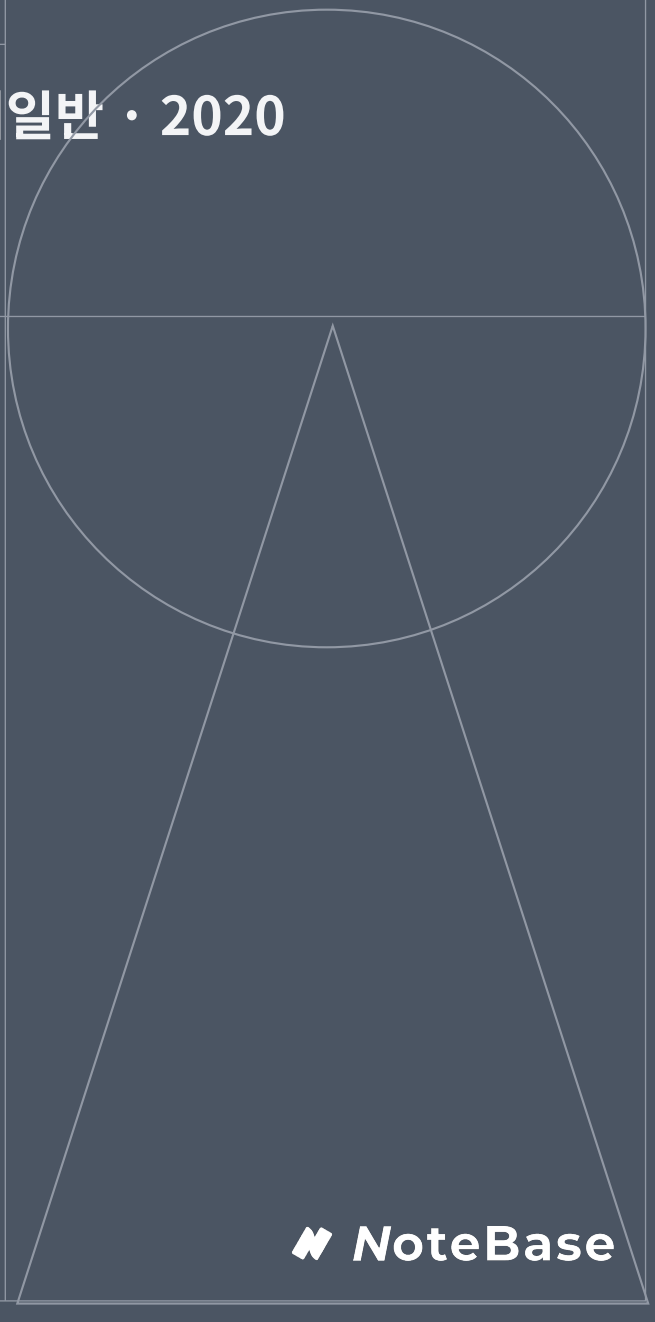
기출OX 해설집

공무원/국회직/9급 · 컴퓨터일반 · 2020



NoteBase

기출OX 해설집

공무원/국회직/9급 · 컴퓨터일반 · 2020

공무원/국회직/9급 · 컴퓨터일반 · 2020

1. 병렬처리 방식과 연관된 기술로 옳지 않은 것은?

① Pipelining
② Superscalar
③ VLIW(Very Long Instruction Word)
④ SMT(Simultaneous Multi-Threading)
⑤ Accumulator

정답 5번

출제 포인트

이 문제는 여러 명령이나 스레드의 실행을 겹치거나 동시에 처리하는 병렬처리 기술과 단순한 프로세서 구성 요소를 구별하는 문제이다.

핵심 개념

Pipelining은 명령어 처리 과정을 여러 단계로 나누어 서로 다른 명령어의 단계가 겹쳐 실행되게 하는 방식이다.
Superscalar는 한 클록 주기에 여러 명령어를 발행하여 복수의 실행 장치에서 병렬로 처리하는 방식이다.
VLIW는 컴파일러가 서로 독립적인 여러 연산을 하나의 긴 명령어에 묶어 동시에 실행하게 하는 방식이다.
SMT는 하나의 프로세서 코어가 여러 하드웨어 스레드의 명령어를 같은 시점에 실행 자원으로 보내도록 하는 방식이다.
Accumulator는 산술·논리 연산의 피연산자나 중간 결과를 누적하여 저장하는 레지스터이며, 병렬처리 방식이 아니다.

NoteBase

선지별 해설

① O — Pipelining은 여러 명령어의 서로 다른 처리 단계를 겹쳐 수행하므로 명령어 수준 병렬성과 연관된다.
② O — Superscalar는 한 클록 주기에 여러 명령어를 발행하고 복수의 실행 장치에서 처리하므로 병렬처리 기술이다.
③ O — VLIW는 컴파일러가 독립적인 연산들을 하나의 긴 명령어로 묶어 병렬 실행하도록 하므로 병렬처리 기술이다.
④ O — SMT는 한 코어에서 여러 스레드의 명령어를 동시에 발행하여 실행 자원의 활용도를 높이는 병렬처리 기술이다.
⑤ X — 틀린 부분은 Accumulator를 병렬처리 방식과 연관된 기술로 본 것이다. 올바르게는 Accumulator는 연산의 피연산자나 중간 결과를 누적 저장하는 레지스터 또는 그 레지스터를 중심으로 한 프로세서 구조이다.

오답 포인트

영문으로 제시된 용어가 모두 프로세서 기술처럼 보이지만, ⑤ Accumulator만 실행의 동시성을 높이는 기법이 아니라 값을 저장하는 레지스터의 명칭이다.

다시 나오면 이렇게 판단

용어가 여러 명령어·연산·스레드의 실행을 겹치거나 동시에 수행하게 하는지 확인한다. Pipelining, Superscalar, VLIW, SMT는 동시성을 높이지만 Accumulator는 연산 결과를 저장한다.

암기 포인트

파이프라인·슈퍼스칼라·VLIW·SMT는 병렬 실행, Accumulator는 누산 저장이다.

NoteBase

2. UDP(User Datagram Protocol)와 TCP(Transmission Control Protocol)에 대한 설명으로 옳지 않은 것은?

① TCP에는 흐름제어(flow control) 기능이 구현되어 있으나 UDP에는 흐름제어 기능이 구현되어 있지 않다.
② TCP에는 혼잡제어(congestion control) 기능이 구현되어 있으나 UDP에는 혼잡제어 기능이 구현되어 있지 않다.
③ TCP 계층에서는, TCP 하위계층에 의하여 전달된 패킷의 실제 전송 순서를 변경하지 않고 데이터 재조립을 수행한다.
④ TCP는 연결 지향적(connection oriented) 프로토콜이나 UDP는 연결 지향적 프로토콜이 아니다.
⑤ UDP는 실시간 데이터 스트리밍(streaming)에 더 적합하다.

정답 3번

출제 포인트

이 문제는 TCP와 UDP의 연결 방식, 흐름·혼잡 제어, 순서 보장 및 실시간 전송 특성을 비교하는 문제이다.

핵심 개념

TCP: 연결을 설정한 뒤 데이터를 신뢰성 있게 전달하는 전송 계층 프로토콜이다. 흐름 제어와 혼잡 제어를 수행하고, 순서 번호를 이용해 순서가 뒤바뀐 데이터를 올바른 순서로 재조립한다.
UDP: 연결 설정 없이 데이터그램을 독립적으로 전송하는 전송 계층 프로토콜이다. 흐름 제어·혼잡 제어·순서 보장·재전송을 제공하지 않는 대신 처리 부담과 지연이 작다.
핵심 대비: TCP는 연결 지향·신뢰성·순서 보장이 핵심이고, UDP는 비연결형·낮은 지연이 핵심이다.

선지별 해설

① O — TCP에는 수신 측의 처리 능력에 맞추어 송신량을 조절하는 흐름 제어 기능이 있지만, UDP에는 흐름 제어 기능이 없다.
② O — TCP는 네트워크의 혼잡 상태에 맞추어 전송량을 조절하는 혼잡 제어를 수행하지만, UDP 자체에는 혼잡 제어 기능이 없다.
③ X — 틀린 표현은 '실제 전송 순서를 변경하지 않고'이다. 하위 계층을 거쳐 도착한 데이터의 순서가 뒤바뀔 수 있으므로, TCP는 순서 번호를 기준으로 이를 원래 순서에 맞게 재배열하고 재조립한다.
④ O — TCP는 통신 전에 연결을 설정하는 연결 지향적 프로토콜이고, UDP는 연결 설정 과정이 없는 비연결형 프로토콜이다.
⑤ O — UDP는 연결 설정, 순서 보장, 재전송 등에 따른 지연이 작아 일부 데이터 손실보다 실시간성이 중요한 음성·영상 스트리밍에 적합하다.

NoteBase

오답 포인트

③은 '재조립한다'는 뒤쪽 표현만 보면 맞는 것처럼 보이지만, TCP의 재조립에는 순서 번호에 따른 순서 복원이 포함된다. 따라서 '전송 순서를 변경하지 않고'라는 표현이 TCP의 순서 보장 기능과 모순된다.

다시 나오면 이렇게 판단

연결 설정, 흐름·혼잡 제어, 재전송, 순서 보장은 TCP의 특성으로 판단하고, 비연결형, 낮은 지연, 실시간 전송은 UDP의 특성으로 판단한다. TCP가 순서가 뒤바뀐 데이터를 받으면 순서 번호를 기준으로 원래 순서를 복원한다.

암기 포인트

TCP는 연결·제어·재전송·순서 보장, UDP는 비연결·저지연·실시간이다.

NoteBase

3. 서버 가상화(virtualization)에서 필요로 하는 하이퍼바이저(hypervisor)에 대한 설명으로 옳은 것만을 <보기>에서 모두 고르면?

<보 기>

ㄱ. 하이퍼바이저는 운영체제(Operating System: OS) 없이 직접 하드웨어를 제어하거나 운영체제 상에서 응용 프로그램처럼 동작할 수 있다.

ㄴ. 하이퍼바이저 상에서 실행되는 게스트(guest) OS는 반드시 특정 하이퍼바이저 상에서 실행될 수 있도록 수정되어야 한다.

ㄷ. Type-2 하이퍼바이저는 특정 OS 상에서 응용 프로그램처럼 동작한다.

ㄹ. 전가상화(full-virtualization)와 반가상화(para-virtualization) 기술 모두에서 하이퍼바이저를 필요로 한다.

ㅁ. KVM(Kernel-based Virtual Machine)은 Type-1 하이퍼바이저로 분류될 수 있다.

