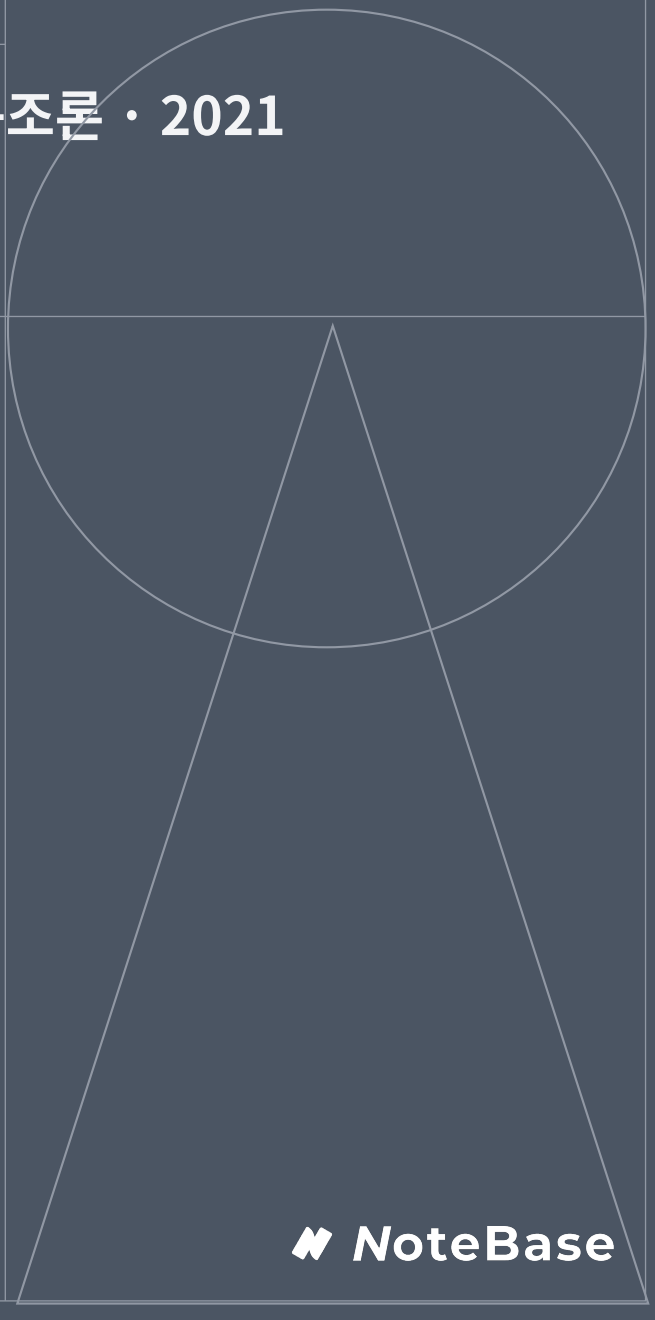
기출OX 해설집

공무원/서울시/7급 · 자료구조론 · 2021



NoteBase

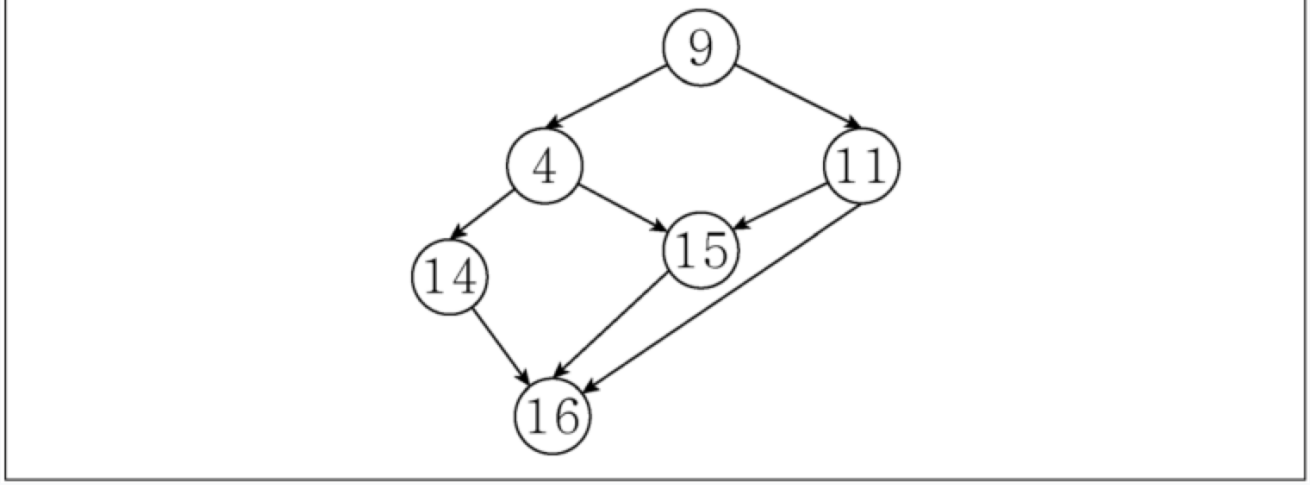
기출OX 해설집

공무원/서울시/7급 · 자료구조론 · 2021

공무원/서울시/7급 · 자료구조론 · 2021

문 1. <보기>의 그래프에 대해 위상 정렬한 결과로 가장 옳지 않은 것은?

<보기>



① 9 - 11 - 15 - 4 - 14 - 16

② 9 - 4 - 14 - 11 - 15 - 16

③ 9 - 11 - 4 - 14 - 15 - 16

④ 9 - 11 - 4 - 15 - 14 - 16

정답 1번

출제 포인트

이 문제는 방향 그래프의 선후행 관계를 읽어 각 정점이 반드시 앞서야 하는 정점을 어기지 않는 위상 정렬 순서를 판단하는 문제이다.

NoteBase

선지별 해설

① X
- 15보다 먼저 나와야 하는 정점은 4와 11이다. 그런데 ①은 11 다음에 15가 나오고 그 뒤에 4가 나오므로 간선 4→15의 선후관계를 위반한다.
- 틀린 표현은 "9-11-15-4-14-16"의 15가 4보다 앞서는 부분이다. 올바르게는 4가 15보다 먼저 나와야 한다.

② O
- 9가 4와 11보다 먼저 나오고, 4가 14보다 먼저 나오며, 11과 4가 모두 15보다 먼저 나온다. 또한 14, 15, 11이 모두 16보다 먼저 나오므로 모든 간선 방향을 만족한다.

③ O
- 9가 먼저 나오고, 11과 4의 상대적 순서는 자유롭다. 이어서 4가 14와 15보다 먼저 나오고, 11도 15보다 먼저 나오며, 마지막에 16이 나와 모든 선후관계를 만족한다.

④ O
- 9가 11과 4보다 먼저 나오고, 4와 11이 모두 15보다 먼저 나온다. 14는 4 뒤에만 있으면 되고 16보다 먼저만 오면 되므로 15 다음에 14가 와도 가능하다.
- 14와 15 사이에는 직접적인 선후 제약이 없으므로 이 둘의 순서는 바뀔 수 있다.

오답 포인트

15와 14의 순서를 고정된 것으로 착각하면 ④를 틀렸다고 보기 쉽다. 이 그래프에서 실제 제약은 4→14, 4→15, 11→15, 14→16, 15→16이므로 14와 15 사이에는 직접 간선이 없다. 반대로 ①은 15 앞에 반드시 있어야 하는 4를 뒤로 보내는 위반을 숨겨 놓은 선지이다.

다시 나오면 이렇게 판단

위상 정렬 문제는 각 정점에 대해 반드시 먼저 나와야 하는 선행 정점만 표시해 두고 순서를 대입하면 된다. 특히 어떤 정점 앞에 와야 하는 부모 정점이 모두 배치되었는지와 마지막 정점이 여러 화살표의 도착점인지부터 확인하면 빠르게 판단할 수 있다.

암기 포인트

위상 정렬은 모든 간선 $u \to v$에 대해 항상 u가 v보다 먼저 나온다.

NoteBase

문 2. <보기>의 C언어 문장을 사용하여 이중 연결 리스트(doubly linked list)에서 p가 가리키는 노드 다음에 new_node가 가리키는 새로운 노드를 삽입하려고 한다. 이를 수행하기 위한 문장 순서로 가장 옳지 않은 것은? (단, 노드에서 이전/다음 노드를 가리키는 포인터는 각각 prev/next이고, 새 노드 삽입 전에 p가 가리키는 노드 다음에는 다른 노드가 반드시 존재한다.)

<보기>

㉮ new_node->prev = p;
㉯ new_node->next = p->next;
㉰ p->next->prev = new_node;
㉱ p->next = new_node;

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

정답 2번

출제 포인트

이 문제는 이중 연결 리스트에서 p 뒤에 새 노드를 삽입할 때 포인터 갱신의 선후관계를 묻는다.

올바른 순서

1. new_node->prev = p - 새 노드의 이전 링크를 p에 연결한다.
2. new_node->next = p->next - 새 노드의 다음 링크에 기존 다음 노드를 저장한다.
3. p->next->prev = new_node - 기존 다음 노드의 prev를 새 노드로 바꾼다.
4. p->next = new_node - 마지막에 p의 next를 새 노드로 바꾸어 삽입을 완료한다.

왜 이 순서: 기존 p->next를 참조하는 작업이 먼저 끝나야 원래 다음 노드를 잃지 않는다.

NoteBase

선지별 해설

1번: O. (가)와 (나)로 새 노드의 prev, next를 먼저 설정하고, (다)로 기존 다음 노드의 prev를 바꾼 뒤 마지막에 (라)로 p의 next를 새 노드로 연결하므로 최종 연결이 정확하다.
2번: X. 틀린 부분은 (라)를 (다)보다 먼저 둔 점이다. p->next = new_node를 먼저 실행하면 이후의 p->next->prev는 원래 다음 노드의 prev가 아니라 new_node->prev를 뜻하게 되어 new_node->prev = new_node가 된다. 올바르게는 (가) - (나) - (다) - (라)처럼 기존 p->next를 참조하는 작업을 모두 끝낸 뒤 (라)를 마지막에 둔다.
3번: O. (다)를 먼저 실행해도 이 시점의 p->next는 아직 원래 다음 노드를 가리키므로 그 노드의 prev를 새 노드로 바꿀 수 있다. 이후 (나), (가), (라)를 수행하면 최종 연결은 정상이다.
4번: O. (다)와 (나)로 원래 다음 노드 관련 연결을 먼저 처리한 뒤 (라)로 p의 next를 새 노드로 바꾸고, 마지막 (가)에서 new_node->prev를 p로 채우므로 최종 구조는 맞다.

