제134회 정보관리기술사 해설집

2024.07.27

국가기술자격 기술사 시험문제

기술사 제 134 회 제 3 교시 (시험시간: 100 분)

분야	정보통신	자격종목	정보관리기술사	수검번호		성명	

※ 다음 문제 중 4 문제를 선택하여 설명하시오. (각 10 점)

1. 실행 중인 애플리케이션에 대한 배포 전략 및 테스트 전략에 대하여 설명하시오.

2. 소프트웨어 테스트에 대하여 설명하시오.
 가. 소프트웨어 테스트 원리
 나. 블랙박스 테스트와 화이트박스 테스트
 다. 명세기반, 구조기반, 경험기반 테스트 기법

3. SBOM(Software Bill of Materials)에 대하여 설명하시오.
 가. 오픈소스 소프트웨어 취약점
 나. SBOM 기반 오픈소스 소프트웨어 관리 방안

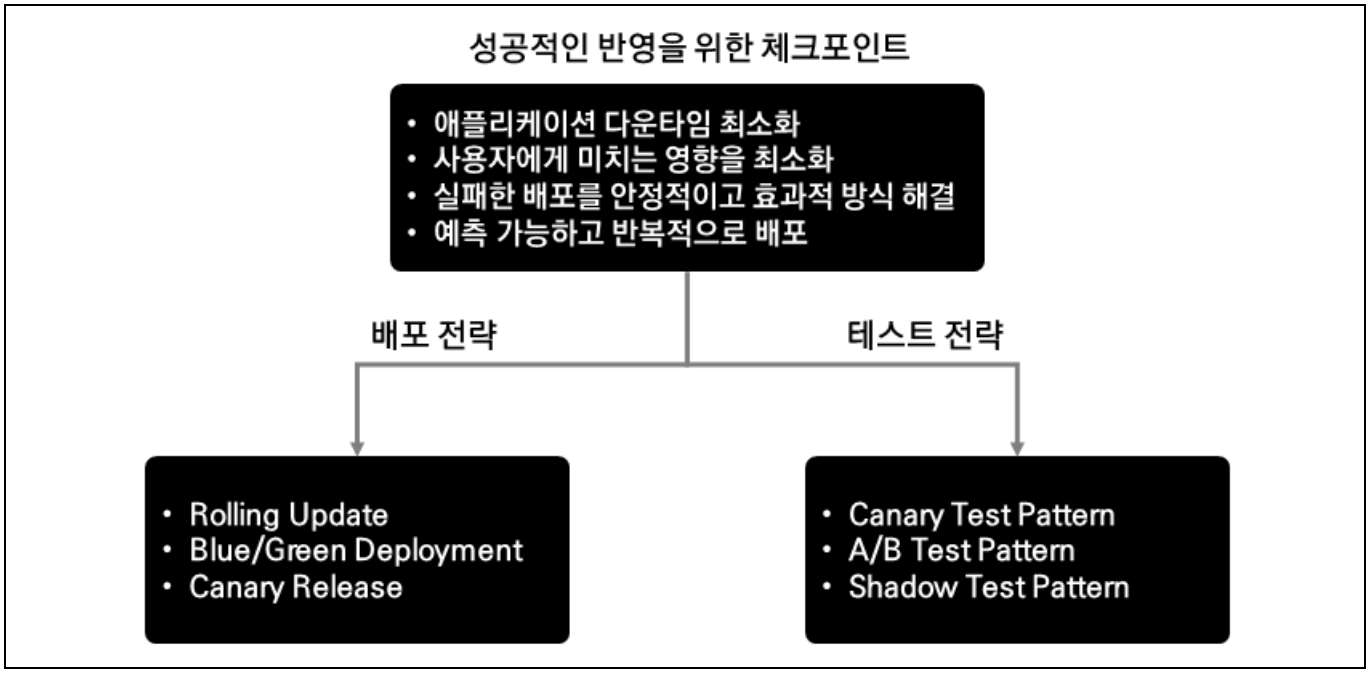
4. 알고리즘의 복잡도를 설명하고 성능을 표기하기 위한 O-Notation의 개념과 유형 및 유형별 연산시간의 차이를 설명하시오.

5. 다차원 색인구조(Multidimensional Index Structure)의 개념, 유형, 활용 사례에 대하여 설명하시오.

6. 일부 오픈소스 라이선스가 개방형(예: MIT, BSD 등)에서 폐쇄형(예: SSPL(Server Side Public License), BSL(Business Source License) 등)으로 변화하고 있다. 이러한 오픈소스 라이선스 정책 변경의 배경 및 소프트웨어 산업에 미치는 영향에 대하여 설명하시오.

01	소프트웨어 배포 및 테스트 전략		
문제	실행 중인 애플리케이션에 대한 배포 전략 및 테스트 전략에 대하여 설명하시오.		
도메인	소프트웨어공학	난이도	상(상/중/하)
키워드	Rolling Update, Blue/Green Deployment, Canary Release, Canary Test Pattern, A/B Test Pattern, Shadow Test Pattern		
출제배경	실시간 애플리케이션에 대한 배포 및 테스트 기법 확인		
참고문헌	ITPE 서브노트, 구글 클라우드 아키텍처 센터(애플리케이션 배포 및 테스트 전략)		
해설자	강남평일야간반 전일 기술사(제 114회 정보관리기술사 / nikki6@hanmail.net)		

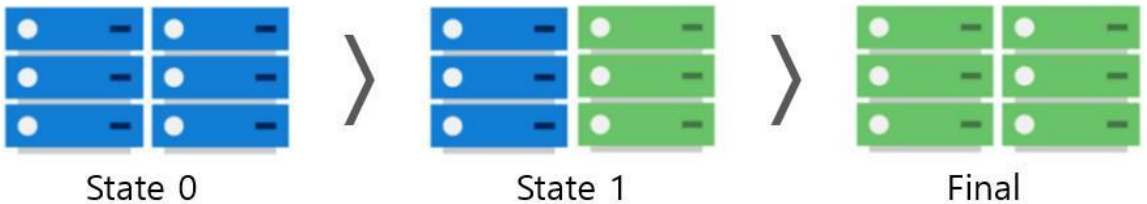
I. 실행 중인 애플리케이션의 성공적 배포를 위한 체크 포인트



- 시스템에 의해 제공하는 비즈니스의 연속성과 안정성을 보장하기 위해 운영 환경에 소스 배포 시 서비스가 중단되지 않도록 코드를 Deploy 및 테스트 할 수 있는 전략과 프로세스 필요

II. 무중단 배포 전략 종류 설명

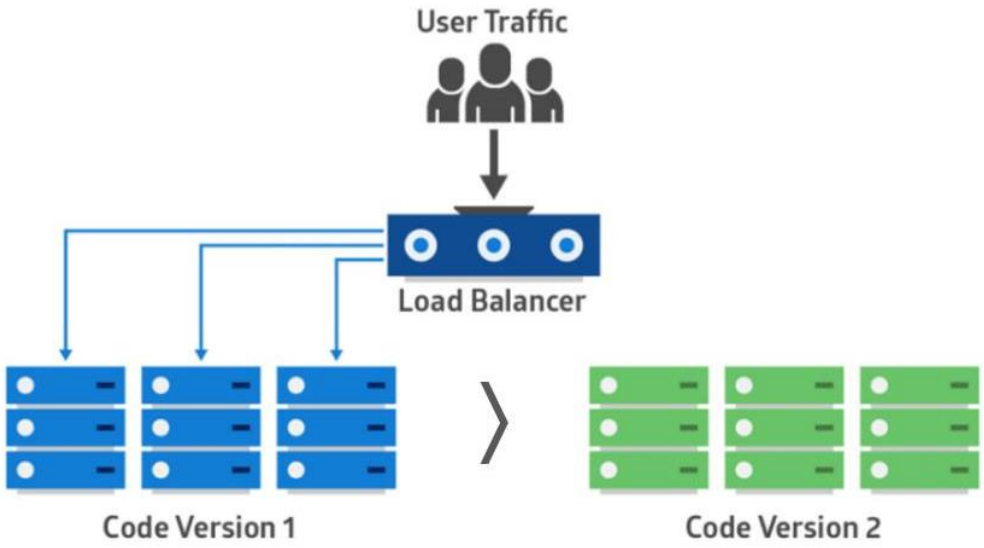
가. Rolling Update

개념도	

설명	- 일반적인 배포를 의미하며, 단순하게 인스턴스(또는 서버)에 대해 동일한 인스턴스를 띄우고, 준비가 되어 있는 상황에서 1 개씩 Rolling 을 통해 점진적으로 인스턴스를 변경하는 기법 - (장점) 관리 및 롤백이 용이 - (단점) 서버 처리 용량에 대한 사전 고려 필요

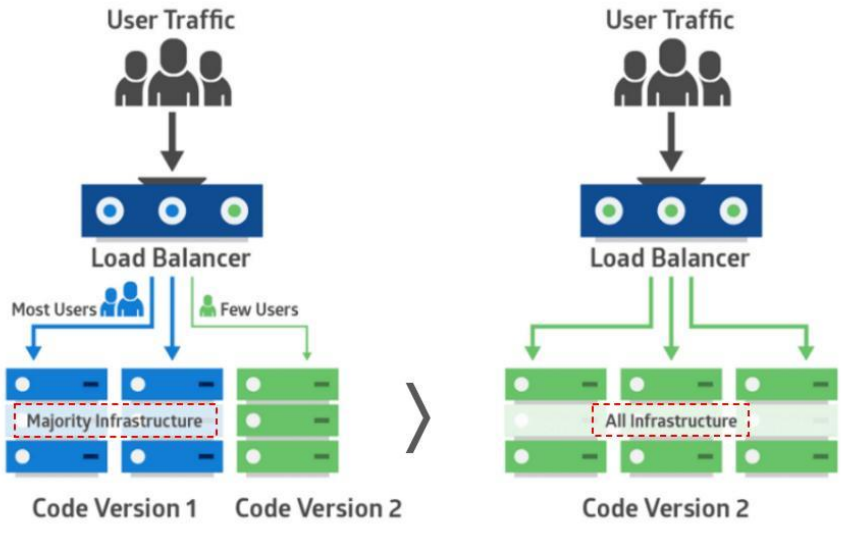
- Ramped, Incremental 이라고도 불리며, 기존에 사용하고 있었던 기본적인 배포 전략 방식

나. Blue/Green Deployment

개념도	
설명	- Old 버전을 블루, New 버전을 그린으로 호명하고, New 버전을 모두 배포 후 서비스 준비가 되었을 때 모든 트래픽을 New 버전으로 한번에 Switching 을 하는 기법 - (장점) 운영 환경에 영향을 주지 않고, 실제 서비스 환경으로 신버전 테스트가 가능 - (단점) 시스템 자원이 두배로 필요하여 비용이 증가

- Red-Black 이라고도 불리며, 서버가 2개가 다 active이기 때문에 Rollbak이 다소 쉽게 처리 가능

다. Canary Release

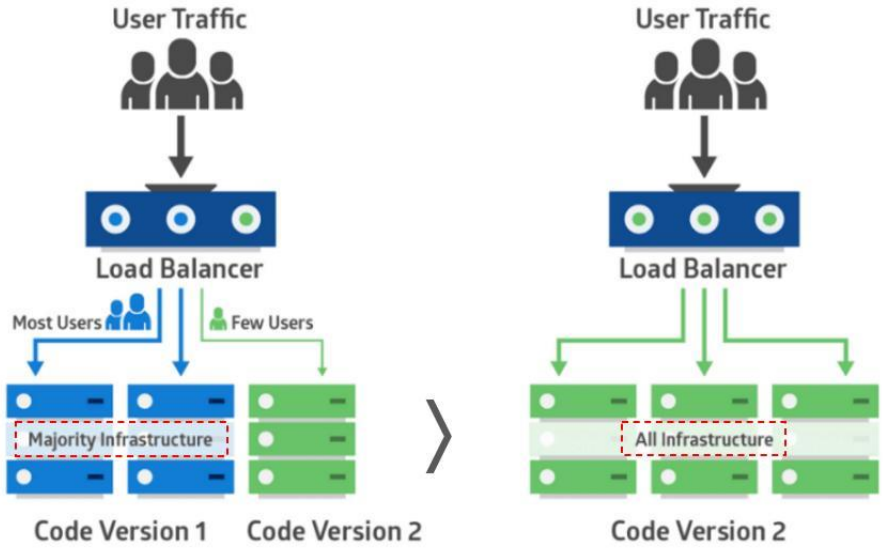
개념도	

설명	- 트래픽 제어를 통해 일부 사용자만 신규 서버로 접속하게 하여 모니터링과 디버깅을 수행한 후 문제가 없는 경우 모든 서버로 교체하는 기법 - (장점) Risk 를 빠르게 감지 가능, A/B 테스트로도 활용 가능 - (단점) 네트워크 트래픽에 대한 제어 부담