① ㄱ, ㄴ, ㅁ
② ㄱ, ㄷ, ㄹ
③ ㄴ, ㄷ, ㅁ
④ ㄱ, ㄴ, ㄷ, ㄹ
⑤ ㄱ, ㄷ, ㄹ, ㅁ

정답 5번

출제 포인트

하이퍼바이저의 역할과 Type-1·Type-2의 실행 구조, 전가상화·반가상화에서의 게스트 OS 수정 여부를 구분하는 문제이다.

핵심 개념

하이퍼바이저는 물리 하드웨어 자원을 여러 가상 머신에 배분하고 각 가상 머신의 실행을 관리하는 가상화 계층이다.

Type-1 하이퍼바이저는 호스트 OS 없이 하드웨어 위에서 직접 동작하며, Type-2 하이퍼바이저는 호스트 OS 위에서 응용 프로그램처럼 동작한다.

전가상화는 수정하지 않은 게스트 OS를 실행할 수 있고, 반가상화는 게스트 OS가 하이퍼바이저와 효율적으로 통신하도록 수정된다는 점이 핵심 차이이다.

NoteBase

선지별 해설

① X
ㄱ과 ㅁ은 옳지만 ㄴ이 틀렸고, 옳은 ㄷ과 ㄹ이 빠져 있다. 틀린 구성은 'ㄱ, ㄴ, ㅁ'이며, 올바른 구성은 'ㄱ, ㄷ, ㄹ, ㅁ'이다.

② X
ㄱ·ㄷ·ㄹ은 옳지만, 옳은 ㅁ이 빠져 있다. 틀린 구성은 'ㄱ, ㄷ, ㄹ'이며, 올바른 구성은 'ㄱ, ㄷ, ㄹ, ㅁ'이다.

③ X
ㄷ과 ㅁ은 옳지만 ㄴ이 틀렸고, 옳은 ㄱ과 ㄹ이 빠져 있다. 틀린 구성은 'ㄴ, ㄷ, ㅁ'이며, 올바른 구성은 'ㄱ, ㄷ, ㄹ, ㅁ'이다.

④ X
ㄱ·ㄷ·ㄹ은 옳지만 ㄴ이 틀렸고, 옳은 ㅁ이 빠져 있다. 틀린 구성은 'ㄱ, ㄴ, ㄷ, ㄹ'이며, 올바른 구성은 'ㄱ, ㄷ, ㄹ, ㅁ'이다.

⑤ O
ㄱ은 Type-1과 Type-2 하이퍼바이저의 동작 방식을 모두 설명하므로 옳다. ㄴ은 틀리다. 게스트 OS의 수정 여부는 특정 하이퍼바이저 제품이 아니라 가상화 방식에 좌우되며, 전가상화에서는 수정하지 않은 게스트 OS를 실행할 수 있고 반가상화에서는 게스트 OS를 수정한다. ㄷ은 Type-2가 호스트 OS 위에서 응용 프로그램처럼 실행되므로 옳다. ㄹ은 전가상화와 반가상화 모두 가상 머신을 관리하는 하이퍼바이저를 사용하므로 옳다. ㅁ은 KVM이 Linux 커널을 하이퍼바이저로 동작하게 하여 일반적으로 Type-1으로 분류될 수 있으므로 옳다. 따라서 옳은 것은 ㄱ·ㄷ·ㄹ·ㅁ이다.

오답 포인트

ㄴ의 '특정 하이퍼바이저 상에서 실행될 수 있도록 수정'이라는 표현이 함정이다. 수정 여부를 결정하는 기준은 하이퍼바이저의 특정 제품명이 아니라 전가상화인지 반가상화인지이다. 또한 KVM은 Linux를 이용한다는 이유만으로 호스트 OS 위의 일반 프로그램인 Type-2로 보면 안 된다.

다시 나오면 이렇게 판단

먼저 Type-1은 하드웨어 직접 실행, Type-2는 호스트 OS 위 실행으로 나눈다. 다음으로 전가상화는 게스트 OS 무수정, 반가상화는 게스트 OS 수정으로 구분한다. 마지막으로 각 판단문의 참·거짓 집합과 조합 선지가 정확히 일치하는지 확인한다.

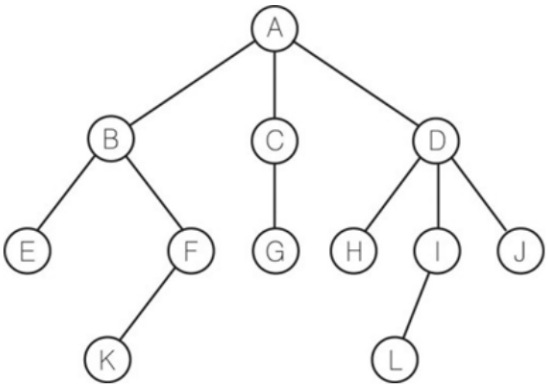
NoteBase

암기 포인트

Type-1은 하드웨어 직결, Type-2는 호스트 OS 위, 전가상화는 게스트 무수정, 반가상화는 게스트 수정이다.
KVM은 Type-1으로 분류할 수 있다.

NoteBase

4. 깊이 우선 탐색 알고리즘을 적용하여 아래의 트리를 탐색한다고 했을 때, 방문 순서를 나타낸 것으로 옳은 것은?



① A-B-C-D-E-F-G-H-I-J-K-L
② A-B-E-F-K-C-G-D-H-I-L-J
③ A-B-E-K-F-C-G-D-H-I-L-J
④ A-E-K-F-B-G-C-H-L-I-J-D
⑤ A-K-L-E-F-G-H-I-J-B-C-D

정답 2번

출제 포인트

이 문제는 트리에서 깊이 우선 탐색을 적용할 때, 한 갈래를 끝까지 탐색한 뒤 되돌아와 다음 갈래로 이동하는 방문 순서를 묻는 문제이다.

올바른 순서

A → B → E → F → K → C → G → D → H → I → L → J

A에서 왼쪽 자식부터 탐색하면 B의 서브트리를 먼저 모두 방문한다. B에서는 E를 방문한 뒤 F로 이동하고, F의 자식 K까지 방문한다. 이후 A로 되돌아와 C와 그 자식 G를 방문한다. 마지막으로 D의 서브트리에서 H, I와 그 자식 L, J의 순서로 방문한다.

왜 이 순서인가: 깊이 우선 탐색에서는 현재 노드의 아직 방문하지 않은 자식으로 계속 내려가며, 더 내려갈 곳이 없을 때에만 부모로 되돌아오기 때문이다.

NoteBase

선지별 해설

① X
A-B-C-D처럼 같은 깊이의 노드를 먼저 방문하는 순서로 시작하므로 깊이 우선 탐색이 아니다. 이는 위에서 아래로 각 깊이를 차례로 훑는 너비 우선 탐색 순서이다. 올바르게는 A-B-E-F-K-C-G-D-H-I-L-J이다.

② O
A에서 B로 내려간 뒤 E를 방문하고, 다시 F와 그 자식 K까지 탐색하여 B의 서브트리를 끝낸다. 이어 C-G를 방문하고 D-H-I-L-J를 방문하므로 깊이 우선 탐색 순서와 일치한다.

③ X
K는 F의 자식인데 F보다 먼저 방문한 A-B-E-K-F 부분이 잘못되었다. 자식 K에 도달하려면 부모 F를 먼저 방문해야 한다. 올바르게는 A-B-E-F-K-C-G-D-H-I-L-J이다.

④ X
E는 B의 자식인데 A 다음에 부모 B를 거치지 않고 E를 방문한 부분이 잘못되었다. 또한 D가 자신의 자식 H, I, J보다 뒤에 놓인 것도 부모 우선 방문 규칙에 어긋난다. 올바르게는 A-B-E-F-K-C-G-D-H-I-L-J이다.

⑤ X
K는 A의 직접 자식이 아니라 A-B-F-K 경로에 있는 노드이므로 A 다음에 바로 방문할 수 없다. L도 A-D-I-L 경로를 거쳐야 한다. 올바르게는 A-B-E-F-K-C-G-D-H-I-L-J이다.

오답 포인트

①은 같은 깊이의 노드를 먼저 나열한 너비 우선 탐색 순서여서 혼동하기 쉽다. 깊이 우선 탐색에서는 형제 노드 C로 넘어가기 전에 B 아래의 E, F, K를 모두 방문해야 한다. 또한 자식은 반드시 연결된 부모를 방문한 뒤에만 방문할 수 있으므로 ③의 K-F, ④의 A-E, ⑤의 A-K 같은 순서는 성립하지 않는다.

다시 나오면 이렇게 판단

시작 노드에서 정해진 방향의 자식을 하나 선택해 더 내려갈 수 없을 때까지 진행하고, 막히면 가장 가까운 분기점으로 되돌아가 아직 방문하지 않은 다음 자식을 탐색한다. 각 자식이 부모보다 먼저 나오지 않는지도 함께 확인한다.

암기 포인트

깊이 우선 탐색은 '한 갈래 끝까지, 막히면 되돌아가기'이다.

NoteBase

5. 다음 C 프로그램의 실행 결과로 옳은 것은?

```
int fun(int n) {
    printf("%d ", n);
    if(n < 3) return 1;
    return (fun(n - 3) + fun(n - 2));
}
int main() {
    int k;
    k = fun(5);
    printf("%d\n", k);
}
```

① 5 1 2 2 0
② 5 2 3 1 0
③ 5 2 3 0 1
④ 5 2 3 0 1 3
⑤ 5 2 3 0 3 1

정답 4번

출제 포인트

재귀 함수가 호출될 때마다 먼저 출력되는 인수의 순서와 각 호출의 반환값을 함께 추적하는 문제이다.

올바른 순서

공식 답 ④는 +의 왼쪽 피연산자를 먼저 평가한다고 보는 출제 관례를 전제로 한다.
1. fun(5)가 5를 출력한다.
2. 이 전제에서 fun(5)는 먼저 fun(2)를 호출하고, fun(2)는 2를 출력한 뒤 1을 반환한다.
3. 이어서 fun(3)이 3을 출력하고, 같은 전제에 따라 fun(0), fun(1)을 차례로 호출하므로 0, 1이 출력된다. 두 호출은 각각 1을 반환하므로 fun(3)은 2를 반환한다.
4. fun(5)는 1 + 2의 결과인 3을 반환하고, main의 마지막 printf가 3을 출력한다.
따라서 출제자가 전제한 전체 출력 순서는 5 2 3 0 1 3이므로 공식 답은 ④이다. 앞의 다섯 수는 각 fun 호출에서 출력되고, 마지막 3은 모든 재귀 호출이 끝난 뒤 main에서 반환값 k를 출력한 것이다.
다만 표준 C에서 fun(n - 3) + fun(n - 2)의 두 피연산자 평가 순서는 지정되어 있지 않다. 따라서 실제 표준만 적용하면 fun(2)와 fun(3), 그리고 fun(3) 안의 fun(0)과 fun(1)의 호출 순서가 달라질 수 있어 출력 순서가 하나로 확정되지 않는다.

