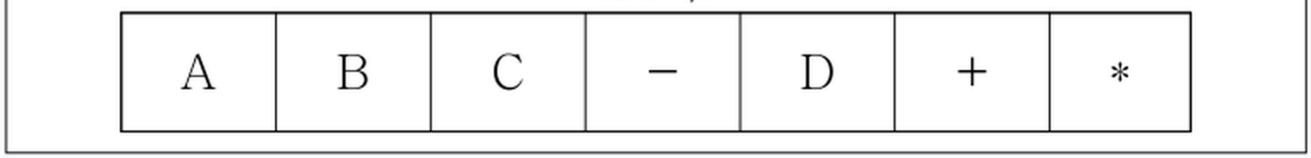
암기 포인트

후위순회는 왼-오-루이며, 부모는 항상 자식들보다 뒤에 나온다.

NoteBase

문 13. <보기>의 후위 표기법(postfix notation)으로 표현된 수식에서 가장 먼저 수행되는 연산은?

<보기>



① A - B

② B + D

③ B - C

④ A * D

정답 3번

출제 포인트

이 문제는 후위 표기식에서 연산자가 나타나는 즉시 직전 두 피연산자에 적용된다는 평가 순서를 묻는 문제이다.

올바른 순서

1. B와 C에 -를 적용한다.
2. 1단계 결과와 D에 +를 적용한다.
3. A와 2단계 결과에 *를 적용한다.

왜 이 순서: 후위 표기식은 연산자가 나오기 전까지 피연산자를 쌓아 두고, 연산자가 나오면 바로 앞의 두 값을 먼저 계산하므로 B-C의 결과가 다음 +의 입력이 되고, 그 결과가 마지막 *의 입력이 된다.

선지별 해설

1번: X. A-B는 수행될 수 없다. A 뒤에는 바로 B가 오지만 그 직후 연산자는 -가 아니라 C이므로 A와 B만으로는 아직 연산이 닫히지 않는다. 틀린 표현은 'A-B가 먼저 수행된다'이고, 올바르게는 'B-C가 먼저 수행된다'이다.

2번: X. B+D는 수행되지 않는다. + 연산자는 D가 나온 뒤 등장하지만, 그 시점의 두 피연산자는 D와 직전에 계산된 B-C의 결과이다. 틀린 표현은 'B+D'이고, 올바르게는 '(B-C)+D'이다.

3번: O. A B C - D + *에서 가장 먼저 만나는 연산자는 -이고, 그 직전 두 피연산자는 B와 C이므로 첫 연산은 B-C이다.

4번: X. A*D는 마지막 단계의 실제 연산 대상이 아니다. *는 맨 끝에서 수행되지만, 그때 결합되는 것은 A와 D가 아니라 A와 (B-C+D)의 결과이다. 틀린 표현은 'A*D'이고, 올바르게는 'A*((B-C)+D)'이다.

NoteBase

오답 포인트

중위표기처럼 왼쪽에서 차례로 계산한다고 보면 1번이나 4번으로 흐르기 쉽다. 후위 표기식은 연산자 위치가 우선순위를 대신하므로, 연산자가 나타났을 때 바로 앞의 두 값이 무엇인지 스택 관점으로 확인해야 한다.

다시 나오면 이렇게 판단

후위 표기식은 왼쪽부터 읽으면서 피연산자는 쌓고 연산자가 나오면 직전 두 값을 꺼내 계산한다고 판단한다. 따라서 '가장 먼저 수행되는 연산'은 가장 먼저 등장하는 연산자가 아니라, 그 연산자 앞에 이미 두 피연산자가 완성되어 있는 첫 연산이다.

암기 포인트

후위 표기식은 피연산자 두 개가 먼저, 연산자 하나가 나중이라는 규칙으로 읽는다.

NoteBase

문 14. <보기>의 프로그램이 실행되면 값이 바뀌는 배열의 요소는?

<보기>

```
int a[5], *p;
p=a;
(*p)++;
```

① a[0]

② a[1]

③ a[2]

④ a[3]

정답 1번

출제 포인트

이 문제는 배열 이름이 첫 번째 원소의 주소로 쓰인다는 점과 포인터 역참조 증가 연산이 어떤 원소를 바꾸는지를 묻는 문제이다.