- 수정한 코드가 워낙 많이 바뀌어서 불안할 경우 점진적으로 배포하는 형태를 통해 위험 감소 가능

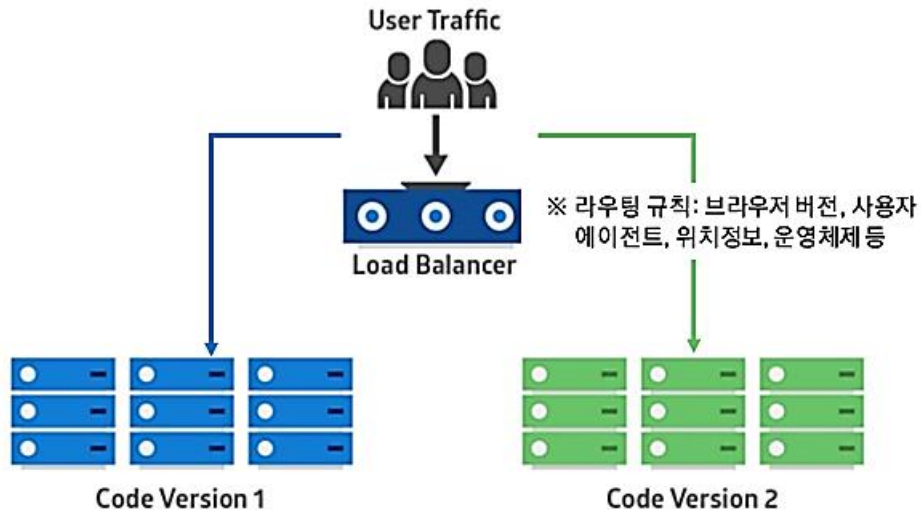
III. 실행 중 애플리케이션 테스트 전략

가. Canary Test Pattern

개념도	
설명	- 카나리아 테스트에서는 다음 다이어그램과 같이 변경사항을 부분적으로 출시한 후 기준 배포와 비교하여 성능을 평가 - 이 테스트 패턴에서는 프로덕션 버전과 함께 애플리케이션의 새 버전을 배포. 다음 프로덕션 버전에서 카나리아 버전으로 트래픽 비율을 분할하고 라우팅하여 카나리아의 성능을 평가 - (장점) 실시간 프로덕션 트래픽 테스트 가능, 빠른 롤백, 제로 다운 타임 - (단점) 느린 출시, 관측 복잡성, 이전 버전과 호환성 고려

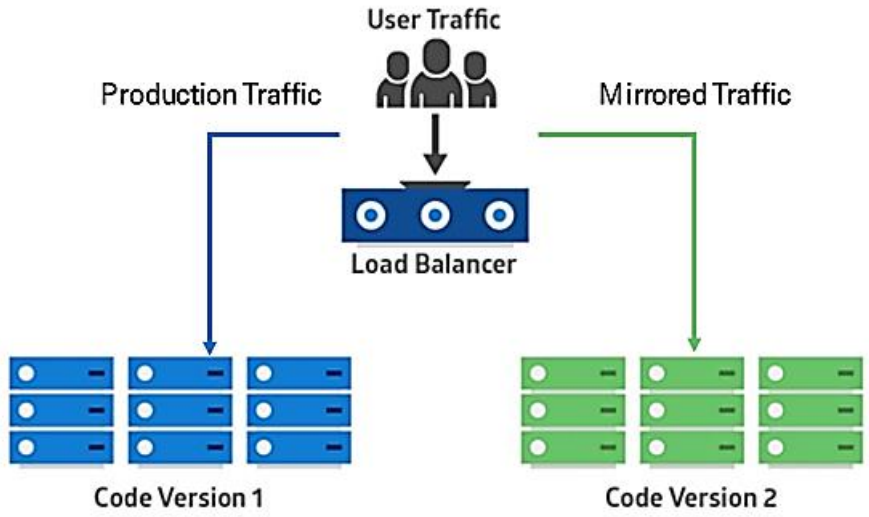
- 카나리아 테스트에서 부분 출시는 다양한 분할 전략을 따를 수 있음
- 예를 들어 애플리케이션에 지리적으로 분산된 사용자가 있는 경우 먼저 새 버전 리전 또는 특정 위치 출시

나. A/B Test Pattern

개념도	
설명	- A/B 테스트는 데이터에서 파생된 결과를 기반으로 예측은 물론 비즈니스 결정을 내리는 데 사용 - A/B 테스트를 수행 경우 그림의 라우팅 규칙에 따라 일부 사용자를 새 기능으로 라우팅 - (장점) 애플리케이션 기능의 효과를 측정하는 데 가장 적합 - (단점) 복잡한 설정, 편향된 샘플링

- 중복 트래픽에 대해 여러 A/B 테스트를 실행하면 모니터링 및 문제해결 프로세스가 어려울 수 있음

다. Shadow Test Pattern

개념도	
설명	- 위의 그림과 같이 새 버전을 사용자로부터 숨기는 방식을 이용하여 새 버전을 현재 버전과 함께 배포하고 실행 - 수신 요청은 테스트 환경에서 미러링되고 재실행. 이 프로세스는 이전에 캡처된 프로덕션 트래픽의 사본이 새로 배포된 서비스에 대해 재실행된 후에 실시간 또는 비동기적으로 발생

	- (장점) 제로 프로덕션 영향, 배포 위험 감소 - (단점) 비용 및 운영 오버헤드

- 여러 방법으로 애플리케이션을 배포하고 출시할 수 있으며, 각 접근법에는 장단점이 존재

IV. 배포 및 테스트 위험을 최소화 하기 위한 권장사항

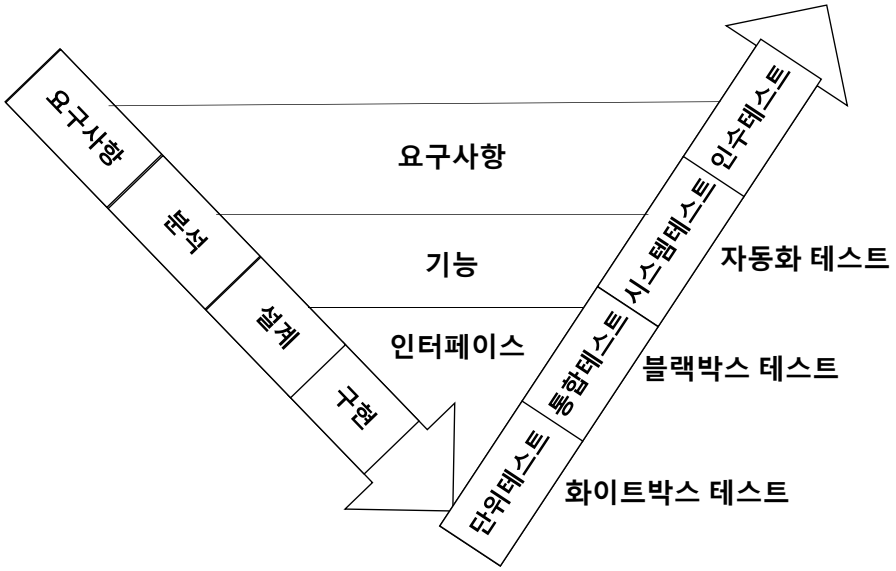
테스트 전략	상세 내용
이전 버전과의 호환성	- 여러 애플리케이션 버전을 동시에 실행하는 경우 데이터베이스가 모든 활성 버전과 호환되는지 확인
지속적 통합/지속적 배포 (CI/CD)	- CI는 종속 항목 확인, 단위 및 통합 테스트, 빌드 프로세스를 통과한 후에만 기능 분기에 체크인된 코드가 기본 분기와 병합되도록 함. 따라서 애플리케이션의 모든 변경사항은 배포되기 전에 테스트 - CD를 사용하면 CI로 빌드된 코드 아티팩트가 패키징되어 하나 이상의 환경에 배포
자동화	- 사용자에게 애플리케이션 업데이트를 지속적으로 제공하는 경우 소프트웨어를 안정적으로 빌드하고, 테스트하고, 배포하는 자동화된 프로세스를 빌드 필요
IaC 및 GitOps	- 복잡한 배포 및 테스트 전략을 관리해야 하는 경우 코드형 인프라(IaC) 및 GitOps 도구를 사용 권장

- 상태 점검에 실패한 배포에 자동 롤백 기능을 설정하여 즉각적인 롤백 전략체계 마련 필요

"끝"

02	소프트웨어 테스트		
문제	소프트웨어 테스트에 대하여 설명하시오. 가. 소프트웨어 테스트 원리 나. 블랙박스 테스트와 화이트박스 테스트 다. 명세기반, 구조기반, 경험기반 테스트 기법		
도메인	소프트웨어공학	난이도	하(상/중/하)
키워드	결함발견, 초기집중, 파레토 법칙, 살충제 패러독스, 오류부재궤변, 불완전성, 정황 의존성		
출제배경	소프트웨어공학의 기본 지식 확인		
참고문헌	ITPE 서브노트		
해설자	강남평일야간반 전일 기술사(제 114회 정보관리기술사 / nikki6@hanmail.net)		

I. 소프트웨어의 품질 향상, 소프트웨어 테스트 개념



개념	- 소프트웨어가 이해관계자의 요구사항을 충족하고, Product의 신뢰성을 향상하기 위해 수행하는 결함 발견 등의 활동

- 소프트웨어의 동작과 성능이 테스트 케이스에 따라 실행하고, 7가지 원리에 의해 테스트케이스를 설계함

II. 소프트웨어 테스트 원리

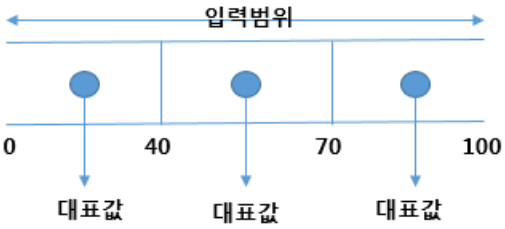
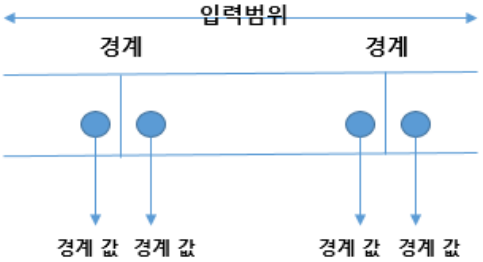
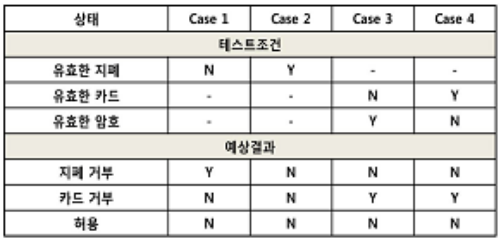
원리	내용	원인
결함발견	- 결함을 100% 제거하는 것이 목적이 아니고, 결함이 존재함을 밝히는 것으로 시간이 지나 언제든 결함이 발생할 가능성이 있음	- 품질제고 위한 테스트는 필수 단계
초기 집중 (요르돈 법칙)	- 개발 설계 시부터 테스트를 고려, 결함의 조기발견 및 재발을 방지 - 초기발견은 유지보수 비용이 절감(설계단계	- 품질비용 감소 - 소프트웨어 설계 오류