NoteBase

선지별 해설

① X — 최종 반환값은 항상 3이므로 마지막에 main이 출력하는 값도 3이어야 한다. 또한 출제자가 전제한 왼쪽 피연산자 우선 평가에서는 5 다음에 fun(2)의 인수 2가 출력된다.
② X — 마지막 값이 main에서 출력하는 최종 반환값 3이 아니므로 옳지 않다.
③ X — 출제자가 전제한 재귀 호출 과정의 출력 5 2 3 0 1까지만 나타내고, main의 printf가 k의 값 3을 한 번 더 출력하는 부분을 빠뜨렸다.
④ O — 왼쪽 피연산자를 먼저 평가한다는 출제 관례에 따르면 fun의 호출 인수는 5, 2, 3, 0, 1의 순서로 출력되고, 모든 계산이 끝난 뒤 main에서 k = 3이 출력되므로 5 2 3 0 1 3이 된다. 따라서 공식 답은 ④이다. 다만 표준 C 자체는 +의 두 피연산자 평가 순서를 지정하지 않는다.
⑤ X — 마지막 값이 main에서 출력하는 최종 반환값 3이 아니므로 옳지 않다. 출제 관례에 따르면 fun(3)의 하위 호출에서는 0, 1이 출력되고, 최종 3은 전체 재귀 계산이 끝난 뒤 main에서 출력된다.

오답 포인트

함수 호출 직후의 printf와 함수가 반환한 뒤 main에서 실행되는 printf를 혼동하기 쉽다. 특히 출제자가 전제한 왼쪽 피연산자 우선 평가에서는 ③이 재귀 호출 중 출력되는 5 2 3 0 1까지만 적고 최종 반환값 3의 출력을 빠뜨린 선지이다. 표준 C에서는 +의 피연산자 평가 순서가 지정되어 있지 않다는 점과 시험이 채택한 평가 관례를 구분해야 한다.

다시 나오면 이렇게 판단

재귀 호출 트리를 그린 뒤 각 노드에서 먼저 인수를 출력하고, 기저 조건의 반환값부터 거꾸로 합산한다.
마지막에는 main에 별도의 출력문이 있는지 확인하여 재귀 과정의 출력과 최종 결과 출력을 구분한다.

암기 포인트

호출할 때 인수를 출력하고, 복귀할 때 반환값을 계산하며, main의 출력은 맨 마지막에 붙는다.

NoteBase

6. I/O장치에 대한 설명으로 옳지 않은 것은?

① 채널을 이용한 입출력 방식은 데이터의 고속성을 위해 CPU의 계속적인 간섭 없이 직접 I/O장치와 기억장치 사이에 자료를 주고받는다.

② DMA(Direct Memory Access)를 이용한 입출력 방식은 기억장치와 입출력 모듈 간의 데이터 전송을 DMA 제어기가 처리하고 CPU가 그 동작을 관리한다.

③ I/O장치는 발생하는 이벤트를 CPU에 알리기 위해 주로 인터럽트를 발생시킨다.

④ I/O장치는 주로 인간 혹은 컴퓨터 외부와의 인터페이스(interface) 역할을 한다.

⑤ Memory-mapped I/O장치는 메모리접근 명령어를 사용해 관리한다.

정답 2번

출제 포인트

이 문제는 채널 방식, DMA, 인터럽트, 인터페이스, Memory-mapped I/O의 역할과 CPU 관여 정도를 구분하는 문제이다.

핵심 개념

채널 방식: 채널이라는 전용 입출력 프로세서가 입출력을 맡아 CPU의 지속적인 개입을 줄이는 방식이다.
DMA: DMA 제어기가 기억장치와 입출력 모듈 사이의 데이터 블록 전송을 직접 수행하는 방식이다. CPU는 전송을 설정·개시하고 완료 통보를 받지만, 전송 동작을 계속 관리하지 않는다.
인터럽트: 입출력장치가 사건 발생이나 작업 완료 등을 CPU에 알리는 신호이다.
Memory-mapped I/O: 입출력장치의 레지스터를 기억장치 주소 공간에 배치하여 일반 메모리 접근 명령어로 제어하는 방식이다.

NoteBase

선지별 해설

① O — 채널 방식에서는 채널이 입출력 처리를 전담하므로 CPU가 입출력장치와 기억장치 사이의 자료 전송에 계속 간섭하지 않는다.
② X — 틀린 표현은 ‘CPU가 그 동작을 관리한다’이다. 올바르게는 DMA 제어기가 기억장치와 입출력 모듈 사이의 데이터 전송을 관리하고, CPU는 전송을 설정·개시한 뒤 완료 시 통보를 받는다.
③ O — 입출력장치는 입력 도착, 오류 발생, 작업 완료 등의 이벤트를 CPU에 알리기 위해 주로 인터럽트를 발생시킨다.
④ O — 입출력장치는 키보드·모니터처럼 인간과 컴퓨터 사이를 연결하거나, 통신장치처럼 컴퓨터와 외부 시스템 사이를 연결하는 인터페이스 역할을 한다.
⑤ O — Memory-mapped I/O는 입출력장치의 레지스터를 메모리 주소 공간에 대응시키므로 일반 메모리 접근 명령어를 사용하여 관리할 수 있다.

오답 포인트

②는 CPU가 DMA 전송을 시작하도록 설정한다는 사실을 이용해 전송 과정까지 CPU가 관리하는 것처럼 바꾼 선지이다. DMA의 목적은 실제 데이터 전송을 DMA 제어기에 맡겨 CPU의 부담을 줄이는 데 있다.

다시 나오면 이렇게 판단

입출력 방식은 ‘실제 데이터 전송을 누가 담당하는가’를 기준으로 판단한다. DMA에서는 DMA 제어기가 기억장치와 입출력 모듈 사이의 전송을 담당하고, CPU는 주로 초기 설정과 완료 확인에 관여한다.

암기 포인트

채널은 입출력 전담, DMA는 직접 전송, 인터럽트는 CPU에 통보, Memory-mapped I/O는 메모리 명령어 사용이다.

NoteBase

7. 다음과 같이 중위 표기법(infix notation)으로 된 식의 후위 표기법(postfix notation)으로 옳은 것은?

(7 + 6 / 2) / 2 + 9 * 4 / 3

① 6 / 2 + 7 / 2 9 * 4 / 3 +
② 6 2 / 7 + 2 / 9 4 3 * / +
③ 7 6 / 2 + 2 / 9 4 * 3 / +
④ 7 6 2 / + 2 / 9 4 3 / + *
⑤ 7 6 2 / + 2 / 9 4 * 3 / +

정답 5번

출제 포인트

이 문제는 연산자의 우선순위와 결합 방향을 지키면서 중위 표기식을 후위 표기식으로 바꾸는 과정을 묻는 문제이다.

올바른 순서

1. 괄호 안의 6 / 2를 먼저 바꾸면 6 2 /가 된다.
2. 여기에 7과 +를 결합하면 괄호 전체는 7 6 2 / +가 된다.
3. 괄호 전체를 2로 나누면 7 6 2 / + 2 /가 된다.
4. 오른쪽의 9 * 4 / 3은 *와 /의 우선순위가 같으므로 왼쪽부터 묶어 9 4 * 3 /가 된다.
5. 두 결과를 마지막 +로 결합하면 7 6 2 / + 2 / 9 4 * 3 / +가 된다.

왜 이 순서: 후위 표기법에서는 이항 연산자가 두 피연산자의 뒤에 놓이며, 괄호와 연산자 우선순위에 따라 먼저 계산될 부분이 먼저 완성되어야 하기 때문이다.

선지별 해설

① X — 6 바로 다음에 /가 나와 /의 두 번째 피연산자가 준비되지 않았으므로 올바른 후위 표기식이 아니다. 올바르게는 6 2 /처럼 두 피연산자 뒤에 /를 써야 하며, 전체 식은 7 6 2 / + 2 / 9 4 * 3 / +가 된다.
② X — 앞부분 6 2 / 7 +는 6 / 2와 7의 위치를 바꾸었고, 9 4 3 * /는 9 / (4 * 3)을 나타낸다. 올바르게는 각각 7 6 2 / +와 9 4 * 3 /이며, 전체 식은 7 6 2 / + 2 / 9 4 * 3 / +가 된다.
③ X — 7 6 / 2 +는 7 / 6 + 2를 나타내므로 괄호 안의 7 + 6 / 2와 다르다. 올바르게는 괄호 안을 7 6 2 / +로 쓰고, 전체 식은 7 6 2 / + 2 / 9 4 * 3 / +가 된다.
④ X — 마지막의 *는 왼쪽 전체와 오른쪽 결과를 곱하게 하며, 9 4 3 / +도 9 + 4 / 3을 나타낸다. 올바르게는 오른쪽을 9 4 * 3 /로 만든 뒤 마지막 연산자를 +로 두어 7 6 2 / + 2 / 9 4 * 3 / +로 써야 한다.
⑤ O — 괄호 안은 7 6 2 / +, 그 결과를 2로 나눈 부분은 7 6 2 / + 2 /, 오른쪽 곱셈·나눗셈 부분은 9 4 * 3 /로 정확히 표현되며 마지막 +가 두 부분을 결합한다.

NoteBase

오답 포인트

핵심 함정은 9 * 4 / 3에서 *와 /의 우선순위가 같다는 점이다. 같은 우선순위의 연산은 왼쪽부터 처리하므로 (9 * 4) / 3이며, 후위 표기는 9 4 * 3 /가 된다. 또한 후위 표기에서는 연산자를 반드시 해당 두 피연산자 뒤에 둔다.

다시 나오면 이렇게 판단

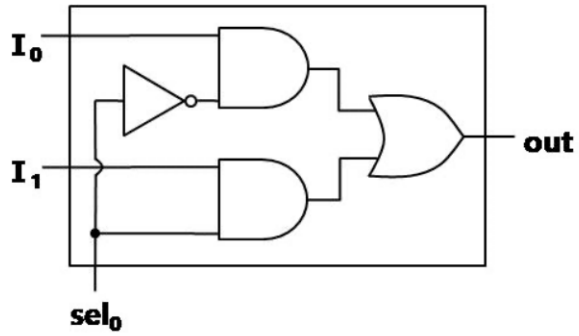
괄호 안을 먼저 후위 표기로 완성하고, 같은 우선순위의 연산은 결합 방향에 따라 처리한 뒤, 마지막에 식 전체를 연결하는 연산자를 붙인다. 각 연산자를 읽는 시점에 앞쪽에 사용할 피연산자 두 개가 있는지도 확인한다.

