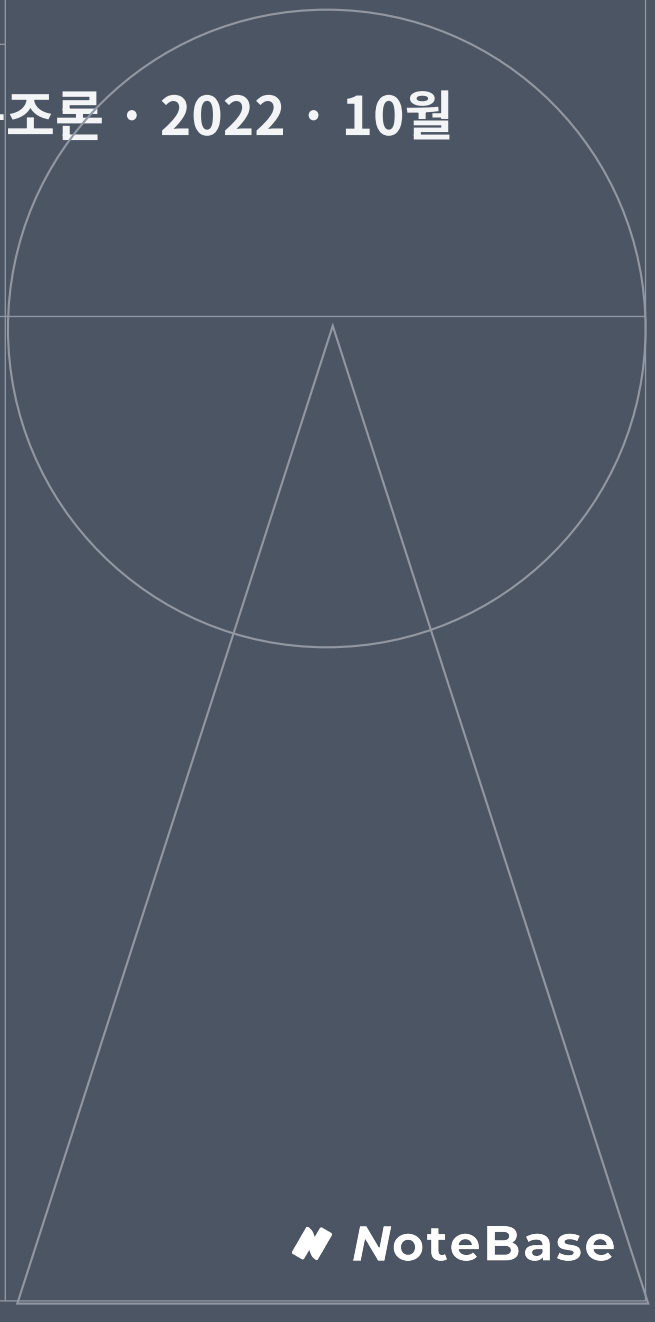
기출OX 해설집

공무원/서울시/7급 · 자료구조론 · 2022 · 10월



NoteBase

기출OX 해설집

공무원/서울시/7급 · 자료구조론 · 2022 · 10월

공무원/서울시/7급 · 자료구조론 · 2022 · 10월

문 1. 주어진 자료에 맞는 정렬(sort) 알고리즘을 선택할 때 고려해야 할 사항으로 가장 옳지 않은 것은?

① 필요한 작업 공간 및 소요 시간

② 정렬할 자료의 양

③ 정렬에 필요한 메모리(memory)의 크기

④ 추가되는 데이터의 배열 상태

정답 4번

출제 포인트

정렬 알고리즘 선택 기준에서 시간, 공간, 메모리, 데이터 규모 같은 핵심 자원 요소와 부수적 특성을 구분하는 문제이다.

핵심 개념

정렬 알고리즘 선택의 일반 기준은 소요 시간, 필요한 작업 공간, 사용 가능한 메모리, 데이터 규모처럼 성능과 자원 제약을 직접 좌우하는 요소이다.
입력 자료의 초기 배열 상태는 일부 알고리즘의 성능에 영향을 줄 수 있으나, 보기 ④처럼 '추가되는 데이터의 배열 상태'를 일반적 선택 기준으로 제시한 표현은 대표성이 약하다.

NoteBase

선지별 해설

① X
필요한 작업 공간 및 소요 시간은 정렬 알고리즘의 대표적인 선택 기준이다.
틀린 부분은 이 보기를 '가장 옳지 않은 것'으로 보는 판단이다.
올바르게는 정렬 알고리즘을 선택할 때 반드시 비교해야 할 기준이다.

② X
정렬할 자료의 양이 커지면 알고리즘의 시간 복잡도 차이가 크게 드러나므로 중요한 선택 기준이다.
틀린 부분은 이 보기를 '가장 옳지 않은 것'으로 보는 판단이다.
올바르게는 데이터의 양은 정렬 알고리즘 선택에 직접 영향을 준다.

③ X
정렬에 사용할 수 있는 메모리 크기는 추가 배열 사용 가능 여부와 전체 구현 방식을 좌우하므로 핵심 기준이다.
틀린 부분은 이 보기를 '가장 옳지 않은 것'으로 보는 판단이다.
올바르게는 정렬에 필요한 메모리 크기도 중요한 선택 기준이다.

④ O
'추가되는 데이터의 배열 상태'는 정렬 알고리즘을 고를 때의 일반적이고 핵심적인 기준으로 보기는 어렵다.
이미 정렬된 정도 같은 입력 상태가 일부 알고리즘의 성능에 영향을 줄 수는 있으나, 보기의 표현은 시간, 공간, 메모리, 데이터 규모에 비해 본질적 선택 기준성이 약하다.

오답 포인트

입력 자료의 상태가 전혀 중요하지 않다고 단정하면 ④를 고르기 어렵다. 이 문항은 '영향을 줄 수 있는 요소'와 '일반적으로 먼저 따지는 핵심 선택 기준'을 구분하는 데 함정이 있다.

다시 나오면 이렇게 판단

정렬 알고리즘 선택 기준을 묻는 문제에서는 먼저 시간 복잡도, 추가 공간, 메모리 제약, 데이터 규모를 본다. 그 다음에 이미 정렬된 정도처럼 특정 알고리즘에서만 민감한 조건을 부차적으로 판단한다.

암기 포인트

정렬 선택의 1차 기준은 시간·공간·메모리·데이터 규모이고, 입력 상태는 보조 기준이다.

NoteBase

문 2. C언어로 작성된 <보기>의 함수는 길이가 n인 배열 array에 저장된 값들을 역순으로 재배치하는 프로그램이다. (가)에 들어갈 내용으로 가장 옳은 것은?

<보기>

```
void reverse(int array[], int n)
{
        int t;
        for (int i = 0; i < n / 2; i++) {
                (가)
        }
}
```

①
```
t = array[i - 1];
array[i - 1] = array[n - i - 1];
array[n - i - 1] = t;
```

②
```
t = array[i - 1];
array[i - 1] = array[n - i];
array[n - i] = t;
```

③
```
t = array[i];
array[i] = array[n - i - 1];
array[n - i - 1] = t;
```

④
```
t = array[i];
array[i] = array[n - i];
array[n - i] = t;
```

정답 3번

출제 포인트

이 문제는 0부터 시작하는 C 배열 인덱스를 이용해 앞쪽 원소와 뒤쪽 대응 원소를 서로 교환하는 역순 배치 규칙을 코드에 정확히 적용할 수 있는지를 묻는다.

NoteBase

사례 정리

배열의 길이는 n이고, 반복문은 i를 0부터 $n/2 - 1$까지 증가시키면서 양끝 원소를 교환한다.
판단에 중요한 조건은 C 배열의 유효 인덱스가 0부터 $n - 1$까지라는 점과, i번째 원소의 대칭 위치가 $n - i - 1$번째라는 점이다.

핵심 개념

배열을 역순으로 만드는 기본 교환 규칙은 앞에서 i번째 원소와 뒤에서 i번째 원소를 맞바꾸는 것이다.
0부터 시작하는 배열에서 뒤에서 i번째 원소의 실제 인덱스는 $n - i - 1$이므로, 교환식은 `array[i]`와 `array[n - i - 1]`를 대상으로 해야 한다.

적용 기준

이 문항에서는 각 선택지가 첫 반복인 $i = 0$에서도 유효한 인덱스를 쓰는지와, 교환 대상이 정확히 대칭 위치인지 확인하면 된다. 유효 범위를 벗어나는 `array[-1]`, `array[n]`이 나오면 즉시 틀린 선택지이다.

선지별 해설

① X: `array[i - 1]`를 사용하므로 첫 반복에서 `array[-1]`을 참조하게 되어 범위를 벗어난다. 또한 앞쪽 원소는 `array[i]`여야 한다. 틀린 표현은 `array[i - 1]`이고, 올바르게는 `array[i]`이다.
② X: `array[i - 1]` 때문에 첫 반복에서 `array[-1]`이 되고, `array[n - i]`도 첫 반복에서 `array[n]`이 되어 범위를 벗어난다. 틀린 표현은 `array[i - 1]`, `array[n - i]`이고, 올바르게는 각각 `array[i]`, `array[n - i - 1]`이다.
③ O: `t = array[i]; array[i] = array[n - i - 1]; array[n - i - 1] = t;`는 앞쪽 i번째 원소와 뒤쪽 대응 원소 $n - i - 1$번째를 정확히 교환하므로 배열을 역순으로 재배치하는 올바른 코드이다.
④ X: 앞쪽 원소를 `array[i]`로 잡은 점은 맞지만, 뒤쪽 대응 위치를 `array[n - i]`로 쓰면 첫 반복에서 `array[n]`이 되어 범위를 벗어난다. 틀린 표현은 `array[n - i]`이고, 올바르게는 `array[n - i - 1]`이다.

오답 포인트

출제자가 노린 함정은 배열 인덱스가 0부터 시작한다는 점을 한 칸씩 밀려서 착각하게 만드는 데 있다. `i-1`이나 `n-i`를 자연스럽게 보이면 맞는 식처럼 보이지만, 첫 반복 $i = 0$을 대입하면 곧바로 잘못임이 드러난다.

다시 나오면 이렇게 판단

배열 교환 코드 문제는 먼저 유효 인덱스 범위를 0부터 $n - 1$까지로 적고, 그다음 왼쪽 i번째와 짝이 되는 오른쪽 인덱스를 $n - i - 1$로 계산해 본다. 이후 첫 반복값 $i = 0$을 대입해 범위 밖 참조가 생기는지 확인하면 빠르게 걸러낼 수 있다.

NoteBase

암기 포인트

0부터 시작하는 길이 n 배열의 마지막 인덱스는 $n-1$이고, i의 대칭 위치는 항상 $n-i-1$이다.

문 3. 비어 있는 최소 힙(min heap)에 <보기>에 주어진 데이터를 순서대로 하나씩 삽입 연산을 수행하여 모두 삽입한 후, 2번의 삭제 연산을 수행하였다. 최소 힙의 가장 마지막 레벨의 가장 왼쪽 노드에 저장된 데이터는? (단, 최소 힙은 완전 이진트리로 구현한다.)

<보기>
153, 98, 180, 128, 105, 177, 192, 99, 116

① 128
② 153
③ 177
④ 192

정답 2번

출제 포인트

이 문제는 최소 힙에 데이터를 순서대로 삽입한 뒤 삭제 연산을 2회 수행했을 때 완전 이진트리 배열 형태가 어떻게 바뀌는지를 묻는 문제이다.

