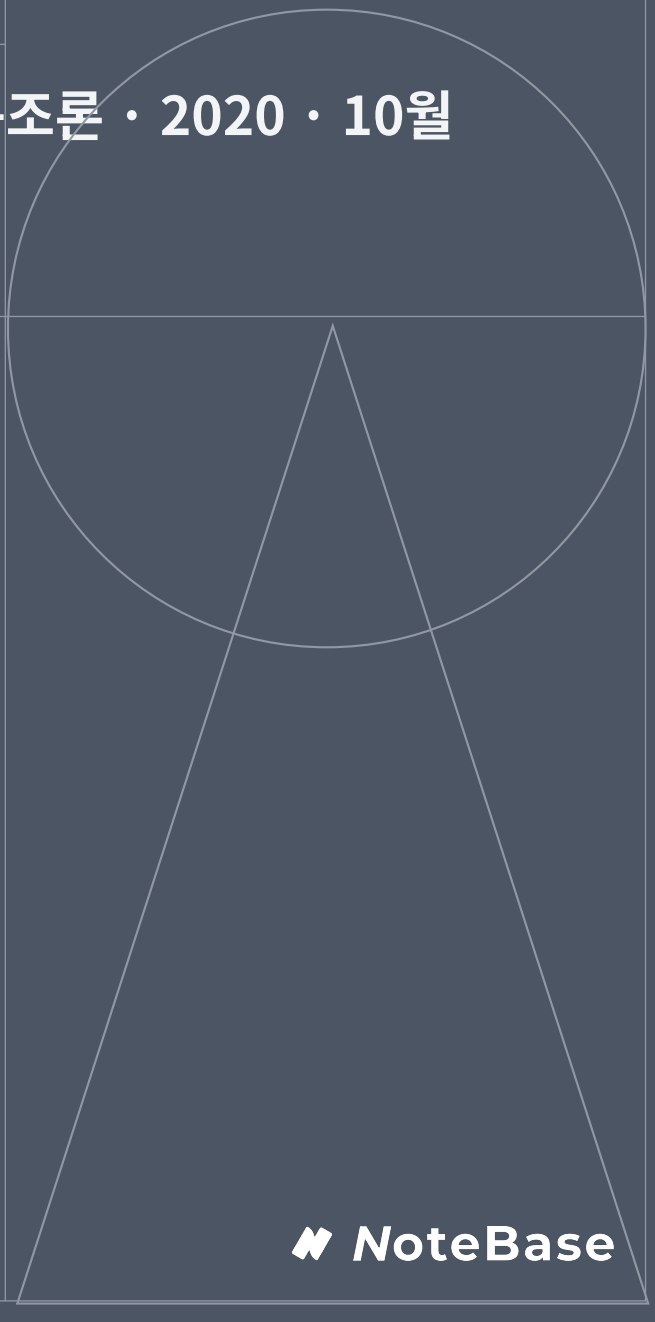
기출OX 해설집

공무원/서울시/7급 · 자료구조론 · 2020 · 10월



NoteBase

기출OX 해설집

공무원/서울시/7급 · 자료구조론 · 2020 · 10월

공무원/서울시/7급 · 자료구조론 · 2020 · 10월

문 1. <보기>의 C언어 함수(func)에서 func(4)+func(9)를 수행한 결과 값은?

<보기>

```
int func(int n) {
  int res = 0;
  if(n <= 1) res = n;
  else {
    int fn, fnm2 = 0, fnm1 = 1;
    for(int i = 2; i <= n; i++) {
      fn = fnm1 + fnm2;
      fnm2 = fnm1;
      fnm1 = fn;
    }
    res = fn;
  }
   return res;
}
```

① 23

② 37

③ 58

④ 94

정답 2번

출제 포인트

반복문으로 구현한 피보나치 수열 함수의 반환값을 추적하여 두 함수값의 합을 구하는 문제이다.

NoteBase

조건 정리

주어진 값: func(4), func(9)
구해야 할 값: func(4)+func(9)
주의할 조건: $n \le 1$이면 n을 그대로 반환하고, $n \ge 2$이면 직전 두 값을 더해 다음 값을 만드는 방식으로 $i = n$까지 반복한다.

사용할 공식

이 함수는 $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$인 피보나치 수열의 F_n을 반환한다.
이 공식을 쓰는 이유: 반복문에서 fn에 직전 두 값의 합을 저장하고, 그 값을 다시 다음 계산의 이전값으로 넘기는 구조이기 때문이다.

계산 과정

1. func(4)는 F_4를 구하는 과정이므로 값이 1, 2, 3으로 진행되어 최종값은 3이다.
2. func(9)는 F_9를 구하는 과정이므로 값이 1, 2, 3, 5, 8, 13, 21, 34로 진행되어 최종값은 34이다.
3. 따라서 func(4)+func(9)는 $3 + 34 = 37$이다.

선지별 해설

① X
23은 계산 결과와 일치하지 않는다. 실제 반환값은 func(4)=3, func(9)=34이므로 합은 23이 아니라 37이다.

② O
func(4)=3, func(9)=34이므로 합은 37이다. 계산 결과와 정확히 일치하므로 옳다.

③ X
58은 함수가 반환한 두 값을 더한 결과가 아니다. 올바르게는 3과 34를 더한 37이어야 한다.

④ X
94도 계산 결과와 맞지 않는다. 이 함수는 n번째 피보나치 수를 반환하므로 func(4)+func(9)의 올바른 값은 94가 아니라 37이다.

오답 포인트

반복문의 시작 인덱스와 피보나치 초기값 0, 1을 헷갈리면 F_9를 21이나 55처럼 잘못 세기 쉽다. 이 문제의 함정은 코드의 변수명이 아니라 반복이 끝난 뒤 fnm1에 어떤 항이 들어 있는지를 정확히 추적하는 데 있다.

NoteBase

다시 나오면 이렇게 판단

재귀나 반복으로 구현된 수열 함수는 기저조건을 먼저 확인하고, 초기값이 무엇인지 본 뒤, 작은 n부터 직접 몇 항만 써서 해당 입력이 정확히 몇 번째 항을 반환하는지 판단하면 된다.

암기 포인트

초기값이 0, 1이고 $i = n$까지 누적하면 이 함수의 반환값은 F_n이다.

NoteBase

문 2. <보기>는 알고리즘의 시간 복잡도(time complexity)를 표시하는 방법 중 하나인 빅오(big-oh) 표기법에 대한 정의이다. 빈칸 (가)와 (나)에 각각 들어갈 수식은?

<보기>
두 개의 함수 $g(n)$과 $f(n)$에 대해서 $O(g(n))$은 다음과 같이 정의한다.
$O(g(n))$ = {f(n): 모든 (가)에 대해서 (나)인 양의 상수 c와 n_0가 존재한다.}

① (가): $n \geq n_0$, (나): $0 \leq f(n) \leq c \times g(n)$
② (가): $n_0 \geq n$, (나): $0 \leq f(n) \leq c \times g(n)$
③ (가): $n \geq n_0$, (나): $0 \leq g(n) \leq c \times f(n)$
④ (가): $n_0 \geq n$, (나): $0 \leq g(n) \leq c \times f(n)$

정답 1번

출제 포인트

이 문제는 빅오 표기법의 정의에서 비교 대상이 $f(n)$인지 $g(n)$인지, 그리고 적용 구간이 $n \geq n_0$인지 구별하는 문제이다.

핵심 개념

빅오 $O(g(n))$는 함수 $f(n)$의 증가 상한을 $g(n)$로 묶는 표기이다.
정의는 어떤 양의 상수 c와 n_0가 존재하여 모든 $n \geq n_0$에 대해 $0 \leq f(n) \leq cg(n)$가 성립하는 것이다.
핵심은 큰 n에서 $f(n)$가 $g(n)$의 상수배 이하로 눌린다는 점이다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 자료구조론 · 2020 · 10월

선지별 해설

① O
$n \geq n_0$ 이후의 구간에서 $f(n)$가 $cg(n)$ 이하이면 빅오 정의에 맞는다. 빅오의 표준 정의를 그대로 적은 선택지이다.

② X
틀린 부분은 $n_0 \geq n$이다.
실제로는 모든 $n \geq n_0$에 대해 비교해야 한다. 빅오는 충분히 큰 입력 이후의 상한을 보는 개념이지, n_0보다 작은 쪽을 보는 개념이 아니다.

③ X
틀린 부분은 $0 \leq g(n) \leq cf(n)$이다.
실제로는 $0 \leq f(n) \leq cg(n)$이어야 한다. 빅오 $O(g(n))$는 $f(n)$를 $g(n)$의 상수배로 위에서 억제하는 관계이지, $g(n)$를 $f(n)$의 상수배로 억제하는 정의가 아니다.

④ X
틀린 부분은 $n_0 \geq n$와 $0 \leq g(n) \leq cf(n)$이다.
실제로는 모든 $n \geq n_0$에 대해 $0 \leq f(n) \leq cg(n)$이어야 한다. 비교 방향과 적용 구간이 모두 뒤집혀 있다.

오답 포인트

$f(n)$와 $g(n)$의 위치를 서로 바꾸거나, $n \geq n_0$를 $n_0 \geq n$로 뒤집는 함정이 자주 나온다. 특히 ③, ④처럼 비교 방향을 바꾸면 빅오가 아니라 전혀 다른 관계가 된다.

다시 나오면 이렇게 판단

빅오가 나오면 먼저 '누가 누구의 상수배 이하인가'를 본다. $O(g(n))$에서는 $f(n)$가 대상이고, 기준 함수는 $g(n)$이며, 판단 구간은 항상 충분히 큰 n, 즉 $n \geq n_0$이다.

암기 포인트

빅오 $O(g(n))$는 '큰 n에서 $f(n)$가 $g(n)$의 상수배 이하'이다.

NoteBase

문 3. 1부터 50까지의 정수가 저장된 이진 탐색 트리(binary search tree)에서 루트노드부터 29가 저장된 노드를 탐색하려고 한다. 탐색하는 과정 중에 방문한 노드들의 순서로 옳지 않은 것은?

① 40 → 5 → 39 → 25 → 29

② 25 → 30 → 26 → 27 → 29

③ 10 → 30 → 15 → 17 → 29

④ 20 → 35 → 17 → 33 → 29

정답 4번

출제 포인트

이 문제는 이진 탐색 트리에서 29를 찾을 때 비교 결과에 따라 탐색 가능 구간의 하한과 상한이 어떻게 좁혀지는지를 묻는다.

올바른 순서

1. 현재 노드가 29보다 크면 왼쪽, 29보다 작으면 오른쪽으로 내려간다.
2. 왼쪽으로 내려가면 상한이 현재 노드 값으로, 오른쪽으로 내려가면 하한이 현재 노드 값으로 갱신된다.
3. 다음 방문 노드는 지금까지 정해진 하한과 상한 사이에 있어야 한다.
4. 29를 만나면 탐색이 종료된다.