암기 포인트

후위 표기법은 피연산자 두 개 뒤에 연산자 하나를 둔다. 괄호 먼저, 곱셈·나눗셈 먼저, 같은 우선순위는 왼쪽부터 처리한다.

NoteBase

8. 다음 논리회로의 기능에 해당하는 것은?



① 멀티플렉서
② 플립플롭
③ 덧셈기
④ 곱셈기
⑤ 감산기

정답 1번

출제 포인트

선택 신호에 따라 여러 입력 중 하나를 출력으로 전달하는 멀티플렉서의 게이트 구성을 판별하는 문제이다. 위쪽 AND 게이트에는 I_0와 반전된 sel_0가, 아래쪽 AND 게이트에는 I_1과 sel_0가 입력되고 두 결과가 OR 게이트에서 합쳐진다.

선지별 해설

① O — 출력식은 $out = I_0\overline{sel_0} + I_1 sel_0$이다. $sel_0 = 0$이면 I_0, $sel_0 = 1$이면 I_1이 출력되므로 2:1 멀티플렉서이다.
② X — 플립플롭은 클록에 따라 1비트 상태를 저장하는 순차논리회로이다. 이 회로에는 클록이나 피드백 저장 구조가 없으므로, 올바르게는 멀티플렉서이다.
③ X — 덧셈기는 입력 비트의 합과 자리올림을 생성한다. 이 회로는 합과 자리올림을 구하지 않고 선택 신호에 따라 입력 하나를 전달하므로, 올바르게는 멀티플렉서이다.
④ X — 곱셈기는 두 수의 곱을 계산하는 산술회로이다. 이 회로에는 곱셈을 위한 부분곱 생성·가산 구조가 없으므로, 올바르게는 멀티플렉서이다.
⑤ X — 감산기는 두 수의 차와 빌림을 생성한다. 이 회로는 차와 빌림을 구하지 않고 입력을 선택하므로, 올바르게는 멀티플렉서이다.

오답 포인트

AND·OR 게이트가 있다는 이유만으로 덧셈기 같은 산술회로로 판단하기 쉽다. 핵심 단서는 하나의 선택 신호가 위 경로에는 반전되어 들어가고 아래 경로에는 그대로 들어가 두 입력 중 한 경로만 활성화한다는 점이다.

NoteBase

다시 나오면 이렇게 판단

선택 신호 S와 그 반전값 $\overline{S}$가 각각 입력 경로의 AND 게이트를 제어하고 그 결과를 OR로 합치면 $Y = I_0\overline{S} + I_1S$ 형태의 2:1 멀티플렉서로 판단한다.

암기 포인트

선택값이 0이면 I_0, 1이면 I_1을 통과시키는 회로가 2:1 멀티플렉서이다.

NoteBase

9. GPGPU(General-Purpose computing on Graphics Processing Units) 기술에 대한 설명으로 옳지 않은 것은?

① GPU에서 그래픽 연산 이외의 목적을 가진 프로그램을 실행할 수 있도록 해주는 기술을 지칭한다.

② CPU 기기에서의 실행을 위해 컴파일 된 모든 응용 프로그램은 GPGPU 기기에서 실행될 수 있다.

③ 하나의 GPGPU 기기에는 많은 수의 단순 ALU(Arithmetic Logic Unit)가 있어 높은 수준의 병렬처리가 가능하고, 이로 인해 일반 CPU에 비하여 프로그램의 병렬 처리속도가 높아진다.

④ OpenCL(Open Computing Language)을 이용하여 프로그래밍할 경우 다양한 제조사의 GPGPU 기기에서 실행 가능한 프로그램을 작성할 수 있다.

⑤ 병렬처리가 불가능한 어떤 프로그램을 GPGPU에서 실행하려 한다면 성능향상을 기대하기 어렵다.

정답 2번

출제 포인트

GPGPU의 목적과 병렬 처리 구조, CPU용 프로그램과 GPU용 프로그램의 실행 방식, OpenCL의 이식성을 구별하는 문제이다.

핵심 개념

GPGPU는 GPU의 다수 연산 자원을 그래픽 이외의 범용 계산에 활용하는 기술이다.
CPU는 복잡한 제어와 순차 처리에 강하고, GPU는 많은 단순 연산 장치로 같은 종류의 계산을 대량 병렬 처리하는 데 강하다.
CPU용으로 컴파일된 프로그램이 GPU에서 자동으로 실행되는 것은 아니다. GPU용 코드와 병렬화 작업, 해당 실행 환경에 맞는 컴파일이 필요하다.
OpenCL은 여러 제조사의 CPU·GPU 등에서 병렬 프로그램을 실행할 수 있도록 하는 개방형 표준이다.

NoteBase

선지별 해설

① O — GPGPU는 GPU를 그래픽 렌더링뿐 아니라 과학 계산, 데이터 처리 등 범용 연산에 활용하는 기술이므로 옳다.
② X — 틀린 부분은 'CPU 기기에서의 실행을 위해 컴파일된 모든 응용 프로그램은 GPGPU 기기에서 실행될 수 있다'이다. CPU용 실행 파일이 GPU에서 그대로 실행되는 것은 아니며, 올바르게는 GPU의 병렬 처리 모델에 맞게 작성하거나 변환한 코드를 해당 GPGPU 환경에 맞게 컴파일해야 한다.
③ O — GPU에는 단순 연산을 담당하는 다수의 연산 장치가 있어 동일하거나 유사한 계산을 동시에 수행할 수 있으므로, 병렬화가 가능한 작업에서는 일반 CPU보다 높은 처리 성능을 낼 수 있다.
④ O — OpenCL은 특정 제조사에 종속되지 않는 병렬 프로그래밍 표준이므로, 이를 이용하면 다양한 제조사의 GPGPU 장치를 대상으로 실행 가능한 프로그램을 작성할 수 있다.
⑤ O — 병렬화할 수 없는 순차적 프로그램은 GPU의 다수 연산 장치를 충분히 활용하지 못하므로 GPGPU에서 의미 있는 성능 향상을 기대하기 어렵다.

오답 포인트

②는 CPU와 GPU가 모두 프로그램을 실행하는 장치라는 점을 이용해 'CPU용 프로그램이 GPU에서도 자동으로 실행된다'고 바꾼 함정이다. GPGPU는 기존 CPU 실행 파일의 단순 호환을 뜻하지 않고, GPU에 맞는 병렬 코드와 실행 환경을 요구한다.

다시 나오면 이렇게 판단

GPGPU 선지는 '범용 연산 활용', '대량 병렬 처리', 'GPU용 코드·컴파일 필요', 'OpenCL의 장치 이식성'의 네 기준으로 판단한다. CPU용 프로그램이 수정 없이 모두 실행된다는 표현은 틀린 것으로 판단한다.

암기 포인트

GPGPU는 GPU로 범용 병렬 연산을 수행하는 기술이며, CPU용 프로그램의 자동 실행 기술이 아니다.

NoteBase

10. 데이터베이스의 특성에 대한 설명으로 옳은 것만을 <보기>에서 모두 고르면?

<보 기>

ㄱ. 실시간 접근성: 데이터의 검색이나 조작을 요구하는 수시적이고 비정형적인 질의에 대하여 즉시 응답할 수 있어야 한다.

ㄴ. 계속적인 변화: 데이터베이스의 상태는 정적이 아니고 동적이므로 현재의 정확한 데이터를 유지해야 한다.

ㄷ. 동시공유: 데이터베이스는 동시에 여러 사용자가 접근할 수 있어야 한다.

ㄹ. 주소에 의한 참조: 데이터베이스 내에 있는 데이터 레코드들은 주소에 의해 참조된다.

① ㄱ, ㄴ
② ㄴ, ㄷ
③ ㄷ, ㄹ
④ ㄱ, ㄴ, ㄷ
⑤ ㄴ, ㄷ, ㄹ

정답 4번

출제 포인트

데이터베이스의 기본 특성인 실시간 접근성, 계속적인 변화, 동시 공유, 내용에 의한 참조를 구별하는 문제이다.

핵심 개념

실시간 접근성: 수시적이고 비정형적인 질의에도 사용자가 요구할 때 즉시 응답할 수 있는 성질이다.
계속적인 변화: 삽입·삭제·갱신을 통해 데이터베이스의 상태가 계속 변하면서도 현재의 정확한 데이터를 유지하는 성질이다.
동시 공유: 여러 사용자가 같은 데이터베이스에 동시에 접근하여 이용할 수 있는 성질이다.
내용에 의한 참조: 데이터의 물리적 주소가 아니라 데이터의 값이나 내용, 즉 질의 조건으로 원하는 레코드를 찾는 성질이다.

NoteBase

선지별 해설

① X — ㄱ은 옳지만 ㄴ도 옳으므로 옳은 항목을 모두 고른 조합이 아니다. 틀린 부분은 ㄴ을 제외한 것이며, 올바른 조합은 ㄱ, ㄴ, ㄷ이다.
② X — ㄴ과 ㄷ은 모두 옳지만 ㄱ도 옳으므로 불완전한 조합이다. 틀린 부분은 ㄱ을 제외한 것이며, 올바른 조합은 ㄱ, ㄴ, ㄷ이다.
③ X — ㄷ은 옳지만 ㄹ은 틀리다. 데이터 레코드는 주소가 아니라 내용이나 값에 의해 참조되므로, 올바른 조합은 ㄱ, ㄴ, ㄷ이다.
④ O — ㄱ의 실시간 접근성, ㄴ의 계속적인 변화, ㄷ의 동시 공유는 모두 데이터베이스의 기본 특성이다. ㄹ은 제외되므로 ㄱ, ㄴ, ㄷ의 조합이 정확하다.
⑤ X — ㄴ과 ㄷ은 옳지만 ㄹ은 틀리고 ㄱ은 빠져 있다. '주소에 의해 참조된다'는 표현은 '내용이나 값에 의해 참조된다'로 고쳐야 하며, 올바른 조합은 ㄱ, ㄴ, ㄷ이다.

오답 포인트

ㄹ의 '주소'가 함정이다. 데이터베이스 이용자는 저장된 물리적 위치를 직접 지정하는 것이 아니라 데이터의 내용이나 조건을 제시하여 원하는 레코드를 참조한다. 또한 '모두 고르면'이라는 조건 때문에 옳은 보기 일부만 포함한 ①과 ②도 정답이 아니다.