핵심 개념

배열명 a는 식에서 보통 첫 번째 원소 a[0]의 주소로 취급된다.
p=a; 는 p가 a[0]를 가리키게 만든다.
(*p)++; 는 포인터 p가 가리키는 값 자체를 1 증가시키므로 바뀌는 대상은 a[0]이다.
배열 원소의 초기값은 정해져 있지 않을 수 있지만, 변경되는 위치는 여전히 a[0]이다.

선지별 해설

① O: p=a; 에서 p는 배열의 첫 원소 a[0]를 가리킨다. 따라서 (*p)++; 는 a[0]의 값을 1 증가시킨다.
② X: 틀린 부분은 a[1]이다. p가 한 칸 이동하는 연산은 없으므로 실제로 증가하는 것은 a[1]이 아니라 a[0]이다.
③ X: 틀린 부분은 a[2]이다. 두 칸 뒤 원소를 가리키려면 p=a+2; 또는 *(p+2) 같은 표현이 필요하지만, 제시된 코드에는 그런 이동이 없다.
④ X: 틀린 부분은 a[3]이다. 세 번째 인덱스 원소를 바꾸려면 p가 a[3]를 가리켜야 하지만 현재 p는 처음부터 끝까지 a[0]만 가리킨다.

오답 포인트

배열명과 포인터를 막연히 같은 것으로만 외우면 p가 어디를 가리키는지 놓치기 쉽다. 이 문항의 함정은 (*p)++를 p++와 혼동하여 포인터가 다음 칸으로 이동한다고 착각하는 데 있다.

NoteBase

다시 나오면 이렇게 판단

먼저 대입문으로 포인터가 어느 주소를 가리키는지 확인한 뒤, 괄호 안의 연산 대상이 포인터 자체인지 가리키는 값인지 구분하면 된다. (*p)++는 값 증가이고, p++는 주소 이동이다.

암기 포인트

a는 시작 주소, *p는 그 주소의 값이다.

NoteBase

문 15. <보기>의 상황과 가장 유사한 자료구조는?

<보기>
슈퍼마켓의 계산대에 줄지어 서 있는 손님 대기선과 유사한 자료구조로서 대기선에 먼저 줄을 선 손님 순으로 한 명씩 계산대 서비스를 받고 나가는 상황

① 스택 링크드 리스트(Stack Linked List)
② 큐 링크드 리스트(Queue Linked List)
③ 위의 두 자료구조 모두 해당 함
④ 위의 두 자료구조 모두 해당하지 않음

정답 2번

출제 포인트

이 문제는 실생활 사례의 처리 순서를 보고 선입선출 구조인 큐와 후입선출 구조인 스택을 구별할 수 있는지를 묻는다.

사례 정리

핵심 사실은 먼저 줄을 선 손님부터 한 명씩 계산을 받고 나간다는 점이다.
판단에 중요한 조건은 대기 순서가 서비스 순서와 같다는 점이다.

핵심 개념

큐는 먼저 들어온 데이터가 먼저 나가는 선입선출(FIFO) 구조이다.
스택은 나중에 들어온 데이터가 먼저 나가는 후입선출(LIFO) 구조이다.
이 사례의 판단 기준은 먼저 온 손님이 먼저 처리되는지 여부이다.

적용 기준

계산대 대기선은 줄의 앞에서 먼저 선 손님이 먼저 빠져나가므로 큐의 삭제 순서와 같다. 스택처럼 가장 나중에 들어온 항목이 먼저 빠지는 구조와는 다르다.

NoteBase

선지별 해설

1번: X. 스택 링크드 리스트는 후입선출 구조라서 나중에 들어온 손님이 먼저 계산받는 상황에 가깝다. 틀린 표현은 계산대 대기선을 스택에 대응시킨 부분이며, 올바르게는 나중에 들어온 항목이 먼저 처리될 때 스택에 해당한다.
2번: O. 큐 링크드 리스트는 선입선출 구조이므로 먼저 줄을 선 손님이 먼저 서비스를 받는 계산대 대기선과 가장 유사하다.
3번: X. 큐와 스택이 모두 해당하는 것은 아니다. 틀린 표현은 두 자료구조 모두 해당한다는 부분이며, 올바르게는 큐 링크드 리스트만 해당한다.
4번: X. 위 상황은 큐에 분명히 해당하므로 두 자료구조 모두 해당하지 않는다는 판단은 틀리다. 올바르게는 먼저 들어온 손님이 먼저 나가므로 큐 구조에 해당한다.

오답 포인트

스택과 큐를 모두 줄 세우는 구조로 막연히 기억하면 3번이나 4번으로 흔들릴 수 있다. 이 문제의 함정은 저장 방식이 아니라 꺼내는 순서에 있다.

