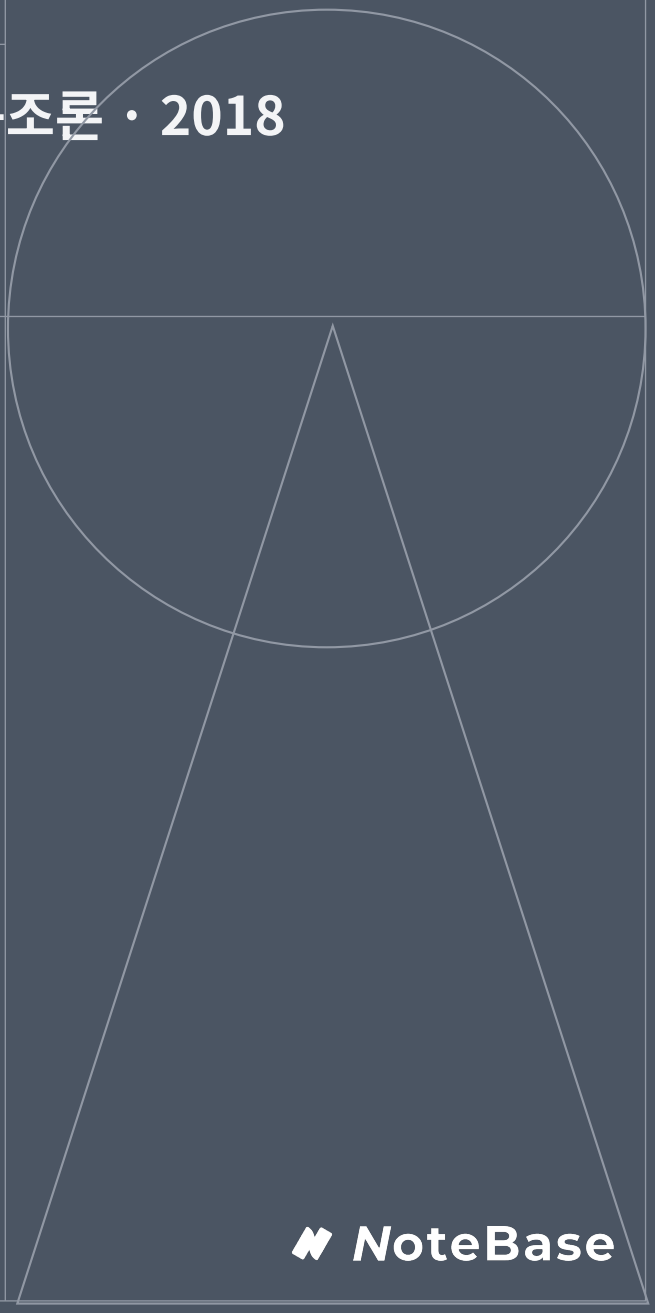
기출OX 해설집

공무원/국가직/7급 · 자료구조론 · 2018



NoteBase

기출OX 해설집

공무원/국가직/7급 · 자료구조론 · 2018

공무원/국가직/7급 · 자료구조론 · 2018

문 1. 다음 C 언어 함수에 의해 구현된 정렬 방식은?

```
void whatsort(int a[ ], int size)
{
  int i, j, temp;
  for (i = (size-1); i > 0; i--) {
    for (j = 1; j <= i; j++) {
      if (a[j-1] > a[j]) {
         temp = a[j-1];
         a[j-1] = a[j];
         a[j] = temp;
      }
    }
  }
}
```

① 삽입 정렬(insertion sort)

② 선택 정렬(selection sort)

③ 힙 정렬(heap sort)

④ 버블 정렬(bubble sort)

정답 4번

출제 포인트

이 문제는 인접한 두 원소를 반복해서 비교·교환하는 코드 패턴을 보고 버블 정렬을 식별할 수 있는지를 묻는 문제이다.

NoteBase

핵심 개념

버블 정렬은 인접한 두 원소를 비교하여 순서가 틀리면 교환하고, 한 번의 바깥 반복이 끝날 때마다 큰 값이 뒤쪽으로 밀려나는 정렬이다.
삽입 정렬은 현재 원소를 이미 정렬된 구간의 알맞은 위치에 끼워 넣는 정렬이다.
선택 정렬은 매 단계에서 최솟값이나 최댓값을 골라 한 번 교환하는 정렬이다.
힙 정렬은 힙을 구성한 뒤 루트 원소를 꺼내며 재정렬하는 정렬이다.

선지별 해설

① X: 삽입 정렬은 기준 원소를 따로 잡아 정렬된 부분에 삽입하는 방식이므로 보통 원소를 뒤로 밀면서 위치를 찾는다. 이 코드에는 삽입할 key를 보관하거나 연속 이동시키는 과정이 없다. 틀린 표현은 "삽입 정렬"이다. 올바르게는 "인접 원소를 비교·교환하는 버블 정렬"이다.
② X: 선택 정렬은 각 단계에서 최솟값 또는 최댓값의 위치를 선택한 뒤 한 번 교환하는 방식이다. 이 코드는 한 패스 안에서 여러 번 인접 교환을 수행하므로 선택 정렬의 구조가 아니다. 틀린 표현은 "선택 정렬"이다. 올바르게는 "매 비교마다 인접 원소를 교환하는 버블 정렬"이다.
③ X: 힙 정렬은 배열을 힙 구조로 만든 다음 루트와 마지막 원소를 교환하고 힙을 다시 맞추는 방식이다. 제시된 코드에는 힙 구성이나 부모·자식 비교가 전혀 없다. 틀린 표현은 "힙 정렬"이다. 올바르게는 "중첩 반복으로 인접 원소를 교환하는 버블 정렬"이다.
④ O: 안쪽 반복에서 a[j-1]와 a[j]를 비교해 순서가 반대이면 즉시 교환한다. 이런 인접 교환을 끝까지 반복하면 각 바깥 반복마다 가장 큰 값이 오른쪽 끝으로 이동하므로 버블 정렬이 맞다.

오답 포인트

정렬 문제에서는 중첩 반복문만 보고 선택 정렬이나 삽입 정렬로 오인하기 쉽다. 이 문항의 함정은 변수 이름이 아니라 비교 대상이 항상 인접한 두 원소인지, 그리고 교환이 한 패스에서 여러 번 일어나는지를 보아야 한다는 점이다.

다시 나오면 이렇게 판단

코드에서 a[j-1]와 a[j]처럼 인접 원소를 비교하고 조건이 맞으면 즉시 swap하는 패턴이 반복되면 먼저 버블 정렬을 떠올리면 된다. 반대로 최솟값 위치를 따로 찾으면 선택 정렬이고, 현재 원소를 끼워 넣으면 삽입 정렬이다.

암기 포인트

인접 비교와 연속 교환은 버블 정렬, 최솟값 선택은 선택 정렬, 끼워 넣기는 삽입 정렬이다.

NoteBase

문 2. 다음 중위 표현식(infix expression)을 후위 표현식(postfix expression)으로 변환한 후, 스택을 이용하여 후위 표현식을 계산하고자 한다. 후위 표현식의 계산 과정에서 스택에 여덟 번째로 삽입(push)되는 값은?

(4 + 2) / 3 + (6 / 2 - 1)

① 1

② 2

③ 3

④ 4

정답 3번

출제 포인트

중위 표현식을 후위 표현식으로 바꾼 뒤, 피연산자와 연산 결과가 스택에 어떤 순서로 push되는지를 추적하는 문제이다.

NoteBase

올바른 순서

1. 중위식을 후위식으로 변환한다. 변환 규칙은 각 연산자를 자신의 두 피연산자 뒤로 보내되 괄호와 우선순위(곱셈·나눗셈이 덧셈·뺄셈보다 먼저)를 지키는 것이다. 이에 따라 (4 + 2)/3은 4 2 + 3 /, (6/2 − 1)은 6 2 / 1 -가 되고, 두 부분식을 마지막 +로 합치면 후위식은 4 2 + 3 / 6 2 / 1 - +가 된다.
2. 후위식을 왼쪽부터 읽으며, 피연산자를 만나면 스택에 push하고(피연산자 push), 연산자를 만나면 위의 두 값을 pop해 계산한 뒤 그 결과를 다시 push한다(결과 push). 이때 결과의 재삽입도 별도의 push로 함께 센다.
3. 토큰을 순서대로 처리하면서 push 횟수와 그때의 스택 상태(왼쪽이 바닥, 오른쪽이 꼭대기)를 적으면 다음과 같다.

토큰 4 → 피연산자 push(1회째, 값 4) → 스택 [4]
토큰 2 → 피연산자 push(2회째, 값 2) → 스택 [4, 2]
토큰 + → 2와 4를 pop, 4+2=6을 push(3회째, 값 6) → 스택 [6]
토큰 3 → 피연산자 push(4회째, 값 3) → 스택 [6, 3]
토큰 / → 3과 6을 pop, 6/3=2를 push(5회째, 값 2) → 스택 [2]
토큰 6 → 피연산자 push(6회째, 값 6) → 스택 [2, 6]
토큰 2 → 피연산자 push(7회째, 값 2) → 스택 [2, 6, 2]
토큰 / → 2와 6을 pop, 6/2=3을 push(8회째, 값 3) → 스택 [2, 3]
토큰 1 → 피연산자 push(9회째, 값 1) → 스택 [2, 3, 1]
토큰 - → 1과 3을 pop, 3-1=2를 push(10회째, 값 2) → 스택 [2, 2]
토큰 + → 2와 2를 pop, 2+2=4를 push(11회째, 값 4) → 스택 [4]

4. 따라서 push 값의 순서는 4, 2, 6, 3, 2, 6, 2, 3, 1, 2, 4이며, 여덟 번째로 push되는 값은 6/2를 계산한 결과 3이다. 여기서 3회째의 6은 후위식에 그대로 적힌 피연산자가 아니라 첫 번째 +(4+2)의 계산 결과라는 점에 주의한다.

선지별 해설

① X: 값 1은 후위식의 6 2 / 1 - 부분에서 피연산자 1을 읽을 때 들어가므로 아홉 번째 push이다. 여덟 번째 값이 1이라는 판단은 틀리며, 올바르게는 1은 아홉 번째로 삽입된다.
② X: 값 2는 여러 번 push된다. 다섯 번째 push는 6/3=2 즉 (4+2)/3의 계산 결과이고(4+2의 결과 6이 세 번째, 3이 네 번째, 그 둘을 나눈 2가 다섯 번째로 push된다), 열 번째 push는 3-1=2 즉 (6/2-1)의 계산 결과이다. 어느 경우도 여덟 번째 push는 아니다. 여덟 번째 값이 2라는 판단은 틀리며, 올바르게는 2는 다섯 번째 또는 열 번째로 삽입된다.
③ O: 6과 2를 읽어 여섯 번째, 일곱 번째 push가 일어난 뒤 /를 계산하면 6/2=3이 다시 push되고, 이 결과 3이 정확히 여덟 번째 push 값이다.
④ X: 값 4는 마지막 +로 2+2를 계산해 얻는 전체 식의 최종 결과이므로 열한 번째 push이다. 여덟 번째 값이 4라는 판단은 틀리며, 올바르게는 4는 마지막 열한 번째로 삽입된다.

NoteBase

오답 포인트

피연산자만 세고 연산 결과의 재삽입을 빼먹으면 push 횟수가 어긋난다. 특히 6 2 /에서 나온 결과 3을 여덟 번째 push로 세지 않으면 답이 1이나 2로 밀리기 쉽다. 또한 후위식에 6이 두 번 나오지만 세 번째 push의 6은 4+2의 계산 결과이고 여섯 번째 push의 6만 실제 피연산자라는 점을 구분해야 한다.