올바른 순서

1. 주어진 9개 값을 차례대로 삽입한다. 각 삽입 때마다 부모보다 작으면 위로 올려 최소 힙 조건을 맞춘다.
2. 모든 삽입이 끝나면 힙은 [98, 99, 177, 105, 128, 180, 192, 153, 116]이 된다.
3. 첫 번째 삭제에서 루트 98을 제거하고 마지막 원소 116을 루트로 올린 뒤 더 작은 자식과 교환하면 [99, 105, 177, 116, 128, 180, 192, 153]이 된다.
4. 두 번째 삭제에서 루트 99를 제거하고 마지막 원소 153을 루트로 올린 뒤 더 작은 자식과 교환하면 최종 힙은 [105, 116, 177, 153, 128, 180, 192]이 된다.

왜 이 순서인가: 삭제 연산은 항상 루트 삭제 후 마지막 원소를 루트로 옮기고 아래로 내려 보내는 복구 과정을 거치므로, 두 번의 삭제 결과를 순서대로 따라가야 마지막 레벨의 위치가 정해진다.

NoteBase

선지별 해설

① X: 128은 최종 힙 [105, 116, 177, 153, 128, 180, 192]에서 마지막 레벨의 두 번째 노드이다. 틀린 표현은 128이 가장 왼쪽 노드라는 판단이다. 올바르게는 마지막 레벨의 가장 왼쪽 노드는 153이고, 128은 그 오른쪽에 있다.
② O: 두 번의 삭제까지 마친 최종 힙은 [105, 116, 177, 153, 128, 180, 192]이다. 마지막 레벨은 153, 128, 180, 192 순서이므로 가장 왼쪽 노드는 153이다.
③ X: 177은 최종 힙의 세 번째 원소로 마지막 레벨이 아니라 그 윗레벨의 오른쪽 자식 위치에 있다. 틀린 표현은 177이 마지막 레벨에 있다는 판단이다. 올바르게는 177은 레벨 2에 있고 마지막 레벨의 가장 왼쪽 노드는 153이다.
④ X: 192는 최종 힙의 마지막 원소로 마지막 레벨의 가장 오른쪽 노드이다. 틀린 표현은 192가 마지막 레벨의 가장 왼쪽 노드라는 판단이다. 올바르게는 192는 맨 오른쪽이고 가장 왼쪽은 153이다.

오답 포인트

삭제 연산 뒤의 힙을 정렬 결과처럼 오해하면 128이나 177을 고르기 쉽다. 이 문제는 값의 크기 순서가 아니라 완전 이진트리의 배열 배치와 삭제 후 재heapify 결과를 묻는다.

다시 나오면 이렇게 판단

힙 문제는 삽입이 끝난 배열 상태를 먼저 확정하고, 삭제마다 루트 제거 → 마지막 원소 이동 → 더 작은 자식과 교환의 순서로 한 단계씩 추적하면 된다. 마지막 레벨 위치를 묻는다면 최종 배열을 완전 이진트리로 다시 읽으면 된다.

암기 포인트

최소 힙 삭제는 루트 제거 후 마지막 원소를 루트로 올리고 아래로 내려 보낸다. 최종 배열의 마지막 레벨 왼쪽 끝은 인덱스 4번째 원소가 아니라 마지막 레벨의 첫 원소이다.

NoteBase

문 4. 후위표기식(postfix expression)으로 표현된 수식 중에 계산 결과가 나머지 수식과 다른 것은?

① A B C + *

② B C * A +

③ C A * B A * +

④ A B * A C * +

정답 2번

출제 포인트

이 문제는 후위표기식을 중위표기식으로 복원해 서로 동치인 식과 동치가 아닌 식을 구분하는지를 묻는다.

핵심 개념

후위표기식은 피연산자를 먼저 놓고 연산자를 뒤에 두는 표기법이다. 스택으로 읽으면 연산자 하나가 나올 때마다 직전 두 식을 묶어 중위식으로 복원할 수 있다. 이 문항에서는 ①, ③, ④가 모두 $A(B+C)$로 정리되고, ②만 $A+BC$로 정리된다.

선지별 해설

① X

`A B C + *`는 먼저 $B+C$를 만든 뒤 여기에 A를 곱하므로 $A(B+C)$이다. 틀린 표현은 이 선지가 나머지와 다른 식이라는 취지이다. 올바르게는 ①은 ③, ④와 같은 식이다.

② O

`B C * A +`는 먼저 BC를 만든 뒤 A를 더하므로 $A+BC$이다. ①, ③, ④의 $A(B+C)=AB+AC$와 결과가 다르므로 정답 선지이다.

③ X

`C A * B A * +`는 CA와 BA를 만든 뒤 더하므로 $CA+BA$이고, 교환법칙을 적용하면 $AC+AB=A(B+C)$이다. 틀린 표현은 이 선지가 다른 식이라는 취지이다. 올바르게는 ③은 ①, ④와 같은 식이다.

④ X

`A B * A C * +`는 AB와 AC를 만든 뒤 더하므로 $AB+AC$이다. 분배법칙으로 $A(B+C)$와 같으므로 ①, ③과 동치이다. 틀린 표현은 이 선지가 다른 식이라는 취지이다. 올바르게는 ④는 ①, ③과 같은 식이다.

NoteBase

오답 포인트

③은 문자 순서가 `CA`, `BA`로 보이기 때문에 전혀 다른 식처럼 착각하기 쉽다. 그러나 곱셈의 교환법칙으로 $CA = AC$, $BA = AB$가 되므로 결국 ①, ④와 같은 식이다. ②는 덧셈 기호가 맨 끝에 있어 ①과 비슷해 보이지만 실제 구조는 $A + (BC)$라는 점이 함정이다.

다시 나오면 이렇게 판단

후위표기식 비교 문제는 각 선지를 바로 계산하려 하지 말고 연산자가 나올 때마다 두 항을 묶어 중위식으로 바꿔 본다. 그다음 교환법칙과 분배법칙까지 적용해 같은 형태로 정리하면 동치 여부가 빠르게 드러난다.

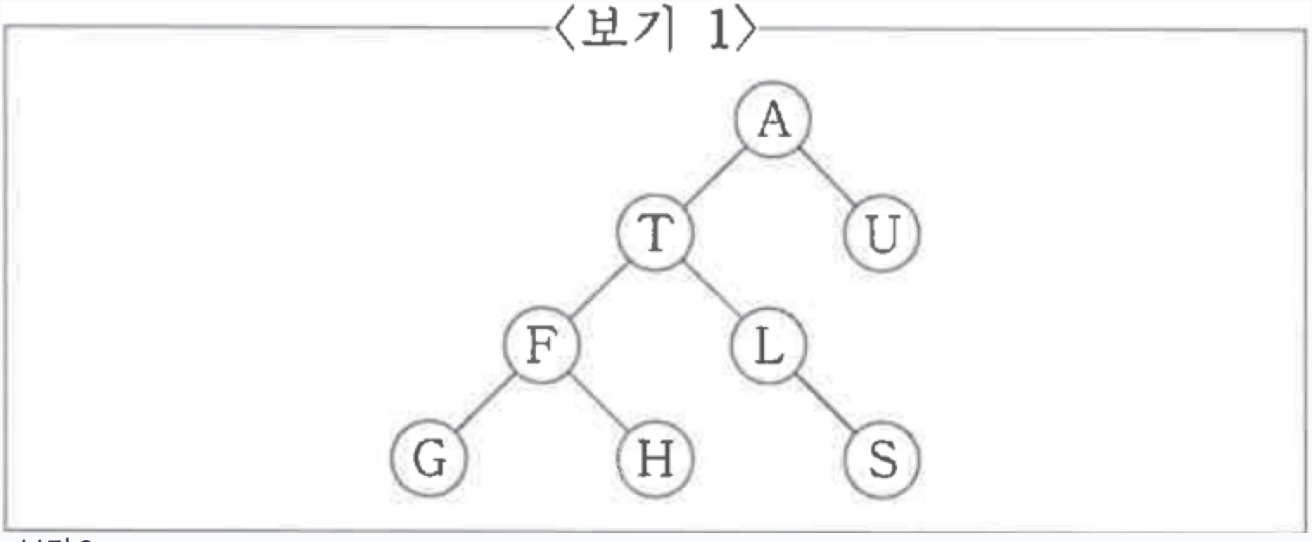
암기 포인트

후위표기식은 스택으로 중위식 복원 후 비교하고, 최종 비교는 동치 변형인 $AB + AC = A(B + C)$로 정리한다.

NoteBase

문 5. <보기 1>과 같이 노드 A를 루트로 하는 이진트리를 <보기 2>의 규칙에 따라 구성할 때, 구성한 배열의 원소를 순서대로 바르게 나열한 것은? (단, 배열의 인덱스는 1부터 시작한다. 원소가 빈칸인 경우에는 '-'로 표시한다. 배열의 마지막이 연속적으로 빈칸인 경우 해당 부분을 생략하여 표시한다.)

<보기 1>



<보기 2>

(가) 노드 i의 부모노드 인덱스 = i / 2 (정수부분)

(나) 노드 i의 왼쪽 자식노드 인덱스 = 2 * i

(다) 노드 i의 오른쪽 자식노드 인덱스 = 2 * i + 1

① A T U F L G H S

② A T F G U L H - S

③ A T U F L - - G H - S

④ A T U F - L - - - G H - - - - S

정답 3번

출제 포인트

이 문제는 이진트리를 1부터 시작하는 배열 인덱스 규칙으로 옮길 때 빈칸 위치까지 정확히 배치할 수 있는지를 묻는다.

NoteBase

선지별 해설

1번 X: 'A T U F L G H S'는 빈칸 없이 노드만 순서대로 적은 형태이다. 6번과 7번 인덱스는 U의 자식이 없으므로 빈칸이어야 하고, 10번 인덱스도 L의 왼쪽 자식이 없으므로 빈칸이어야 한다. 틀린 표현은 'A T U F L G H S'이고, 올바르게는 'A T U F L - - G H - S'이다.
2번 X: 'A T F G U L H - S'는 부모-자식 인덱스 규칙을 무시하고 왼쪽으로만 채운 형태이다. 3번 인덱스에는 A의 오른쪽 자식 U가 와야 하는데 F가 들어가 있어 처음부터 구조가 어긋난다. 틀린 표현은 3번 위치의 'F'를 포함한 전체 배열 배치이고, 올바르게는 'A T U F L - - G H - S'이다.
3번 O: 1:A, 2:T, 3:U, 4:F, 5:L, 6:-, 7:-, 8:G, 9:H, 10:-, 11:S로 배치하면 제시된 트리와 인덱스 규칙이 모두 맞는다. 마지막 연속 빈칸은 없으므로 여기서 그대로 끝내면 된다.
4번 X: 'A T U F - L - - - G H - - - - S'는 L과 그 아래 노드들을 한 단계씩 뒤로 밀어 넣은 형태이다. L은 T의 오른쪽 자식이므로 5번 인덱스에 와야 하고, G와 H는 F의 자식이므로 8번과 9번에 와야 하며, S는 11번 인덱스에 와야 한다. 틀린 표현은 5번 위치의 '-' 이후 전체 배치이고, 올바르게는 'A T U F L - - G H - S'이다.

오답 포인트

빈칸을 무시하고 레벨 순서로 노드만 적거나, 자식이 없는 자리의 '-'를 마지막 공백처럼 생략하면 틀린다. 이 문항의 함정은 6번, 7번, 10번처럼 중간에 생기는 빈칸도 배열 원소로 유지해야 한다는 점이다.