오답 포인트

함정은 p->next를 한 번 바꾼 뒤에도 p->next가 여전히 원래 다음 노드를 가리킨다고 착각하는 데 있다. 2번처럼 (라)를 먼저 수행하면 뒤의 p->next->prev는 기존 다음 노드가 아니라 새 노드 자신의 prev를 건드리게 된다.

다시 나오면 이렇게 판단

이중 연결 리스트 삽입은 기존 링크를 잃지 않는 순서로 판단한다. 기존 p->next를 읽거나 수정하는 문장을 먼저 처리하고, p->next 자체를 새 노드로 바꾸는 문장은 마지막에 두면 된다.

암기 포인트

새 노드 준비, 기존 다음 노드 보정, 마지막에 p->next 교체가 이중 연결 리스트 삽입의 핵심 순서이다.

NoteBase

문 3. <보기>의 C언어 재귀 함수는 크기가 n인 영문과 숫자로 구성된 문자 배열 str[]이 회문(palindrome)이면 1을 반환하고, 그렇지 않으면 0을 반환한다. ㉠과 ㉡에 들어갈 내용을 옳게 짝지은 것은? (단, 회문(palindrome)이란 문자열을 거꾸로 해도 원래의 문자열과 동일한 문자열을 말한다. 예시: 1234321, abcba, a)

<보기>

```
int is_palin(char str[], int n) {
    if ( n <= 1 ) return 1;
    if ( str[0] == str[n-1] ) return is_palin ( ㉠ , ㉡ );
    else return 0;
}
```

① ㉠: str, ㉡: n-1

② ㉠: str, ㉡: n-2

③ ㉠: str+1, ㉡: n-1

④ ㉠: str+1, ㉡: n-2

정답 4번

출제 포인트

이 문제는 재귀 함수가 회문 판별을 위해 비교가 끝난 양 끝 문자를 제외한 부분문자열로 문제를 축소하는 방식을 묻는다.

핵심 개념

회문 판별 재귀는 첫 문자와 마지막 문자가 같으면 가운데 부분문자열만 다시 검사한다. 따라서 시작 위치는 한 칸 뒤로 이동한 str+1이 되고, 검사 길이는 양 끝 두 문자를 제외하므로 n-2가 된다.

NoteBase

선지별 해설

1번 X: ㄱ이 str이면 다시 첫 문자부터 검사하므로 이미 비교한 왼쪽 끝 문자를 또 포함하게 된다. 올바르게는 ㄱ이 str+1이어야 한다.
2번 X: ㄴ이 n-2인 점은 맞지만 ㄱ이 str이면 왼쪽 끝 문자를 제거하지 못한다. 올바르게는 str+1, n-2를 함께 넘겨야 한다.
3번 X: ㄱ이 str+1인 점은 맞지만 ㄴ이 n-1이면 길이가 한 글자만 줄어 이미 비교한 오른쪽 끝 문자를 다시 포함한다. 올바르게는 양 끝 두 문자를 제외한 n-2가 되어야 한다.
4번 O: 첫 문자 str[0]와 마지막 문자 str[n-1]가 같다면 남은 검사 대상은 가운데 부분문자열이다. 따라서 시작 위치는 str+1이고 길이는 n-2이므로 is_palin(str+1, n-2)가 맞다.

오답 포인트

재귀 호출에서 길이만 1 줄이거나 포인터를 그대로 두기 쉽지만, 이 단계에서는 양 끝 두 문자를 이미 소비했다는 점이 핵심이다. str를 그대로 두거나 n만 하나 줄이면 이미 확인한 문자를 다시 검사하게 된다.

다시 나오면 이렇게 판단

재귀 호출 직전에 현재 단계에서 소비한 요소 수를 센다. 양 끝을 비교했다면 시작 위치는 1칸 전진하고 길이는 2 줄어드는지 확인하면 된다.

암기 포인트

양 끝 비교 재귀는 포인터 +1, 길이 -2이다.

NoteBase

문 4. <보기>는 중위 표기식(infix expression)으로 표현된 수식의 앞부분이다. 이 수식을 스택을 활용하여 후위 표기식(postfix expression)으로 변환할 때, 문자 F까지 처리한 후의 스택의 상태에 해당하는 것은?

<보기>

(A-(B+C)*D/(E+F ...

① 

② 

③ 

④ 

정답 4번

NoteBase

출제 포인트

중위 표기식을 후위 표기식으로 바꿀 때 연산자 우선순위와 괄호 처리 규칙에 따라 문자 F까지 읽은 직후 연산자 스택이 어떻게 남는지를 묻는 문제이다.

올바른 순서

1. `(`, `A`, `-`, `(`, `B`, `+`, `C`를 처리하면 스택은 아래에서부터 `(`, `-`, `(`, `+`가 된다.
2. `)`를 만나면 `+`를 출력하고 짝이 되는 `(`를 제거하므로 스택은 `(`, `-`만 남는다.
3. `*`, `D`, `/`를 처리하면 `*`는 `/`보다 먼저 출력되고 스택은 `(`, `-`, `/`가 된다.
4. 다시 `(`를 만나 push하고 `E`, `+`, `F`를 처리하면 스택은 아래에서부터 `(`, `-`, `/`, `(`, `+`가 된다.

왜 이 순서인가: 닫는 괄호 전의 `+`는 아직 출력되지 않고, 괄호 안 연산은 바깥의 `/`, `-`보다 스택 위에 쌓인 채 유지되기 때문이다.

선지별 해설

① X. 보이는 스택이 위에서부터 `+`, `(`, `/`, `-`로만 되어 있어 맨 아래의 바깥 괄호 `(`가 빠져 있다. 올바르게는 `F`까지 처리한 뒤에도 처음 만난 바깥 괄호가 스택 바닥에 남아 있어야 한다.
② X. 보이는 스택이 위에서부터 `+`, `/`, `*`, `-`, `(` 형태이므로 내부 괄호 `(`가 빠져 있고 이미 pop된 `*`가 남아 있다. 올바르게는 `/`를 읽는 순간 `*`는 우선순위 때문에 출력되고, 새로 열린 괄호 `(`가 `/` 위에 올라간다.
③ X. 보이는 스택이 위에서부터 `+`, `(`, `/`, `*`, `-`, `(` 형태이므로 `*`가 아직 스택에 남아 있다고 본 경우이다. 올바르게는 `*`는 `/`를 만나는 시점에 먼저 빠져나가므로 `F`까지 처리한 뒤 스택에 남지 않는다.
④ O. 보이는 스택이 위에서부터 `+`, `(`, `/`, `-`, `(` 형태이고, 이를 아래에서부터 읽으면 `(`, `-`, `/`, `(`, `+`이다. 이는 `F`까지 처리한 직후의 실제 연산자 스택 상태와 일치한다.

오답 포인트

가장 흔한 실수는 `)`를 만났을 때 안쪽 `(`와 `+`를 어떻게 처리하는지, 그리고 `/`를 읽을 때 기존 `*`를 먼저 pop해야 한다는 점을 놓치는 것이다. `*`를 스택에 남겨 두거나 바깥 괄호를 지워 버리면 ②, ③, ① 같은 오답으로 흐른다.

다시 나오면 이렇게 판단

중위식의 일부까지 처리한 스택 문제는 문자 자체보다 사건 순서를 따라가면 된다. `닫는 괄호를 만나면 괄호 안 연산자를 pop한다`, `같거나 높은 우선순위 연산자가 스택 위에 있으면 먼저 pop한다`, `새 괄호를 만나면 그대로 push한다`의 세 규칙으로 스택을 갱신하면 된다.

암기 포인트

괄호는 경계, `)`는 정리, 새 연산자보다 우선순위가 높은 기존 연산자는 먼저 출력된다고 기억하면 된다.

NoteBase

문 5. 시간 복잡도 함수 $f(n)$의 점근 표기법에 대한 설명으로 가장 옳지 않은 것은?

① $f(n) = \Theta(n^2)$이면 $f(n) = O(n^2)$이다.

② $f(n) = O(n^2)$이면 $f(n) = \Theta(n^2)$이다.

③ $f(n) = O(n^2)$이면 $f(n) = O(n^3)$이다.

④ $f(n) = \Omega(n^2)$이면 $f(n) = \Omega(n)$이다.

정답 2번

출제 포인트

이 문제는 점근 표기법의 포함 관계와 O, Ω, Θ의 의미를 정확히 구별하는지를 묻는 문제이다.

핵심 개념

$O(g(n))$는 상한을 뜻하므로 $f(n)$이 결국 $g(n)$보다 크지 않은 정도로 증가함을 나타낸다.
$\Omega(g(n))$는 하한을 뜻하므로 $f(n)$이 결국 $g(n)$만큼은 증가함을 나타낸다.
$\Theta(g(n))$는 상한과 하한을 동시에 만족하는 타이트한 경계를 뜻하므로 $f(n)$과 $g(n)$의 증가 차수가 같은 경우이다.

선지별 해설

① O
$f(n) = \Theta(n^2)$이면 정의상 $f(n) = O(n^2)$이면서 동시에 $f(n) = \Omega(n^2)$이다. 따라서 Θ 관계에서 O 관계를 바로 이끌 수 있다.

② X
틀린 부분은 $f(n) = O(n^2)$이면 반드시 $f(n) = \Theta(n^2)$라고 본 점이다.
$O(n^2)$는 상한만 보장하므로 $f(n) = n$, $f(n) = 1$처럼 n^2보다 훨씬 느리게 증가하는 함수도 포함된다. 올바르게는 $f(n) = \Theta(n^2)$이면 $f(n) = O(n^2)$라고 해야 한다.

③ O
$f(n) = O(n^2)$이면 어떤 상수배의 n^2보다 eventually 작다. 그런데 n^2는 n^3보다 느리게 증가하므로 $O(n^2)$인 함수는 자동으로 $O(n^3)$이기도 하다.

④ O
$f(n) = \Omega(n^2)$이면 $f(n)$은 적어도 이차 함수 수준으로 증가한다. 이차 함수 수준 이상이면 선형 함수 수준 이상도 당연히 만족하므로 $f(n) = \Omega(n)$이다.

