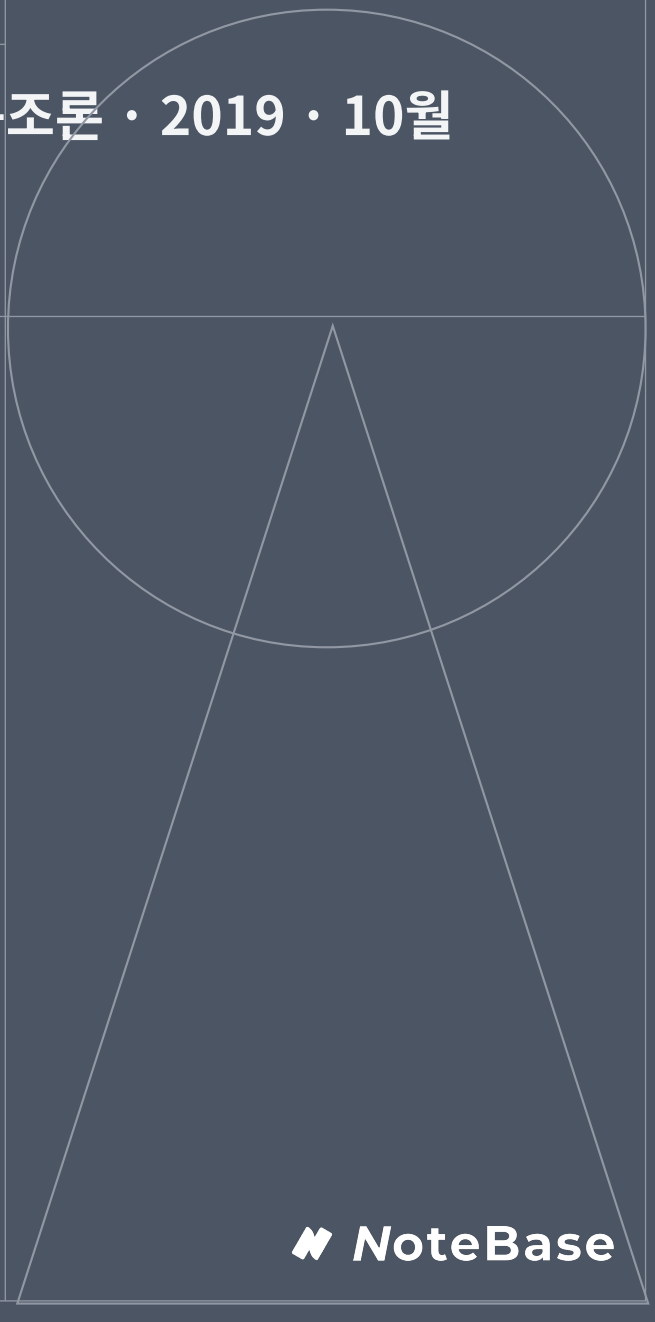
기출OX 해설집

공무원/서울시/7급 · 자료구조론 · 2019 · 10월



NoteBase

기출OX 해설집

공무원/서울시/7급 · 자료구조론 · 2019 · 10월

공무원/서울시/7급 · 자료구조론 · 2019 · 10월

문 1. <보기>와 같은 함수에서 f(4)를 수행한 결과 값은?

<보기>

```
int f(int n)
{
    if (n<=1) return n;
    return (5*f(n-1)-6*f(n-2));
}
```

① 9

② 19

③ 56

④ 65

정답 4번

정답

정답은 4번이다.

출제 포인트

이 문제는 재귀식 $f(n) = 5f(n-1) - 6f(n-2)$에 기저값을 대입해 $f(4)$를 순서대로 구할 수 있는지를 묻는 문제이다.

조건 정리

주어진 조건: $n \leq 1$이면 $f(n) = n$이다.
구해야 할 값: $f(4)$이다.
주의할 조건: $f(4)$를 바로 계산할 수 없으므로 $f(2)$, $f(3)$를 먼저 구한 뒤 차례로 대입해야 한다.

NoteBase

사용할 공식

재귀식은 $f(n) = 5f(n-1) - 6f(n-2)$이다.
이 공식을 쓰는 이유: 문제에서 함수값이 바로 주어진 것이 아니라 이전 두 항으로 다음 항을 만드는 점화 형태이므로, 작은 n부터 값을 계산해 올라가야 하기 때문이다.

계산 과정

1. 기저값은 $f(0) = 0$, $f(1) = 1$이다.
2. $f(2) = 5f(1) - 6f(0) = 5 \times 1 - 6 \times 0 = 5$이다.
3. $f(3) = 5f(2) - 6f(1) = 5 \times 5 - 6 \times 1 = 25 - 6 = 19$이다.
4. $f(4) = 5f(3) - 6f(2) = 5 \times 19 - 6 \times 5 = 95 - 30 = 65$이다.

선지 판단

1번 X: 9는 $f(4)$의 계산 결과와 다르다. $f(4) = 95 - 30 = 65$이므로 "결과값이 9이다"라는 판단은 틀렸고, 올바르게는 65이다.
2번 X: 19는 $f(3)$의 값이다. 한 단계 전 값과 $f(4)$를 혼동한 것이며, 올바르게는 65이다.
3번 X: 56은 점화식에 값을 정확히 대입한 결과가 아니다. $5f(3) - 6f(2) = 95 - 30$을 계산하면 65이므로 올바르게는 65이다.
4번 O: $f(4) = 65$이므로 맞는 선택지이다.

오답 포인트

기저값 $f(0)$, $f(1)$을 먼저 확정하지 않으면 이후 계산이 모두 흔들린다. 또 19처럼 바로 앞 단계의 값 $f(3)$을 최종값으로 착각하거나, 마지막 뺄셈을 부정확하게 계산하면 오답이 된다.

다시 나오면 이렇게 판단

재귀식 문제는 기저값 확인 -> 작은 인덱스부터 순차 계산 -> 마지막 식에 대입한 뒤 산술 검산의 순서로 판단한다.

암기 포인트

재귀함수 값 계산은 기저값을 먼저 정하고 $f(2)$부터 한 단계씩 올려 가면 된다.

NoteBase

문 2. <보기>의 C언어 2차원 배열에 대한 설명으로 가장 옳지 않은 것은?

int mat[3][4] = {1, 2, 3, 4, 5, 6};

① mat[2][2]의 값은 0이다.
② mat[1] == &mat[1][0]이고, mat[1]의 자료형은 int*이다.
③ **mat은 mat[0][0]과 같은 의미이고, 자료형은 int*이다.
④ *(mat[2] + 3)은 mat[2][3]을 의미한다.

정답 3번

정답

정답은 3번이다.

출제 포인트

이 문제는 C 언어 2차원 배열의 초기화 결과와 배열식이 포인터로 변환될 때의 의미 및 자료형을 구분하는지를 묻는 문제이다.

핵심 개념

2차원 배열 `int mat[3][4]`는 3행 4열의 정수 배열이다.
초기값 `{1, 2, 3, 4, 5, 6}`은 행 우선으로 채워져 `mat[0]={1,2,3,4}`, `mat[1]={5,6,0,0}`, `mat[2]={0,0,0,0}`가 된다.
식에서 `mat`는 `int (*)[4]`로, `mat[1]`은 `int *`로 해석되며, `**mat`의 결과 자료형은 `int`이다.

NoteBase

선지별 해설

① O
`mat[2][2]`는 세 번째 행의 세 번째 원소이다. 여섯 개만 초기화되었으므로 남은 원소들은 0으로 자동 초기화되어 `mat[2][2]`는 0이다.

② O
`mat[1]`은 두 번째 행을 나타내는 배열식이고, 식으로 사용될 때 그 시작 원소 주소인 `&mat[1][0]`와 같은 값으로 변환된다. 시험 문맥에서는 이때 `int *`로 본다.

③ X
`**mat`는 `*(*mat)`이므로 첫 번째 행의 첫 번째 원소인 `mat[0][0]`와 같은 의미이다. 그러나 결과 자료형은 포인터가 아니라 `int`이다. 틀린 표현은 `자료형은 int*이다`이고, 올바르게는 `자료형은 int이다`이다.

④ O
`mat[2]`는 세 번째 행의 시작을 가리키고, `mat[2] + 3`은 그 행의 네 번째 원소를 가리킨다. 따라서 `*(mat[2] + 3)`은 `mat[2][3]`과 같고 값도 0이다.

오답 포인트

배열명과 역참조 결과를 모두 포인터라고 뭉뚱그리면 ③처럼 틀리기 쉽다. `mat`에서 한 번 역참조하면 한 행이고, 두 번 역참조하면 개별 원소가 되므로 최종 자료형이 `int`로 바뀐다는 점이 함정이다.

다시 나오면 이렇게 판단

2차원 배열 식은 한 단계씩 벗겨서 해석하면 된다. `mat`는 행을 가리키고, `*mat`는 첫 행, `**mat`는 첫 원소이다. 원소까지 내려간 식의 자료형은 그 원소형인 `int`라고 판단한다.

암기 포인트

`mat`는 행 포인터, `mat[1]`은 행의 첫 원소 포인터, `**mat`는 포인터가 아니라 첫 원소 값이다.

NoteBase

문 3. 함수 func1(), func2(), func3(), func4()의 시간 복잡도를 빅오(Big-O) 표기법으로 표현한 것으로서 옳지 않은 것은?

①

```
void func1(int n)
{
   int i, j;
   for (i=0; i<n; i++)
      for (j=i; j<n; j++)
         printf("%d %d\n", i, j);
}
```

$O(n^2)$

②

```
void func2(int n)
{
   int i;
   for (i=1; i<=n; i=i*2)
        printf("%d\n", i);
}
```

$O(\log_2 n)$

③

```
int func3(int n)
{
   if (n<=1) return 1;
   else return (n*func3(n-1));
}
```

$O(n)$

④

```
int func4(int n)
{
   int k=0;
   while (n>2) {
      n=n/2; k++;
   }
   return k;
}
```

$O(n \cdot \log_2 n)$

정답 4번

정답

정답은 4번이다.