다시 나오면 이렇게 판단

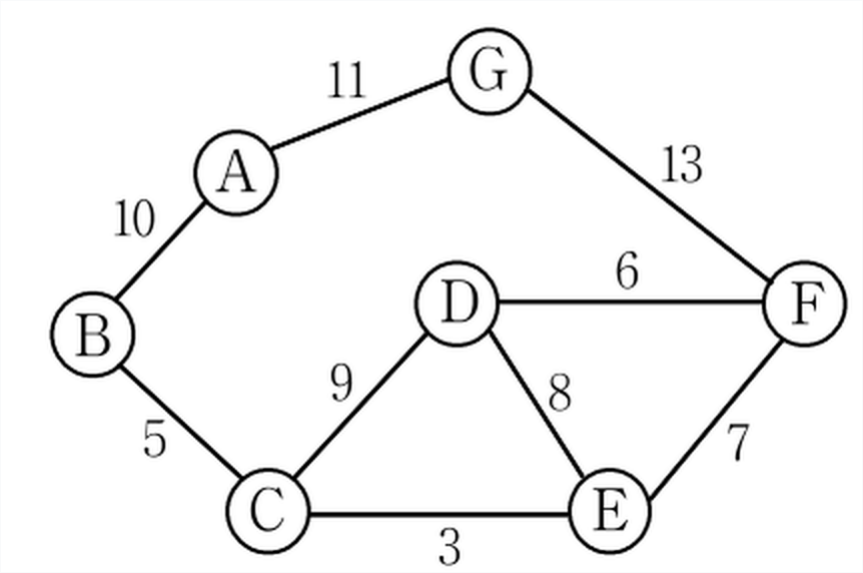
후위식 계산 문제는 피연산자 push와 연산 결과 재push를 모두 한 줄씩 번호 매겨 추적하면 된다.
토큰·동작·스택 상태를 나란히 적으면 몇 번째 push인지 헷갈리지 않으며, 중간 결과도 독립된 push로 세어야 한다.

암기 포인트

후위식의 push 횟수는 숫자만 세는 것이 아니라 계산 결과의 재삽입까지 함께 세는 것이다.

NoteBase

문 3. Prim 알고리즘을 사용하여 다음 가중 그래프(weighted graph)의 최소 비용 신장 트리(minimum cost spanning tree)를 구성할 때, 최소 비용과 마지막으로 선택되는 간선은? (단, 시작 정점은 A이다)



① 42, (A, G)

② 43, (A, G)

③ 44, (F, G)

④ 45, (F, G)

정답 1번

출제 포인트

이 문제는 가중 그래프에서 Prim 알고리즘을 A에서 시작해 한 단계씩 적용하면서, 현재 트리와 연결되는 간선 중 최솟값만 선택하는 과정을 정확히 추적하는지 묻는 문제이다.

NoteBase

선지별 해설

① O: A에서 시작하면 먼저 (A,B)=10, 다음 (B,C)=5, 다음 (C,E)=3, 다음 (E,F)=7, 다음 (F,D)=6, 마지막으로 (A,G)=11을 선택한다. 총비용은 10+5+3+7+6+11=42이고 마지막 간선도 (A,G)이므로 옳다.

② X: 마지막 간선을 (A,G)로 본 점은 맞지만 총비용 43이 틀리다. Prim 적용 결과 최소 비용은 42이다. 틀린 표현은 "43"이고, 올바르게는 "42, (A,G)"이다.

③ X: 마지막 간선을 (F,G)로 보면 안 된다. D까지 포함한 뒤 G를 연결하는 후보는 (A,G)=11과 (F,G)=13인데 더 작은 11을 택하므로 마지막 간선은 (A,G)이다. 총비용도 44가 아니라 42이다. 틀린 표현은 "44, (F,G)"이고, 올바르게는 "42, (A,G)"이다.

④ X: (F,G)=13을 마지막에 택하면 Prim의 최소 선택 원칙에 어긋난다. G를 연결할 때는 (A,G)=11이 더 작으므로 (F,G)는 선택되지 않는다. 따라서 총비용 45도 성립하지 않는다. 틀린 표현은 "45, (F,G)"이고, 올바르게는 "42, (A,G)"이다.

오답 포인트

많이 틀리는 지점은 전체 간선 중 작은 값만 나열하는 것이다. Prim은 항상 현재까지 선택된 정점 집합과 바깥 정점을 잇는 간선만 비교한다. 그래서 마지막 단계에서 G를 붙일 때도 (F,G)=13이 아니라 경계 간선 중 더 작은 (A,G)=11을 골라야 한다.

다시 나오면 이렇게 판단

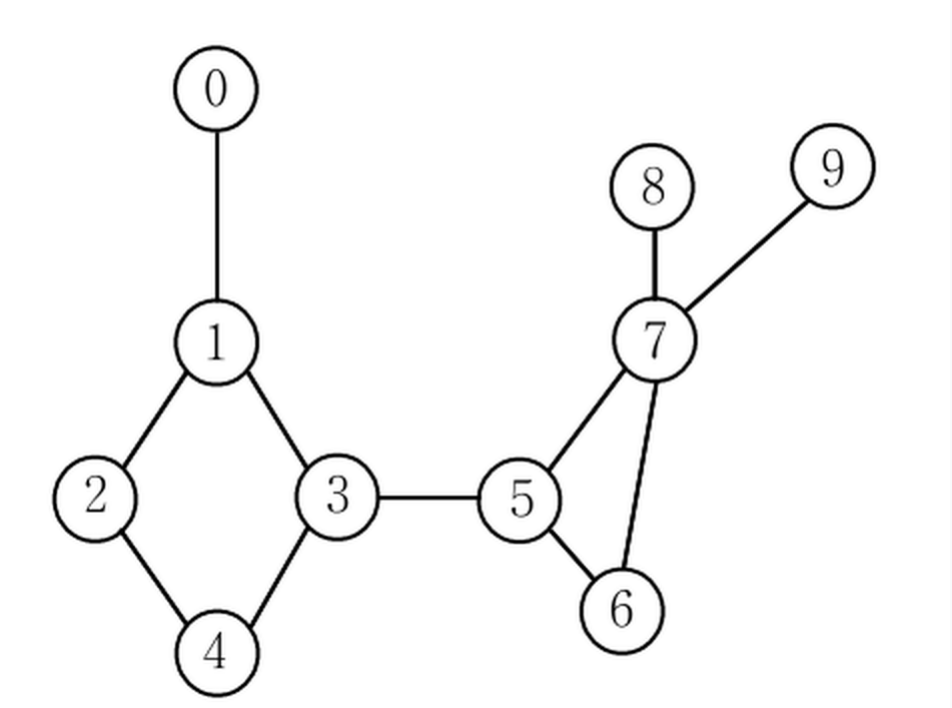
Prim 문제는 시작 정점을 고정한 뒤 현재 트리 안 정점과 밖 정점을 잇는 후보 간선만 매 단계 적어 두면 된다. 각 단계에서 그 후보들 중 최소 가중치를 선택하면 마지막 간선과 총비용이 함께 정리된다.

암기 포인트

Prim은 현재 트리의 경계 간선 중 최소를 고르는 알고리즘이므로, 전체 최소 간선이 아니라 경계 최소 간선을 계속 선택한다.

NoteBase

문 4. 시작 정점이 6일 때, 다음 그래프에 대한 깊이 우선 탐색(DFS: Depth First Search)의 방문 순서는? (단, 인접한 정점들은 오름차순으로 방문한다)



① 6, 5, 3, 1, 0, 2, 4, 7, 8, 9

② 6, 5, 7, 3, 8, 9, 1, 4, 0, 2

③ 6, 5, 3, 4, 2, 1, 0, 7, 8, 9

④ 6, 5, 7, 3, 1, 4, 0, 2, 8, 9

정답 1번

출제 포인트

이 문제는 그래프 도식에서 각 정점의 인접 관계를 읽고, DFS가 현재 정점에서 방문 가능한 가장 작은 번호의 인접 정점으로 끝까지 내려간다는 순서를 판단하는 문제이다. 도식에서 각 정점의 인접 정점을 정리하면 0={1}, 1={0, 2, 3}, 2={1, 4}, 3={1, 4, 5}, 4={2, 3}, 5={3, 6, 7}, 6={5, 7}, 7={5, 6, 8, 9}, 8={7}, 9={7}이다. 이 인접 리스트를 먼저 적어 두면 각 단계에서 다음 정점을 착오 없이 고를 수 있다.

NoteBase

선지별 해설

① O: 6의 인접 정점은 5, 7이므로 먼저 5로 간다. 5의 인접 정점은 3, 6, 7이므로 미방문이면서 가장 작은 3으로 가고, 3에서는 1로, 1에서는 0으로 내려간다. 0은 인접 정점이 1뿐이라 더 갈 곳이 없어 1로 백트래킹하고, 1의 남은 미방문 정점 2로 간 뒤 2의 인접 정점 4를 방문한다(경로 0→1→2→4). 4에서도 갈 곳이 없어 2, 1, 3을 거쳐 5까지 백트래킹한 뒤(경로 4→3→5), 5의 남은 미방문 정점 7로 올라가 7의 인접 정점 8, 9를 차례로 방문한다. 따라서 6, 5, 3, 1, 0, 2, 4, 7, 8, 9가 맞다.
② X: 5 다음에 7을 방문한 부분이 틀렸다. 5의 인접 정점은 3, 6, 7이므로 미방문 정점 중 오름차순 우선은 3이다. 틀린 표현은 "6, 5, 7"이고, 올바르게는 "6, 5, 3"부터 시작해 6, 5, 3, 1, 0, 2, 4, 7, 8, 9 순으로 진행해야 한다.
③ X: 3 다음에 4를 방문한 부분이 틀렸다. 3의 인접 정점은 1, 4, 5이므로 미방문 정점 중 먼저 가야 할 정점은 1이다. 틀린 표현은 "6, 5, 3, 4"이고, 올바르게는 "6, 5, 3, 1"로 이어져 최종적으로 6, 5, 3, 1, 0, 2, 4, 7, 8, 9가 된다.
④ X: 선지 ④의 순서 6, 5, 7, 3, 1, 4, 0, 2, 8, 9에서 가장 먼저 어긋나는 지점은 5 다음이다. 5의 인접 정점은 3, 6, 7이므로 미방문 중 가장 작은 3으로 가야 하는데 7로 갔다. 이 첫 오류 이후의 전개는 모두 정답과 달라진다(예컨대 1의 인접 정점은 0, 2, 3뿐이라 1에서 4로 바로 갈 수도 없다). 올바른 순서는 ①과 같이 6, 5, 3, 1, 0, 2, 4, 7, 8, 9이다.

오답 포인트

DFS를 단순히 "작은 번호부터 한 번씩 방문하는 순회"로 외우면 5에서 7로 가거나 3에서 4로 가는 오답이 나온다. 이 문제의 함정은 오름차순 정렬보다도 "현재 정점에서 갈 수 있는 가장 작은 미방문 정점으로 끝까지 내려간다"는 깊이 우선 원칙에 있다.

다시 나오면 이렇게 판단

현재 정점의 인접 정점을 오름차순으로 적고, 가장 작은 미방문 정점으로 계속 내려간 뒤 더 이상 갈 곳이 없을 때만 직전 정점으로 되돌아가면 된다. DFS 순서는 "전진 우선, 막히면 백트래킹"으로 판단된다.

암기 포인트

DFS는 작은 번호를 고르되, 한 갈래를 끝까지 내려간 뒤 돌아오는 순회이다.

NoteBase

문 5. C 언어 함수로 구현된 <정렬 알고리즘>과 크기가 5인 배열 list를 이용하여 sort(list, 5)를 수행하였다. <설명>에서 옳은 것만을 모두 고르면? (단, sort(list, 5) 수행 전에 list 배열에 4, 9, 8, 6, 3이 순서대로 저장되어 있다)

<정렬 알고리즘>

```
#define SWAP(x, y, t) ((t)=(x), (x)=(y), (y)=(t))
void sort(int list[ ], int n)
{
  int i, j, least, temp;
  for (i = 0; i < n - 1; i++) {
    least = i;
    for (j = i + 1; j < n; j++)
      if (list[j] < list[least]) least = j;
    SWAP(list[i], list[least], temp);
  }
}
```

<설명>

ㄱ. sort(list, 5)의 수행이 완료될 때까지 SWAP은 4회 수행된다.