다시 나오면 이렇게 판단

루트부터 인덱스를 붙이고, 각 노드에 대해 왼쪽은 $2i$, 오른쪽은 $2i+1$을 바로 적어 내려가면 된다. 자식이 없는 자리는 중간이라면 '-'로 남기고, 맨 뒤에만 연속 빈칸이 있을 때만 생략된다.

암기 포인트

배열 이진트리는 레벨순 저장이 아니라 인덱스 규칙 저장이며, 중간 빈칸은 지우지 않는다.

NoteBase

문 6. 버킷(bucket)당 슬롯(slot)이 1개이고 모두 10개의 버킷으로 구성된 해시 테이블(hash table)에 대해서 해시함수(hash function) h가 <보기 1>과 같이 정의되며, 충돌(collision) 처리는 선형 조사법(linear probing)을 사용한다. 8개의 데이터를 해시 테이블에 삽입했을 때 해시 테이블의 모습이 <보기 2>와 같다면, 이와 같은 형태의 테이블이 나타날 수 없는 입력 순서는?

<보기 1>

h(k) = k 10

〈보기 2〉

[0]	80
[1]	61
[2]	30
[3]	
[4]	44
[5]	65
[6]	64
[7]	77
[8]	17
[9]	

① 77 65 44 61 80 30 17 64

② 61 44 80 65 77 17 64 30

③ 80 61 44 30 77 65 64 17

④ 44 64 65 80 61 30 77 17

정답 4번

출제 포인트

이 문제는 해시함수 h(k)=k 10와 선형 조사법을 적용할 때 충돌이 난 원소들의 삽입 선후관계가 최종 테이블 배치를 어떻게 바꾸는지를 묻는 문제이다.

NoteBase

올바른 순서

최종 테이블에서 핵심은 [4], [5], [6]의 배치이다.
1. 44는 해시값이 4이므로 [4]에 먼저 자리잡는다.
2. 65가 64보다 먼저 들어오면 [5]가 비어 있으므로 65는 [5]에 들어간다.
3. 그 다음 64가 들어오면 [4]는 44, [5]는 65가 이미 차 있으므로 선형 조사로 [6]에 들어간다.
4. 80, 61, 30, 77, 17은 각각 [0], [1], [2], [7], [8]에 배치되어 나머지 모양을 완성한다.
왜 이 순서인가: 64와 65는 모두 충돌 구간 [4]에서 연속으로 밀려나는 원소이므로 둘의 선후관계가 [5]와 [6]의 주인을 결정한다.

선지별 해설

1번: O. 65가 먼저 들어가 [5]를 차지한 뒤 64가 나중에 들어가면 [4]와 [5]를 지나 [6]에 놓인다. 따라서 최종 테이블이 [0]=80, [1]=61, [2]=30, [4]=44, [5]=65, [6]=64, [7]=77, [8]=17로 맞게 만들어진다.
2번: O. 61, 44, 80, 65, 77, 17을 먼저 넣어도 각 원소는 목표 위치에 들어가고, 그 뒤 64는 [4]와 [5]의 충돌 때문에 [6]으로, 30은 [0]과 [1]의 충돌 때문에 [2]로 들어간다. 최종 배치가 <보기 2>와 같다.
3번: O. 80, 61, 44, 30, 77까지 넣은 뒤 65가 [5]를 차지하고, 그 다음 64가 [4]와 [5]를 지나 [6]에 들어간다. 마지막 17도 [7] 충돌 후 [8]에 들어가므로 <보기 2>를 그대로 만든다.
4번: X. 틀린 부분은 64가 65보다 먼저 입력된 순서이다. 64가 먼저 들어오면 44 때문에 [4]에서 충돌한 뒤 곧바로 [5]를 차지하고, 뒤이어 들어온 65는 [5]가 이미 차 있으므로 [6]에 들어간다. 올바르게는 64와 65의 선후가 바뀌어 65가 먼저, 64가 나중에 들어와야 [5]=65, [6]=64가 된다.

오답 포인트

64와 65는 모두 4번대 슬롯에서 충돌을 겪기 때문에 해시값만 보고 같은 결과가 나온다고 착각하기 쉽다. 선형 조사법에서는 같은 충돌 구간을 공유하는 원소들의 입력 순서가 실제 저장 위치를 바꾼다는 점이 함정이다.

다시 나오면 이렇게 판단

선형 조사 해시 문제는 최종 테이블에서 충돌이 있었던 연속 구간을 먼저 찾고, 그 구간의 앞칸부터 누가 먼저 들어와야 하는지를 역으로 추적하면 된다. 특히 같은 해시값이나 같은 조사 구간을 공유하는 원소들의 선후관계를 먼저 확인하면 정답을 빠르게 가를 수 있다.

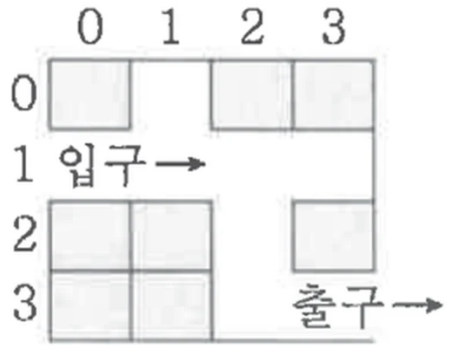
암기 포인트

선형 조사법은 먼저 들어온 원소가 앞칸을 차지하고, 나중 원소가 뒤칸으로 밀린다.

NoteBase

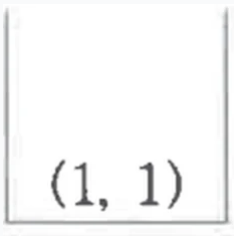
문 7. <보기 1>은 2차원 미로(maze)를 배열로 표현한 것이며, <보기 2>는 이 구조를 이용하여 미로를 찾는 과정이다. 스택에 저장될 미로의 각 위치를 (행 번호, 열 번호)로 표현할 때, <보기 1>에서 입구는 (1, 0), 출구는 (3, 3)이다. <보기 1>의 미로에 대해서 <보기 2>의 미로찾기를 실행하는 과정 중에 스택에 나타날 수 없는 것은?

〈보기 1〉



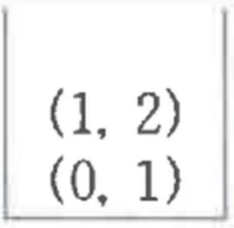
〈보기 2〉

```
def maze_search()
        현재 위치를 입구로 초기화
        while(현재 위치가 출구가 아니면)
                (현재 위치를 방문한 것으로 표시)
                (현재 위치에서 이동이 가능한 위치 중에 아직
                방문하지 않은 위치들을 스택에 push)
                if(스택이 비어있으면)
                        return 실패
                else
                        스택에서 pop하여 현재 위치로 설정
        return 성공
```

① 

NoteBase

② 

③ 

④ 

정답 2번

출제 포인트

이 문제는 깊이 우선 탐색 방식의 미로 찾기에서 push와 pop의 선후관계를 따라 스택에 남을 수 있는 좌표 집합을 추적하는 문제이다.

올바른 순서

1. 현재 위치를 입구 (1,0)으로 두고 방문 표시를 한 뒤, 이동 가능한 칸 (1,1)을 스택에 push한다.
2. (1,1)에 도달하면 미방문 이웃인 (0,1)과 (1,2)를 스택에 push하고, 마지막에 push된 좌표를 pop하여 다음 현재 위치로 사용한다.
3. 이후에도 현재 위치는 스택에서 pop되어 나온 좌표이므로 스택에는 남지 않고, 아직 처리하지 않은 분기점 좌표만 스택에 남는다.
4. 출구 방향으로 진행하는 동안 남겨 둔 분기점 위에 새 좌표가 추가될 수 있으므로, (0,1)이나 (1,3) 위에 (3,3)이 쌓인 상태는 가능하다.

왜 이 순서: 현재 위치는 스택에 저장된 후보를 pop하여 얻는 값이므로, 이미 현재 위치가 된 좌표가 같은 시점의 스택 안에 다시 들어 있는 상태는 만들 수 없다.

NoteBase

선지별 해설

① O
입구 (1,0)에서 이동 가능한 칸은 (1,1) 하나뿐이므로 첫 반복에서 스택이 (1,1)만 담은 상태는 나타날 수 있다.

② X
틀린 부분은 (1,2)가 이미 pop되어 현재 위치가 된 뒤에도 스택 안에 남아 있다는 점이다. (1,1)에서 (0,1)과 (1,2)를 push한 뒤 (1,2)를 pop하여 이동하면 스택에는 남은 분기점만 남는다. 그 다음 (1,2)에서 새로 push되는 좌표는 (1,3)이나 (2,2)이지 (1,2) 자체가 아니다.
올바르게는 (1,3)이 스택에 나타나려면 (0,1) 또는 (2,2) 같은 아직 처리하지 않은 분기점과 함께 남을 수는 있어도, 이미 현재 위치로 꺼낸 (1,2)가 동시에 스택에 남을 수는 없다.

③ O
(1,1)에서 (0,1)과 (1,2)를 차례로 push하면 스택에 (1,2)와 (0,1)이 함께 쌓인 상태가 나타날 수 있다. 이는 두 좌표가 모두 아직 방문하지 않은 다음 후보이기 때문이다.

④ O
(1,2)에서 (1,3)을 남겨 둔 채 (2,2)로 진행하고, 다시 (3,2)에서 출구 (3,3)를 push하면 스택에 (3,3), (1,3), (0,1)처럼 출구 후보와 이전 분기점이 함께 남은 상태가 나타날 수 있다.

오답 포인트

현재 위치와 스택에 대기 중인 후보를 구분하지 않으면 ②처럼 이미 pop된 좌표가 스택에도 남아 있다고 착각하기 쉽다. 이 알고리즘의 스택은 앞으로 이동할 후보를 담는 공간이지, 지금 서 있는 위치를 보관하는 공간이 아니다.

다시 나오면 이렇게 판단

DFS 스택 추적 문제는 현재 위치를 먼저 정하고, 그 위치에서 갈 수 있는 미방문 후보만 스택에 쌓는다는 원칙으로 본다. 어떤 좌표가 현재 위치로 pop된 뒤 같은 시점의 스택에도 보이면 그 선지는 바로 배제할 수 있다.

암기 포인트

스택에는 후보만 남고 현재 위치는 남지 않는다.

NoteBase

문 8. <보기>의 인접 행렬(adjacency matrix)은 도시 사이의 경로값을 나타낸 것이다. (가) 서울에서 평택, (나) 서울에서 대구까지 각각 최단 경로값을 구하고자 한다. (가)와 (나)의 최단 경로값의 합은? (단, '∞'은 두 도시 간의 직접적인 연결이 없음을 의미하고, 출발지와 도착지가 같은 경로는 없다.)

〈보기〉

	서울	성남	평택	기흥	대구
서울	0	20	30	35	∞
성남	20	0	25	23	∞
평택	30	25	0	30	50
기흥	35	23	30	0	40
대구	∞	∞	50	40	0

① 75

② 95

③ 105

④ 125

정답 3번

출제 포인트

이 문제는 인접 행렬에서 직접 경로와 우회 경로의 가중치 합을 비교하여 최단 경로값을 구하는 문제이다.

조건 정리

주어진 값: 서울, 성남, 평택, 기흥, 대구 사이의 경로값이 인접 행렬로 제시되어 있다.
구해야 할 값: (가) 서울→평택의 최단 경로값과 (나) 서울→대구의 최단 경로값의 합이다.
주의할 조건: ∞는 직접 연결이 없다는 뜻일 뿐 우회 경로까지 없다는 뜻은 아니다. 출발지와 도착지가 같은 경로는 고려하지 않는다.