출제 포인트

반복문과 재귀호출의 실행 횟수를 합산하거나 점화식으로 바꾸어 각 함수의 Big-O를 판정하는 문제이다.

조건 정리

선지 ①은 이중 반복문의 총 반복 횟수를 묻는다.
선지 ②는 반복할 때마다 값이 2배가 되는 경우의 횟수를 묻는다.
선지 ③은 한 번에 하나씩 줄어드는 재귀호출의 깊이를 묻는다.
선지 ④는 매 반복마다 n이 절반으로 줄어드는 while문의 횟수를 묻는다.

사용할 공식

이 문제는 반복 횟수 합과 재귀 점화식을 사용한다.

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

$$T(n) = T(n-1) + O(1) \Rightarrow O(n)$$

$$T(n) = T(n/2) + O(1) \Rightarrow O(\log n)$$

값이 1씩 줄면 선형, 2배 또는 1/2로 변하면 로그로 판단할 수 있다.

NoteBase

계산 과정

1. func1은 바깥 반복이 n번, 안쪽 반복이 $n, n-1, \dots, 1$번 실행되어 전체가 $\frac{n(n+1)}{2}$번이므로 $O(n^2)$이다.
2. func2는 $i = 1, 2, 4, 8, \dots$처럼 2배씩 증가하므로 반복 횟수는 $2^k \le n$을 만족하는 k의 개수와 같아 $O(\log n)$이다.
3. func3은 호출할 때마다 $n \to n-1$로 1씩 줄고 각 단계의 추가 연산은 상수이므로 깊이가 n인 재귀여서 $O(n)$이다.
4. func4는 while 한 번마다 $n \to n/2$가 되어 절반씩 감소하므로 반복 횟수는 $n/2^k \le 2$를 만족하는 k에 비례하여 $O(\log n)$이다.

선지별 해설

① O: 안쪽 반복의 총합이 $n + (n-1) + \cdots + 1 = \frac{n(n+1)}{2}$이므로 $O(n^2)$가 맞다.
② O: 제어변수 i가 2배씩 증가하므로 반복 횟수는 로그 개수이고 $O(\log n)$이 맞다.
③ O: 재귀가 $n, n-1, n-2, \dots, 1$로 한 단계씩 내려가므로 총 호출 수가 n에 비례하여 $O(n)$이 맞다.
④ X: 틀린 표현은 `func4의 시간 복잡도는 O(n log n)이다`라는 부분이다. 올바르게는 while 반복이 n을 절반씩 줄이므로 반복 횟수는 로그 개수이고 시간 복잡도는 $O(\log n)$이다.

오답 포인트

④에서 `while 안에 n이 있으니 O(n log n)일 것`이라고 섞어 보는 오류가 가장 흔하다. 실제로는 루프 몸체 안에 n번 도는 내부 반복이 없고, 단순히 상태값 n이 절반으로 줄어들 뿐이므로 로그 시간이다.

다시 나오면 이렇게 판단

코드의 시간 복잡도는 `한 번 실행되는 몸체의 비용`과 `그 몸체가 몇 번 반복되는지`를 분리해서 본다. 1씩 줄거나 늘면 선형, 누적합이면 제곱형이 될 수 있고, 2배 증가나 절반 감소가 보이면 먼저 로그를 의심한다.

암기 포인트

반으로 줄면 로그, 한 칸씩 줄면 선형, 누적합 $1 + 2 + \cdots + n$은 제곱형이다.

NoteBase

문 4. 배열로 구현한 최대 히프(max heap)에 숫자 3, 4, 5, 6, 7, 8, 9를 순서대로 삽입하였다. 이 최대 히프에서 루트에 존재하는 값을 삭제하는 연산을 2회 수행하였을 때, 배열의 마지막에 위치한 숫자는?

① 3

② 4

③ 5

④ 6

정답 3번

정답

정답은 3번이다.

출제 포인트

최대 힙에서 삽입 시 상향 이동과 루트 삭제 시 마지막 원소 이동 후 하향 이동을 배열 상태로 끝까지 추적하는 문제이다.

올바른 순서

숫자 3, 4, 5, 6, 7, 8, 9를 차례로 삽입하면 최대 힙 배열은 [9, 6, 8, 3, 5, 4, 7]이 된다.
1회 삭제에서는 루트 9를 제거하고 마지막 원소 7을 루트로 올린 뒤 더 큰 자식 8과 교환하므로 [8, 6, 7, 3, 5, 4]가 된다.
2회 삭제에서는 루트 8을 제거하고 마지막 원소 4를 루트로 올린 뒤 더 큰 자식 7과 교환하므로 [7, 6, 4, 3, 5]가 된다.
따라서 배열의 마지막 원소는 5이다.
왜 이 순서인가: 힙 삭제는 마지막 원소를 루트에 놓고 부모-자식 대소관계가 회복될 때까지 아래로 내려 보내는 절차로 결정된다.

선지별 해설

① X: 두 번째 삭제 뒤 배열은 [7, 6, 4, 3, 5]이므로 3은 네 번째 원소이다. 틀린 선택값은 3이며, 올바르게는 마지막 원소 5를 골라야 한다.
② X: 두 번째 삭제 뒤 4는 세 번째 원소이다. 틀린 선택값은 4이며, 올바르게는 마지막 원소 5를 골라야 한다.
③ O: 두 번째 삭제 뒤 배열 [7, 6, 4, 3, 5]의 끝 원소가 5이므로 옳다.
④ X: 두 번째 삭제 뒤 6은 두 번째 원소이다. 틀린 선택값은 6이며, 올바르게는 마지막 원소 5를 골라야 한다.

NoteBase

오답 포인트

삽입이 끝난 직후 배열 [9, 6, 8, 3, 5, 4, 7]의 마지막 원소 7에서 멈추거나, 삭제를 하면 루트만 바뀐다고 생각하는 실수가 잦다. 실제로는 삭제마다 마지막 원소를 루트로 올린 뒤 하향 이동이 일어나므로 끝 원소도 다시 확인해야 한다.

다시 나오면 이렇게 판단

힙 배열 문제는 삽입이면 부모보다 클 때까지 위로 올리고, 삭제이면 마지막 원소를 루트로 올린 뒤 더 큰 자식과 바꾸며 아래로 내리는 규칙만 단계별로 적으면 된다. 각 연산 뒤 배열을 한 줄씩 남기면 마지막 위치 판단이 흔들리지 않는다.

암기 포인트

최대 힙 삭제는 마지막 원소를 루트로 올린 뒤 큰 자식 쪽으로 하향 이동하는 연산이다.

NoteBase

문 5. 5개의 버킷(bucket)이 있고, 버킷당 2개의 슬롯(slot)을 가지는 해시 테이블과 여기에 해싱을 위한 해시 함수 hash_function()이 <보기 2>에 있다. 버킷 번호는 0, 1, 2, 3, 4로 설정하며, 해시 함수 hash_function()은 탐색키가 문자열인 경우에 해시값을 반환한다. 문자열 데이터가 <보기 1> 순서대로 해싱 과정이 수행될 때, 처음으로 오버플로가 발생하는 버킷 번호는?

<보기 1>

“array”, “data”, “graph”, “heap”, “list”, “buffer”, “circle”, “bit”

<보기 2>

```
int hash_function(char key[])
{
    int diff=key[0]-'a';
    int hash=diff%5;
    return hash;
}
```

① 0

② 1

③ 2

④ 3

정답 2번

정답

정답은 2번이다.

출제 포인트

이 문제는 해시 함수 $hash = (key[0] - 'a')\%5$로 각 문자열의 버킷 번호를 구한 뒤, 버킷별 슬롯 수를 순서대로 누적해 처음 오버플로가 발생하는 시점을 찾는 문제이다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 자료구조론 · 2019 · 10월

조건 정리

주어진 값:
- 버킷은 5개이고 번호는 0, 1, 2, 3, 4이다.
- 각 버킷의 슬롯 수는 2개이다.
- 해시 함수는 문자열 첫 글자에 대해 $key[0] - 'a'$를 구한 뒤 5로 나눈 나머지를 반환한다.

구해야 할 값:
- <보기 1>의 문자열을 순서대로 넣을 때 처음 오버플로가 나는 버킷 번호이다.

주의할 조건:
- 오버플로는 슬롯 2개가 찬 순간이 아니라 3번째 원소가 같은 버킷에 들어가려는 순간 발생한다.

사용할 공식

공식은 $hash = (key[0] - 'a')\%5$이다.
이 공식을 쓰는 이유:
문제의 C 함수가 첫 글자의 알파벳 위치 차이를 구해 5로 나눈 나머지를 그대로 버킷 번호로 반환하기 때문이다.

계산 과정

1. array는 첫 글자 a이므로 $(a - a)\%5 = 0$이어서 0번 버킷에 들어간다.
2. data는 d이므로 $(d - a)\%5 = 3$, graph는 g이므로 $6\%5 = 1$, heap은 h이므로 $7\%5 = 2$이다.
3. list는 l이므로 $11\%5 = 1$이어서 1번 버킷에 두 번째로 들어간다. 이 시점의 1번 버킷은 graph, list로 가득 찬 상태이다.
4. buffer는 b이므로 $1\%5 = 1$이어서 1번 버킷에 세 번째로 들어가려 한다. 1번 버킷은 슬롯 2개를 이미 모두 사용했으므로 여기서 처음 오버플로가 발생한다.
5. 이후 circle은 2번 버킷, bit는 1번 버킷으로 가지만, 최초 오버플로는 이미 buffer에서 결정되었다.