ㄴ. SWAP이 두 번째 수행된 후 배열 list에는 3, 4, 8, 6, 9가 순서대로 저장되어 있다.

ㄷ. 정렬 알고리즘의 시간 복잡도를 빅세타(Θ) 표기법으로 표현한 것은 $\Theta(n \log n)$이다.

① ㄱ

② ㄱ, ㄴ

③ ㄱ, ㄷ

④ ㄴ, ㄷ

정답 2번

출제 포인트

이 문제는 선택 정렬의 바깥 반복이 진행될 때마다 최솟값 위치를 찾고 SWAP을 수행하는 절차와 그 시간 복잡도를 함께 묻는 문제이다.

NoteBase

올바른 순서

1. i=0에서 전체 구간의 최솟값 3을 찾아 list[0]과 바꿔 배열이 3, 9, 8, 6, 4가 된다.
2. i=1에서 남은 구간의 최솟값 4를 찾아 list[1]과 바꿔 배열이 3, 4, 8, 6, 9가 된다.
3. i=2에서 남은 구간의 최솟값 6을 찾아 list[2]와 바꿔 배열이 3, 4, 6, 8, 9가 된다.
4. i=3에서도 코드상 SWAP(list[3], list[3], temp)가 한 번 더 실행되어 총 SWAP 호출 수는 4회가 된다.

왜 이 순서인가: 선택 정렬은 각 바깥 반복에서 미정렬 구간의 최솟값 하나를 앞자리로 확정한 뒤 다음 위치로 넘어가는 구조이기 때문이다.

선지별 해설

① X: ①은 ㄱ만 고른 선택지이다. ㄱ은 맞지만 ㄴ도 맞으므로 불완전하다. 올바르게는 ㄱ, ㄴ만 고른 ②가 맞다.
② O: ㄱ은 바깥 for문이 i=0, 1, 2, 3으로 총 4번 돌 때마다 SWAP 문장이 매번 실행되므로 맞다. ㄴ도 두 번째 SWAP 직후 배열이 3, 4, 8, 6, 9가 되므로 맞다.
③ X: ③은 ㄷ을 포함하므로 틀리다. 선택 정렬은 각 i마다 남은 원소를 끝까지 비교하므로 비교 횟수가 대략 (n-1)+(n-2)+...+1이 되어 Θ(n^2)이다. 올바르게는 ㄱ, ㄴ만 고른 ②가 맞다.
④ X: ④도 ㄷ을 포함하므로 틀리다. 틀린 표현은 'Θ(nlogn)'이고, 올바르게는 선택 정렬의 시간 복잡도는 Θ(n^2)이다. 따라서 정답 조합은 ㄴ, ㄷ이 아니라 ㄱ, ㄴ이다.

오답 포인트

ㄱ에서 실제 원소 교환이 없으면 SWAP을 세지 않는다고 착각하기 쉽다. 그러나 이 코드는 least==i여도 SWAP 매크로를 호출한다. 또 단순 정렬이라고 해서 병합 정렬이나 힙 정렬의 Θ(nlogn)과 혼동하면 ㄷ을 참으로 오판하기 쉽다.

다시 나오면 이렇게 판단

정렬 코드 추적 문제는 바깥 반복 횟수, 각 반복이 끝난 뒤 확정되는 자리, 코드에 적힌 연산 문장의 실제 실행 여부를 먼저 본다. 복잡도는 안쪽 반복이 남은 구간 전체를 매번 훑는지 확인하면 판단할 수 있다.

암기 포인트

선택 정렬은 한 번 돌 때마다 최솟값 하나를 앞에 고정하며, 전체 비교 횟수는 삼각합 형태이므로 시간 복잡도는 Θ(n^2)이다.

NoteBase

문 6. 공백 AVL 트리에 데이터를 순서대로 삽입하여 AVL 트리를 구성할 때, 나머지와 다른 모습의 AVL 트리가 구성되는 데이터 키(key)들의 순서는?

① 3, 4, 6, 5, 7, 8

② 5, 4, 7, 3, 6, 8

③ 7, 5, 8, 6, 4, 3

④ 8, 7, 5, 4, 6, 3

정답 1번

출제 포인트

AVL 트리는 같은 키 집합이라도 삽입 순서에 따라 회전이 일어나는 위치가 달라져 최종 트리 모양이 달라질 수 있음을 묻는 문제이다.
AVL 트리는 각 노드의 균형 인수(BF = 왼쪽 서브트리 높이 − 오른쪽 서브트리 높이)의 절댓값이 1을 넘는 순간, 불균형이 처음 생긴 노드에서 회전으로 균형을 맞춘다. 회전 유형은 그 노드에서 무거운 쪽으로 두 단계 내려간 방향으로 정해진다. 왼쪽-왼쪽(LL)은 우회전, 오른쪽-오른쪽(RR)은 좌회전으로 한 번에 풀고, 왼쪽-오른쪽(LR)과 오른쪽-왼쪽(RL)은 두 번 회전한다. 오름차순 삽입(예: 3→4→6)은 오른쪽으로만 이어져 RR형이라 좌회전으로, 내림차순 삽입(예: 8→7→5)은 왼쪽으로만 이어져 LL형이라 우회전으로 풀린다.

올바른 순서

①은 3→4→6에서 한 번 균형 조정이 일어나 4가 루트가 되고, 이후 5→7→8을 넣으면 마지막에 다시 좌회전이 일어나 최종 루트가 6이 된다.
②~④는 중간 회전 시점은 달라도 최종적으로 루트 5, 왼쪽 4-3, 오른쪽 7-6-8 구조로 수렴한다.
왜 이 순서: ①은 큰 값이 뒤에 연속으로 들어오면서 오른쪽 부분트리의 높이가 다시 커져 루트가 6으로 바뀌고, ②~④는 5를 중심으로 좌우가 채워져 같은 균형 구조가 만들어진다.

NoteBase

선지별 해설

① O
- 삽입 과정: 3, 4를 넣은 뒤 6을 넣으면 3-4-6이 오른쪽으로만 이어져 3에서 BF가 −2인 RR형이 되고, 좌회전이 일어나 루트가 4가 된다. 이어 5, 7을 넣고 8을 넣으면 4의 오른쪽 서브트리가 다시 무거워져(BF −2) 4에서 좌회전이 일어나 루트가 6으로 바뀐다.
- 최종 트리는 루트 6, 왼쪽 부분트리 4-3-5, 오른쪽 부분트리 7-8 형태가 된다. ②~④와 루트부터 다르므로 나머지와 다른 모양의 AVL 트리가 된다.

② X
- 삽입 과정: 5, 4, 7은 그대로 균형 잡힌 루트 5가 되고, 3, 6, 8을 넣어도 어느 노드의 BF 절댓값도 1을 넘지 않아 회전이 한 번도 일어나지 않는다.
- 최종 구조는 루트 5, 왼쪽 4-3, 오른쪽 7-6-8이 된다.
- 틀린 표현: 나머지와 다른 모양의 AVL 트리가 된다.
- 올바르게는: ③, ④와 같은 모양의 AVL 트리가 된다.

③ X
- 삽입 과정: 7, 5, 8, 6, 4까지는 루트 7이 유지되며 왼쪽(5 쪽)이 무거워지는 상태(7의 BF +1)이다. 마지막 3을 넣으면 5-4-3이 LL형이 되어 7에서 BF가 +2가 되고, 7에서 우회전이 일어나 루트 5 구조로 정리된다.
- 최종 구조는 ②, ④와 같은 루트 5, 왼쪽 4-3, 오른쪽 7-6-8이다.
- 틀린 표현: 나머지와 다른 모양의 AVL 트리가 된다.
- 올바르게는: ②, ④와 같은 모양의 AVL 트리가 된다.

④ X
- 삽입 과정: 8, 7, 5는 왼쪽으로만 이어지는 LL형이라 8에서 BF가 +2가 되고 우회전이 일어나 루트가 7이 된다(내림차순 삽입이므로 좌회전이 아니라 LL형 우회전임에 유의한다). 이후 4, 6을 넣고 3을 넣으면 5-4-3이 다시 LL형이 되어 7에서 우회전이 일어나 최종적으로 루트 5가 된다.
- 최종 구조는 루트 5, 왼쪽 4-3, 오른쪽 7-6-8이다.
- 틀린 표현: 나머지와 다른 모양의 AVL 트리가 된다.
- 올바르게는: ②, ③과 같은 모양의 AVL 트리가 된다.

오답 포인트

같은 키 집합 {3, 4, 5, 6, 7, 8}이므로 결국 같은 AVL 트리가 될 것이라고 보면 틀리기 쉽다. AVL은 삽입 순간 불균형이 처음 발생한 노드에서 즉시 회전하므로, 어느 시점에 어떤 노드가 루트로 올라오는지가 최종 모양을 바꾼다.

NoteBase

다시 나오면 이렇게 판단

AVL 순서 문제는 모든 과정을 완전히 그리기보다 삽입 직후 처음 균형이 깨지는 노드와 회전 뒤 새 루트가 누구인지부터 추적하면 된다. 각 선택지의 최종 루트와 좌우 자식 배치만 비교해도 다른 모양을 빠르게 가를 수 있다.

암기 포인트

같은 키 집합이라도 AVL 트리는 삽입 순서가 다르면 회전 이력이 달라지고 최종 트리 모양도 달라진다.

NoteBase

문 7. 다음 키(key) 값을 갖는 9개의 데이터를 순서대로 삽입하여 이진 탐색 트리(binary search tree)를 구성하였다. 구성된 이진 탐색 트리에서 루트 노드를 삭제한 후 구성되는 이진 탐색 트리에 대한 설명으로 옳지 않은 것은?

5, 6, 2, 8, 4, 1, 9, 3, 7

① 루트 노드의 키는 4와 6 중 한 개의 값을 가진다.

② 중위 순회(inorder traversal)할 경우 키 값이 1, 2, 3, 4, 6, 7, 8, 9인 노드들을 순서대로 방문한다.

③ 모든 단말 노드들의 키 값들의 합은 21이다.

④ 모든 노드들의 차수(degree)들의 합은 7이다.

정답 3번

출제 포인트

이 문제는 주어진 삽입 순서로 이진 탐색 트리를 만든 뒤 루트 삭제 규칙을 적용하여 삭제 후 트리의 구조와 성질을 판단하는 문제이다.

사례 정리

핵심 사실 1: 삽입 순서는 5, 6, 2, 8, 4, 1, 9, 3, 7이다.
핵심 사실 2: 처음 루트는 5이고, 왼쪽 서브트리의 최대값은 4, 오른쪽 서브트리의 최소값은 6이다.
판단에 중요한 조건: 루트 5를 삭제하면 대체 루트는 중위 전임자 4 또는 중위 후속자 6이 된다.

핵심 개념

이진 탐색 트리는 각 노드에 대해 왼쪽 서브트리의 키는 더 작고 오른쪽 서브트리의 키는 더 큰 트리이다. 자식이 둘인 노드를 삭제할 때는 중위 전임자 또는 중위 후속자로 그 자리를 대체하고, 대체에 사용한 노드를 원래 자리에서 한 번 더 제거한다.

적용 기준

이 사례에서는 루트 삭제 후 결과가 전임자 4를 쓰는 경우와 후속자 6을 쓰는 경우 두 가지로 나뉠 수 있는지를 먼저 본다. 두 경우 모두 남는 키 집합은 1, 2, 3, 4, 6, 7, 8, 9로 같고, 중위 순회 결과도 항상 오름차순이 된다.