다시 나오면 이렇게 판단

데이터베이스의 네 특성을 '즉시 응답, 지속적 갱신, 여러 사용자, 내용으로 검색'에 대응시킨다. 주소·위치로 참조한다고 하면 내용에 의한 참조와 반대이므로 틀린 설명으로 판단한다.

암기 포인트

데이터베이스는 실시간 접근, 계속 변화, 동시 공유, 내용 참조이다. 주소 참조가 아니라 내용 참조이다.

NoteBase

11. 스택(stack)은 자료구조에서 말하는 구조체 외에도, 메모리에 지정된 특정 영역을 말하며 프로그램 실행에서 항상 사용된다. 이러한 스택에 대한 설명으로 옳지 않은 것은?

① CPU마다 스택포인터 용도로 사용하는 레지스터가 있다.
② 스택은 LIFO(Last-In First-Out)구조를 가지고 있다.
③ 버퍼 오퍼플로우(buffer overflow) 보안공격에 이용되기도 한다.
④ 운영체제의 문맥교환(context switching)을 할 때 사용된다.
⑤ CPU 레지스터의 내용을 스택에 저장할 수 없다.

정답 5번

출제 포인트

스택의 LIFO 동작 방식과 CPU의 스택 포인터, 버퍼 오버플로우, 문맥 교환 및 레지스터 저장에서 스택이 수행하는 역할을 묻는 문제이다.

핵심 개념

스택은 나중에 저장한 항목을 먼저 꺼내는 LIFO 방식의 메모리 영역이다. CPU는 스택 포인터로 스택의 현재 위치를 관리하며, 함수 호출 정보나 지역 데이터뿐 아니라 레지스터 값도 스택에 저장했다가 복원할 수 있다.

선지별 해설

① O — CPU에는 스택의 맨 위 또는 현재 위치를 가리키는 스택 포인터 용도의 레지스터가 있다.
② O — 스택은 가장 나중에 들어간 데이터가 가장 먼저 나오는 LIFO 구조이다.
③ O — 스택에 배치된 버퍼의 경계를 넘어 데이터를 쓰면 반환 주소 등의 인접 정보를 훼손할 수 있어 버퍼 오버플로우 공격에 이용되기도 한다.
④ O — 문맥 교환 때 운영체제는 실행을 재개할 수 있도록 레지스터 등 CPU 문맥을 보존해야 하며, 이 과정에서 커널 스택 등의 메모리 영역이 사용될 수 있다.
⑤ X — 틀린 표현은 ‘CPU 레지스터의 내용을 스택에 저장할 수 없다’이다. 올바르게는 CPU 레지스터의 내용을 스택에 저장할 수 있으며, 이후 다시 레지스터로 복원할 수도 있다.

오답 포인트

⑤의 ‘저장할 수 없다’가 함정이다. 스택은 함수의 지역 데이터만 저장하는 공간이 아니며, 레지스터 보존과 복원에도 사용된다. ④도 문맥 정보가 반드시 스택에만 저장된다는 뜻이 아니라 문맥 교환 과정에서 스택이 사용될 수 있다는 진술이므로 옳다.

NoteBase

다시 나오면 이렇게 판단

스택 관련 선지는 LIFO, 스택 포인터, 함수 호출 정보, 레지스터 값의 저장·복원이라는 네 가지 기능을 기준으로 판단한다. '레지스터 값을 스택에 저장할 수 없다'처럼 저장·복원 가능성을 부정하면 틀린 진술이다.

암기 포인트

스택은 LIFO, SP로 위치 관리, 레지스터는 저장 후 복원 가능이다.

NoteBase

12. 관계형 데이터 모델(relational data model)에서 릴레이션에 대한 설명으로 옳지 않은 것은?

① 릴레이션을 구성하는 속성 값은 서브속성으로 다시 세분화할 수 있다.
② 데이터베이스 스키마(database schema)는 정적이고, 데이터베이스 인스턴스(database instance)는 동적이다.
③ 릴레이션 안에는 똑같은 튜플이 존재할 수 없다.
④ 릴레이션의 속성 사이에 순서는 의미가 없다.
⑤ 릴레이션의 속성의 명칭은 유일하나, 속성을 구성하는 값은 같을 수 있다.

정답 1번

출제 포인트

관계형 데이터 모델에서 릴레이션을 이루는 속성값의 원자성, 스키마와 인스턴스의 차이, 튜플의 유일성, 속성 순서와 이름의 성질을 구별하는 문제이다.

핵심 개념

릴레이션: 같은 속성 구조를 가진 튜플들의 집합이다. 관계형 모델에서는 각 속성값을 더 나누지 않는 하나의 원자값으로 취급한다.
데이터베이스 스키마: 데이터베이스의 구조와 제약조건을 정의한 것으로 비교적 정적이다.
데이터베이스 인스턴스: 특정 시점에 저장된 실제 데이터의 상태로, 삽입·삭제·갱신에 따라 동적으로 변한다.
튜플과 속성: 릴레이션은 집합이므로 같은 튜플이 중복되지 않으며, 속성의 배열 순서는 의미가 없다. 한 릴레이션 안에서 속성 이름은 서로 구별되어야 하지만 속성값은 같을 수 있다.

선지별 해설

① X - 틀린 표현은 '속성 값은 서브속성으로 다시 세분화할 수 있다'이다. 관계형 모델의 속성값은 하나의 원자값으로 취급하므로 더 작은 서브속성으로 분해되는 구조를 갖지 않는다. 올바르게는 '릴레이션을 구성하는 각 속성값은 원자값이어야 한다'이다.
② O - 데이터베이스 스키마는 구조에 관한 정의이므로 비교적 정적이고, 데이터베이스 인스턴스는 특정 시점의 실제 데이터이므로 데이터 변경에 따라 동적이다.
③ O - 릴레이션은 튜플의 집합이므로 완전히 동일한 튜플이 중복되어 존재할 수 없다.
④ O - 릴레이션에서 속성은 이름으로 식별하므로 속성을 나열한 순서 자체에는 의미가 없다.
⑤ O - 한 릴레이션의 속성 이름은 각 속성을 구별하기 위해 유일해야 하지만, 서로 다른 속성에 저장된 값은 같을 수 있다.

NoteBase

오답 포인트

①의 ‘세분화할 수 있다’는 표현이 함정이다. 현실의 값은 의미상 나눌 수 있어 보여도 관계형 모델의 한 속성값은 해당 릴레이션 안에서 더 분해하지 않는 원자값으로 취급한다. 또한 ⑤에서는 이름의 유일성과 값의 유일성을 혼동하지 않아야 한다.

다시 나오면 이렇게 판단

관계형 모델의 기본 성질을 ‘값은 원자적, 튜플은 중복 불가, 행과 열의 순서는 무의미, 속성 이름은 유일, 속성값은 중복 가능’으로 나누어 각 선지의 표현과 대조한다. 스키마는 구조, 인스턴스는 특정 시점의 데이터라는 기준도 함께 적용한다.

암기 포인트

값은 원자적, 튜플은 유일, 순서는 무의미하다. 속성 이름은 달라야 하지만 값은 같아도 된다.

NoteBase

13. IPv4 주소체계 기반의 어떤 네트워크 상에서 두 컴퓨터 A, B가 각각 192.168.0.1과 192.168.0.65의 주소를 사용할 때, 이 두 컴퓨터가 서로 다른 서브넷(subnet) 상에 존재하기 위해 사용해야 하는 서브넷 마스크(subnet mask)로 옳은 것은?

① 0.0.0.0
② 255.255.255.0
③ 255.255.255.192
④ 255.255.255.128
⑤ 255.255.128.0

정답 3번

출제 포인트

두 IPv4 주소에 서브넷 마스크를 비트 AND 연산하여 얻은 네트워크 주소가 서로 달라지는 마스크를 찾는 문제이다.

조건 정리

주어진 주소는 192.168.0.1과 192.168.0.65이다. 두 주소는 마지막 옥텟만 다르며, 1은 0~63 구간에 있고 65는 64~127 구간에 있다. 두 컴퓨터가 다른 서브넷에 있으려면 두 주소의 네트워크 주소가 달라야 한다.

사용할 공식

공식은 Network = IP \& Mask이다. 이 공식을 쓰는 이유는 서브넷 소속이 IP 주소와 서브넷 마스크의 비트 AND 결과인 네트워크 주소로 결정되기 때문이다.

계산 과정

255.255.255.192는 /26 마스크이며 마지막 옥텟은 이진수 11000000이다. 따라서 블록 크기는 256-192=64이고 서브넷 범위는 0~63, 64~127, 128~191, 192~255로 나뉜다. 192.168.0.1은 192.168.0.0/26에, 192.168.0.65는 192.168.0.64/26에 속하므로 두 네트워크 주소가 서로 다르다.

NoteBase

선지 판단

① X — 0.0.0.0은 /0 마스크이므로 두 주소 모두 같은 네트워크 0.0.0.0으로 계산된다. 서로 다른 서브넷으로 나누려면 255.255.255.192(/26)를 사용해야 한다.
② X — 255.255.255.0은 /24 마스크이므로 두 주소의 네트워크 주소가 모두 192.168.0.0이다. 서로 다르게 나누려면 255.255.255.192(/26)를 사용해야 한다.
③ O — 255.255.255.192는 /26 마스크로, 192.168.0.1은 192.168.0.0/26에 속하고 192.168.0.65는 192.168.0.64/26에 속한다.
④ X — 255.255.255.128은 /25 마스크로 0~127이 하나의 서브넷이므로 1과 65가 모두 192.168.0.0/25에 속한다. 서로 다르게 나누려면 경계를 64에 두는 255.255.255.192(/26)를 사용해야 한다.
⑤ X — 255.255.128.0은 /17 마스크이므로 두 주소의 네트워크 주소가 모두 192.168.0.0이다. 서로 다르게 나누려면 255.255.255.192(/26)를 사용해야 한다.

오답 포인트

④의 255.255.255.128은 주소 128부터 서브넷을 나누므로 마지막 옥텟이 1과 65인 두 주소를 분리하지 못한다. 필요한 경계는 64이며, 이 경계를 만드는 마지막 옥텟 값이 192이다.

다시 나오면 이렇게 판단

마스크의 마지막 옥텟이 128, 192, 224처럼 제시되면 256에서 그 값을 빼 블록 크기를 구한다. 각 IP가 어느 블록에 속하는지 확인하고, 블록이 다르면 서로 다른 서브넷으로 판단한다.

