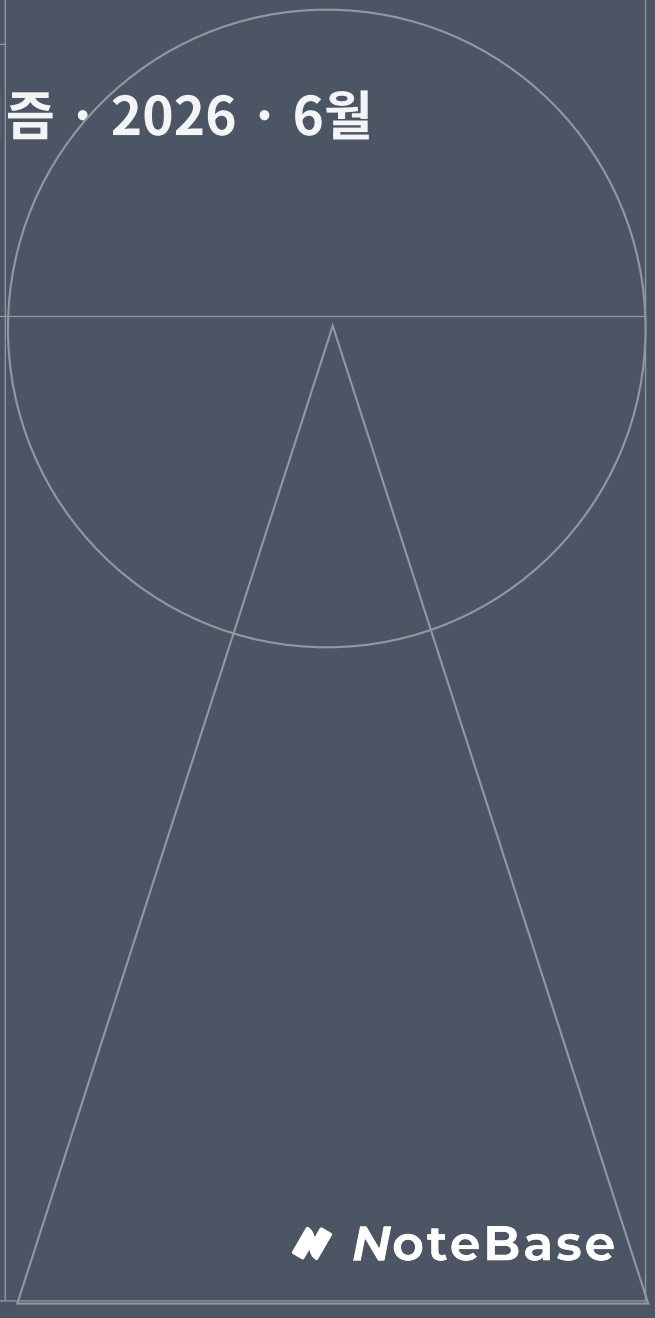
기출OX 해설집

공무원/서울시/9급 · 알고리즘 · 2026 · 6월



NoteBase

기출OX 해설집

공무원/서울시/9급 · 알고리즘 · 2026 · 6월

공무원/서울시/9급 · 알고리즘 · 2026 · 6월

1. 알고리즘의 시간복잡도 분석에 영향을 줄 수 있는 요인으로 옳은 것만을 <보기>에서 모두 고른 것은?

<보기>

ㄱ. 프로그래밍 언어
ㄴ. 주요 연산의 시행 횟수
ㄷ. 입력의 크기
ㄹ. 프로그래머의 역량

① ㄱ, ㄴ
② ㄴ, ㄷ
③ ㄱ, ㄷ, ㄹ
④ ㄴ, ㄷ, ㄹ

정답 2번

출제 포인트

알고리즘의 시간복잡도를 결정하는 요인과 실제 실행시간에만 영향을 줄 수 있는 요인을 구분하는 문제이다.

핵심 개념

시간복잡도는 입력의 크기가 커질 때 알고리즘의 주요 연산 시행 횟수가 어떻게 증가하는지를 나타내는 척도이다.
입력의 크기는 독립변수이고, 주요 연산의 시행 횟수는 이를 기준으로 세는 실행 비용이다. 반면 프로그래밍 언어는 실제 실행시간의 상수 차이를 만들 수 있지만 알고리즘 자체의 증가 차수를 바꾸는 요인은 아니다.
프로그래머의 역량도 동일한 알고리즘의 시간복잡도를 분석하는 직접 요인이 아니다.

NoteBase

선지별 해설

① X — ㄱ은 프로그래밍 언어이므로 알고리즘 자체의 시간복잡도를 결정하지 않는다. ㄴ은 주요 연산의 시행 횟수이므로 맞지만, 올바른 조합은 ㄴ, ㄷ이다.
② O — ㄴ의 주요 연산 시행 횟수는 시간 비용을 측정하는 기준이고, ㄷ의 입력 크기는 그 횟수의 증가 양상을 결정하는 기준이므로 두 항목 모두 맞다.
③ X — ㄷ은 맞지만 ㄱ과 ㄹ은 시간복잡도 분석의 직접 요인이 아니다. 프로그래밍 언어나 프로그래머의 역량이 아니라 입력 크기에 따른 주요 연산의 시행 횟수를 분석해야 하므로, 올바른 조합은 ㄴ, ㄷ이다.
④ X — ㄴ과 ㄷ은 맞지만 ㄹ의 프로그래머 역량은 동일한 알고리즘의 시간복잡도를 결정하는 직접 요인이 아니다. ㄹ을 제외한 ㄴ, ㄷ이 올바른 조합이다.

오답 포인트

프로그래밍 언어와 프로그래머의 역량이 프로그램의 실제 속도나 구현 품질에는 영향을 줄 수 있다는 점 때문에 ㄱ이나 ㄹ을 고르기 쉽다. 그러나 시간복잡도 분석은 실행 환경이나 개인 능력보다 입력 크기에 따라 주요 연산 횟수가 증가하는 양상을 다룬다.

다시 나오면 이렇게 판단

시간복잡도의 영향 요인을 물으면 먼저 입력 크기를 확인하고, 그 크기에 따라 반복·비교·대입 같은 주요 연산이 몇 번 수행되는지를 센다. 언어, 하드웨어, 프로그래머 역량처럼 실제 실행시간이나 구현 품질에 관련된 요소는 알고리즘 자체의 증가 차수와 구분한다.

암기 포인트

시간복잡도는 입력 크기와 주요 연산 횟수의 증가 관계이다.

NoteBase

2. 시간복잡도에 대한 점근적 표기법 중 가장 옳지 않은 것은?

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

② $f(n)=\Theta(g(n))$이면 $f(n)=\Omega(g(n))$이다.

③ $f(n)=O(g(n))$이면 $f(n)\neq\Omega(g(n))$이다.

④ $f(n)=O(g(n))$이고 $g(n)=O(h(n))$이면 $f(n)=O(h(n))$이다.

정답 3번

출제 포인트

이 문제는 점근적 상한 O, 하한 Ω, 동치적 차수 Θ의 포함 관계와 O의 추이성을 묻는 문제이다.

핵심 개념

$f(n) = O(g(n))$: 충분히 큰 n에서 $f(n)$이 상수배의 $g(n)$보다 빠르게 증가하지 않는다는 점근적 상한이다.
$f(n) = \Omega(g(n))$: 충분히 큰 n에서 $f(n)$이 상수배의 $g(n)$보다 작게 증가하지 않는다는 점근적 하한이다.
$f(n) = \Theta(g(n))$: $f(n)$이 $g(n)$과 같은 점근적 증가 차수를 가진다는 뜻이며, $O(g(n))$와 $\Omega(g(n))$를 동시에 만족한다.

선지별 해설

① O — $f(n) = \Theta(g(n))$이면 $f(n)$은 $g(n)$의 점근적 상한 조건을 만족하므로 $f(n) = O(g(n))$이다.
② O — $f(n) = \Theta(g(n))$이면 $f(n)$은 $g(n)$의 점근적 하한 조건도 만족하므로 $f(n) = \Omega(g(n))$이다.
③ X — 틀린 표현은 '$f(n) = O(g(n))$이면 $f(n) \neq \Omega(g(n))$'이다. O라는 사실만으로 Ω가 아님을 결론 낼 수 없다. 예를 들어 $f(n) = g(n)$이면 $f(n) = O(g(n))$이면서 동시에 $f(n) = \Omega(g(n))$이다. 올바르게는 '$f(n) = O(g(n))$만으로 $f(n)$이 $\Omega(g(n))$인지 여부를 판단할 수 없다'이다.
④ O — 점근적 상한은 추이적이다. $f(n) = O(g(n))$이고 $g(n) = O(h(n))$이면 두 상수배의 상한을 이어서 $f(n) = O(h(n))$을 얻는다.

오답 포인트

③은 O와 Ω를 서로 배타적인 표기로 오해하게 만든다. 두 표기는 각각 상한과 하한을 나타내므로 한 함수 관계가 둘을 동시에 만족할 수 있으며, 바로 그 경우가 Θ이다.

다시 나오면 이렇게 판단

먼저 $\Theta(g(n))$이면 $O(g(n))$와 $\Omega(g(n))$를 모두 만족한다는 포함 관계를 적용한다. 이어서 O 또는 Ω 하나만 주어졌을 때 반대쪽 관계를 부정할 수는 없으며, 같은 종류의 점근적 상한 관계는 연쇄하여 추이적으로 판단한다.