NoteBase

선지별 해설

① O
루트가 두 자식을 가진 노드이므로 삭제 후 루트는 왼쪽 서브트리 최대값 4 또는 오른쪽 서브트리 최소값 6으로 대체된다.
② O
이진 탐색 트리의 중위 순회 결과는 항상 오름차순이다. 루트 5를 삭제한 뒤 남는 키는 1, 2, 3, 4, 6, 7, 8, 9이므로 방문 순서는 이 순서가 된다.
③ X
전임자 4를 루트로 쓰면 단말 노드는 1, 3, 7, 9이고 합은 20이다. 후속자 6을 루트로 써도 단말 노드는 1, 3, 7, 9이고 합은 20이다. '모든 단말 노드들의 키 값들의 합은 21이다'가 틀린 표현이며, 올바르게는 '모든 단말 노드들의 키 값들의 합은 20이다'이다.
④ O
트리의 모든 노드 차수의 합은 전체 간선 수와 같고, 노드가 8개이면 간선은 7개이다. 따라서 삭제 후 모든 노드들의 차수 합은 7이다.

오답 포인트

③은 삭제된 루트 5만 제외하고 단말 노드 구성을 다시 확인하지 않으면 틀리기 쉽다. 루트를 4로 바꾸는 경우와 6으로 바꾸는 경우를 모두 따져도 단말 노드는 1, 3, 7, 9로 같다는 점이 함정이다.

다시 나오면 이렇게 판단

루트 삭제가 나오면 먼저 대체 노드가 중위 전임자인지 후속자인지를 정하고, 그다음 실제로 이동하는 노드와 원래 자리에서 사라지는 노드를 함께 반영해 단말 노드와 차수를 다시 계산한다.

암기 포인트

자식이 둘인 BST 노드 삭제는 전임자 또는 후속자로 대체한 뒤 원래 그 노드를 제거하는 방식으로 정리된다.

NoteBase

문 8. 다음 인접 행렬은 5개 도시 간의 직접 거리를 보여준다. 인천-대구 구간 최단 경로 거리와 광주-대구 구간 최단 경로 거리 사이의 차이 값은? (단, ∞는 두 도시 간에 직접적인 연결이 없음을 의미한다)

	대전	광주	서울	인천	대구
대전	0	10	15	25	9
광주	10	0	18	20	∞
서울	15	18	0	3	17
인천	25	20	3	0	∞
대구	9	∞	17	∞	0

① 1

② 2

③ 3

④ 4

정답 1번

출제 포인트

이 문제는 인접 행렬에서 직접 연결이 없는 구간을 중간 도시를 거쳐 계산한 뒤 두 최단거리의 차이를 구하는 문제이다.

조건 정리

주어진 값: 인천-대구와 광주-대구는 직접 연결이 없어 값이 ∞로 표시된다.
구해야 할 값: 인천-대구 최단거리와 광주-대구 최단거리의 차이값이다.
주의할 조건: ∞는 두 도시 사이에 직접 간선이 없다는 뜻이므로, 다른 도시를 경유한 경로는 계속 비교해야 한다.

사용할 공식

최단 경로는 시작 도시에서 도착 도시까지 가능한 경로들의 거리 합 중 최솟값이다.
이 원리를 쓰는 이유: 표에는 직접 거리만 주어졌으므로, ∞가 나온 구간은 대전이나 서울을 거치는 경로들의 합을 비교해야 하기 때문이다.

NoteBase

계산 과정

1. 인천-대구의 경로를 비교하면 인천-서울-대구는 3+17=20, 인천-대전-대구는 25+9=34,
인천-광주-대전-대구는 20+10+9=39, 인천-서울-대전-대구는 3+15+9=27이다. 따라서 인천-대구 최단거리는 20이다.
2. 광주-대구의 경로를 비교하면 광주-대전-대구는 10+9=19, 광주-서울-대구는 18+17=35,
광주-인천-서울-대구는 20+3+17=40이다. 따라서 광주-대구 최단거리는 19이다.
3. 두 최단거리의 차이는 20-19=1이다.

선지별 해설

① O: 인천-대구 최단거리는 20이고 광주-대구 최단거리는 19이므로 차이는 1이다.
② X: 틀린 표현은 차이값을 2로 본 부분이다. 올바르게는 20-19=1이므로 차이값은 2가 아니라 1이다.
③ X: 틀린 표현은 차이값을 3으로 본 부분이다. 올바르게는 인천-대구 20, 광주-대구 19이므로 차이값은 3이 아니라 1이다.
④ X: 틀린 표현은 차이값을 4로 본 부분이다. 올바르게는 두 최단거리의 차는 4가 아니라 1이다.

오답 포인트

∞를 경로 자체가 없다는 뜻으로 오해하면 계산이 막힌다. 이 문제의 함정은 직접 연결이 없더라도 서울이나 대전을 경유하면 더 짧은 경로가 생긴다는 점을 놓치게 만드는 데 있다.

다시 나오면 이렇게 판단

인접 행렬에서 ∞가 보이면 먼저 직접 연결이 없다는 뜻으로 해석하고, 그다음 가능한 2단계 경로와 3단계 경로의 합을 차례로 비교해 최솟값을 고르면 된다.

암기 포인트

인접 행렬의 ∞는 도착 불가가 아니라 직접 간선 없음이다. 최단 경로 문제는 경유 경로의 합까지 비교해야 한다.

NoteBase

문 9. 키(key) 집합에 속하는 각 키에 대해 탐색이 일어날 확률이 알려져 있을 때, 최적 이진 탐색 트리(optimal binary search tree)를 구성함으로써 한 번의 탐색에 대한 평균 비교 횟수를 최소화할 수 있다. $p(x)$를 키 x에 대한 탐색 확률이라고 할 때, 키 집합 $\{1, 2, 3\}$에 속하는 각 키에 대한 탐색 확률은 다음과 같다. 키 값이 각각 1, 2, 3인 3개의 데이터를 사용하여 구성한 최적 이진 탐색 트리에서 탐색 1회당 평균 비교 횟수는? (단, $p(1) + p(2) + p(3) = 1$이므로, 1, 2, 3 외의 키를 탐색하는 경우는 없다)

$p(1) = 0.6,\ \ p(2) = 0.3,\ \ p(3) = 0.1$

① 1.4

② 1.5

③ 1.6

④ 1.7

정답 2번

출제 포인트

탐색 확률이 주어졌을 때 최적 이진 탐색 트리의 구조를 정하고, 평균 비교 횟수인 확률 가중 깊이 합을 계산하는 문제이다.

조건 정리

주어진 값:
- $p(1) = 0.6, p(2) = 0.3, p(3) = 0.1$
- 실패 탐색은 없으므로 성공 탐색만 평균에 반영된다.

구해야 할 값:
- 최적 이진 탐색 트리에서 탐색 1회당 평균 비교 횟수

주의할 조건:
- 비교 횟수는 루트 깊이 1부터 센다.
- BST이므로 키의 순서 1, 2, 3이 유지되어야 한다.

사용할 공식

평균 비교 횟수는 각 키의 탐색 확률과 그 키를 찾을 때의 비교 횟수를 곱해 더한 값이다.

$E = \sum p(i) \times c_i$

여기서 c_i는 키 i의 비교 횟수이다.

이 공식을 쓰는 이유:

이 문제는 각 키가 서로 다른 확률로 검색되므로, 단순 평균이 아니라 확률을 가중치로 둔 평균 비교 횟수를 구해야 하기 때문이다.

NoteBase

계산 과정

1. 루트를 1로 두면 BST 조건 때문에 2는 오른쪽 자식, 3은 그 오른쪽 자식이 된다.
2. 이때 비교 횟수는 $c_1 = 1$, $c_2 = 2$, $c_3 = 3$이다.
3. 평균 비교 횟수는

$0.6 \times 1 + 0.3 \times 2 + 0.1 \times 3 = 0.6 + 0.6 + 0.3 = 1.5$

이다.

4. 루트를 2로 두면 $0.6 \times 2 + 0.3 \times 1 + 0.1 \times 2 = 1.7$이고, 루트를 3으로 두는 경우는 이보다 더 크다.
5. 따라서 최적 이진 탐색 트리의 평균 비교 횟수는 1.5이다.

선지별 해설

① X

- 제시값 1.4는 틀리다. 최적 구조는 루트 1, 오른쪽 자식 2, 그 오른쪽 자식 3이므로 평균 비교 횟수는 1.5이다.
- 1.4는 3의 비교 횟수를 잘못 2번으로 본 값 ($0.6 \times 1 + 0.3 \times 2 + 0.1 \times 2$)인데, BST에서 1이 루트이고 2도 포함되면 3은 깊이 3이 된다.
- 틀린 값 1.4 → 올바르게는 1.5이다.

② O

- 루트를 1로 잡으면 탐색 확률이 가장 큰 키가 한 번 비교로 끝나고, 나머지 키는 깊이 2와 3이 된다.
- 따라서 평균 비교 횟수는 $0.6 \times 1 + 0.3 \times 2 + 0.1 \times 3 = 1.5$이므로 옳다.

③ X

- 제시값 1.6은 틀리다. 실제 최적값은 1.5이다.
- 1.6은 가중합 계산을 잘못 더한 값이다. $0.6 + 0.6 + 0.3 = 1.5$인데 마지막 합산을 잘못하면 이런 오답이 나온다.
- 틀린 값 1.6 → 올바르게는 1.5이다.

④ X

- 제시값 1.7은 틀리다. 1.7은 키 2를 루트로 둔 트리의 평균 비교 횟수이다.
- 키 2를 루트로 두면 비교 횟수는 $c_1 = 2$, $c_2 = 1$, $c_3 = 2$여서 $0.6 \times 2 + 0.3 \times 1 + 0.1 \times 2 = 1.7$이 된다. 하지만 이는 최적 트리가 아니다.
- 틀린 값 1.7 → 올바르게는 1.5이다.

오답 포인트

확률이 가장 큰 키를 무조건 가운데 값 2에 둔다고 생각하면 ④처럼 1.7을 고르기 쉽다. 이 문제는 키의 대소 관계를 지키는 BST 조건과 확률 가중 평균을 함께 봐야 하므로, 루트 후보별 가중합을 직접 비교해야 한다.

NoteBase

다시 나오면 이렇게 판단

최적 이진 탐색 트리 문제는 먼저 가능한 루트를 정하고, 각 키의 깊이를 적은 뒤, $\sum p(i) \times c_i$를 계산해 가장 작은 값을 고르면 된다. 키 순서는 유지되고 확률이 큰 키를 더 얕게 두는 구조가 유리하다는 점을 함께 본다.

암기 포인트

최적 이진 탐색 트리의 평균 비교 횟수는 확률과 깊이의 가중합이며, 정답은 가장 작은 $\sum p(i) \times c_i$이다.

NoteBase

문 10. 다음의 해시 함수들을 사용하는 이중 해싱(double hashing)에서 해시 테이블의 크기는 7이며 0부터 6까지의 인덱스를 가진다. $h(x)$는 첫 번째 조사 위치를 결정하는 기본적인 해시 함수이고 $f(x)$는 충돌 발생 시 조사 위치 간격을 결정하는 추가 해시 함수로서, i번째 충돌 발생 시 다음 조사 위치를 결정하는 해시 함수는 $h_i(x)$가 된다. 공백 해시 테이블에 일련의 키(key) 값 9, 10, 2, 3, 16, 13, 11을 가지는 레코드들을 순서대로 삽입한 후, 해시 테이블의 위치별 키 값을 보여주는 것은? (단, 해시 테이블 버킷(bucket)당 슬롯(slot) 수는 1개이다)

h(x) = x 7
f(x) = 5 - (x 5)
h_i(x) = (h(x) + i × f(x)) 7

①

[0]	[1]	[2]	[3]	[4]	[5]	[6]
13	16	9	10	11	2	3

②

[0]	[1]	[2]	[3]	[4]	[5]	[6]
16	13	9	10	11	2	3

③

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	13	9	10	11	2	16

④

[0]	[1]	[2]	[3]	[4]	[5]	[6]
13	11	9	10	2	3	16

정답 3번

출제 포인트

이 문제는 이중 해싱에서 기본 해시 위치와 재조사 간격을 순서대로 적용하여 최종 저장 위치를 구할 수 있는지를 묻는다.