NoteBase

오답 포인트

O 표기를 정확한 차수로 오해하면 ②를 맞는 말처럼 보기 쉽다. 이 문항의 함정은 O가 타이트한 경계가 아니라 상한만 준다는 점을 놓치게 만드는 데 있다.

다시 나오면 이렇게 판단

Θ가 나오면 상한과 하한을 동시에 준다고 본다. O는 큰 쪽으로 느슨해질 수 있고, Ω는 작은 쪽으로 느슨해질 수 있으므로 $O(n^2) \Rightarrow O(n^3)$, $\Omega(n^2) \Rightarrow \Omega(n)$는 성립하지만 그 역이나 $O \Rightarrow \Theta$는 일반적으로 성립하지 않는다.

암기 포인트

Θ는 상하한 동시, O는 상한, Ω는 하한이다.

NoteBase

문 6. <보기 1>은 오름차순으로 숫자를 정렬하기 위한 알고리즘을 C언어로 구현한 함수이다. <보기 2>의 길이가 7인 정수 배열을 sort() 함수에 입력하여 오름차순 정렬 연산을 수행하고자 한다. i=4일 때의 정렬을 완료한 직후, 배열의 중간 정렬 결과는?

<보기 1>

```
void sort(int list[], int n)  // n은 항목의 개수임
{
    int i, j, key;
    for(i=1; i<n; i++) {
        key = list[i];
        for(j=i-1; j>=0 && list[j]>key; j--)
            list[j+1] = list[j];
        list[j+1] = key;
    }
}
```

<보기 2>

5	3	8	4	9	1	6

① 1, 3, 4, 5, 8, 9, 6

② 3, 1, 4, 5, 6, 8, 9

③ 3, 4, 5, 8, 9, 1, 6

④ 3, 5, 8, 4, 9, 1, 6

정답 3번

출제 포인트

삽입 정렬에서 반복 변수 *i*가 끝날 때마다 구간 0~*i*만 정렬 완료 상태가 된다는 점을 추적하는 문제이다.

NoteBase

올바른 순서

1. 초기 배열은 5, 3, 8, 4, 9, 1, 6이다.
2. $i = 1$에서는 3을 앞의 5와 비교해 삽입하므로 3, 5, 8, 4, 9, 1, 6이 된다.
3. $i = 2$에서는 8이 5보다 크므로 이동이 없어서 3, 5, 8, 4, 9, 1, 6이 유지된다.
4. $i = 3$에서는 4를 8, 5 앞에 삽입하므로 3, 4, 5, 8, 9, 1, 6이 된다.
5. $i = 4$에서는 9가 이미 앞 구간의 마지막 8보다 크므로 이동이 없어서 3, 4, 5, 8, 9, 1, 6이 된다.

왜 이 순서인가: 삽입 정렬은 현재 원소를 앞쪽의 이미 정렬된 부분배열에 끼워 넣는 방식이므로, i가 끝날 때마다 앞의 $i + 1$개만 정렬된다.

선지별 해설

① X

이 배열은 1이 맨 앞으로 들어간 상태이므로 $i = 5$까지 진행된 뒤의 결과이다. 틀린 표현은 $i = 4$ 완료 직후라는 점이며, 올바르게는 $i = 5$ 완료 직후의 상태이다.

② X

삽입 정렬에서 $i = 4$가 끝나면 앞의 다섯 원소 3, 4, 5, 8, 9가 오름차순이어야 한다. 이 선지는 앞부분에 3, 1이 남아 있어 정렬된 접두 구간이 성립하지 않는다. 올바르게는 3, 4, 5, 8, 9, 1, 6이어야 한다.

③ O

$i = 4$는 다섯 번째 원소 9를 이미 정렬된 3, 4, 5, 8 뒤에 삽입하는 단계이다. 9는 그대로 유지되고 뒤의 1, 6은 아직 처리되지 않으므로 3, 4, 5, 8, 9, 1, 6이 맞다.

④ X

이 배열은 3을 5 앞에 넣은 첫 번째 삽입만 끝난 상태이다. 틀린 표현은 $i = 4$ 완료 직후라는 점이며, 올바르게는 $i = 1$ 완료 직후의 상태이다.

오답 포인트

가장 흔한 착오는 반복 변수 i를 배열의 네 번째 원소 처리로 착각하는 점이다. 이 코드에서 i는 1부터 시작하므로 $i = 4$는 다섯 번째 원소 9를 삽입한 직후를 뜻한다. 또 삽입 정렬은 매 단계마다 앞부분만 정렬되고 뒤쪽 미처리 원소는 그대로 남는다는 점을 놓치기 쉽다.

다시 나오면 이렇게 판단

삽입 정렬 추적 문제는 각 단계가 끝날 때마다 0~i 구간만 오름차순인지 먼저 확인하면 된다. 그다음 현재 key가 앞부분 어디에 들어가는지만 보고, i 뒤의 미처리 구간은 그대로 남는지 확인하면 중간 결과를 빠르게 판별할 수 있다.

NoteBase

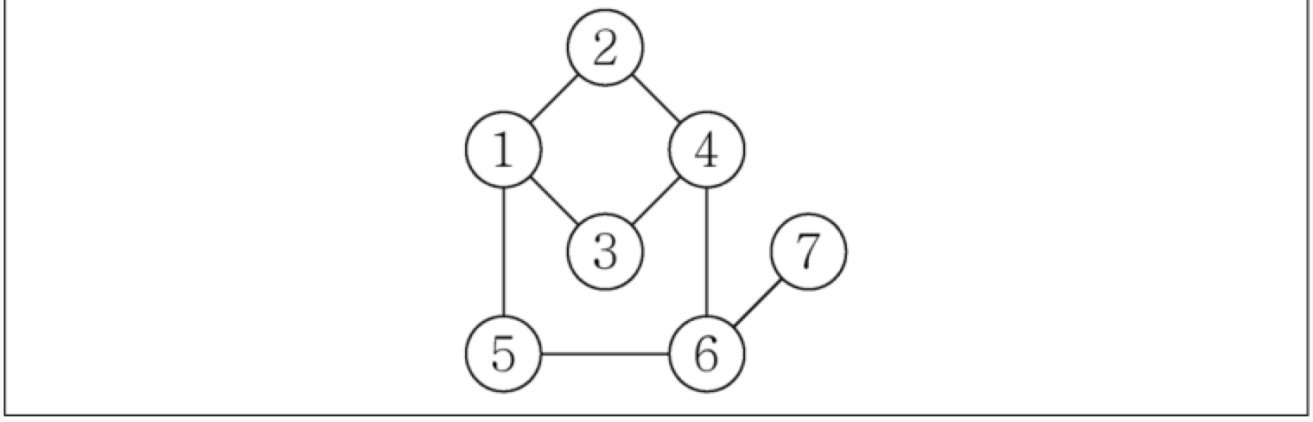
암기 포인트

삽입 정렬은 i가 끝날 때마다 앞의 $i+1$개만 정렬 완료 상태가 된다.

NoteBase

문 7. <보기>의 그래프를 1번 노드에서 시작하여 너비 우선 탐색(breadth first search)을 수행하려 한다. 1번 노드를 첫 번째로 방문한다고 할 때, 다섯 번째로 방문하게 되는 노드는? (만약, 방문할 수 있는 노드가 둘 이상 있다면, 번호가 작은 쪽을 먼저 방문한다.)

<보기>



① 4

② 5

③ 6

④ 7

정답 1번

출제 포인트

이 문제는 그래프 도식에서 각 노드의 인접 관계를 읽고, BFS에서 큐에 들어가는 순서와 실제 방문 순서를 정확히 판단할 수 있는지를 묻는다.

선지별 해설

① O: 1번에서 시작하면 인접한 노드 2, 3, 5를 번호가 작은 순서대로 먼저 방문 대상으로 넣는다. 그다음 2를 처리할 때 새로 발견되는 노드가 4이므로 방문 순서는 1, 2, 3, 5, 4, 6, 7이 되고 다섯 번째는 4이다.
② X: '다섯 번째로 방문하는 노드가 5이다'라는 판단이 틀렸다. 5는 시작 노드 1과 직접 연결되어 있어 첫 확장 단계에서 이미 방문 대상에 포함되므로 실제로는 네 번째로 방문된다.
③ X: '다섯 번째로 방문하는 노드가 6이다'라는 판단이 틀렸다. 6은 5를 처리한 뒤에야 새로 발견되므로 실제로는 여섯 번째로 방문된다.
④ X: '다섯 번째로 방문하는 노드가 7이다'라는 판단이 틀렸다. 7은 6과 연결된 말단 노드이므로 6이 방문된 뒤에야 방문 가능해져 실제로는 일곱 번째로 방문된다.

NoteBase

오답 포인트

BFS에서는 그림 모양대로 깊게 따라가지 않고, 현재 레벨의 인접 노드들을 먼저 모두 처리한다는 점을 놓치면 틀린다. 특히 4와 6을 연속으로 따라가며 깊이 우선처럼 판단하면 ③이나 ④로 오인하기 쉽다.

다시 나오면 이렇게 판단

시작 노드에서 인접 노드를 번호순으로 큐에 넣고, 큐의 앞에서 하나씩 꺼내며 아직 방문하지 않은 인접 노드만 뒤에 추가하는 순서로 방문열을 적으면 된다.

암기 포인트

BFS는 가까운 레벨부터, DFS는 한 경로를 끝까지라는 차이로 구분된다.

NoteBase

문 8. 리스트 $L = (l_1, l_2, \cdots, l_n)$에 대해서, $head(L)$은 L의 첫 번째 원소 l_1을, $tail(L)$은 $(l_2, l_3, \cdots, l_n)$을 나타낸다. 즉, $head(L)$은 주어진 리스트의 맨 첫 번째 원소를 의미하며 $tail(L)$은 그 원소를 제외한 나머지 원소로 이루어진 리스트를 의미한다. $reverse(L)$은 $(l_n, l_{n-1}, \cdots, l_1)$를 나타내며, 주어진 리스트의 원소 순서를 반대로 뒤집어서 만든 리스트를 의미한다. $reverse(L)$과 항상 같은 결과를 나타내는 것은? (단, +는 두 원소 또는 리스트를 순서대로 이어서 만든 리스트를 나타낸다.)