선지별 해설

① X: 0번 버킷에는 array만 들어가므로 원소 수가 1개이다. 틀린 표현은 '처음 오버플로가 0번 버킷에서 난다'는 판단이다. 올바르게는 0번 버킷은 끝까지 여유가 있고, 처음 오버플로는 1번 버킷에서 발생한다.
② O: 1번 버킷에는 graph, list, buffer가 순서대로 몰린다. 슬롯이 2개뿐이므로 세 번째 원소인 buffer가 들어갈 때 처음 오버플로가 발생한다.
③ X: 2번 버킷에는 heap과 circle이 들어가서 정확히 2개만 찬다. 틀린 표현은 '2번 버킷이 처음 넘친다'는 판단이다. 올바르게는 2번 버킷은 가득 찰 뿐이고, 최초 오버플로 버킷은 1번이다.
④ X: 3번 버킷에는 data만 들어가므로 오버플로가 날 수 없다. 틀린 표현은 '3번 버킷이 처음 넘친다'는 판단이다. 올바르게는 3번 버킷은 원소 1개 상태로 끝나고, 처음 오버플로는 1번 버킷이다.

NoteBase

오답 포인트

가장 흔한 실수는 두 가지이다. 첫째, 알파벳 위치를 1부터 세어 해시값을 잘못 구하는 실수이다. 이 함수는 a를 0으로 본다. 둘째, 슬롯 2개가 꽉 찬 순간을 오버플로로 착각하는 실수이다. 가득 참은 정상 상태이고, 3번째 삽입 시도가 오버플로이다.

다시 나오면 이렇게 판단

문자열 해시 충돌 문제는 첫 글자 기준 해시값을 모두 적은 뒤 버킷별 누적 개수를 순서대로 센다. 버킷의 슬롯 수가 2개이면 1개와 2개는 정상이고, 3개째가 들어가는 순간이 최초 오버플로이다.

암기 포인트

해시 충돌 문제는 '해시값 계산 후 버킷별 누적 개수 확인'으로 정리되고, 슬롯 수보다 1개 더 들어갈 때 오버플로가 발생한다.

NoteBase

문 6. 그래프(graph)와 관련된 설명 중 가장 옳지 않은 것은?

① 무방향 그래프(undirected graph)를 인접 행렬로 표현하면 대칭행렬이 된다.

② 15개의 정점(vertex), 20개의 간선(edge)으로 이루어진 무방향 그래프를 인접 리스트(adjacency list)로 표현할 때, 인접 리스트에 존재하는 노드의 총 개수는 30이다.

③ 20개의 정점으로 구성된 무방향 완전 그래프(undirected complete graph)에 존재하는 간선의 총 개수는 190이다.

④ 100개의 정점으로 구성된 그래프에 대한 신장 트리(spanning tree)는 99개의 간선으로 연결된다.

정답 2번

정답

정답은 2번이다.

출제 포인트

이 문제는 무방향 그래프의 표현 방식과 완전그래프, 신장트리의 기본 성질을 정확히 구분하는지를 묻는 문제이다.

핵심 개념

무방향 그래프의 인접 행렬은 간선이 양방향으로 동일하므로 대칭행렬이 된다.
무방향 그래프의 인접 리스트에서는 간선 1개가 양 끝 정점의 리스트에 각각 1번씩 기록되어 총 2개의 리스트 노드를 만든다.
정점이 n개인 무방향 완전그래프의 간선 수는 $n(n-1)/2$이고, 정점이 n개인 신장트리의 간선 수는 $n-1$이다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 자료구조론 · 2019 · 10월

선지별 해설

① O

무방향 그래프를 인접 행렬로 표현하면 $A[i][j] = A[j][i]$가 성립하므로 대칭행렬이 된다.

② X

틀린 부분은 "30"이다.

무방향 그래프의 간선 20개는 인접 리스트에서 양 끝 정점 쪽에 각각 1번씩 나타나므로 리스트 노드 수는 $2 \times 20 = 40$개이다.

올바르게는 "인접 리스트에 존재하는 노드의 총 개수는 40이다"가 된다.

③ O

정점이 20개인 무방향 완전그래프의 간선 수는 $20 \times 19/2 = 190$이므로 맞는 설명이다.

④ O

트리는 정점이 n개이면 간선이 항상 $n - 1$개이다.

따라서 100개의 정점을 포함하는 신장트리는 99개의 간선으로 연결된다.

오답 포인트

인접 리스트의 노드 수를 정점 수와 혼동하거나, 무방향 간선이 리스트에 두 번 기록된다는 점을 놓치면 ②를 맞는 설명으로 오해하기 쉽다.

다시 나오면 이렇게 판단

그래프 표현 문제가 나오면 먼저 표현 방식별 계산 기준을 분리해서 본다. 인접 행렬은 대칭 여부를, 인접 리스트는 무방향 간선 1개가 2번 저장된다는 점을, 트리는 간선 수가 $n - 1$이라는 점을 바로 대입하면 된다.

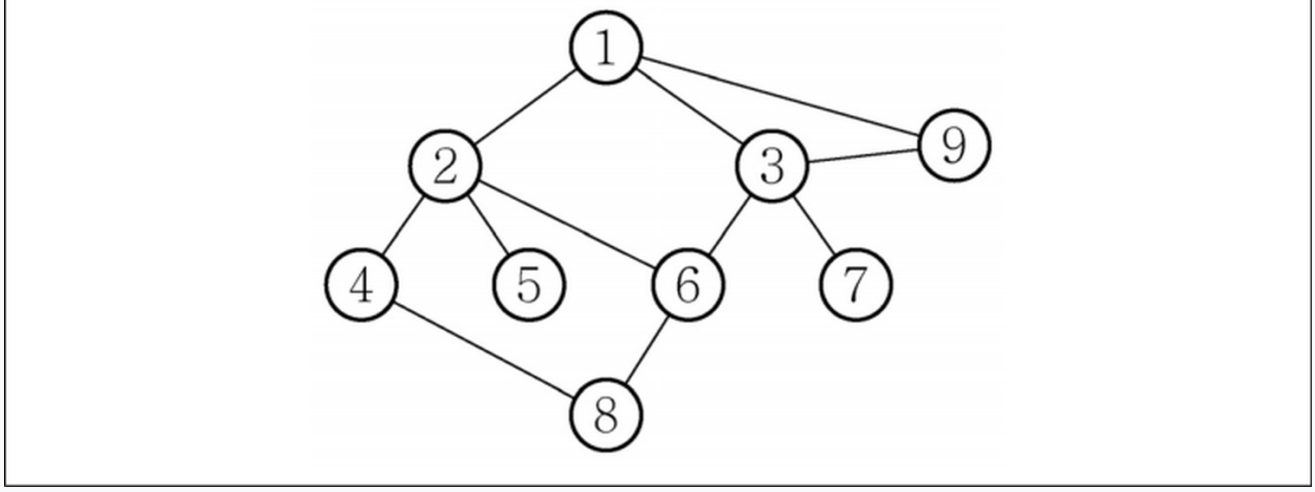
암기 포인트

무방향 인접 리스트는 간선 수의 2배, 완전그래프는 $n(n - 1)/2$, 트리는 $n - 1$이다.

NoteBase

문 7. <보기>의 그래프에서 깊이 우선 탐색(Depth First Search) 결과를 나타낸 것들 중 가장 옳지 않은 것은?

<보기>



① 1 - 2 - 5 - 6 - 3 - 7 - 9 - 8 - 4

② 1 - 2 - 5 - 6 - 8 - 4 - 3 - 7 - 9

③ 1 - 3 - 9 - 7 - 6 - 2 - 4 - 8 - 5

④ 1 - 3 - 9 - 6 - 2 - 5 - 8 - 4 - 7

정답 4번

정답

정답은 4번이다.

출제 포인트

이 문제는 그래프의 인접 관계를 보면서 깊이 우선 탐색의 방문 순서가 가능한지 판단하는 문제이다. DFS는 한 정점에 들어가면 그 정점의 미방문 인접 정점을 모두 처리한 뒤에야 상위 정점으로 되돌아간다는 점이 핵심이다.

NoteBase

선지별 해설

① O: 1→2→5까지 내려간 뒤 2로 되돌아와 6으로 가고, 다시 3→7→9를 처리한 다음 6→8→4로 마무리하면 되는 순서이다. 각 이동이 현재 정점의 인접 정점을 따라 끝까지 내려가는 DFS의 전형적인 방문과 맞는다.
② O: 1→2→5 후에 6으로 이동하고, 6에서 8→4를 먼저 처리한 뒤 3으로 돌아가 7→9를 방문하면 되는 순서이다. 8은 6의 인접 정점이므로 6 다음에 8이 오는 구간이 자연스럽다.
③ O: 1→3→9 후 3으로 되돌아와 7을 방문하고, 다시 6→2→4→8→5로 내려가면 되는 순서이다. 특히 4와 8은 직접 연결되어 있으므로 2의 하위 탐색에서 4 다음에 8이 나오는 흐름이 DFS와 맞는다.
④ X: 잘못된 부분은 6→2→5 다음에 바로 8이 나오는 구간이다. 2에서 5를 방문한 뒤에는 아직 4가 미방문 인접 정점으로 남아 있으므로 2의 탐색을 끝내기 전에는 6으로 되돌아갈 수 없다. 또한 8은 2의 인접 정점이 아니라 4 또는 6을 통해서만 방문된다. 틀린 표현은 2-5-8-4이고, 올바르게는 2-5-4-8 또는 2-4-8-5처럼 4가 8보다 먼저 나와야 한다.

오답 포인트

④의 2-5-8-4를 보면 많은 수험생이 DFS를 단순히 연결된 정점을 아무 순서로 방문하는 방식으로 오해한다. 그러나 DFS는 현재 들어간 정점의 미방문 인접 정점을 모두 소진하기 전에는 상위 정점으로 빠져나갈 수 없으므로, 2의 하위 정점인 4를 남겨 둔 채 8로 넘어가는 판단이 함정이다.