NoteBase

암기 포인트

$\Theta = O \cap \Omega$, 그리고 O는 추이적이다.

NoteBase

3. 분할 정복(divide-and-conquer) 기법에 대한 설명으로 가장 옳은 것은?

① 문제를 더 이상 나눌 수 없을 때까지 작은 문제로 분할한 후, 그 결과를 결합하여 해결한다.

② 문제를 해결할 때 항상 현재 단계에서 최적인 선택을 반복한다.

③ 이미 계산된 결과를 저장하여 동일한 계산을 반복하지 않도록 한다.

④ 모든 가능한 경우를 탐색하여 최적해를 찾는 방식이다.

정답 1번

출제 포인트

분할 정복의 핵심 절차인 분할, 정복, 결합을 다른 알고리즘 설계 기법과 구별하는 문제이다.

핵심 개념

분할 정복: 원래 문제를 더 작은 부분 문제로 나누고, 각 부분 문제를 해결한 뒤, 그 결과를 결합하여 전체 문제의 해를 구하는 기법이다.
분할 정복은 '나눈 뒤 각각 해결하고 결합'하고, 탐욕법은 '매 단계의 최선 선택', 동적 계획법은 '중복되는 부분 문제의 결과 저장·재사용', 완전 탐색은 '가능한 모든 경우 확인'을 핵심으로 한다.

선지별 해설

① O — 문제를 더 이상 나눌 수 없을 때까지 작은 문제로 분할하고, 각 부분 문제의 해를 구한 뒤 결합하여 전체 문제를 해결한다는 설명은 분할 정복의 구조와 일치한다.
② X — 틀린 표현은 '항상 현재 단계에서 최적인 선택을 반복한다'이다. 이는 탐욕법에 대한 설명이며, 올바르게는 문제를 부분 문제로 분할하여 각각 해결한 뒤 그 결과를 결합한다고 해야 한다.
③ X — 틀린 표현은 '이미 계산된 결과를 저장하여 동일한 계산을 반복하지 않도록 한다'이다. 이는 동적 계획법의 메모이제이션 또는 테이블 저장에 대한 설명이며, 올바르게는 분할된 부분 문제를 해결한 결과를 결합하여 전체 해를 구한다고 해야 한다.
④ X — 틀린 표현은 '모든 가능한 경우를 탐색하여 최적해를 찾는다'이다. 이는 완전 탐색에 대한 설명이며, 올바르게는 문제를 더 작은 부분 문제로 나누어 해결하고 그 결과를 결합한다고 해야 한다.

오답 포인트

②의 '현재 단계에서 최적', ③의 '계산 결과 저장', ④의 '모든 가능한 경우 탐색'은 모두 대표적인 알고리즘 설계 키워드이지만 각각 탐욕법, 동적 계획법, 완전 탐색을 가리킨다. 분할 정복의 결정적 표현은 '분할한 부분 문제의 결과를 결합'하는 것이다.

NoteBase

다시 나오면 이렇게 판단

선지에서 문제를 작은 부분 문제로 나누는지, 각 부분 문제를 해결하는지, 그 결과를 결합하는지를 차례로 확인한다. 이 세 요소가 있으면 분할 정복으로 판단한다.

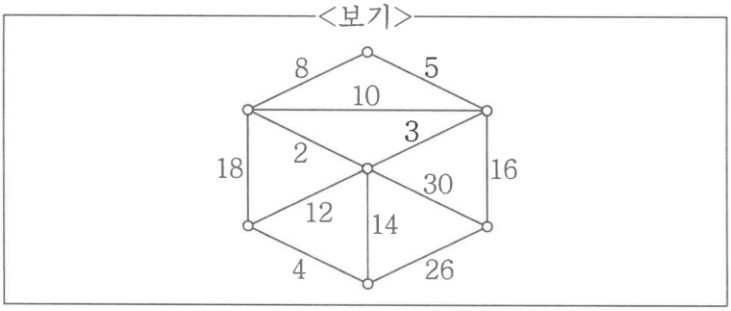
암기 포인트

분할 정복은 '나누고, 풀고, 합친다'이다.

NoteBase

4. 무방향 가중 그래프가 <보기>와 같이 주어져 있다. <보기>에서 간선 위의 수는 그 간선의 가중치를 의미한다. 이 그래프의 최소 신장 트리(MST, minimum spanning tree)를 구했을 때, 포함된 간선의 가중치의 합은?

<보기>



① 32 ② 34
③ 38 ④ 42

정답 4번

출제 포인트

이 문제는 가중치가 작은 간선부터 사이클이 생기지 않도록 선택하는 크루스칼 알고리즘으로 최소 신장 트리의 가중치 합을 구하는 문제이다.

조건 정리

주어진 값: 정점은 7개이고, 간선 가중치는 2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30이다.
구해야 할 값: 모든 정점을 연결하는 최소 신장 트리에 포함된 간선 가중치의 합이다.
주의할 조건: 정점이 7개이므로 신장 트리는 정확히 $7 - 1 = 6$개의 간선을 가지며, 사이클을 만드는 간선은 선택하지 않는다.

사용할 공식

크루스칼 알고리즘은 가중치가 작은 간선부터 검토하여 서로 다른 연결 요소를 잇는 간선만 선택한다.

정점 수를 n이라 하면 신장 트리의 간선 수는 $n - 1$이다. 이 원리를 쓰는 이유는 그래프의 모든 정점을 최소 비용으로 연결하되 사이클은 허용하지 않아야 하기 때문이다.

NoteBase

계산 과정

1. 가중치 2인 좌상단-중앙 간선과 가중치 3인 중앙-우상단 간선을 선택한다.
2. 가중치 4인 좌하단-하단 간선과 가중치 5인 상단-우상단 간선을 선택한다.
3. 가중치 8과 10인 간선은 이미 연결된 정점들 사이에 사이클을 만들므로 제외한다.
4. 가중치 12인 좌하단-중앙 간선을 선택하여 두 연결 요소를 합친다.
5. 가중치 14인 중앙-하단 간선은 사이클을 만들므로 제외한다.
6. 아직 연결되지 않은 우하단 정점은 가중치 16인 우상단-우하단 간선으로 연결한다.
7. 선택된 6개 간선의 합은 $2 + 3 + 4 + 5 + 12 + 16 = 42$이다.

선지 판단

① X — 32는 선택된 6개 간선의 합과 일치하지 않는다. 틀린 값 32는 올바르게는 42이다.
② X — 34는 최소 신장 트리의 가중치 합이 아니다. 틀린 값 34는 올바르게는 42이다.
③ X — 38은 모든 7개 정점을 연결하는 최소 신장 트리의 가중치 합과 일치하지 않는다. 틀린 값 38은 올바르게는 42이다.
④ O — 사이클 없이 모든 정점을 연결하는 간선 2, 3, 4, 5, 12, 16의 합이 42이므로 옳다.

오답 포인트

가중치가 작은 간선 6개를 기계적으로 더하면 안 된다. 가중치 8과 10은 이미 연결된 정점들 사이에 사이클을 만들므로 제외해야 하며, 우하단 정점을 연결하려면 가중치 16인 간선이 반드시 필요하다.

다시 나오면 이렇게 판단

간선을 가중치 오름차순으로 검토하고, 사이클이 생기지 않는 간선만 선택하며, 정점 수를 n이라 할 때 선택한 간선 수가 $n - 1$개가 되는 순간 멈춘다.

암기 포인트

크루스칼 알고리즘은 '가중치가 작은 순서, 사이클은 제외, 간선은 정점 수보다 1개 적게'로 기억한다.

NoteBase

5. <보기>의 숫자 배열에 버블 정렬(bubble sort) 알고리즘을 사용하여 오름차순으로 정렬하려고 한다. 첫 번째 패스 이후 배열의 상태로 가장 옳은 것은?

<보기>

7 1 3 5 2 4 9 6 8

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

정답 2번

출제 포인트

버블 정렬을 오름차순으로 수행할 때 첫 번째 패스에서 인접한 두 원소를 왼쪽부터 차례로 비교·교환한 뒤 배열이 어떻게 변하는지를 묻는 문제이다.

올바른 순서

첫 번째 패스에서는 왼쪽 원소가 오른쪽 원소보다 크면 두 원소를 교환하며 배열의 끝까지 진행한다.
7과 1부터 차례로 비교하면 배열은 다음과 같이 변한다.
1 7 3 5 2 4 9 6 8
1 3 7 5 2 4 9 6 8
1 3 5 7 2 4 9 6 8
1 3 5 2 7 4 9 6 8
1 3 5 2 4 7 9 6 8
1 3 5 2 4 7 6 9 8
1 3 5 2 4 7 6 8 9
따라서 첫 번째 패스가 끝나면 가장 큰 값 9가 맨 뒤에 확정된다. 앞에서 교환되어 오른쪽으로 이동한 값은 바로 다음 원소와 다시 비교되므로, 한 번의 패스에서도 여러 번 연속으로 이동할 수 있다.