조건 정리

해시 테이블 크기는 7이고 인덱스는 0부터 6까지이다.
삽입 순서는 9, 10, 2, 3, 16, 13, 11이다.
기본 해시는 H(x)=x 7, 추가 해시는 f(x)=5-(x 5), 충돌 시 재조사 위치는 h_i(x)=(H(x)+i× f(x)) 7이다.
버킷당 슬롯 수는 1이므로 충돌이 나면 다음 재조사 위치로 이동한다.

사용할 공식

기본 위치는 H(x)=x 7로 정하고, 충돌이 나면 간격 f(x)=5-(x 5)만큼 이동한다.
이 문제는 각 키를 주어진 순서대로 삽입하면서 충돌 횟수에 따라 h_i(x)=(H(x)+i× f(x)) 7를 적용하면 된다.

NoteBase

계산 과정

9는 $H(9) = 2$이므로 [2]에 저장된다.
10은 $H(10) = 3$이므로 [3]에 저장된다.
2는 $H(2) = 2$에서 충돌하고 $f(2) = 3$이므로 h_1(2)=(2+3) 7=5에서 [5]에 저장된다.
3은 $H(3) = 3$에서 충돌하고 $f(3) = 2$이므로 $h_1(3) = 5$도 충돌, h_2(3)=(3+2× 2) 7=0이어서 [0]에 저장된다.
16은 $H(16) = 2$에서 충돌하고 $f(16) = 4$이므로 h_1(16)=(2+4) 7=6이어서 [6]에 저장된다.
13은 $H(13) = 6$에서 충돌하고 $f(13) = 2$이므로 h_1(13)=(6+2) 7=1이어서 [1]에 저장된다.
11은 $H(11) = 4$이므로 [4]에 바로 저장된다.
따라서 최종 해시 테이블은 [0]=3, [1]=13, [2]=9, [3]=10, [4]=11, [5]=2, [6]=16이다.

선지별 해설

1번 X: [0]에 13이 들어간다고 본 배열이다. 실제로 13은 $H(13) = 6$에서 시작해 $h_1(13) = 1$로 이동하므로 [1]에 저장된다. 틀린 표현은 "[0]에 13"이고, 올바르게는 "[1]에 13"이다.
2번 X: [0]에 16이 들어간다고 본 배열이다. 실제로 16은 $H(16) = 2$, $f(16) = 4$이므로 첫 재조사 위치 $h_1(16) = 6$에서 [6]에 저장된다. 틀린 표현은 "[0]에 16"이고, 올바르게는 "[6]에 16"이다.
3번 O: 계산 결과와 정확히 일치한다. 최종 배열은 [0]=3, [1]=13, [2]=9, [3]=10, [4]=11, [5]=2, [6]=16이다.
4번 X: 여러 키의 위치가 뒤바뀐 배열이다. 11은 $H(11) = 4$이므로 [4]에 있어야 하고, 2는 충돌 후 [5], 3은 두 번째 재조사 후 [0]에 있어야 한다. 틀린 표현은 "[1]에 11, [4]에 2, [5]에 3"이고, 올바르게는 "[1]에 13, [4]에 11, [5]에 2, [0]에 3"이다.

오답 포인트

충돌이 난 키만 따로 떼어 보지 않고, 삽입 순서 전체를 끝까지 따라가야 한다는 점이 함정이다. 특히 3은 한 번 더 충돌해서 [0]으로 가고, 13은 6에서 시작하지만 충돌 후 [1]로 이동한다는 점을 놓치기 쉽다.

다시 나오면 이렇게 판단

이중 해싱 문제는 각 키마다 기본 위치를 먼저 쓰고, 충돌이 날 때마다 추가 해시값으로 이동한 위치를 순서대로 기록하면 된다. 이전 키가 이미 차지한 칸까지 반영하면서 한 칸씩 확정하면 배열을 안정적으로 복원할 수 있다.

암기 포인트

이중 해싱은 기본 위치 $H(x)$를 찍고, 충돌하면 간격 $f(x)$만큼 재조사한다는 구조이다.

NoteBase

문 11. 다음 일련의 스택 연산에 의해 스택에서 삭제되는 자료를 순서대로 바르게 나열한 것은? (단, push(x)는 스택에 자료 x를 삽입하는 연산이고, pop()은 스택에서 자료 한 개를 삭제하는 연산이다)

push(9), pop(), push(5), push(7), push(2), pop(), push(9), pop(), push(4), pop(), pop(), push(6), push(3), pop(), push(2), pop(), pop()

① 9, 5, 7, 2, 9, 4, 6, 3

② 9, 2, 9, 4, 7, 3, 2, 6

③ 2, 3, 6, 4, 9, 2, 7, 5

④ 9, 5, 9, 4, 7, 6, 2, 2

정답 2번

출제 포인트

스택의 후입선출 구조에서 각 pop()이 그 시점에 가장 나중에 들어간 값을 제거한다는 점을 순서대로 추적하는 문제이다.

올바른 순서

1. push(9) 다음 pop()에서 9가 삭제된다.
2. push(5), push(7), push(2) 다음 pop()에서 가장 위의 2가 삭제된다.
3. 이어서 push(9) 다음 pop()에서 9가 삭제되고, push(4) 다음 pop()에서 4가 삭제된다.
4. 그다음 pop()에서 남아 있던 7이 삭제된다.
5. push(6), push(3) 다음 pop()에서 3이 삭제된다.
6. push(2) 다음 pop()에서 2가 삭제되고, 마지막 pop()에서 6이 삭제된다.

왜 이 순서인가: 스택은 마지막에 넣은 값이 먼저 나오는 LIFO 구조이므로 각 pop()은 그 순간의 top 값을 제거한다.

NoteBase

선지별 해설

① X: 첫 번째 삭제값 9는 맞지만 두 번째 삭제값이 5일 수 없다. push(5), push(7), push(2) 뒤의 top은 2이므로 두 번째 값은 2이다. 틀린 표현은 초반의 "9, 5"이고, 올바르게는 "9, 2"이다.
② O: 실제 삭제 순서는 9, 2, 9, 4, 7, 3, 2, 6이다. 각 pop()마다 가장 최근에 push된 값부터 빠지므로 연산열과 정확히 일치한다.
③ X: 첫 삭제값이 2일 수 없다. 첫 연산은 push(9) 다음 바로 pop()이므로 처음 빠지는 값은 9이다. 틀린 표현은 맨 앞의 "2"이고, 올바르게는 "9"로 시작한다.
④ X: 두 번째 삭제값을 5로 본 것이 틀렸고 마지막 삭제값을 2로 본 것도 틀렸다. push(5), push(7), push(2) 뒤에는 2가 먼저 삭제되며, 마지막에는 push(6), push(3), push(2) 이후 남아 있던 6이 삭제된다. 틀린 표현은 두 번째 값 5와 마지막 값 2이고, 올바르게는 두 번째 값은 2, 마지막 값은 6이다.

오답 포인트

push된 수를 입력 순서대로 그대로 읽거나, pop()이 한 번 실행된 뒤 스택 상태가 바뀐다는 점을 놓치면 오답이 된다. 특히 5는 초기에 들어가 스택 바닥에 남으므로 이 연산열에서는 삭제되지 않는다는 점이 함정이다.

다시 나오면 이렇게 판단

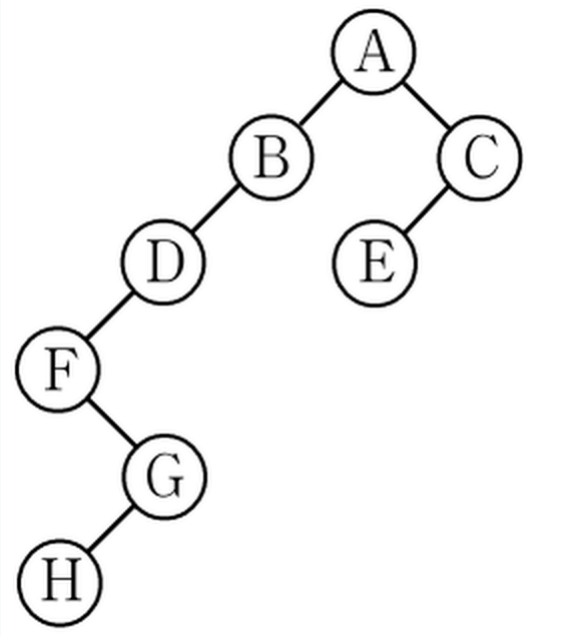
절차형 스택 문제는 연산을 왼쪽부터 보면서 push는 top에 쌓고 pop이 나오면 그 시점의 top만 기록한 뒤 제거하는 방식으로 판단한다. 각 pop() 뒤에 남은 스택 상태를 바로 갱신하면 순서를 헷갈리지 않는다.

암기 포인트

스택은 후입선출 구조이므로 가장 나중에 넣은 값이 가장 먼저 나온다.

NoteBase

문 12. 다음 이진 트리(binary tree)를 <보기>에서 제시한 방법을 사용하여 1차원 배열로 표현할 때, 필요한 배열의 최소 크기는? (단, 배열의 인덱스(index)는 0부터 시작하며 배열의 크기는 배열 원소의 개수이다)



<보기>

◯ 루트 노드는 배열의 인덱스가 1인 위치에 저장된다.

◯ 배열 인덱스가 k인 위치에 있는 노드가 자식 노드를 가질 경우, 왼쪽 자식 노드는 인덱스가 $2k$인 위치에 있으며 오른쪽 자식 노드는 인덱스가 $2k + 1$인 위치에 있다.

① 17

② 18

③ 34

④ 35

정답 4번

출제 포인트

이 문제는 이진 트리를 배열로 표현할 때 루트가 인덱스 1에 저장되고 왼쪽 자식은 2k, 오른쪽 자식은 2k+1이 된다는 규칙으로 각 노드의 인덱스를 끝까지 따라가며, 배열 크기는 최댓값 인덱스가 아니라 최댓값 인덱스+1임을 묻는 문제이다.

NoteBase

선지별 해설

① X: 17은 노드 G의 인덱스이다. A=1, B=2, D=4, F=8, G=17까지는 맞지만 G의 왼쪽 자식 H는 34가 된다. 틀린 표현은 17을 필요한 배열 크기로 본 점이며, 올바르게는 H까지 저장하려면 34번 인덱스까지 필요하므로 배열 크기는 35이다.
② X: 18은 중간 계산을 잘못 끊은 값이다. 최대 인덱스는 18이 아니라 H의 34이다. 틀린 표현은 자식 인덱스를 끝까지 따라가지 않은 점이며, 올바르게는 최대 인덱스를 34로 계산한 뒤 배열 크기를 35로 잡아야 한다.
③ X: 34는 마지막 노드 H가 저장되는 인덱스이다. 그러나 문제는 마지막 사용 인덱스가 아니라 배열의 크기를 묻는다. 틀린 표현은 최대 인덱스와 배열 크기를 같게 본 점이며, 올바르게는 인덱스가 0부터 시작하므로 0~34를 담는 배열 크기는 35이다.
④ O: A=1, B=2, C=3, D=4, E=6, F=8, G=17, H=34가 된다. 가장 큰 인덱스가 34이고 배열 인덱스는 0부터 시작하므로 필요한 최소 배열 크기는 35가 된다.

오답 포인트

이 문항의 함정은 마지막 노드의 인덱스와 배열 크기를 혼동하게 만드는 점이다. 특히 ③처럼 34를 보면 끝값처럼 보이지만, 배열 크기는 원소 개수이므로 최대 인덱스에 1을 더해야 한다.