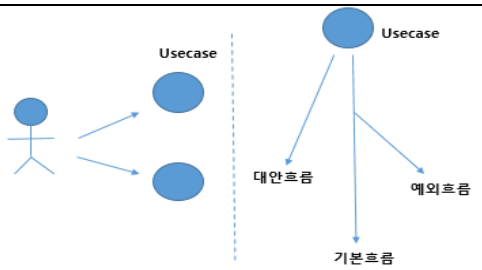
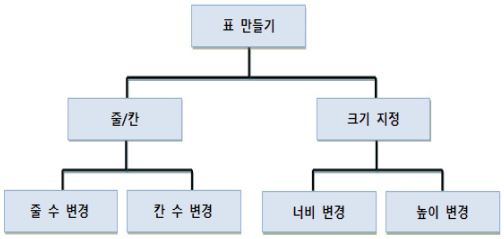
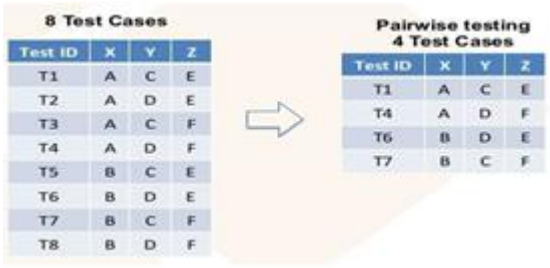
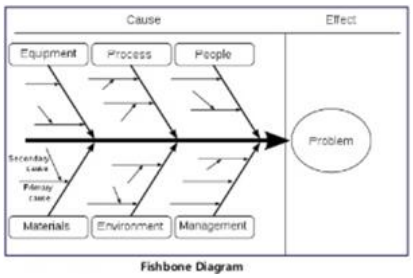
	비용 1 → 운영시점 비용 100 이상)	
파레토 법칙	- 결함의 80%는 20%의 코드에 집중 - 결함이 높은 곳에 자원 집중	- 결함 집중 - 이식성이 높은 코드
오류-부재 궤변	- 결함을 모두 제거했다고 품질이 우수한 것 아님	- 타 시스템과 연동되면서 오류 발생률 증가 가능
불완전성	- 완벽한 테스팅은 불가능	- 인적, 물리적 자원 한계
살충제 패러독스	- 동일한 테스트케이스에 내성이 생겨 오류 발견이 안됨	- 전문 테스터 부족 - 테스터의 역량 부족
정황 의존성	- 테스트는 테스트 주변 환경에 영향을 받음	- 외부요소, 심리요소

- 소프트웨어 테스트는 결함발견 원리에 의해 테스트 프로세스를 명세화하여 테스터에 의해 반드시 수행

III. 블랙박스 테스트와 화이트박스 테스트 상세 설명

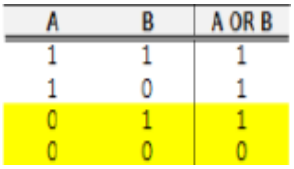
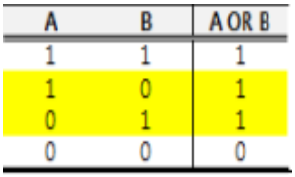
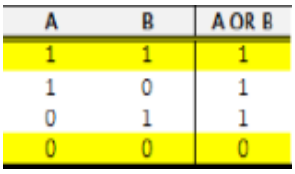
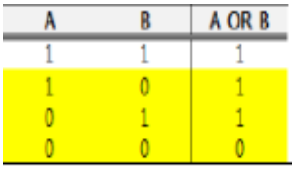
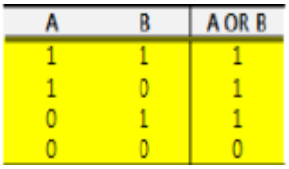
가. 블랙박스 테스트 상세 설명

기법	개념도	상세 설명
동등 클래스 분할 기법 (Equivalence Class Partitioning)	 - 등가 분할 된 대표 값을 이용 테스트 케이스 도출	- 프로그램의 입력 도메인을 등가영역들로 분할 후 각 영역별로 대표되는 값들을 선정하여 테스트케이스를 설계하는 방법 - 예) 입력데이터 x값이 0 ~ 100 사이여야 한다면 TC를 (x < 0), (x = 50), (x > 100)으로 분할하여 적용
경계값 분석 (Boundary Value Analysis)	 - 결함은 경계 값 근처에서 많이 발생 이용	- 입력 영역의 분할 클래스의 경계값으로 테스트 케이스를 설계하는 방법 - 등가 분할 기법의 확장, 등가분할 된 경계의 유효한 값, 경계의 가장 가까운 유효하지 않는 값 선택 테스트 진행 - 예) x값이 0~100사이여야 한다면 TC를 (x=0), (x=100), (x=-0.01), (x=100.1)로 정의
의사결정 테이블 테스팅 (Decision Table Testing)	 - 조건과 결과를 참/거짓으로 표현	- 주요한 의사결정 요소들을 표(결정테이블)로 만들고, 요소들간의 결합에 의한 테스트 케이스 설계 - 각 의사결정 요소들의 조합을 통해 다양한 형태의 테스트 시나리오를 도출

상태 전이 테스팅 (State transition Testing)		- 상태전이 다이어그램 통해 이벤트, 액션, 활동, 상태 변화로 발생되는 관계, 동작 파악하여 테스트 - 임베디드SW 테스트시 적용
유스케이스 테스팅 (Use Case Testing)		- 유즈케이스를 통해 도출되는 비즈니스 시나리오(기본 흐름, 대체 흐름)를 기반으로 테스트를 명세화하여 테스트 - 컴포넌트/단위 레벨 유즈케이스 테스팅 - 시스템 레벨 유즈케이스 테스팅
분류 트리 기법 (Classification Tree Method)		- SW의 일부 또는 전체를 트리 구조로 분석 및 표현하여 테스트 케이스를 설계하는 기법 - 트리구조를 시각화 하여 테스트 케이스를 설계하므로 불필요한 중복 및 누락을 회피
페어와이즈 테스팅 (Paireise Testing)		- 대부분 결함이 2 개의 요소(Pair)의 상호작용에 기인한다는 것에 착안하여, 각 값들이 다른 파라미터의 값과 최소 한번씩은 조합을 이루도록 구성하는 테스트 기법 - 모든 조합을 포함하지 않으므로 결함을 찾지 못하는 경우 발생 가능
원인-결과 그래프 (Cause Effect Graph)		- 입력 데이터간 관계가 출력에 영향을 미치는 상황을 체계적으로 분석하여 테스트 케이스 설계 및 테스트 - 원인,결과에 근거한 테스트 케이스 생성하며 시스템 외부 동작만 고려 - 원인: 시스템 내부 입력 상태 - 결과: 시스템 변환 또는 원인 조합으로 인한 출력 상태
오류예측 기법 (Error Guessing)	예) 입력값 없이 Return 친다. 문법에 어긋난 입력을 시험한다 등	- 각 시험 기법들이 놓치기 쉬운 오류들을 감각과 경험으로 찾아 검증 - Ad-hoc Testing이라고도 하며 직관과 경험에 의한 특정 형태의 결함 예측 및 해당 결함을 드러내는 테스트 케이스 설계 기법.

- 블랙 박스 테스트: 소프트웨어가 수행할 특정 기능을 알기 위해서 각 기능이 완전히 작동되는 것을 입증하는 테스트

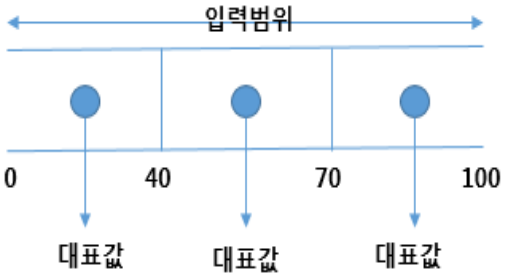
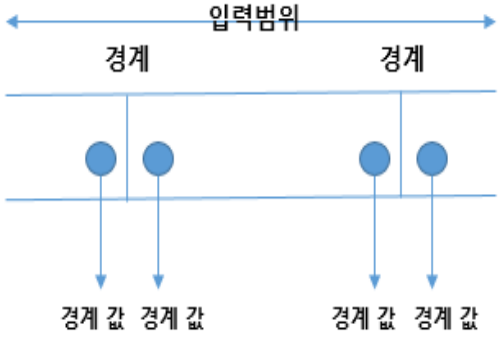
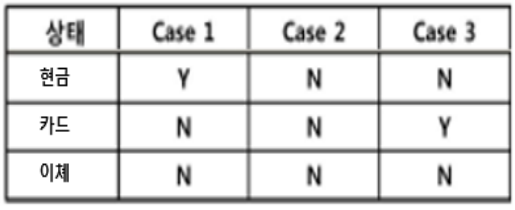
나. 화이트박스 테스트 상세 설명

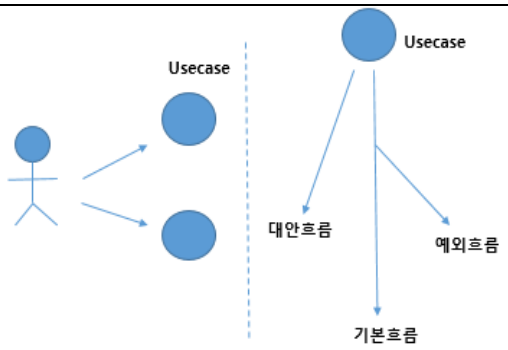
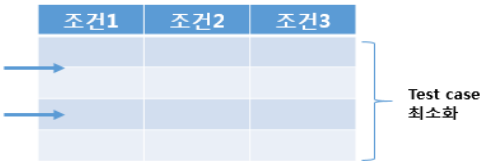
기법	설명	사례
제어구조 시험 (Control Structure Testing)	- McCabe 에 의해 제안된 대표적 White Box Test 기법 - 프로그램의 처리 흐름을 제어하는 방법 및 수행 제어를 위해 사용되는 문장의 구조	- 순차형(순차 구조, Sequence 형) - 선택형(분기구조, If Then Else 형) - 반복형(반복구조, Do While 형)
루프 시험 (Loop Testing)	- 프로그램 루프 구조에 국한해서 실시하는 기법 - 루프 시험의 대상 결함 : 초기화 결함, 인덱싱 및 증가의 결함, 루프의 경계선에서 나타나는 경계 오류 - 루프의 유형 : 단순루프, 중첩루프, 연결루프, 비구조적 루프	- for, while 등 - go to 등
구문 커버리지 (Statement Coverage)	- 프로그램을 구성하는 모든 구문들이 최소한 한번은 실행될 수 있는 입력 데이터를 테스트 데이터로 선정 - 프로그램 내 모든 구문의 테스트를 보장	- 소스코드 if(a>0 or b>0) call join - 테스트케이스 a = 3, b = 4
결정 커버리지 (Decision Coverage)	- 프로그램 내의 전체 결정문이 적어도 한번은 참과 거짓의 결과를 수행하는 테스트 케이스 생성 - 모든 분기문을 테스트하는 방법	
조건 커버리지 (Condition Coverage)	- 결정 명령문 내의 각 조건이 적어도 한 번은 참과 거짓의 결과가 되도록 수행하는 테스트 케이스 - 모든 조건을 커버하는 방법	
조건/결정 커버리지 (Condition/Decision Coverage)	- 전체 조건식뿐만 아니라 개별 조건식도 참 한번, 거짓 한번 결과가 되도록 수행하는 테스트 케이스	
변경조건/결정 커버리지 (MC/DC)	- 각 개별 조건식이 다른 개별 조건식에 영향을 받지 않고 전체 조건식에 독립적으로 영향을 주도록 하는 테스트 케이스	
다중조건/결정 커버리지(Multiple Condition Coverage)	- 결정 포인트 내에 있는 모든 개별식 조건의 모든 조합을 고려한 커버리지	

- 화이트 박스 테스트: 모듈의 원시 코드를 오픈 시킨 상태에서 원시 코드의 논리적인 모든 경로를 테스트

IV. 명세 기반, 구조 기반, 경험 기반 테스트 기법 상세 설명

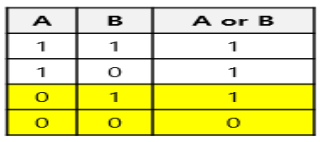
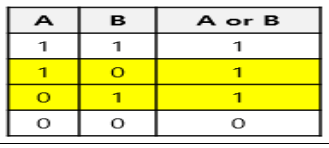
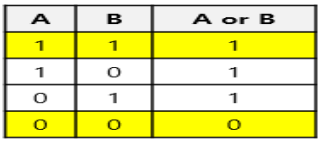
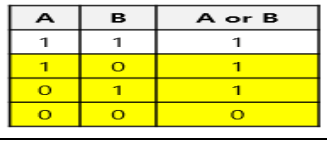
가. 명세 기반 테스트 기법