다시 나오면 이렇게 판단

실생활 사례형 자료구조 문제는 먼저 들어간 것이 먼저 나오는지, 나중에 들어간 것이 먼저 나오는지만 확인하면 된다. 전자는 큐, 후자는 스택이다.

암기 포인트

줄 서기는 큐, 접시 쌓기는 스택이다.

NoteBase

문 16. <보기>의 my_recursion(7)을 호출하여 얻은 결과값은?

<보기>

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

① 8

② 13

③ 21

④ 34

정답 3번

출제 포인트

이 문제는 종료 조건이 $n <= 1$인 재귀식 $f(n) = f(n-2) + f(n-1)$을 작은 값부터 차례대로 계산해 $f(7)$을 구할 수 있는지를 묻는다.

조건 정리

주어진 값: $n <= 1$이면 값은 1이다.
구해야 할 값: $f(7)$이다.
주의할 조건: 종료 조건이 $n <= 1$이므로 $f(0) = 1$, $f(1) = 1$에서 시작한다.

사용할 공식

$f(n) = f(n-2) + f(n-1) \quad (n > 1),\ f(0) = f(1) = 1$
이 문제는 현재 값이 바로 앞 두 값의 합으로 정해지는 피보나치형 재귀이므로, 작은 n부터 순서대로 계산하면 된다.

계산 과정

1. $f(2) = f(0) + f(1) = 1 + 1 = 2$이다.
2. $f(3) = f(1) + f(2) = 1 + 2 = 3$, $f(4) = f(2) + f(3) = 2 + 3 = 5$이다.
3. $f(5) = f(3) + f(4) = 3 + 5 = 8$, $f(6) = f(4) + f(5) = 5 + 8 = 13$이다.
4. $f(7) = f(5) + f(6) = 8 + 13 = 21$이다.

NoteBase

선지별 해설

1번: X. 8은 $f(5)$ 값이다. 틀린 표현은 결과값을 8로 본 점이며, 올바르게는 $f(7) = 21$이다.
2번: X. 13은 $f(6)$ 값이다. 틀린 표현은 결과값을 13으로 본 점이며, 올바르게는 $f(7) = 21$이다.
3번: O. $f(7) = f(5) + f(6) = 8 + 13 = 21$이므로 맞는 선택지이다.
4번: X. 34는 $f(8) = f(6) + f(7) = 13 + 21$일 때 나오는 값이다. 틀린 표현은 결과값을 34로 본 점이며, 올바르게는 $f(7) = 21$이다.

오답 포인트

종료 조건 $n <= 1$을 놓치면 시작값을 잘못 잡기 쉽고, 계산 도중 나온 8이나 13을 최종값으로 착각하기 쉽다. 또 21 다음 값인 34를 한 단계 더 진행한 값으로 잘못 고르는 경우도 많다.

다시 나오면 이렇게 판단

재귀식 값 문제는 종료 조건을 먼저 숫자로 확정한 뒤, 작은 n부터 표처럼 채워 목표 값까지 올라가면 된다.
중간값 $f(5)$, $f(6)$을 적어 두면 $f(7)$은 바로 확인된다.

암기 포인트

$n <= 1$이면 둘 다 1로 시작하고, 그다음부터는 앞의 두 값을 더해 한 칸씩 올리는 구조이다.

NoteBase

문 17. 최대 히프(max heap) 구조의 특성으로 가장 옳지 않은 것은?

① 부모노드(parent node)의 키 값이 자식노드(child node)의 키 값 보다 항상 크거나 같다.

② 왼쪽 서브트리(subtree)에 있는 노드들의 키 값들은 오른쪽 서브트리(subtree)에 있는 노드들의 키 값들 보다 항상 작거나 같다.

③ 루트노드(root node)의 키 값이 최댓값이다.

④ 완전 이진 트리(complete binary tree)의 특성을 갖는다.

정답 2번

출제 포인트

최대 히프의 핵심 성질인 부모-자식 간 대소관계와 완전 이진 트리 조건을, 이진 탐색 트리의 좌우 서브트리 정렬 규칙과 구별하는 문제이다.