왜 이 순서인가: 앞 단계의 비교 결과가 다음 단계에서 허용되는 값의 범위를 결정하므로, 한 번 정해진 범위를 벗어나는 값은 이후 경로에 올 수 없다.

선지별 해설

1번: O. 40은 29보다 크므로 왼쪽으로, 5는 29보다 작으므로 오른쪽으로, 39는 29보다 크므로 왼쪽으로, 25는 29보다 작으므로 오른쪽으로 가면 29에 도달할 수 있다. 방문 값 5, 39, 25, 29는 각 단계의 허용 범위를 벗어나지 않는다.

2번: O. 25 다음에는 오른쪽 서브트리로 들어가므로 30이 올 수 있고, 30 다음에는 왼쪽 서브트리로 내려가므로 26이 올 수 있다. 이어서 26보다 큰 27을 거쳐 29로 가는 경로도 허용 범위 안에 있다.

3번: O. 10은 29보다 작으므로 오른쪽, 30은 29보다 크므로 왼쪽, 15와 17은 29보다 작으므로 계속 오른쪽으로 내려가 29에 도달할 수 있다. 각 노드는 10보다 크고 30보다 작은 범위를 유지한다.

4번: X. 20은 29보다 작으므로 이후 경로는 20의 오른쪽 서브트리에 있어야 한다. 따라서 35 다음에 오는 노드도 20보다 커야 하는데, 17은 20보다 작아 이 경로에 올 수 없다. 틀린 표현은 "17"이고, 올바르게는 그 자리에 20보다 크고 35보다 작은 값이 와야 한다. 예를 들면 20→35→25→33→29 같은 경로는 가능하다.

NoteBase

오답 포인트

중간 노드와 29의 대소만 보면 17<29이므로 가능하다고 착각하기 쉽다. 그러나 BST 탐색은 직전 노드뿐 아니라 앞에서 이미 정해진 전체 범위를 함께 지켜야 하므로, 20의 오른쪽 서브트리에 들어간 뒤에는 20보다 작은 값이 다시 나올 수 없다.

다시 나오면 이렇게 판단

BST 경로 문제는 매 단계마다 목표값과의 비교 방향을 정한 뒤, 지금까지 만들어진 허용 구간을 함께 적어 두면 된다. 다음 값이 그 구간 밖으로 나가면 그 경로는 바로 불가능하다.

암기 포인트

BST 탐색 경로는 좌우 방향만이 아니라 조상 노드들이 만든 값의 범위를 끝까지 유지한다.

NoteBase

문 4. 큐(queue)를 적용할 수 있는 상황이 아닌 것은?

① 도착한 순서대로 프린트 처리

② 비동기적 데이터 전송(파일 입·출력, 파이프, 소켓)

③ 프로세스 스케줄링

④ 텍스트 에디터에서의 작업 취소(undo) 과정

정답 4번

출제 포인트

이 문제는 큐의 FIFO 특성이 필요한 상황과 스택의 LIFO 특성이 필요한 상황을 구분하는지를 묻는 문제이다.

핵심 개념

큐는 먼저 들어온 데이터가 먼저 나가는 FIFO 구조이다. 스택은 나중에 들어온 데이터가 먼저 나가는 LIFO 구조이다. 프린트 대기열, 비동기 입출력 버퍼, 준비 큐 기반 스케줄링은 도착 순서 보존이 중요하므로 큐가 적합하다. 반면 undo는 가장 최근 작업을 먼저 취소해야 하므로 스택이 적합하다.

선지별 해설

① O
도착한 순서대로 프린트를 처리하는 것은 먼저 들어온 작업을 먼저 처리하는 방식이므로 큐에 해당한다. 프린트 스풀러의 대기열은 대표적인 FIFO 적용 사례이다.

② O
비동기적 데이터 전송에서 파일 입출력, 파이프, 소켓 버퍼는 먼저 들어온 데이터를 순서대로 처리하거나 전달하는 구조가 기본이므로 큐를 적용할 수 있다. 데이터 흐름의 순서를 유지하는 점이 핵심이다.

③ O
프로세스 스케줄링에서 준비 큐, 대기 큐처럼 실행 대기 중인 프로세스를 관리할 때 큐가 사용된다. 특히 FCFS 같은 스케줄링은 도착 순서 자체가 판단 기준이므로 큐와 직접 연결된다.

④ X
텍스트 에디터의 작업 취소는 가장 최근에 수행한 작업부터 되돌려야 하므로 LIFO 구조가 필요하다. 틀린 표현은 "큐를 적용"이다. 올바르게는 "텍스트 에디터에서의 작업 취소 과정에는 스택을 적용한다"이다.

NoteBase

오답 포인트

④는 일상적으로 작업이 순서대로 쌓인다는 점만 보고 큐로 착각하기 쉽다. 그러나 undo의 기준은 입력 순서가 아니라 최근 작업의 역순 처리이므로 FIFO가 아니라 LIFO로 판단해야 한다.

다시 나오면 이렇게 판단

문제 상황이 도착 순서 보존이면 큐, 최근 작업부터 되돌리거나 꺼내야 하면 스택으로 판단한다. 판단 기준은 FIFO인지 LIFO인지이다.

암기 포인트

큐는 선입선출, 스택은 후입선출, undo는 스택이다.

NoteBase

문 5. <보기>는 가중치가 부여된 무방향 그래프(weighted undirected graph)를 인접 행렬(adjacency matrix)로 표현한 것이다. 이 그래프에 대한 최소 비용 신장 트리(minimum spanning tree)에서 간선(edge)의 가중치를 모두 더한 값은?

<보기>

	A	B	C	D	E
A	0	8	3	∞	∞
B	8	0	5	4	∞
C	3	5	0	2	6
D	∞	4	2	0	7
E	∞	∞	6	7	0

① 14

② 15

③ 16

④ 17

정답 2번

출제 포인트

가중치가 작은 간선부터 사이클이 생기지 않게 선택하는 크루스칼 알고리즘으로 최소 신장 트리의 총 가중치를 구하는 문제이다.

조건 정리

주어진 간선은 AB=8, AC=3, BC=5, BD=4, CD=2, CE=6, DE=7이다.
무방향 그래프이므로 인접 행렬은 대칭으로 읽고, ∞는 간선이 없음을 뜻한다.
정점이 5개이므로 신장 트리는 사이클 없이 간선 4개를 선택해야 한다.

NoteBase

사용할 공식

최소 신장 트리는 가중치가 가장 작은 간선부터 보되, 이미 연결된 정점들 사이에 사이클을 만들면 그 간선은 제외하는 크루스칼 알고리즘으로 구한다.
이 문제는 인접 행렬이 주어졌으므로 간선 목록을 뽑아 오름차순으로 정렬한 뒤 사이클 여부만 판단하면 된다.

계산 과정

간선을 가중치 오름차순으로 정리하면 CD(2), AC(3), BD(4), BC(5), CE(6), DE(7), AB(8)이다.
먼저 CD(2), AC(3), BD(4)를 선택하면 A-B-C-D가 모두 연결된다.
다음 BC(5)는 B-D-C-B의 사이클을 만들므로 제외한다.
남은 정점 E를 가장 싸게 잇는 간선은 CE(6)이므로 이를 선택한다.
따라서 최소 비용 신장 트리의 가중치 합은 2+3+4+6=15이다.

선지별 해설

① X
14는 CD(2)+AC(3)+BD(4)+BC(5)=14처럼 계산한 경우인데, 이 경우 E가 연결되지 않았고 BC는 사이클도 만든다. 틀린 표현은 '최소 비용 신장 트리의 합이 14이다'라는 판단이다. 올바르게는 E를 연결하는 CE(6)을 포함해야 하므로 합은 15이다.

② O
CD(2), AC(3), BD(4), CE(6)을 선택하면 모든 정점이 연결되고 사이클이 없다. 가중치 합은 15이므로 옳은 선택지이다.

③ X
16은 CD(2), AC(3), BC(5), CE(6)을 선택한 값으로 볼 수 있는데, B를 연결할 때 BC(5)보다 BD(4)가 더 작다. 틀린 표현은 '최소 비용 신장 트리의 합이 16이다'라는 판단이다. 올바르게는 BC 대신 BD를 택해 합이 15가 된다.

④ X
17은 AB(8)을 포함해 AC(3), CD(2), CE(6)을 고른 경우처럼 볼 수 있다. 그러나 B는 AB(8)보다 BD(4)로 더 작게 연결할 수 있다. 틀린 표현은 '최소 비용 신장 트리의 합이 17이다'라는 판단이다. 올바르게는 AB를 넣지 않고 BD를 넣어 총합 15를 만든다.

오답 포인트

가중치가 작은 간선 몇 개를 단순히 더하면 된다고 생각하면 사이클 검사나 미연결 정점 확인을 놓치기 쉽다. 특히 BC(5)는 값이 작아 보여도 이미 B와 C가 연결된 뒤에는 버려야 하고, E를 잇는 가장 작은 간선 CE(6)을 반드시 포함해야 한다.

NoteBase

다시 나오면 이렇게 판단

최소 신장 트리 문제는 인접 행렬을 간선 목록으로 바꾼 뒤 가중치 오름차순 정렬, 사이클 검사, 남은 정점 연결 여부 확인의 순서로 판단한다. 정점이 5개이면 최종 간선 수가 4개인지까지 함께 검산하면 된다.

암기 포인트

최소 신장 트리는 작은 간선을 무조건 고르는 문제가 아니라 사이클 없이 모든 정점을 잇는 가장 작은 간선 집합을 고르는 문제이다.

NoteBase

문 6. C언어로 작성된 <보기>의 함수는 입력된 문자열을 검사하여 0 또는 1을 반환한다. 이 함수의 인자로 넣었을 때 1을 반환하는 문자열은? (단, push()는 문자 하나를 스택에 삽입하는 함수이고, pop()는 문자 하나를 스택에서 추출하는 함수이다.)

<보기>

```
int isTrue(char str[])
{
        int length = strlen(str);
        for (int i = 0; i < length; i++)
                push(str[i]);
        for (int i = 0; i < length; i++)
                if (pop() != str[i])
                        return 0;
        return 1;
}
```

① AAAAAAaaaaaa

② aaaAAAaaaAAA

③ AAAaaaaaaAAA

④ AaAaAaAaAaAa

정답 3번

출제 포인트

이 문제는 스택의 후입선출 특성 때문에 문자열이 역순으로 비교된다는 점, 즉 이 함수가 회문 문자열인지 검사하는 흐름을 묻는 문제이다.