기법	개념도	특징
동등 클래스 분할		- 데이터의 구간별 대표 값을 도출하여 테스트하는 기법. - 상호 독립적 등가 집합 - 다양한 입력 조건들을 갖춘 Test Case의 유형들을 분할 (상식적 경험에 의존 – Heuristic)
경계 값 분석		- 경계 값 주변에서 결함이 많이 발생하는 원리 이용,경계 값 입력 - 모든 테스트 레벨 - 경험적,결함 발견 높음 - 유효, 비유효 경계 값 고려 테스트 케이스 설계 - 경계치에 치중하며 출력 유형도 고려하는 특성
의사 결정 테이블		- 조건에 따른 참(Y)/거짓(N) 조합(결정 테이블)으로 테스트케이스 작성. - 조건과 상황기반 - 비즈니스 규칙의 명세화
상태전이		- 전이 트리 구성하여 상태 변화 요소들을 조합, 테스트 케이스 작성 - 임베디드 시스템 - 상태의 전이, 상태를 변화시키는 입력과 이벤트의 모든 조합을 포함하는 전이 테이블을 정의 후 테스트 케이스를 설계 - 모델상의 결함, 누락을 효과적으로 발견

유즈케이스		- usecase 명세서를 이용한 테스트 케이스 설계 - 기본흐름과 대안흐름 - 프로세스 흐름
페어와이즈 조합		- 상호작용 조합을 이용한 테스트 케이스 최소화 - 상호작용 요소 식별 - 경험적 의미 조합

- 명세 기반 테스트 기법: 시스템에서 제공하는 기능 및 메뉴 등 명세를 기반으로 테스트 케이스를 설계하는 기법

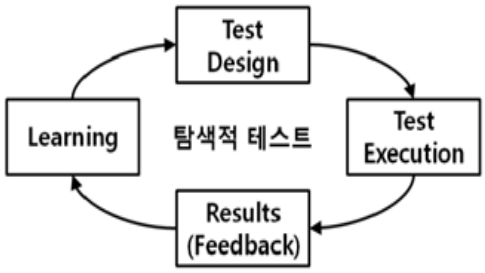
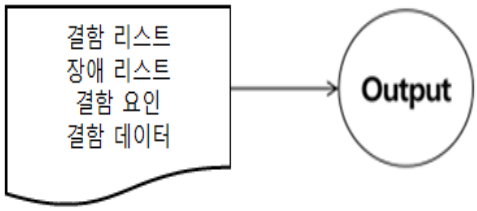
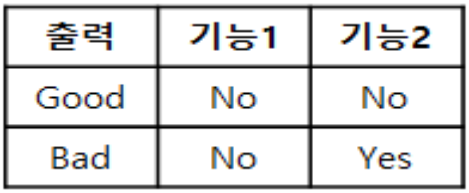
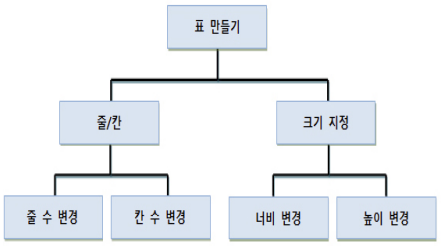
나. 구조 기반 테스트 기법

기법	설명	사례
제어구조	- 프로그램 논리 복잡도 기반 테스트 케이스 설계 기법 - MaCabe 회전 복잡도 = 노드경로–노드 수 + 2	- 복잡도 10~2 0정상, 50이상일 경우 복잡
Loop 테스트	- 프로그램의 Loop 구조에 국한해서 실시하는 기법 - 초기화, 인덱싱/증가, 루프경계선 결함 발견 목적 - 단순, 중첩, 연결, 비구조적 유형 존재	- for, while 등 - go to 등
구문 커버리지	- 프로그램을 구성하는 모든 문장들이 최소한 한번은 실행될 수 있는 입력 데이터를 테스트 데이터로 선정 - 프로그램 내 모든 구문을 보장하는 방법	- 소스코드 If(a>0 or b>0) call join - 테스트케이스 a = 3, b = 4
결정 커버리지	- 프로그램 내의 전체 결정문이 적어도 한번은 참과 거짓의 결과를 수행하는 테스트 케이스 생성 - 모든 분기문을 테스트 하는 방법	
조건 커버리지	- 결정 명령문 내의 각 조건이 적어도 한 번은 참과 거짓의 결과가 되도록 수행하는 테스트 케이스 - 모든 조건을 커버하는 방법	
조건/결정 커버리지	- 전체 조건식 뿐만 아니라 개별 조건식도 참 한번, 거짓 한번 결과가 되도록 수행하는 테스트 케이스	
변경조건/결정 커버리지	- 각 개별 조건식이 다른 개별 조건식에 영향을 받지 않고 전체 조건식에 독립적으로 영향을 주도록 하는 테스트 케이스	

다중조건/결정커버리지	- 결정 포인트 내에 있는 모든 개별식 조건의 모든 조합을 고려한 커버리지	A B A or B 1 1 1 1 0 1 0 1 1 0 0 0

- 구조 기반 테스트 기법: 코드와 개발 설계 등의 SW 구현 정보를 기반으로 테스트 케이스를 설계하는 기법

다. 경험 기반 테스트 기법

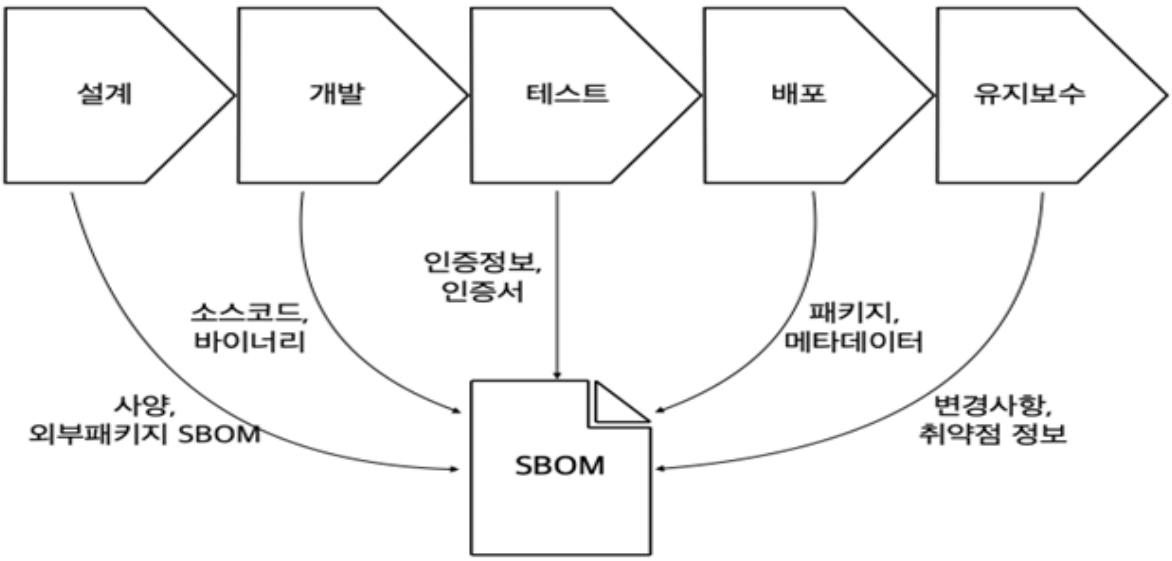
기법	개념도	특징
탐색적 기법		- 학습과 테스트 디자인, 테스트 수행 동시에 하는 Heuristic 테스트 기법 - Time Boxing, Note, 회고 절차 필요
오류 추정		- 가능한 결함을 나열하고 결함이나 오류를 추정에 의해 검출 및 수정 - 결함 리스트/요인/데이터 활용 통한 테스트 케이스 도출
체크 리스트		- 테스트하고 평가해야 할 내용과 경험을 분류하여 나열해 놓은 체크 리스트 기반 테스트 수행
분류 트리		- 흐름을 트리구조로 시각화하여 테스트 케이스 설계 - 입력 변수들 간 조합

- 경험 기반 테스트 기법: 유사 어플리케이션이나 기술에서의 경험, 직관, 테스터의 기술 능력으로부터 테스트 케이스를 추출하는 기법

"끝"

03	SBOM(Software Bill of Materials)		
문제	SBOM(Software Bill of Materials)에 대하여 설명하시오. 가. 오픈소스 소프트웨어 취약점 나. SBOM 기반 오픈소스 소프트웨어 관리 방안		
도메인	인공지능	난이도	중(상/중/하)
키워드	Zeroday attack, SBOM, Scan Tool, OSS 거버넌스		
출제배경	시장의 오픈소스 활용도 향상에 따라 발생 가능한 문제점과 해결책 SBOM 개념 이해 확인		
참고문헌	ITPE 서브노트 https://www.oss.kr/oss_guide/show/9d25996e-ade5-47af-9406-5b99bd03f10f https://blog.naver.com/softwidesec/223498890174		
해설자	NS반 멘토 백현 기술사(제 122회 정보관리기술사 / snuoo@naver.com)		

I. 오픈소스 취약점을 최소화 방안, SBOM(Software Bill of Materials)의 개요



정의	소프트웨어 패키지 및 구성 요소 등 고유하게 식별 가능한 메타데이터, 저작권 및 라이선스 등 소프트웨어 콘텐츠에 대한 정보를 포함하는 공식 명세서

- OSS를 사용하면 Time-to-market 실현이 가능하나, 라이선스 이슈 등의 문제가 발생할 수 있어 대책 필요

II. 오픈소스 소프트웨어 취약점

가. 관리적 측면의 오픈소스 SW 보안 위협

구분	보안 위협	설명
사용	- 오픈소스 SW 사용 현황 부재	- 자유로운 배포 특징으로 오픈소스 배포처에서 현황파악 불가 - 기업내 오픈소스 SW 사용 현황 관리 부실 빈번
	- 오픈소스 커뮤니티 소스코드 맹목적 사용	- Github 등 대형 오픈소스 커뮤니티 소스 맹목적 신뢰 - 타이포스쿼팅(Typosquatting) 오픈소스 SW 보안 위협 존재

프로세스	- 오픈소스 SW 취약점 점검 프로세스 부재	- 직접 개발한 SW와 비교하여 취약점 점검 수준 저하 - 오픈소스 SW의 패키지 형태 사용으로 점검 예외 빈번
	- 오픈소스 SW 취약점 패치 검토 및 조치 지연	- 오픈소스 SW 패치시 Side-Effect 검토 등 조치 지연 - 오픈소스 SW취약점 발생시 조치주체의 부재 및 모호성으로 인한 즉각적인 대응 부재

- 그 외 오픈소스 SW 보안 사고 책임 소재의 문제 및 직접 개발하지 않은 SW의 책임감 결여 문제 존재

나. 기술적 측면의 오픈소스 SW 보안 위협

구분	보안 위협	설명
공격	- 공개 SW로 Zero Day 공격 가능성 증가	- 소스가 공개 되어있는 특징으로 Zero Day 공격 가능 - 해커 커뮤니티 통한 취약 공격 코드 확산 가능
	- 오픈소스 SW 내 악성코드 삽입 배포	- 해커 원격 제어 가능(Remote Code Execution) 악성코드 배포 - 크립토재킹(Cryptojacking) 코드 주입 가상화폐 채굴에 악용
조치	- 오픈소스 SW 취약점 자체 조치 불가	- 오픈소스 SW 배포사의 취약점 패치 의존 - Work Around(임시 조치 방안)으로 대응 불가피
	- 오픈소스 SW 보안 취약점 패치의 호환성 문제	- 취약점 패치시 호환성 문제 발생으로 조치 불가 상황 가능 - 호환성 문제 해결시까지 보안 취약점 조치 지연 문제점