다시 나오면 이렇게 판단

방문 순서가 주어지면 각 구간을 현재 DFS 스택 기준으로 읽으면 된다. 어떤 정점에 들어간 뒤 그 정점의 미방문 인접 정점이 남아 있는데도 다른 갈래로 빠지면 불가능한 순서이고, 비인접 정점으로의 점프가 보이면 바로 배제하면 된다.

암기 포인트

DFS는 한 번 들어간 정점의 미방문 인접 정점을 모두 소진한 뒤에만 되돌아간다. 하위 정점을 남긴 채 다른 갈래로 점프하는 순서는 성립하지 않는다.

NoteBase

문 8. <보기>의 이진 트리에 대한 함수 traverseA와 traverseB의 운행 방식을 가장 옳게 나타낸 것은?

<보기>

```
struct NODE{
    char name[10];
    struct NODE *left;
    struct NODE *right;
};
typedef struct NODE Node;
```

```
void traverseA(Node *x)
{
    if (x!=NULL){
        traverseA(x->left);
        printf("%s\n", x->name);
        traverseA(x->right);
    }
}

void traverseB(Node *x)
{
    if (x!=NULL){
        traverseB(x->left);
        traverseB(x->right);
        printf("%s\n", x->name);
    }
}
```

① traverseA: 중위 순회, traverseB: 후위 순회

② traverseA: 후위 순회, traverseB: 중위 순회

③ traverseA: 중위 순회, traverseB: 전위 순회

④ traverseA: 전위 순회, traverseB: 후위 순회

정답 1번

정답

정답은 1번이다.

출제 포인트

이 문제는 재귀 호출과 출력문의 위치를 보고 이진 트리의 전위 순회, 중위 순회, 후위 순회를 구별하는 문제이다.

핵심 개념

전위 순회는 노드를 먼저 방문한 뒤 왼쪽, 오른쪽으로 가는 방식으로 순서가 노드-왼쪽-오른쪽이다.
중위 순회는 왼쪽 서브트리 방문 뒤 노드를 방문하고 마지막에 오른쪽 서브트리를 방문하는 방식으로 순서가 왼쪽-노드-오른쪽이다.
후위 순회는 왼쪽, 오른쪽 서브트리를 모두 방문한 뒤 마지막에 노드를 방문하는 방식으로 순서가 왼쪽-오른쪽-노드이다.

NoteBase

선지별 해설

① O: traverseA는 `traverseA(x->left)` 다음에 `printf`, 그다음 `traverseA(x->right)`를 수행하므로 순서가 왼쪽-노드-오른쪽이어서 중위 순회이다. traverseB는 왼쪽 재귀, 오른쪽 재귀를 모두 마친 뒤 `printf`를 수행하므로 왼쪽-오른쪽-노드 순서인 후위 순회이다.
② X: 틀린 부분은 traverseA를 후위 순회, traverseB를 중위 순회로 본 점이다. 실제로는 traverseA가 중위 순회이고 traverseB가 후위 순회이다.
③ X: traverseA를 중위 순회로 본 점은 맞지만 traverseB를 전위 순회로 본 점이 틀리다. 전위 순회는 노드를 먼저 출력해야 하므로 실제로는 traverseB가 아니라 노드-왼쪽-오른쪽 구조일 때 성립한다. traverseB는 후위 순회이다.
④ X: traverseB를 후위 순회로 본 점은 맞지만 traverseA를 전위 순회로 본 점이 틀리다. 전위 순회는 `printf`가 두 재귀 호출보다 먼저 나와야 하며, 실제로는 traverseA가 아니라 중위 순회이다.

오답 포인트

재귀 함수 문제에서는 함수 이름보다 `printf`의 위치가 판단 기준이다. ③과 ④처럼 한쪽만 맞고 다른 한쪽을 바꿔 놓으면 헷갈리기 쉬운데, 출력이 가운데 있으면 중위, 맨 마지막이면 후위라는 위치 기준을 놓치면 오답이 된다.

다시 나오면 이렇게 판단

재귀 순회 코드를 보면 출력문이 재귀 호출들 사이의 어디에 놓였는지만 본다. 출력이 먼저면 전위, 왼쪽 호출 뒤 가운데면 중위, 양쪽 호출 뒤 마지막이면 후위로 판단된다.

암기 포인트

루트 출력 위치만 보면 된다. 앞이면 전위, 가운데면 중위, 뒤면 후위이다.

NoteBase

문 9. 레드-블랙 트리(red-black tree)에 대한 설명으로 가장 옳지 않은 것은?

① 레드-블랙 트리에서 임의의 노드(node)의 두 자식트리(child tree)의 높이(height) 차이는 1 이하이다.

② 레드-블랙 트리에 내부 노드가 n개 있으면, 높이는 최대 $2\log_2(n+1)$이다.

③ 레드-블랙 트리의 루트 노드는 흑색(black) 노드이다.

④ 레드-블랙 트리에서 적색(red) 노드의 모든 자식 노드는 흑색 노드이다.

정답 1번

정답

정답은 1번이다.

출제 포인트

이 문제는 레드-블랙 트리의 균형 조건과 기본 색 규칙을 AVL 트리의 높이 균형 조건과 구별하는지를 묻는 문제이다.

핵심 개념

레드-블랙 트리: 색 규칙을 이용해 전체 높이를 대략 $O(\log n)$으로 유지하는 균형 이진 탐색트리이다. 결정 차이축은 각 노드의 국소 높이 차이가 아니라 black-height와 색 제약이다. 키워드는 루트 흑색, 적색 노드의 자식 흑색, 경로별 동일 black-height이다.
AVL 트리: 임의의 노드에서 왼쪽과 오른쪽 부분트리의 높이 차이를 1 이하로 제한하는 엄격한 높이 균형 이진 탐색트리이다. 결정 차이축은 각 노드의 두 자식트리 높이 차이이다. 키워드는 높이 차이 1 이하이다.

NoteBase

선지별 해설

① X
레드-블랙 트리는 모든 노드에서 두 자식트리의 높이 차이가 1 이하라고 제한하지 않는다. 이는 AVL 트리의 균형 조건에 해당한다. 틀린 표현은 "높이 차이는 1 이하"이다. 올바르게는 레드-블랙 트리는 경로상의 흑색 노드 수를 맞추는 방식으로 균형을 유지하며, 자식트리 높이 차이가 항상 1 이하일 필요는 없다.

② O
레드-블랙 트리는 내부 노드가 n개일 때 높이가 최대 $2\log_2(n+1)$ 이하로 제한된다. black-height 제약 때문에 최악의 경우에도 높이가 로그 범위를 벗어나지 않는다.

③ O
레드-블랙 트리의 기본 성질 중 하나가 루트는 흑색이라는 점이다. 삽입이나 삭제 뒤 재조정이 끝난 상태의 루트는 흑색 노드로 본다.

④ O
레드 노드의 자식은 모두 흑색이어야 한다. 적색 노드 아래에 다시 적색 노드가 오면 연속된 red-red 위반이 발생하므로 레드-블랙 트리의 성질을 깨뜨린다.

오답 포인트

①은 균형 이진 탐색트리라는 공통점 때문에 AVL 트리의 조건을 레드-블랙 트리에 섞어 놓은 선지이다. 레드-블랙 트리는 "높이 차이 1 이하"로 외우는 구조가 아니라 "색 규칙과 black-height 유지"로 외우는 구조라는 점이 핵심 함정이다.

다시 나오면 이렇게 판단

선지에 "임의의 노드의 두 자식트리 높이 차이"가 나오면 먼저 AVL 조건인지 의심한다. 반대로 루트 흑색, 적색 노드의 자식 흑색, 경로별 흑색 수 유지, 높이의 로그 상한이 나오면 레드-블랙 트리의 성질로 판단할 수 있다.

암기 포인트

AVL은 높이 차이 1 이하, 레드-블랙은 색 규칙과 동일 black-height이다.

NoteBase

문 10. 서로 다른 5개의 값을 저장하는 이진 탐색 트리로 구성 가능한 개수는?

① 14개

② 21개

③ 42개

④ 128개

정답 3번

정답

정답은 3번이다.

출제 포인트

서로 다른 n개의 값을 저장하는 이진 탐색 트리의 개수가 카탈란 수 C_n로 계산된다는 점을 묻는 문제이다.

조건 정리

주어진 값: 서로 다른 값 5개
구해야 할 값: 구성 가능한 이진 탐색 트리의 개수
주의할 조건: 값이 서로 다르면 각 값의 상대적 대소관계만 중요하며, 가능한 BST 개수는 카탈란 수로 센다.

사용할 공식

가능한 이진 탐색 트리의 개수는 카탈란 수 C_n이며, $C_n = \frac{1}{n+1}\binom{2n}{n}$이다.
이 공식을 쓰는 이유: 서로 다른 n개의 키로 만들 수 있는 BST는 루트를 하나 정하면 왼쪽 부분트리와 오른쪽 부분트리도 다시 같은 문제로 분해되므로 카탈란 수 점화식을 따른다.

계산 과정

1. 서로 다른 5개의 값을 저장하는 BST 개수는 C_5로 계산한다.
2. $C_5 = \frac{1}{6}\binom{10}{5}$이다.
3. $C_5 = \frac{1}{6} \times 252 = 42$

따라서 구성 가능한 이진 탐색 트리의 개수는 42개이다.

NoteBase

선지별 해설

① X
'14개'가 틀렸고, 올바르게는 '42개'이다. 14는 서로 다른 4개의 값을 저장할 때의 카탈란 수 C_4에 해당한다. 이 문항은 값이 5개이므로 한 단계 더 큰 C_5를 써야 한다.

② X
'21개'가 틀렸고, 올바르게는 '42개'이다. 21은 좌우 대칭인 경우를 한 번만 세거나 경우의 수를 반만 계산했을 때 나오는 값이다. BST의 좌우가 바뀌면 다른 트리이므로 모두 세면 42개이다.

③ O
'42개'가 맞다. 서로 다른 5개의 값으로 만들 수 있는 BST 개수는 카탈란 수 C_5이며, 계산 결과가 정확히 42이다.