NoteBase

선지별 해설

① X — 1 2 3 4 5 6 7 8 9는 배열 전체가 완전히 정렬된 상태이다. 틀린 부분은 첫 번째 패스만으로 모든 원소가 정렬되었다는 점이며, 올바르게는 첫 번째 패스 후 1 3 5 2 4 7 6 8 9가 된다.
② O — 왼쪽부터 인접 원소를 비교하여 큰 값을 오른쪽으로 보내면 첫 패스 후 1 3 5 2 4 7 6 8 9가 된다. 특히 9가 6, 8과 차례로 교환되어 맨 뒤에 놓인다.
③ X — 1 7 3 5 2 4 6 9 8은 일부 비교만 반영하고 패스를 끝까지 수행하지 않은 상태이다. 7은 뒤의 3, 5, 2, 4와 계속 비교·교환되어야 하고, 9도 마지막의 8과 교환되어야 한다. 올바르게는 1 3 5 2 4 7 6 8 9가 된다.
④ X — 7 1 3 5 2 4 9 8 6은 첫 비교인 7과 1의 교환조차 반영하지 않았고, 끝부분도 버블 정렬의 왼쪽부터 진행한 첫 패스 결과와 다르다. 올바르게는 7과 1을 먼저 교환하고 끝까지 인접 비교를 계속하여 1 3 5 2 4 7 6 8 9가 된다.

오답 포인트

첫 번째 패스를 전체 정렬 완료로 착각하면 ①을 고르게 된다. 또한 한 번 교환한 원소의 이동이 끝났다고 생각하면 ③처럼 일부 교환만 반영하게 된다. 버블 정렬의 한 패스는 인접 비교를 배열의 끝까지 계속하여 가장 큰 원소 하나를 맨 뒤에 확정하는 과정이다.

다시 나오면 이렇게 판단

오름차순 버블 정렬의 한 패스는 왼쪽부터 인접한 두 값을 비교하고, 왼쪽 값이 더 크면 교환한 뒤 한 칸 오른쪽의 인접 쌍을 다시 비교하는 방식으로 추적한다. 첫 패스가 끝나면 최댓값이 맨 뒤에 있는지도 확인한다.

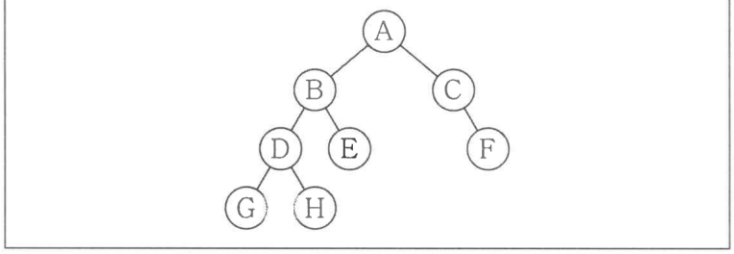
암기 포인트

버블 정렬의 첫 패스는 큰 값이 오른쪽으로 떠올라 최댓값을 맨 뒤에 확정한다.

NoteBase

6. <보기>의 트리를 후위 순회(postorder traversal)한 경우 노드의 방문 순서로 가장 옳은 것은?

<보기>



① A − B − D − G − H − E − C − F
② G − H − D − E − F − B − C − A
③ G − H − D − E − B − F − C − A
④ G − D − H − B − E − A − C − F

정답 3번

출제 포인트

트리의 후위 순회 규칙인 왼쪽 서브트리 → 오른쪽 서브트리 → 루트의 방문 순서를 묻는 문제이다.

올바른 순서

1. D의 왼쪽 자식 G와 오른쪽 자식 H를 방문한 뒤 D를 방문한다: G → H → D
2. B의 오른쪽 자식 E를 방문한 뒤 B를 방문한다: G → H → D → E → B
3. C의 오른쪽 자식 F를 방문한 뒤 C를 방문한다: F → C
4. 마지막으로 전체 트리의 루트 A를 방문한다.

따라서 전체 후위 순회 결과는 G → H → D → E → B → F → C → A이다. 자식 노드를 모두 처리한 뒤에야 그 부모 노드를 방문하기 때문에 이 순서가 된다.

선지별 해설

① X — A를 맨 처음 방문한 부분이 틀렸다. 후위 순회에서는 루트를 마지막에 방문한다. 올바르게는 G → H → D → E → B → F → C → A이다.

② X — F를 B보다 먼저 방문한 부분이 틀렸다. 왼쪽 서브트리 B의 모든 노드가 끝난 뒤 오른쪽 서브트리 C로 이동하므로 B가 F보다 먼저이다. 올바르게는 G → H → D → E → B → F → C → A이다.

③ O — 왼쪽 서브트리에서 G → H → D → E → B, 오른쪽 서브트리에서 F → C를 차례로 방문하고 마지막에 루트 A를 방문하므로 후위 순회 규칙과 일치한다.

④ X — D를 방문하기 전에 오른쪽 자식 H를 방문해야 하는데 D가 먼저 나오고, 루트 A도 마지막이 아니라 중간에 나온 부분이 틀렸다. 올바르게는 G → H → D → E → B → F → C → A이다.

NoteBase

오답 포인트

①처럼 루트를 먼저 방문하는 전위 순회와 혼동하거나, ②처럼 왼쪽 서브트리를 끝내기 전에 오른쪽 서브트리의 F로 이동하는 것이 함정이다. 후위 순회에서는 각 부모가 두 자식보다 뒤에 와야 한다.

다시 나오면 이렇게 판단

후위 순회는 각 노드에서 왼쪽 서브트리를 끝내고, 오른쪽 서브트리를 끝낸 다음, 해당 노드를 기록한다. 가장 바깥의 루트는 항상 마지막에 놓인다.

암기 포인트

후위 순회는 왼쪽 → 오른쪽 → 루트, 즉 부모는 자식 뒤에 방문한다.

NoteBase

7. <보기>의 의사코드(pseudo code)가 설명하는 정렬 알고리즘으로 가장 옳은 것은? (단, 입력은 크기가 n인 배열 A이고, 출력은 정렬된 배열 A이다.)

<보기>

```
for i = 2 to n {
      key = A[i]
      j = i - 1

      while j > 0 and A[j] > key
            A[j + 1] = A[j]
            j = j - 1

      A[j + 1] = key
}
```

① 삽입 정렬(insertion sort)
② 선택 정렬(selection sort)
③ 병합 정렬(merge sort)
④ 퀵 정렬(quick sort)

정답 1번

출제 포인트

주어진 의사코드가 이미 정렬된 앞부분에 현재 원소를 알맞은 위치로 끼워 넣는 삽입 정렬인지 판별하는 문제이다.

핵심 개념

삽입 정렬은 현재 원소를 key로 두고, 앞쪽의 더 큰 원소들을 한 칸씩 오른쪽으로 이동한 뒤 빈자리에 key를 삽입하는 정렬이다.
선택 정렬은 남은 구간의 최솟값 또는 최댓값을 골라 현재 위치와 교환한다.
병합 정렬은 배열을 나눈 뒤 정렬된 부분 배열들을 병합한다.
퀵 정렬은 피벗을 기준으로 작은 값과 큰 값을 분할하여 정렬한다.

NoteBase

선지별 해설

① O — i번째 원소를 key로 저장하고, 앞쪽에서 key보다 큰 원소를 오른쪽으로 한 칸씩 이동한 다음 key를 빈자리에 넣으므로 삽입 정렬이다.
② X — 틀린 표현은 '선택 정렬'이다. 선택 정렬이라면 남은 구간에서 최솟값 또는 최댓값을 선택하여 현재 위치와 교환해야 하며, 이 의사코드는 올바르게는 삽입 정렬이다.
③ X — 틀린 표현은 '병합 정렬'이다. 병합 정렬이라면 배열을 분할하고 정렬된 부분 배열을 병합하는 과정이 나타나야 하며, 이 의사코드는 올바르게는 삽입 정렬이다.
④ X — 틀린 표현은 '퀵 정렬'이다. 퀵 정렬이라면 피벗을 정하고 피벗을 기준으로 배열을 분할하는 과정이 나타나야 하며, 이 의사코드는 올바르게는 삽입 정렬이다.

오답 포인트

반복문이 중첩된 모습만으로 정렬 알고리즘을 판단하면 선택 정렬과 혼동하기 쉽다. 이 의사코드의 결정적 단서는 key를 따로 보관하고, 앞쪽의 큰 원소들을 교환하지 않고 한 칸씩 밀어 낸 뒤 key를 삽입한다는 점이다.

다시 나오면 이렇게 판단

key 저장 → 앞쪽 큰 원소를 오른쪽으로 이동 → 빈자리에 key 삽입의 흐름이면 삽입 정렬로 판단한다. 최솟값 선택 후 교환은 선택 정렬, 분할 후 병합은 병합 정렬, 피벗 기준 분할은 퀵 정렬이다.

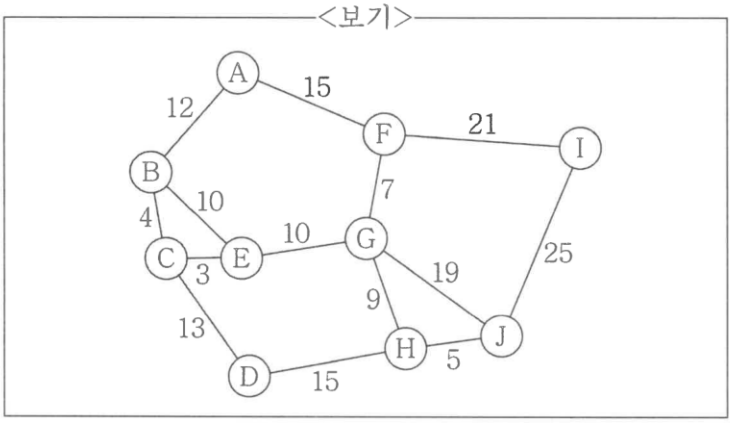
암기 포인트

key를 뽑고 큰 값을 밀어서 끼워 넣으면 삽입 정렬이다.