- 오픈소스 SW 보안 위협 관리 위해 전용 Tool 도입, 프로세스 수립, 시큐어 코딩 등의 관리 방안 필요

III. SBOM 기반 오픈소스 소프트웨어 관리 방안

가. 관리적 측면의 SBOM 기반 오픈소스 소프트웨어 관리 방안

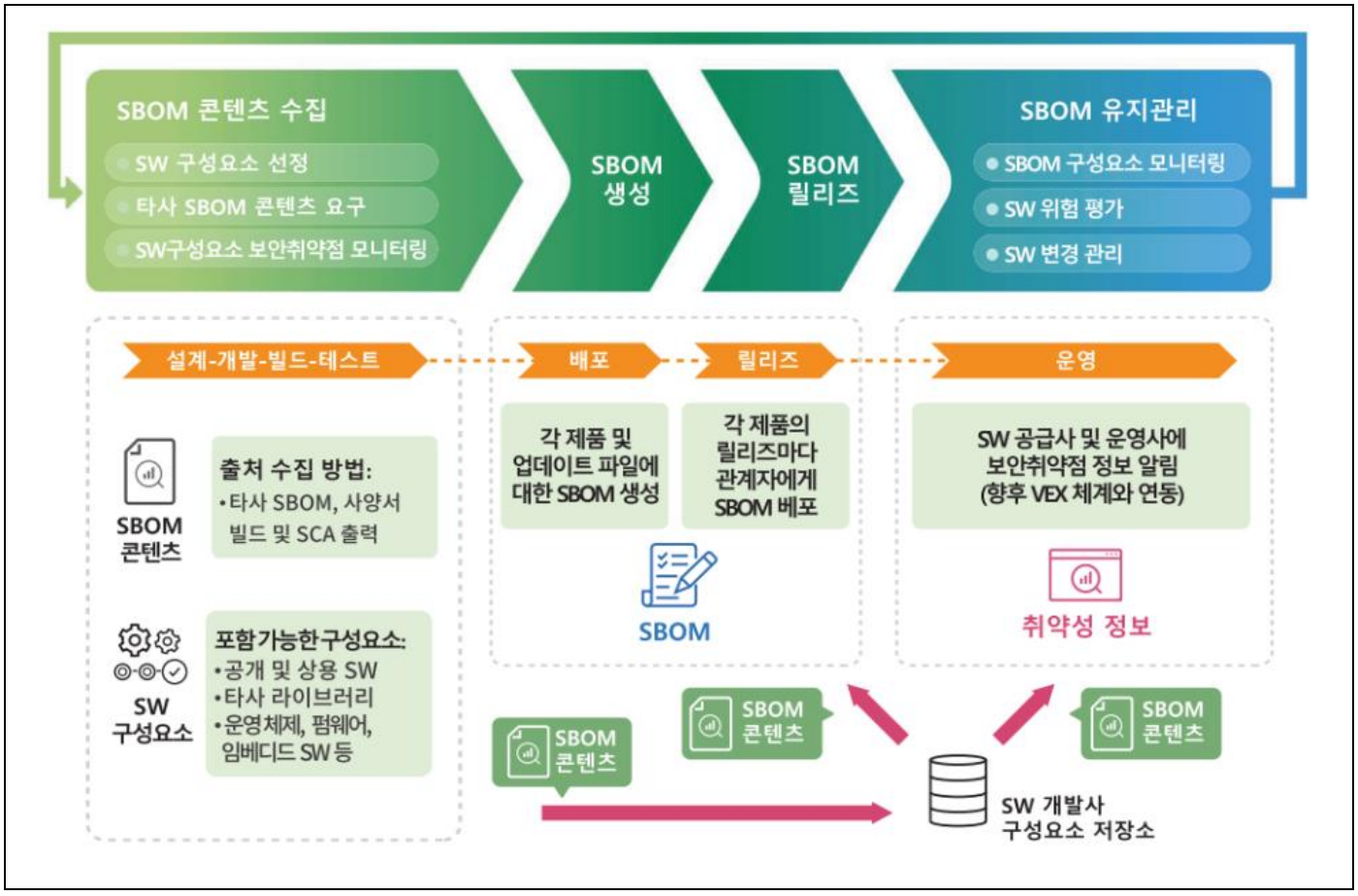
구분	관리방안	설명
SBOM 관리정책 수립	- OSS 라이선스 관리 정책의 적용 범위	- OSS 사용범위에 대해 정의를 해 내부/외부를 구분해 SBOM 적용 의무사항의 범위를 정의
	- SBOM 관리 R&R 수립	- 조직내.외부에 소프트웨어 개발 및 공급에 참여하는 부서 및 구성원에 대한 적절한 역할과 책임을 부여
	- 보안취약점 관리정책	- OSS의 유입에서부터 폐기 전 과정에 사용된 OSS를 추적하고 관리할 수 있도록 제반 점검과 절차 및 R&R에 대한 정책
프로세스	- SBOM 관리 절차 수립	- 조직내에 소프트웨어 개발 및 공급을 위한 기획, 구현, 검증, 제품화 절차에 부합되게 관련 부서 및 구성원에 대한 적절한 프로세스를 구축 및 운영
	- SBOM 기반 개발 절차 수립	- 오픈소스 전문 업체 통한 유지보수 및 보안 점검 수행

나. 기술적 측면의 SBOM 기반 오픈소스 소프트웨어 관리 방안

구분	관리방안	설명
개발	- SBOM 관리 속성 정의	- SBOM 검증도구, 공급자, 저작권자, 버전, CVE ID, 릴리즈 날짜와 같은 관리 속성을 정의해 적용
	- 배포 및 서비스에서 라이선스 관리	- SBOM을 통해서 배포 및 서비스에서 발생할 수 있는 라이선스 이슈를 최소화하기 위해 라이선스를 분류 및 관리
툴	- SBOM 생성 도구 정의	- 정적 분석도구, OSS 종속된 분석도구, 소스코드 분석도구와 같은 툴을 사용해 관리 속성을 식별해 SBOM을 생성 관리

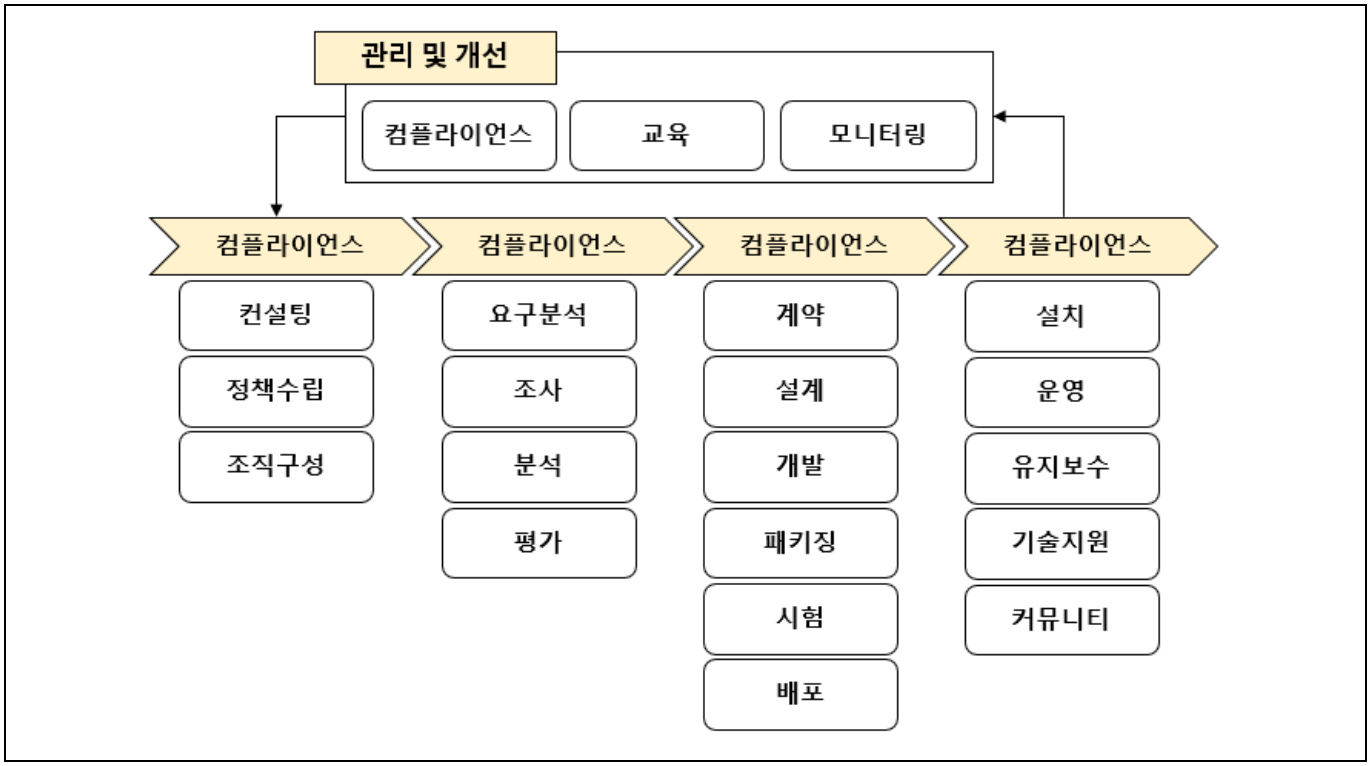
- 최근 아파치 Log4j 보안 취약점 발생에 따라 추가 해결방안으로 오픈소스 거버넌스 프레임워크 적용 필요

다. SW공급망 측면의 SBOM 기반 오픈소스 소프트웨어의 관리방안



- 공급사 및 운영사의 SW 공급망에 대한 SBOM 유통 체계를 구축해 검증된 정보로 OSS가 유통되도록 관리

IV. SBOM 기반 오픈소스 관리를 위한 체계, 오픈소스 거버넌스 프레임워크



- 오픈소스 소프트웨어 거버넌스 프레임워크 적용으로 SBOM 기반 오픈소스 소프트웨어를 SDLC 상 관리

"끝"

04	알고리즘의 복잡도		
문제	알고리즘의 복잡도를 설명하고 성능을 표기하기 위한 O-Notation의 개념과 유형 및 유형별 연산 시간의 차이를 설명하시오.		
도메인	알고리즘	난이도	중(상/중/하)
키워드	수행시간, 성능, 메모리, 빅오, 빅오메가, 빅세타		
출제배경	알고리즘의 개념 기본 확인		
참고문헌	ITPE 기술사회 자료집		
해설자	정상반멘토 정상 기술사(제 124회 정보관리기술사 / itpe_peak@naver.com)		

I. 알고리즘 복잡도의 설명

가. 알고리즘 복잡도의 정의

정의	알고리즘의 성능을 평가하기 위해, 수행 시간 및 메모리 사용량 등을 나타내는 척도	
특징	재귀적 알고리즘	- 자기 자신을 호출하는 알고리즘, 재귀 호출에 따라 스택 메모리 사용
	동적계획법	- 작은 하위 문제들을 해결하고 그 결과를 저장하여 전체 문제를 해결하는 알고리즘
	탐욕적 알고리즘	-작은 하위 문제들을 해결하고 그 결과를 저장하여 전체 문제를 해결하는 알고리즘

- 시간복잡도와 공간복잡도를 주로 이용

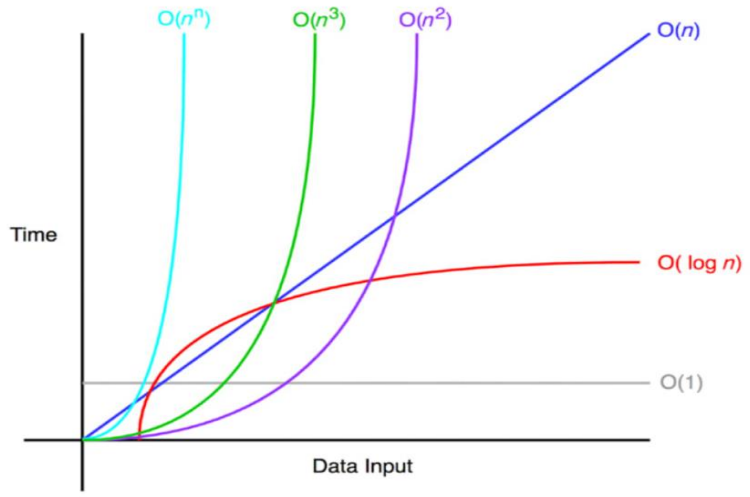
나. 알고리즘 복잡도의 유형

구분	설명
공간 복잡도	- 프로그램을 실행 및 완료하는데 필요한 저장공간을 확인하는 복잡도 - 고정 공간 : 알고리즘과 무관한 공간. 코드 저장 공간, 단순 변수 및 상수 - 가변 공간 : 알고리즘 실행과 관련있는 공간. 실행 중 동적으로 필요한 공간 - 표현 : 주로 O(빅오) 표기법을 이용
시간 복잡도	- 알고리즘의 수행 시간을 분석할 때, 기본 연산의 실행 횟수를 수행 시간을 사용하는 복잡도 - O(빅오) 표기법 : 입력데이터가 최악일 때를 기준으로 평가하기 위해 사용 - Ω(빅오메가) 표기법 : 입력데이터가 최상일 때를 기준으로 평가하기 위해 사용 - Θ(빅세타) 표기법 : 빅오, 빅오메가 둘 다를 포함하는 개념으로, 알고리즘 수행시간의 하한인 동시에 상한을 표시하는 방법