NoteBase

올바른 순서

1. `strlen(str)`로 문자열 길이를 구한다.
2. 문자열의 각 문자를 처음부터 끝까지 모두 스택에 `push`한다.
3. 스택에서 `pop`하면 문자가 끝에서부터 역순으로 나온다.
4. 꺼낸 문자를 원래 문자열의 앞에서부터 `str[i]`와 비교하여 하나라도 다르면 0을 반환하고, 끝까지 모두 같으면 1을 반환한다.

왜 이 순서: 모든 문자를 먼저 스택에 넣어야 역순 추출이 가능하므로, 이 함수는 결국 원문과 역순 문자열이 같은지 비교한다.

선지별 해설

① X
- `AAAAAAaaaaaaa`의 역순은 `aaaaaaaAAAAAA`이므로 원문과 다르다. 첫 비교부터 `a`와 `A`가 달라 0을 반환한다.
- 틀린 표현: `AAAAAAaaaaaaa`가 1을 반환한다.
- 올바르게는: 앞뒤가 문자와 대소문자까지 완전히 대칭인 문자열만 1을 반환한다.

② X
- `aaaAAAaaaAAA`의 역순은 `AAAaaaAAAaaa`이므로 원문과 다르다. 앞쪽의 `aaa`와 뒤쪽의 `AAA`가 서로 뒤집혀 대소문자 위치가 맞지 않는다.
- 틀린 표현: `aaaAAAaaaAAA`가 1을 반환한다.
- 올바르게는: 역순으로 뒤집어도 같은 배열이 되는 회문 문자열이어야 1을 반환한다.

③ O
- `AAAaaaaaaAAA`의 역순도 `AAAaaaaaaAAA`이다. 양 끝의 `AAA`와 가운데의 `aaaaaa`가 좌우 대칭이므로 모든 `pop()` 결과가 `str[i]`와 일치하여 1을 반환한다.

④ X
- `AaAaAaAaAaAa`는 반복 패턴처럼 보이지만 역순은 `aAaAaAaAaAaA`이다. 첫 문자부터 `A`와 `a`가 달라 원문과 역순이 같지 않다.
- 틀린 표현: `AaAaAaAaAaAa`가 대칭 문자열이므로 1을 반환한다.
- 올바르게는: 교차 패턴처럼 보여도 뒤집었을 때 각 위치의 문자와 대소문자가 그대로 같아야 1을 반환한다.

오답 포인트

이 함수가 문자열을 그대로 다시 읽는다고 생각하면 틀린다. `push`를 모두 끝낸 뒤 `pop`으로 비교하므로 실제로는 역순 비교가 이루어진다. 특히 ④처럼 규칙적으로 반복되는 문자열은 회문처럼 보이지만, 대소문자까지 같은 위치에 와야 하므로 쉽게 낚인다.

NoteBase

다시 나오면 이렇게 판단

`push`를 전부 한 뒤 `pop` 결과를 원문과 비교하는 코드는 원문과 역순 문자열의 일치 여부로 바꿔 읽는다. 따라서 이런 문제는 각 선택지가 회문인지, 그리고 대소문자까지 정확히 대칭인지부터 확인하면 된다.

암기 포인트

전부 `push` 후 전부 `pop`이면 역순 비교이고, 정답 조건은 회문 문자열이다.

NoteBase

문 7. <보기>는 단순 연결 리스트(singly linked list)에서 원소들의 순서를 역순으로 변환하는 C언어 프로그램이다. (가), (나), (다)에 들어갈 내용을 옳게 짝지은 것은?

<보기>

```
typedef struct node {
    int key;
    struct node *link;
} ListNode;

ListNode *reverse(ListNode *L)
{
    ListNode *p, *q, *r;
    p = L;
    q = NULL;
    r = NULL;
    while (p != NULL) {
        [ (가) ];
        [ (나) ];
        p = p → link;
        [ (다) ];
    }
    return q;
}
```

① (가): q = p, (나): r = q, (다): q → link = r

② (가): r = q, (나): q = p, (다): q → link = r

③ (가): r = q, (나): q = p, (다): r = q → link

④ (가): q = p, (나): r = q, (다): r = q → link

정답 2번

출제 포인트

단순 연결 리스트를 역순으로 뒤집을 때 이전 노드, 현재 노드, 다음 노드 포인터의 선후관계를 정확히 유지하는지를 묻는 문제이다.

올바른 순서

1. r = q - 지금까지 뒤집어 둔 부분 리스트의 머리를 따로 저장한다.
2. q = p - 현재 노드를 뒤집힌 리스트의 새 머리 후보로 잡는다.
3. p = p -> link - 원래 리스트의 다음 노드로 이동한다.
4. q -> link = r - 현재 노드가 이전까지 뒤집어 둔 리스트를 가리키게 만든다.

왜 이 순서인가: r은 q를 덮어쓰기 전에 저장되어야 하고, p의 다음 노드는 링크를 반전시키기 전에 확보되어야 하기 때문이다.

NoteBase

선지별 해설

1번: X. (가)에서 q = p를 먼저 수행하면 (나)의 r = q도 현재 노드를 가리키게 된다. 그러면 (다)의 q -> link = r에서 현재 노드가 자기 자신을 가리키는 자기 루프가 생긴다. 틀린 표현은 "q = p; r = q"의 순서이며, 올바르게는 "r = q; q = p"이다.
2번: O. r이 이전 reversed head를 보관하고, q가 현재 노드를 잡은 뒤, p를 다음 노드로 이동시키고, 마지막에 q -> link = r로 링크를 반전시킨다. 남은 리스트와 뒤집은 리스트의 머리를 모두 잃지 않는다.
3번: X. (가), (나)는 맞지만 (다)의 r = q -> link는 링크 방향을 바꾸지 않고 r 변수에 값만 다시 대입한다. 틀린 표현은 "r = q -> link"이며, 올바르게는 "q -> link = r"이다.
4번: X. 1번과 같이 q = p를 먼저 두어 r이 이전까지 뒤집은 리스트가 아니라 현재 노드를 가리키게 된다. 또한 (다)도 링크를 반전시키는 문장이 아니라 포인터 복사 문장이다. 틀린 표현은 "q = p; r = q"와 "r = q -> link"이며, 올바르게는 "r = q; q = p; q -> link = r"이다.

오답 포인트

이 알고리즘은 값 복사와 링크 변경을 구분하지 못하면 틀리기 쉽다. q를 먼저 덮어쓰면 이전 reversed head를 잃고, q -> link를 먼저 바꾸면 아직 처리하지 않은 나머지 리스트의 진입점을 잃는다.

다시 나오면 이렇게 판단

단순 연결 리스트 역순 문제는 "이전 노드 저장 -> 현재 노드 지정 -> 다음 노드로 전진 -> 현재 링크 반전"의 의존관계로 판단한다. 앞 단계의 산출물이 뒤 단계의 입력이 되는지를 확인하면 순서를 고를 수 있다.

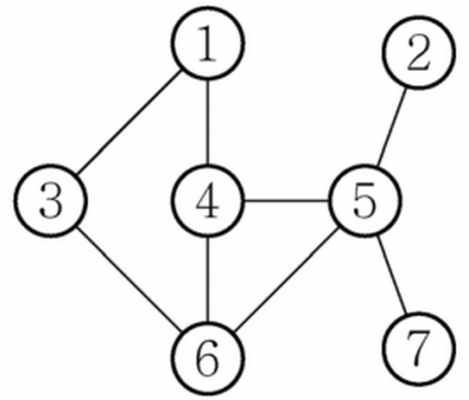
암기 포인트

reverse의 핵심은 "이전 저장, 현재 지정, 다음 이동, 링크 반전"의 4단계이다.

NoteBase

문 8. 시작점이 1일 때, <보기>의 그래프에 대한 너비 우선 탐색(breadth first search)의 방문 순서를 바르게 나열한 것은? (단, 같은 우선순위의 정점들은 숫자가 작은 정점을 먼저 방문한다.)

<보기>



① 1-2-3-4-5-6-7

② 1-3-4-5-2-7-6

③ 1-3-4-5-6-2-7

④ 1-3-4-6-5-2-7

정답 4번

출제 포인트

이 문제는 너비 우선 탐색에서 큐에 들어간 순서와 같은 우선순위에서는 작은 번호를 먼저 처리하는 규칙을 함께 적용해 방문 순서를 판단하는 문제이다.

올바른 순서

1. 1을 방문하고 인접한 3, 4를 번호순으로 큐에 넣는다.
2. 3을 방문하고 새로 발견되는 6을 큐 뒤에 넣는다.
3. 4를 방문하고 6은 이미 큐에 있으므로 다시 넣지 않으며, 새로 발견되는 5를 큐 뒤에 넣는다.
4. 6을 방문하고 인접한 3, 4, 5는 이미 방문했거나 큐에 있으므로 추가되는 정점이 없다.
5. 5를 방문하고 새로 발견되는 2, 7을 번호순으로 큐 뒤에 넣는다.
6. 2를 방문한다.
7. 7을 방문한다.

왜 이 순서인가: 6은 3을 처리할 때 먼저 큐에 들어가고, 5는 그 다음 4를 처리할 때 큐에 들어가므로 6이 5보다 먼저 방문된다.

NoteBase

선지별 해설

① X: 시작점 1의 인접 정점은 3과 4뿐이므로 2가 두 번째로 방문될 수 없다. 틀린 표현은 "1-2"이다. 올바르게는 "1-3-4"로 시작한다.
② X: 5를 6보다 먼저 방문한 부분이 틀리다. 6은 3을 처리할 때 이미 큐에 들어가 있고, 5는 4를 처리할 때 뒤늦게 들어가므로 "1-3-4-5"가 아니라 "1-3-4-6"이 된다. 올바르게는 "1-3-4-6-5-2-7"이다.
③ X: "4-5-6"의 선후관계가 바뀌었다. 5는 4에서 발견되지만 6은 그보다 앞선 3에서 이미 발견되어 큐 앞쪽에 있으므로 6이 먼저 방문된다. 올바르게는 "1-3-4-6-5-2-7"이다.
④ O: 1에서 3, 4를 넣고, 3에서 6을 먼저 발견한 뒤 4에서 5를 발견하므로 방문 순서는 1-3-4-6-5-2-7이 된다.

오답 포인트

③의 "4-5-6"처럼 현재 보고 있는 정점의 인접 정점을 바로 다음 방문으로 착각하기 쉽다. BFS는 현재 정점의 이웃이라고 해서 즉시 방문하는 방식이 아니라, 먼저 큐에 들어간 정점부터 꺼내므로 6이 5보다 앞선다는 점이 핵심 함정이다.