④ X
'128개'가 틀렸고, 올바르게는 '42개'이다. 128은 각 노드가 독립적으로 왼쪽이나 오른쪽을 선택한다고 보고 2^K 식으로 세는 오산에 가깝다. BST는 대소관계 제약 때문에 독립 선택 구조가 아니므로 카탈란 수로 계산해야 한다.

오답 포인트

BST 개수를 일반적인 이진트리 경우의 수처럼 세거나, 바로 앞 단계 카탈란 수 $C_4 = 14$를 대입하는 실수가 잦다. 특히 좌우 대칭을 같은 트리로 오인하면 21처럼 반쪽 값이 나온다.

다시 나오면 이렇게 판단

서로 다른 값 n개로 만들 수 있는 BST 개수를 묻는 문항은 먼저 카탈란 수 문제인지 확인하고, $C_n = \frac{1}{n+1}\binom{2n}{n}$ 또는 점화식으로 바로 계산하면 된다.

암기 포인트

서로 다른 n개의 값으로 만드는 BST 개수는 카탈란 수 C_n이다.

NoteBase

문 11. 설명 중 가장 옳지 않은 것은?

① 자료 구조는 자료 객체의 집합뿐 아니라 그들의 관계, 즉 자료 객체의 원소에 합법적으로 적용될 연산들까지 기술하는 것이다.

② 시간 복잡도(time complexity)를 측정하는 방법으로 알고리즘의 기본 연산 수행 횟수를 세는 방법이 있다.

③ 알고리즘이 수행될 때 소요되는 메모리 양을 분석한 것을 공간 복잡도(space complexity)라고 한다.

④ 빅오(big-O) 표기법은 시간 복잡도를 나타내고, 오메가(omega, Ω) 표기법은 공간 복잡도를 나타낸다.

정답 4번

정답

정답은 4번이다.

출제 포인트

이 문제는 자료구조의 정의, 시간 복잡도와 공간 복잡도의 의미, 점근 표기법의 역할을 구별하는 문제이다.

핵심 개념

자료구조는 자료 원소의 집합만이 아니라 원소 사이의 관계와 그 자료에 적용할 수 있는 연산까지 포함해 표현하는 개념이다.
시간 복잡도는 입력 크기에 따라 연산 수행 횟수가 어떻게 증가하는지를 나타내는 개념이다.
공간 복잡도는 알고리즘 실행에 필요한 메모리 사용량을 나타내는 개념이다.
빅오 표기와 오메가 표기는 시간과 공간을 나누는 표기가 아니라 어떤 복잡도 함수의 상한과 하한을 나타내는 점근 표기이다.

선지별 해설

① O: 자료구조는 데이터의 모임만 뜻하지 않고 데이터 사이의 관계와 허용되는 연산까지 함께 다루므로 옳은 설명이다.
② O: 시간 복잡도는 보통 기본 연산이 몇 번 수행되는지를 세어 측정하므로 옳은 설명이다.
③ O: 알고리즘 수행 중 필요한 기억 공간의 양을 분석한 것이 공간 복잡도이므로 옳은 설명이다.
④ X: 틀린 부분은 '빅오 표기법은 시간 복잡도를 나타내고, 오메가 표기법은 공간 복잡도를 나타낸다'는 표현이다. 실제로는 빅오 표기법은 어떤 복잡도의 상한, 오메가 표기법은 어떤 복잡도의 하한을 나타내며 둘 다 시간 복잡도와 공간 복잡도 모두에 사용할 수 있다.

오답 포인트

4번은 시간 복잡도와 공간 복잡도라는 대상과 빅오·오메가라는 표기 방법을 서로 대응시키도록 섞어 놓은 함정이다. 수험생이 '빅오'만 익숙하게 외우고 표기법의 의미를 상한·하한으로 정리하지 않으면 잘 속기 쉽다.

NoteBase

다시 나오면 이렇게 판단

복잡도 문제에서는 먼저 무엇을 재는지부터 구분한다. 연산 횟수이면 시간 복잡도, 메모리 사용량이면 공간 복잡도이고, 그다음 빅오·오메가·세타는 그 복잡도를 어떤 경계로 표현하는 표기인지 따로 판단한다.

암기 포인트

시간과 공간은 측정 대상이고, 빅오와 오메가는 표현 방식이다.

NoteBase

문 12. 스택(stack)과 큐(queue)에 대한 설명으로 가장 옳지 않은 것은?

① 스택은 제일 먼저 삽입된 원소가 제일 나중에 삭제된다.

② 큐의 응용 분야로 작업 스케줄링(job scheduling)이 적합하다.

③ 스택의 응용 분야 중 하나는 서브루틴 호출(subroutine call)이다.

④ 스택의 추상 데이터 타입은 1차원 배열로 구현할 수 있으나, 큐의 경우는 불가능하다.

정답 4번

정답

정답은 4번이다.

출제 포인트

이 문제는 스택과 큐의 동작 원리와 대표 구현 방법을 구별할 수 있는지를 묻는다.

핵심 개념

스택은 나중에 넣은 원소가 먼저 나오는 LIFO 구조이다.
큐는 먼저 넣은 원소가 먼저 나오는 FIFO 구조이다.
스택과 큐는 모두 추상 데이터 타입이므로 배열이나 연결 리스트처럼 여러 방식으로 구현할 수 있다.

선지별 해설

1번 O: 스택은 LIFO 구조이므로 가장 먼저 삽입된 원소는 가장 나중에 삭제된다. 선지의 설명이 스택의 기본 성질과 일치한다.
2번 O: 큐는 먼저 들어온 작업을 먼저 처리하는 FIFO 구조이므로 작업 스케줄링, 대기열 처리 같은 응용에 적합하다.
3번 O: 서브루틴 호출과 함수 호출의 복귀 순서는 가장 나중에 호출한 것이 먼저 끝나는 형태이므로 호출 스택을 사용한다.
4번 X: 틀린 부분은 '큐의 경우는 불가능하다'는 표현이다. 큐도 1차원 배열로 구현할 수 있으며, 보통 front와 rear를 두고 선형 큐나 원형 큐 형태로 구현한다.

오답 포인트

4번은 자료구조의 논리적 성질과 물리적 구현 방법을 섞어 놓은 함정이다. 스택은 배열로 구현된다는 사실만 기억하고 큐는 연결 리스트만 가능하다고 착각하면 틀리기 쉽다.

NoteBase

다시 나오면 이렇게 판단

선지가 동작 원리를 묻는지, 구현 가능 여부를 묻는지 먼저 구분한다. 순서 성질은 스택=LIFO, 큐=FIFO로 판단하고, 구현 가능 여부는 둘 다 배열과 연결 리스트로 구현 가능하다고 정리하면 된다.

암기 포인트

스택은 LIFO, 큐는 FIFO, 둘 다 배열과 연결 리스트로 구현 가능하다.

NoteBase

문 13. 원형 연결 리스트(circular linked list)의 각 노드는 DATA 필드와 LINK 필드로 구성된다. DATA 필드는 데이터 값을 저장하고, LINK 필드는 포인터(pointer) 값을 저장한다. 첫 번째 노드를 가리키는 포인터를 head, 마지막 노드를 가리키는 포인터를 tail이라고 하면, 이 리스트의 첫 번째 노드의 주소를 가리키는 값은?

① tail → LINK

② tail → DATA

③ tail → LINK → LINK

④ tail → LINK → DATA

정답 1번

정답

정답은 1번이다.

출제 포인트

이 문제는 원형 연결 리스트에서 tail 포인터와 마지막 노드의 LINK 필드가 무엇을 가리키는지를 묻는 문제이다.

핵심 개념

원형 연결 리스트는 마지막 노드의 LINK가 다시 첫 번째 노드를 가리키는 연결 리스트이다.
head는 첫 번째 노드를 가리키는 포인터이고, tail은 마지막 노드를 가리키는 포인터이다.
따라서 첫 번째 노드의 주소는 tail이 직접 갖는 것이 아니라 마지막 노드의 LINK 필드인 tail -> LINK에 들어 있다.

NoteBase

선지별 해설

① O
마지막 노드를 가리키는 포인터가 tail이므로, 그 노드의 LINK 필드는 원형 구조상 첫 번째 노드를 가리킨다.
따라서 첫 번째 노드의 주소는 tail -> LINK이다.

② X
틀린 부분은 DATA이다.
DATA 필드는 노드의 데이터 값을 저장하는 공간이지 다음 노드의 주소를 저장하는 공간이 아니다. 올바르게는 첫 번째 노드의 주소가 tail -> LINK에 저장된다.

③ X
틀린 부분은 LINK -> LINK이다.
tail -> LINK는 이미 첫 번째 노드의 주소이므로, 한 번 더 LINK를 따라가면 두 번째 노드의 주소가 된다.
올바르게는 첫 번째 노드의 주소가 tail -> LINK이다.

④ X
틀린 부분은 LINK -> DATA이다.
tail -> LINK는 첫 번째 노드를 가리키고, 그 뒤의 DATA는 첫 번째 노드에 저장된 데이터 값이다. 주소가 아니라 데이터이므로 올바르게는 첫 번째 노드의 주소가 tail -> LINK이다.

오답 포인트

DATA 필드와 LINK 필드의 역할을 뒤섞으면 ②나 ④로 흔들리기 쉽다. 또 tail이 마지막 노드를 가리킨다는 사실은 알지만, 원형 구조에서는 마지막 노드의 LINK가 첫 번째 노드로 되돌아간다는 점을 놓치면 ③처럼 한 번 더 따라간 값을 고르게 된다.

다시 나오면 이렇게 판단

연결 리스트 문제에서는 먼저 포인터가 노드 자체를 가리키는지, 노드 안의 LINK가 다음 노드를 가리키는지를 구분한다. 원형 연결 리스트라면 마지막 노드의 LINK가 첫 번째 노드로 돌아간다는 구조를 바로 적용하면 된다.