- 주로 빅오 표기법을 사용하며, 알고리즘 유형별 다양한 시간복잡도가 존재

II. O-Notation의 개념과 유형

가. O-Notation의 개념



정의	데이터 수 N에 대해 복잡도가 어떤 함수로 나타나는가를 간단히 표현하기 위한 알고리즘표현법

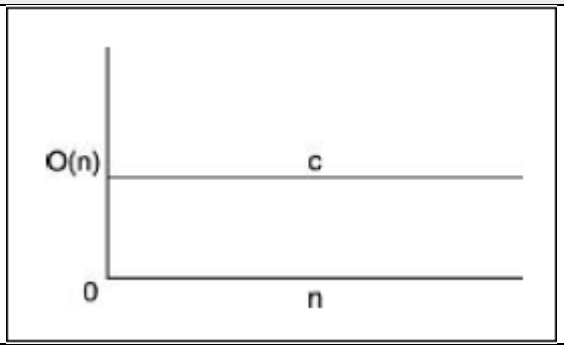
- 매개변수 N은 알고리즘에서 문제의 크기를 나타내며, 처리할 자료 항목의 수로서 수행 시간에 가장 많은 영영향을 끼침

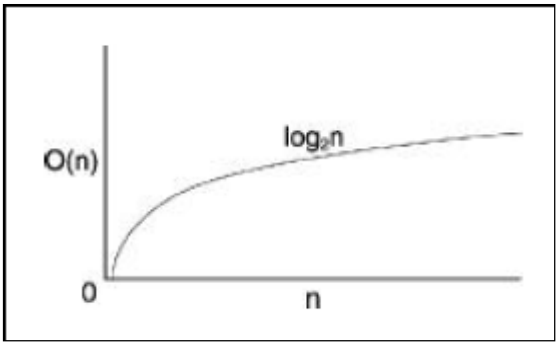
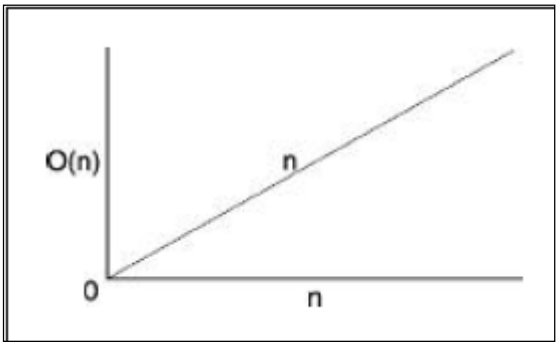
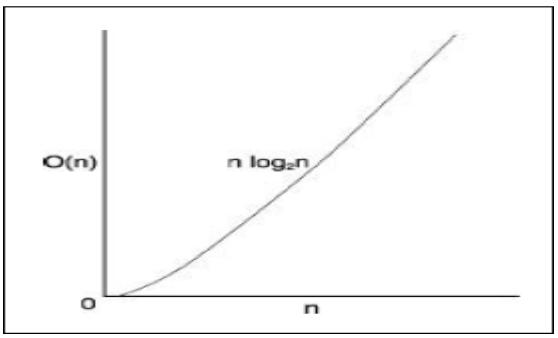
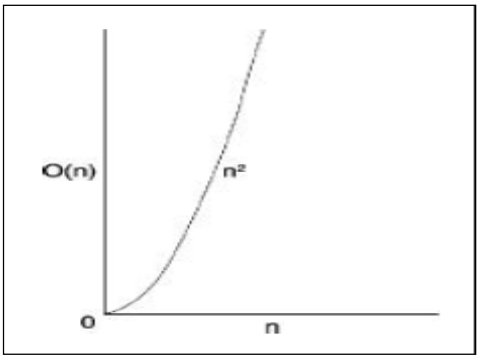
나. O-Notation의 유형

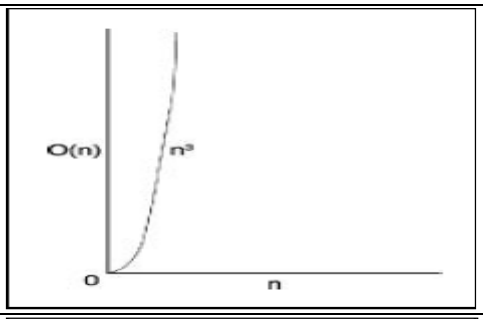
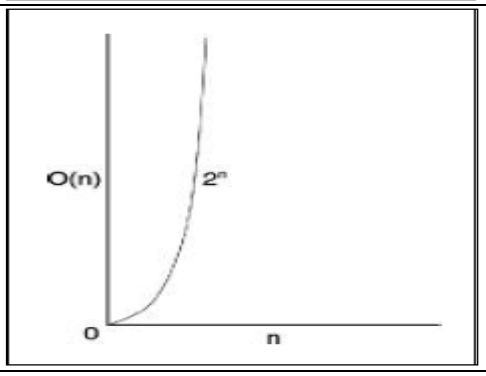
유형	설명	사례
O(1)	상수형, 입력 크기와 무관하게 바로 해를 구함	Hash Function
O(log N)	로그형. 입력 자료를 나누어 그 중 하나만 처리	이진탐색(Binary Search)
O(N)	선입. 입력 자료를 차례로 하나씩 모두 처리	단순탐색(Find Item)
O(N log N)	분할과 합병형, 자료를 분할하여 각각 처리하고 합병	퀵 정렬(Quick Sort)
$O(N^2)$	제곱형. 주요처리 (기본 연산) loop 구조가 2중인 경우	Bubble Sort
$O(N^3)$	세제곱형. 주요처리 (기본 연산) loop가 3 중인 경우	Finding the Shortest Path
$O(2^n)$	지수형. 가능한 해결방법 모두를 다 검사하여 처리함	Dynamic Programming

- $O(1) < O(\log N) < O(N) < O(N \log N) < O(N^2) < O(N^3) < O(2^n) < O(N!)$

III. O-Notation의 유형별 연산 시간의 차이

구분	설명	그래프
O(1)	- 상수 함수 - 알고리즘의 수행에 걸리는 시간이 자료집합의 크기에 상관없이 항상 동일함을 의미 - 대체로 이런 함수들이 가장 빠른 것으로 간주 - 상수 함수는 자료의 크기와 무관하게 항상 같은 속도로 작동	

O(log N)	- 기수가 2인 로그 함수(logarithm function, 또는 대수 對數함수). - 기수 2 로그에서 수직 성분은 자료집합의 크기가 두 배가 될 때마다 1 증가 - 예) 1의 로그는 0, 2의 로그는 1, 4의 로그는 3, 8의 로그는 4 - 로그기반 알고리즘은 자료의 크기에 의존적인 알고리즘들 중에서는 가장 효율적인 것으로 알려짐. - log n 함수는 데이터의 크기에 따라달라지나, 데이터가 추가될수록 더 효율적	
O(N)	- 선형(linear, 線形) 함수 - 자료 크기가 증가함에 따라 일정한 비율로 증가 예) 어떤 O(n) 알고리즘이 1000개의 항목을 20초에 처리한다면, 2000개의 항목은 대략 40초에 처리 - 선형 함수는 자료의 크기와 직접적인 관계로 변함. 데이터가 두 배 증가하면 계산 시간도 두 배로 증가.	
O(N log N)	- 일반적으로 정렬(sorting) 알고리즘이 가져야 할 최소한의 복잡도로 간주 - 앞에 나온 모든 함수들보다 비효율적이나, 이후 나올 좀 더 복잡한 함수들보다는 효율적이기 때문에 이 정도면 비교적 효율적이라고 간주됨. - 데이터의 크기에 따라 변하나, 기울기가 비교적 완만한 곡선을 가지기 때문에 효율적인 편	
$O(N^2)$	- 자료가 증가함에 따라 급격히 커지기 때문에 대부분의 과제들에 대해 비효율적이라고 간주 예) 항목 1000개를 20초에 처리하는 알고리즘이 항목 2000개를 처리하려면 80초가 걸리며, 항목 개수가 2배 증가하면 시간은 4배로 증가. - 전형적인 예는 for 루프 안에 다른 for 루프가 내포된 형태. - 기울기가 매우 급하므로 피하는 것이 효율적임.	

$O(N^3)$	- $O(N^2)$ 과 비슷해 보이나 증가율은 훨씬 더 큼 예) 1000개를 20초에 처리하는 알고리즘이 2000개를 처리하려면 160초가 필요하게 됨(8 배).	
$O(2^n)$	- 기수 2 지수 - 항목 개수가 1 증가할 때마다 함수는 2 배가 됨. - 매우 비효율적이므로 최대한 피해야 함. - 기수 2 지수 함수는 비효율적. 자료크기가 1 증가할 때마다 알고리즘을 수행하는 데 필요한 시간이 2 배로 늘어남.	

"끝"

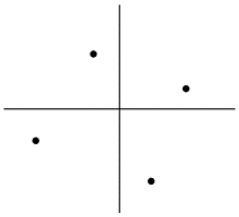
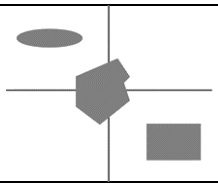
05	다차원 색인구조(Multidimensional Index Structure)		
문제	다차원 색인구조(Multidimensional Index Structure)의 개념, 유형, 활용 사례에 대하여 설명하시오.		
도메인	데이터베이스	난이도	중 (상/중/하)
키워드	PAM, SAM, k-d트리, k-d-b트리, GridFile, QuadTree, R-트리		
출제배경	이미지나 멀티미디어 데이터와 같은 비정형 데이터의 효율적 검색 및 활용 증가에 따른 이해		
참고문헌	ITPE 서브노트		
해설자	BP반 김찬일 기술사(제 130회 정보관리기술사 / s2carey@naver.com)		

I. 다차원 색인 구조의 개념

가. 다차원 색인 구조 정의

- 이미지나 멀티미디어 데이터와 같은 비정형 데이터의 효율적 검색을 위해 여러 개의 필드(애트리뷰트)를 동시에 키(key)로 사용한 색인구조

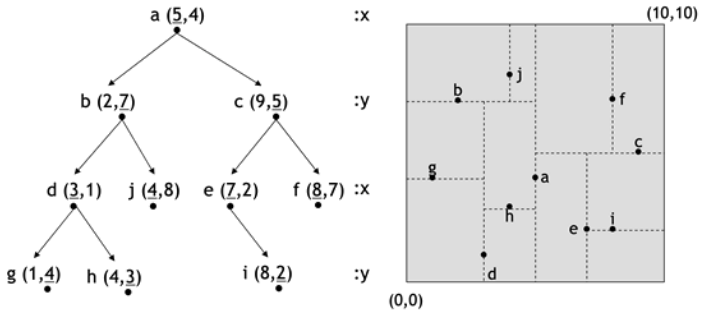
나. 다차원 색인 구조 유형

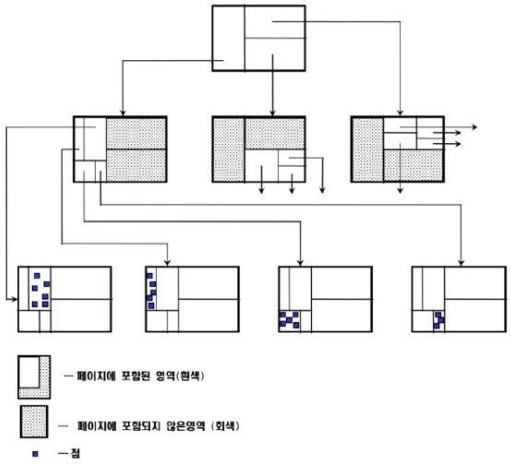
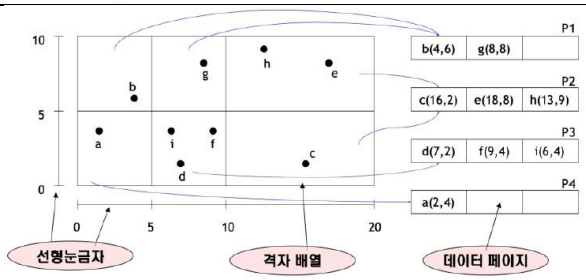
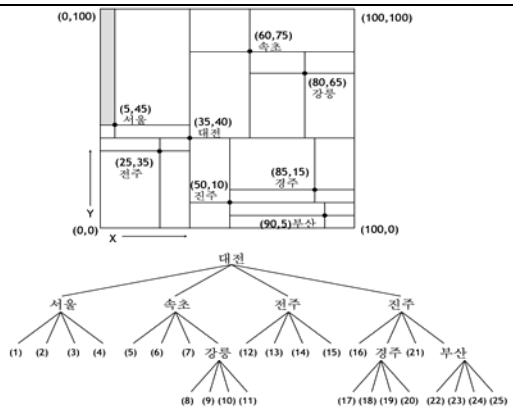
유형	개념도	다차원 색인구조	설명
PAM (Point Access Method)		k-d 트리, k-d-b 트리, Grid File, MLGF(Multi-Level Grid File), QuadTree	- 다차원의 점(Point) 데이터를 저장 및 검색
SAM (Spatial Access Method)		R-Tree, R⁺-트리, R*-트리	- 선, 면 등과 같은 크기를 갖는 다차원 데이터를 저장 및 검색