다시 나오면 이렇게 판단

루트에서 가장 깊은 노드까지 부모-자식 인덱스 규칙을 순서대로 적용해 최대 인덱스를 먼저 구한 뒤, 배열 크기를 묻는 경우에는 그 값에 1을 더해 판단한다.

암기 포인트

배열 트리 문제는 최대 인덱스와 배열 크기를 구분하며, 배열 크기=최대 인덱스+1이다.

NoteBase

문 13. 행 우선(row major) 순서로 저장되는 3차원 배열 a[3][30][10]이 있을 때, 첫 번째 원소인 a[0][0][0]의 메모리 주소가 2048이면 a[2][15][2]의 메모리 주소는? (단, 배열 a에서 각 원소의 크기는 4바이트이다)

① 2800

② 2804

③ 5052

④ 5056

정답 4번

출제 포인트

이 문제는 3차원 배열의 row major 저장 원리를 이용해 특정 원소의 메모리 주소를 계산하는 문제이다.

조건 정리

- 배열 크기는 3 × 30 × 10이다.
- 시작 주소는 a[0][0][0] = 2048이다.
- 구할 원소는 a[2][15][2]이다.
- 각 원소의 크기는 4바이트이다.
- row major이므로 마지막 첨자부터 연속 저장된다.

사용할 공식

3차원 배열 $a[i][j][k]$의 row major 주소는 $\mathrm{LOC}(a[i][j][k]) = \mathrm{base} + ((i \times 30 \times 10) + (j \times 10) + k) \times 4$이다.
이 공식을 쓰는 이유는 앞 차원의 전체 블록 수와 다음 차원의 묶음 수를 먼저 건너뛴 뒤 마지막 첨자를 더해 선형 위치를 만든 다음, 원소 크기 4바이트를 곱해야 실제 바이트 주소가 되기 때문이다.

계산 과정

1. 선형 인덱스를 구하면 $(2 \times 30 \times 10) + (15 \times 10) + 2 = 600 + 150 + 2 = 752$이다.
2. 바이트 오프셋은 $752 \times 4 = 3008$이다.
3. 실제 주소는 $2048 + 3008 = 5056$이다.

NoteBase

선지별 해설

① X: 2800은 선형 인덱스 752를 바이트 변환하지 않고 시작 주소에 그대로 더한 값이다. 틀린 표현은 오프셋을 원소 개수 그대로 더한 점이며, 올바르게는 752 × 4 = 3008바이트를 더해 5056이 된다.
② X: 2804는 원소 크기 4바이트를 전체 오프셋에 곱하지 않고 일부만 반영한 값이다. 틀린 표현은 전체 선형 인덱스에 4를 곱하지 않은 점이며, 올바르게는 752 전체에 4를 곱해 5056이 된다.
③ X: 5052는 정답보다 4바이트 작아 원소 하나를 덜 센 값이다. 틀린 표현은 a[2][15][2]를 바로 앞 원소처럼 계산한 점이며, 올바르게는 마지막 첨자 2까지 포함해 5056이 된다.
④ O: row major 기준 선형 인덱스가 752이고, 바이트 오프셋이 3008이므로 주소는 2048 + 3008 = 5056이다.

오답 포인트

배열 주소 문제에서는 선형 인덱스와 바이트 오프셋을 혼동하기 쉽다. 특히 row major로 원소 번호 752를 구한 뒤 원소 크기 4바이트를 곱하지 않으면 ① 같은 오답이 나오고, 마지막 첨자를 하나 덜 세면 ③ 같은 오프바이원 오답이 나온다.

다시 나오면 이렇게 판단

다차원 배열 주소 문제는 먼저 저장 방식이 row major인지 확인하고, 첨자를 1차원 선형 인덱스로 바꾼 다음, 마지막에 원소 크기를 곱해 바이트 주소로 변환하는 순서로 판단한다.

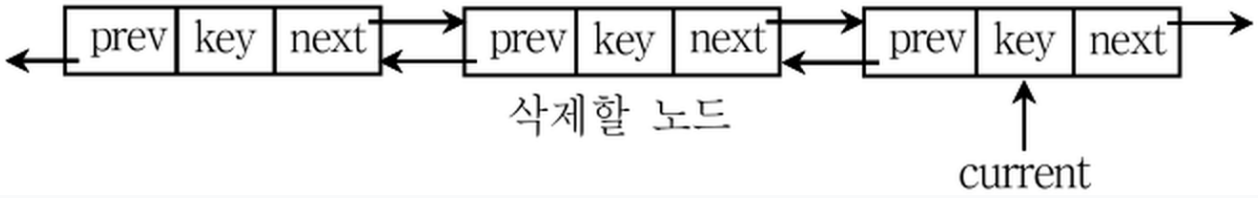
암기 포인트

row major 주소는 앞차원 누적 개수로 선형 인덱스를 만든 뒤 원소 크기를 곱해 구한다.

NoteBase

문 14. 다음은 C 언어 구조체와 이를 이용한 이중 연결 리스트(doubly linked list)이다. 포인터 current가 가리키는 노드의 왼쪽 노드를 삭제할 때 수행할 C 언어 문장을 순서대로 바르게 나열한 것은? (단, 삭제 전에 포인터 current가 가리키는 노드의 왼쪽에 최소 2개의 노드가 있다고 가정하며, 삭제 노드에 대한 메모리 할당 해제는 생략한다)

```
struct Dnode {
  int key;
  struct Dnode *prev;
  struct Dnode *next;
};
```



①
```
current->next = current->next->next;
current->next->prev = current;
```

②
```
current->next->prev = current;
current->next = current->next->next;
```

③
```
current->prev = current->prev->prev;
current->prev->next = current;
```

④
```
current->prev->next = current;
current->prev = current->prev->prev;
```

정답 3번

출제 포인트

이 문제는 이중 연결 리스트에서 current의 왼쪽 노드를 삭제할 때 prev 링크를 어느 순서로 갱신해야 하는지를 묻는 문제이다.

올바른 순서

1. current->prev를 삭제 노드의 이전 노드로 옮긴다: current->prev = current->prev->prev;
2. 새로 연결된 왼쪽 노드의 next를 current로 맞춘다: current->prev->next = current;

왜 이 순서: current->prev를 먼저 한 칸 더 왼쪽으로 옮겨야 두 번째 문장에서 삭제 노드가 아니라 삭제 노드의 이전 노드를 가리키게 된다.

NoteBase

선지별 해설

① X
- current->next와 current->next->prev를 바꾸므로 current의 오른쪽 노드를 삭제하는 코드에 해당한다.
- 틀린 표현은 next 방향 갱신이다. 올바르게는 current의 왼쪽 노드를 삭제하므로 current->prev와 current->prev->next를 갱신해야 한다.

② X
- ①과 같은 방향의 링크를 순서만 바꿔 수정한 코드이므로 여전히 오른쪽 노드를 대상으로 한다.
- 틀린 표현은 current->next와 current->next->prev이다. 올바르게는 current->prev = current->prev->prev; 후 current->prev->next = current; 순서가 된다.

③ O
- current->prev = current->prev->prev;로 삭제 노드를 건너뛰어 current의 이전 노드를 한 칸 더 왼쪽 노드로 바꾼다.
- 이어서 current->prev->next = current;로 새 이전 노드의 next를 current에 연결하므로 삭제 노드가 리스트에서 빠진다.

④ X
- 첫 문장 current->prev->next = current;를 실행할 때 current->prev는 아직 삭제할 노드를 가리키므로 삭제 노드의 next만 다시 current로 맞추는 효과에 그친다.
- 틀린 표현은 첫 단계의 연결 대상이다. 올바르게는 먼저 current->prev를 current->prev->prev로 바꾼 뒤 새 이전 노드의 next를 current에 연결해야 한다.

오답 포인트

왼쪽 노드 삭제라는 말만 보고 ④처럼 next 연결을 먼저 떠올리기 쉽다. 그러나 포인터가 아직 삭제 노드를 가리키는 상태인지, 삭제 노드의 이전 노드를 가리키는 상태인지를 구분하지 못하면 선후관계를 거꾸로 잡게 된다.

다시 나오면 이렇게 판단

이중 연결 리스트에서 인접 노드를 삭제할 때는 삭제 대상 자체가 아니라 삭제 대상의 양옆 노드를 서로 직접 연결해야 한다. current가 삭제할 노드의 오른쪽에 있으면 먼저 current->prev를 삭제 대상의 이전 노드로 넘긴 뒤, 그 새 prev의 next를 current로 연결하는 순서로 판단한다.

암기 포인트

왼쪽 삭제는 prev를 먼저 건너뛰고, 그다음 새 prev의 next를 current에 붙이는 구조이다.

NoteBase

문 15. 다음 C 언어 프로그램의 출력 결과는?

```
#include <stdio.h>

int i;

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

int main()
{
  int result;
  i = 0;
  result = RecFunc(5);
  printf("%d, %d\n", i, result);
}
```

① 13, 5

② 13, 8

③ 15, 5

④ 15, 8

정답 4번

출제 포인트

이 문제는 재귀 함수의 반환값과 전체 호출 횟수를 각각 추적해 최종 출력값을 구하는 문제이다.

조건 정리

주어진 값: 전역 변수 i는 함수가 호출될 때마다 1씩 증가한다.
구해야 할 값: RecFunc(5)를 호출한 뒤의 i와 result 값이다.
주의할 조건: n<=1이면 1을 반환하고, 그보다 크면 RecFunc(n-1)+RecFunc(n-2)를 반환한다.

NoteBase

사용할 공식

반환값은 $R(n) = R(n-1) + R(n-2), R(0) = R(1) = 1$로 본다.
호출 횟수는 함수에 진입할 때마다 i가 증가하므로 $T(n) = 1 + T(n-1) + T(n-2), T(0) = T(1) = 1$로 본다.
이 공식을 쓰는 이유는 이 함수가 피보나치 형태로 두 번 재귀 호출되고, 호출이 시작되는 순간마다 i++가 먼저 실행되기 때문이다.

계산 과정

1. 반환값을 구하면 $R(2) = 2, R(3) = 3, R(4) = 5, R(5) = 8$이다.
2. 호출 횟수를 구하면 $T(2) = 3, T(3) = 5, T(4) = 9, T(5) = 15$이다.
3. 따라서 printf에서 출력되는 값은 i=15, result=8이다.

선지별 해설

① X: 13, 5는 둘 다 틀리다. 5는 RecFunc(4)의 반환값에 해당하고, 호출 횟수도 전체 진입 횟수를 다 세면 15가 된다. 올바르게는 15, 8이다.
② X: 13, 8에서 8은 맞지만 13이 틀리다. 이 함수는 호출될 때마다 i++가 실행되므로 RecFunc(5) 전체 호출 수는 15회이다. 올바르게는 15, 8이다.
③ X: 15, 5에서 15는 맞지만 5가 틀리다. 반환값은 피보나치형 재귀를 그대로 계산한 RecFunc(5)=8이다. 올바르게는 15, 8이다.
④ O: 호출 횟수는 15회이고 반환값은 8이므로 15, 8이 맞다.

오답 포인트

호출 횟수와 반환값을 같은 재귀식으로 착각하면 틀리기 쉽다. 반환값은 $R(n) = R(n-1) + R(n-2)$이지만, 호출 횟수는 현재 호출 1회를 포함해야 하므로 $T(n) = 1 + T(n-1) + T(n-2)$라는 점이 함정이다.

다시 나오면 이렇게 판단

재귀 출력 문제는 반환값 재귀식과 호출 횟수 재귀식을 분리해서 본다. 전역 변수 증가 시점이 함수 진입 전반에 걸쳐 적용되는지 먼저 확인하면 호출 횟수 오답을 줄일 수 있다.