NoteBase

8. 무방향 가중 그래프가 <보기>와 같이 주어져 있다. 그래프에서 간선 위의 수는 그 간선의 가중치를 의미한다. 이 그래프의 정점 A에서 다른 모든 정점까지의 최단 경로의 길이(가중치의 합)로 가장 옳은 것은?

<보기>



① 201 ② 206
③ 211 ④ 216

정답 4번

출제 포인트

무방향 가중 그래프에서 정점 A로부터 각 정점까지의 최단경로 거리를 구한 뒤 그 합을 계산하는 문제이다.

조건 정리

주어진 값: 각 간선 위의 수는 그 간선의 가중치이다.
구해야 할 값: A에서 나머지 정점 B, C, D, E, F, G, H, I, J까지의 최단거리 합이다.
주의할 조건: 한 정점에 먼저 도착한 경로가 최단경로라는 보장은 없으므로, 더 작은 누적 가중치의 경로가 있는지 비교해야 한다.

사용할 공식

최단거리의 합은 $\sum_{v \neq A} \text{dist}(A, v)$이다. 모든 간선의 가중치가 양수이므로 A에서 시작하여 누적 거리가 가장 작은 정점을 차례로 확정하는 다익스트라 원리를 적용할 수 있다.

NoteBase

계산 과정

1. A와 직접 연결된 정점은 B=12, F=15이다.
2. B를 거치면 C=12+4=16이고, E는 처음에 12+10=22이다. 그러나 C를 거치면 E=16+3=19이므로 19로 갱신된다. D는 C를 거쳐 16+13=29이다.
3. F를 거치면 G=15+7=22, I=15+21=36이다.
4. G를 거치면 H=22+9=31, J=22+19=41이다. H를 거치면 J=31+5=36으로 갱신된다.
5. 따라서 최단거리는 B=12, C=16, D=29, E=19, F=15, G=22, H=31, I=36, J=36이다.
6. 합은 12 + 16 + 29 + 19 + 15 + 22 + 31 + 36 + 36 = 216이다.

선지 판단

① X — 최단거리의 합은 216이므로 201은 계산 결과와 일치하지 않는다. 틀린 값 201을 올바르게 고치면 216이다.
② X — 최단거리의 합은 216이므로 206은 계산 결과와 일치하지 않는다. 틀린 값 206을 올바르게 고치면 216이다.
③ X — 최단거리의 합은 216이므로 211은 계산 결과와 일치하지 않는다. 틀린 값 211을 올바르게 고치면 216이다.
④ O — A에서 나머지 모든 정점까지의 최단거리 합이 216이므로 옳다.
계산 결과가 216이므로 정답은 4번이다.

오답 포인트

A-B-E의 거리 22를 그대로 확정하면 안 된다. A-B-C-E의 거리는 12+4+3=19이므로 E가 갱신된다. 또한 G-J의 직접 거리로 계산한 41보다 G-H-J를 이용한 거리가 9+5=14로 더 짧으므로, A에서 J까지의 거리도 41이 아니라 36이다.

다시 나오면 이렇게 판단

시작 정점의 거리를 0으로 두고 누적 거리가 가장 작은 미확정 정점을 하나씩 확정한다. 새 경로의 누적 거리가 기존 값보다 작으면 갱신하고, 모든 최단거리가 확정된 뒤에만 그 값을 합한다.

암기 포인트

최단경로는 간선 수가 아니라 가중치의 합이 최소인 경로이며, 더 짧은 우회 경로가 나오면 반드시 거리를 갱신한다.

NoteBase

9. 레드-블랙 트리(red-black tree)에 대한 설명으로 옳은 것만을 <보기>에서 모두 고른 것은?

<보기>

ㄱ. 루트 노드는 항상 검은색이다.
ㄴ. 검은 노드는 검은 노드를 자식으로 가질 수 없다.
ㄷ. 어떤 노드에서 리프(NIL)까지 가는 모든 경로에는 같은 수의 빨간 노드가 존재한다.
ㄹ. 레드-블랙 트리는 완전 이진 트리(complete binary tree)이다.

① ㄱ
② ㄴ, ㄷ
③ ㄱ, ㄷ, ㄹ
④ ㄱ, ㄴ, ㄷ, ㄹ

정답 1번

출제 포인트

레드-블랙 트리가 균형을 유지하기 위해 만족해야 하는 색상 규칙과 경로 규칙을 구별하는 문제이다.

핵심 개념

레드-블랙 트리는 각 노드에 빨간색 또는 검은색을 부여하고 일정한 규칙을 지켜 높이를 균형에 가깝게 유지하는 이진 탐색 트리이다. 루트와 모든 리프(NIL)는 검은색이고, 빨간 노드의 자식은 모두 검은색이며, 한 노드에서 그 아래의 모든 NIL까지 가는 경로에는 같은 수의 검은 노드가 존재한다. 완전 이진 트리일 필요는 없다.

NoteBase

선지별 해설

① O
ㄱ만 고른 선지이다. 루트 노드는 항상 검은색이므로 옳다.

② X
ㄴ과 ㄷ을 고른 선지이지만 두 문장 모두 틀리다. ㄴ의 틀린 표현은 '검은 노드는 검은 노드를 자식으로 가질 수 없다'이다. 실제로는 빨간 노드가 빨간 자식을 가질 수 없으며, 검은 노드는 검은 자식을 가질 수 있다. ㄷ의 틀린 표현은 '같은 수의 빨간 노드'이다. 실제로는 어떤 노드에서 각 NIL까지 가는 모든 경로에 같은 수의 검은 노드가 존재한다.

③ X
ㄱ은 옳지만 ㄷ과 ㄹ이 틀리므로 세 문장을 함께 고를 수 없다. ㄷ은 '빨간 노드'를 '검은 노드'로 고쳐야 한다. ㄹ의 '완전 이진 트리'도 틀린 표현이며, 레드-블랙 트리는 색상 및 경로 조건을 만족하는 이진 탐색 트리이지 모든 레벨이 완전 이진 트리 형태로 채워져야 하는 트리는 아니다.

④ X
ㄱ만 옳고 ㄴ, ㄷ, ㄹ은 틀리므로 모두 고른 선지는 정답이 아니다. ㄴ은 빨간 노드의 자식이 빨간색일 수 없다는 규칙으로, ㄷ은 모든 NIL 경로의 검은 노드 수가 같다는 규칙으로 고쳐야 하며, ㄹ은 완전 이진 트리가 아니라 균형에 가까운 이진 탐색 트리라고 고쳐야 한다.

오답 포인트

ㄴ은 연속될 수 없는 색을 검은색으로 바꿔 놓았고, ㄷ은 경로마다 같아야 하는 노드의 색을 빨간색으로 바꿔 놓았다. 또한 '균형에 가까운 트리'를 '완전 이진 트리'와 같은 개념으로 보면 ㄹ을 잘못 고르게 된다.

다시 나오면 이렇게 판단

루트·NIL은 검은색인지, 빨간 노드가 빨간 자식을 가지지 않는지, 각 NIL 경로의 검은 노드 수가 같은지를 차례로 확인한다. 완전 이진 트리 여부는 레드-블랙 트리의 조건이 아니다.

암기 포인트

루트와 NIL은 검정, 빨강 다음은 검정, 모든 NIL 경로의 검정 수는 동일이다.

NoteBase

10. <보기>는 두 문자열 X와 Y가 주어졌을 때 최장 공통 부분 수열(LCS, longest common subsequence)의 길이를 구하는 알고리즘의 점화식이다. $c[i,j]$는 문자열 X_i와 Y_j의 LCS 길이라고 정의할 때, (가), (나)에 들어갈 내용으로 가장 옳은 것은? (단, LCS는 두 문자열에 공통으로 존재하는 가장 긴 부분 수열을 뜻한다. 이때, 문자의 연속성은 보장하지 않으며 순서는 유지하여야 한다.)

<보기>

$$c[i,j]=\begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ \text{(가)} & \text{if } i,j>0 \text{ and } x[i]=y[j] \\ \text{(나)} & \text{if } i,j>0 \text{ and } x[i]\neq y[j] \end{cases}$$

	(가)	(나)
①	$c[i-1,j]+1$	$\max\{c[i-1,j],c[i,j-1]\}$
②	$c[i,j-1]+1$	$\max\{c[i-1,j],c[i,j-1]\}+1$
③	$c[i-1,j-1]+1$	$\max\{c[i-1,j],c[i,j-1]\}$
④	$c[i-1,j-1]+1$	$\max\{c[i-1,j],c[i,j-1]\}+1$

정답 3번

출제 포인트

이 문제는 두 문자열의 마지막 문자가 같은 경우와 다른 경우를 나누어 LCS 길이의 점화식을 완성하는 문제이다.

조건 정리

주어진 값: $c[i, j]$는 접두 문자열 X_i와 Y_j의 LCS 길이이다.
구해야 할 값: $x[i] = y[j]$일 때의 (가)와 $x[i] \neq y[j]$일 때의 (나)이다.
주의할 조건: 부분 수열은 연속할 필요는 없지만 원래 문자열에서의 순서는 유지해야 한다.

사용할 공식

c[i,j]=cases0 & i=0 또는 j=0 \ c[i-1,j-1]+1 & x[i]=y[j] \ \c[i-1,j],c[i,j-1]\ & x[i]≠ y[j]cases
마지막 문자가 같으면 그 문자를 공통 부분 수열에 추가할 수 있으므로 두 문자열에서 마지막 문자를 모두 제외한 LCS에 1을 더한다. 마지막 문자가 다르면 두 문자를 함께 새 공통 문자로 셀 수 없으므로 어느 한쪽의 마지막 문자를 제외한 두 경우 중 큰 값을 택한다.