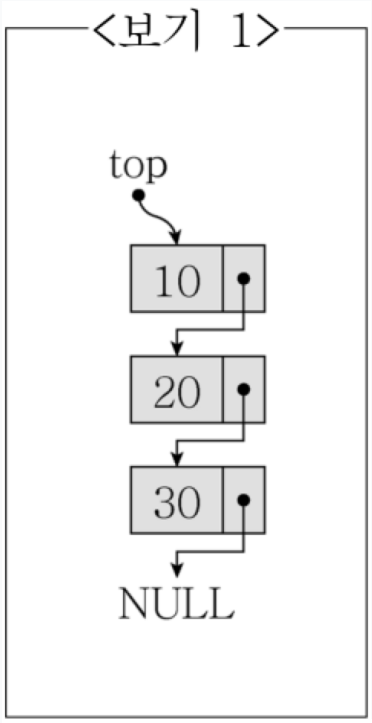
암기 포인트

원형 연결 리스트는 마지막 노드의 LINK가 첫 노드를 가리키므로 첫 노드 주소는 tail -> LINK이다.

NoteBase

문 14. <보기 1>은 연결 리스트(linked list)로 구현된 스택(stack)이고, <보기 2>는 연결 리스트로 구현된 스택으로부터 데이터 항목의 개수를 반환하는 stack_size() 함수이다. 여기서 변수 top은 스택의 최상단의 노드를 가리키는 포인터 변수이다. <보기 2>의 ㉠, ㉡에 들어갈 코드로서 옳은 것은?

<보기 1>



<보기 2>

```
typedef struct LinkedNode{
        int data;
        struct LinkedNode *link;
}Node;
int stack_size()
{
        Node *p;
        int count=0;
        for (p=top;[   ㉠   ];[   ㉡   ])
              count++;
        return count;
}
```

① ㉠: p!=NULL, ㉡: p=p->link

② ㉠: p!=p->link, ㉡: p=p->link

NoteBase

③ ㉠: p!=NULL, ㉡: p++

④ ㉠: p!=p->link, ㉡: p++

정답 1번

정답

정답은 1번이다.

출제 포인트

이 문제는 연결 리스트로 구현한 스택 그림을 보고, 노드 개수를 셀 때 반복 변수 `p`를 `top`에서 시작하여 `NULL`이 될 때까지 `link`를 따라 이동시켜야 한다는 점을 묻는 문제이다.

선지별 해설

① O

`for (p=top; p!=NULL; p=p->link)`가 되어야 현재 노드가 존재하는 동안 하나씩 세고 다음 노드로 정확히 이동한다. 마지막 노드의 `link`가 `NULL`이므로 모든 노드를 센 뒤 반복이 끝난다.

② X

조건식 `p!=p->link`는 현재 노드 주소와 다음 노드 주소를 비교하는 식이어서 순회 종료 조건이 아니다. 마지막에는 `p`가 `NULL`이 된 뒤에도 다시 `p->link`를 보게 되어 올바른 조건이 될 수 없다. 틀린 표현은 `p!=p->link`이고, 올바르게는 `p!=NULL`이다.

③ X

증감식 `p++`는 연결된 다음 노드로 가는 연산이 아니라 포인터 주소를 메모리상 다음 칸으로 증가시키는 연산이다. 연결 리스트는 연속 배열이 아니므로 `p++`로는 올바른 순회가 되지 않는다. 틀린 표현은 `p++`이고, 올바르게는 `p=p->link`이다.

④ X

조건식도 `p!=p->link`로 잘못되었고 증감식도 `p++`로 잘못되었다. 연결 리스트 순회는 `NULL`을 종료 조건으로 두고 `link`를 따라 이동해야 한다. 틀린 표현은 `p!=p->link`, `p++`이고, 올바르게는 `p!=NULL`, `p=p->link`이다.

오답 포인트

배열 순회처럼 `p++`를 넣거나, 종료 조건을 `p!=p->link`처럼 현재 노드와 다음 노드 비교로 착각하면 틀리기 쉽다. 연결 리스트는 메모리에 연속 저장된다는 보장이 없으므로 다음 원소 접근은 항상 `link`를 따라가야 한다.

NoteBase

다시 나오면 이렇게 판단

연결 리스트의 개수 세기 문제는 `시작 포인터 설정 -> NULL까지 반복 -> link로 다음 노드 이동`의 세 요소를 먼저 확인하면 된다. 즉 `for (p=head; p!=NULL; p=p->link)` 형태인지 보면 판단할 수 있다.

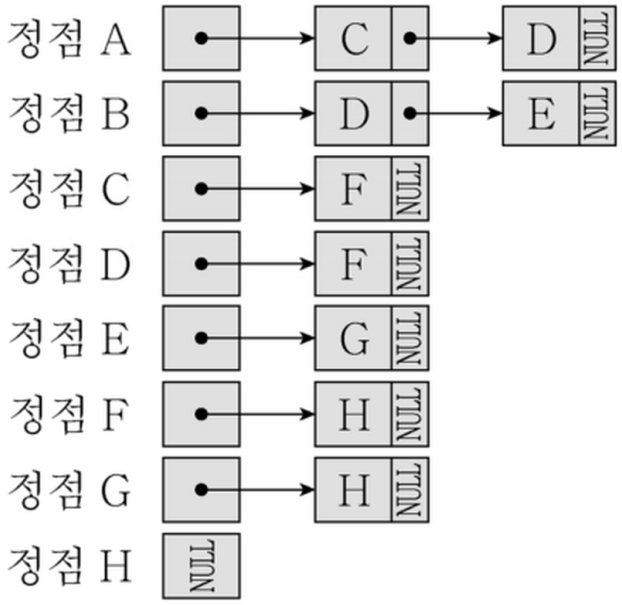
암기 포인트

연결 리스트 순회는 `NULL`까지 세고, 이동은 `p=p->link`로 한다.

NoteBase

문 15. <보기>는 인접 리스트(adjacency list)로 표현한 방향 그래프이다. 이 그래프에 대하여 위상 정렬(topological sort)을 수행한 결과로서 가장 옳지 않은 것은?

<보기>



① 위상 정렬 결과: A, B, C, D, E, F, G, H

② 위상 정렬 결과: A, C, B, E, G, D, F, H

③ 위상 정렬 결과: B, E, G, A, D, C, F, H

④ 위상 정렬 결과: B, E, G, H, D, F, A, C

정답 4번

정답

정답은 4번이다.

출제 포인트

이 문제는 인접 리스트로 표현된 방향 그래프에서 간선의 선후 관계를 읽고, 그 제약을 모두 만족하는 위상 정렬 순서인지 판단하는 문제이다. 그림에서 읽히는 간선은 A→C, A→D, B→D, B→E, C→F, D→F, E→G, F→H, G→H이다.

NoteBase

선지별 해설

① O
A가 C와 D보다 앞에 있고, B가 D와 E보다 앞에 있으며, C와 D가 F보다 앞에 있고, E와 G 및 F와 H의 선후도 모두 지켜진다. 따라서 위상 정렬 결과로 가능하다.

② O
A가 C와 D보다 앞에 있고, B가 D와 E보다 앞에 있다. 또한 E가 G보다 앞, D와 C가 F보다 앞, F와 G가 H보다 앞이므로 모든 간선 제약을 만족한다. 따라서 위상 정렬 결과로 가능하다.

③ O
B, E, G가 먼저 나와도 문제되지 않는다. 위상 정렬은 연결되지 않은 정점들의 상대적 순서를 자유롭게 둘 수 있기 때문이다. A가 C와 D보다 앞에 있고, C와 D가 F보다 앞에 있으며, F가 H보다 앞에 있으므로 이 순서도 가능하다.

④ X
B→D, B→E, E→G는 만족하지만, H가 F보다 먼저 나와 F→H를 위반한다. 또 D가 A보다 먼저 나와 A→D를 위반하고, F가 C보다 먼저 나와 C→F도 위반한다. 틀린 표현은 "B, E, G, H, D, F, A, C가 위상 정렬 결과가 된다"는 부분이다. 올바르게는 A는 D보다 앞, C와 D는 F보다 앞, F와 G는 H보다 앞에 와야 한다.

오답 포인트

A의 인접 리스트가 C 다음 D로 이어진 것을 C→D로 잘못 읽기 쉽다. 이 그림은 A에서 C와 D로 각각 간선이 나간다는 뜻이다. 또한 위상 정렬은 모든 정점이 한 줄로 예쁘게 이어질 필요가 없고, 간선으로 직접 제약된 선후만 지키면 된다는 점을 놓치면 ③을 오답으로 보기 쉽다.

다시 나오면 이렇게 판단

먼저 인접 리스트를 간선 목록으로 바꾼 뒤, 각 보기에서 모든 간선 $u \to v$에 대해 반드시 u가 v보다 앞에 있는지만 검사하면 된다. 하나라도 선후가 뒤집히면 그 순서는 위상 정렬이 아니다.

암기 포인트

위상 정렬의 기준은 전체 모양이 아니라 간선의 방향이다. 모든 간선에서 출발 정점이 도착 정점보다 앞이면 가능하다.

NoteBase

문 16. 입력 리스트를 특정한 키(key) 값보다 작은 값을 갖는 자료와 큰 값을 갖는 자료로 분리하여 한 개의 리스트를 논리적으로 두 개의 서브(sub) 리스트로 재배열한 후, 각각의 서브 리스트에 대해 순환적으로 같은 정렬 방법을 적용하여 입력 리스트를 정렬하는 방법은?

① 버블 정렬(Bubble Sort)

② 히프 정렬(Heap Sort)

③ 퀵 정렬(Quick Sort)

④ 삽입 정렬(Insertion Sort)

정답 3번

정답

정답은 3번이다.

출제 포인트

이 문제는 피벗을 기준으로 리스트를 작은 부분과 큰 부분으로 분할한 뒤 각 부분 리스트에 같은 방법을 재귀적으로 적용하는 정렬의 특징을 묻는 문제이다.