다시 나오면 이렇게 판단

BFS 순서 문제는 방문 시점보다 큐에 들어가는 시점을 기준으로 선후관계를 판단한다. 시작 정점의 인접 정점을 번호순으로 큐에 넣고, 각 단계에서는 아직 발견되지 않은 정점만 큐 뒤에 추가하면 순서가 정리된다.

암기 포인트

BFS는 가까워 보이는 정점 우선이 아니라 먼저 큐에 들어간 정점 우선이다.

NoteBase

문 9. 이진 트리(binary tree)를 배열로 저장할 때 <보기 1>과 같은 특성을 갖는다고 가정하자. 이진 트리를 <보기 2>와 같은 형태의 배열로 저장했을 때, 이 트리에 대한 설명으로 가장 옳은 것은?

<보기 1>

- 배열 인덱스 i에 저장된 노드의 부모노드 인덱스는 $\lfloor i/2 \rfloor$
- 배열 인덱스 i에 저장된 노드의 왼쪽 자식노드 인덱스는 $2i$
- 배열 인덱스 i에 저장된 노드의 오른쪽 자식노드 인덱스는 $2i + 1$

<보기 2>

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
	20	7	28	1	15		30			11	17				36

① 이 트리는 포화 이진 트리(full binary tree)이다.

② 이 트리는 완전 이진 트리(complete binary tree)이다.

③ 단말 노드(leaf node) 개수는 3개이다.

④ 이 트리는 이진 탐색 트리(binary search tree)이다.

정답 4번

출제 포인트

배열 인덱스로 복원한 이진 트리에서 포화 이진 트리, 완전 이진 트리, 단말 노드 수, 이진 탐색 트리 조건을 구별하는 문제이다.

핵심 개념

포화 이진 트리(full binary tree)는 모든 내부 노드가 자식을 2개 가지는 트리이다.
완전 이진 트리(complete binary tree)는 마지막 레벨을 제외한 모든 레벨이 차 있고 마지막 레벨도 왼쪽부터 빈칸 없이 채워진 트리이다.
단말 노드는 자식이 없는 노드이다.
이진 탐색 트리(binary search tree)는 각 노드에서 왼쪽 서브트리의 값은 더 작고 오른쪽 서브트리의 값은 더 큰 트리이다.

NoteBase

선지별 해설

① X: 28은 오른쪽 자식 30만 있고 왼쪽 자식이 없으며, 30도 오른쪽 자식 36만 가진다. 따라서 모든 내부 노드가 자식 2개를 갖지 않으므로 포화 이진 트리가 아니다. 올바르게는 자식이 하나뿐인 내부 노드가 있는 일반 이진 트리이다.
② X: 배열에서 6번 인덱스가 비어 있는데 7번 인덱스에 30이 저장되어 있고, 더 아래 레벨의 10, 11, 15번 인덱스에도 노드가 있다. 완전 이진 트리는 빈칸 없이 왼쪽부터 채워져야 하므로 아니다. 올바르게는 중간 빈칸이 있는 비완전 이진 트리이다.
③ X: 단말 노드는 1, 11, 17, 36으로 모두 4개이다. 따라서 3개라는 설명은 틀리다. 올바르게는 단말 노드 개수는 4개이다.
④ O: 배열을 트리로 복원하면 20의 왼쪽에는 7, 1, 15, 11, 17이 있고 모두 20보다 작으며, 오른쪽에는 28, 30, 36이 있고 모두 20보다 크다. 또한 7의 왼쪽 1은 7보다 작고 오른쪽 15는 7보다 크며, 15의 왼쪽 11은 15보다 작고 오른쪽 17은 15보다 크다. 28의 오른쪽 30과 30의 오른쪽 36도 대소관계를 만족하므로 이 트리는 이진 탐색 트리이다.

오답 포인트

완전 이진 트리와 포화 이진 트리를 같은 말로 착각하면 ①과 ②를 함께 고르기 쉽다. 이 문항은 배열 인덱스의 빈칸 6을 놓치게 만들고, 단말 노드도 끝값 36을 빠뜨리기 쉽게 구성되어 있다.

다시 나오면 이렇게 판단

배열 문제는 인덱스 1을 루트로 두고 부모-자식을 먼저 복원한 뒤, 자식이 없는 노드 수를 세고, 중간 빈칸 유무로 완전성을 본 다음, 각 노드의 왼쪽<부모<오른쪽 대소관계로 BST 여부를 판단한다.

암기 포인트

완전은 왼쪽부터 빈칸 없이, BST는 왼쪽<부모<오른쪽, 단말은 자식 0개이다.

NoteBase

문 10. 공백 AVL 트리에 키(key)를 차례대로 저장하여 AVL 트리를 구축한다고 할 때, 생성되는 AVL 트리의 모양이 다른 것은?

① 2, 3, 5, 6, 7, 9

② 3, 5, 6, 2, 7, 9

③ 5, 2, 3, 7, 6, 9

④ 6, 7, 9, 2, 3, 5

정답 2번

출제 포인트

AVL 트리에 같은 키 집합을 넣더라도 삽입 순서에 따라 회전 시점과 최종 트리 모양이 달라질 수 있는지를 묻는 문제이다.

핵심 개념

AVL 트리는 각 노드의 왼쪽 서브트리와 오른쪽 서브트리의 높이 차가 1 이하가 되도록 회전으로 균형을 맞추는 이진 탐색 트리이다.
같은 키 집합이라도 삽입 순서가 달라지면 처음 불균형이 발생하는 노드와 적용되는 LL, LR, RR, RL 회전이 달라져 최종 모양이 달라질 수 있다.
이 문항에서는 ①, ③, ④가 모두 루트 6, 왼쪽 자식 3, 오른쪽 자식 7의 형태로 수렴하고, ②만 루트 5의 형태로 끝난다.

선지별 해설

① X: 2, 3, 5를 넣으면 RR 회전으로 루트가 3이 되고, 이후 6, 7, 9를 넣으면 최종적으로 루트 6, 왼쪽 3(자식 2, 5), 오른쪽 7(오른쪽 9) 형태가 된다. 따라서 모양이 다른 경우가 아니다. 올바르게는 ①은 ③, ④와 같은 모양이다.
② O: 3, 5, 6을 넣으면 루트 5가 되고, 이어서 2, 7, 9를 넣으면 최종적으로 루트 5, 왼쪽 3(왼쪽 2), 오른쪽 7(자식 6, 9) 형태가 된다. 다른 세 선택지와 루트부터 다르므로 모양이 다른 경우가 맞다.
③ X: 5, 2, 3에서 LR 회전이 일어나 루트 3이 되지만, 이후 7, 6, 9를 넣으면 다시 균형을 맞춘 결과가 ①과 같은 루트 6 형태로 정리된다. 따라서 모양이 다른 경우가 아니다. 올바르게는 ③은 ①, ④와 같은 모양이다.
④ X: 6, 7, 9에서 RR 회전으로 루트 7이 된 뒤 2, 3, 5를 넣으면 왼쪽 부분트리에서 균형이 맞춰져 최종적으로 ①, ③과 같은 루트 6 형태가 된다. 따라서 모양이 다른 경우가 아니다. 올바르게는 ④는 ①, ③과 같은 모양이다.

오답 포인트

같은 키 집합이므로 최종 AVL 트리도 모두 같을 것이라고 단정하면 틀리기 쉽다. AVL은 중위순회 결과는 같아도 회전이 일어나는 시점이 다르면 루트와 부분트리 배치가 달라질 수 있다는 점이 함정이다.

NoteBase

다시 나오면 이렇게 판단

삽입 순서를 앞에서부터 따라가며 불균형이 처음 발생하는 노드를 찾고, 그때 적용되는 회전 종류만 추적하면 된다. 모든 키를 다 넣은 뒤 루트와 좌우 자식 배치가 같은지 비교하면 서로 같은 모양인지 빠르게 판별할 수 있다.

암기 포인트

같은 키 집합이라도 AVL의 최종 모양은 삽입 순서와 회전 시점이 결정한다.

NoteBase

문 11. <보기>의 중위 표기식(infix expression)을 전위 표기식(prefix expression)으로 변환한 것으로 옳은 것은?

<보기>
((x / y) - z) + (u * v * w) - ((x * z) / u)

① -+-/xyz**uvw/*xzu
② -+-/xyz**uvw*/xzu
③ -/xyz+**uvw/*-xzu
④ -/xyz+**uvw-/*xzu

정답 1번

출제 포인트

중위 표기식을 전위 표기식으로 바꿀 때 괄호, 연산자 우선순위, 좌결합을 유지하는지를 묻는 문제이다.

올바른 순서

1. (x/y)를 먼저 전위로 바꾸면 $/xy$이다.
2. $((x/y)-z)$는 앞 결과를 왼쪽 피연산자로 두므로 $-/xyz$가 된다.
3. $(u*v*w)$는 곱셈이 좌결합이므로 $((u*v)*w)$로 보고 $**uvw$가 된다.
4. $((x*z)/u)$는 $(x*z)$를 먼저 묶어 $/*xzu$가 된다.
5. 전체 식은 $(((x/y)-z)+(u*v*w))-((x*z)/u)$이므로 먼저 $+-/xyz**uvw$, 마지막에 이를 $((x*z)/u)$와 결합해 $-+-/xyz**uvw/*xzu$가 된다.

왜 이 순서: 바깥쪽 +와 −는 같은 우선순위이지만 좌결합이므로 왼쪽 덧셈 부분을 먼저 만들고 마지막에 오른쪽 항을 뺀다.

선지별 해설

① O: 전체 구조가 $(((x/y)-z)+(u*v*w))-((x*z)/u)$를 정확히 반영한다. 즉 $-+-/xyz**uvw/*xzu$는 원식과 같은 전위 표기식이다.

② X: 끝부분 $*/xzu$가 틀렸다. 이는 $((x/z)*u)$처럼 읽히므로 원래의 $((x*z)/u)$와 다르다. 올바르게는 $/*xzu$가 되어야 한다.

③ X: 시작 부분 $-/xyz+$가 틀렸다. 이렇게 쓰면 $((x/y)-z)$를 왼쪽 한 덩어리로 처리한 뒤 $(u*v*w)$를 더하는 구조가 깨진다. 끝부분 $/*-xzu$도 원문에 없는 음수 처리를 포함한다. 올바르게는 전체가 $-+-/xyz**uvw/*xzu$가 되어야 한다.

④ X: 시작을 $-/xyz+$로 두면 왼쪽 부분식 $((x/y)-z)+(u*v*w)$가 무너진다. 또한 $-/*xzu$는 마지막 뺄셈의 위치를 잘못 둔 형태이다. 올바르게는 전체가 $-+-/xyz**uvw/*xzu$가 되어야 한다.