NoteBase

계산 과정

1. $x[i] = y[j]$이면 공통인 마지막 문자 하나를 사용하므로 $c[i, j] = c[i-1, j-1] + 1$이다.
2. $x[i] \neq y[j]$이면 X의 마지막 문자를 제외한 $c[i-1, j]$와 Y의 마지막 문자를 제외한 $c[i, j-1]$를 비교한다.
3. 따라서 $c[i, j] = \max\{c[i-1, j], c[i, j-1]\}$이며, (가)와 (나)가 모두 이에 맞는 선지는 ③이다.

선지 판단

① X — (나)는 맞지만, (가)의 $c[i-1, j] + 1$은 Y 쪽 마지막 문자를 제외하지 않아 공통인 마지막 문자를 중복 반영할 수 있다. 올바르게는 (가)가 $c[i-1, j-1] + 1$이다.
② X — (가)의 $c[i, j-1] + 1$은 두 문자열의 공통 마지막 문자를 처리할 때 대각선 이전 상태를 사용하지 않은 식이다. 또한 문자가 다른 경우에는 새 공통 문자가 생기지 않으므로 (나)에 1을 더할 수 없다. 올바르게는 (가)가 $c[i-1, j-1] + 1$, (나)가 $\max\{c[i-1, j], c[i, j-1]\}$이다.
③ O — 문자가 같을 때는 대각선 값 $c[i-1, j-1]$에 1을 더하고, 다를 때는 위쪽 값과 왼쪽 값 중 큰 값인 $\max\{c[i-1, j], c[i, j-1]\}$을 사용하므로 점화식과 일치한다.
④ X — (가)는 맞지만, 문자가 다른 경우에는 두 마지막 문자를 공통 문자로 추가할 수 없으므로 (나)의 바깥에 붙은 +1이 잘못되었다. 올바르게는 (나)가 $\max\{c[i-1, j], c[i, j-1]\}$이다.
따라서 정답은 3번이다.

오답 포인트

핵심 함정은 문자가 같은 경우와 다른 경우 모두에 1을 더한다고 착각하는 것이다. 1은 두 마지막 문자가 같아 그 문자를 LCS에 실제로 추가할 때만 더하며, 이때 비교 기준도 위나 왼쪽이 아니라 두 문자를 모두 제외한 대각선 상태이다.

다시 나오면 이렇게 판단

LCS 점화식은 마지막 문자를 먼저 비교한다. 같으면 '대각선에 1 추가', 다르면 '위와 왼쪽 중 최댓값'으로 판단한다.

암기 포인트

LCS는 같으면 대각선 +1, 다르면 위·왼쪽의 최댓값이다.

NoteBase

11. <보기>는 정수 n을 입력하면 계산 결과를 출력(return)하는 알고리즘의 의사코드(pseudo code)이다. f(n)을 호출했을 때, 출력되는 값이 100보다 크게 되는 정수 n의 최솟값은?

<보기>

```
f(n)
  if (n<1) return 1
  else return f(n-1) + f(n-3) + 1
```

① 9 ② 10
③ 11 ④ 12

정답 2번

출제 포인트

재귀함수의 종료 조건과 점화식을 순서대로 적용하여 함수값이 처음 100을 초과하는 정수 n을 구하는 문제이다.

조건 정리

주어진 조건: $n < 1$이면 $f(n) = 1$이다.
점화식: $n \geq 1$이면 $f(n) = f(n-1) + f(n-3) + 1$이다.
구해야 할 값: $f(n) > 100$을 처음 만족하는 정수 n이다.

사용할 공식

$f(n) = f(n-1) + f(n-3) + 1$
이 점화식을 쓰는 이유는 $n \geq 1$일 때 의사코드가 두 재귀 호출의 반환값과 1을 더해 결과를 반환하기 때문이다.
$n - 3$이 0보다 작아져도 종료 조건에 따라 그 함수값은 1이다.

계산 과정

종료 조건에서 $f(0) = f(-1) = f(-2) = 1$이다.
점화식을 차례로 적용하면 $f(1) = 3$, $f(2) = 5$, $f(3) = 7$, $f(4) = 11$, $f(5) = 17$, $f(6) = 25$, $f(7) = 37$, $f(8) = 55$, $f(9) = 81$, $f(10) = 119$이다.
따라서 $f(9) = 81 \leq 100$이고 $f(10) = 119 > 100$이므로 처음 100을 초과하는 때는 $n = 10$이다.

NoteBase

선지 판단

① 9: X — $f(9) = 81$이므로 100보다 크지 않다. 틀린 값 9가 아니라 올바른 최솟값은 10이다.
② 10: O — $f(10) = 119 > 100$이고 바로 앞의 $f(9) = 81 \le 100$이므로 최솟값이다.
③ 11: X — $n = 10$에서 이미 100을 초과하므로 11은 최솟값이 아니다. 올바르게는 10이다.
④ 12: X — $n = 10$에서 이미 조건을 만족하므로 12는 최솟값이 아니다. 올바르게는 10이다.

오답 포인트

함수값이 100 이상인 때가 아니라 100보다 큰 때를 찾아야 하며, 조건을 만족하는 아무 정수가 아니라 최초의 정수를 찾아야 한다. 따라서 $f(10)$만 확인하는 데서 끝내지 않고 바로 앞 값 $f(9)$도 함께 비교해야 한다.

다시 나오면 이렇게 판단

재귀식의 종료 조건에서 필요한 초기값을 먼저 정한 뒤 작은 n부터 점화식을 계산한다. 기준을 처음 넘은 값과 그 직전 값을 함께 확인하면 최솟값을 확정할 수 있다.

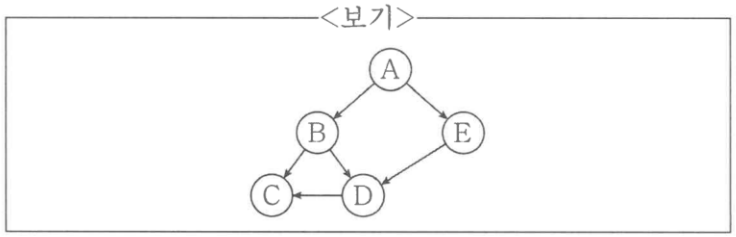
암기 포인트

최초 초과 확인은 직전 값 $\le$ 기준, 현재 값 $>$ 기준의 두 조건으로 판단한다.

NoteBase

12. <보기> 그래프의 위상정렬(topological sort)의 결과로 가장 옳은 것은?

<보기>



① A→B→E→D→C
② A→E→D→C→B
③ A→B→C→E→D
④ A→B→C→D→E

정답 1번

출제 포인트

위상정렬은 모든 방향 간선의 선행 정점이 후행 정점보다 먼저 나오도록 정점을 배열하는 방법이다. 이 문제는 그래프의 간선이 나타내는 선후 관계를 모두 만족하는 순서를 찾는 문제이다.

올바른 순서

그래프의 선후 관계는 A→B, A→E, B→C, B→D, E→D, D→C이다. 따라서 A가 B와 E보다 먼저이고, B와 E가 모두 D보다 먼저이며, D가 C보다 먼저여야 한다. ①의 A→B→E→D→C는 이 조건을 모두 만족한다. B와 E 사이에는 직접적인 간선이 없으므로 A→E→B→D→C도 가능한 위상정렬이지만 선지에는 없다.

선지별 해설

① O — A가 B와 E보다 먼저 나오고, B와 E가 모두 D보다 먼저 나오며, D가 C보다 먼저 나온다. 모든 간선의 방향을 만족한다.
② X — D와 C가 B보다 먼저 나온 부분이 틀렸다. B→D와 B→C이므로 B가 D와 C보다 먼저 나와야 한다. 올바르게는 A→E→B→D→C와 같이 배열할 수 있다.
③ X — C가 D보다 먼저 나온 부분이 틀렸다. D→C이므로 D가 C보다 먼저 나와야 하고, E→D도 지켜야 한다. 올바르게는 A→B→E→D→C이다.
④ X — C가 D보다 먼저이고 D가 E보다 먼저 나온 부분이 틀렸다. D→C와 E→D에 따라 E→D→C의 순서를 지켜야 한다. 올바르게는 A→B→E→D→C이다.

오답 포인트

A에서 시작한다는 사실만 확인하고 고르면 안 된다. 특히 E→D와 D→C가 연속된 제약이므로 E가 D보다 먼저, D가 C보다 먼저인지 확인해야 한다. 또한 B→D와 B→C 때문에 B도 D와 C보다 먼저 나와야 한다.

NoteBase

다시 나오면 이렇게 판단

각 간선을 '출발 정점이 도착 정점보다 먼저'라는 제약으로 바꾼 뒤, 선지의 순서가 모든 제약을 만족하는지 하나씩 대조한다. 직접 연결되지 않은 정점끼리는 순서가 바뀔 수 있으므로 하나의 고정된 배열만 외우지 않는다.

암기 포인트

위상정렬의 핵심은 모든 간선 u→v에서 u가 v보다 먼저 나오는 것이다. 이 그래프의 핵심 사슬은 A→E→D→C이며, B도 D와 C보다 앞선다.

NoteBase

13. 이진 탐색 트리(BST, binary search tree)에 대한 설명으로 가장 옳지 않은 것은?