암기 포인트

피보나치형 재귀에서 반환값과 호출 횟수는 다르다. i++가 함수 시작에 있으면 호출 횟수 식에 항상 현재 호출 1회가 추가된다.

NoteBase

문 16. 다음은 키(key)의 오름차순으로 정렬된 배열을 보여준다. 키 24에 대한 보간 탐색(interpolation search)을 수행할 때, 첫 번째 탐색 위치는? (단, 배열에서 크기가 가장 작은 키와 가장 큰 키의 값은 미리 주어지고, 위치의 차이는 키 값의 차이에 비례한다는 가정 하에 탐색 위치를 계산하며, 소수점 이하는 반올림한다)

위치	0	1	2	3	4	5	6	7	8	9	10	11	12	13
키	3	10	15	19	24	30	40	65	70	74	80	82	87	93

① 2

② 3

③ 4

④ 5

정답 2번

출제 포인트

이 문제는 보간 탐색의 첫 탐색 위치를 비례식으로 계산하고 소수점 이하를 반올림하는지를 묻는 문제이다.

조건 정리

주어진 값: 가장 작은 키는 3, 가장 큰 키는 93, 인덱스 범위는 0부터 13까지이다.
구해야 할 값: 키 24를 찾을 때의 첫 번째 탐색 위치이다.
주의할 조건: 위치 차이는 키 값의 차이에 비례한다고 가정하고, 계산 결과의 소수점 이하는 반올림한다.

사용할 공식

보간 탐색의 첫 위치는 $pos = low + \frac{(key - a[low])(high - low)}{a[high] - a[low]}$ 로 계산한다.
이 공식을 쓰는 이유는 배열의 양 끝 키 값과 목표 키의 상대적 비율을 이용해, 목표 키가 전체 인덱스 구간의 어느 지점에 있을지를 추정하는 탐색이기 때문이다.

계산 과정

1. $low = 0$, $high = 13$, $a[low] = 3$, $a[high] = 93$, $key = 24$를 대입한다.
2. $pos = 0 + \frac{(24-3)(13-0)}{93-3} = \frac{21 \times 13}{90} = \frac{273}{90} \approx 3.033$ 이다.
3. 소수점 이하는 반올림하므로 첫 번째 탐색 위치는 3이다.

NoteBase

선지별 해설

① X: 2는 비율 계산 결과 3.033을 반올림하지 않거나 구간 비율을 작게 잡았을 때 나오는 오답이다. 올바른 첫 탐색 위치는 3이다.
② O: 보간 탐색 공식에 값을 대입하면 3.033이 나오고, 이를 반올림하면 3이 된다. 따라서 2번이 맞다.
③ X: 4는 배열에서 키 24가 실제로 저장된 위치를 답한 값이다. 문제는 실제 저장 위치가 아니라 첫 번째로 추정하는 탐색 위치를 묻고 있으므로 올바르게는 3이다.
④ X: 5는 목표 키가 구간의 더 뒤쪽에 있다고 과대 추정한 값이다. 키 24는 전체 키 범위 3~93에서 앞부분에 있으므로 비례식 결과도 3 근처가 되어야 하며, 올바르게는 3이다.

오답 포인트

이 문제의 함정은 실제 저장 위치 4와 첫 추정 위치 3을 혼동하는 데 있다. 또한 (24 − 3)/(93 − 3)로 구한 비율을 인덱스 구간 13에 곱한 뒤 반올림해야 하는데, 반올림을 빼먹으면 2나 3 사이에서 헷갈릴 수 있다.

다시 나오면 이렇게 판단

보간 탐색 문제는 먼저 양 끝 인덱스와 양 끝 키 값을 정리하고, 목표 키의 상대적 비율을 구한 뒤, 그 비율에 인덱스 구간 길이를 곱해 첫 위치를 계산한다. 마지막에는 문제 조건에 따라 반올림인지 버림인지 확인한다.

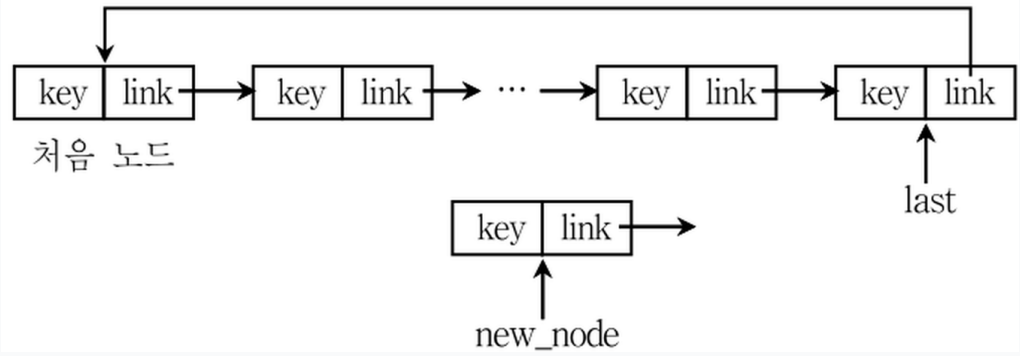
암기 포인트

보간 탐색의 첫 위치는 값의 비율로 잡는 위치이며, 계산 순서는 값 차이 비율 → 인덱스 구간에 곱셈 → 반올림이다.

NoteBase

문 17. 다음은 C 언어 구조체와 이를 이용한 원형 연결 리스트(circular linked list)이다. 원형 연결 리스트의 맨 마지막에 포인터 new_node가 가리키는 노드를 삽입할 때 수행할 C 언어 문장을 순서대로 바르게 나열한 것은? (단, new_node의 삽입 전에 원형 연결 리스트는 공백 리스트가 아니고, last는 원형 연결 리스트의 마지막 노드를 가리키는 포인터다)

```
struct Cnode {
  int key;
  struct Cnode *link;
};
```



①
```
new_node->link = last->link;
last = new_node;
last->link = new_node;
```

②
```
new_node->link = last->link;
last->link = new_node;
last = new_node;
```

③
```
last->link = new_node;
new_node->link = last->link;
last = new_node;
```

④
```
last = new_node;
last->link = new_node;
new_node->link = last->link;
```

정답 2번

출제 포인트

이 문제는 공백이 아닌 원형 연결 리스트의 맨 뒤에 새 노드를 삽입할 때, 첫 노드 포인터를 보존하면서 마지막 노드와 새 노드의 링크를 어떤 순서로 갱신해야 하는지를 묻는 문제이다.

NoteBase

올바른 순서

1. new_node->link = last->link; - 새 노드가 기존 첫 노드를 가리키게 하여 원형 구조의 시작점을 먼저 보존한다.
2. last->link = new_node; - 기존 마지막 노드가 새 노드를 가리키게 연결한다.
3. last = new_node; - 마지막 노드 포인터를 새 노드로 갱신한다.

왜 이 순서인가: last를 바꾸기 전에 기존 첫 노드인 last->link를 먼저 저장해야 하고, 기존 마지막 노드의 link를 새 노드로 연결한 뒤에야 last를 새 노드로 옮겨도 원형 구조가 유지된다.

선지별 해설

① X: 첫 줄까지는 맞지만 두 번째 줄에서 last = new_node;를 먼저 수행하면 마지막 줄의 last->link = new_node;는 새 노드 자신의 link를 자기 자신으로 바꾸게 된다. 틀린 표현은 갱신 순서가 앞선 점이다. 올바르게는 new_node->link = last->link; 후에 last->link = new_node;를 수행하고 마지막에 last = new_node;로 갱신한다.

② O: 새 노드가 먼저 기존 첫 노드인 last->link를 가리키고, 그 다음 기존 마지막 노드가 새 노드를 가리키며, 마지막에 last를 새 노드로 바꾸므로 원형 연결 리스트의 연결이 끊기지 않는다.

③ X: 첫 줄 last->link = new_node;를 먼저 수행하면 기존 마지막 노드의 link가 이미 새 노드를 가리키게 된다. 이어서 new_node->link = last->link;를 실행하면 새 노드의 link가 기존 첫 노드가 아니라 자기 자신을 가리키게 된다. 틀린 표현은 첫 두 단계의 선후관계이다. 올바르게는 먼저 new_node->link = last->link;로 기존 첫 노드를 저장해야 한다.

④ X: last = new_node;를 맨 먼저 수행하면 기존 마지막 노드에 접근하는 기준점이 사라지고, 이후 last->link = new_node;와 new_node->link = last->link;는 모두 새 노드의 자기 참조만 만들게 된다. 틀린 표현은 last 포인터를 가장 먼저 갱신한 점이다. 올바르게는 기존 마지막 노드와 기존 첫 노드의 연결을 모두 처리한 뒤 마지막에 last를 갱신한다.

오답 포인트

함정은 last가 마지막 노드를 가리킨다는 사실만 보고 last를 먼저 바꾸는 것이다. 원형 연결 리스트에서는 last->link가 기존 첫 노드이므로, 이 값을 보존하기 전에 last 자체나 last->link를 성급히 바꾸면 첫 노드 정보가 사라지거나 새 노드의 자기 순환이 생긴다.

다시 나오면 이렇게 판단

원형 연결 리스트의 끝 삽입은 기존 첫 노드 보존 -> 기존 마지막 노드와 새 노드 연결 -> 마지막 포인터 갱신의 순서로 판단한다. 즉, 새 노드가 누구를 가리킬지 먼저 정하고, 기존 노드의 link를 바꾼 뒤, 포인터 변수 last는 가장 나중에 바꾼다.

NoteBase

암기 포인트

원형 연결 리스트 끝 삽입은 첫 노드 보존이 먼저이고 last 갱신이 마지막이다.

NoteBase

문 18. 다음 5×5 희소 행렬(sparse matrix)을 구조체 배열로 표현하기 위해 <규칙>의 표현 규칙을 사용하여 <코드>의 C 언어 코드를 작성하였다. <코드>의 ㉠에 들어갈 내용은? (단, 행렬의 행 번호와 열 번호는 0부터 시작한다)

$$\begin{bmatrix} 0 & 5 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

<코 드>

```
typedef struct {
  int row;
  int col;
  int val;
} term;

term A[ ] = _____ ㉠ _____;
```

<규칙>

◯ 첫 번째 구조체인 A[0]의 row, col, val에는 희소 행렬의 행 개수, 열 개수, 값이 0이 아닌 원소 개수를 순서대로 저장한다.

◯ 두 번째부터의 나머지 구조체는 값이 0이 아닌 희소 행렬 원소를 표현하기 위해 사용되며, 각 구조체의 row, col, val에 해당 원소의 행 번호, 열 번호, 값을 순서대로 저장한다.

◯ 값이 0이 아닌 희소 행렬 원소 정보는 행 번호의 오름차순으로 구조체 배열 A에 저장되며, 행 번호가 같은 경우에는 열 번호의 오름차순으로 저장된다.

① {{4, 4, 4}, {0, 1, 5}, {0, 2, 2}, {1, 3, 1}, {3, 0, 3}}

② {{4, 4, 4}, {3, 0, 3}, {0, 1, 5}, {0, 2, 2}, {1, 3, 1}}

③ {{5, 5, 4}, {0, 1, 5}, {0, 2, 2}, {1, 3, 1}, {3, 0, 3}}

④ {{5, 5, 4}, {3, 0, 3}, {0, 1, 5}, {0, 2, 2}, {1, 3, 1}}

NoteBase

정답 3번

출제 포인트

희소 행렬의 3원소 구조체 배열 표현에서 A[0]에 저장하는 메타정보와, 0이 아닌 원소를 행 우선으로 나열하는 순서를 정확히 적용하는지를 묻는 문제이다.

핵심 개념

희소 행렬의 3원소 표현은 각 항목을 {row, col, val}로 저장하는 방식이다.
A[0]은 실제 원소가 아니라 {행 개수, 열 개수, 0이 아닌 원소 개수}를 저장한다.
그 뒤의 항목들은 값이 0이 아닌 원소만 저장하며, 행 번호 오름차순으로 정렬하고 같은 행에서는 열 번호 오름차순으로 정렬한다.