- 비정형 데이터의 유사성에 기반한 내용검색에 활용되며, 대표적으로 k-d트리, QuadTree, R-Tree가 활용

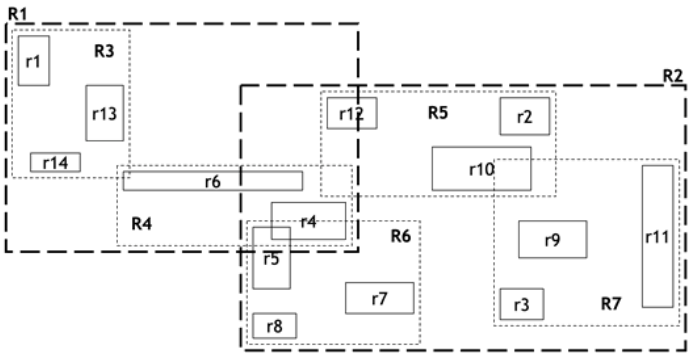
II. 다차원 유형 상세 설명

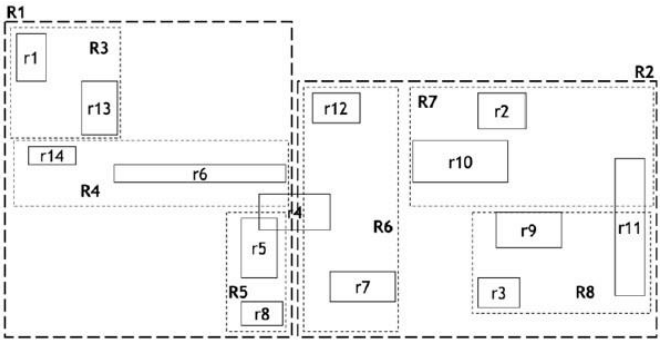
가. PAM 기반

유형	개념도	설명
k-d 트리		-Binary Search Tree를 다차원 공간으로 Straightforward하게 확장한 것 -기본 구조와 알고리즘은 Binary Search Tree와 유사하지만 트리의 레벨 차원을 번갈아 가며 비교 - 균형 트리가 아님 - 소규모 다차원 점 데이터 인덱싱에 적합

k-d-b 트리		- k-d 트리와 B-트리의 특성을 결합한 구조입니다. 이 트리는 다차원 공간에서 데이터를 효율적으로 저장하고 검색하기 위해 설계
Grid File		- 다차원 데이터의 효율적인 검색과 관리를 위해 설계된 파일 구조. 주로 다차원 데이터베이스에서 사용되며, 데이터 포인트를 다차원 그리드 공간에 매핑하여 관리
QuadTree		- 공간을 순환적으로 분해하는 계층적(hierarchical) 자료 구조 - 공간을 (크기가) 동일하지 않은 4개의 부속 공간으로 분할 - 한노드 탐색비용: O(log4N)

나. SAM 기반

색인구조	개념도	구조설명
R-Tree		설명 - R 트리는 B 트리와 비슷한데 다차원의 공간 데이터를 저장하는 색인 특징 - MBR(Minimum Bounding Rectangle) 이용 - 선, 면, 도형 등 다양한 다차원 공간 데이터의 저장 - 완전 균형 트리

R+-트리		- R+ 트리는 R 트리와 K-D-B 트리의 중간 형태로써 겹치는 데이터는 여러 노드에 중복하여 저장 하는 색인 자료구조
R*-트리	- 최소 경계 사각형(MBR, Minimum Bounding Rectangle)의 넓이와 다른 - MBR 과의 겹침(overlap) 영역의 최소화는 색인 자료구조	

- 다차원 색인구조는 공간, 멀티미디어 분야에서 활발하게 활용이 가능함.

III. 다차원 색인구조 활용 사례

가. 공간 데이터 관리 및 멀티미디어 분야

구분	활용사례	설명
공간 데이터 관리	- GIS(지리 정보 시스템)	- R-트리와 같은 구조를 사용하여 지리적 데이터를 효율적으로 저장, 검색, 분석
	- 위성 영상 분석	- Quad-트리 등을 사용하여 위성 이미지 데이터를 분할하고, 빠른 검색 및 분석을 지원
	- 도시 계획 및 관리	- Oct-트리 등을 사용하여 3D 공간 데이터를 관리, 건물 위치 및 도시 구조 분석
멀티미디어	- 이미지 데이터베이스 검색	- SS-트리, Quad-트리 등을 사용하여 이미지의 색상, 텍스처 등 특징을 기반으로 유사 이미지 검색
	- 비디오 검색	- k-d 트리 등을 사용하여 비디오 메타데이터의 다차원 인덱싱을 통해 관련 비디오를 빠르게 검색
	- 음악 데이터베이스 관리	- 다차원 색인구조를 사용하여 음악 특징(예: 주파수, 템포) 기반의 빠른 검색 및 추천

- 주로 지리 정보 시스템에서 자주 활용되며 멀티미디어 외에도 여러 분야에서 활용이 가능함.

나. 과학 데이터 분석 및 소셜네트워크 분석 분야

구분	활용사례	설명
과학 데이터 분석	- 기후 데이터 분석	- k-d-b 트리 등을 사용하여 다차원 기후 데이터를 분석, 특정 조건 하에서의 데이터 검색
	- 생물정보학	- 다차원 색인구조를 이용하여 유전자 데이터 등의 복잡한 생물학적 데이터를 분석 및 검색
	- 금융 데이터 분석	- 고차원 금융 데이터(예: 주가, 거래량)의 분석을 위해 다양한 다차원

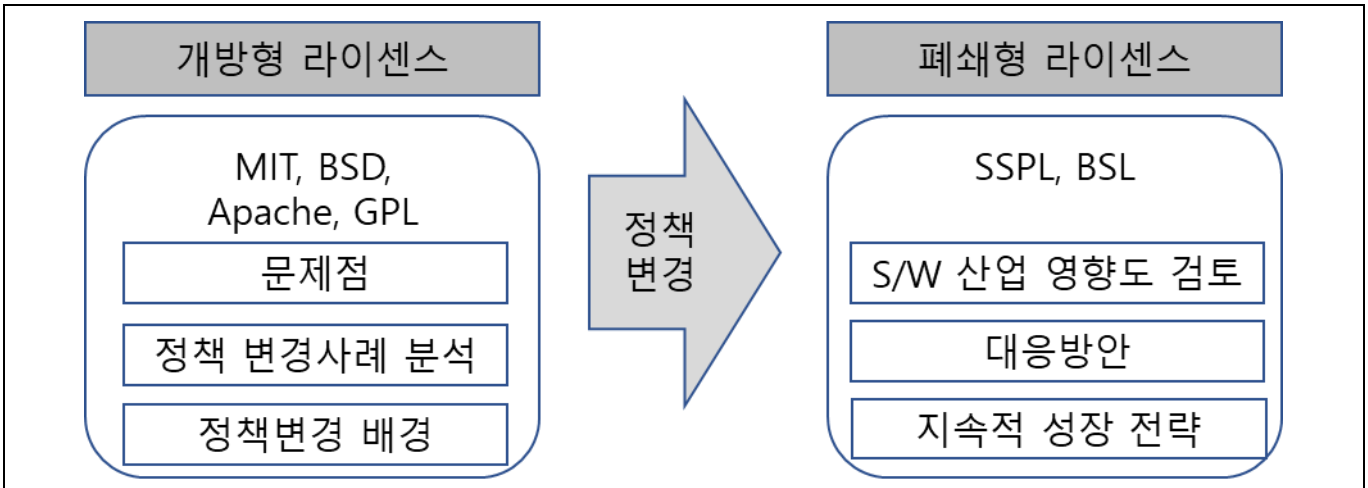
		색인구조 사용
소셜 네트워크 분석	- 소셜 미디어 데이터 분석	- 다차원 색인구조를 사용하여 소셜 미디어에서 발생하는 다양한 데이터(예: 좋아요, 댓글)의 분석 및 검색
	- 영향력 분석	- 사용자 활동, 관계 등을 다차원 데이터로 분석하여 영향력 있는 사용자나 주제를 파악

- 여러 분야에서 사용되며 적절한 유형을 활용분야에 적용하는것이 중요.

"끝"

06	오픈소스 라이선스 정책 변경		
문제	일부 오픈소스 라이선스가 개방형(예: MIT, BSD 등)에서 폐쇄형(예: SSPL(Server Side Public License), BSL(Business Source License) 등)으로 변화하고 있다. 이러한 오픈소스 라이선스 정책 변경의 배경 및 소프트웨어 산업에 미치는 영향에 대하여 설명하시오.		
도메인	소프트웨어공학	난이도	상(상/중/하)
키워드	오픈소스 소프트웨어, 개발형 라이선스, 폐쇄형 라이선스		
출제배경	오프소스 라이선스 정책 변경에 대한 이해와 배경		
참고문헌	오픈소스SW 기업의 라이선스 모델 전환 이슈: 주요 논쟁과 전망 - 공개SW 포털 https://www.oss.kr/oss_guide/show/0591b468-522c-4170-943d-8a1df5af81c9 https://www.oss.kr/oss_guide/show/33fa7773-1c5c-4d11-b07d-3c3e66b34b0f		
해설자	정주행 조종흥 기술사(제127회 정보관리기술사 / choheung@naver.com)		

I. 오픈소스 라이선스 정책변경에 대한 개요



구분	설명
개방형 S/W 라이선스 (Open Source S/W License)	소프트웨어 소스 코드를 공개하여 누구나 사용, 수정, 배포할 수 있도록 권한을 명시하여 허용하는 소프트웨어를 의미 각 라이선스는 특정한 법적 조건과 의무를 포함하고, 사용자는 이를 준수해야 함 MIT, Apache, GPL 등
폐쇄형 S/W 라이선스 (Close Source S/W License)	저작물의 사용, 수정, 배포를 엄격히 제한하는 라이선스. 저작권자가 저작물에 대한 독점적인 권리를 보유하며, 사용자에게 특정 조건 하에서만 저작물을 사용할 수 있는 권한을 부여 SSPL, BSL, Proprietary License 등

- 오픈소스 라이선스 정책 변경에 대한 배경과 그에 따른 사례를 분석할 필요 있음

II. 개방형에서 폐쇄형 라이선스로 정책 변경을 하는 배경과 사례

가. 폐쇄형 라이선스로 정책 변경을 하는 배경

구분	정책 변경 배경	설명

상업적 측면	수익 창출	무료로 배포되기 때문에 직접적인 수익 창출이 어려움 수익 손실을 보완하기 위해 폐쇄형 라이선스 전환
	비즈니스 모델 보호	소프트웨어의 독점적 사용권을 판매 구독 모델을 적용하여 안정적인 수익원 확보 지속 가능한 비즈니스 모델 유지
	투자 보호	기업이 개발한 기술과 노하우는 중요한 자산
	경쟁 우위 확보	기업 경쟁의 우위가 약화될 수 있음 폐쇄형 정책을 통해 핵심 기술을 보호, 시작에서 경쟁력을 유지
기술적/보안 측면	보안 강화	누구나 소스코드를 열람, 수정할 수 있기 때문에 잠재적인 보안 취약점 공개될 수 있음 악의적인 공격자들이 보안 취약점을 악용할 위험 존재 소스코드 비공개 유지하면, 보안 취약점 외부 노출 가능성을 낮추고, 내부적 보안 강화를 위한 조치를 취할 수 있음
	품질 관리 및 일관성 유지	다양한 기여자들로 인해 코드 품질이 일정하지 않거나, 일관성이 부족할 수 있음 소프트웨어의 안정성과 신뢰성에 영향을 미칠 수 있음 폐쇄형 정책을 통해 개발 프로세스와 코드베이스를 내부에서 엄격하게 관리 높은 품질의 소프트웨어를 유지할 수 있으며, 일관된 사용자 경험을 제공
지속 가능한 개발 모델 추구	지속 가능한 수익 모델	오픈소스 프로젝트의 유지보수와 발전을 위해서는 자금이 필요함. 개방형 라이선스만으로는 지속 가능한 수익 모델을 구축하기 어려운 경우가 많음
	직접적인 수익 창출	폐쇄형 라이선스를 통해 직접적인 수익을 창출하고, 프로젝트의 지속 가능성을 확보하려는 시도가 늘어나고 있음.