① 중복 키를 허용하지 않는 경우, 모든 노드에 대해 왼쪽 서브트리의 키는 해당 노드의 키보다 작고, 오른쪽 서브트리의 키는 해당 노드의 키보다 크다.

② 루트 노드는 항상 전체 원소의 중앙값이어야 한다.

③ BST에서 중위 순회(inorder traversal)를 하면 키 값이 오름차순으로 출력된다.

④ 탐색 · 삽입 · 삭제 연산의 시간복잡도는 트리의 높이에 비례한다.

정답 2번

출제 포인트

이 문제는 이진 탐색 트리의 정렬 규칙, 루트의 선택, 중위 순회 결과, 기본 연산의 시간복잡도를 묻는 문제이다.

핵심 개념

이진 탐색 트리는 각 노드를 기준으로 왼쪽 서브트리에는 더 작은 키, 오른쪽 서브트리에는 더 큰 키를 배치하여 탐색을 효율화한 이진 트리이다. 루트는 입력·삽입 순서나 트리 구성 방식에 따라 달라지므로 전체 원소의 중앙값일 필요가 없다. 중위 순회는 왼쪽 서브트리, 현재 노드, 오른쪽 서브트리 순서로 방문한다.

선지별 해설

① O — 중복 키를 허용하지 않는 BST에서는 모든 노드에 대해 왼쪽 서브트리의 키가 해당 노드의 키보다 작고, 오른쪽 서브트리의 키가 해당 노드의 키보다 크다.

② X — 틀린 표현은 '루트 노드는 항상 전체 원소의 중앙값'이다. BST가 만족해야 하는 것은 각 노드를 기준으로 한 좌우 키의 대소 관계이며, 올바르게는 루트가 입력·삽입 순서나 구성 방식에 따라 정해져 중앙값이 아닐 수도 있다고 해야 한다.

③ O — 중위 순회는 왼쪽, 노드, 오른쪽 순서로 방문하므로 중복 키가 없는 BST의 키는 오름차순으로 출력된다.

④ O — 탐색·삽입·삭제는 루트에서 아래로 한 경로를 따라 진행하므로 시간복잡도는 트리 높이를 h라 할 때 $O(h)$이다. 균형 트리에서는 $O(\log n)$, 한쪽으로 치우친 트리에서는 최악의 경우 $O(n)$이 된다.

오답 포인트

BST의 좌우 서브트리가 항상 같은 크기라고 오해하면 ②의 '중앙값'을 맞는 설명으로 판단하기 쉽다. BST의 정렬 조건과 균형 조건은 서로 다르며, 일반 BST는 균형을 보장하지 않는다.

NoteBase

다시 나오면 이렇게 판단

BST 문제에서는 먼저 각 노드의 왼쪽 < 현재 노드 < 오른쪽 관계를 확인한다. 이어서 중위 순회는 정렬된 순서를 만들고, 탐색·삽입·삭제의 비용은 노드 수 자체가 아니라 트리 높이에 좌우된다는 기준으로 판단한다.

암기 포인트

BST는 좌소우대, 중위 순회는 오름차순, 연산 비용은 높이 h에 비례한다. 루트가 중앙값이라는 보장은 없다.

NoteBase

14. <보기>는 문자열 탐색 알고리즘인 라빈-카프(Rabin-Karp) 알고리즘에 대한 설명이다. 옳은 설명을 모두 고른 것은?

<보기>

ㄱ. 문자열 탐색을 위해 해시 함수를 사용한다.
ㄴ. 알고리즘의 효율성은 해시 함수 계산 비용을 줄이는 데에 있다.
ㄷ. 패턴의 해시값과 본문 부분 문자열의 해시값이 일치하더라도 실제 문자가 일치하는지 추가 확인이 필요하다.

① ㄱ
② ㄱ, ㄴ
③ ㄴ, ㄷ
④ ㄱ, ㄴ, ㄷ

정답 4번

출제 포인트

라빈-카프 알고리즘에서 해시 함수와 롤링 해시를 사용하는 이유, 그리고 해시 충돌에 대비한 실제 문자열 비교의 필요성을 묻는 문제이다.

핵심 개념

라빈-카프 알고리즘은 패턴과 본문의 각 부분 문자열을 해시값으로 비교하여 일치 후보를 빠르게 찾는 문자열 탐색 알고리즘이다. 롤링 해시는 현재 부분 문자열의 해시값을 이용해 다음 부분 문자열의 해시값을 갱신하므로 매 위치에서 해시를 처음부터 계산하는 비용을 줄인다. 다만 서로 다른 문자열도 같은 해시값을 가질 수 있는 해시 충돌이 있으므로, 해시값이 같으면 실제 문자열까지 비교해야 한다.

선지별 해설

① X — ㄱ만 옳다고 한 부분이 틀렸다. 라빈-카프 알고리즘이 해시 함수를 사용한다는 ㄱ은 옳지만, ㄴ과 ㄷ도 옳다. 올바르게는 ㄱ, ㄴ, ㄷ을 모두 골라야 한다.
② X — ㄱ과 ㄴ만 옳다고 한 부분이 틀렸다. 해시값이 일치해도 해시 충돌 가능성이 있으므로 실제 문자의 일치 여부를 추가로 확인해야 한다는 ㄷ도 옳다. 올바르게는 ㄱ, ㄴ, ㄷ을 모두 골라야 한다.
③ X — ㄴ과 ㄷ만 옳다고 한 부분이 틀렸다. 라빈-카프 알고리즘은 문자열 탐색에 해시 함수를 사용하므로 ㄱ도 옳다. 올바르게는 ㄱ, ㄴ, ㄷ을 모두 골라야 한다.
④ O — ㄱ은 해시 함수의 사용, ㄴ은 롤링 해시에 의한 해시 계산 비용 절감, ㄷ은 해시 충돌에 대비한 실제 문자열 확인을 각각 정확히 설명한다. 따라서 ㄱ, ㄴ, ㄷ이 모두 옳다.

오답 포인트

해시값이 같으면 문자열도 반드시 같다고 생각하면 ㄷ을 제외하기 쉽다. 해시값의 일치는 일치 후보를 뜻할 뿐이며, 충돌 가능성 때문에 실제 문자열 비교가 최종 확인 단계로 필요하다.

NoteBase

다시 나오면 이렇게 판단

라빈-카프 문제에서는 '해시로 후보 탐색 → 롤링 해시로 다음 구간의 계산 절감 → 해시 일치 시 실제 문자열 비교'의 세 단계를 기준으로 각 진술을 판단한다.

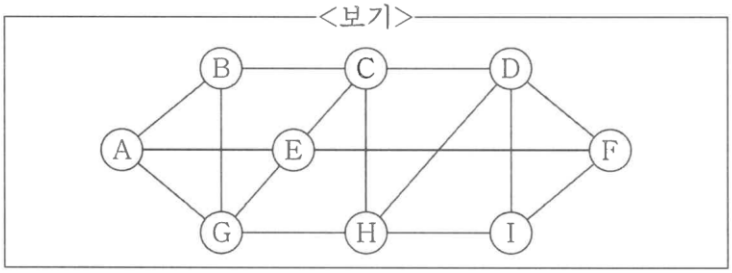
암기 포인트

라빈-카프는 해시로 찾고, 롤링으로 줄이고, 충돌은 문자 비교로 확인한다.

NoteBase

15. <보기>의 그래프에서 A부터 너비 우선 탐색(BFS, breadth first search)을 수행하는 경우, 정점의 방문 순서로 가장 옳은 것은? (단, 탐색되지 않은 이웃(neighbor)이 2개 이상 있는 경우 알파벳 순서로 탐색한다.)

<보기>



① A − B − C − D − F − I − H − G − E
② A − B − E − G − C − F − H − D − I
③ A − B − G − E − C − H − F − D − I
④ A − B − C − D − F − E − G − H − I

정답 2번

출제 포인트

이 문제는 A에서 시작하는 너비 우선 탐색에서 같은 깊이의 정점을 큐에 넣는 순서와, 미방문 이웃이 여러 개일 때의 알파벳 우선 규칙을 적용하는 문제이다. A의 이웃은 B, E, G이고, 이후 각 정점을 큐에서 꺼내며 새 이웃을 알파벳 순서로 추가한다.

NoteBase

선지별 해설

① X
- A 다음에 B가 방문되면 B에서 새로 발견되는 C는 큐의 기존 E, G 뒤에 들어가므로 C가 E, G보다 먼저 방문될 수 없다. 또한 F에서 발견되는 I는 이미 큐에 있는 H와 D 뒤에 들어간다.
- 틀린 순서 'A-B-C-D-E-I-H-G-F' → 올바른 순서는 'A-B-E-G-C-F-H-D-I'이다.

② O
- A에서 B, E, G를 차례로 큐에 넣는다. B에서는 C, E에서는 F, G에서는 H를 새로 넣고, 이어 C에서는 D, F에서는 I를 넣는다. 따라서 방문 순서는 A-B-E-G-C-F-H-D-I이다.

③ X
- A의 미방문 이웃이 B, E, G이므로 알파벳 순서상 E가 G보다 먼저 큐에 들어가며, 방문 순서도 E가 G보다 먼저여야 한다.
- 틀린 순서 'A-B-G-E-C-H-F-D-I' → 올바른 순서는 'A-B-E-G-C-F-H-D-I'이다.