① $l_{n-1} + reverse(tail(L))$

② $head(L) + reverse(tail(L))$

③ $reverse(tail(L)) + head(L)$

④ $reverse(tail(L)) + l_{n-1}$

정답 3번

출제 포인트

이 문제는 head, tail, reverse의 정의와 리스트 연결 연산으로 reverse(L)을 재구성할 수 있는지를 묻는 문제이다.

핵심 개념

head(L)은 리스트의 첫 원소 l_1이다. tail(L)은 첫 원소를 제외한 나머지 리스트 $(l_2, l_3, \ldots, l_n)$이다. reverse(tail(L))은 $(l_n, l_{n-1}, \ldots, l_2)$이므로 reverse(L)을 만들려면 마지막에 head(L)을 이어야 한다.

선지별 해설

① X. 틀린 부분은 맨 앞의 l_{n-1}이다. reverse(tail(L))은 $(l_n, l_{n-1}, \ldots, l_2)$인데 여기에 l_{n-1}를 앞에 붙이면 $(l_{n-1}, l_n, l_{n-1}, \ldots, l_2)$가 되어 원소가 중복되고 l_1이 빠진다. 올바르게는 reverse(tail(L)) + head(L)이다.
② X. 틀린 부분은 head(L)을 앞에 둔 순서이다. head(L) + reverse(tail(L))은 $(l_1, l_n, l_{n-1}, \ldots, l_2)$가 되어 첫 원소가 맨 앞에 남으므로 reverse(L)이 아니다. 올바르게는 head(L)을 맨 뒤에 이어야 한다.
③ O. reverse(tail(L))은 $(l_n, l_{n-1}, \ldots, l_2)$이고 head(L)은 l_1이므로 이를 순서대로 이어 붙이면 $(l_n, l_{n-1}, \ldots, l_2, l_1)$이 되어 reverse(L)와 같다.
④ X. 틀린 부분은 마지막 항을 l_{n-1}로 둔 점이다. reverse(tail(L)) + l_{n-1}은 $(l_n, l_{n-1}, \ldots, l_2, l_{n-1})$가 되어 끝 원소가 l_1이 아니고 l_{n-1}가 중복된다. 올바르게는 reverse(tail(L)) + head(L)이다.

오답 포인트

tail(L)에 이미 l_2부터 l_n까지가 들어 있으므로 reverse(tail(L)) 뒤에 무엇을 붙여야 하는지만 보면 된다. l_{n-1}를 다시 붙이거나 head(L)을 앞에 두면 역순 완성이 깨진다.

NoteBase

다시 나오면 이렇게 판단

리스트의 reverse를 분해해서 묻는 문제는 먼저 tail을 뒤집어 (l_n, …, l_2)를 만든 뒤, 빠진 첫 원소인 head(L)=l_1를 맨 뒤에 붙이는지 확인하면 된다.

암기 포인트

reverse(L)은 reverse(tail(L)) 뒤에 head(L)을 붙인 형태이다.

NoteBase

문 9. <보기>에 주어진 정수 데이터에 대하여 해싱(hashing) 수행 시, 오버플로우(overflow) 문제를 처리하기 위해 체이닝(chaining) 방식을 이용한다. 여기서, 해싱을 위한 해시테이블(hash table)은 5개의 버킷(0~4번)을 가지며, 해시함수는 나머지 연산자(mod 또는 %)를 포함하는 계산함수이다. 모든 데이터 항목에 대한 해싱이 완료된 이후, 해시테이블 내에서 가장 긴 리스트를 가지는 버킷 주소(번호)는?

<보기>

0	1	4	9	16	25	36	49	64	67	28

① 1

② 2

③ 3

④ 4

정답 4번

출제 포인트

이 문제는 해시 함수 key 5로 각 정수를 버킷에 배치한 뒤, 체이닝으로 연결된 리스트의 길이를 비교해 최장 버킷 주소를 찾는 문제이다.

조건 정리

주어진 값: 0, 1, 4, 9, 16, 25, 36, 49, 64, 67, 28
구해야 할 값: 해싱이 끝난 뒤 가장 긴 리스트를 가지는 버킷 주소
주의할 조건: 버킷은 0번부터 4번까지 5개이며, 해시 함수는 각 수를 5로 나눈 나머지이다.

사용할 공식

해시 주소는 h(x)=x 5이다.
이 공식을 쓰는 이유: 문제에서 해시함수가 나머지 연산자를 포함한다고 했으므로, 각 데이터가 어느 버킷에 들어가는지는 5로 나눈 나머지로 바로 결정된다.

NoteBase

계산 과정

1. 각 값을 5로 나눈 나머지를 구한다. 0→0, 1→1, 4→4, 9→4, 16→1, 25→0, 36→1, 49→4, 64→4, 67→2, 28→3이다.
2. 버킷별로 묶으면 0번 버킷은 {0, 25}, 1번 버킷은 {1, 16, 36}, 2번 버킷은 {67}, 3번 버킷은 {28}, 4번 버킷은 {4, 9, 49, 64}이다.
3. 리스트 길이는 각각 2, 3, 1, 1, 4이므로 가장 긴 리스트는 4번 버킷이다.

선지별 해설

① X: 선지가 제시한 주소 '1'은 최장 체인 버킷이 아니다. 1번 버킷에는 1, 16, 36이 들어가 길이가 3이므로, 올바르게는 '가장 긴 리스트를 가지는 버킷 주소는 4번이다'가 된다.
② X: 선지가 제시한 주소 '2'는 67만 들어가는 버킷이어서 길이가 1이다. 최장 버킷으로 보려면 나머지 값의 빈도를 세어야 하며, 올바른 주소는 4번이다.
③ X: 선지가 제시한 주소 '3'은 28만 들어가 길이가 1이다. 3번을 고르는 것은 각 수를 단순히 순서대로 배치하거나 마지막 값만 보고 판단한 경우에 생기는 오답이며, 올바른 주소는 4번이다.
④ O: 4번 버킷에는 4, 9, 49, 64가 들어가 길이가 4이다. 다른 버킷의 길이 2, 3, 1, 1보다 가장 길므로 정답이다.

오답 포인트

체이닝 문제에서 자주 틀리는 지점은 실제로 버킷별 원소 개수를 세지 않고, 중간에 눈에 띄는 나머지 하나만 보고 선택하는 것이다. 특히 1번 버킷도 3개로 길어 보여 헷갈리지만, 4번 버킷은 4개이므로 최장 리스트가 된다.

다시 나오면 이렇게 판단

나머지 해싱 문제는 모든 데이터를 값으로 바꾼 뒤, 버킷별로 묶고 마지막에 각 버킷의 개수만 비교하면 된다. 체이닝 여부를 묻는 순간에는 원소의 저장 순서보다 버킷별 길이 비교가 핵심이다.

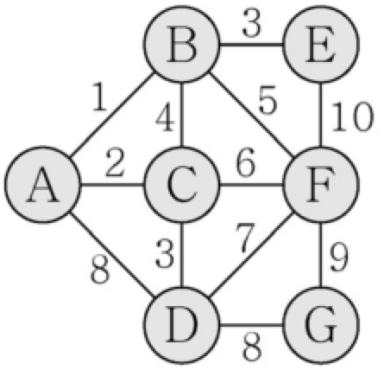
암기 포인트

체이닝 해시의 정답은 각 키의 나머지를 먼저 구하고, 같은 나머지의 개수를 세면 바로 나온다.

NoteBase

문 10. 다익스트라(Dijkstra)의 최단 경로 알고리즘을 이용하여 <보기>의 그래프의 정점 A에서 각 정점으로의 최단 경로를 구할 때, 최단 경로 발견 순서가 정점 E 바로 다음인 정점은? (단, 각 간선 위의 숫자는 양 정점들 사이의 거리를 의미한다.)

<보기>



① B

② C

③ D

④ F

정답 3번

출제 포인트

이 문제는 가중치 그래프에서 다익스트라 알고리즘을 적용하여 시작 정점 A로부터의 최단거리 확정 순서를 판단하는 문제이다.

NoteBase

선지별 해설

① X
B는 A에서의 최단거리가 1이므로 가장 먼저 확정된다. 따라서 E 바로 다음 정점이 될 수 없다는 점이 틀렸다.
틀린 표현은 "E 바로 다음인 정점은 B"이고, 올바르게는 "B는 E보다 먼저 확정된다"이다.

② X
C는 A에서의 최단거리가 2이므로 B 다음으로 확정된다. 따라서 E 다음 정점이라는 판단이 틀렸다.
틀린 표현은 "E 바로 다음인 정점은 C"이고, 올바르게는 "C는 E보다 먼저 확정된다"이다.

③ O
A에서 시작하면 B=1, C=2, D=8로 시작한다. B를 확정하면 E=4, F=6이 되고, C를 확정하면 D=5로 갱신된다.
그다음 E=4가 확정되고, 이후 남은 후보 중 최솟값은 D=5이므로 E 바로 다음 정점은 D가 맞다.

④ X
F의 최단거리는 6이므로 D의 최단거리 5보다 늦게 확정된다. 따라서 E 다음 정점이 F라는 판단이 틀렸다.
틀린 표현은 "E 바로 다음인 정점은 F"이고, 올바르게는 "F는 D가 확정된 뒤에 확정된다"이다.

오답 포인트

초기 간선 가중치만 보고 E 다음을 F로 착각하기 쉽지만, 다익스트라는 직전 확정 정점에서의 갱신 결과를 누적해서 본다. 이 문제에서는 C를 확정한 뒤 D가 8에서 5로 줄어들기 때문에 E 다음 순서가 F가 아니라 D로 바뀐다.

다시 나오면 이렇게 판단

다익스트라 순서 문제는 매 단계마다 미확정 정점의 현재 최단거리 표를 갱신한 뒤 가장 작은 값을 고르면 된다. 초기 거리만 보지 말고 직전 단계에서 새로 줄어든 값이 있는지를 함께 확인하면 된다.

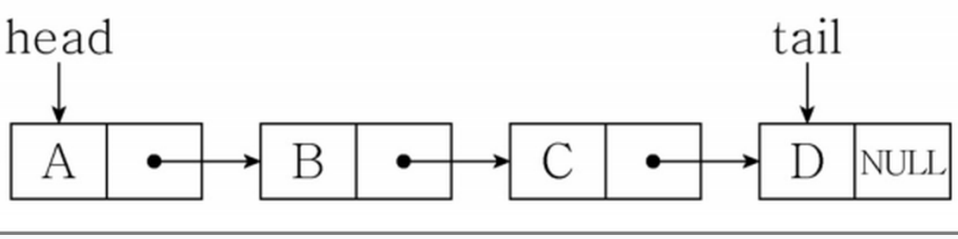
암기 포인트

다익스트라는 매 단계의 현재 최솟값 확정 순서이며, 갱신으로 더 짧아진 정점이 다음 순서를 바꾼다.