핵심 개념

버블 정렬은 인접한 두 원소를 비교·교환하면서 큰 값이나 작은 값을 끝으로 보내는 방식이다.
힙 정렬은 힙을 구성한 뒤 루트 원소를 꺼내며 정렬하는 방식이다.
퀵 정렬은 피벗을 하나 정하고 피벗보다 작은 부분과 큰 부분으로 분할한 뒤 각 부분을 재귀적으로 정렬하는 방식이다.
삽입 정렬은 원소를 하나씩 꺼내 이미 정렬된 부분 리스트의 알맞은 위치에 삽입하는 방식이다.

선지별 해설

① X: 버블 정렬은 인접 원소를 반복 비교하여 교환하는 방식이다. 틀린 부분은 피벗 기준 분할과 재귀 적용이라는 설명을 버블 정렬에 대응시킨 점이다. 올바르게는 인접 교환을 반복해 정렬한다.
② X: 힙 정렬은 완전이진트리 형태의 힙을 만든 뒤 최대값 또는 최소값을 하나씩 꺼내 정렬한다. 틀린 부분은 리스트를 피벗 기준의 두 서브 리스트로 나눈다는 설명이다. 올바르게는 힙 구조를 유지하며 루트 원소를 선택한다.
③ O: 퀵 정렬은 특정 키값, 즉 피벗을 기준으로 작은 값과 큰 값을 양쪽으로 분할하고 각 부분 리스트에 같은 정렬을 재귀적으로 적용한다. 발문의 핵심 표현인 분할과 순환 적용이 퀵 정렬의 정의와 일치한다.
④ X: 삽입 정렬은 현재 원소를 앞쪽의 정렬된 부분에 끼워 넣는 방식이다. 틀린 부분은 두 개의 서브 리스트로 분할한 뒤 재귀적으로 처리한다는 설명이다. 올바르게는 정렬된 구간을 한 칸씩 확장하며 적절한 위치에 삽입한다.

NoteBase

오답 포인트

발문에 재귀적으로 같은 정렬 방법을 적용한다는 표현만 보고 분할정복 계열이라는 점을 놓치면 힙 정렬이나 삽입 정렬과 혼동하기 쉽다. 특히 특정한 키값보다 작은 부분과 큰 부분으로 나눈다는 문장이 피벗 분할이라는 사실을 바로 떠올리지 못하면 오답으로 이어진다.

다시 나오면 이렇게 판단

정렬 문제에서 피벗, 분할, 작은 값과 큰 값의 양쪽 분리, 각 부분에 대한 재귀 적용이 함께 나오면 퀵 정렬로 판단할 수 있다. 인접 교환이면 버블 정렬, 힙 구성과 루트 제거이면 힙 정렬, 정렬된 부분에 끼워 넣기이면 삽입 정렬로 구분된다.

암기 포인트

퀵 정렬은 피벗으로 분할하고 재귀적으로 정렬하는 방식이다.

NoteBase

문 17. 트리(tree)에 대한 설명 중 가장 옳지 않은 것은?

① 2-3 트리는 균형 트리(balanced tree)이다.

② 트리는 계층적인 자료를 표현하는 데 이용된다.

③ 균형 트리(balanced tree)는 불균형 트리에 비해 노드 추가가 간단하고 빠르다.

④ 경사 트리(skewed tree)는 이진 트리(binary tree)에 속하며, 배열(array) 표현법을 사용하면 기억공간의 낭비가 많아진다.

정답 3번

정답

정답은 3번이다.

출제 포인트

이 문제는 균형 트리와 경사 트리의 성질, 그리고 트리의 표현 방식이 가지는 특징을 구분하는 문제이다.

핵심 개념

균형 트리는 트리의 높이가 한쪽으로 심하게 치우치지 않도록 유지하여 탐색 효율을 안정적으로 확보하는 트리이다.
2-3 트리는 각 내부 노드가 2개 또는 3개의 자식을 가지며 모든 리프가 같은 레벨에 있는 대표적 균형 탐색 트리이다.
경사 트리는 모든 노드가 한쪽 자식만 가지는 형태로 치우친 이진 트리이며, 배열로 표현하면 빈 칸이 많이 생겨 공간 낭비가 커진다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 자료구조론 · 2019 · 10월

선지별 해설

① O
2-3 트리는 모든 리프의 깊이가 같게 유지되는 균형 트리이므로 옳다.

② O
트리는 부모-자식 관계로 데이터를 연결하므로 조직도, 디렉터리 구조처럼 계층적인 자료를 표현하는 데 적합하다.

③ X
틀린 부분은 "불균형 트리에 비해 노드 추가가 간단하고 빠르다"이다.
균형 트리는 삽입 뒤에도 높이 균형을 유지해야 하므로 분할, 재조정, 회전 같은 추가 처리가 필요할 수 있어 구현과 삽입 과정이 더 복잡하다.
올바르게는 "균형 트리는 불균형 트리보다 탐색 성능은 안정적이지만, 노드 추가는 더 복잡할 수 있다"이다.

④ O
경사 트리는 한쪽으로만 뻗은 이진 트리이므로 이진 트리의 한 형태이다.
이 구조를 배열로 표현하면 실제 노드가 없는 위치까지 인덱스를 비워 두게 되어 기억공간 낭비가 많아진다.

오답 포인트

③은 균형 트리의 장점인 탐색 효율을 삽입의 단순성과 혼동하게 만드는 선지이다. 균형 트리는 검색 성능을 높이는 대신 삽입과 삭제에서 균형 유지 비용이 추가된다는 점이 함정이다.

다시 나오면 이렇게 판단

트리 관련 선지는 먼저 무엇을 비교하는지 본다. 탐색 효율을 말하면 균형 유지의 장점을 떠올리고, 삽입·삭제의 단순성을 말하면 재조정 비용이 있는지 따져 보면 된다. 배열 표현이 나오면 완전 이진 트리에는 유리하고 경사 트리에는 비효율적이라는 점으로 판단할 수 있다.

암기 포인트

균형 트리는 탐색에 강하고, 경사 트리는 배열 표현에 약하다.

NoteBase

문 18. 1차원 배열에 저장된 히프(heap) 구조를 갖는 데이터 중에서 최대 히프(max heap)의 성질을 만족하지 않는 배열 상태는?

① 8 4 7 3 2 6 5 1

② 8 6 7 2 4 1 5 3

③ 8 7 6 5 4 1 2 3

④ 8 7 6 5 4 3 2 1

정답 2번

정답

정답은 2번이다.

출제 포인트

부모 노드가 자식 노드보다 크거나 같아야 하는 최대 힙의 성질을 배열에서 판별하는 문제이다.

핵심 개념

최대 힙은 완전이진트리를 1차원 배열에 저장했을 때 모든 부모가 자신의 자식보다 크거나 같은 구조이다. 이 문제처럼 배열의 첫 원소를 루트로 두는 일반적 표현에서는 i번 노드의 자식이 $2i$, $2i + 1$번에 오므로 각 부모-자식의 대소관계만 보면 된다.

선지별 해설

① X: 8은 4, 7보다 크고 4는 3, 2보다 크며 7은 6, 5보다 크고 3은 1보다 크다. 모든 부모가 자식 이상이므로 최대 힙의 성질을 만족한다. 틀린 표현은 '최대 힙의 성질을 만족하지 않는다'이다. 올바르게는 '최대 힙의 성질을 만족한다'이다.
② O: 4번째 원소 2의 자식으로 8번째 원소 3이 온다. 최대 힙에서는 부모가 자식 이상이어야 하는데 $2 < 3$이므로 이 지점에서 최대 힙 성질이 깨진다. 따라서 이 배열은 최대 힙이 아니다.
③ X: 8은 7, 6보다 크고 7은 5, 4보다 크며 6은 1, 2보다 크고 5는 3보다 크다. 부모-자식 관계가 모두 최대 힙 조건을 만족한다. 틀린 표현은 '최대 힙의 성질을 만족하지 않는다'이다. 올바르게는 '최대 힙의 성질을 만족한다'이다.
④ X: 각 부모를 보면 $8 \geq 7, 6$, $7 \geq 5, 4$, $6 \geq 3, 2$, $5 \geq 1$이 성립한다. 따라서 배열 전체가 최대 힙이다. 틀린 표현은 '최대 힙의 성질을 만족하지 않는다'이다. 올바르게는 '최대 힙의 성질을 만족한다'이다.

오답 포인트

루트와 그 바로 아래 단계만 보고 최대 힙이라고 단정하거나 아니라고 단정하면 틀리기 쉽다. 이 문항의 함정은 ②에서 맨 아래 부모-자식 쌍인 2와 3을 놓치게 만드는 점이다.

NoteBase

다시 나오면 이렇게 판단

배열을 힙으로 볼 때는 마지막 내부 노드까지 차례로 내려가며 모든 부모가 각 자식보다 크거나 같은지만 확인하면 된다. 한 곳이라도 부모 < 자식이 나오면 최대 힙이 아니다.

암기 포인트

최대 힙은 '부모 ≥ 자식'만 끝까지 보면 되고, 위층만 보지 말고 마지막 부모 노드까지 내려가야 한다.

NoteBase

문 19. 〈보기 1〉의 인접 행렬로 표현된 그래프 G에 〈보기 2〉의 알고리즘으로 최소비용신장트리(minimum cost spanning tree)를 생성할 때, 추가되는 간선의 순서로서 옳은 것은? (단, s는 시작 정점을 의미하며, 본 문항에서는 s를 0번 정점으로 한다.)