④ X
- C와 D는 A에서 거리가 각각 2와 3이므로, 거리 1인 E와 G 및 거리 2인 C, F, H를 처리하기 전에 D를 방문할 수 없다.
- 틀린 순서 'A-B-C-D-E-F-G-H-I' → 올바른 순서는 'A-B-E-G-C-F-H-D-I'이다.

오답 포인트

그래프에 그려진 위치나 경로 하나를 따라가는 방식으로 순서를 정하면 ①이나 ④처럼 깊이 우선 탐색에 가까운 순서를 고르기 쉽다. BFS에서는 정점을 발견한 즉시 큐에 넣으므로, B에서 C를 발견해도 이미 큐에 들어 있던 E와 G가 C보다 먼저 방문된다.

다시 나오면 이렇게 판단

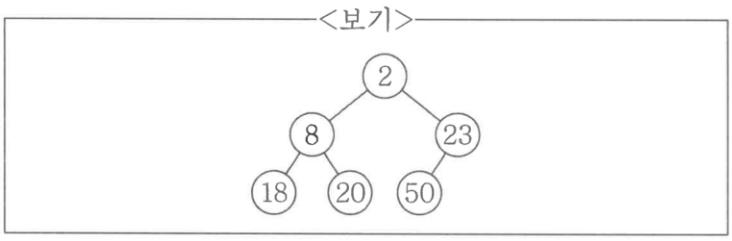
시작 정점을 방문한 뒤 미방문 이웃을 정해진 우선순위로 큐에 넣고, 큐의 맨 앞 정점을 하나씩 꺼내면서 그 정점의 미방문 이웃만 뒤에 추가한다. 이 문항의 큐 변화는 [B, E, G] → [E, G, C] → [G, C, F] → [C, F, H] → [F, H, D] → [H, D, I]이다.

암기 포인트

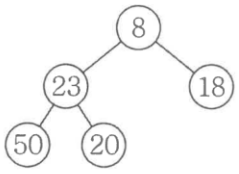
BFS는 큐를 사용하며 가까운 정점부터 방문한다. 같은 단계의 후보는 문제에서 제시한 우선순위에 따라 큐에 들어간 순서대로 처리된다.

16. <보기>의 최소 힙(min heap)에서 루트 노드를 삭제한 경우, 최소 힙 조건을 만족하도록 연산을 거친 후의 구조로 가장 옳은 것은?

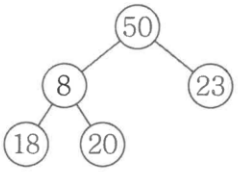
<보기>



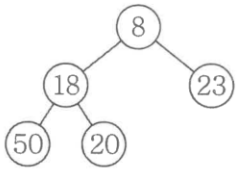
①



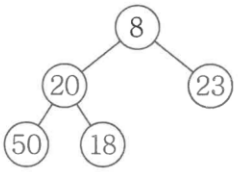
②



③



④



정답 3번

출제 포인트

최소 힙에서 루트 노드를 삭제한 뒤 마지막 노드를 루트로 옮기고, 더 작은 자식과 차례로 교환하여 최소 힙 조건을 회복하는 과정을 묻는 문제이다.

올바른 순서

1. 루트 2를 삭제한다.
2. 마지막 노드 50을 빈 루트 자리로 옮긴다. 이때 임시 구조의 레벨 순서는 50, 8, 23, 18, 20이다.
3. 50의 두 자식 8과 23 중 더 작은 8과 교환한다. 레벨 순서는 8, 50, 23, 18, 20이 된다.
4. 내려간 50의 두 자식 18과 20 중 더 작은 18과 교환한다. 최종 레벨 순서는 8, 18, 23, 50, 20이 된다.

각 단계에서 더 작은 자식과 교환해야 부모가 두 자식보다 작거나 같은 최소 힙 조건을 회복할 수 있다.

NoteBase

선지별 해설

① X — 루트의 왼쪽 자식을 23, 오른쪽 자식을 18로 둔 부분이 잘못되었고, 23의 자식인 20이 부모 23보다 작아 최소 힙 조건도 위반한다. 올바르게는 루트 8의 왼쪽 자식이 18, 오른쪽 자식이 23이고, 18의 자식이 50과 20이어야 한다.

② X — 50을 루트로 옮긴 직후 하향 교환을 하지 않은 구조이므로 50이 자식 8과 23보다 커서 최소 힙 조건을 위반한다. 올바르게는 50을 먼저 8과, 이어서 18과 교환하여 루트가 8인 구조로 만들어야 한다.

③ O — 50을 마지막 노드에서 루트로 옮긴 뒤 8, 다시 18과 교환하면 레벨 순서가 8, 18, 23, 50, 20이 된다. 모든 부모가 자식보다 작거나 같으므로 최소 힙 조건을 만족한다.

④ X — 두 번째 하향 교환에서 50과 20을 교환하여 왼쪽 자식을 20으로 둔 부분이 잘못되었고, 그 결과 자식 18이 부모 20보다 작다. 올바르게는 두 자식 18과 20 중 더 작은 18을 50과 교환하여 왼쪽 자식을 18로 두어야 한다.

오답 포인트

마지막 노드 50을 루트로 옮기는 데서 연산이 끝난다고 보면 ②를 고르기 쉽다. 또한 하향 교환에서는 임의의 자식이 아니라 두 자식 중 더 작은 값과 교환해야 하므로, 두 번째 단계에서 20을 선택한 ④도 최소 힙이 되지 않는다.

다시 나오면 이렇게 판단

최소 힙의 루트 삭제 문제는 마지막 노드를 루트로 이동한 뒤, 현재 노드가 자식보다 클 동안 두 자식 중 더 작은 값과 교환하는 순서로 판단한다. 교환이 끝나면 모든 부모가 각 자식보다 작거나 같은지 다시 확인한다.

암기 포인트

최소 힙 삭제는 '마지막 값을 루트로, 더 작은 자식을 따라 아래로'이다.

NoteBase

17. 해시 테이블의 충돌 문제 해결을 위한 방법 중 개방 주소법(open addressing)에 해당하지 않는 것은?
① 선형 조사법(linear probing)
② 체이닝(chaining)
③ 이차 조사법(quadratic probing)
④ 이중 해싱(double hashing)

정답 2번

출제 포인트

해시 충돌 해결법 중 개방 주소법에 속하는 방법과 별도의 저장 구조를 사용하는 체이닝을 구분하는 문제이다.

핵심 개념

개방 주소법: 충돌이 발생하면 해시 테이블 안의 다른 빈 슬롯을 조사하여 원소를 저장하는 방법이다. 키워드는 테이블 내부의 빈 주소 탐색이다.
체이닝: 같은 해시 주소로 충돌한 원소들을 연결 리스트 등의 별도 구조에 묶어 저장하는 방법이다. 키워드는 버킷별 연결 구조이다.
선형 조사법·이차 조사법·이중 해싱은 조사 위치를 정하는 방식은 다르지만 모두 테이블 내부의 다른 슬롯을 찾는 개방 주소법이다.

선지별 해설

① O — 선형 조사법은 충돌한 위치에서 일정한 간격으로 다음 슬롯을 차례로 조사하므로 개방 주소법에 해당한다.
② X — 체이닝은 충돌한 원소를 연결 리스트 등의 별도 구조에 저장하므로 개방 주소법에 해당하지 않는다. 틀린 표현은 '체이닝이 개방 주소법에 해당한다'는 분류이며, 올바르게는 체이닝은 분리 연결법에 해당한다.
③ O — 이차 조사법은 조사 간격을 제곱 형태로 변화시키며 테이블 내부의 빈 슬롯을 찾으므로 개방 주소법에 해당한다.
④ O — 이중 해싱은 두 번째 해시 함수로 조사 간격을 정해 테이블 내부의 다른 슬롯을 찾으므로 개방 주소법에 해당한다.

오답 포인트

체이닝도 해시 충돌을 해결한다는 공통점 때문에 개방 주소법으로 착각하기 쉽다. 그러나 개방 주소법은 테이블 내부의 빈 슬롯을 찾고, 체이닝은 같은 주소의 원소를 별도 연결 구조에 모은다는 차이가 있다.

NoteBase

다시 나오면 이렇게 판단

충돌 후 저장 위치를 기준으로 판단한다. 테이블 내부의 다른 슬롯을 조사하면 개방 주소법이고, 연결 리스트 등 별도 구조에 저장하면 체이닝이다.

암기 포인트

선형·이차·이중은 빈 주소를 조사하고, 체이닝은 충돌 원소를 연결한다.

NoteBase

18. <보기>는 막대 자르기(rod cutting) 문제의 재귀적 의사코드(pseudo code)이다. 막대의 길이 정수 n과 가격 배열 p를 입력 받아, 막대를 잘랐을 때 가능한 이익의 최댓값을 구한다. 이때, (가)에 들어갈 내용으로 가장 옳은 것은? (단, 배열 p의 길이는 n이며, $p[i]$는 길이가 i인 막대의 가격이다.)

<보기>

```
CUT_ROD(p,n)
    if n==0
      return 0
    q=-∞
    for i=1 to n
      q=max{q,p[i]+ _____(가)_____ }
    return q
```

① CUT_ROD(p,i-1)
② CUT_ROD(p,i)
③ CUT_ROD(p,i+1)
④ CUT_ROD(p,n-i)

정답 4번

출제 포인트

막대 자르기의 최적 부분 구조를 이용해 재귀식을 완성하는 문제이다. 길이 n인 막대에서 첫 조각의 길이를 i로 정하면, 그 조각의 가격 $p[i]$에 남은 길이 $n-i$의 최대 이익을 더해야 한다.

