KAIST 전산학부 입시학습 자료

전산학 주요 교과목에 대한 석사과정 구술면접 준비를 위한 학습 보조자료입니다. 각 과목의 핵심 키워드와 함께 구술면접에서 나올 가능성이 있는 질문들을 교과목별로 정리하였습니다. 2014년에 공개된 자료를 바탕으로 2024년에 전산학부 전임교원의 피드백을 받아 업데이트 하였습니다. 본 자료는 전산학부 **기초과목 입시 준비를 위한 학습 보조자료로만** 활용하시기 바랍니다. 아래 내용을 참고하여 교과서 내용을 리뷰를 체계적으로 하는데 도움이 될 것으로 기대됩니다.

전산학부 입시위원회

2024.8.1

Programming Language / Compiler

〈Keywords〉

- Syntax and Semantics
- Identifiers
- First-order Function, First-Class Function, Recursion
- Mutable Data Structure, Variables
- Lazy Evaluation
- Continuations, Low-Level Representation of Interpreters
- Memory, Garbage Collection
- Types, Type Checking, Typed Recursion
- Variants
- Type Inference
- Parametric Polymorphism, Subtype Polymorphism
- Type Soundness

〈Preliminaries〉

- 자바에서 public, protected, private 키워드가 있는데 아무것도 안 쓸 경우 default로 적용되는 범위는? 그렇게 디자인된 이유는?
  package-private: 같은 패키지 내에서만 접근 가능. 캡슐화를 강화하고 외부에서 변수가 접근되는 것을 막아 안전한 설계를 유도하기 위함.
- C++에서 서로 다른 타입의 오브젝트를 가리키는 포인터를 사용할 수 있나?
  void 포인터를 이용해 형변환하면 가능.*
- 전역 변수 사용의 장단점은?
  장점: 접근이 용이함, 함수 스택에 관계 없이 동일한 값 유지
  단점: 값의 예측 불가능성 (코드 여러 곳에서 접근하므로), 변수 이름 충돌 가능성

〈Function〉

- Higher-order function이 무엇인지 설명하고, 현대 프로그래밍에서 이 개념을 활용하여 쓰이는 대표적인 함수를 제시하시오.
  하나 이상의 함수를 인자로 받거나 함수를 반환하는 함수. 대표적인 예시로는 리스트의 각 원소에 함수를 적용하는 함수인 map()이 있음.

〈Lazy Evaluation〉

- 파라미터 패싱 방식에는 eager evaluation 방식과 lazy evaluation 방식이 있다. 두 방식 의 차이점을 비교 설명하세요. call-by-value와 call-by-name 파라미터 패싱 방식은 각 각 어느 방식에 속하는지 구분하세요.
  Eager evaluation: 함수 호출시 매개변수를 먼저 evaluate 함. 직관적임. Call-by-value 방식이 이에 해당됨.
  Lazy evaluation: 함수 내에서 인자가 실제로 사용될 때 evaluate 함. 경우에 따라 불필요한 evaluation을 막을 수 있음. Call-by-name 방식이 이에 해당됨.

〈Memory, Garbage Collection〉

- Rust와 Java는 C/C++를 대체하여 시스템 프로그래밍을 쉽게 하기 위해 대두된 대표적인 언어들이다. 메모리 관리 측면에서 기존 C/C++의 한계를 제시하고, 두 언어가 그 한계를 각각 어떻게 극복했는지 설명하고, 어떤 장단점이 있는지 설명하시오.
  C/C++에서는 메모리 관리를 개발자가 직접 수행해야 하므로 메모리 누수, 댕글링 포인터와 같은 문제가 빈번하게 발생.
  Rust: Ownership 시스템과 M (xor) A를 통해 컴파일 타임에 메모리 안전성을 보장. 가비지 컬렉션 오버헤드가 없고 메모리 안전성이 확실히 보장되지만, 러닝 커브가 있다는 단점이 있음.
  Java: 자동 GC를 통해 메모리 관리. 런타임 단계에서 메모리를 자동으로 해제해 관리 단순화. 개발이 편해지지만 GC 오버헤드로 인한 성능 저하 문제가 있음
- RAII(Resource acquisition is initialization)가 무엇이고, 어느 상황에서 사용하면 좋을지 설명하시오.
  RAII는 자원 획득이 초기화라는 의미로, 객체가 생성될 때 필요한 자원을 획득하고, 객체의 생명주기가 끝날 때 자원을 자동으로 반환하는 디자인 패턴임. 동적 메모리 할당, 파일 입출력, Lock 관리 등에 유용하게 쓰일 수 있음.
- 가비지 컬렉션을 위해 Mark and Sweep 기법이 크게 어떻게 이루어지는지 간단히 설명하고, 이 기법의 장단점을 설명하시오.
  Mark and Sweep은 접근 가능한 객체를 찾아 표시하는 Mark 단계와 표시되지 않은 객체를 메모리에서 해제는 Sweep 단계를 반복해서 수행함. 자동으로 메모리 누수를 방지하지만, 성능 저하가 우려되고 memory fragmentation이 일어날 수 있음.