NoteBase

문 11. <보기>와 같은 형태의 단순 연결 리스트(singly linked list)에서 add_last 연산과 delete_last 연산의 수행 시간에 대한 설명으로 가장 옳은 것은? (단, add_last는 tail 포인터가 가리키는 노드 다음에 새로운 노드를 생성하여 연결하는 연산이고, delete_last는 tail 포인터가 가리키는 노드를 삭제하는 연산이다.)

<보기>



① 두 연산 모두 O(1) 시간에 수행된다.

② add_last는 O(1) 시간에 수행되나, delete_last는 O(1) 시간에 수행될 수 없다.

③ add_last는 O(1) 시간에 수행될 수 없으나, delete_last는 O(1) 시간에 수행된다.

④ add_last도 O(1) 시간에 수행될 수 없고, delete_last도 O(1) 시간에 수행될 수 없다.

정답 2번

출제 포인트

이 문제는 head와 tail만 있는 단순 연결 리스트 도식에서 마지막 삽입은 tail로 바로 처리할 수 있지만 마지막 삭제는 직전 노드 탐색이 필요한지를 보고 시간 복잡도를 판단하는 문제이다.

선지별 해설

1번 X. add_last는 tail이 마지막 노드를 직접 가리키므로 새 노드를 tail 다음에 연결하고 tail만 갱신하면 되어 O(1)이다. 그러나 delete_last는 tail의 직전 노드를 알아야 연결을 끊고 tail을 갱신할 수 있으므로 head부터 순회해야 한다. 틀린 표현은 "두 연산 모두 O(1)"이고, 올바르게는 "add_last는 O(1)이고 delete_last는 일반적으로 O(n)이다"이다.

2번 O. tail 포인터가 있으므로 add_last는 마지막 노드 뒤에 새 노드를 붙이고 tail을 새 노드로 바꾸면 끝나서 O(1)이다. 반면 단순 연결 리스트는 이전 노드 포인터가 없으므로 delete_last를 하려면 마지막 노드의 직전 노드를 찾기 위한 순회가 필요해 O(1)에 수행될 수 없다.

3번 X. 틀린 표현은 "add_last는 O(1) 시간에 수행될 수 없으나"와 "delete_last는 O(1) 시간에 수행된다"이다. 올바르게는 "add_last는 O(1)에 가능하고, delete_last는 직전 노드 탐색 때문에 O(1)에 불가능하다"이다.

4번 X. 틀린 표현은 "add_last도 O(1) 시간에 수행될 수 없고"이다. tail 포인터가 있는 이상 add_last는 O(1)에 가능하다. 올바르게는 "add_last는 O(1)이고 delete_last는 O(1)이 아니다"이다.

NoteBase

오답 포인트

tail 포인터가 보이면 끝 삽입과 끝 삭제가 모두 빠르다고 묶어서 외우기 쉽다. 그러나 단순 연결 리스트에서는 delete_last 때 필요한 정보가 마지막 노드 자체가 아니라 그 직전 노드라는 점이 함정이다.

다시 나오면 이렇게 판단

단순 연결 리스트의 끝 연산은 직전 노드 정보가 필요한지를 먼저 본다. 새 노드를 뒤에 붙이는 add_last는 tail만 알면 되지만, 마지막 노드를 지우는 delete_last는 직전 노드를 찾아야 하므로 순회가 필요하다.

암기 포인트

단순 연결 리스트는 tail이 있으면 뒤에 붙이기는 O(1)이고, 뒤에서 지우기는 직전 노드 탐색 때문에 O(n)이다.

NoteBase

문 12. 힙(heap)와 힙 정렬(heap sort)에 대한 <보기>의 설명 중 옳은 것을 모두 고른 것은?

<보기>

ㄱ. 최소 힙(min heap)에서 최댓값은 $O(1)$ 시간에 찾을 수 있다.

ㄴ. 최소 힙에서 3번째로 작은 값은 루트의 왼쪽 자식 또는 루트의 오른쪽 자식에 저장된다.

ㄷ. 100개의 원소가 저장된 힙의 높이는 6이다. (단, 루트 노드 하나만 있는 트리의 높이는 0으로 정의한다.)

ㄹ. 힙 정렬의 시간 복잡도는 $O(n\log_2 n)$이다.

① ㄱ, ㄴ

② ㄷ, ㄹ

③ ㄱ, ㄴ, ㄹ

④ ㄴ, ㄷ, ㄹ

정답 2번

출제 포인트

최소 힙의 순서 성질, 완전이진트리 높이 계산, 힙 정렬의 시간 복잡도를 정확히 구분하는 문제이다.

핵심 개념

최소 힙은 부모가 자식보다 작거나 같은 완전이진트리이므로 루트에는 최솟값만 즉시 보장된다.
최댓값은 리프들 중 하나에 있으므로 루트만 보고 바로 알 수 없다.
100개의 원소를 가진 힙의 높이는 완전이진트리 성질로 판단하며, 높이 h에서 노드 수 범위는 2^h개 이상 $2^{h+1}-1$개 이하이다.
힙 정렬은 힙 구성 $O(n)$ 후 삭제를 n번 반복하여 전체가 $O(n\log_2 n)$이다.

NoteBase

선지별 해설

① X

ㄱ과 ㄴ을 모두 옳다고 본 점이 틀리다. ㄱ의 틀린 표현은 "최댓값은 O(1) 시간에 찾을 수 있다"이다. 최소 힙에서 루트는 최솟값만 보장하므로 최댓값은 리프들을 확인해야 하며 보통 $O(n)$이다. ㄴ의 틀린 표현은 "3번째로 작은 값은 루트의 왼쪽 자식 또는 오른쪽 자식에 저장된다"이다. 실제로는 세 번째로 작은 값이 더 작은 자식의 자식에 있을 수도 있으므로 루트의 두 자식으로 한정할 수 없다.

② O

ㄷ과 ㄹ만 옳게 고른 선택지이다. ㄷ은 높이 6인 완전이진트리가 가질 수 있는 노드 수가 64개 이상 127개 이하이므로 100개 원소 힙의 높이가 6이라는 뜻이다. ㄹ은 힙 정렬이 각 삭제 단계마다 $O(\log_2 n)$이 걸리고 이를 n번 반복하므로 전체가 $O(n\log_2 n)$이라는 뜻이다.

③ X

틀린 부분은 ㄱ, ㄴ, ㄹ을 모두 옳다고 본 점이다. ㄹ은 옳지만 ㄱ과 ㄴ이 틀리므로 이 선택지는 성립하지 않는다. 올바르게는 ㄷ, ㄹ만 옳다.

④ X

틀린 부분은 ㄴ, ㄷ, ㄹ을 모두 옳다고 본 점이다. ㄷ과 ㄹ은 옳지만 ㄴ은 세 번째로 작은 값의 위치를 루트의 두 자식으로 제한한 점이 틀리다. 올바르게는 ㄷ, ㄹ만 옳다.

오답 포인트

최소 힙에서 루트가 최솟값이라는 사실을 최댓값까지 확장해서 오해하기 쉽다. 또 두 번째로 작은 값이 루트의 자식 중 하나라는 사실은 맞지만, 이를 세 번째로 작은 값까지 그대로 확대하면 ㄴ처럼 틀리게 된다.

다시 나오면 이렇게 판단

힙 문제는 먼저 루트가 보장하는 값이 무엇인지 확인한다. 최소 힙은 최솟값만 즉시 보장하고, 그 밖의 순위 값은 자식이나 손자까지 열어 두고 판단한다. 높이는 완전이진트리의 노드 수 범위로 계산하고, 힙 정렬은 $O(n)$ 힙 구성과 n번의 $O(\log n)$ 삭제로 본다.

암기 포인트

최소 힙은 루트=최솟값, 최댓값은 리프 쪽, 100개 완전이진트리 높이는 6, 힙 정렬은 $O(n\log n)$이다.

NoteBase

문 13. C언어로 작성된 함수의 시간복잡도를 빅오표기법으로 표현할 때, 가장 낮은 시간복잡도를 가지는 것은?

(가)

```
void func1(int n)
{
    int i, j, k=0;

    for (i=0; i<n; i++)
      for (j=i; j<n; j++)
        k = i * j;
}
```

(나)

```
void func2(int n)
{
    int i, j, a=0;

    for (i=1; i<=n; i=i*2)
      for (j=1; j<=n; j++)
        a = i + j;
}
```

(다)

```
void func3(int n)
{
    int i, j, k=0;

    for (i=0; i<n; ++i)
      for (j=1; j<1000; ++j)
        ++k;
}
```

NoteBase

(라)

```
int func4(int n)
{
    int i, k=0;
    while (n>2) {
        n--;  k++;
        for (i=0; i<n; i++)
            k++;
    }
    return k;
}
```

① (가)

② (나)

③ (다)

④ (라)

정답 3번

출제 포인트

각 함수의 반복문 실행 횟수를 세어 빅오표기법으로 단순화하고, 그중 가장 낮은 차수의 시간복잡도를 고르는 문제이다.

조건 정리

- 주어진 값: (가)~(라) 네 함수의 반복 구조
- 구해야 할 값: 각 함수의 시간복잡도와 그중 가장 낮은 것
- 주의할 조건: 상수 횟수 반복은 $O(1)$로 보고, 반복 횟수의 합은 최고차항만 남겨 비교한다.

사용할 공식

반복문의 총 실행 횟수가 시간복잡도를 결정한다.

$\sum_{i=0}^{n-1}(n-i)=\frac{n(n+1)}{2}=O(n^2)$, $O(\log n)\times O(n)=O(n\log n)$, $O(n)\times O(1)=O(n)$

이 공식을 쓰는 이유: 네 함수 모두 한 줄 연산 자체보다 반복문이 몇 번 도는지가 지배적인 비용이기 때문이다.