선지별 해설

① X
- A[0]의 {4, 4, 4}가 틀렸다. 주어진 행렬은 5×5이므로 행 개수와 열 개수는 모두 5이다.
- 0이 아닌 원소 개수 4는 맞지만, 올바르게는 {{5, 5, 4}, {0, 1, 5}, {0, 2, 2}, {1, 3, 1}, {3, 0, 3}}가 된다.

② X
- 첫 구조체 {4, 4, 4}가 틀린 데다, 0이 아닌 원소의 저장 순서도 틀렸다.
- {3, 0, 3}을 맨 앞에 두면 행 번호 오름차순 규칙을 어기게 된다. 올바르게는 0행의 원소들부터 {0, 1, 5}, {0, 2, 2}, {1, 3, 1}, {3, 0, 3} 순으로 저장한다.

③ O
- 행렬의 크기는 5행 5열이므로 A[0]은 {5, 5, 4}이다.
- 0이 아닌 원소는 (0,1)=5, (0,2)=2, (1,3)=1, (3,0)=3의 네 개이고, 행 우선·열 우선 순서로 나열하면 {{5, 5, 4}, {0, 1, 5}, {0, 2, 2}, {1, 3, 1}, {3, 0, 3}}가 된다.

④ X
- A[0]의 {5, 5, 4}는 맞지만, 그 뒤의 원소 나열 순서가 틀렸다.
- {3, 0, 3}은 3행 원소이므로 0행과 1행 원소보다 뒤에 와야 한다. 올바르게는 {{5, 5, 4}, {0, 1, 5}, {0, 2, 2}, {1, 3, 1}, {3, 0, 3}}이다.

NoteBase

오답 포인트

A[0]을 0이 아닌 원소의 개수만 담는 칸으로 보고 행·열 크기를 4로 잘못 적거나, 실제 값 3이 먼저 눈에 띄어 {3, 0, 3}을 앞에 두면 틀리기 쉽다. 이 문제의 함정은 행렬의 크기 정보와 저장 순서를 동시에 확인하지 않으면 ④처럼 부분적으로만 맞는 선지에 끌린다는 점이다.

다시 나오면 이렇게 판단

먼저 행렬의 전체 크기와 0이 아닌 원소 개수를 세어 A[0]을 확정한다. 그다음 0이 아닌 원소의 좌표를 모두 적고, 행 번호 오름차순, 같은 행이면 열 번호 오름차순으로 다시 배열하면 정답을 안정적으로 고를 수 있다.

암기 포인트

희소 행렬 3원소 표현은 A[0]=크기+개수, A[1]부터는 행 우선·열 우선 순서이다.

NoteBase

문 19. 정점이 4개인 무방향(undirected) 완전 그래프(complete graph)에서 만들어질 수 있는 신장 트리(spanning tree)의 총 개수는?

① 12

② 14

③ 16

④ 18

정답 3번

출제 포인트

이 문제는 완전그래프 K_n의 신장 트리 수를 Cayley 공식 n^{n-2}로 구할 수 있는지를 묻는다.

조건 정리

정점 수는 4개이다.
그래프는 무방향 완전그래프이므로 모든 서로 다른 두 정점이 연결되어 있다.
구해야 할 값은 이 그래프에서 만들 수 있는 서로 다른 신장 트리의 총 개수이다.

사용할 공식

완전그래프 K_n의 신장 트리 수는 n^{n-2}이다.
이 문제는 정점이 4개인 완전그래프 K_4이므로 모든 라벨이 붙은 신장 트리의 개수를 세는 Cayley 공식을 그대로 적용할 수 있다.

계산 과정

$4^{4-2} = 4^2 = 16$
따라서 K_4에서 만들 수 있는 신장 트리는 16개이다.

선지 판단

1번 12: X. 12라는 값은 K_4의 경로형 신장 트리만 세었을 때 나올 수 있는 수이고, 중심 정점 하나에 나머지 세 정점이 붙는 별형 신장 트리 4개가 빠져 있다. 올바른 값은 16이다.
2번 14: X. Cayley 공식으로 계산한 총 개수는 16이므로 14는 맞지 않는다. 틀린 값 14 대신 올바른 값은 16이다.
3번 16: O. $4^{4-2} = 16$이므로 완전그래프 K_4의 신장 트리 수와 일치한다.
4번 18: X. 신장 트리 수는 정확히 16개이므로 18은 과대계산한 값이다. 틀린 값 18 대신 올바른 값은 16이다.

NoteBase

오답 포인트

완전그래프의 간선 수 $\binom{4}{2} = 6$을 보고 간선 3개를 고르는 조합으로 바로 세면 사이클이 섞여 과대계산하기 쉽다. 반대로 경로형 트리만 떠올리면 12로 과소계산하게 된다.

다시 나오면 이렇게 판단

정점이 n개인 완전그래프의 신장 트리 수를 묻으면 먼저 Cayley 공식 n^{n-2} 적용 가능 여부를 확인하고, 그다음 n값만 대입해 계산하면 된다.

암기 포인트

완전그래프의 신장 트리 수는 $K_n \rightarrow n^{n-2}$이다. K_4는 16개이다.

NoteBase

문 20. 일련의 키(key) 값 2, 1, 8, 9, 7, 3, 6을 가지는 7개의 데이터를 순서대로 삽입하여 레드-블랙 트리(red-black tree)를 구성하였다. 구성된 레드-블랙 트리를 후위 순회(postorder traversal)할 경우 방문 노드들의 키 값을 방문 순서대로 바르게 나열한 것은?

① 1, 3, 6, 2, 9, 8, 7

② 1, 3, 7, 6, 9, 8, 2

③ 1, 6, 3, 2, 9, 8, 7

④ 1, 6, 3, 7, 9, 8, 2

정답 2번

출제 포인트

레드-블랙 트리에서 삽입 후 재색칠과 회전을 반영한 최종 트리를 만든 다음, 그 트리를 후위 순회한 결과를 묻는 문제이다. 레드-블랙 트리는 (1) 루트는 흑색, (2) 적색 노드의 자식은 모두 흑색(적색이 연속될 수 없음), (3) 임의의 노드에서 리프까지의 모든 경로가 같은 수의 흑색 노드를 지난다는 속성을 지켜야 한다. 새 노드는 항상 적색으로 삽입되므로 부모도 적색이면 (2)번 속성이 깨져 보정이 강제된다. 보정은 삼촌이 적색이면 재색칠(Case 1), 삼촌이 흑색이면서 좌우로 꺾인 형태이면 회전 후 재색칠(Case 2·3)로 처리한다.

올바른 순서

1. 2, 1, 8 삽입 - 2가 루트(흑색)가 되고 1과 8이 각각 좌우 자식(적색)이 된다.
2. 9 삽입 - 9는 8의 오른쪽 자식(적색)으로 붙는데 부모 8이 적색이라 속성이 깨진다. 삼촌 1도 적색이므로(Case 1) 부모 8과 삼촌 1을 흑색, 조부모 2를 적색으로 바꾸고, 2는 루트이므로 다시 흑색으로 되돌린다. 결과는 2(흑) 아래 1(흑), 8(흑), 8의 오른쪽 9(적)이다.
3. 7, 3 삽입 - 7은 8의 왼쪽 자식(적색)으로 붙는데 부모 8이 흑색이라 문제가 없다. 이어 3은 7의 왼쪽 자식(적색)으로 붙는데 부모 7이 적색이라 속성이 깨진다. 이때 삼촌은 8의 오른쪽 자식 9이고 9도 적색이므로(Case 1) 부모 7과 삼촌 9를 흑색, 조부모 8을 적색으로 재색칠한다. 결과는 8(적) 아래 왼쪽 7(흑)[그 왼쪽 3(적)], 오른쪽 9(흑)이다.
4. 6 삽입 - 6은 3의 오른쪽 자식(적색)으로 들어가는데 부모 3이 적색이고, 이때 삼촌(7의 오른쪽 자식)은 없어 흑색으로 간주되므로 Case 2·3에 해당한다. 6이 3의 오른쪽, 3이 7의 왼쪽이라 좌우로 꺾인 형태(LR)이므로, 먼저 6의 부모 3을 기준으로 좌회전해 일직선 형태로 만든 뒤, 6의 새 부모가 된 조부모 7을 기준으로 우회전한다. 회전이 끝나면 6이 서브트리의 루트(흑색)가 되고 3(왼쪽)과 7(오른쪽)이 각각 6의 자식(적색)이 된다.

왜 이 순서: 최종 트리는 루트 2(흑), 왼쪽 자식 1(흑), 오른쪽 자식 8(적)이며 8의 왼쪽 서브트리 루트가 6(흑)[좌 3(적), 우 7(적)], 오른쪽 자식이 9(흑)이다. 후위 순회는 왼쪽-오른쪽-루트 순서이므로 왼쪽 1, 오른쪽 서브트리(3, 7, 6, 9, 8), 마지막 루트 2가 되어 1, 3, 7, 6, 9, 8, 2로 나열된다.

NoteBase

선지별 해설

① X
- 틀린 이유: 후위 순회에서는 루트 2가 반드시 마지막에 와야 하는데, 여기서는 마지막이 7이고 2가 네 번째(중간)에 있다. 또한 6은 3과 7의 부모이므로 자식인 7이 6보다 먼저 방문되어야 하는데 6이 7보다 앞에 놓여 순서가 어긋난다.
- 올바르게는: 1, 3, 7, 6, 9, 8, 2가 된다.

② O
- 맞는 이유: 최종 트리에서 루트 2의 왼쪽 서브트리는 1뿐이고, 오른쪽 서브트리(루트 8)는 왼쪽에 6(자식 3·7), 오른쪽에 9를 둔다. 후위 순회하면 왼쪽 1, 오른쪽 서브트리 3·7·6·9·8, 마지막에 루트 2가 되어 1, 3, 7, 6, 9, 8, 2가 된다.

③ X
- 틀린 이유: 6을 3보다 먼저 방문했는데, 후위 순회에서 부모는 자식보다 뒤에 와야 하므로 6은 자식 3·7을 모두 방문한 뒤에 와야 한다. 또한 루트 2가 마지막이 아니라 네 번째에 놓이고 마지막이 7로 잘못되어 있다.
- 올바르게는: 6이 아니라 3, 7을 먼저 방문한 뒤 6을 방문하고 전체 마지막은 2가 되어야 한다.

④ X
- 틀린 이유: 마지막이 2인 점은 옳으나, 8의 왼쪽 서브트리를 6, 3, 7 순으로 적었다. 6은 이 서브트리의 루트이므로 자식 3(왼쪽)과 7(오른쪽)을 모두 방문한 뒤에 나와야 한다.
- 올바르게는: 왼쪽 서브트리 순서는 6, 3, 7이 아니라 3, 7, 6이다.

오답 포인트

이 문항의 함정은 이진 탐색 트리 모양만 대충 그리거나, 6 삽입 때 발생하는 회전을 빼먹는 데 있다. 특히 3-7-6 구간에서 6이 가운데 키로 서브트리 루트가 된다는 점을 놓치면 3, 7, 6 순서가 아니라 6, 3, 7처럼 잘못 적기 쉽다.

다시 나오면 이렇게 판단

레드-블랙 트리 문제는 단순 삽입 모양이 아니라 보정까지 끝난 최종 트리를 먼저 확정해야 한다. 부모와 삼촌이 모두 red이면 재색칠, 부모 red와 삼촌 black에서 꺾인 형태가 나오면 회전으로 가운데 키가 서브트리 루트가 된다고 판단한 뒤 후위 순회 왼쪽-오른쪽-루트를 적용하면 된다.

암기 포인트

후위 순회는 왼쪽-오른쪽-루트이고, 레드-블랙 트리의 꺾인 삽입 보정에서는 가운데 키가 서브트리 루트가 된다.

NoteBase