〈Type System〉

- Type system이 sound하다는 것, 그리고 complete하다는 것의 의미를 각각 설명하시오.
  Soundness: 프로그램이 타입 검사를 통과하면 실행 시 타입 오류가 발생하지 않음.
  Completeness: 타입 검사시 올바른 프로그램은 모두 통과됨. 타입 시스템이 과도하게 엄격하지 않음을 보장.
- 프로그램 실행 전 컴파일 시에 타입을 정적으로 검사하는 프로그래밍 언어가 그렇지 않은 프로그래밍 언어에 비해 갖는 장점을 설명하시오. 또한, 그럼에도 불구하고 그렇지 않은 언어가 사용되는 이유를 설명하시오.
  프로그램을 먼저 컴파일 해 건전성 (soundness) 확보. 안정적이고 코드의 가독성이 높으며, 컴파일 단계에서 최적화를 통해 성능 향상을 기대할 수 있음. 동적 타입 검사는 유연한 타입을 가지고 있어 더 빠른 개발이 가능.
- Static typing과 dynamic typing의 차이점은? C++/JAVA은 어떤 typing을 사용하는가?
  Static typing은 컴파일 시점에 변수의 타입이 결정되어 있으나, Dynamic typing은 런타임 시점에 결정됨. C++/JAVA는 모두 Static Typing 언어.
- Option type이 무엇이고, 어느 상황에서 사용하면 좋을지 설명하시오.

함수에서 반환 값이 값일 수도 있고, 아닐 수도 있음을 표현. 값이 없을 수 있는 경우, Null pointer 에러 방지, 명확한 에러 처리에 유용하게 쓰일 수 있음.

<Compiler>

- 중간 표현(Intermediate Representation, IR)은 컴파일러 설계에서 매우 중요한 요소이다. IR이 사용되는 이유를 간단히 설명하라.

  추상화: 소스 코드의 언어적 세부 사항을 추상화하여 컴파일러의 다른 단계에서 공통적으로 사용할 수 있는 표준화된 표현을 제공.
  최적화 용이성: IR는 프로그램의 구조를 명확하게 나타내기 때문에 코드의 분석과 최적화가 용이함 (Dead Code Elimination, 루프 최적화, 상수 전파 등).
  타겟 독립성: IR는 특정 하드웨어나 아키텍처에 종속되지 않기 때문에, 다양한 타겟 플랫폼에 대해 동일한 IR에서 여러 형태의 머신 코드를 생성할 수 있음.

<기타>

- 아래 두 Javascript 코드(code1.js 및 code2.js)의 실행 결과가 다른 원인을 설명하고, 그 원인 각각의 장단점을 설명하시오.

// code1.js	**// code2.js**
let l1 = [1, 2]; let l2 = [100, 200]; let s = 0; for (var x of l1) { for (var x of l2) { } s += x; } console.log(s) **// Result: 400**	let l1 = [1, 2]; let l2 = [100, 200]; let s = 0; for (let x of l1) { for (let x of l2) { } s += x; } console.log(s) **// Result: 3**

var는 선언된 스코프 외부에서도 참조 가능하여 내부 loop에서 값이 200으로 변함.
let은 블록 스코프를 가지므로 더해지는 x는 1과 2임.
var를 이용하면 유연한 코드 작성이 가능하지만 의도하지 않은 값 변경이 쉽게 일어날 수 있어 유지보수를 어렵게 만듦. let을 이용하면 유연성은 떨어지지만 변수의 사용 범위가 명확해지고 버그의 원인을 쉽게 찾을 수 있음.