사용할 공식

d(u,v)=(u에서 v까지 가능한 각 경로의 가중치 합)이다.
이 원리를 쓰는 이유는 인접 행렬의 각 값이 간선의 가중치이므로, 서울에서 목적지까지 갈 수 있는 여러 경로의 합을 비교하면 최단 경로값을 결정할 수 있기 때문이다.

NoteBase

계산 과정

1. (가) 서울→평택은 직접 가면 30이다. 우회 경로는 서울→성남→평택이 20+25=45, 서울→기흥→평택이 35+30=65, 서울→성남→기흥→평택이 20+23+30=73이므로 최단값은 30이다.
2. (나) 서울→대구는 직접 연결이 없으므로 우회만 비교한다. 서울→기흥→대구는 35+40=75, 서울→평택→대구는 30+50=80, 서울→성남→기흥→대구는 20+23+40=83, 서울→성남→평택→대구는 20+25+50=95이므로 최단값은 75이다.
3. 따라서 합은 30+75=105이다.

선지별 해설

① X

75는 서울→대구의 최단 경로값만 계산한 값이다. (가) 서울→평택의 최단값 30이 빠졌으므로 합으로는 틀리다. 틀린 값 75 → 올바르게는 105이다.

② X

95는 서울→성남→평택→대구의 경로값 20+25+50과 같아서 (나) 하나를 비효율 경로로 계산한 값에 가깝다. 서울→대구의 최단값은 서울→기흥→대구의 75이고, 여기에 (가)의 30을 더해야 하므로 올바르게는 105이다.

③ O

서울→평택의 최단값은 30이고, 서울→대구의 최단값은 75이다. 두 값을 더하면 105이므로 맞다.

④ X

125는 최단 경로가 아닌 긴 경로값을 섞어 더한 과대계산이다. 예를 들어 서울→기흥→대구 75에 평택↔대구 50을 덧붙인 값과 같아서 문제에서 요구한 두 최단 경로의 합이 아니다. 틀린 값 125 → 올바르게는 105이다.

오답 포인트

직접 연결이 보이면 바로 선택하거나, ∞를 보고 도달 불가로 오해하면 틀리기 쉽다. 이 문제에서는 서울→평택은 직접 연결 30이 우회보다 짧고, 서울→대구는 직접 연결이 없지만 서울→기흥→대구 75가 가장 짧다는 점을 각각 따로 계산해야 한다.

다시 나오면 이렇게 판단

인접 행렬 최단 경로 문제는 출발지와 도착지를 하나씩 정해 가능한 짧은 우회 경로 몇 개를 먼저 적고, 직접 경로와 가중치 합을 비교하는 순서로 판단한다. 직접 연결이 없는 경우에는 연결된 중간 도시를 거치는 경로들의 합을 비교하면 된다.

NoteBase

암기 포인트

최단 경로값은 경로 수가 아니라 가중치 합의 최솟값이며, ∞는 직행 부재를 뜻할 뿐 우회 불가를 뜻하지 않는다.

NoteBase

문 9. 관계식 $T(n)$이 <보기>와 같을 때, 옳지 않은 것은?

<보기>

$T(n) = 2T(n/2) + n,\ \ T(0) = T(1) = 1$

① $T(n) = \Omega(n^2)$

② $T(n) = \Theta(n\log n)$

③ $T(n) = O(n^2)$

④ $T(n) = O(n\log n)$

정답 1번

출제 포인트

이 문제는 점화식 $T(n) = 2T(n/2) + n$에 마스터 정리를 적용하여 점근적 복잡도를 판별할 수 있는지를 묻는다.

조건 정리

주어진 식은 $T(n) = 2T(n/2) + n$이다.
여기서 $a = 2, b = 2, f(n) = n$이다.
비교 기준은 $n^{\log_b a} = n^{\log_2 2} = n$이다.

사용할 공식

마스터 정리에서 $T(n) = aT(n/b) + f(n)$이고 $f(n) = \Theta(n^{\log_b a})$이면 $T(n) = \Theta(n^{\log_b a}\log n)$이 된다.
이 문제는 $f(n) = n$이므로 $n^{\log_b a} = n$과 정확히 같은 차수인 경우에 해당한다.

계산 과정

1. $a = 2, b = 2$이므로 $n^{\log_b a} = n^{\log_2 2} = n$이다.
2. 추가 비용 $f(n) = n$과 비교하면 $f(n) = \Theta(n^{\log_b a})$가 성립한다.
3. 따라서 마스터 정리의 두 번째 경우이므로 $T(n) = \Theta(n\log n)$이다.

선지 판단

① X: $T(n) = \Theta(n\log n)$은 n^2보다 느리게 증가하므로 $\Omega(n^2)$가 아니다. 틀린 표현은 "$\Omega(n^2)$"이고, 올바르게는 "$O(n^2)$이면서 $\Omega(n\log n)$이다"이다.
② O: 위 계산 결과 그대로 $T(n) = \Theta(n\log n)$이므로 옳다.
③ O: $n\log n$은 충분히 큰 n에서 n^2보다 작으므로 $\Theta(n\log n) \subset O(n^2)$이다.
④ O: $T(n) = \Theta(n\log n)$이면 당연히 $O(n\log n)$도 성립하므로 옳다.

NoteBase

오답 포인트

발문이 "옳지 않은 것"이라는 점을 놓치면 ②를 고르고 끝내기 쉽다. 또 $\Theta(n \log n)$를 구한 뒤 포함관계까지 확인하지 않으면 ③과 ④의 참 여부를 잘못 판단하기 쉽다.

다시 나오면 이렇게 판단

점화식 문제는 먼저 a, b, $f(n)$를 분리하고 $n^{\log_b a}$와 $f(n)$를 비교한다. 그다음 구한 최종 차수로 Θ, O, Ω의 포함관계를 다시 확인하면 선지를 안정적으로 고를 수 있다.

암기 포인트

$f(n) = n^{\log_b a}$와 같은 차수이면 마스터 정리 결과는 $\Theta(n^{\log_b a} \log n)$이다.

NoteBase

문 10. <보기>는 정렬된 정수 배열 array에서 주어진 키 값(target)에 대한 이진 탐색(binary search)을 재귀(recursion) 방법을 통해 수행하는 함수 binarySearch()를 나타낸 것이다. (가)와 (나)에 들어갈 코드를 옳게 짝지은 것은?

〈보기〉

```
const int NOT_IN_ARRAY = -1;       // 배열에 없음
const int ARRAY_UNORDERED = -2; // 배열이 정렬되어 있지 않음
const int LIMITS_REVERSED = -3;   // 최대 색인이 최소 색인보다 큼

/* array[]: 입력 배열, lower: 이진 탐색 범위의 하단 색인, upper: 이진 탐색
범위의 상단 색인, target: 탐색하고자 하는 주어진 키 */
int binarySearch(int array[], int lower, int upper, int target) {
        int center, range;
        range = upper - lower;
        if ( range < 0 ) {
                return LIMITS_REVERSED;
        } else if( ( range == 0 ) && ( array[lower] != target ) ) {
                return NOT_IN_ARRAY;
        }
        if ( array[lower] > array[upper] )
                return ARRAY_UNORDERED;

        center = ( ( range ) / 2 ) + lower;
        if ( target == array[center] ) {
                return center;
        } else if ( target < array[center] ) {
                return _____(가)_____;
        } else {
                return _____(나)_____;
        }
}
```

① (가) binarySearch(array, lower - 1, center, target)
(나) binarySearch(array, center - 1, upper + 1, target)

② (가) binarySearch(array, lower, center - 1, target)
(나) binarySearch(array, center + 1, upper, target)

③ (가) binarySearch(array, center + 1, upper - 1, target)
(나) binarySearch(array, lower - 1, center, target)

④ (가) binarySearch(array, center + 1, upper, target)
(나) binarySearch(array, lower, center - 1, target)

정답 2번

출제 포인트

이 문제는 재귀 이진 탐색에서 비교 결과에 따라 탐색 구간의 하한과 상한을 어떻게 갱신하는지를 묻는다.

NoteBase

올바른 순서

1. range = upper - lower로 현재 탐색 구간의 크기를 계산한다.
2. range < 0이면 색인 범위가 뒤집힌 경우이므로 오류 코드를 반환한다.
3. 원소가 하나뿐인데 array[lower] != target이면 더 찾을 값이 없으므로 NOT_IN_ARRAY를 반환한다.
4. 중앙 위치 center를 구한 뒤 target == array[center]이면 해당 위치를 반환하고, 더 작으면 lower ~ center - 1, 더 크면 center + 1 ~ upper로 재귀 호출한다.

왜 이 순서: 중앙 원소를 이미 비교했으므로 다음 재귀에서는 center를 제외해야 하고, 비교 결과에 따라 남는 절반 구간으로만 범위를 줄여야 한다.

선지별 해설

① X: (가)의 binarySearch(array, lower - 1, center, target)는 하한을 현재 구간 밖으로 넓히고 이미 검사한 center를 다시 포함한다. (나)의 binarySearch(array, center - 1, upper + 1, target)도 범위를 줄이지 않고 오히려 확장한다. 틀린 표현은 (가) lower - 1, center, (나) center - 1, upper + 1이며, 올바르게는 (가) lower, center - 1, (나) center + 1, upper이다.

② O: target < array[center]이면 중앙보다 왼쪽 절반만 남으므로 (가)는 binarySearch(array, lower, center - 1, target)이어야 한다. target > array[center]이면 오른쪽 절반만 남으므로 (나)는 binarySearch(array, center + 1, upper, target)이어야 한다. 두 경우 모두 이미 비교한 center를 제외한다는 점이 맞다.

③ X: (가)의 binarySearch(array, center + 1, upper - 1, target)는 target < array[center]인데도 오른쪽 구간으로 가므로 비교 방향이 반대이다. (나)의 binarySearch(array, lower - 1, center, target)도 target > array[center]인데 왼쪽으로 가고 하한까지 잘못 넓힌다. 틀린 표현은 왼쪽/오른쪽 구간 선택과 경계 갱신 방식이며, 올바르게는 (가) lower, center - 1, (나) center + 1, upper이다.

④ X: (가)와 (나)가 서로 뒤바뀌었다. target < array[center]일 때는 왼쪽 구간 lower ~ center - 1로 가야 하고, target > array[center]일 때는 오른쪽 구간 center + 1 ~ upper로 가야 한다. 틀린 표현은 (가) center + 1, upper, (나) lower, center - 1이며, 올바르게는 두 재귀 호출의 위치가 서로 바뀌어야 한다.

오답 포인트

이진 탐색은 비교한 중앙 원소를 다시 포함하지 않고 탐색 범위를 반드시 절반으로 줄여야 한다. lower - 1, upper + 1처럼 범위를 넓히거나, 작은 경우와 큰 경우의 이동 방향을 바꾸면 바로 오답이 된다.

다시 나오면 이렇게 판단

재귀 이진 탐색은 비교 방향과 구간 축소를 함께 본다. target < array[center]이면 상한만 center - 1로 줄이고, target > array[center]이면 하한만 center + 1로 올리며, 이미 비교한 center는 다시 넣지 않는다.