NoteBase

오답 포인트

전위 표기에서는 연산자를 앞에 적는다는 형식만 기억하고 원래 식의 결합 순서를 놓치기 쉽다. 특히 $(u * v * w)$의 좌결합과 $((x * z)/u)$의 내부 묶음을 동시에 관리하지 않으면 ②처럼 마지막 부분식이 바뀌고, ③·④처럼 바깥 −의 위치까지 틀어진다.

다시 나오면 이렇게 판단

중위에서 전위로 바꿀 때는 가장 바깥 연산자를 먼저 찾고, 왼쪽 부분식과 오른쪽 부분식을 각각 다시 전위로 바꾼 뒤 이어 붙이면 된다. 같은 우선순위의 연산자가 연속되면 좌결합인지 우결합인지부터 확인하는 방식이 안정적이다.

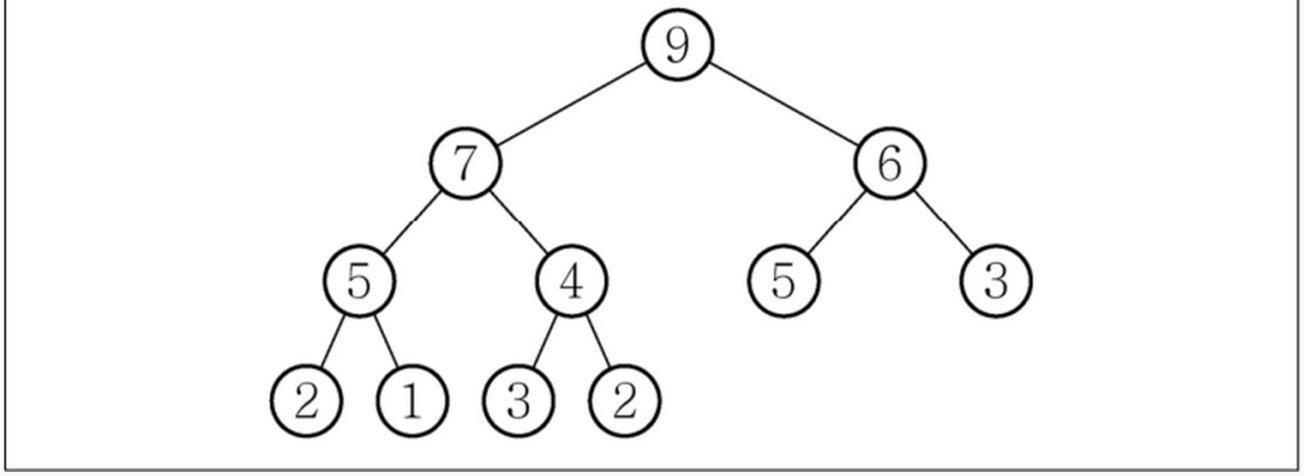
암기 포인트

전위 표기식은 바깥 연산자, 왼쪽 부분식 전체, 오른쪽 부분식 전체의 순서로 읽는다.

NoteBase

문 12. <보기>의 최대 히프(max heap)에 8을 삽입한 후, 최댓값을 삭제하는 연산을 수행하였다. 이때, 루트 노드의 좌·우 자식노드의 값은?

<보기>



① 좌: 7, 우: 5

② 좌: 7, 우: 6

③ 좌: 7, 우: 8

④ 좌: 8, 우: 7

정답 2번

출제 포인트

이 문제는 최대 힙에서 삽입 시 상향 이동과 최대값 삭제 시 하향 이동을 순서대로 적용하여 최종 루트의 좌우 자식 값을 구하는 문제이다.

NoteBase

선지별 해설

① X: 초기 힙의 레벨 순서는 9, 7, 6, 5, 4, 5, 3, 2, 1, 3, 2이다. 여기에 8을 삽입하면 마지막 자리에 들어간 뒤 부모 5, 다시 부모 6과 교환되어 9, 7, 8, 5, 4, 6, 3, 2, 1, 3, 2, 5가 된다. 이후 최대값 9를 삭제하면 마지막 원소 5가 루트로 올라오고, 더 큰 자식 8과 교환한 뒤 다시 6과 교환되어 최종 힙은 8, 7, 6, 5, 4, 5, 3, 2, 1, 3, 2가 된다. 따라서 우 자식이 5라는 표현이 틀렸고, 올바르게는 좌 7 우 6이다.
② O: 삽입 후 8은 6보다 크고 9보다 작으므로 루트의 오른쪽 자식 위치까지 올라간다. 그 상태에서 9를 삭제하면 마지막 원소 5가 루트로 올라온 뒤 8, 6과 차례로 교환된다. 최종 루트는 8이고 좌우 자식은 7과 6이므로 좌 7 우 6이 맞다.
③ X: 삭제 연산이 끝난 뒤 8은 루트가 된다. 따라서 우 자식이 8이라는 표현이 틀렸고, 올바르게는 루트가 8이며 그 오른쪽 자식은 6이다.
④ X: 삽입 직후에는 루트의 자식이 7과 8처럼 보일 수 있으나, 문제는 그 다음에 최대값 삭제까지 수행한 결과를 묻는다. 삭제 후 8은 왼쪽 자식이 아니라 루트가 되므로 좌 8 우 7이라는 표현이 틀렸고, 올바르게는 좌 7 우 6이다.

오답 포인트

삽입 결과만 보고 바로 답을 고르면 ③이나 ④로 흔들리기 쉽다. 이 문항은 삽입과 삭제를 모두 수행한 최종 상태를 묻기 때문에, 삽입 후 한 번 더 힙 재구성을 해야 한다는 점이 함정이다.

다시 나오면 이렇게 판단

힙 문제는 트리를 바로 눈으로만 보지 말고 레벨 순서 배열로 바꾼 뒤, 삽입은 마지막 칸에 넣고 위로 올리며, 삭제는 마지막 원소를 루트에 올린 뒤 더 큰 자식과 교환하면서 내려가면 된다.

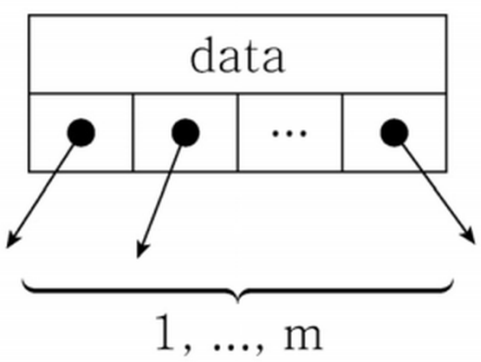
암기 포인트

최대 힙은 삽입은 위로, 삭제는 큰 자식 쪽으로 아래로 이동한다.

NoteBase

문 13. m차 트리(m-ary tree)는 각 노드(node)의 자식 노드가 최대 m개인 트리를 의미한다. <보기>의 그림과 같은 연결 리스트(linked list)로 구현된 m차 트리의 노드의 수가 n개일 때, 이 트리의 널 포인터(null pointer)의 총 개수는?

<보기>



① $n(m+1)+1$

② $n(m+1)-1$

③ $n(m-1)+1$

④ $n(m-1)-1$

정답 3번

출제 포인트

이 문제는 연결 리스트로 구현된 m차 트리에서 전체 자식 포인터 수와 실제 사용된 포인터 수를 비교해 널 포인터 수를 구하는 문제이다.

조건 정리

주어진 값: 노드 수는 n개이고, 각 노드는 자식 포인터 칸을 m개 가진다.
구해야 할 값: 널 포인터의 총 개수이다.
주의할 조건: 트리에서 실제로 연결된 포인터 수는 간선 수와 같고, 노드가 n개이면 간선 수는 항상 $n-1$개이다.

사용할 공식

널 포인터 수는 전체 포인터 칸 수 - 실제 연결된 포인터 수이다.
즉, 널 포인터 수는 $mn-(n-1)$이다.
이 공식을 쓰는 이유: 각 노드는 자식 포인터 칸을 m개씩 가지므로 전체 포인터 칸 수는 mn개이고, 실제 자식을 가리키는 포인터 수는 트리의 간선 수와 같아서 $n-1$개이기 때문이다.

NoteBase

계산 과정

1. 각 노드는 자식 포인터 칸을 m개 가지므로 전체 포인터 칸 수는 mn개이다.
2. 노드가 n개인 트리의 간선 수는 항상 $n-1$개이므로 null이 아닌 포인터 수는 $n-1$개이다.
3. 따라서 널 포인터 수는 $mn-(n-1)=mn-n+1=n(m-1)+1$이다.

선지별 해설

① X: $n(m+1)+1$은 각 노드의 포인터 칸 수를 m이 아니라 $m+1$로 잘못 센 값이다. 이 문제의 노드 구조는 자식 포인터가 m개이므로 올바르게는 $n(m-1)+1$이다.
② X: $n(m+1)-1$은 전체 포인터 칸 수를 mn이 아니라 $n(m+1)$로 잘못 잡은 식이다. 올바른 출발식은 $mn-(n-1)$이다.
③ O: 전체 자식 포인터 칸 mn개에서 실제 연결된 포인터 $n-1$개를 빼면 $mn-(n-1)=n(m-1)+1$이므로 옳다.
④ X: $n(m-1)-1$은 $mn-(n-1)$을 전개할 때 상수항 $+1$을 -1로 잘못 처리한 값이다. 올바르게는 $mn-n+1=n(m-1)+1$이다.

오답 포인트

이 문항의 함정은 '전체 포인터 칸 수'와 '실제로 사용된 포인터 수'를 구분하지 못하게 만드는 데 있다. ①과 ②는 노드마다 포인터 수를 m보다 크게 세는 실수이고, ④는 $mn-(n-1)$의 부호를 잘못 전개한 실수이다.

다시 나오면 이렇게 판단

트리의 널 포인터 개수를 묻는 문제는 먼저 노드당 포인터 칸 수로 전체 칸 수를 구하고, 다음에 트리의 간선 수 $n-1$를 빼는 순서로 판단한다. 트리에서는 '실제로 연결된 포인터 수 = 간선 수'라는 대응을 바로 떠올리면 된다.

암기 포인트

m차 트리의 연결 표현에서 널 포인터 수는 '전체 mn개 - 사용된 $n-1$개'이므로 $n(m-1)+1$이다.

NoteBase

문 14. 최악의 경우 시간 복잡도가 $O(nlogn)$인 정렬 알고리즘은?

① 삽입 정렬(insertion sort)

② 퀵 정렬(quick sort)

③ 합병 정렬(merge sort)

④ 버블 정렬(bubble sort)

정답 3번

출제 포인트

이 문제는 대표 정렬 알고리즘들의 최악 시간 복잡도를 구분하는 문제이다.