암기 포인트

/26 = 255.255.255.192 = 블록 크기 64이다. 따라서 마지막 옥텟의 경계는 0, 64, 128, 192이다.

NoteBase

14. 스래싱(thrashing)에 대한 설명으로 옳은 것은?

① 두 개 이상의 작업이 서로 상대방의 작업이 끝나기만을 기다리고 있기 때문에 결과적으로 아무것도 완료되지 못하는 현상을 의미한다.

② CPU 버스트가 짧은 프로세스에게 우선순위를 항상 부여한다면, 상대적으로 CPU 버스트가 긴 프로세스가 계속해서 지연되는 것을 의미한다.

③ CPU 버스트가 긴 프로세스가 CPU를 양보할 때까지 다른 모든 프로세스들이 기다리는 것을 의미한다.

④ 페이지 부재가 너무 자주 일어나 프로세스가 실행에 보내는 시간보다 페이지 교체에 더 많은 시간을 소비하는 현상을 의미한다.

⑤ CPU가 프로그램을 실행하고 있을 때, 입출력 하드웨어 등의 장치나 예외 상황이 발생하여 처리가 필요한 경우 CPU에게 알려 처리할 수 있도록 하는 것을 의미한다.

정답 4번

출제 포인트

스래싱의 정의를 교착 상태, 기아 상태, 호위 효과, 인터럽트와 구별하는 문제이다.

핵심 개념

스래싱(thrashing): 페이지 부재가 지나치게 자주 발생하여 프로세스가 실제 명령을 실행하는 시간보다 페이지를 교체하는 데 더 많은 시간을 쓰는 현상이다.
교착 상태는 작업들이 서로 필요한 자원을 기다려 모두 진행하지 못하는 현상이고, 기아 상태는 특정 프로세스가 자원을 계속 배정받지 못하는 현상이다. 호위 효과는 실행 시간이 긴 프로세스 때문에 짧은 프로세스들이 함께 오래 기다리는 현상이며, 인터럽트는 처리할 사건이 발생했음을 CPU에 알리는 신호이다.

NoteBase

선지별 해설

① X
틀린 표현은 이를 스래싱의 의미로 본 부분이다. 서로 상대 작업의 종료나 자원 해제를 기다려 아무 작업도 완료되지 못하는 현상은 교착 상태(deadlock)이다.

② X
틀린 표현은 이를 스래싱의 의미로 본 부분이다. 짧은 CPU 버스트를 가진 프로세스에 계속 우선순위를 주어 긴 프로세스가 무기한 지연되는 현상은 기아 상태(starvation)이다.

③ X
틀린 표현은 이를 스래싱의 의미로 본 부분이다. CPU 버스트가 긴 프로세스가 CPU를 오래 점유하여 다른 프로세스들이 함께 기다리는 현상은 호위 효과(convoy effect)이다.

④ O
페이지 부재가 지나치게 자주 발생하여 실제 실행보다 페이지 교체에 더 많은 시간을 소비하는 현상은 스래싱의 정확한 설명이다.

⑤ X
틀린 표현은 이를 스래싱의 의미로 본 부분이다. 입출력 장치나 예외 상황 등이 처리가 필요함을 CPU에 알려 현재 실행 흐름을 잠시 중단하고 처리하게 하는 것은 인터럽트(interrupt)이다.

오답 포인트

①~③과 ⑤는 모두 운영체제에서 발생하는 지연·중단 현상을 설명하지만 원인이 다르다. 특히 ③의 '긴 프로세스가 CPU를 양보할 때까지 다른 프로세스들이 기다린다'는 표현은 페이지 교체가 아니라 CPU 점유에서 생기는 호위 효과이다. 스래싱은 반드시 '빈번한 페이지 부재'와 '과도한 페이지 교체'가 핵심이다.

다시 나오면 이렇게 판단

페이지 부재와 페이지 교체가 반복되어 실제 실행 시간이 줄어든다는 내용이면 스래싱으로 판단한다. 서로 기다리면 교착 상태, 특정 프로세스만 계속 밀리면 기아 상태, 긴 프로세스 뒤에서 여러 프로세스가 기다리면 호위 효과, 장치나 예외가 CPU에 처리를 알리면 인터럽트로 구별한다.

암기 포인트

스래싱은 '실행보다 페이지 교체가 더 바쁜 상태'이다.

NoteBase

15. 다음 중 양자컴퓨팅(quantum computing)에 대한 설명으로 옳은 것만을 <보기>에서 모두 고르면?

<보 기>

ㄱ. 양자 얽힘(entanglement), 중첩(superposition)과 같은 양자 역학 현상을 이용한다.

ㄴ. 0, 1 또는 0과 1의 상태를 동시에 가질 수 있는 큐비트(Qbit)가 계산의 기본 단위이다.

ㄷ. QKD(Quantum Key Distribution) 시스템에서는 양자 얽힘 현상을 이용하지 않는다.

ㄹ. 이산로그(discrete logarithm) 문제를 다항시간의 복잡도로 풀 수 있는 방법을 제공한다.

ㅁ. 소인수분해(integer factorization) 문제를 다항시간(polynomial time)의 복잡도로 풀 수 있는 방법이 존재하여 기존의 모든 NP(Non-deterministic Polynomial)문제를 다항시간 내에 풀 수 있다.

① ㄱ, ㄴ, ㄷ

② ㄱ, ㄴ, ㄹ

③ ㄴ, ㄷ, ㅁ

④ ㄱ, ㄴ, ㄹ, ㅁ

⑤ ㄴ, ㄷ, ㄹ, ㅁ

정답 2번

출제 포인트

양자컴퓨팅의 기본 원리인 중첩·얽힘·큐비트와 양자 알고리즘이 해결할 수 있는 문제의 범위를 구별하는 문항이다.

핵심 개념

중첩은 큐비트가 측정 전까지 0과 1의 상태가 결합된 양자 상태로 존재하는 성질이고, 얽힘은 여러 큐비트의 상태가 서로 강한 상관관계를 이루는 현상이다.
큐비트는 양자정보의 기본 단위이며 0, 1 또는 두 상태의 중첩으로 표현된다.
쇼어 알고리즘은 소인수분해와 이산로그 문제를 양자컴퓨터에서 다항시간에 해결할 수 있지만, 이것이 모든 NP 문제를 다항시간에 해결한다는 뜻은 아니다.

NoteBase

선지별 해설

① X
ㄷ은 옳지 않고 ㄹ은 옳은데, 이 조합은 ㄷ을 포함하고 ㄹ을 빠뜨렸다. 올바른 조합은 ㄱ, ㄴ, ㄹ이다.

② O
ㄱ은 양자컴퓨팅이 중첩과 얽힘 같은 양자역학 현상을 이용한다는 설명이고, ㄴ은 큐비트의 성질과 역할을 바르게 설명한다. ㄹ도 쇼어 알고리즘이 이산로그 문제를 다항시간에 해결할 수 있다는 내용이므로 ㄱ, ㄴ, ㄹ의 조합이 옳다.

③ X
이 조합에 포함된 ㄷ과 ㅁ은 모두 옳지 않으며, 옳은 ㄱ과 ㄹ도 빠져 있다. ㄷ은 '양자 얽힘 현상을 이용하지 않는다'가 아니라 얽힘을 이용하는 QKD 방식도 존재한다고 고쳐야 하고, ㅁ은 '모든 NP 문제'가 아니라 소인수분해와 이산로그 같은 특정 문제를 다항시간에 해결할 수 있다고 고쳐야 한다. 올바른 조합은 ㄱ, ㄴ, ㄹ이다.

④ X
ㅁ의 '기존의 모든 NP 문제를 다항시간 내에 풀 수 있다'는 부분이 틀렸다. 쇼어 알고리즘은 소인수분해를 다항시간에 해결하지만 모든 NP 문제에 대한 다항시간 알고리즘은 아니다. 따라서 ㅁ을 제외한 ㄱ, ㄴ, ㄹ이 올바른 조합이다.

⑤ X
ㄷ과 ㅁ을 포함하고 옳은 ㄱ을 빠뜨린 조합이다. ㄷ은 얽힘 기반 QKD가 존재하므로 틀리고, ㅁ은 특정 문제의 양자 가속을 모든 NP 문제로 부당하게 확대하므로 틀리다. 올바른 조합은 ㄱ, ㄴ, ㄹ이다.

오답 포인트

ㄷ은 모든 QKD가 반드시 얽힘을 사용한다는 뜻이 아니라, 'QKD 시스템에서는 얽힘을 이용하지 않는다'는 단정이 틀렸다는 점이 핵심이다. 또한 ㅁ은 소인수분해를 빠르게 푸는 쇼어 알고리즘의 성과를 모든 NP 문제로 확대해 해석하도록 유도한다.

다시 나오면 이렇게 판단

중첩·얽힘·큐비트에 관한 기본 정의를 먼저 확인하고, 알고리즘의 성능 주장은 '어떤 특정 문제를 빠르게 푸는가'와 '모든 NP 문제를 푸는가'를 구분하여 판단한다. '전혀 이용하지 않는다', '모든 문제를 푼다'와 같은 전면적 표현은 반례가 있는지 확인한다.

NoteBase

암기 포인트

큐비트는 중첩, QKD에는 얽힘 기반 방식도 존재, 쇼어 알고리즘은 소인수분해·이산로그에 강하지만 모든 NP 문제의 해결책은 아니다.

NoteBase

16. IEEE 802.11 무선랜 표준에 대한 설명으로 옳지 않은 것은?

① MIMO(Multi-Input Multi-Output) 등과 같은 다중 안테나 기술이 적용된 경우, 일반적으로 사용하는 안테나 개수가 증가함에 따라 전송 속도가 높아진다.

② 3GPP LTE 표준보다 전송 속도(data rate)가 빠를 수 있다.

③ OFDM(Orthogonal Frequency-Division Multiplexing) 기법은 IEEE 802.11b 규격의 기기에도 사용이 가능하며, 최고 전송 속도를 11Mbps까지 높일 수 있다.

④ 특정 기기가 사용하는 채널의 대역폭(bandwidth)은 가변적일 수 있다.

⑤ 다수의 채널(channel)을 사용하여 근접한 거리에 있는 서로 다른 두 쌍의 기기가 서로 간섭을 받지 않고 동시에 통신할 수 있다.

정답 3번

출제 포인트

IEEE 802.11 무선랜의 다중 안테나, 전송 속도, 변조 방식, 채널 대역폭과 채널 분리의 특징을 구별하는 문제이다. 특히 IEEE 802.11b가 사용하는 전송 방식과 OFDM을 혼동하지 않아야 한다.