NoteBase

계산 과정

1. (가)는 바깥 반복이 n번, 안쪽 반복이 $n-i$번이므로 총합이 $n+(n-1)+\cdots+1$이 되어 $O(n^2)$이다.
2. (나)는 바깥 반복이 1, 2, 4, …로 증가하므로 $O(\log n)$번, 안쪽 반복이 매번 $O(n)$번이라서 전체가 $O(n\log n)$이다.
3. (다)는 바깥 반복이 n번이고 안쪽 반복의 상한 1000은 상수이므로 전체가 $O(n\cdot 1)=O(n)$이다.
4. (라)는 while이 거의 n번 돌고, 그 안의 for가 $n-1, n-2, \ldots, 2$번 실행되므로 총합이 등차수열 합이 되어 $O(n^2)$이다.

선지별 해설

① X
(가)는 안쪽 반복이 $j=i$부터 $n-1$까지 돌아서 총 실행 횟수가 $n+(n-1)+\cdots+1$이다. 따라서 시간복잡도는 $O(n^2)$이다. '가장 낮은 시간복잡도'가 아니라 '이차 시간복잡도를 가진다'로 고쳐야 한다.

② X
(나)는 바깥 반복이 두 배씩 증가하므로 $O(\log n)$번이고, 안쪽 반복이 매번 n번 실행되어 전체는 $O(n\log n)$이다. '가장 낮은 시간복잡도'가 아니라 '선형로그 시간복잡도를 가진다'로 고쳐야 한다.

③ O
(다)는 안쪽 반복의 상한 1000이 입력 n과 무관한 상수이므로 안쪽 반복 전체를 $O(1)$로 본다. 따라서 바깥 반복 n번만 남아 전체는 $O(n)$이고, 네 함수 중 가장 낮다.

④ X
(라)는 while이 한 번 돌 때마다 n이 1씩 줄고, 줄어든 n만큼 다시 for가 돈다. 따라서 총 실행 횟수는 $(n-1)+(n-2)+\cdots+2$에 비례하여 $O(n^2)$이다. '가장 낮은 시간복잡도'가 아니라 '이차 시간복잡도를 가진다'로 고쳐야 한다.

오답 포인트

1000이 큰 수로 보이더라도 입력 n과 무관하면 상수이므로 $O(1)$이다. 또한 $j=i$처럼 시작점이 바뀌는 반복과 while 안에서 길이가 줄어드는 for는 단순 곱셈이 아니라 합으로 세어야 하므로, 이를 놓치면 (가)와 (라)를 과소평가하기 쉽다.

다시 나오면 이렇게 판단

반복문의 증가 규칙이 입력 n에 비례하는지, 두 배씩 변하는지, 상수인지부터 구분한다. 그다음 중첩 반복은 곱으로, 반복 길이가 줄거나 시작점이 이동하면 합으로 계산하면 된다. 상수 반복은 $O(1)$, 1, 2, 4, … 꼴은 $O(\log n)$, $1+2+\cdots+n$ 꼴은 $O(n^2)$로 판단한다.

NoteBase

암기 포인트

상수 반복은 $O(1)$, 두 배 증가 반복은 $O(\log n)$, 등차수열 합은 $O(n^2)$이다.

NoteBase

문 14. <보기>와 같이 단순 연결 리스트(singly linked list)가 메모리에 저장되어 있다. 이 외의 다른 정보는 메모리에 저장되어 있지 않다고 할 때, 리스트에 저장된 원소를 처음부터 링크를 따라 순서대로 바르게 나열한 것은?

<보기>

주소	데이터	다음 노드 주소 링크
100	31	NULL
104	41	116
108	59	112
112	26	104
116	53	100

① 59 - 26 - 41 - 53 - 31

② 31 - 41 - 59 - 26 - 53

③ 53 - 26 - 59 - 41 - 31

④ 31 - 53 - 41 - 26 - 59

정답 1번

출제 포인트

이 문제는 단순 연결 리스트에서 머리 노드를 찾고 링크를 따라 전체 저장 순서를 복원하는지를 묻는 문제이다.

올바른 순서

1. 주소 108의 데이터 59 — 108은 어떤 노드의 링크에도 등장하지 않으므로 머리 노드이다.
2. 주소 112의 데이터 26 — 108의 링크가 112이므로 두 번째 원소이다.
3. 주소 104의 데이터 41 — 112의 링크가 104이므로 그다음 원소이다.
4. 주소 116의 데이터 53 — 104의 링크가 116이므로 다음 원소이다.
5. 주소 100의 데이터 31 — 116의 링크가 100이고 100의 링크는 NULL이므로 마지막 원소이다.

왜 이 순서: 단순 연결 리스트는 현재 노드의 링크가 다음 노드의 주소를 가리키므로, 어떤 링크에서도 가리켜지지 않는 노드가 시작점이 되고 NULL을 만날 때까지 따라가면 전체 순서가 복원된다.

NoteBase

선지별 해설

① O
59 - 26 - 41 - 53 - 31 순서는 108 → 112 → 104 → 116 → 100을 따라간 결과와 일치한다.

② X
틀린 부분: 첫 원소를 31로 둔 점이 틀리다.
올바르게는: 31은 주소 100에 저장되어 있고 링크가 NULL이므로 마지막 원소이다. 전체 순서는 59 - 26 - 41 - 53 - 31이다.

③ X
틀린 부분: 53을 첫 원소로 두고 26 뒤에 59를 둔 점이 틀리다.
올바르게는: 53은 주소 116의 데이터로서 104 다음에 오고, 59는 어떤 링크에도 가리켜지지 않는 주소 108의 데이터이므로 맨 앞에 와야 한다. 전체 순서는 59 - 26 - 41 - 53 - 31이다.

④ X
틀린 부분: 31을 첫 원소로 두고 53을 두 번째 원소로 둔 점이 틀리다.
올바르게는: 31은 NULL을 가지는 마지막 노드의 데이터이고, 53은 41 다음에 오는 네 번째 원소이다. 전체 순서는 59 - 26 - 41 - 53 - 31이다.

오답 포인트

주소의 크기 순서나 데이터의 크기 순서로 나열하면 틀린다. 이 문항의 함정은 100, 104, 108, 112, 116이 정렬되어 보인다는 점과 31이 맨 위 행에 있어 시작점처럼 보인다는 점이다. 실제 시작점은 표의 첫 행이 아니라 어떤 링크에서도 가리켜지지 않는 주소 108이다.

다시 나오면 이렇게 판단

단순 연결 리스트 표가 나오면 먼저 다음 주소 링크 칼럼에 한 번도 나타나지 않는 주소를 찾는다. 그 주소를 머리 노드로 잡고 링크를 따라가며 데이터를 적으면 저장 순서가 복원된다. 링크가 NULL인 노드는 항상 마지막 노드이다.

암기 포인트

단순 연결 리스트는 가리켜지지 않는 주소에서 시작하고 NULL에서 끝난다.

NoteBase

문 15. 스레드 이진트리(threaded binary tree)에 대한 설명으로 가장 옳지 않은 것은?

① 노드가 n개일 때, 순회 연산 시간은 $O(\log_2 n)$이다.

② 스택(stack)을 사용하지 않고 중위 순회가 가능하다.

③ 이진트리(binary tree)에 비해서 삽입/삭제 연산 시간이 증가한다.

④ 스레드인지 자식 노드에 대한 포인터인지 구별하기 위한 추가적인 필드가 필요하다.

정답 1번

출제 포인트

이 문제는 스레드 이진트리의 순회 특성과 일반 이진트리 대비 구조적 차이를 구별하는 문제이다.

핵심 개념

스레드 이진트리는 NULL 포인터를 중위 선행자나 중위 후속자로 연결해 순회를 쉽게 만든 이진트리이다. 일반 이진트리는 자식 링크만 관리하지만, 스레드 이진트리는 자식 링크와 스레드 링크를 함께 구분해 관리한다.

선지별 해설

① X
- 틀린 부분은 "순회 연산 시간은 O($\log_2$n)"이라는 표현이다.
- 스레드 이진트리는 스택이나 재귀 없이 중위 순회를 쉽게 할 수 있지만, 전체 순회에서는 결국 모든 노드를 방문하므로 시간은 O(n)이다.
- 올바르게는 "노드가 n개일 때 전체 순회 연산 시간은 O(n)이다."가 된다.

② O
- 스레드는 비어 있는 링크를 중위 선행자 또는 후속자로 연결한 것이므로, 스택을 사용하지 않고도 중위 순회를 진행할 수 있다.

③ O
- 삽입과 삭제에서는 일반 이진트리의 부모-자식 링크뿐 아니라 스레드 링크도 함께 수정해야 한다. 이 때문에 구현과 갱신 부담이 커져 일반 이진트리보다 연산이 복잡해진다.

④ O
- 어떤 링크가 실제 자식을 가리키는지, 어떤 링크가 스레드인지를 구별해야 하므로 보통 tag 비트나 추가 필드가 필요하다.

NoteBase

오답 포인트

스레드 이진트리는 순회를 빠르게 만든다는 표현 때문에 ①의 시간복잡도까지 줄어든다고 착각하기 쉽다. 이 구조는 "다음 노드로 이동하는 방법"을 단순하게 만들 뿐, 전체 순회에서 방문해야 하는 노드 수 자체를 줄이지는 않는다.

다시 나오면 이렇게 판단

트리 순회 문제에서는 "이동 편의성"과 "전체 방문 횟수"를 분리해서 본다. 스택 제거 여부는 구현 방법의 차이이고, 전체 순회 시간은 모든 노드를 방문하면 O(n)으로 판단한다.

암기 포인트

스레드 이진트리는 순회를 쉽게 만들지만, 전체 순회 시간 자체를 O(log n)으로 줄이지는 않는다.

NoteBase

문 16. C언어에서 3차원 배열 A[5][6][7]이 선언되었을 때, 메모리 상에서 원소 A[2][3][4]보다 앞에 있는 원소의 개수는? (단, 다차원 배열의 원소는 행우선 방식으로 순서대로 저장된다고 가정한다. 즉, A[0][0][0], A[0][0][1], A[0][0][2], ... 순으로 저장된다.)

① 9

② 56

③ 59

④ 109

정답 4번

출제 포인트

이 문제는 C언어의 행우선 저장 방식에서 3차원 배열 원소의 선형 인덱스를 구하는 문제이다.