조건 정리

주어진 값: 막대의 전체 길이 n, 길이 i인 막대의 가격 $p[i]$이다.
구해야 할 값: 길이 n인 막대를 잘라 얻을 수 있는 최대 이익이다.
주의할 조건: 첫 조각의 길이가 i이면 남은 막대의 길이는 전체 길이에서 i를 뺀 $n-i$이다.

사용할 공식

$$r_n = \max_{1 \le i \le n} \{p[i] + r_{n-i}\}, \qquad r_0 = 0$$

여기서 r_k는 길이 k인 막대에서 얻을 수 있는 최대 이익이다. 첫 조각의 길이 i를 하나씩 시도한 뒤, 첫 조각의 가격과 남은 길이의 최적 이익을 더한 값 중 최댓값을 택하므로 이 재귀식을 사용한다.

NoteBase

계산 과정

1. 반복문에서 첫 조각의 길이를 i로 정한다.
2. 첫 조각을 떼고 남는 길이는 $n - i$이다.
3. 해당 선택의 최대 이익 후보는 $p[i]$ + CUT_ROD (p, $n - i$)이다.
4. 모든 $i = 1, 2, \ldots, n$에 대한 후보 중 최댓값을 q에 저장하므로 (가)는 CUT_ROD (p, $n - i$)이다.

선지 판단

① X — CUT_ROD (p, $i - 1$)은 남은 길이를 첫 조각 길이 i만으로 정하므로 전체 길이 n을 반영하지 못한다. 틀린 표현 $i - 1$은 $n - i$로 고쳐야 한다.
② X — CUT_ROD (p, i)은 이미 가격 $p[i]$를 더한 첫 조각과 같은 길이 i를 다시 푸는 호출이다. 틀린 표현 i는 첫 조각을 제외한 남은 길이 $n - i$로 고쳐야 한다.
③ X — CUT_ROD (p, $i + 1$)도 남은 길이를 나타내지 않으며, 선택한 조각보다 1 큰 길이를 재귀 호출할 근거가 없다. 틀린 표현 $i + 1$은 $n - i$로 고쳐야 한다.
④ O — CUT_ROD (p, $n - i$)는 길이 i인 첫 조각을 자른 뒤 남은 길이 $n - i$에서 얻는 최대 이익을 정확히 구한다. 따라서 $p[i]$ + CUT_ROD (p, $n - i$)가 올바른 후보 이익이다.

오답 포인트

재귀 호출의 인수를 반복 변수 i 자체로 착각하기 쉽다. 그러나 i는 이번에 떼어 낸 첫 조각의 길이이고, 재귀 호출이 해결해야 할 부분 문제의 크기는 남은 길이 $n - i$이다.

다시 나오면 이렇게 판단

재귀식의 각 항에서 먼저 이번 단계가 소비한 크기를 확인하고, 전체 크기에서 그 크기를 뺀 값을 다음 부분 문제의 인수로 잡는다. 이 문제에서는 전체 길이 n에서 첫 조각 길이 i를 뺀 $n - i$가 된다.

암기 포인트

막대 자르기는 '첫 조각의 가격 + 남은 길이의 최적 이익'이므로 $p[i] + r_{n-i}$이다.

NoteBase

19. 정렬 알고리즘에 대한 설명 중 가장 옳지 않은 것은?

① 선택 정렬(selection sort) 알고리즘은 n개의 원소를 갖는 리스트를 정렬하는 경우, 총 $n(n-1)/2$회의 비교연산이 진행된다.

② 힙 정렬(heap sort) 알고리즘은 완전 이진 트리 형태의 자료구조 힙을 활용한다.

③ 합병 정렬(merge sort) 알고리즘은 동적 계획법(dynamic programming) 방식을 기반으로 동작한다.

④ 기수 정렬(radix sort) 알고리즘은 자릿수가 d인 자연수 n개로 구성된 리스트를 정렬할 때, $d=n$이라면 $O(n^2)$의 시간복잡도를 갖는다.

정답 3번

출제 포인트

선택 정렬, 힙 정렬, 합병 정렬, 기수 정렬의 핵심 동작 방식과 시간복잡도를 구별하는 문제이다.

핵심 개념

선택 정렬은 매 단계에서 남은 원소 중 최솟값 또는 최댓값을 선택해 제자리에 놓는 비교 정렬이다. 힙 정렬은 완전 이진 트리 형태의 힙을 이용하는 비교 정렬이다. 합병 정렬은 리스트를 나누고 각각 정렬한 뒤 합치는 분할 정복 정렬이다. 기수 정렬은 자릿수별로 원소를 분류하는 비비교 정렬이며, 자릿수가 d이고 원소 수가 n일 때 자릿값의 범위를 상수로 보면 시간복잡도는 $O(dn)$이다.

선지별 해설

① O — 선택 정렬은 첫 위치에서 $n-1$회, 다음 위치에서 $n-2$회처럼 비교하므로 총 비교 횟수는 $(n-1)+(n-2)+\cdots+1=n(n-1)/2$회이다.

② O — 힙은 마지막 레벨을 제외한 모든 레벨이 채워지고 마지막 레벨은 왼쪽부터 채워지는 완전 이진 트리 형태의 자료구조이므로, 힙 정렬은 이를 활용한다.

③ X — 틀린 표현은 '동적 계획법 방식'이다. 합병 정렬은 문제를 작은 부분 리스트로 나눈 뒤 정렬된 결과를 합치는 '분할 정복 방식'을 기반으로 동작한다.

④ O — 기수 정렬은 각 자릿수를 차례로 처리하므로 자릿값 범위를 상수로 보면 $O(dn)$이다. $d=n$이면 $O(n^2)$이 되어 선지의 설명이 옳다.

오답 포인트

③의 '작은 문제를 해결한다'는 인상 때문에 동적 계획법과 혼동하기 쉽다. 합병 정렬은 부분 문제의 해를 저장해 중복 계산을 줄이는 방식이 아니라, 입력을 둘로 나누어 정렬한 결과를 병합하는 분할 정복 방식이다.

NoteBase

다시 나오면 이렇게 판단

정렬 알고리즘의 대표 구조를 먼저 대응시킨다. 선택 정렬은 남은 구간에서 선택, 힙 정렬은 완전 이진 트리형 힙, 합병 정렬은 분할 후 병합, 기수 정렬은 자릿수별 반복 처리로 판단한다.

암기 포인트

합병 정렬은 '나누고 합친다'는 분할 정복이고, 동적 계획법이 아니다.

NoteBase

20. <보기>에서 설명하는 문제 부류에 속하지 않는 것으로 가장 옳은 것은?

<보기>

- 현재 이 부류 문제에 대한 다항 시간의 알고리즘이 알려져 있지 않다.
- 이 부류의 문제 사이에는 다항 시간 환원(reduction)이 가능하다.
- 이 부류에 속하는 문제 중 어느 한 문제만 다항식 시간에 해결되면, 모든 다른 문제들이 다항 시간에 해결 가능하다.

① 만족성(SAT) 문제
② 부분 집합의 합(subset sum) 문제
③ 0-1 배낭 결정(knapsack decision) 문제
④ 모든 쌍 최단 경로(all pairs shortest paths) 문제

정답 4번

출제 포인트

보기의 세 특징이 설명하는 NP-완전 문제를 식별하고, 각 선지가 NP-완전 문제인지 다항 시간에 해결 가능한 P 문제인지 구분하는 문제이다.

핵심 개념

NP-완전 문제는 NP에 속하면서 모든 NP 문제가 다항 시간에 환원되는 문제이다. NP-완전 문제 하나를 다항 시간에 해결하면 모든 NP 문제가 다항 시간에 해결되어 P=NP가 된다.
P 문제는 다항 시간 알고리즘이 알려진 문제이며, NP-완전 문제와 구별된다.

선지별 해설

① O — 만족성(SAT) 문제는 대표적인 NP-완전 결정 문제이므로 보기가 설명하는 부류에 속한다.
② O — 부분 집합의 합(subset sum) 결정 문제는 합이 목표값이 되는 부분 집합의 존재 여부를 묻는 NP-완전 문제이다.
③ O — 0-1 배낭 결정(knapsack decision) 문제는 주어진 제한과 목표를 만족하는 선택의 존재 여부를 묻는 NP-완전 문제이다.
④ X — 모든 쌍 최단 경로 문제는 Floyd-Warshall 알고리즘 등으로 다항 시간에 해결할 수 있으므로 P에 속한다. 틀린 표현은 이 문제가 보기의 NP-완전 문제 부류에 속한다는 판단이며, 올바르게는 NP-완전 문제의 예가 아니라 다항 시간에 해결 가능한 문제이다.

NoteBase

오답 포인트

④도 그래프의 최단 경로를 다루어 계산량이 커 보이지만, 입력 크기에 대한 다항 시간 알고리즘이 이미 알려져 있다. 문제의 체감 난도가 아니라 다항 시간 알고리즘의 존재 여부로 구분해야 한다.

다시 나오면 이렇게 판단

‘다항 시간 알고리즘이 알려져 있지 않음’, ‘문제 사이의 다항 시간 환원’, ‘하나를 다항 시간에 풀면 모두 다항 시간에 해결’이 함께 제시되면 NP-완전을 떠올린다. 그다음 각 선지에 이미 알려진 다항 시간 알고리즘이 있는지 확인한다.

암기 포인트

SAT·부분 집합의 합·0-1 배낭 결정은 NP-완전, 모든 쌍 최단 경로는 P이다.

NoteBase