핵심 개념

MIMO는 여러 송신·수신 안테나를 이용해 복수의 데이터 흐름을 전송하여 전송률을 높일 수 있는 기술이다.
IEEE 802.11b는 DSSS 계열 방식과 CCK를 사용하여 최고 11*Mbps*를 지원하며, OFDM을 사용하는 규격이 아니다. OFDM은 대표적으로 IEEE 802.11a와 802.11g 등에 사용된다.
무선랜은 규격과 설정에 따라 채널 대역폭을 달리할 수 있고, 서로 겹치지 않는 채널을 이용하면 가까운 여러 통신 쌍의 동시 통신에서 간섭을 줄일 수 있다.

NoteBase

선지별 해설

① O — MIMO에서는 여러 안테나로 복수의 공간 스트림을 전송할 수 있으므로, 일반적으로 안테나 수와 지원 공간 스트림 수가 늘어나면 전송 속도를 높일 수 있다.
② O — IEEE 802.11n·ac 등 일부 무선랜 규격의 전송률은 LTE의 전송률보다 높을 수 있으므로 '빠를 수 있다'는 표현은 타당하다.
③ X — 틀린 표현은 'OFDM 기법은 IEEE 802.11b 규격의 기기에도 사용이 가능하다'는 부분이다. 올바르게는 IEEE 802.11b가 DSSS 계열 방식과 CCK로 최고 11*Mbps*를 지원하고, OFDM은 대표적으로 IEEE 802.11a·g 등에 사용된다고 해야 한다.
④ O — IEEE 802.11 무선랜은 규격과 운용 설정에 따라 서로 다른 채널 대역폭을 사용할 수 있으므로 특정 기기가 사용하는 채널 대역폭은 가변적일 수 있다.
⑤ O — 가까운 두 통신 쌍에 서로 겹치지 않는 채널을 각각 배정하면 동일 채널 충돌을 피하고 동시에 통신할 수 있다.

오답 포인트

③은 '최고 11*Mbps*'라는 802.11b의 속도 자체는 맞게 제시하면서 그 속도를 구현하는 방식을 OFDM으로 바꾸어 놓았다. 맞는 수치가 포함되어 있어 문장 전체를 옳다고 판단하기 쉬운 선지이다.

다시 나오면 이렇게 판단

무선랜 규격 문제에서는 '규격명 — 변조·전송 방식 — 최고 전송률'을 한 묶음으로 확인한다. 802.11b가 보이면 DSSS·CCK와 11*Mbps*를 연결하고, OFDM이 보이면 802.11a·g 계열인지 확인한다.

암기 포인트

802.11b는 DSSS·CCK로 11*Mbps*, 802.11a·g는 OFDM이다.

NoteBase

17. 객체지향(object-oriented) 언어인 C++에 대한 설명으로 옳지 않은 것만을 <보기>에서 모두 고르면?

<보 기>

ㄱ. 다중 상속(multiple inheritance)이 불가능하다.

ㄴ. 함수 오버로딩(overloading)이 가능하나 한 클래스(class) 내에서 동일한 이름을 가진 다수의 함수는 정의될 수 없다.

ㄷ. 클래스를 통해 데이터 캡슐화(encapsulation)와 추상화(abstraction)를 지원한다.

ㄹ. 다형성(polymorphism)과 상속(inheritance)을 지원한다.

ㅁ. 다형성(polymorphism)의 지원을 위해 동적 바인딩(dynamic binding)을 사용한다.

① ㄱ, ㄴ
② ㄱ, ㄴ, ㅁ
③ ㄱ, ㄷ, ㄹ
④ ㄴ, ㄹ, ㅁ
⑤ ㄷ, ㄹ, ㅁ

정답 1번

출제 포인트

C++의 객체지향 특성인 다중 상속, 함수 오버로딩, 캡슐화, 추상화, 상속, 다형성 및 동적 바인딩을 정확히 구분하고 옳지 않은 설명만 고르는 문제이다.

핵심 개념

다중 상속: 하나의 파생 클래스가 둘 이상의 기반 클래스를 상속받는 기능이며 C++에서 지원한다.
함수 오버로딩: 같은 범위에서 이름은 같지만 매개변수의 개수나 자료형 등이 다른 함수를 여러 개 정의하는 기능이다.
캡슐화와 추상화: 클래스는 데이터와 연산을 묶고 접근을 통제하며, 필요한 특성만 드러내는 추상화를 구현하는 수단이다.
상속과 다형성: 기존 클래스의 특성을 물려받아 확장할 수 있고, 같은 인터페이스가 객체의 실제 형식에 따라 다르게 동작할 수 있다. 런타임 다형성은 가상 함수를 통한 동적 바인딩으로 구현한다.
따라서 ㄱ은 C++에서 다중 상속이 가능하므로 틀리고, ㄴ은 오버로딩을 설명하면서 동일한 이름의 여러 함수를 정의할 수 없다고 했으므로 틀리다. ㄷ, ㄹ, ㅁ은 옳다.

NoteBase

선지별 해설

① O — ㄱ과 ㄴ만 옳지 않다. ㄱ의 '다중 상속이 불가능하다'는 표현은 틀렸으며, 올바르게는 C++에서 다중 상속이 가능하다. ㄴ의 '동일한 이름을 가진 다수의 함수는 정의될 수 없다'는 표현도 틀렸으며, 올바르게는 매개변수 목록이 다르면 한 클래스 안에서도 같은 이름의 함수를 여러 개 오버로딩할 수 있다.
② X — ㅁ을 옳지 않은 설명에 포함한 부분이 틀렸다. 동적 바인딩은 가상 함수 호출을 실행 시점의 실제 객체 형식에 연결하여 런타임 다형성을 지원하므로 ㅁ은 옳다. 올바른 조합은 ㄱ, ㄴ이다.
③ X — ㄴ을 빼고 ㄷ과 ㄹ을 옳지 않은 설명에 포함한 부분이 틀렸다. 클래스는 캡슐화와 추상화를 지원하고 C++는 다형성과 상속을 지원하므로 ㄷ과 ㄹ은 옳다. 올바른 조합은 ㄱ, ㄴ이다.
④ X — ㄱ을 빼고 ㄹ과 ㅁ을 옳지 않은 설명에 포함한 부분이 틀렸다. C++는 상속과 다형성을 지원하며 동적 바인딩으로 런타임 다형성을 구현할 수 있으므로 ㄹ과 ㅁ은 옳다. 올바른 조합은 ㄱ, ㄴ이다.
⑤ X — ㄷ, ㄹ, ㅁ을 모두 옳지 않은 설명으로 묶은 부분이 틀렸다. 세 설명은 각각 클래스의 캡슐화·추상화, C++의 상속·다형성, 동적 바인딩에 의한 런타임 다형성을 올바르게 설명한다. 올바른 조합은 ㄱ, ㄴ이다.

오답 포인트

ㄴ의 앞부분인 '함수 오버로딩이 가능하다'만 보고 문장 전체를 옳다고 판단하기 쉽다. 오버로딩은 오히려 같은 이름의 함수를 여러 개 정의하는 기능이므로, 뒤의 '정의될 수 없다'가 앞부분과 모순된다. 또한 동적 바인딩은 모든 다형성이 아니라 주로 가상 함수에 의한 런타임 다형성과 연결된다는 점을 구분해야 한다.

다시 나오면 이렇게 판단

각 문장을 기능 지원 여부와 구현 조건으로 나누어 판단한다. C++는 다중 상속을 지원하고, 같은 이름이라도 매개변수 목록이 다르면 오버로딩할 수 있다. 클래스는 캡슐화와 추상화의 수단이며, 가상 함수의 동적 바인딩은 런타임 다형성을 구현한다. 그런 다음 틀린 문장만 모아 조합 선지와 대조한다.

암기 포인트

C++는 다중 상속 가능, 오버로딩은 같은 이름·다른 매개변수, 가상 함수의 동적 바인딩은 런타임 다형성이다.

NoteBase

18. 다음의 C 언어로 작성된 프로그램에서 calc함수의 실행시간(또는 실행 단계 수)를 점근 표기법으로 표시했을 때 옳은 것은?

```
int calc(int n) {
        int i, j, ret = 0;
        if (n <= 0)
                return 0;
        for (i = 1; i <= n; i ++)
                ret += 1;
        return ret + calc(n - 1);
}
```

① $O(\log_2 n)$
② $O(n\log_2 n)$
③ $O(n)$
④ $O(n^2)$
⑤ $O(n^3)$

정답 4번

출제 포인트

재귀 호출의 각 단계에서 반복문이 수행되는 횟수를 모두 합산하여 전체 시간복잡도를 구하는 문제이다.

조건 정리

입력값이 n인 단계의 for문은 n번 실행된다. 그 뒤 calc($n-1$)을 호출하므로 입력 크기는 1씩 감소하며, $n \le 0$이면 재귀가 종료된다. 따라서 각 단계의 실행 횟수는 $n, n-1, \ldots, 1$이다.

사용할 공식

점화식은 $T(n) = T(n-1) + \Theta(n)$이다. 이를 전개할 때 자연수의 합 $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$을 사용한다. 재귀 단계마다 길이가 달라지는 반복문 비용을 모두 더해야 하기 때문이다.

계산 과정

1. 최초 호출의 반복문은 n번 실행된다.
2. 이어지는 호출들의 반복문은 각각 $n-1, n-2, \ldots, 1$번 실행된다.
3. 전체 반복 횟수는 $n + (n-1) + \cdots + 1 = \frac{n(n+1)}{2}$이다.
4. 최고차항이 n^2이므로 실행시간은 $O(n^2)$이다.

NoteBase

선지 판단

① X — $O(\log_2 n)$은 입력 크기가 단계마다 일정 비율로 줄어드는 경우에 주로 나타난다. 이 함수는 입력을 1씩 줄이고 각 단계에서 반복문까지 실행하므로 틀리며, 올바르게는 $O(n^2)$이다.
② X — 재귀 깊이는 n이고 단계별 반복 횟수도 달라 단순히 n과 $\log_2 n$을 곱할 수 없다. $O(n\log_2 n)$이 아니라 올바르게는 $O(n^2)$이다.
③ X — 재귀 호출 횟수만 보면 $O(n)$처럼 보이지만, 각 호출 내부의 반복문 비용을 빠뜨린 판단이다. 올바르게는 $O(n^2)$이다.
④ O — 전체 반복 횟수가 $\frac{n(n+1)}{2}$이므로 최고차항에 따라 $O(n^2)$이다.
⑤ X — 반복 횟수의 합은 1차식들의 합이므로 2차식이 된다. $O(n^3)$이 아니라 올바르게는 $O(n^2)$이다.