<보기 1>

	0	1	2	3	4	5	6	7	8
0	0	3	0	2	0	0	0	0	4
1	3	0	0	0	0	0	0	4	0
2	0	0	0	6	0	1	0	2	0
3	2	0	6	0	1	0	0	0	0
4	0	0	0	1	0	0	0	0	8
5	0	0	1	0	0	0	8	0	0
6	0	0	0	0	0	8	0	0	0
7	0	4	2	0	0	0	0	0	0
8	4	0	0	0	8	0	0	0	0

〈보기 2〉

MST(G, s)

{

1. 그래프 G에서 시작 정점 s를 가지고 초기 신장 트리를 만듦
2. 현재 트리의 정점들과 인접한 정점들 중에서 간선의 가중치가 가장 작은 정점 v를 선택
3. 선택된 정점 v와 간선을 신장 트리에 추가 (단, 사이클(cycle)이 발생하지 않아야 함)
4. 현재 신장 트리에 모든 정점이 포함되지 않았다면 2번 단계로 이동

}

① (0, 3), (3, 4), (4, 8), (3, 2), (2, 7), (7, 1), (2, 5), (5, 6)

② (0, 3), (3, 4), (0, 1), (1, 7), (7, 2), (2, 5), (0, 8), (5, 6)

③ (0, 3), (0, 1), (0, 8), (1, 7), (7, 2), (2, 3), (2, 5), (5, 6)

④ (0, 3), (3, 4), (0, 8), (0, 1), (1, 7), (7, 2), (2, 5), (5, 6)

정답 2번

정답

정답은 2번이다.

NoteBase

출제 포인트

이 문제는 시작 정점 0에서 Prim 알고리즘을 적용할 때 현재 신장 트리와 바깥 정점을 잇는 최소 간선을 어떤 순서로 선택하는지를 묻는다.

올바른 순서

1. (0,3) - 초기 트리 {0}에서 후보 간선은 (0,1)=3, (0,3)=2, (0,8)=4이고 최소는 (0,3)이다.
2. (3,4) - 트리 {0,3}에서 후보는 (0,1)=3, (0,8)=4, (3,2)=6, (3,4)=1이고 최소는 (3,4)이다.
3. (0,1) - 트리 {0,3,4}에서 후보는 (0,1)=3, (0,8)=4, (3,2)=6, (4,8)=8이고 최소는 (0,1)이다.
4. (1,7) - 트리 {0,1,3,4}에서 후보는 (1,7)=4, (0,8)=4, (3,2)=6, (4,8)=8이다. 이 문항의 의도대로 정점 번호 순으로 최소 키를 고르면 7이 먼저 확정된다.
5. (7,2) - 트리 {0,1,3,4,7}에서 후보는 (7,2)=2, (0,8)=4, (3,2)=6, (4,8)=8이고 최소는 (7,2)이다.
6. (2,5) - 트리 {0,1,2,3,4,7}에서 후보는 (2,5)=1, (0,8)=4, (4,8)=8이고 최소는 (2,5)이다.
7. (0,8) - 트리 {0,1,2,3,4,5,7}에서 후보는 (0,8)=4, (5,6)=8, (4,8)=8이고 최소는 (0,8)이다.
8. (5,6) - 마지막 남은 정점 6은 (5,6)=8로 연결된다.

왜 이 순서인가: Prim은 매 단계마다 현재 트리 내부와 외부를 가르는 cut를 기준으로, 그 cut를 가로지르는 최소 가중치 간선만 선택한다.

선지별 해설

1번: X. (3,4) 다음 단계에서 현재 트리 {0,3,4}의 최소 후보는 (0,1)=3인데, ①은 (4,8)=8을 먼저 넣어 Prim의 선택 규칙을 어긴다. 이어서 (3,2)=6을 (2,7)=2보다 먼저 둔 점도 틀리다. 올바르게는 (0,1)을 선택한 뒤 (1,7), (7,2)로 진행된다.

2번: O. (0,3) -> (3,4) -> (0,1) -> (1,7) -> (7,2) -> (2,5) -> (0,8) -> (5,6)의 순서는 매 단계마다 현재 트리와 바깥 정점을 잇는 최소 간선을 고른 결과와 일치한다.

3번: X. 세 번째 간선으로 (0,8)=4를 두었지만, 그 시점의 최소 후보는 (3,4)=1이므로 순서가 틀린다. 또 (2,3)은 정점 3이 이미 트리에 들어온 뒤의 간선이라 2를 편입시키는 간선이 될 수 없다. 올바르게는 3은 이미 (0,3)으로 들어왔고, 2는 (7,2)로 편입된다.

4번: X. (3,4) 다음 단계에서 (0,8)=4를 (0,1)=3보다 먼저 선택한 것이 틀리다. Prim은 후보 간선 전체 중 최소 가중치를 먼저 고르므로 세 번째 간선은 반드시 (0,1)이어야 한다. 올바르게는 (0,3), (3,4), (0,1), (1,7) 순으로 진행된다.

오답 포인트

이 문항의 함정은 현재 트리에 인접한 간선만 후보라는 점을 놓치고, 전체 그래프에서 눈에 띄는 간선을 임의로 고르는 데 있다. 특히 ①처럼 큰 가중치인 (4,8)을 너무 빨리 넣거나, ③처럼 이미 트리에 포함된 정점 3을 다시 연결하는 간선을 편입 간선처럼 보는 실수가 자주 나온다. ④는 3과 4가 연결된 뒤에도 0에서 나가는 4짜리 간선을 먼저 보고 3짜리 (0,1)을 놓치게 만든다.

NoteBase

다시 나오면 이렇게 판단

매 단계마다 현재 트리에 포함된 정점 집합을 먼저 적어 두고, 그 집합과 바깥 정점을 잇는 간선만 후보로 남긴 뒤 최소 가중치 하나를 선택하는 방식으로 판단한다. 이미 트리 안에 있는 두 정점을 잇는 간선은 후보가 아니며, 동률이면 문제에서 의도한 정점 번호 순 기준을 함께 본다.

암기 포인트

Prim은 현재 트리에서 바깥으로 최소 간선 1개씩 확장하는 알고리즘이고, Kruskal처럼 전체 간선을 한꺼번에 가중치 순으로 고르는 방식이 아니다.

NoteBase

문 20. 이진 트리(binary tree)에 대한 설명 중 가장 옳지 않은 것은?

① 100개로 구성되어 있는 완전 이진 트리(complete binary tree)에서 최하위 수준(level)에는 36개의 단말 노드가 존재한다.

② 이진 트리에서 좌/우측 자식 노드를 모두 가진 노드의 개수가 100이라면, 자식 노드가 없는 단말 노드의 개수는 101이다.

③ 이진 트리를 배열(array) 표현법으로 구현할 때, 인덱스 101번에 저장된 노드의 부모 노드 인덱스는 50번이다.

④ 이진 트리를 링크(link) 표현법으로 구현할 때, 이진 트리에 포함된 노드의 개수가 100이라면, NULL 링크의 개수는 101이다.

정답 1번

정답

정답은 1번이다.

출제 포인트

이 문제는 완전 이진 트리의 마지막 레벨 노드 수, 이진 트리의 단말 노드 수 공식, 배열 표현의 부모 인덱스, 링크 표현의 NULL 링크 수를 구분하는 문제이다.

핵심 개념

완전 이진 트리는 마지막 레벨만 덜 찰 수 있고 노드는 왼쪽부터 채워지는 이진 트리이다.
이진 트리에서는 자식이 2개인 노드 수를 n_2, 단말 노드 수를 n_0라 하면 $n_0 = n_2 + 1$이 성립한다.
배열 표현에서 1번 인덱스를 루트로 두면 노드 i의 부모는 $\lfloor i/2 \rfloor$이다.
링크 표현에서는 노드가 n개일 때 전체 자식 링크는 $2n$개이고 실제 간선은 $n - 1$개이므로 NULL 링크는 $n + 1$개이다.

NoteBase

선지별 해설

① X

100개의 완전 이진 트리는 위 레벨이 먼저 꽉 차고 마지막 레벨이 왼쪽부터 채워진다. 1~6레벨까지는 $2^6 - 1 = 63$개가 찬다. 따라서 마지막 레벨에는 $100 - 63 = 37$개의 노드가 존재한다. 최하위 수준의 단말 노드는 36개가 아니라 37개이다. 틀린 표현은 "36개의 단말 노드"이고, 올바르게는 "37개의 단말 노드"이다.

② O

이진 트리에서 자식이 2개인 노드 수를 n_2, 단말 노드 수를 n_0라 하면 항상 $n_0 = n_2 + 1$이다. 따라서 좌우 자식을 모두 가진 노드가 100개이면 단말 노드는 101개이다.

③ O

배열 표현법에서 루트가 1번 인덱스이면 부모 인덱스는 $\lfloor i/2 \rfloor$이다. 따라서 101번 노드의 부모 인덱스는 $\lfloor 101/2 \rfloor = 50$이다.

④ O

링크 표현법에서 노드가 100개이면 전체 자식 링크 칸은 200개이다. 이 중 실제로 연결된 링크는 간선 수와 같은 99개이므로 NULL 링크 수는 $200 - 99 = 101$개이다.

오답 포인트

①은 마지막 레벨의 노드 수를 셀 때 37번째 노드를 빠뜨리기 쉽다는 점을 노린 선지이다. ②와 ④는 각각 $n_0 = n_2 + 1$, NULL 링크 수 $= n + 1$ 공식을 정확히 기억하지 못하면 흔들리기 쉽다.

다시 나오면 이렇게 판단

완전 이진 트리 문제는 먼저 위 레벨까지 꽉 찬 노드 수를 뺀 뒤 마지막 레벨 노드 수를 계산하면 된다. 일반 이진 트리의 단말 노드 수와 NULL 링크 수는 각각 $n_0 = n_2 + 1$, $n + 1$ 공식을 바로 대입하면 판단할 수 있다. 배열 표현은 부모가 $\lfloor i/2 \rfloor$인지 먼저 확인하면 된다.

암기 포인트

완전 이진 트리의 마지막 레벨은 남은 노드 수만큼 채워지며, 이진 트리는 $n_0 = n_2 + 1$, NULL 링크 수는 $n + 1$, 배열 부모는 $\lfloor i/2 \rfloor$이다.

NoteBase