- 간단한 표현식 1 + 2 * 3 을 중심으로, Operational semantics와 Denotational semantics의 차이를 설명하시오.

  *Operational Semantics: 프로그램의 실행 과정을 통해 의미를 정의함. 2 * 3을 계산해 6을 얻고 1 + 6을 계산해 최종 결과 7을 얻음.*
  *Denotational Semantics: 프로그램의 의미를 수학적 함수로 정의함. +와 *는 각각 정수 덧셈과 곱셈을 나타내는 함수로 정의됨. 표현식을 함수 f(x, y, z) = x + y * z로 나타내고 값을 대입해 7을 얻음.*

- 아래의 코드에서 sum 변수에 배열 array의 모든 요소를 더하는 작업을 수행한다. 컴파일러가 이 루프를 최적화할 수 있는 방법을 제안하고, 이 방법이 성능 향상에 어떻게 기여하는지 설명하라.

```
int sum = 0;
for (int i = 0; i < 1000; i++) {
    sum += array[i];
}
```

"루프 언롤링(Loop Unrolling)" 기법을 사용할 수 있음. 이는 반복 횟수를 줄이고, 각 반복에서 더 많은 작업을 수행하도록 루프 본체의 코드를 복사하는 최적화 기법임. 이를 통해 루프의 반복 제어 오버헤드를 줄일 수 있으며, 메모리와 캐시 효율성을 향상시킬 수 있음.

```
int sum = 0;
for (int i = 0; i < 1000; i += 4) {
    sum += array[i];
    sum += array[i + 1];
    sum += array[i + 2];
    sum += array[i + 3];
}
```

- 다음은 프로시저 *read(a, b)*의 의사코드이다.

```
c := 3.0 * a + b;
if c = 0 { a := 1; } else { a := 1.0/c + 1.0/b; }
```

이 프로그램은 특정 조건을 만족하는 a값과 b값이 들어오면 오류를 일으킨다. 다음 중 프로그램의 오류를 막을 수 있는 가장 약한 조건은?

1) b > 0
2) a>0 and b>0
3) a ≠ -b/3
4) b ≠ 0
5) 3.0*a ≠ 0 and b ≠ 0

*4) b ≠ 0. 이 조건이 적용된다면 1.0/b과1.0/c (c = 3.0 * a + b ≠ 0일 때)을 계산할 때 division-by-zero가 생길 수 없음.*

Architecture

<Keywords>

- ISA (Instruction set architecture)
- Pipeline Architecture
- Control Hazard
- Memory Hierarchy, Cache
- Virtual Memory
- ILP (Instruction level parallelism): Superscalar, Out-of-order execution
- Multiprocessor, Cache coherence
- Multi-core Architecture
- I/O system

<ISA (Instruction set architecture)>

- CISC와 RISC의 차이점은 무엇인가?
  CISC: 복잡한 명령어 세트와 다양한 주소 지정 모드 등으로 다양한 기능을 제공함. 한 명령어로 복잡한 작업을 수행할 수 있음. 코드 밀도가 높고 컴파일러 설계가 비교적 단순함.
  RISC: 단순한 명령어 세트와 고정된 명령어 길이를 사용하여 파이프라이닝과 병렬 처리를 최적화하고, 전력 소모와 열 발생을 줄임. 효율성과 성능을 극대화해 최종 성능을 높임.
- Register File과 Cache의 차이는 무엇인가?
  Register File: CPU 내부에 위치함. 즉시 사용할 데이터와 명령어를 저장하며 접근 속도가 매우 빠름.
  Cache: CPU와 메인 메모리 사이의 중간 저장소로 자주 사용되는 데이터를 임시로 저장하여 메모리 접근 시간을 줄임. Register File보다 용량이 크지만 접근 속도는 상대적으로 느림.
- Intel과 AMD가 만드는 CPU의 공통점과 차이점은 무엇인가?
  두 CPU 모두 x86 아키텍처를 기반으로 만들어져 매우 유사한 Instruction set을 제공함. 명령어 처리를 위해 구현된 마이크로아키텍처가 다름.