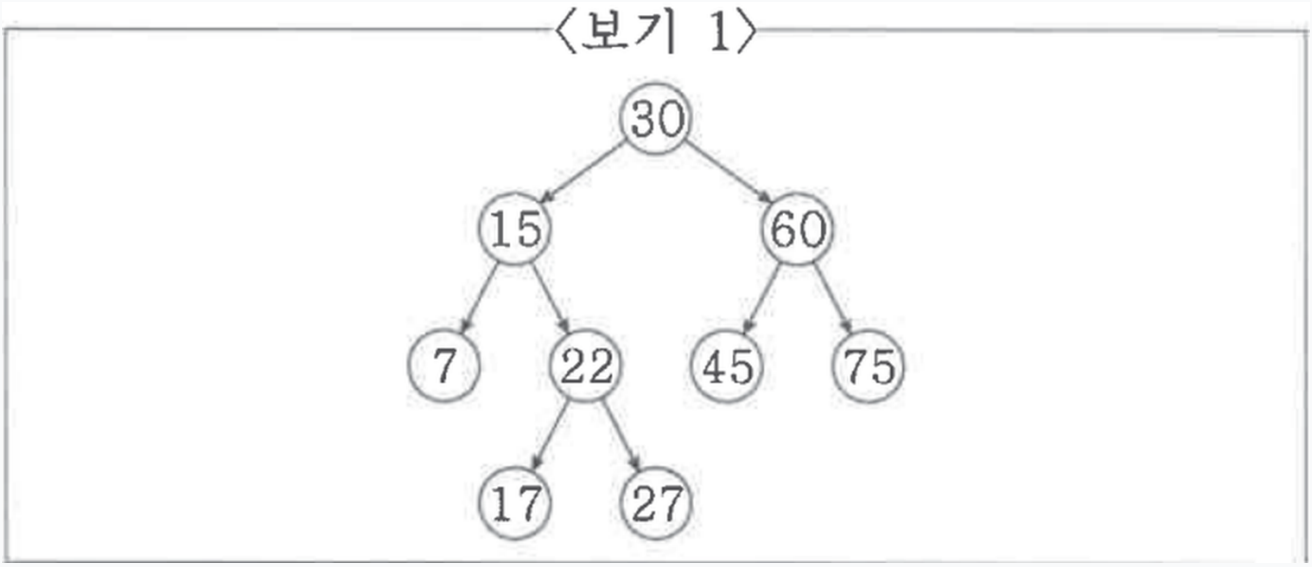
암기 포인트

작으면 왼쪽 lower ~ center - 1, 크면 오른쪽 center + 1 ~ upper이다.

NoteBase

문 11. <보기 1>의 이진트리(binary tree) T에 대한 설명으로 옳은 것을 <보기 2>에서 모두 고른 것은?

<보기 1>



<보기 2>

ㄱ. 노드 22의 형제(sibling)는 노드 45이다.

ㄴ. T의 전위(preorder) 순회 결과는 노드 30 15 7 22 17 27 60 45 75 순이다.

ㄷ. T의 중위(inorder) 순회 결과는 노드 7 15 17 22 27 30 45 60 75 순이다.

ㄹ. T의 레벨 순서(level-order) 순회 결과는 노드 30 15 60 7 22 45 75 17 27 순이다.

① ㄱ, ㄴ, ㄷ

② ㄱ, ㄴ, ㄹ

③ ㄴ, ㄷ, ㄹ

④ ㄱ, ㄴ, ㄷ, ㄹ

정답 3번

출제 포인트

이 문제는 이진트리 도식에서 부모-형제 관계와 전위·중위·레벨 순회 결과를 정확히 읽어 낼 수 있는지를 묻는다.

NoteBase

선지별 해설

① X: ①은 ㄱ, ㄴ, ㄷ을 모두 옳다고 본 조합이다. 그러나 ㄱ이 틀리므로 오답이다. 노드 22의 부모는 15이고, 같은 부모 15의 다른 자식은 7이므로 22의 형제는 7이다. 틀린 표현은 "노드 22의 형제는 노드 45이다"이고, 올바르게는 "노드 22의 형제는 노드 7이다"이다. ㄴ의 전위 순회 30 15 7 22 17 27 60 45 75와 ㄷ의 중위 순회 7 15 17 22 27 30 45 60 75는 맞다.
② X: ②는 ㄱ, ㄴ, ㄹ을 옳다고 본 조합이다. 그러나 ㄱ이 틀리므로 오답이다. ㄹ의 레벨 순서 30 15 60 7 22 45 75 17 27은 각 레벨을 왼쪽에서 오른쪽으로 방문한 결과로 맞다. 틀린 표현은 여전히 "노드 22의 형제는 노드 45이다"이고, 올바르게는 "노드 22의 형제는 노드 7이다"이다.
③ O: ③은 ㄴ, ㄷ, ㄹ만 옳다고 본 조합이다. 전위 순회는 루트-왼쪽-오른쪽이므로 30 15 7 22 17 27 60 45 75가 되고, 중위 순회는 왼쪽-루트-오른쪽이므로 7 15 17 22 27 30 45 60 75가 되며, 레벨 순회는 30 15 60 7 22 45 75 17 27이 된다. 따라서 ㄴ, ㄷ, ㄹ은 모두 옳다.
④ X: ④는 ㄱ, ㄴ, ㄷ, ㄹ이 모두 옳다고 본 조합이다. 그러나 ㄱ이 틀리므로 오답이다. 45는 60의 자식이어서 22와 부모가 다르다. 틀린 표현은 "형제는 노드 45"이고, 올바르게는 "형제는 같은 부모 15를 공유하는 노드 7"이다.

오답 포인트

22와 45가 그림에서 비슷한 높이에 보인다는 이유만으로 형제로 오인하면 틀린다. 형제 판단은 같은 레벨이 아니라 같은 부모인지로 결정된다. 또 순회 문제에서 전위·중위·레벨 순서의 기준을 섞으면 ㄴ, ㄷ, ㄹ 중 하나를 잘못 지우기 쉽다.

다시 나오면 이렇게 판단

먼저 각 노드의 부모-자식 관계를 표시한 뒤, 형제는 같은 부모를 가진 노드인지 확인한다. 그 다음 전위는 루트부터, 중위는 루트를 가운데에, 레벨 순회는 층별 왼쪽부터라는 기준으로 순서를 한 번씩 직접 써 보면 조합 선지를 바로 걸러낼 수 있다.

암기 포인트

형제는 같은 부모, 전위는 루트-왼쪽-오른쪽, 중위는 왼쪽-루트-오른쪽, 레벨 순회는 층별 왼쪽부터이다.

NoteBase

문 12. <보기>는 주어진 연결 리스트에서 끝에서부터 m번째에 위치한 요소(Element)를 반환하는 함수 findMToLastElement()를 C언어로 구현한 함수를 나타낸다. 이 함수에서 최종적으로 반환되는 mBehind 포인터는 끝에서부터 m번째에 위치한 요소를 가리킨다. (가)와 (나)에 들어갈 코드를 옳게 짝지은 것은?

〈보기〉

```
typedef struct Element{
        int data;
        struct Element* next;
} Element; //요소 구조체 정의

Element* findMToLastElement(Element *head, int m) {
/* head: 연결 리스트의 첫번째 요소를 가리키는 포인터, m: 연결
리스트의 m번째 요소 */
        Element *current, *mBehind;
        int i;
        current = head;
        mBehind = head;
        for (i = 0; i < m; i++) {
                if (current -> next) {
                        (가)
                } else {
                        return NULL;
                }
        }
        while (current -> next) {
                (가)
                (나)
        }
        return mBehind;
}
```

① (가) mBehind = mBehind -> next;, (나) head = head -> next;

② (가) current = current -> next;, (나) head = head -> next;

③ (가) mBehind = mBehind -> next;, (나) current = current -> next;

④ (가) current = current -> next;, (나) mBehind = mBehind -> next;

정답 4번

출제 포인트

이 문제는 연결 리스트에서 두 포인터의 간격을 먼저 만든 뒤, 같은 속도로 함께 이동시키는 절차를 묻는 문제이다.

NoteBase

올바른 순서

1. `current`와 `mBehind`를 모두 `head`로 둔다.
2. `for`문에서 `current`만 먼저 전진시켜 `mBehind`보다 앞서게 만든다.
3. 그 뒤 `while`문에서는 `current`와 `mBehind`를 한 칸씩 함께 이동시킨다.
4. `current`가 마지막 노드에 도달하면 `mBehind`가 끝에서부터 m번째 요소를 가리키게 된다.

왜 이 순서: 먼저 간격을 만든 다음 그 간격을 유지한 채 같이 이동해야 마지막에 `mBehind`가 원하는 위치에 남기 때문이다.

선지별 해설

① X: `for`문에서는 간격을 만드는 단계이므로 `mBehind = mBehind -> next;`가 아니라 `current = current -> next;`가 들어가야 한다. `mBehind`를 먼저 움직이면 두 포인터 사이 간격이 생기지 않는다. 또 `while`문에서 `head = head -> next;`를 쓰는 것도 틀리며, 올바르게는 `mBehind = mBehind -> next;`이다.

② X: (가) 자리에 `current = current -> next;`가 들어가는 점은 맞지만, (나)에 `head = head -> next;`를 넣으면 반환 대상 포인터를 추적하지 못한다. 틀린 표현은 `head = head -> next;`이고, 올바르게는 `mBehind = mBehind -> next;`이다.

③ X: `while`문에서 `current = current -> next;`는 필요하지만 `for`문에서 `mBehind = mBehind -> next;`를 먼저 넣는 순간 선행 간격이 사라진다. 틀린 표현은 (가)의 `mBehind = mBehind -> next;`이고, 올바르게는 `current = current -> next;`이다.

④ O: `for`문에서 `current = current -> next;`로 `current`를 먼저 앞세우고, `while`문에서 `mBehind = mBehind -> next;`를 함께 이동시키면 두 포인터 간 거리 유지가 가능하다. 문항의 의도대로 마지막에 `mBehind`가 끝에서부터 m번째 요소를 가리키게 된다.

오답 포인트

출제자는 `head`, `current`, `mBehind` 세 포인터 이름을 섞어 놓고 어떤 포인터가 실제 반환 대상을 추적하는지 헷갈리게 만든다. 특히 `head`는 초기 시작점일 뿐이고, 실제로 함께 이동해야 하는 포인터는 `mBehind`라는 점이 함정이다.

다시 나오면 이렇게 판단

연결 리스트의 뒤쪽 위치를 찾는 투 포인터 문제는 먼저 선행 포인터를 앞질러 보내 간격을 만든 뒤, 끝까지 두 포인터를 함께 이동시키는 구조인지 확인하면 된다.

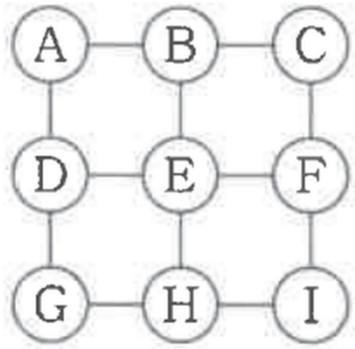
암기 포인트

뒤에서 m번째 탐색은 `current`가 먼저, 이후에는 `current`와 `mBehind`가 함께 이동한다.

NoteBase

문 13. <보기>에 주어진 무방향 그래프에서 깊이 우선 탐색 방식으로 방문할 수 있는 정점(vertex)들의 순서를 나열한 결과로 가장 옳지 않은 것은?

<보기>



① A-D-E-F-C-B-I-H-G

② A-D-G-H-E-B-C-F-I

③ A-D-G-H-E-F-I-B-C

④ E-D-G-H-I-F-C-B-A

정답 3번

출제 포인트

이 문제는 무방향 그래프에서 DFS의 핵심 성질인 '현재 정점에서 갈 수 있는 미방문 정점이 남아 있으면 끝까지 내려가고, 없을 때만 되돌아간다'를 순서로 판별하는 문제이다.