핵심 개념

삽입 정렬은 원소를 앞의 정렬된 구간에 끼워 넣는 방식이어서 역순 입력에서는 이동과 비교가 많이 발생해 최악 시간이 $O(n^2)$이다.
퀵 정렬은 분할 정복 정렬이지만 피벗이 계속 한쪽으로 치우치면 분할이 불균형해져 최악 시간이 $O(n^2)$이다.
합병 정렬은 배열을 나누고 다시 병합하는 모든 단계에서 전체 원소를 처리하며 단계 수가 $\log n$이므로 최악 시간도 $O(n\log n)$이다.
버블 정렬은 인접 원소를 반복 교환하므로 최악 시간이 $O(n^2)$이다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 자료구조론 · 2020 · 10월

선지별 해설

① X
삽입 정렬은 역순 배열에서 각 원소가 앞쪽으로 끝까지 이동할 수 있어 최악 시간이 $O(n^2)$이다.
틀린 표현은 "최악의 경우 시간 복잡도가 $O(n\log n)$"이다.
올바르게는 "삽입 정렬의 최악 시간 복잡도는 $O(n^2)$이다."이다.

② X
퀵 정렬은 평균적으로는 $O(n\log n)$이지만, 피벗 선택이 계속 불리하면 한쪽 부분배열만 크게 남아 최악 시간이 $O(n^2)$이 된다.
틀린 표현은 "최악의 경우 시간 복잡도가 $O(n\log n)$"이다.
올바르게는 "퀵 정렬의 최악 시간 복잡도는 $O(n^2)$이다."이다.

③ O
합병 정렬은 분할 깊이가 $\log n$이고 각 단계의 병합 비용이 $O(n)$이어서 최악의 경우에도 전체 시간이 $O(n\log n)$이다.
정렬 과정이 입력 분포에 크게 흔들리지 않는다는 점이 정답 근거이다.

④ X
버블 정렬은 인접한 두 원소를 여러 번 비교하고 교환하는 방식이라 최악의 경우 비교와 교환이 누적되어 $O(n^2)$이다.
틀린 표현은 "최악의 경우 시간 복잡도가 $O(n\log n)$"이다.
올바르게는 "버블 정렬의 최악 시간 복잡도는 $O(n^2)$이다."이다.

오답 포인트

퀵 정렬의 평균 시간 복잡도 $O(n\log n)$를 최악 시간 복잡도와 혼동하면 ②를 고르기 쉽다. 이 문제는 평균이 아니라 최악의 경우를 묻는다는 점이 함정이다.

다시 나오면 이렇게 판단

정렬 문제에서 시간 복잡도를 물으면 평균인지 최악인지부터 확인한다. 최악 기준에서는 합병 정렬만 안정적으로 $O(n\log n)$이고, 삽입 정렬·퀵 정렬·버블 정렬은 모두 $O(n^2)$ 가능성이 있다고 판단한다.

암기 포인트

합병 정렬은 최악도 $O(n\log n)$, 삽입 정렬·퀵 정렬·버블 정렬은 최악 $O(n^2)$이다.

NoteBase

문 15. 데크(deque: double ended queue)에 대해 <보기>와 같이 4가지 연산을 사용할 수 있다고 하자. 데크의 연산을 사용하여 스택과 큐를 구현한다고 할 때, 필요한 데크의 연산을 옳게 짝지은 것은? (여기서 데크를 가리키는 포인터를 d, 스택을 가리키는 포인터를 s, 큐를 가리키는 포인터를 q라고 한다.)

<보기>
• insert_front(d, item): 데크의 첫 번째 원소로 item 삽입
• insert_rear(d, item): 데크의 마지막 원소로 item 삽입
• delete_front(d): 데크의 첫 번째 원소 삭제
• delete_rear(d): 데크의 마지막 원소 삭제

① push(s, item): insert_rear(d, item), pop(s): delete_rear(d), enqueue(q, item): insert_rear(d, item), dequeue(q): delete_front(d)

② push(s, item): insert_rear(d, item), pop(s): delete_front(d), enqueue(q, item): insert_rear(d, item), dequeue(q): delete_rear(d)

③ push(s, item): insert_front(d, item), pop(s): delete_front(d), enqueue(q, item): insert_front(d, item), dequeue(q): delete_front(d)

④ push(s, item): insert_front(d, item), pop(s): delete_rear(d), enqueue(q, item): insert_front(d, item), dequeue(q): delete_rear(d)

정답 1번

출제 포인트

이 문제는 데크에서 같은 쪽으로 넣고 빼면 스택이 되고, 반대쪽으로 넣고 빼면 큐가 된다는 연산 대응을 묻는 문제이다.

핵심 개념

스택은 한쪽 끝에서만 삽입과 삭제가 일어나는 LIFO 구조이다.
큐는 한쪽 끝에서 삽입하고 반대쪽 끝에서 삭제하는 FIFO 구조이다.
데크는 앞과 뒤 양쪽에서 삽입과 삭제가 모두 가능한 구조이므로, 스택은 같은 끝을 삽입과 삭제에 함께 쓰고 큐는 서로 반대 끝을 삽입과 삭제에 나누어 쓰면 된다.

NoteBase

선지별 해설

① O
push를 insert_rear로 하고 pop을 delete_rear로 하면 같은 뒤쪽 끝에서 넣고 빼므로 스택이 된다.
enqueue를 insert_rear로 하고 dequeue를 delete_front로 하면 뒤로 넣고 앞으로 빼므로 FIFO 큐가 된다.

② X
스택의 pop을 delete_front로 두면 push는 뒤에서 하고 pop은 앞에서 하게 되어 먼저 들어간 원소가 먼저 나올 수 있으므로 LIFO가 아니다.
또한 큐의 dequeue를 delete_rear로 두면 enqueue와 dequeue가 모두 뒤쪽에서 일어나 FIFO가 아니라 같은 끝 연산이 된다.
틀린 표현은 pop(s)=delete_front(d), dequeue(q)=delete_rear(d)이다. 올바르게는 스택은 push와 pop을 같은 쪽에 두고, 큐는 enqueue와 dequeue를 반대쪽에 두어야 한다.

③ X
스택을 insert_front와 delete_front로 구현하는 것은 같은 앞쪽 끝을 쓰므로 가능하다.
하지만 큐를 insert_front와 delete_front로 구현하면 삽입과 삭제가 모두 앞쪽에서 일어나 가장 최근에 넣은 원소가 바로 삭제될 수 있어 FIFO가 아니다.
틀린 표현은 dequeue(q)=delete_front(d)이다. 올바르게는 enqueue를 insert_front로 했다면 dequeue는 delete_rear가 되어야 한다.

④ X
큐를 insert_front와 delete_rear로 구현하는 것은 앞에서 넣고 뒤에서 빼므로 FIFO가 되어 맞다.
하지만 스택을 insert_front와 delete_rear로 구현하면 삽입은 앞에서 하고 삭제는 뒤에서 하게 되어 먼저 들어간 원소가 먼저 빠질 수 있으므로 LIFO가 아니다.
틀린 표현은 pop(s)=delete_rear(d)이다. 올바르게는 push를 insert_front로 했다면 pop은 delete_front가 되어야 한다.

오답 포인트

스택과 큐를 모두 데크의 두 끝으로 구현할 수 있다는 점만 기억하고, 같은 끝을 써야 하는지 반대 끝을 써야 하는지를 섞으면 ③과 ④처럼 절반만 맞는 선지에 걸리게 된다.

다시 나오면 이렇게 판단

삽입과 삭제 결과가 가장 나중 원소부터 나오면 스택이고 가장 먼저 들어간 원소부터 나오면 큐이다. 따라서 스택은 같은 끝, 큐는 반대 끝이라는 기준으로 각 연산 쌍을 바로 판정할 수 있다.

NoteBase

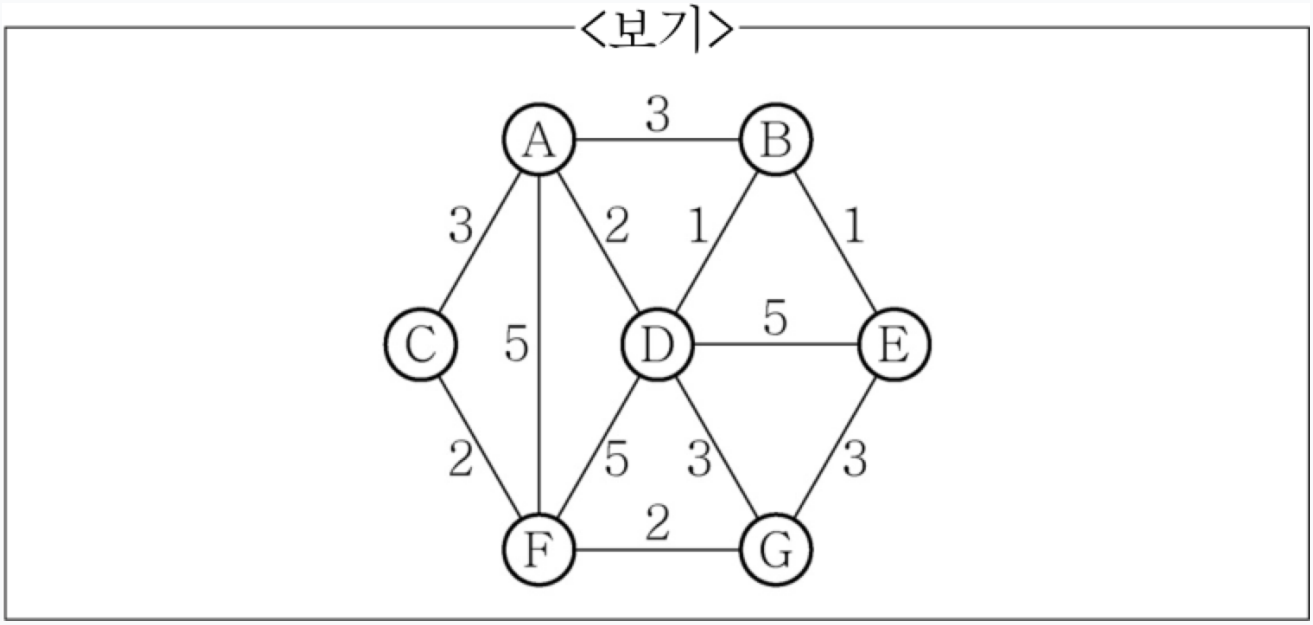
암기 포인트

데크에서 스택은 같은 끝, 큐는 반대 끝이다.

NoteBase

문 16. 다익스트라(Dijkstra)의 최단 경로 알고리즘을 이용하여 <보기>의 그래프의 정점 E에서 각 정점으로의 최단 경로를 구할 때, 정점 E를 선택한 이후 세 번째로 선택되는 정점은? (단, 각 간선 위의 숫자는 양 정점들 사이의 거리를 의미한다.)

<보기>



① A

② C

③ D

④ G

정답 4번

출제 포인트

이 문제는 가중치 그래프에서 다익스트라 알고리즘의 선택 순서를 추적할 수 있는지를 묻는다.