조건 정리

- 배열의 크기는 A[5][6][7]이다.
- 저장 순서는 행우선(row-major)이다.
- A[0][0][0], A[0][0][1], A[0][0][2], ... 순서로 저장된다.
- 구해야 할 값은 A[2][3][4]보다 메모리상 앞에 있는 원소 수이다.
- 첨자는 0부터 시작하므로 앞에 있는 원소 수는 해당 원소의 선형 인덱스와 같다.

사용할 공식

행우선 3차원 배열에서 A[i][j][k]의 선형 인덱스는 $i \times (6 \times 7) + j \times 7 + k$이다.
이 공식을 쓰는 이유는 첫 번째 첨자 하나가 바뀔 때마다 뒤의 6×7개 원소가 한 묶음으로 지나가고, 두 번째 첨자 하나가 바뀔 때마다 7개 원소가 한 묶음으로 지나가기 때문이다.

계산 과정

1. 첫 번째 첨자 2는 앞선 2개의 큰 블록을 뜻하므로 $2 \times 6 \times 7 = 84$이다.
2. 두 번째 첨자 3은 현재 블록 안에서 앞선 3개의 행을 뜻하므로 $3 \times 7 = 21$이다.
3. 세 번째 첨자 4는 현재 행 안에서 앞선 4개의 원소를 뜻한다.
4. 따라서 앞에 있는 원소 수는 $84 + 21 + 4 = 109$이다.

NoteBase

선지별 해설

① X
9는 첨자 2, 3, 4를 단순히 더한 값일 뿐 메모리상 앞선 원소 수가 아니다. 올바르게는 차원별 묶음 크기를 반영하여 $2 \times 42 + 3 \times 7 + 4 = 109$로 계산한다.

② X
56은 행우선 저장에서 첫 번째 첨자가 건너뛰는 큰 블록 크기 $6 \times 7 = 42$를 제대로 반영하지 못한 값이다. 올바르게는 첫 번째 첨자만으로도 이미 84개의 원소가 앞에 있다.

③ X
59도 각 차원의 가중치가 42, 7, 1이라는 점을 적용하지 않은 오답이다. 올바르게는 앞선 2차원 블록 84개, 같은 블록의 앞선 행 21개, 같은 행의 앞선 원소 4개를 더해 109가 된다.

④ O
행우선 저장에서는 마지막 첨자가 가장 빨리 변하므로 앞에 있는 원소 수는 선형 인덱스와 같다. 계산 결과가 109이므로 옳다.

오답 포인트

첨자값 2, 3, 4를 그대로 더하거나, 각 첨자가 건너뛰는 묶음 크기 42, 7, 1를 놓치면 오답이 된다. 다차원 배열 주소 계산에서는 각 첨자에 뒤 차원들의 크기 곱을 가중치로 붙여야 한다.

다시 나오면 이렇게 판단

행우선 다차원 배열 문제는 마지막 차원부터 묶음 크기를 만든 뒤 각 첨자에 그 가중치를 곱해 더하는 방식으로 판단한다. 이 문항에서는 가중치가 42, 7, 1이므로 $2 \times 42 + 3 \times 7 + 4$로 계산한다.

암기 포인트

행우선 3차원 배열의 앞선 원소 수는 첫 첨자 × 뒤 두 차원의 곱 + 둘째 첨자 × 마지막 차원의 크기 + 셋째 첨자이다.

NoteBase

문 17. <보기>의 2-3 트리에 키값 2, 3, 4, 5, 6, 7, 8을 순서대로 삽입할 경우, 각 삽입 단계별 결과에서 루트 노드가 되는 키값은?

<보기>



① (1,2) - (2) - (2) - (2,4) - (2,4) - (4) - (4)

② (1,2) - (3) - (3) - (4) - (4,6) - (4,7) - (4)

③ (2) - (3) - (2,3) - (2,3) - (3,4) - (3,4) - (4,5)

④ (1,2) - (3) - (3) - (3,4) - (4,5) - (4,7) - (4)

정답 1번

출제 포인트

이 문제는 2-3 트리에서 삽입이 일어날 때 리프 분할과 부모 승격 때문에 루트 키가 단계별로 어떻게 변하는지를 추적하는 문제이다.

올바른 순서

1. 2 삽입: 기존 루트 1의 빈 칸에 2가 들어가 루트는 (1,2)가 된다.
2. 3 삽입: 루트가 (1,2,3)으로 넘치므로 가운데 키 2가 승격되어 루트는 (2)가 된다.
3. 4 삽입: 4는 오른쪽 리프에 들어가고 분할이 없으므로 루트는 (2)로 유지된다.
4. 5 삽입: 오른쪽 리프가 (3,4,5)로 넘쳐 가운데 키 4가 승격되므로 루트는 (2,4)가 된다.
5. 6 삽입: 가장 오른쪽 리프에 들어가고 분할이 없으므로 루트는 (2,4)로 유지된다.
6. 7 삽입: 가장 오른쪽 리프가 넘쳐 6이 부모로 승격되고, 부모 루트가 (2,4,6)으로 넘치므로 다시 분할되어 새 루트는 (4)가 된다.
7. 8 삽입: 새 루트 4의 오른쪽 서브트리로 들어가며 추가 분할이 없으므로 루트는 (4)로 유지된다.

왜 이 순서인가: 2-3 트리는 리프에서 넘친 노드의 가운데 키가 부모로 올라가고, 부모도 넘치면 같은 규칙으로 다시 분할되기 때문이다.

NoteBase

선지별 해설

① O: (1,2) → (2) → (2) → (2,4) → (2,4) → (4) → (4) 순서가 2-3 트리 삽입 규칙과 정확히 일치한다. 3 삽입 때는 가운데 키 2가 루트가 되고, 7 삽입 때는 6의 승격으로 루트가 넘친 뒤 다시 분할되어 4가 새 루트가 된다.
② X: 틀린 표현은 3 삽입 후 루트를 (3)으로 본 부분과 그 뒤의 (4), (4,6), (4,7)이다. 올바르게는 3 삽입 후 가운데 키 2가 승격되어 루트는 (2)이고, 5 삽입 후에는 (2,4), 6 삽입 후에도 (2,4), 7 삽입 후에는 루트 분할로 (4)가 된다.
③ X: 틀린 표현은 첫 단계부터 루트를 (2)로 본 부분이다. 올바르게는 초기 루트에 1이 이미 있으므로 2 삽입 직후 루트는 (1,2)이다. 이후에도 3 삽입 후는 (2), 5 삽입 후는 (2,4), 7 삽입 후는 (4)여야 하므로 전체 흐름이 맞지 않는다.
④ X: 틀린 표현은 3 삽입 후 루트를 (3)으로 본 부분, 4 삽입 후 (3,4), 5 삽입 후 (4,5), 6 삽입 후 (4,7)로 본 부분이다. 올바르게는 승격되는 키는 항상 넘친 노드의 가운데 키이므로 3 삽입 후는 (2), 4 삽입 후는 (2), 5 삽입 후와 6 삽입 후는 (2,4), 7 삽입 후는 (4)이다.

오답 포인트

가장 흔한 함정은 승격되는 키를 새로 삽입한 큰 값으로 착각하는 점이다. 2-3 트리 분할에서는 항상 가운데 키가 부모로 올라가므로 3 삽입 때 3이 아니라 2가 루트가 되고, 7 삽입 때도 7이 아니라 6이 먼저 올라간 뒤 루트 분할 결과 4가 새 루트가 된다.

다시 나오면 이렇게 판단

2-3 트리 삽입 문제는 삽입 위치를 리프에서 먼저 찾고, 노드가 넘치면 가운데 키가 부모로 승격된다는 규칙만 단계별로 적용하면 된다. 부모까지 넘치면 같은 분할 규칙을 한 번 더 적용하여 최종 루트를 판단할 수 있다.

암기 포인트

2-3 트리 삽입은 리프 삽입 후 가운데 키 승격, 부모가 넘치면 다시 가운데 키 승격으로 정리된다.

NoteBase

문 18. 어떤 그래프를 나타내는 인접 행렬(adjacency matrix)이 <보기>와 같이 주어졌을 때, 이로부터 알 수 있는 그래프에 대한 설명으로 가장 옳지 않은 것은?

<보기>

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

① 이 그래프의 간선(edge)은 방향이 없거나, 모든 간선(edge)이 양방향이다.

② 노드의 개수는 총 5개이다.

③ 어떤 노드도 자기 자신을 잇는 루프(loop)가 없다.

④ 이 그래프는 연결 그래프이다.

정답 3번

출제 포인트

인접 행렬의 대칭성, 행렬의 크기, 주대각선 원소를 보고 그래프의 방향성, 노드 수, 루프 존재, 연결성을 판단하는 문제이다.

자료 읽는 순서

1. 행과 열의 수를 보고 노드 개수를 확인한다.
2. 행렬이 주대각선을 기준으로 대칭인지 보고 방향성을 판단한다.
3. 주대각선의 값이 1인 위치가 있는지 보고 루프 존재를 확인한다.
4. 1이 있는 위치를 따라 모든 노드가 하나로 이어지는지 보고 연결성을 판단한다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 자료구조론 · 2021

선지별 해설

① O
행렬이 주대각선을 기준으로 대칭이므로 $A_{ij} = A_{ji}$이다. 이런 인접 행렬은 무방향 그래프이거나 모든 간선이 양방향인 방향 그래프로 해석할 수 있다.

② O
행과 열이 각각 5개이므로 인접 행렬의 크기는 5 × 5이다. 인접 행렬의 차수는 곧 노드 수이므로 노드는 총 5개이다.

③ X
3행은 [0, 1, 1, 0, 0]이므로 3행 3열의 값이 1이다. 인접 행렬에서 주대각선(3행 3열)의 1은 3번 노드가 자기 자신과 이어지는 루프를 뜻한다. 따라서 "어떤 노드도 자기 자신을 잇는 루프가 없다"는 설명은 옳지 않으며, 올바르게는 "3번 노드에는 자기 자신을 잇는 루프가 있다"이다.

④ O
각 노드의 인접 관계를 정리하면 1번-{2번}, 2번-{1번, 3번, 4번, 5번}, 4번-{2번, 5번}, 5번-{2번, 4번}이다. 1번, 3번, 4번, 5번이 모두 2번을 통해 이어지고 4번과 5번도 서로 연결되므로, 5개 노드가 하나의 연결 성분에 포함된다. 따라서 이 그래프는 연결 그래프이다.