- 기업이 장기적으로 지속 가능한 성장을 이루고, 시장에서의 경쟁력을 유지하는 데 도움

나. 폐쇄형 라이선스 정책 변경한 기업의 사례

정책 변경 사례	변경 사항	설명
Redis의 듀얼 라이선스 전환	2024년 BSD 라이선스에서 RSALv2와 SSPLv1 듀얼 라이선스로 전환	대형 클라우드 서비스 제공업체의 독점적 활용 방지, 개발자 보상 강화 목적. RSALv2는 소스코드를 자유롭게 사용하되, 상업화 및 서비스 제공을 제한하고, SSPLv1은 상업적 활용을 허용하되 서비스의 전체 소

		스코드를 공개하도록 요구.
MongoDB의 SSPL 전환	2018년 AGPL 라이선스에서 SSPL로 전환	클라우드 서비스 제공자가 MongoDB를 서비스로 제공할 경우, 해당 서비스의 전체 소스 코드를 공개하도록 요구 클라우드 서비스 제공자의 무임승차 방지, MongoDB의 상업적 이익 보호 목적.
CockroachDB의 BSL 채택	BSL을 채택하여 특정 기간 동안 소프트웨어의 상업적 사용을 제한하고, 이후 개방형 라이선스로 전환	초기 개발 비용 회수. 장기적으로 커뮤니티의 참여 유도 전략.

- 개방형 라이선스를 폐쇄형 라이선스로 정책 변경에 따른 소프트웨어 산업에 미치는 영향을 분석 필요

III. 폐쇄형 라이선스 정책 변경이 소프트웨어 산업에 미치는 영향

가. 오프소스 커뮤니티와 기업의 활용 전략에 미치는 영향

구분	사례	영향
오픈소스 커뮤니티 분열	Redis의 라이선스 변경 **Redis Labs :** 2018년에 Redis 모듈에 대한 라이선스를 변경하여, SSPL(Server Side Public License)과 Commons Clause를 도입	**커뮤니티 반발:** . Redis의 일부 커뮤니티 멤버들은 새로운 라이선스가 오픈소스 정신에 어긋난다고 생각하여 반발. . 커뮤니티 내에서 분열을 일으켰고, 일부 개발자들은 Redis의 오픈소스 버전을 포크하여 새로운 프로젝트를 시작했음. **포크 프로젝트:** . Redis의 라이선스 변경에 반발하여, 일부 커뮤니티 멤버들은 RDBTools와 같은 포크 프로젝트를 시작 . 오픈소스 프로젝트가 라이선스 변경으로 인해 분열되는 대표적인 사례
	ElasticSearch의 라이선스 변경 **lastic N.V. :** 2021년에 ElasticSearch와 Kibana의 라이선스를 오픈소스 라이선스에서 SSPL로 변경	**커뮤니티 반발:** . 많은 커뮤니티 멤버들이 ElasticSearch의 라이선스 변경에 반발 . ElasticSearch 프로젝트의 신뢰성과 오픈소스 정신에 대한 의문을 불러일으켰음. **포크 프로젝트:** . AWS는 ElasticSearch의 포크 프로젝트인 OpenSearch를 시작했음. . 라이선스 변경에 반발한 커뮤니티와 기업들이 오픈소스 정신을 유지하려는 노력을 보여줌.
기업의 오픈소스 활용 전략의 변화	Oracle의 MySQL 인수 후 라이선스 변경	**기업의 비용 증가:** . MySQL을 사용하는 기업들은 상업적 라이선스 비

	Oracle : . 2010년에 Sun Microsystems를 인수하면서 MySQL의 소유권을 가지게 되었음. . MySQL의 라이선스를 더욱 엄격하게 관리 . 상업적 용도에 대한 유료 라이선스를 도입	용을 지불, 일부 기업들에게 비용 부담을 증가시켰음. **대체 소프트웨어로 전환:** . 일부 기업들은 MySQL의 라이선스 변경에 따라 MariaDB와 같은 대체 오픈소스 데이터베이스로 전환 . MariaDB는 MySQL의 포크로, 오픈소스 라이선스를 유지
	MongoDB의 SSPL 도입 2018년에 라이선스를 AGPL에서 SSPL로 변경	**클라우드 제공업체의 대응**: . AWS는 MongoDB의 라이선스 변경에 대응하여 DocumentDB라는 자체 서비스를 개발 . MongoDB의 라이선스 변경이 클라우드 제공업체의 전략에 직접적인 영향을 미친 사례 **기업의 선택 변화**: . MongoDB를 사용하던 기업들은 라이선스 변경에 따라 DocumentDB와 같은 대체 서비스를 고려하게 되었음. . 기업들이 새로운 라이선스 조건에 맞춰 소프트웨어 사용 전략을 재평가하도록 했음.

- 정책 변경은 기업들이 소프트웨어 활용 전략을 재평가하고 대체 솔루션을 찾게 만드는 중요한 요인이 됨

나. 오프소스 생태계의 다변화와 AI와 데이터 기술에 대해 미치는 영향

영향	사례	영향
오픈소스 생태계의 확장 및 혁신	TensorFlow와 PyTorch의 경쟁과 협력 두 개의 주요 오픈소스 딥러닝 프레임워크로, AI 연구와 개발에 중요한 역할을 하고 있음 구글의 TensorFlow와 페이스북의 PyTorch는 서로 경쟁하면서도, 오픈소스 생태계를 통해 협력	**혁신 촉진:** . 두 프레임워크 간의 경쟁은 기능, 성능, 사용 편의성 등의 측면에서 지속적인 혁신을 촉진 . AI 연구자들과 개발자들에게 더 나은 도구와 기술을 제공하게 됨 **커뮤니티 기여:** . TensorFlow와 PyTorch는 모두 강력한 오픈소스 커뮤니티를 가지고 있으며, 많은 개발자들이 기여 . 프레임워크의 발전을 가속화하고, 다양한 분야에서의 응용을 가능하게 함
	Apache Spark의 성장 Apache Spark는 대규모 데이터 처리 및 분석을 위한 오픈소스 엔진으로,	**생태계 확장:** . Spark는 다양한 데이터 처리 및 분석 도구와 통합되어, 오픈소스 생태계를 확

	데이터 기술 분야에서 중요한 역할 Spark는 다양한 데이터 소스와의 통합, 빠른 처리 속도, 사용 편의성 덕분에 빠르게 성장	장하는 데 기여 . 데이터 과학자들과 엔지니어들이 통합된 환경에서 작업할 수 있도록 함. **다양한 응용 분야:** . Spark의 강력한 기능은 금융, 헬스케어, 리테일 등 다양한 산업 분야에서의 데이터 분석 및 처리를 가능 . 오픈소스 기술의 적용 범위를 넓히고, 혁신을 촉진하고 있음.
AI 및 데이터 기술의 접근성 및 발전	Hugging Face의 Transformers 라이브러리는 자연어 처리(NLP) 모델을 쉽게 사용할 수 있도록 하는 오픈소스 라이브러리 BERT, GPT-3, GPT-4 등 최신 AI 모델들을 간편하게 활용 가능	**접근성 향상:** . Transformers 라이브러리는 복잡한 AI 모델을 쉽게 사용할 수 있도록 하여, 연구자와 개발자뿐만 아니라 일반 사용자들도 AI 기술에 접근할 수 있게 함. . AI 기술의 대중화에 기여함. **커뮤니티 및 협업:** . Hugging Face는 활발한 오픈소스 커뮤니티를 통해 지속적인 업데이트와 개선 . AI 기술의 발전을 가속화하고, 다양한 분야에서의 응용을 가능하게 함.
	Google의 데이터셋 공개 정책 구글은 다양한 고품질 데이터셋을 오픈소스로 공개하여 연구자들과 개발자들이 이를 활용할 수 있도록 지원	**연구 촉진:** . 공개 데이터셋은 연구자들이 새로운 알고리즘과 모델을 개발하고 테스트할 수 있는 기회를 제공 . AI 및 데이터 과학 연구의 발전을 촉진 **공정성 및 윤리성 강화:** . 구글은 데이터셋 공개와 함께 데이터의 공정성과 윤리성을 강조 . AI 모델의 편향성을 줄이고, 공정하고 투명한 AI 기술 개발을 지원

- 오픈소스 생태계의 다변화와 AI 및 데이터 기술의 발전은 서로 밀접하게 연결되어 있고 생태계의 확장과 AI 및 데이터 기술의 접근성을 높이고, 기술 발전을 촉진하는데 중요한 역할을 하고 있음.

IV. 폐쇄형 라이선스 정책 변경에 대응 방안과 지속 발전을 위한 전략

구분	핵심 사항	설명
대응 방안	커뮤니티 유지 및 지원	- 기존 오픈소스 커뮤니티와의 지속적인 소통 강화 - 커뮤니티 기여자들에게 감사 표시 및 인센티브 제공 - 커뮤니티 포크 프로젝트에 대한 협력 및 지원
	고객 및 사용자 교육	- 라이선스 변경에 대한 명확한 설명 및 교육 제공

		- 라이선스 변경으로 인한 이점과 필요성 강조 - 자주 묻는 질문(FAQ)과 가이드라인 제공
	대체 솔루션 제공	- 기존 오픈소스 버전의 유지보수 및 지원 제공 - 유료 서비스와 무료 서비스의 명확한 구분 - 대체 가능한 오픈소스 솔루션 소개 및 지원
지속 발전 전략	혁신적인 기능 개발 및 제공	- 사용자 요구사항 반영한 혁신적인 기능 개발 - 차별화된 기능과 성능 제공을 통한 경쟁력 강화 - 정기적인 업데이트 및 패치 제공
	강력한 기술 지원 및 서비스	- 유료 사용자에게 24/7 기술 지원 제공 - 전문 엔지니어 팀을 통한 신속한 문제 해결 - 다양한 서비스 플랜 제공(예: 기본, 프리미엄)
	파트너십 및 협력 강화	- 주요 기업 및 기술 파트너와의 협력 강화 - 공동 마케팅 및 기술 개발 프로젝트 추진 - 산업 표준화 기구와의 협력
	교육 및 훈련 프로그램 제공	- 사용자 및 개발자 교육 프로그램 운영 - 온라인 및 오프라인 워크숍, 세미나 개최 - 자격증 프로그램 도입
	사용자 피드백 수집 및 반영	- 정기적인 사용자 설문조사 및 피드백 수집 - 피드백을 반영한 제품 개선 및 기능 추가 - 사용자 요구사항에 대한 신속한 대응

- 기업은 라이선스 변경으로 인한 부정적인 영향을 최소화하고, 장기적인 발전을 도모하기 위해 대응방안과 지속적 발전전략을 수립해야 함.

"끝"

ITPE 기술사회
제134회 정보처리기술사 기출문제 해설집

대 상 정보관리기술사, 컴퓨터시스템응용기술사, 정보통신기술사, 정보시스템감리사 시험
발행일 2024년 07월 27일
집 필 강정배PE, 전일PE, 백현PE, 조종흥PE, 정상PE, 김찬일PE
출 판 **ITPE(Information Technology Professional Engineer)**
주 소 ITPE 대치점 서울시 강남구 선릉로 86길 17 선릉엠티빌딩 7층
ITPE 선릉점 서울시 강남구 선릉로 86길 15, 3층 IT교육센터 아이티피이
ITPE 강남점 서울시 강남구 테헤란로 52길 21 파라다이스벤처타워 3층 303호
ITPE 영등포점 서울시 영등포구 당산동2가 하나비즈타워 7층 ITPE
연락처 070-4077-1267 / itpe@itpe.co.kr