NoteBase

선지별 해설

1번 A: X. E에서 시작하면 초기 최단거리는 B=1, G=3, D=5이다. 먼저 B가 선택된 뒤 B를 거치면 D가 2, A가 4로 갱신되므로 A는 세 번째가 아니라 그보다 늦게 선택된다. 틀린 표현은 A가 세 번째로 선택된다는 점이며, 올바르게는 A의 최단거리는 4가 되어 G와 D 다음 순서가 된다.
2번 C: X. C는 E와 직접 연결되어 있지 않고, 가장 짧은 경로도 E-B-A-C로 7이다. 따라서 초반 선택 대상이 될 수 없다. 틀린 표현은 C가 세 번째라는 점이며, 올바르게는 C는 A가 선택된 뒤에야 유력해진다.
3번 D: X. E에서 시작할 때 B=1, D=5, G=3으로 시작하지만, B를 선택한 뒤 D는 E-B-D 경로로 2가 된다. 그래서 D는 E 다음 첫 번째가 아니라 두 번째로 선택된다. 틀린 표현은 D가 세 번째라는 점이며, 올바르게는 D는 E 다음 두 번째로 선택된다.
4번 G: O. E에서 시작하면 초기 거리는 B=1, G=3, D=5이다. 먼저 B가 선택되고, B를 거쳐 D가 2로 갱신되어 다음에 D가 선택된다. 그 뒤 남은 후보 중 최단거리는 G=3이므로 E 이후 세 번째로 선택되는 정점은 G이다.

오답 포인트

초기 인접 정점의 거리만 보고 G를 바로 두 번째로 고르면 틀린다. 다익스트라는 더 짧은 경로가 갱신되는지를 매 단계 확인해야 하므로, B를 선택한 뒤 D가 5에서 2로 줄어드는 갱신을 놓치면 순서를 잘못 판단하게 된다.

다시 나오면 이렇게 판단

시작 정점의 인접 정점으로 초기 거리를 적은 뒤, 가장 작은 값을 하나 선택하고 그 정점을 거쳐 더 짧아지는 거리만 즉시 갱신하면서 선택 순서를 추적하면 된다.

암기 포인트

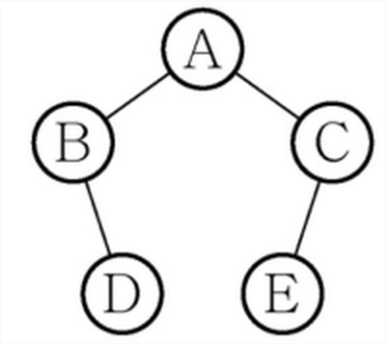
다익스트라는 매 단계에서 현재 최단거리 확정 정점을 고르고, 그 정점을 거친 거리 갱신 후 다시 최소값을 고르는 알고리즘이다.

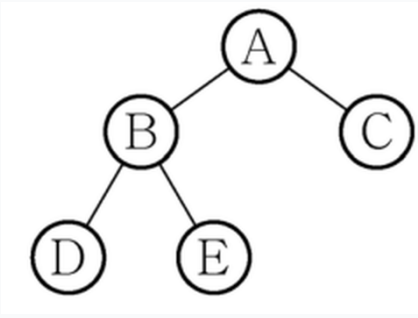
NoteBase

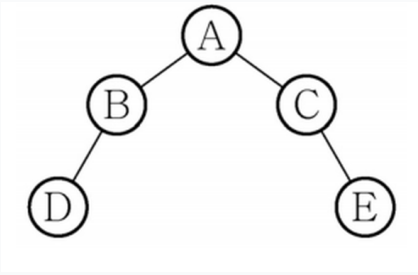
문 17. <보기>는 5개의 노드로 구성된 이진 트리를 전위, 중위, 후위 순회한 결과이다. 이 결과로 알 수 있는 이진 트리의 구성은?

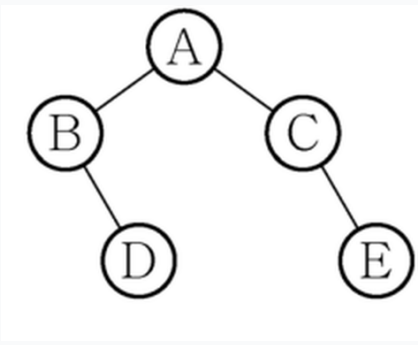
<보기>

- 전위 순회: A - B - D - C - E
- 중위 순회: B - D - A - E - C
- 후위 순회: D - B - E - C - A

① 

② 

③ 

④ 

정답 1번

NoteBase

출제 포인트

이 문제는 전위 순회에서 루트를 찾고, 중위 순회에서 좌우 부분트리의 경계를 나눈 뒤, 각 자식의 방향까지 복원하여 맞는 이진 트리 도식을 고르는 문제이다.

선지별 해설

① O이다. 전위 순회의 첫 노드 A가 루트이고, 중위 순회에서 A의 왼쪽은 B-D, 오른쪽은 E-C이므로 왼쪽 부분트리와 오른쪽 부분트리가 나뉜다. 왼쪽 부분트리는 루트 B에 오른쪽 자식 D가 붙고, 오른쪽 부분트리는 루트 C에 왼쪽 자식 E가 붙으므로 제시된 순회 결과와 모두 일치한다.
② X이다. 이 도식은 B의 왼쪽 자식이 D이고 오른쪽 자식이 E인 구조이다. 올바르게는 B는 왼쪽 자식이 없고 오른쪽 자식만 D를 가져야 하며, E는 B의 자식이 아니라 C의 왼쪽 자식이어야 한다.
③ X이다. 이 도식은 B의 왼쪽 자식이 D이고 C의 오른쪽 자식이 E인 구조이다. 올바르게는 중위 순회 B-D-A-E-C를 만족하려면 D는 B의 오른쪽 자식이어야 하고 E는 C의 왼쪽 자식이어야 한다.
④ X이다. 이 도식은 B의 오른쪽 자식이 D인 점은 맞지만 C의 오른쪽 자식이 E인 구조이다. 올바르게는 오른쪽 부분트리의 중위 순회가 E-C가 되어야 하므로 E는 C의 왼쪽 자식이어야 한다.

오답 포인트

B와 D가 연결되고 C와 E가 연결된다는 사실만 보고 넘어가면 ③이나 ④에 끌리기 쉽다. 이 문항의 함정은 연결 여부가 아니라 자식의 좌우 방향이며, 그 방향 정보는 중위 순회가 결정한다.

다시 나오면 이렇게 판단

전위 순회에서 첫 노드를 루트로 잡고, 중위 순회에서 그 루트를 기준으로 왼쪽 구간과 오른쪽 구간을 자른다. 각 구간에 같은 작업을 반복해 자식의 좌우 방향을 복원한 뒤, 마지막에 후위 순회가 루트로 끝나는지 확인하면 된다.

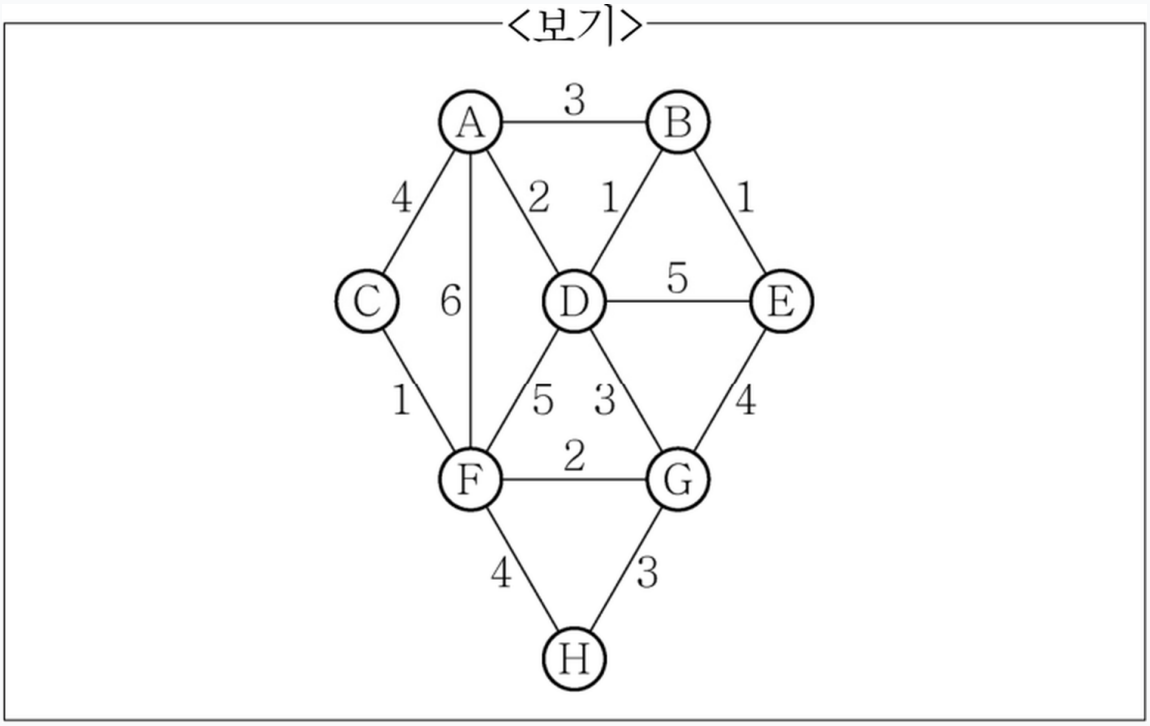
암기 포인트

전위는 루트 먼저, 중위는 왼-루-오, 후위는 루트 마지막이므로 트리 복원에서 좌우 방향은 중위 순회가 결정한다.

NoteBase

문 18. <보기>의 가중그래프(weighted graph)에 대해, Prim의 알고리즘을 적용해서 최소 비용 신장 트리(minimum spanning tree)를 구성하고자 한다. 정점 D에서 시작할 경우, 정점 D를 선택한 이후 네 번째로 선택되는 정점은?

<보기>



① E

② F

③ G

④ H

정답 3번

출제 포인트

이 문제는 가중 그래프에서 Prim 알고리즘을 시작 정점 D부터 적용할 때, 현재 선택된 정점 집합과 바깥 정점을 잇는 간선 중 최솟값을 매 단계 고르는 순서를 추적하는 문제이다.