핵심 개념

최대 히프: 완전 이진 트리 형태를 유지하면서 각 부모노드의 키 값이 자식노드의 키 값보다 크거나 같은 구조이다. 키워드는 부모 >= 자식이다.
이진 탐색 트리: 각 노드를 기준으로 왼쪽 서브트리의 키 값은 작고 오른쪽 서브트리의 키 값은 큰 구조이다. 키워드는 왼쪽 < 루트 < 오른쪽이다.
구별 기준: 히프는 상하 관계를 보지만, 이진 탐색 트리는 좌우 정렬 관계를 본다.

NoteBase

선지별 해설

① O
부모노드의 키 값이 자식노드의 키 값보다 항상 크거나 같다는 설명은 최대 히프의 기본 성질 그대로이다.

② X
틀린 부분은 "왼쪽 서브트리에 있는 노드들의 키 값들은 오른쪽 서브트리에 있는 노드들의 키 값들보다 항상 작거나 같다"이다.
최대 히프는 부모와 자식 사이의 대소관계만 보장할 뿐, 왼쪽 서브트리와 오른쪽 서브트리 사이의 전역적인 크기 순서는 보장하지 않는다.
올바르게는 "왼쪽 서브트리와 오른쪽 서브트리 사이에는 항상 성립하는 대소관계가 없다"이다.

③ O
최대 히프에서는 어떤 자식도 자신의 부모보다 클 수 없으므로, 루트노드가 전체 노드 중 최대 키 값을 가진다.

④ O
이진 히프로 구현하는 최대 히프는 삽입과 삭제를 효율적으로 하기 위해 완전 이진 트리 형태를 유지한다.

오답 포인트

2번은 이진 탐색 트리의 성질을 최대 히프의 성질처럼 바꿔 놓은 선지이다. 히프도 트리이므로 좌우 서브트리 크기 비교가 있을 것처럼 착각하면 틀리기 쉽다.

다시 나오면 이렇게 판단

힙 관련 선지는 먼저 부모와 자식의 대소관계만 확인하면 된다. 선지가 왼쪽 서브트리와 오른쪽 서브트리의 정렬 관계를 말하면 히프가 아니라 이진 탐색 트리 성질인지부터 구분하면 된다.

암기 포인트

최대 히프는 부모가 크고, 이진 탐색 트리는 왼쪽이 작다.

NoteBase

문 18. 퀵 정렬 알고리즘(Quick Sort Algorithm)의 평균 시간 복잡도로 가장 옳은 것은? (단, N은 데이터의 개수이다.)

① O(log N)

② O(N · log N)

③ $O(N^2 \cdot \log N)$

④ $O(N^2)$

정답 2번

출제 포인트

이 문제는 퀵 정렬의 평균 시간 복잡도와 최악 시간 복잡도를 구분하는지를 묻는 문제이다.

핵심 개념

퀵 정렬은 피벗을 기준으로 데이터를 분할한 뒤 재귀적으로 정렬하는 분할 정복 알고리즘이다.
평균적으로 분할이 비교적 균형을 이루면 각 단계에서 N개 원소를 한 번씩 처리하고 재귀 깊이가 $\log N$이므로 평균 시간 복잡도는 $O(N \cdot \log N)$이다.
반대로 분할이 매번 한쪽으로 치우치면 재귀 깊이가 N에 가까워져 최악 시간 복잡도는 $O(N^2)$이 된다.

선지별 해설

① X
- $O(\log N)$은 탐색 깊이만 로그로 줄어드는 경우에 가까운 복잡도이다.
- 퀵 정렬은 한 번 분할할 때마다 부분 배열의 원소들을 비교해야 하므로 단계당 비용이 N 수준으로 든다.
- 틀린 부분은 "$O(\log N)$"이다. 올바르게는 평균 시간 복잡도는 $O(N \cdot \log N)$이다.

② O
- 퀵 정렬은 평균적으로 분할이 균형에 가깝게 이루어지면 레벨 수가 $\log N$, 각 레벨 처리량이 N이므로 평균 시간 복잡도는 $O(N \cdot \log N)$이다.

③ X
- $O(N^2 \cdot \log N)$은 퀵 정렬의 대표적인 평균 시간 복잡도가 아니다.
- 퀵 정렬은 레벨마다 전체 원소를 한 번씩 처리하므로 평균은 N과 $\log N$의 곱으로 정리된다.
- 틀린 부분은 "$N^2 \cdot \log N$"이다. 올바르게는 평균 시간 복잡도는 $O(N \cdot \log N)$이다.