<Pipelined Architecture>

- 파이프라이닝은 왜 하나요?
  CPU의 명령어 처리 효율을 높이기 위해 사용됨. 명령어를 여러 개의 작은 단계로 나누어 동시에 처리함으로써 Clock Frequency를 높이고 결론적으로 IPC(Instructions per Cycle)을 높임.
- 파이프라인 해저드(Pipeline Hazard)란 무엇인가요? 그리고 종류에는 무엇이 있나요?
  파이프라인 해저드(Pipeline Hazard)는 파이프라인의 정상적인 명령어 흐름을 방해하는 상황을 말함.
  데이터 해저드: 명령어 간의 데이터 종속성으로 인해 발생
  제어 해저드: 분기 결과와 다른 명령어가 파이프라인에 유입
  구조적 해저드: 하드웨어 자원 충돌로 인해 발생
- 파이프라이닝에서 Branch Prediction의 역할은 무엇인가?
  조건 분기 명령어의 결과를 예측하여 제어 해저드로 인한 파이프라인 스톨을 최소화하고 CPU의 처리 효율성을 높임.

〈Memory Hierarchy〉

- 캐시가 필요한 이유는? Cache hit ratio 에 대해 설명하시오.
  메모리 접근에 상당한 시간 소요 => CPU 내부에 데이터 저장소(캐시)를 만들어 자주 쓰이는 값을 저장
  원하는 데이터가 Cache에 있는 경우 Cache Hit라고 하며, 전체 데이터 접근 시도 대비 Cache Hit의 비율이 Cache Hit Ratio임.
- 메모리 접근하는데 x 사이클이 걸리고 캐시에 접근하는데 y 사이클이 걸리며 캐시 hit rate 가 h %일 때 effective access time은?
  *EAT = y + (1 - h/100) * x*
- 페이지 폴트는 언제 발생하는가? 페이지 폴트 비율과 cache miss 비율 중 큰 것은? 그 이유는?
  메모리에 없는 페이지에 접근하려 할 때 Page Fault 발생. 캐시가 메모리에 비해 더 빈번하게 접근되고, 캐시의 공간 크기가 페이지에 비해 매우 작으므로 cache miss가 더 빈번하게 일어남.

〈Virtual Memory〉

- 캐시 메모리와 메인 메모리의 주소 지정 방식의 차이점이 무엇인가?
  일반적으로 캐시 메모리는 Logical Address로 동작하는 반면, 메인 메모리는 Physical Address로 작동함.

〈ILP (Instruction Level Parallelism)〉

- Superscalar와 Out-of-Order execution(OoO)의 모두 CPU의 성능을 높이기 위한 기법이다. 두 기법의 공통점과 차이점에 대해서 설명하시오.
  두 기법 모두 명령어 레벨 병렬성(ILP; Instruction-Level Parallelism)을 활용하여 동시에 여러 명령어를 처리함으로써 성능을 높임.
  Superscalar: 여러 개의 파이프라인을 이용해 2개 이상의 명령어를 동시에 실행할 수 있게 함.
  OoO: 명령어의 의존성을 분석하여 순서에 상관없이 독립적인 명령어를 먼저 실행함. 명령어의 동적 재정렬을 통해 더 높은 효율성을 달성합니다.
- 슈퍼스칼라와 VLIW(Very Long Instruction Word)의 차이점은 무엇인가?
  두 기법 모두 여러 명령어를 병렬로 실행하여 성능을 향상시킴.
  VLIW: 컴파일러가 컴파일 시점에 명령어를 병렬로 묶어 긴 명령어 단위로 만듦. 하드웨어는 컴파일 결과를 그대로 실행함.