오답 포인트

③처럼 재귀 깊이만 세어 $O(n)$으로 판단하는 것이 핵심 함정이다. 전체 시간은 재귀 호출 횟수뿐 아니라 각 호출에서 수행되는 for문의 비용까지 합산해야 한다.

다시 나오면 이렇게 판단

재귀 함수 안에 반복문이 있으면 먼저 한 단계의 비용을 구하고, 입력이 어떻게 줄어드는지 확인한 뒤 모든 단계의 비용을 합산한다. 이 문제는 $T(n) = T(n-1) + \Theta(n)$으로 세운 후 자연수의 합으로 판단한다.

암기 포인트

1씩 감소하는 재귀에서 각 단계가 현재 입력 크기만큼 반복되면 $n + (n-1) + \cdots + 1 = \Theta(n^2)$이다.

NoteBase

19. 초기 페이지 프레임은 비어 있다고 가정할 때, 3개의 페이지 프레임을 가진 기억장치에서 다음과 같이 페이지 참조열이 주어졌다. LRU(Least Recently Used) 알고리즘을 적용했을 때 발생하는 페이지 부재의 수는?

2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 5, 3

① 8
② 9
③ 10
④ 11
⑤ 12

정답 3번

출제 포인트

이 문제는 3개의 페이지 프레임에 LRU 교체 원리를 적용하여 전체 페이지 부재 횟수를 구하는 문제이다.

조건 정리

주어진 값: 페이지 프레임은 3개이고, 참조열은 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 5, 3이다.
구해야 할 값: 전체 페이지 부재 횟수이다.
주의할 조건: 초기 프레임은 모두 비어 있으므로 처음 적재되는 페이지도 부재로 센다. 프레임이 가득 찬 뒤 부재가 발생하면 현재 프레임의 페이지 중 가장 오랫동안 참조되지 않은 페이지를 교체한다.

사용할 공식

페이지 부재 수=Σ 현재 프레임에 없는 페이지를 참조한 횟수
이 원리를 쓰는 이유는 LRU가 참조 페이지의 존재 여부를 먼저 확인하고, 없는 경우에만 가장 오래 사용되지 않은 페이지를 교체하기 때문이다.

NoteBase

계산 과정

참조 뒤의 프레임을 최근 사용 순서와 무관한 집합 형태로 나타내면 다음과 같다. 괄호의 부재 누계로 횟수를 확인할 수 있다.
1. 2 → {2}(1), 3 → {2, 3}(2), 4 → {2, 3, 4}(3)이다.
2. 2는 적중이므로 누계는 3이다. 1은 가장 오래 사용하지 않은 3을 교체하여 {2, 4, 1}(4)이 된다.
3. 5는 4를 교체하여 {2, 1, 5}(5), 6은 2를 교체하여 {1, 5, 6}(6)이 된다.
4. 2는 1을 교체하여 {5, 6, 2}(7), 1은 5를 교체하여 {6, 2, 1}(8)이 된다.
5. 다음 2는 적중하여 누계가 8로 유지된다. 3은 6을 교체하여 {1, 2, 3}(9), 5는 1을 교체하여 {2, 3, 5}(10)이 된다. 마지막 3은 적중한다.
따라서 전체 페이지 부재 수는 10이다.

선지 판단

① X — 페이지 부재 수는 8이 아니라 10이다. 8은 열 번째 참조인 2까지의 부재 누계로, 뒤의 3과 5에서 발생하는 두 번의 부재를 누락한 값이다. 올바르게는 10이다.
② X — 페이지 부재 수는 9가 아니라 10이다. 9는 열한 번째 참조인 3까지의 부재 누계로, 그다음 5의 부재를 누락한 값이다. 올바르게는 10이다.
③ O — 참조열 전체를 LRU로 처리하면 페이지 부재가 10회 발생하므로 계산 결과와 일치한다.
④ X — 페이지 부재 수는 11이 아니라 10이다. 이미 프레임에 있는 페이지를 다시 참조한 4번째 2, 10번째 2, 마지막 3은 적중이므로 부재로 세지 않는다. 올바르게는 10이다.
⑤ X — 페이지 부재 수는 12가 아니라 10이다. 전체 13회 참조 중 적중은 3회이므로 부재는 10회이다. 올바르게는 10이다.

오답 포인트

초기 프레임에 페이지를 넣는 동작을 부재에서 빼거나, 참조할 때마다 무조건 교체한다고 생각하면 횟수가 달라진다. 특히 4번째 2, 10번째 2, 마지막 3은 프레임 안에 있으므로 부재가 아니다. 또한 LRU는 가장 먼저 들어온 페이지가 아니라 가장 오랫동안 사용되지 않은 페이지를 교체한다.

다시 나오면 이렇게 판단

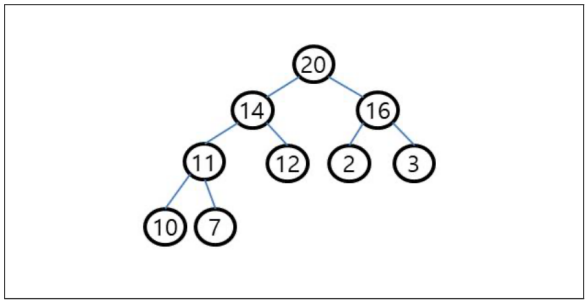
각 참조마다 먼저 프레임 안에 같은 페이지가 있는지 확인한다. 있으면 적중으로 처리하고 최근 사용 시점만 갱신한다. 없으면 부재 수를 1 증가시키고, 빈 프레임이 없을 때만 마지막 사용 시점이 가장 오래된 페이지를 교체한다.

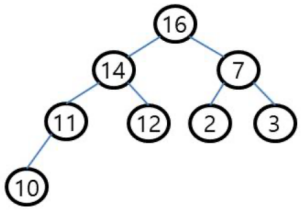
암기 포인트

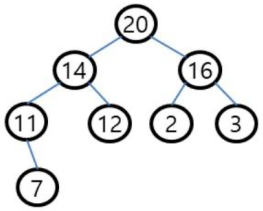
LRU는 '가장 오래전에 사용한 페이지'를 교체하며, 초기 적재도 부재이고 프레임 안의 페이지 재참조는 적중이다.

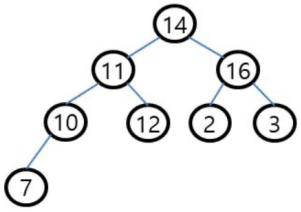
NoteBase

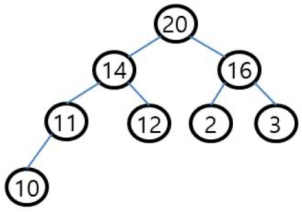
20. 아래의 최대힙(max heap)에서 노드를 한 개 삭제하는 연산을 실행하였을 때의 결과로 옳은 것은?

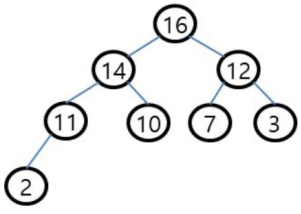


① 

② 

③ 

④ 

⑤ 

정답 1번

NoteBase

출제 포인트

최대 힙에서 최댓값인 루트 노드를 삭제한 뒤, 마지막 노드를 루트로 옮기고 아래 방향으로 재정렬하는 절차를 묻는 문제이다.

올바른 순서

1. 루트의 20을 삭제한다. — 최대 힙의 삭제 연산은 루트에 있는 최댓값을 제거한다.
2. 마지막 노드 7을 루트로 옮기고 기존 마지막 자리를 제거한다. — 완전 이진 트리의 형태를 유지한다.
3. 7의 두 자식 14와 16 중 더 큰 16과 7을 교환한다. — 부모가 두 자식보다 크거나 같아야 하는 최대 힙의 성질을 회복한다.
4. 내려간 7을 자식 2, 3과 비교한다. — 7이 두 자식보다 크므로 더 이상 교환하지 않는다.

최종 레벨 순회 결과는 16, 14, 7, 11, 12, 2, 3, 10이 된다. 마지막 노드를 먼저 루트로 옮겨야 완전 이진 트리 형태를 유지한 채 필요한 경로에서만 하향 재정렬할 수 있다.

선지별 해설

① O — 루트가 16이고, 오른쪽 자식으로 내려간 7이 2와 3보다 크다. 모든 부모가 자식보다 크거나 같으며 완전 이진 트리의 형태도 유지되므로 올바른 결과이다.

② X — 틀린 부분은 루트 20을 그대로 두고 말단의 10만 삭제한 구조이다. 올바르게는 루트 20을 삭제하고 마지막 노드 7을 루트로 옮긴 뒤 16과 교환하여 ①의 구조가 되어야 한다.

③ X — 틀린 부분은 14의 자식으로 더 큰 값 16을 둔 구조이다. 부모 14가 자식 16보다 작으므로 최대 힙의 성질을 위반한다. 올바르게는 16이 루트가 되고, 7이 16의 오른쪽 자식이 되어야 한다.

④ X — 틀린 부분은 루트 20을 남겨 두고 마지막 노드 7만 삭제한 구조이다. 올바르게는 최댓값 20을 삭제하고 마지막 노드 7을 루트로 옮겨 하향 재정렬해야 한다.

⑤ X — 이 구조 자체는 최대 힙의 조건을 만족하지만, 한 번의 표준 삭제 연산으로 얻는 배열 순서를 임의로 다시 배치한 결과이다. 삭제 후에는 마지막 노드 7이 내려가는 경로에서만 교환하므로 올바른 결과는 16, 14, 7, 11, 12, 2, 3, 10이다.

오답 포인트

②와 ④처럼 단순히 마지막 잎 노드를 지우는 것으로 착각하기 쉽다. 또한 ⑤처럼 남은 모든 원소로 힙을 새로 구성할 필요는 없다. 최대 힙의 삭제는 루트 제거, 마지막 노드 이동, 더 큰 자식과의 하향 교환 순서로 진행된다.

NoteBase

다시 나오면 이렇게 판단

최대 힙 삭제 문제에서는 먼저 루트를 제거하고 마지막 노드를 빈 루트로 옮긴다. 그다음 옮긴 노드가 두 자식 중 더 큰 값보다 작을 때만 그 자식과 교환하며, 부모가 두 자식보다 크거나 같아지는 지점에서 멈춘다.

암기 포인트

최대 힙 삭제는 '루트 삭제 → 마지막 노드 이동 → 큰 자식과 아래로 교환'이다.

NoteBase