오답 포인트

③은 비대각 원소만 보고 지나가면 맞는 설명처럼 보이기 쉽다. 인접 행렬에서는 주대각선의 1이 루프를 뜻하므로 3행 3열의 값을 확인하지 않으면 함정에 걸리게 된다.

다시 나오면 이렇게 판단

인접 행렬 문제는 먼저 행렬 크기로 노드 수를 보고, 다음에 대칭성으로 방향성을 판단하며, 주대각선으로 루프를 확인한 뒤, 마지막으로 1의 연결 경로를 따라 연결성을 판정하면 된다.

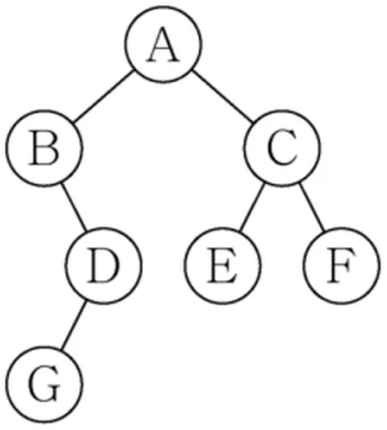
암기 포인트

주대각선의 1은 루프, 대칭 인접 행렬은 무방향 그래프라는 점이 핵심이다.

NoteBase

문 19. <보기 1>의 트리의 루트(root)가 A노드일 때, 루트 노드 A를 인자로 하여 <보기 2>의 함수를 수행한 결과는?

<보기 1>



<보기 2>

```
typedef struct Node{
    char data;  //  노드에 저장된 문자
    struct Node *left, *right;  //  자식 포인터
} Node;
void traverse(Node *node) {
    if( !node ) return;
    traverse(node->left);
    printf(" %c", node->data);
    traverse(node->right);
    printf(" %c", node->data);
}
```

① A B D G C E F G D B E F C A

② A B D G G D B C E E F F C A

③ B G D A E C F G D B E F C A

④ B G G D D B A E E C F F C A

정답 4번

NoteBase

출제 포인트

이 문제는 이진트리의 자식 방향을 정확히 읽고, `traverse(node->left) -> 현재 노드 출력 -> traverse(node->right) -> 현재 노드 재출력` 순서를 전체 트리에 적용하는 문제이다.

선지별 해설

① X: 맨 앞이 `A`로 시작하는 점이 틀리다. 이 함수는 루트 A를 바로 출력하지 않고 왼쪽 서브트리를 먼저 모두 순회한 뒤 A를 출력한다. 또한 리프 G는 방문 시 `G G`로 연속 출력되어야 하는데 ①은 G가 한 번만 나온 뒤 뒤쪽에서 다시 나온다. 올바르게는 `B G G D D B A E E C F F C A`이다.
② X: 역시 `A`를 첫 글자로 둔 점이 틀리다. 루트 A는 왼쪽 서브트리 `B G G D D B`가 끝난 뒤에 출력된다. 또한 B의 오른쪽 자식 D는 왼쪽 자식 G를 먼저 처리한 뒤 `D D`가 연속으로 나와야 하는데 ②는 `B D G G D B`처럼 D를 너무 일찍 출력했다. 올바르게는 `B G G D D B A E E C F F C A`이다.
③ X: 왼쪽 서브트리 처리 결과가 틀리다. 리프 G는 자식이 없으므로 호출 한 번에 `G G`가 연속 출력되어야 하는데 ③은 `B G D`처럼 G를 한 번만 출력하고 곧바로 D로 넘어간다. 또 루트 A는 왼쪽 서브트리가 완전히 끝난 뒤에 나와야 하므로 `B G G D D B` 뒤에 위치해야 한다. 올바르게는 `B G G D D B A E E C F F C A`이다.
④ O: A의 왼쪽 서브트리 B부터 계산하면 B는 왼쪽 자식이 없으므로 먼저 `B`를 출력하고, 오른쪽 자식 D에서 왼쪽 리프 G를 처리해 `G G`, 이어서 D를 두 번 출력해 `D D`, 마지막으로 B를 다시 출력하므로 `B G G D D B`가 된다. 그다음 루트 A를 출력하고, 오른쪽 서브트리 C에서는 E가 `E E`, C가 한 번, F가 `F F`, C가 한 번 더 출력되어 `E E C F F C`가 된다. 마지막으로 A를 다시 출력하므로 전체 결과는 `B G G D D B A E E C F F C A`이다.

오답 포인트

중위순회라고 보고 `left-root-right`까지만 적용하면 오답이 된다. 이 함수는 오른쪽 서브트리까지 끝낸 뒤 현재 노드를 한 번 더 출력하므로 리프는 같은 문자가 두 번 연속 나오고, 내부 노드도 각자의 위치에서 두 번 등장한다는 점이 함정이다.

다시 나오면 이렇게 판단

재귀 함수의 출력 위치를 먼저 표시한 뒤, 리프부터 한 단계씩 문자열을 합치면 된다. 이 문제에서는 `리프는 자기 문자 2번`, `내부 노드는 왼쪽 결과 + 자기 문자 + 오른쪽 결과 + 자기 문자` 규칙으로 아래에서 위로 계산하면 된다.

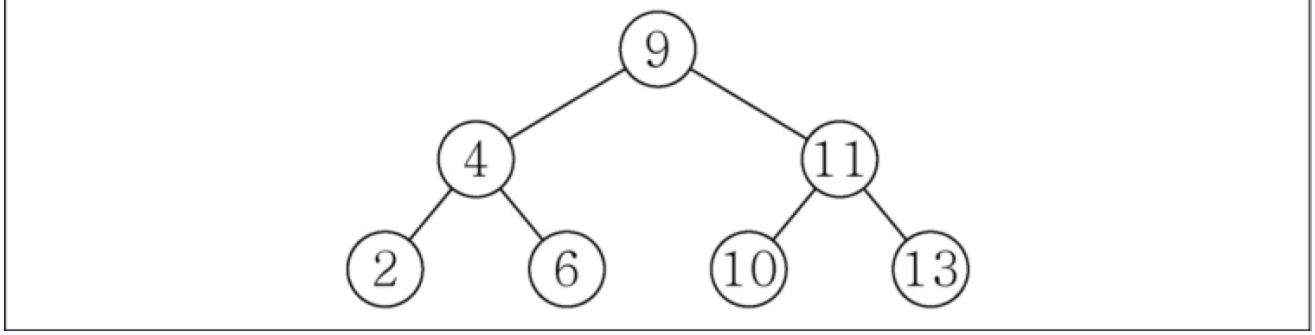
암기 포인트

`left-root-right-root`이면 리프는 두 번, 내부 노드는 서브트리 사이와 끝에서 한 번씩 출력된다.

NoteBase

문 20. <보기>의 이진 탐색 트리(binary search tree)가 만들어질 수 없는 입력순서는?

<보기>



① 9 - 11 - 6 - 4 - 2 - 13 - 10

② 9 - 4 - 11 - 2 - 6 - 13 - 10

③ 9 - 11 - 4 - 6 - 2 - 13 - 10

④ 9 - 4 - 11 - 2 - 6 - 10 - 13

정답 1번

출제 포인트

이 문제는 이진 탐색 트리에 값을 삽입할 때 각 부분트리의 루트가 그 부분트리에 속한 다른 값보다 먼저 입력되어야 한다는 선후관계를 묻는 문제이다.

올바른 순서

1. 9가 전체 트리의 루트가 된다.
2. 9보다 작은 값들 중 가장 먼저 입력되는 값이 왼쪽 부분트리의 루트가 된다. 이 그림에서는 4가 2와 6보다 먼저 나와야 한다.
3. 9보다 큰 값들 중 가장 먼저 입력되는 값이 오른쪽 부분트리의 루트가 된다. 이 그림에서는 11이 10과 13보다 먼저 나와야 한다.
4. 그다음 2와 6은 4의 왼쪽과 오른쪽에, 10과 13은 11의 왼쪽과 오른쪽에 배치된다.

왜 이 순서: 부분트리의 자식 노드는 그 부분트리의 루트가 먼저 생성된 뒤에야 원하는 위치로 내려갈 수 있기 때문이다.

NoteBase

선지별 해설

① X
- 9보다 작은 값들 중 6이 4보다 먼저 입력되므로 왼쪽 부분트리의 루트가 6이 된다. 그러면 4는 6의 왼쪽 자식으로 들어가므로 제시된 트리처럼 4가 왼쪽 부분트리의 루트가 될 수 없다.
- 틀린 표현: "9-11-6-4-2-13-10"
- 올바르게는: 왼쪽 부분트리의 루트 4가 2와 6보다 먼저 입력되어야 하므로 "9-11-4-6-2-13-10"처럼 4가 먼저 나와야 한다.

② O
- 9 다음에 4와 11이 각각 왼쪽과 오른쪽 부분트리의 루트가 된다. 이후 2와 6은 4의 왼쪽과 오른쪽에, 13과 10은 11의 오른쪽과 왼쪽에 정상적으로 삽입되어 제시된 트리가 만들어진다.

③ O
- 11이 4보다 먼저 나와도 문제는 없다. 왼쪽 부분트리에서는 4가 6과 2보다 먼저 나오고, 오른쪽 부분트리에서는 11이 13과 10보다 먼저 나오므로 두 부분트리 모두 제시된 구조를 유지한다.

④ O
- 4와 11이 각 부분트리의 루트로 먼저 자리 잡고, 2와 6, 10과 13이 각각 그 자식 위치로 들어가므로 그림과 정확히 같은 이진 탐색 트리가 만들어진다.

오답 포인트

전체 입력 순서를 그대로 외우려 하면 헷갈리기 쉽다. 핵심은 각 부분트리에서 가장 먼저 등장한 값이 그 부분트리의 루트가 된다는 점이다. ①은 6이 4보다 먼저 나와 왼쪽 부분트리의 루트가 바뀐다는 점을 놓치면 틀리기 쉽다.

다시 나오면 이렇게 판단

삽입 순서로 BST를 만드는 문제는 전체 루트를 먼저 확인하고, 그다음 왼쪽 집합과 오른쪽 집합을 나눈 뒤 각 집합에서 가장 먼저 등장한 값이 부분트리의 루트가 되는지만 보면 된다. 자식 후보가 부모 후보보다 먼저 나오면 그 선택지는 바로 배제된다.

암기 포인트

BST 삽입에서는 먼저 나온 값이 그 구간의 루트가 된다.

NoteBase