선지별 해설

① O: A→D→E→F→C까지 계속 내려간 뒤, C의 남은 미방문 인접 정점 B를 방문하고 되돌아와 F→I→H→G로 마무리하는 순서이므로 가능하다.

② O: A→D→G→H→E→B→C→F→I는 각 단계가 항상 현재 정점의 미방문 인접 정점으로 이어진다. C에서 F, F에서 I로 내려가며 DFS 규칙을 어기지 않는다.

③ X: A→D→G→H→E→F→I까지는 가능하지만, 그 다음에 B로 가는 부분이 불가능하다. I를 방문한 뒤에는 F로 되돌아오게 되는데, F에는 아직 미방문 인접 정점 C가 남아 있으므로 C를 먼저 방문해야 한다. 틀린 표현은 'I 다음에 B를 방문한다'는 부분이며, 올바르게는 A→D→G→H→E→F→I→C→B처럼 F의 남은 갈래를 먼저 처리하는 형태가 되어야 한다.

④ O: E→D→G→H→I→F→C→B→A는 시작 정점을 E로 두고 한 갈래를 끝까지 내려간 뒤 역방향으로 이어지는 미방문 인접 정점을 차례로 방문한 경우이므로 DFS 순서로 가능하다.

NoteBase

오답 포인트

③은 모든 연속 쌍이 간선으로 연결되어 있어 겉보기에는 그럴듯하지만, DFS는 단순히 '연결만 되면 되는' 순회가 아니다. F에서 I를 갔다가 돌아온 뒤에도 C가 남아 있는데 B로 건너뛴다고 착각하면 틀린다.

다시 나오면 이렇게 판단

DFS 순서를 볼 때는 연속한 두 정점의 연결 여부만 보지 않는다. 어떤 정점으로 되돌아왔을 때 그 정점에 미방문 인접 정점이 남아 있으면, 그 갈래를 먼저 모두 방문한 뒤에야 더 위로 backtracking할 수 있다고 판단하면 된다.

암기 포인트

DFS는 현재 갈래를 끝까지 판 뒤에야 옆갈래로 넘어가는 탐색이다.

NoteBase

문 14. <보기>에서 함수 f()의 시간복잡도는?

〈보기〉

```
int f(int n)
{
   int count = 0;
   for (int i = n; i > 0; i /= 2)
      for (int j = 0; j < i; j++)
         count += 1;
   return count;
}
```

① $\Theta(n)$

② $\Theta(n \log n \log n)$

③ $\Theta(n \log n)$

④ $\Theta(n^2)$

정답 1번

출제 포인트

바깥 반복문이 $n, n/2, n/4, \ldots$로 줄어들 때 안쪽 반복문의 총 실행 횟수를 등비수열의 합으로 판단하는 문제이다.

조건 정리

주어진 값: 바깥 반복문은 $i = n$에서 시작해 매 반복마다 $i /= 2$로 절반이 된다.
구해야 할 값: `count += 1`의 총 실행 횟수이다.
주의할 조건: 바깥 반복문 횟수만 $\log n$이라고 보고 전체를 $\Theta(n \log n)$로 곱하면 안 된다. 안쪽 반복문의 길이도 함께 줄어든다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 자료구조론 · 2022 · 10월

사용할 공식

총 실행 횟수는
$T(n) = n + \frac{n}{2} + \frac{n}{4} + \cdots + 1$
이다.
이 공식을 쓰는 이유는 바깥 반복문이 한 번 돌 때마다 안쪽 반복문이 현재 i만큼 실행되고, 그 i가 매번 절반으로 감소하기 때문이다.

계산 과정

1. 첫 번째 바깥 반복에서는 안쪽 반복문이 n번 실행된다.
2. 다음 반복들에서는 $n/2, n/4, n/8, \ldots$번씩 실행된다.
3. 따라서 전체 실행 횟수는
$T(n) = n + \frac{n}{2} + \frac{n}{4} + \cdots < 2n$
이다.
4. 또한 첫 항만 보아도 $T(n) \geq n$이다.
5. 상한과 하한이 모두 n에 비례하므로 $T(n) = \Theta(n)$이다.

선지 판단

① O이다. 총 실행 횟수가 $n + n/2 + n/4 + \cdots$이므로 $\Theta(n)$이 맞다.
② X이다. $\Theta(n \log n \log n)$이 되려면 로그 요인이 두 번 생기는 구조가 있어야 하는데, 이 코드는 그런 중첩이 아니다. 틀린 표현은 로그가 두 번 붙는다는 해석이며, 올바르게는 등비수열의 합이 선형으로 수렴한다.
③ X이다. 바깥 반복문 횟수만 $\Theta(\log n)$이라고 보고 안쪽 반복문을 매번 n번으로 고정해 곱하면 나오는 오답이다. 틀린 표현은 안쪽 반복문이 항상 n번 돈다는 가정이며, 올바르게는 안쪽 반복문 길이가 i와 함께 절반으로 줄어든다.
④ X이다. $\Theta(n^2)$이 되려면 안쪽 반복문이 대체로 n번씩 n회 가까이 반복되어야 한다. 틀린 표현은 중첩 반복문이면 항상 제곱 시간이라는 판단이며, 올바르게는 반복 변수의 감소 방식까지 함께 봐야 한다.

오답 포인트

③처럼 바깥 반복문이 $\log n$회라는 사실만 보고 안쪽 반복문의 초기 길이 n을 그대로 곱하면 $\Theta(n \log n)$로 오판하기 쉽다. 이 문제의 함정은 안쪽 반복 횟수도 $n, n/2, n/4, \ldots$로 함께 감소한다는 점이다.

다시 나오면 이렇게 판단

반복문이 중첩되어 있어도 안쪽 반복 횟수가 바깥 변수와 함께 줄어드는지 먼저 본다. 매 단계 실행 횟수를 항으로 적어 합이 등비수열인지 확인하면 시간복잡도를 빠르게 판단할 수 있다.

암기 포인트

절반씩 줄어드는 반복에서 각 단계 비용이 $n, n/2, n/4, \ldots$이면 전체 합은 $\Theta(n)$이다.

NoteBase

문 15. 8개의 정수를 퀵 정렬(quick sort) 알고리즘으로 정렬한다. 첫 번째 분할(partition) 연산을 수행한 결과가 <보기>와 같다고 할 때 설명으로 가장 옳은 것은?

<보기>
2 5 1 8 10 13 12 11

① 피봇으로 사용된 것은 8 또는 10이다.
② 피봇이 8일 수는 있지만, 10은 아니다.
③ 8은 피봇이 아니고, 10은 피봇이 될 수 있다.
④ 8과 10은 피봇이 아니다.

정답 1번

출제 포인트

이 문제는 퀵정렬의 첫 분할 결과에서 피벗이 만족해야 하는 위치 조건을 판단할 수 있는지를 묻는다.

핵심 개념

분할이 끝난 피벗은 왼쪽에 자신보다 작은 값들만, 오른쪽에 자신보다 큰 값들만 오게 되는 원소이다.
첫 분할 결과만 보고는 피벗이 하나로 고정되지 않을 수 있으며, 이 조건을 만족하는 원소가 여러 개이면 모두 가능하다.

선지별 해설

1번: O. 8의 왼쪽에는 2, 5, 1이 있어 모두 8보다 작고, 오른쪽에는 10, 13, 12, 11이 있어 모두 8보다 크다. 10의 왼쪽에는 2, 5, 1, 8이 있어 모두 10보다 작고, 오른쪽에는 13, 12, 11이 있어 모두 10보다 크다. 따라서 8 또는 10이 피벗일 수 있다.
2번: X. 틀린 부분은 "10은 아니다"이다. 10도 왼쪽 원소가 모두 10보다 작고 오른쪽 원소가 모두 10보다 크므로 피벗이 될 수 있다. 올바르게는 "피벗은 8일 수도 있고 10일 수도 있다"이다.
3번: X. 틀린 부분은 "8은 피벗이 아니고"이다. 8도 분할 조건을 만족하므로 피벗이 될 수 있다. 올바르게는 "8과 10은 모두 피벗이 될 수 있다"이다.
4번: X. 틀린 부분은 "8과 10은 피벗이 아니다"이다. 8과 10은 각각 자신의 왼쪽에는 더 작은 값만, 오른쪽에는 더 큰 값만 두고 있으므로 모두 피벗 가능하다. 올바르게는 "8과 10은 모두 피벗이 될 수 있다"이다.

오답 포인트

피벗이 반드시 하나만 정해진다고 생각하면 8이나 10 중 하나를 배제하게 된다. 이 문항은 "분할 후 피벗의 위치 조건"만 확인하면 되는데, 실제 초기 배열이나 구현 방식을 상상하다가 오답으로 흐르기 쉽다.

NoteBase

다시 나오면 이렇게 판단

결과 배열의 각 원소를 기준으로 왼쪽이 전부 더 작은지, 오른쪽이 전부 더 큰지를 확인한다. 이 조건을 만족하는 원소는 그 분할의 피벗이 될 수 있다.

암기 포인트

퀵정렬 분할의 피벗은 왼쪽은 작게, 오른쪽은 크게 가르는 경계 원소이다.

NoteBase

문 16. 비어있는 스택에 <보기 1>에 주어진 숫자들을 순서대로 <보기 2>의 규칙을 따라 처리할 때, 숫자들을 모두 처리한 후 스택에 저장된 숫자들을 모두 pop을 이용하여 출력한 결과로 가장 옳은 것은? (단, <보기 2>의 top은 가장 최근에 스택에 push된 값을 가리킨다.)

<보기 1>

1, 5, 0, 0, -2, 6, 1, -2, 0, 10

<보기 2>

• 처리하려는 숫자를 K라고 하자.

• K가 0이 아닌 경우 아래의 조건 가운데 해당하는 항목을 수행한다.

- 스택이 비어있거나 top이 가리키는 숫자가 K보다 큰 경우: K를 push 한다.

- top이 가리키는 숫자가 K보다 작거나 같은 경우: pop한 뒤 추출된 숫자에 K를 더한 값을 push 한다.

• K가 0인 경우 아래 조건 가운데 해당하는 항목을 수행한다.

- 스택이 비어있는 경우: 다음 숫자를 처리한다.

- 스택이 비어있지 않은 경우: pop한다.

① 10

② 10 -2 1 4

③ 10 6 3 1

④ 11 4

정답 4번

출제 포인트

이 문제는 스택의 top과 현재 값 K를 비교하여 push, pop, 합산 push를 순서대로 적용한 뒤 최종 pop 출력 순서를 묻는 문제이다.

NoteBase

올바른 순서

1. 1 처리 -> 스택이 비어 있으므로 1을 push하여 [1]이 된다.
2. 5 처리 -> top인 1이 5보다 작거나 같으므로 1을 pop하고 $1 + 5 = 6$을 push하여 [6]이 된다.
3. 0, 0 처리 -> 첫 번째 0에서 6을 pop하여 []가 되고, 두 번째 0은 스택이 비어 있으므로 그대로 넘어간다.
4. -2 처리 -> 스택이 비어 있으므로 -2를 push하여 [-2]가 된다.
5. 6 처리 -> top인 -2가 6보다 작거나 같으므로 -2를 pop하고 $-2 + 6 = 4$를 push하여 [4]가 된다.
6. 1, -2 처리 -> 1은 top 4보다 작으므로 push하여 [4, 1], -2도 top 1보다 작으므로 push하여 [4, 1, -2]가 된다.
7. 0 처리 -> top인 -2를 pop하여 [4, 1]이 된다.
8. 10 처리 -> top인 1이 10보다 작거나 같으므로 1을 pop하고 $1 + 10 = 11$을 push하여 [4, 11]이 된다.
9. 마지막 pop 출력 -> top부터 꺼내므로 11, 4 순서로 출력된다.