④ X
- $O(N^2)$은 퀵 정렬의 최악 시간 복잡도에 해당한다.
- 피벗 선택이 계속 불균형하면 이런 경우가 생기지만, 문제는 평균 시간 복잡도를 묻고 있다.
- 틀린 부분은 "평균 시간 복잡도"와 "$O(N^2)$"의 연결이다. 올바르게는 평균은 $O(N \cdot \log N)$, 최악은 $O(N^2)$이다.

NoteBase

오답 포인트

퀵 정렬에서 많이 헷갈리는 지점은 평균과 최악을 뒤섞는 점이다. ④의 $O(N^2)$은 익숙한 값이라 쉽게 고르기 쉽지만, 그것은 피벗 분할이 계속 치우칠 때의 최악 경우이다. ①처럼 재귀 깊이만 보고 $\log N$으로 판단하거나, ③처럼 N과 $\log N$의 관계를 과장하는 것도 흔한 오답이다.

다시 나오면 이렇게 판단

분할 정복 정렬은 한 단계 처리량과 재귀 깊이를 곱해 판단한다. 퀵 정렬은 한 단계 처리량이 N, 평균 재귀 깊이가 $\log N$이므로 평균은 $O(N \cdot \log N)$이다. 한쪽 분할만 반복되는 최악 경우만 $O(N^2)$로 본다.

암기 포인트

퀵 정렬은 평균 $N\log N$, 최악 N^2이다.

NoteBase

문 19. C언어에서 정수형 2차원 배열 A[2][5]에 대한 표현 중 주소 값이 아닌 것은?

① A

② *A

③ *A[1]

④ *(A+1)

정답 3번

출제 포인트

이 문제는 C 언어의 2차원 배열 식이 평가될 때 배열이 포인터로 decay되는 경우와 실제로 정수값이 되는 경우를 구별하는지를 묻는다.

핵심 개념

A는 전체 2차원 배열 이름이지만 식에서 첫 번째 행을 가리키는 포인터로 decay되어 값은 A[0]의 주소가 된다. *A는 첫 번째 행 배열 A[0]이고 식에서 다시 첫 원소의 주소로 decay되어 A[0][0]의 주소가 된다. A[1]과 *(A+1)은 둘 다 두 번째 행 배열을 뜻하며 식에서 A[1][0]의 주소로 decay된다. 반면 *A[1]은 두 번째 행의 첫 원소 자체이므로 정수값이다.

선지별 해설

1번 X: A는 식에서 첫 번째 행을 가리키는 포인터로 decay되므로 주소 값이다. 틀린 표현은 이 선지를 주소 값이 아닌 것으로 보는 점이다. 올바르게는 A는 A[0]의 주소를 나타낸다.
2번 X: *A는 첫 번째 행 배열 A[0]을 뜻하고 식에서 다시 첫 원소 쪽으로 decay되므로 A[0][0]의 주소 값으로 취급된다. 틀린 표현은 주소 값이 아니라는 판단이다. 올바르게는 *A는 첫 번째 행의 시작 주소를 나타낸다.
3번 O: *A[1]은 *(A[1])로 해석된다. A[1]은 두 번째 행 배열이고 이것이 decay되면 A[1][0]의 주소가 되며, 다시 *를 붙이면 그 주소가 가리키는 실제 원소값 A[1][0]이 된다. 따라서 주소 값이 아니라 정수값이다.
4번 X: *(A+1)은 두 번째 행 배열 자체를 뜻한다. 이 식도 일반 식 문맥에서는 A[1]과 같은 의미로 decay되어 A[1][0]의 주소 값이 된다. 틀린 표현은 주소 값이 아니라는 판단이다. 올바르게는 *(A+1)은 두 번째 행의 시작 주소로 본다.

오답 포인트

*(A+1)과 *A[1]의 결합 방향을 헷갈리면 틀리기 쉽다. *(A+1)은 행 전체를 가리킨 뒤 그 행이 다시 주소로 decay되는 식이고, *A[1]은 두 번째 행의 첫 원소를 꺼낸 값이라서 결과형이 전혀 다르다.

NoteBase

다시 나오면 이렇게 판단

배열 식이 나오면 먼저 괄호와 첨자 결합을 확인한 뒤, 결과가 배열인지 원소인지부터 구분한다. 결과가 배열이면 식 문맥에서 시작 주소로 decay되고, 결과가 원소이면 주소가 아니라 실제 값이 된다.

암기 포인트

배열이면 주소로 decay되고, 원소까지 꺼내면 값이 된다. *A[1]만 원소값이고 A, *A, *(A+1)은 모두 시작 주소 계열이다.