NoteBase

선지별 해설

① X: E는 D 다음에 B가 선택된 뒤, 경계 간선 중 가장 작은 B-E의 가중치 1로 선택된다. 네 번째가 아니라 D 이후 두 번째로 선택되는 정점이다. 틀린 표현은 '네 번째'이고, 올바르게는 'D를 선택한 이후 두 번째로 선택되는 정점은 E이다.'이다.
② X: F는 G가 선택된 뒤 F-G의 가중치 2로 연결된다. 네 번째가 아니라 D 이후 다섯 번째로 선택되는 정점이다. 틀린 표현은 '네 번째'이고, 올바르게는 'D를 선택한 이후 다섯 번째로 선택되는 정점은 F이다.'이다.
③ O: D에서 시작하면 먼저 B가 가중치 1로 선택되고, 다음은 B-E의 가중치 1로 E가 선택된다. 그다음은 D-A의 가중치 2로 A가 선택되며, 이후 경계 간선 중 최소는 D-G의 가중치 3이므로 D를 선택한 이후 네 번째로 선택되는 정점은 G이다.
④ X: H는 G나 F가 먼저 트리에 들어와야 후보가 된다. 따라서 초기 네 단계 안에는 선택될 수 없고, 실제로는 마지막에 선택된다. 틀린 표현은 '네 번째'이고, 올바르게는 'H는 마지막에 선택되는 정점이다.'이다.

오답 포인트

Prim은 그래프 전체에서 가장 짧은 간선을 고르는 알고리즘이 아니라, 현재 트리와 바깥 정점을 잇는 경계 간선 중 최소를 고르는 알고리즘이다. 또 이 문항은 시작 정점 D를 뺀 뒤의 순서를 묻기 때문에 전체 선택 순서와 'D 이후 몇 번째'를 혼동하면 오답이 된다.

다시 나오면 이렇게 판단

시작 정점이 들어 있는 현재 트리를 먼저 적고, 그 집합에서 바깥으로 나가는 간선만 남겨 최솟값을 고르면 된다. 선택 순서는 D→B→E→A→G이므로 D를 선택한 이후 네 번째 정점은 G이다.

암기 포인트

Prim은 현재 트리의 경계 간선 최소 선택의 반복이며, 이 문항의 핵심 순서는 D→B→E→A→G이다.

NoteBase

문 19. <보기>의 해시 함수 h1(k)와 h2(k)를 사용하여 이중 해싱법(double hashing)으로 키(key) 값 8, 9, 2, 1, 5, 6을 가지는 레코드들을 크기 7의 공백 해시 테이블에 차례대로 삽입했을 경우 생성되는 배열의 모습으로 가장 옳은 것은? (단, h1(k)는 첫 번째 조사 위치를 결정하는 해시 함수이고, h2(k)는 충돌 발생 시 조사 간격을 결정하기 위한 해시 함수이다.)

<보기>

• h1(k) = k 7

• h2(k) = 7 - (k 5)

①

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	8	9	2	1	5	6

②

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	1	2	8	9	5	6

③

[0]	[1]	[2]	[3]	[4]	[5]	[6]
2	8	9		6	5	1

④

[0]	[1]	[2]	[3]	[4]	[5]	[6]
2	8	9	1		5	6

정답 3번

출제 포인트

이 문제는 이중 해싱에서 $h1(k)$로 첫 위치를 정하고, 충돌이 나면 $h2(k)$만큼 건너뛰며 빈 칸을 찾는 탐사 순서를 정확히 추적하는 문제이다.

올바른 순서

1. 8: $h1(8) = 1$이므로 [1]에 저장된다.
2. 9: $h1(9) = 2$이므로 [2]에 저장된다.
3. 2: $h1(2) = 2$에서 충돌하고 $h2(2) = 5$이므로 다음 조사 위치는 (2+5) 7=0이다. [0]이 비어 있어 [0]에 저장된다.
4. 1: $h1(1) = 1$에서 충돌하고 $h2(1) = 6$이므로 [0], [6] 순으로 조사한다. [0]은 차 있고 [6]이 비어 있어 [6]에 저장된다.
5. 5: $h1(5) = 5$이므로 [5]에 저장된다.
6. 6: $h1(6) = 6$에서 충돌하고 $h2(6) = 6$이므로 [5], [4] 순으로 조사한다. [5]는 차 있고 [4]가 비어 있어 [4]에 저장된다.

최종 배열은 [0]=2, [1]=8, [2]=9, [3]=공백, [4]=6, [5]=5, [6]=1이다.

NoteBase

선지별 해설

① X
- [0]이 공백으로 남아 있고 [3]=2, [4]=1로 배치된 점이 틀리다. 2는 $h1(2) = 2$ 충돌 뒤 [0]으로 가고, 1은 [1] 충돌 뒤 [0]도 충돌하므로 [6]에 들어간다.
- 올바르게는 [0]=2, [3]=공백, [4]=6, [6]=1이다.

② X
- 8과 9가 각각 [3], [4] 쪽으로 밀린 점이 틀리다. 첫 조사 위치가 비어 있으면 추가 탐사를 하지 않으므로 8은 [1], 9는 [2]에 바로 저장된다.
- 올바르게는 [1]=8, [2]=9이며, 1과 2도 각각 [6], [0]에 배치된다.

③ O
- [0]=2, [1]=8, [2]=9, [3]=공백, [4]=6, [5]=5, [6]=1이 충돌 처리 과정을 모두 반영한 최종 배열이다.

④ X
- [3]=1, [4]=공백, [6]=6으로 배치한 점이 틀리다. 1은 조사 간격 6으로 [1] 다음 [0], [6]을 확인하므로 [3]으로 가지 않고, 6도 [6] 다음 [5], [4]를 확인하므로 [6]에 남지 않는다.
- 올바르게는 [3]=공백, [4]=6, [6]=1이다.

오답 포인트

이중 해싱을 선형 조사처럼 한 칸씩 미는 방식으로 오해하면 오답이 된다. 특히 $h2(1) = 6$, $h2(6) = 6$은 조사 간격이 6이라는 뜻이지 다음 칸이 6번이라는 뜻이 아니므로, 각 단계마다 7 연산으로 실제 다음 위치를 다시 계산해야 한다.

다시 나오면 이렇게 판단

각 키마다 먼저 $h1(k)$로 시작 위치를 적고, 충돌이 날 때만 $h2(k)$를 더해 7을 취한다. 탐사식 (h1(k)+i h2(k)) 7을 삽입 순서대로 끝까지 추적하면 배열형 선지는 바로 걸러진다.

암기 포인트

이중 해싱은 시작 위치와 점프 간격을 분리한다. $h1$은 첫 칸, $h2$는 충돌 뒤 이동 폭이라는 점이 핵심이다.

NoteBase

문 20. <보기 1>은 list라는 배열에 저장된 n개의 원소를 오름차순으로 정렬하는 알고리즘을 C언어로 표현한 함수이다. 배열 list1과 list2에 저장된 값들이 <보기 2>와 같다고 할 때, 이 배열들을 <보기 1>의 함수를 이용하여 정렬한다면, ㉠ 연산이 실행되는 각각의 횟수는?

<보기 1>

```
void sort(int list[], int n)
{
        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>

	[0]	[1]	[2]	[3]	[4]
list1	1	2	3	4	5
list2	5	4	3	2	1

① list1: 0회, list2: 10회

② list1: 0회, list2: 15회

③ list1: 4회, list2: 10회

④ list1: 4회, list2: 15회

정답 1번

출제 포인트

이 문제는 삽입정렬에서 ㉠ 연산이 실제로 수행되는 횟수, 즉 key보다 큰 앞쪽 원소를 오른쪽으로 밀어내는 횟수를 단계별로 세는 문제이다.

조건 정리

주어진 함수는 삽입정렬이다.
㉠은 `list[j+1] = list[j];` 이므로, 현재 key보다 큰 원소를 한 칸씩 뒤로 미는 이동 연산이다.
구해야 할 값은 list1과 list2에서 ㉠이 각각 몇 번 실행되는지이다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 자료구조론 · 2020 · 10월

사용할 공식

삽입정렬에서 *i*번째 반복의 ㉠ 실행 횟수는 key보다 큰 앞쪽 원소의 개수이다.
전체 실행 횟수는 각 반복의 이동 횟수를 더한 값이다.
즉, 총 횟수는 Σ_i=1^n-1(i번째 key보다 큰 앞쪽 원소 수)로 센다.
이 기준을 쓰는 이유는 ㉠이 비교문이 아니라 실제 이동문이기 때문이다.

계산 과정

list1 = [1, 2, 3, 4, 5]는 이미 오름차순이므로 각 단계에서 key보다 큰 앞쪽 원소가 없다. 따라서 이동 횟수는 0회이다.
list2 = [5, 4, 3, 2, 1]는 완전한 내림차순이므로 각 단계에서 앞쪽 원소가 모두 key보다 크다.
$i = 1$에서는 4를 넣기 위해 5를 한 번 밀어 1회이다.
$i = 2$에서는 3을 넣기 위해 5, 4를 밀어 2회이다.
$i = 3$에서는 2를 넣기 위해 5, 4, 3을 밀어 3회이다.
$i = 4$에서는 1을 넣기 위해 5, 4, 3, 2를 밀어 4회이다.
따라서 list2의 총 이동 횟수는 $1 + 2 + 3 + 4 = 10$회이다.

선지별 해설

① O: list1은 이미 정렬되어 있어 ㉠이 한 번도 실행되지 않고, list2는 $1 + 2 + 3 + 4 = 10$회 실행된다.
② X: list1이 0회인 점은 맞지만 list2가 15회라는 점이 틀리다. ㉠은 이동 연산만 세므로 list2는 10회가 맞다. 틀린 표현은 `list2 15회`이고, 올바르게는 `list2 10회`이다.
③ X: list2가 10회인 점은 맞지만 list1이 4회라는 점이 틀리다. list1은 이미 오름차순이라 어떤 단계에서도 앞 원소를 밀지 않는다. 틀린 표현은 `list1 4회`이고, 올바르게는 `list1 0회`이다.
④ X: list1 4회, list2 15회가 모두 틀리다. list1은 0회, list2는 10회가 맞다.

오답 포인트

㉠의 실행 횟수를 비교 횟수나 `list[j+1] = key;`까지 포함한 대입 횟수로 착각하면 15회 같은 오답이 나온다. 이 문제는 내부 for문의 본문에 있는 이동문만 세는지, 아니면 조건 검사 전체를 세는지를 구분하는 함정이 있다.

다시 나오면 이렇게 판단

삽입정렬 횟수 문제는 먼저 무엇을 세는지 확인한 뒤, 각 단계에서 key보다 큰 앞쪽 원소가 몇 개인지만 세어 누적하면 된다. 이미 정렬된 배열은 0회, 완전 역순 배열은 $1 + 2 + \cdots + (n - 1)$회가 된다.

암기 포인트

삽입정렬의 이동 횟수는 key보다 큰 앞원소 수의 합이며, 오름차순 배열은 0회, 역순 배열은 삼각수 형태로 누적된다.

NoteBase