왜 이 순서인가: top이 K보다 작거나 같을 때는 기존 top을 유지하지 않고 pop한 뒤 합산값 하나로 바꾸므로, 중간 상태를 누적해서 따라가야 최종 출력이 결정된다.

선지별 해설

① X

결과를 10으로 본 것은 마지막 10만 남는다고 본 경우이다. 실제로는 마지막 단계에서 1과 10이 합쳐져 11이 되고, 그 아래의 4도 그대로 남아 있으므로 출력은 10이 아니다. 틀린 표현은 "10"이고, 올바르게는 "11 4"이다.

② X

중간에 들어간 값들을 단순 나열한 결과이다. 이 규칙은 0에서 pop이 일어나고, top이 K보다 작거나 같을 때는 pop 후 합산값 하나만 다시 push한다. 따라서 -2와 1은 최종 출력 시점까지 그대로 남지 않는다. 틀린 표현은 "10 -2 1 4"이고, 올바르게는 "11 4"이다.

③ X

6과 1이 따로 남는다고 본 경우인데, 6은 이후 0에서 pop되어 사라지고 1은 마지막 10과 합쳐져 11이 된다. 또한 pop 출력은 top부터이므로 남는 값의 순서도 11, 4이다. 틀린 표현은 "10 6 3 1"이고, 올바르게는 "11 4"이다.

④ O

규칙을 끝까지 적용하면 최종 스택은 [4, 11]이다. 스택은 top부터 pop하므로 출력 결과는 11, 4가 된다.

오답 포인트

0이 나오면 무조건 0을 저장한다고 착각하거나, top이 K보다 작거나 같을 때 기존 값과 K가 둘 다 남는다고 착각하기 쉽다. 이 문항의 핵심 함정은 "pop 후 합산값 하나만 다시 push한다"는 규칙과 "최종 출력은 top부터"라는 점이다.

NoteBase

다시 나오면 이렇게 판단

스택 규칙 문제는 입력값마다 현재 스택 상태를 한 줄씩 갱신하면서 본다. 특히 0은 삭제 규칙인지 확인하고, 비교 조건이 성립할 때 기존 top이 유지되는지 사라지는지부터 판단하면 최종 출력 순서를 안정적으로 구할 수 있다.

암기 포인트

이 문제의 규칙은 "0은 pop, top ≤ K이면 pop 후 합산 push, top > K이면 그대로 push"로 정리된다.

NoteBase

문 17. n개의 정점(vertex) $V_0, \cdots, V_{n-1}$를 갖는 방향 그래프(directed graph)를 인접 행렬(adjacency matrix)로 표현할 때, 정점 V_x에서부터 V_y로의 간선(edge) (V_x, V_y)가 있으면 인접 행렬 $a[x][y]$에 1을 저장하고, 그렇지 않은 경우는 0을 저장한다. 정점 V_i의 진입차수(in-degree)와 진출차수(out-degree)를 계산하는 코드는?

①
```
in_degree = out_degree = 0;
for(j = 0; j <= n - 1; j++)
    in_degree = in_degree + a[ j ][ i ];
for(j = 0; j <= n - 1; j++)
    out_degree = out_degree + a[ i ][ j ];
```

②
```
in_degree = out_degree = 0;
for(j = 0; j <= n - 1; j++)
    in_degree = in_degree + a[ i ][ j ];
for(j = 0; j <= n - 1; j++)
    out_degree = out_degree + a[ j ][ i ];
```

③
```
in_degree = out_degree = 0;
for(i = 0; i <= n - 1; i++)
    in_degree = in_degree + a[ i ][ j ];
for(i = 0; i <= n - 1; i++)
    out_degree = out_degree + a[ j ][ i ];
```

④
```
in_degree = out_degree = 0;
for(i = 0; i++; i <= n - 1)
    in_degree = in_degree + a[ j ][ i ];
for(i = 0; i++; i <= n - 1)
    out_degree = out_degree + a[ i ][ j ];
```

정답 1번

출제 포인트

인접 행렬에서 방향 그래프의 진입차수는 열의 합으로, 진출차수는 행의 합으로 계산한다는 점을 묻는 문제이다.

핵심 개념

진입차수는 다른 정점에서 V_i로 들어오는 간선 수이다.
진출차수는 V_i에서 다른 정점으로 나가는 간선 수이다.
인접 행렬에서 $a[x][y]$는 V_x에서 V_y로 가는 간선의 존재를 뜻하므로, 열 i의 합은 V_i의 진입차수이고 행 i의 합은 V_i의 진출차수이다.

NoteBase

선지별 해설

① O이다. 첫 번째 반복문은 $a[j][i]$를 모두 더하므로 열 i의 합을 구해 V_i의 진입차수를 계산한다. 두 번째 반복문은 $a[i][j]$를 모두 더하므로 행 i의 합을 구해 V_i의 진출차수를 계산한다.
② X이다. 틀린 부분은 진입차수와 진출차수의 행·열을 서로 바꾼 점이다. 올바르게는 진입차수는 $a[j][i]$의 합이고, 진출차수는 $a[i][j]$의 합이다.
③ X이다. 틀린 부분은 특정 정점 V_i의 차수를 구해야 하는데 반복 변수까지 i로 두어 인덱스의 역할이 뒤섞인 점이다. 이 코드는 기준 정점 i를 고정하지 못한다. 올바르게는 j를 0부터 $n-1$까지 변화시키며 $a[j][i]$와 $a[i][j]$를 각각 더해야 한다.
④ X이다. 틀린 부분은 for문의 형식이 잘못되어 조건식과 증감식의 위치가 바뀐 점이다. 또한 반복 변수도 ①처럼 j를 순회해야 한다. 올바르게는 for (j = 0; j <= n - 1; j++) 형태로 열과 행을 각각 합해야 한다.

오답 포인트

②처럼 행과 열의 의미를 뒤집으면 진입차수와 진출차수가 서로 바뀐다. ③은 고정해야 할 정점 인덱스 i를 반복 변수로 다시 사용한다는 점이 함정이고, ④는 for문의 기본 형식 자체가 깨져 있다는 점을 함께 확인해야 한다.

다시 나오면 이렇게 판단

방향 그래프의 인접 행렬이 나오면 먼저 $a[x][y]$가 출발 정점 x, 도착 정점 y를 뜻하는지 확인한다. 그 다음 들어오는 간선은 열의 합, 나가는 간선은 행의 합이라는 기준으로 각 선택지를 판별한다.

암기 포인트

방향 그래프 인접 행렬은 열의 합이 진입차수이고 행의 합이 진출차수이다.

NoteBase

문 18. <보기>는 8개의 정수로 된 초기 입력 자료 40, 25, 71, 35, 17, 43, 95, 68을 오름차순으로 정렬한 결과이다. 이때 적용한 정렬 방법은?

<보기>
초기 입력 자료: 40 25 71 35 17 43 95 68
<<<<<<<<< 정렬 수행 과정 >>>>>>>>>
과정1 >> 25 40 71 35 17 43 95 68
과정2 >> 25 40 35 71 17 43 95 68
과정3 >> 25 35 40 71 17 43 95 68
과정4 >> 25 35 40 71 17 43 95 68
과정5 >> 25 35 40 71 17 43 68 95
과정6 >> 25 35 40 71 17 43 68 95
과정7 >> 17 25 35 40 43 68 71 95

① 삽입 정렬(insertion sort)
② 퀵 정렬(quick sort)
③ 버블 정렬(bubble sort)
④ 병합 정렬(merge sort)

정답 4번

출제 포인트

이 문제는 정렬 과정이 작은 정렬 블록의 생성과 병합 순서로 진행되는지를 보고 병합 정렬의 흐름을 판별하는 문제이다.

올바른 순서

1. 인접한 원소 2개씩을 정렬된 작은 블록으로 만든다. [40, 25]는 [25, 40], [71, 35]는 [35, 71], [17, 43]은 그대로, [95, 68]은 [68, 95]가 된다.
2. 앞쪽의 두 정렬 블록 [25, 40]과 [35, 71]을 병합하여 [25, 35, 40, 71]을 만든다.
3. 뒤쪽의 두 정렬 블록 [17, 43]과 [68, 95]를 병합하여 [17, 43, 68, 95]를 만든다.
4. 마지막으로 두 개의 길이 4 정렬 블록 [25, 35, 40, 71]과 [17, 43, 68, 95]를 병합하여 전체 오름차순 [17, 25, 35, 40, 43, 68, 71, 95]를 만든다.

왜 이 순서인가: 병합 정렬은 뒤 단계의 병합이 앞 단계에서 이미 정렬된 부분 배열을 입력으로 요구하므로, 부분 정렬이 먼저 끝나야 다음 병합이 가능하다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 자료구조론 · 2022 · 10월

선지별 해설

① X
삽입 정렬은 왼쪽의 정렬 구간이 한 칸씩 커지면서 새 원소를 알맞은 위치에 끼워 넣는 방식이다. 제시된 과정에서는 길이 2 블록, 길이 4 블록처럼 정렬된 부분 배열을 만든 뒤 병합하고 있으므로 흐름이 다르다.
틀린 표현: 삽입 정렬
올바르게는: 삽입 정렬이라면 과정4 이전에 17이 이미 앞쪽 정렬 구간 안으로 삽입되어 맨 앞으로 이동하는 모습이 나타나야 한다.

② X
퀵 정렬은 피벗을 기준으로 작은 값과 큰 값을 좌우로 분할한 뒤 각 부분을 다시 재귀적으로 정렬한다. 제시된 과정에는 피벗 중심의 분할 흔적이 없고, 인접한 작은 블록을 차례로 병합하는 모습이 나타난다.
틀린 표현: 퀵 정렬
올바르게는: 퀵 정렬이라면 특정 피벗을 기준으로 좌우 분할 결과가 먼저 드러나야 한다.

③ X
버블 정렬은 한 번의 패스마다 인접 원소를 계속 교환하여 가장 큰 값이 맨 뒤로 이동한다. 실제 버블 정렬의 첫 패스라면 71이 17, 43, 68과 계속 비교되며 뒤로 밀려야 하는데, 제시된 과정은 중간에서 멈추지 않고 부분 배열 병합으로 바뀐다.
틀린 표현: 버블 정렬
올바르게는: 버블 정렬이라면 첫 패스 종료 시점에 전체 배열을 따라 인접 교환이 연속해서 일어난 결과가 보여야 한다.

④ O
과정1, 2, 4, 5는 길이 2 부분 배열을 정렬하는 단계이고, 과정3과 6은 정렬된 길이 2 블록을 길이 4 블록으로 병합하는 단계이며, 과정7은 두 길이 4 블록을 최종 병합하는 단계이다. 이는 병합 정렬의 전형적인 진행 방식이다.

오답 포인트

과정1에서 40과 25가 바뀌고 과정5에서 95와 68이 바뀌는 모습만 보고 인접 교환 정렬이라고 오해하기 쉽다. 그러나 버블 정렬이나 삽입 정렬은 전체 배열을 한 방향으로 훑으며 정렬 구간을 넓혀 가는 반면, 이 문제는 길이 2, 길이 4처럼 정렬된 블록을 만든 뒤 병합한다는 점이 결정적 차이이다.