NoteBase

문 20. <보기>의 부울 연산식(Boolean formula)의 값을 1로 만드는 x, y, z의 값들이 아닌 것은? (단, +는 or, ·은 and이다.)

<보기>

$(\overline{x} + y + z) \cdot (x \cdot \overline{y} + \overline{z}) \cdot (x + \overline{y} + \overline{z})$

① x = 0, y = 0, z = 0

② x = 0, y = 1, z = 0

③ x = 1, y = 1, z = 0

④ x = 0, y = 1, z = 1

정답 4번

출제 포인트

이 문제는 부울 연산식에서 보수와 OR, AND 규칙을 적용해 각 대입값에서 식의 최종값이 1인지 0인지 판별하는 문제이다.

조건 정리

주어진 식은 $(\bar{x} + y + z) \cdot (x \cdot \bar{y} + \bar{z}) \cdot (x + \bar{y} + \bar{z})$이다.
+는 OR이므로 항 중 하나라도 1이면 1이 된다.
·는 AND이므로 모든 인수가 1이어야 1이 된다.
보수는 $\bar{0} = 1, \bar{1} = 0$이다.

사용할 공식

값을 대입한 뒤 각 괄호를 순서대로 계산한다.
OR은 하나라도 1이면 1, AND는 모두 1일 때만 1이 되므로 이 규칙만 정확히 적용하면 된다.

계산 과정

식을 F라 두면 $F = (\bar{x} + y + z) \cdot (x \cdot \bar{y} + \bar{z}) \cdot (x + \bar{y} + \bar{z})$이다.
① $x = 0, y = 0, z = 0$이면 $(1 + 0 + 0) \cdot (0 \cdot 1 + 1) \cdot (0 + 1 + 1) = 1 \cdot 1 \cdot 1 = 1$이다.
② $x = 0, y = 1, z = 0$이면 $(1 + 1 + 0) \cdot (0 \cdot 0 + 1) \cdot (0 + 0 + 1) = 1 \cdot 1 \cdot 1 = 1$이다.
③ $x = 1, y = 1, z = 0$이면 $(0 + 1 + 0) \cdot (1 \cdot 0 + 1) \cdot (1 + 0 + 1) = 1 \cdot 1 \cdot 1 = 1$이다.
④ $x = 0, y = 1, z = 1$이면 $(1 + 1 + 1) \cdot (0 \cdot 0 + 0) \cdot (0 + 0 + 0) = 1 \cdot 0 \cdot 0 = 0$이다.

NoteBase

선지별 해설

① O

$x=0, y=0, z=0$을 대입하면 식의 값이 1이 된다. 따라서 '이 값이 식을 1로 만든다'는 판단은 옳다.

② O

$x=0, y=1, z=0$을 대입하면 두 번째 괄호의 $\bar{z}=1$이 되어 전체 식이 1이 된다. 따라서 식을 1로 만드는 값이 맞다.

③ O

$x=1, y=1, z=0$을 대입하면 첫 번째 괄호는 $y=1$ 때문에 1이고, 두 번째와 세 번째 괄호도 각각 1이 되어 전체 식이 1이 된다. 따라서 식을 1로 만드는 값이 맞다.

④ X

$x=0, y=1, z=1$을 대입하면 두 번째 괄호 $x \cdot \bar{y} + \bar{z}$가 0이고 세 번째 괄호 $x + \bar{y} + \bar{z}$도 0이어서 전체 식의 값은 1이 아니라 0이다. 틀린 표현은 '식을 1로 만든다'는 부분이며, 올바르게는 '이 값에서는 식의 값이 0이 된다'이다.

오답 포인트

$z=1$일 때 $\bar{z}=0$이 된다는 점을 놓치면 ④도 1처럼 보이기 쉽다. 특히 두 번째 괄호의 $x \cdot \bar{y}$를 $x \cdot y$처럼 잘못 읽거나, AND에서 0이 하나만 있어도 전체가 0이 된다는 점을 놓치면 오답이 나온다.

다시 나오면 이렇게 판단

부울식 대입 문제는 먼저 보수값을 바꾼 뒤 각 괄호를 OR로 계산하고, 마지막에 괄호들끼리 AND로 묶어 전체값을 판단하면 된다. 중간 괄호 하나라도 0이면 전체 AND 결과는 0이 된다고 보면 된다.

암기 포인트

부울식 대입은 '보수 먼저, OR 다음, AND 마지막' 순서로 보면 계산이 가장 안정적이다.

NoteBase