다시 나오면 이렇게 판단

정렬 과정 문제는 먼저 중간 결과가 전체 배열을 한 번씩 훑는 방식인지, 아니면 정렬된 부분 배열을 만든 뒤 합치는 방식인지부터 본다. 부분 배열의 크기가 2, 4, 8처럼 커지며 합쳐지면 병합 정렬로 판단할 수 있다.

NoteBase

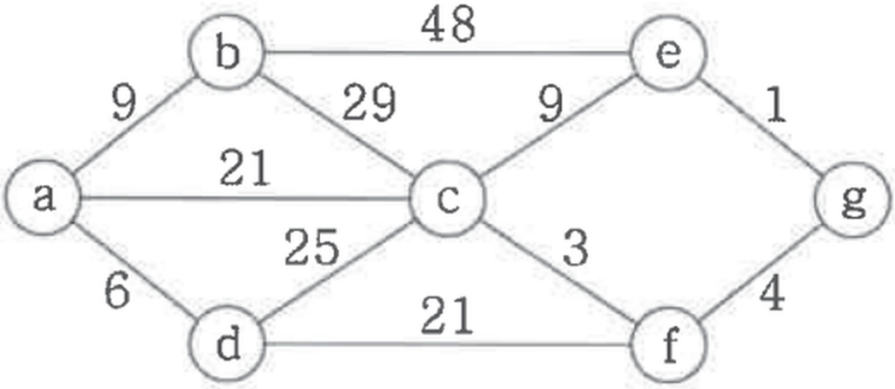
암기 포인트

병합 정렬은 잘게 나누어 정렬하고, 정렬된 블록끼리 합치며 완성하는 정렬이다.

NoteBase

문 19. <보기>의 그래프에서 정점(vertex) a를 시작 노드로 하여 Prim 알고리즘을 적용해 최소 신장 트리(minimum spanning tree)를 생성할 때 포함되는 간선(edge)을 순서대로 바르게 나열한 것은?

〈보기〉



① (a, d), (a, b), (d, f), (f, c), (f, g), (g, e)

② (a, b), (a, d), (d, f), (f, g), (g, e), (f, c)

③ (a, d), (a, b), (a, c), (c, f), (g, e), (f, g)

④ (e, g), (c, f), (f, g), (a, d), (a, b), (a, c)

정답 1번

출제 포인트

이 문제는 Prim 알고리즘에서 현재 트리와 연결되는 경계 간선 중 최소 가중치 간선을 반복 선택하는 순서를 묻는 문제이다.

올바른 순서

시작 집합은 {a}이다.

1. (a, d) - 시작 정점 a에서 나가는 간선 중 최소 가중치는 6이다.
2. (a, b) - 현재 경계 간선은 (a, b)=9, (a, c)=21, (d, c)=25, (d, f)=21이고 이 중 최소는 9이다.
3. (d, f) - 경계 간선 최소값이 21로 동률이며, 제시 보기의 정답 흐름은 (d, f)를 먼저 선택한 경우이다.
4. (f, c) - f가 추가되면 새 경계 간선 (f, c)=3이 가장 작다.
5. (f, g) - 다음 최소는 (f, g)=4이다.
6. (g, e) - g가 트리에 들어온 뒤에야 (g, e)=1이 후보가 되며 마지막으로 e가 연결된다.

왜 이 순서인가: Prim은 매 단계에서 현재 트리와 트리 밖 정점을 잇는 경계 간선만 비교한다. 따라서 전역 최소인 (g, e)=1도 g가 트리에 들어오기 전에는 선택할 수 없다.

NoteBase

선지별 해설

① O
(a, d)와 (a, b)를 먼저 고른 뒤 (d, f)를 선택하면 새 후보인 (f, c)=3, (f, g)=4, (g, e)=1이 차례로 열리므로 제시 순서가 성립한다.

② X
틀린 부분은 첫 간선을 (a, b)로 둔 점이다. 시작 정점 a에서 바로 선택할 수 있는 최소 간선은 (a, d)=6이므로 올바르게는 맨 앞이 (a, d)이어야 한다.

③ X
틀린 부분은 마지막 두 간선의 선후관계이다. g는 (f, g)가 먼저 선택되어 트리에 포함된 뒤에야 (g, e)가 후보가 되므로 올바르게는 … , (f, g), (g, e) 순서이다.

④ X
틀린 부분은 (e, g)를 첫 간선으로 둔 점이다. Prim은 시작 정점 a가 포함된 현재 트리에서 바깥으로 나가는 간선만 후보로 삼으므로 올바르게는 첫 간선이 (a, d)이며, (e, g)는 초기에 후보 자체가 될 수 없다.

오답 포인트

가중치가 가장 작은 간선만 전역적으로 찾는 방식으로 오해하면 (g, e)=1을 너무 일찍 고르게 된다. Prim은 전체 그래프의 최소 간선이 아니라 현재 트리의 경계 최소 간선을 고른다는 점이 함정이다. 또한 21의 동률이 보이더라도 그 다음 단계에서 새로 열린 더 작은 간선이 연쇄적으로 선택된다는 흐름을 함께 봐야 한다.

다시 나오면 이렇게 판단

Prim 순서 문제는 현재 트리에 포함된 정점을 먼저 표시하고, 그 집합에서 바깥으로 나가는 간선만 후보로 적어 최소값을 하나씩 고르는 방식으로 판단한다. 아직 트리에 닿지 않은 두 정점 사이의 간선은 가중치가 작아도 선택할 수 없다고 보면 된다.

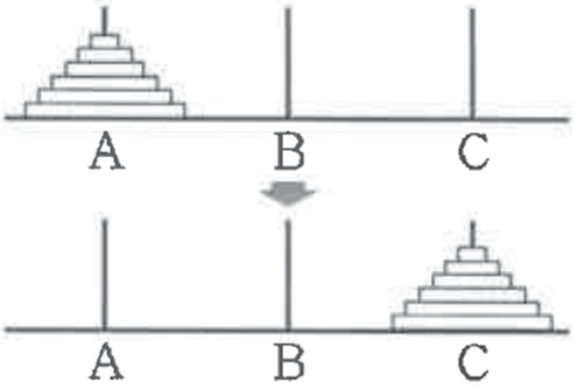
암기 포인트

Prim은 전역 최소가 아니라 현재 트리의 경계 최소를 고르는 알고리즘이다.

NoteBase

문 20. <보기 1>과 같이 세 개의 막대 A, B, C가 주어져 있고, 막대 A에는 N개의 크기가 서로 다른 원판이 큰 것부터 아래에 놓이도록 차례로 쌓여 있다. 한 막대의 맨 위에 있는 원판 하나를 꺼내 다른 막대의 맨 위에 놓을 수 있는데, 이때 작은 원판 위에 큰 원판을 올려놓을 수 없다. 막대 A에 있는 원판을 모두 막대 C로 옮기는 방법을 <보기 2>와 같이 재귀함수로 작성하고자 할 때, <보기 2>의 (가)에 들어갈 내용은? (단, N은 3 이상이다. hanoi(N, start, to, via)는 막대 start의 맨 위에 있는 N개의 원판을 막대 to로 옮기는 함수이고, move(1, start, to)는 start의 맨 위 원판 1개를 to로 옮기는 함수이다.)

〈보기 1〉



〈보기 2〉

```
void hanoi(int N, int A, int C, int B){
    if(N == 1)
        move(1, A, C);
    else {
        (가)
    }
}
```

① hanoi(N-1, A, B, C); move(1, A, B); hanoi(N-1, B, C, A);

② hanoi(N-1, A, B, C); move(1, A, C); hanoi(N-1, B, C, A);

③ hanoi(N-1, A, C, B); move(1, A, B); hanoi(N-1, C, A, B);

④ move(1, A, B); hanoi(N-1, A, C, B); move(1, B, C);

정답 2번

NoteBase

출제 포인트

이 문제는 하노이 탑 재귀 함수에서 시작 막대, 목적 막대, 보조 막대의 역할을 올바르게 바꾸며 호출하는 순서를 묻는 문제이다.

올바른 순서

1. hanoi(N-1, A, B, C) — 가장 큰 원판 위의 N-1개 원판을 보조 막대 B로 옮긴다.
2. move(1, A, C) — 가장 큰 원판 1개를 A에서 C로 옮긴다.
3. hanoi(N-1, B, C, A) — B에 임시로 옮겨 둔 N-1개 원판을 C로 옮긴다.

왜 이 순서인가: 가장 큰 원판을 A에서 C로 옮기려면 먼저 그 위의 N-1개가 B로 비켜 있어야 하고, 가장 큰 원판이 C로 간 뒤에야 B의 N-1개를 C 위에 다시 쌓을 수 있다.

선지별 해설

① X

첫 두 단계 중 move(1, A, B)가 틀리다. N-1개를 B로 옮긴 뒤에는 가장 큰 원판이 A에 남으므로 그 원판은 목적지 C로 가야 한다. 틀린 표현은 move(1, A, B)이고, 올바르게는 move(1, A, C)이다.

② O

hanoi(N-1, A, B, C)로 작은 원판 N-1개를 보조 막대 B로 옮기고, move(1, A, C)로 가장 큰 원판을 목적지 C로 보낸 뒤, hanoi(N-1, B, C, A)로 작은 원판들을 다시 C로 옮기므로 하노이 탑의 표준 재귀 순서와 일치한다.

③ X

첫 호출 hanoi(N-1, A, C, B)가 틀리다. 가장 큰 원판을 아직 옮기지 않았는데 작은 원판들을 미리 C로 옮기면 목적지 C가 비어 있지 않아 가장 큰 원판을 C에 놓는 단계가 성립하지 않는다. 또 move(1, A, B)도 목적지가 잘못되었다. 올바르게는 hanoi(N-1, A, B, C); move(1, A, C); hanoi(N-1, B, C, A)이다.

④ X

처음의 move(1, A, B)가 틀리다. 보조 막대 B를 먼저 작은 원판으로 점유하면, 이어지는 hanoi(N-1, A, C, B)에서 더 큰 원판을 B에 임시 저장해야 하는 순간 규칙에 어긋나게 된다. 틀린 표현은 첫 단계의 move(1, A, B)이며, 올바르게는 먼저 hanoi(N-1, A, B, C)를 수행해 작은 원판 전체를 B로 옮겨야 한다.

오답 포인트

출제자가 노린 함정은 move 한 번의 출발점과 도착점만 외우고 재귀호출의 의미를 놓치는 데 있다. 특히 ①은 가장 큰 원판의 목적지가 C라는 점을 놓치기 쉽고, ③은 작은 원판을 최종 목적지 C로 먼저 보내도 된다고 착각하기 쉽고, ④는 보조 막대를 너무 일찍 점유해 이후 재귀 단계가 막힌다는 점을 놓치기 쉽다.

NoteBase

다시 나오면 이렇게 판단

하노이 탑 재귀식은 항상 작은 원판 N-1개를 보조 막대로 치운다 → 가장 큰 원판 1개를 목적 막대로 옮긴다 → 치워 둔 N-1개를 목적 막대로 다시 옮긴다의 세 단계로 판단한다. 매 호출에서 start, to, via 자리에 어떤 막대가 들어가는지만 추적하면 된다.

암기 포인트

하노이 탑은 작은 것 치우기 → 큰 것 이동 → 작은 것 다시 쌓기의 3단계 재귀 구조이다.

NoteBase