〈Cache〉

- 캐시를 구성하는 컴포넌트에 무엇이 있는가? 각 컴포넌트는 어떤 역할을 하는가?
  태그 메모리: 캐시에 저장된 각 블록의 주소를 저장
  데이터 메모리: 메인 메모리에서 가져온 데이터를 저장
  제어 회로: 캐시 메모리 관리 및 Hit/Miss 판단
- 캐시에서 태그 매칭이 무엇이고 왜 필요한가?
  메모리 주소의 태그 부분을 캐시의 태그와 비교하는 것. 주소가 일치하는지 비교해 가져오고자 하는 데이터가 캐시에 있는지 확인함.
- 캐시에서 블록(라인) 크기를 크게 했을 때와 작게 했을 때 어떤 장단점이 있을까?
  크게 했을 때: Spatial Locality 활용 극대화, 크기에 따라 캐시 용량 낭비 및 성능에 악영향 (Miss일 때)
  작게 했을 때: 블록 가져오는 속도가 빨라짐, cache 교체 빈도가 증가함.
- Direct-mapped cache와 set-associative cache의 장단점은 무엇인가?
  Direct-mapped cache: 구조가 간단해 검색 속도가 빠름. 그러나 여러 주소가 하나의 블록으로 매핑되는 경우 지속적으로 Cache Miss 발생.
  Set-associative cache: 앞에서 설명한 Cache Miss 경우를 해결. 그러나 구조가 복잡하고 검색 속도가

느림.

- Write-through cache와 write-back cache에 대해서 설명해 보시오.
  CPU에서 쓰기 연산을 시행했을 때 캐시에 쓴 내용이 언제 메모리에 쓰일 것인지 결정.
  Write-through cache: 데이터가 쓰여질 때 메모리에도 쓰여짐.
  Write-back cache: 데이터가 쓰여진 캐시 블록이 교체 될 때 내용이 메모리에 쓰여짐.
- Write-through cache는 write buffer를 보통 사용하는데 이의 역할은 무엇인가?
  메모리 접근 시간이 캐시 접근 시간에 비해 느림 => Write buffer가 메모리에 대한 쓰기를 비동기적으로 처리해 CPU의 성능 저하를 최소화함.
- Cache에서 사용하는 replacement policy에는 어떤 것이 있는가?
  LRU(Least Recently Used), FIFO(First-In, First-Out), Least Frequently Used (LFU), Random 등

<Multiprocessor, Multi-core>

- 코어가 10개 있는 cpu에서 프로세스 하나를 균등하게 처리하면 이상적인 경우, 시간이 얼마나 줄어드는가? 그런데 프로세스에서 분할되지 않는 20%의 작업이 있다고 하면, 시간이 얼마나 줄어드는가? 이제 코어가 수백개, 혹은 무한개 있다고 하면 얼마나 줄어드는가?
  균등하게 처리하는 경우: 시간이 1/10배로 단축 (90% 감소)
  20%는 분할되지 않는 경우: 80%의 workload는 1/10배, 20%는 단일 코어가 처리 => 72% 감소.
  코어가 많아 질 수록 감소율이 80%로 근사.

OS

<Keywords>

- Kernel
- Process, Thread, Scheduling
- Virtual Memory
- Address Translation, Paging, Swapping
- Concurrency: Locks, Conditional Variable, Semaphore
- Storage
- File System, Crash Consistency

<Kernel>

- Kernel을 필요에 의해 어느정도 수정했다고 하자. 이 kernel이 제대로 작동하는지를 알기 위해서 어떤 test를 해야 하나?
  부팅 테스트, 파일 시스템 / 네트워크 / 메모리 관리 테스트, 성능 테스트, 회귀 테스트 등
- Interrupt란 무엇인가? interrupt를 두 종류로 나눈다면 어떻게 되는가? (software interrupt/hardware interrupt) 두 interrupt의 차이는 무엇인가? 두 interrupt의 handler 를 서로 구분해서 구현해야 하는 것이 좋은가, 아니어도 상관 없는가?
  Interrupt: 이벤트 처리를 위해 CPU가 현재 작업을 중단하고 다른 작업을 수행하는 것.
  SW interrupt: 프로그램 등이 인위적으로 발생시킨 interrupt. 시스템 콜 등을 활용하기 위해 사용.
  HW interrupt: 타이머, 입력 기기 등 CPU 외부의 장치들이 비동기적으로 발생시킨 interrupt.
  요구 사항과 우선 순위가 다르므로 handler는 구분해서 구현하는 것이 좋음.
- 시스템 콜과 인터럽트의 차이는? 인터럽트가 걸리면 어떤 일이 일어나고 처리 후에 어떻게 이전 상태로 돌아가는가?
  System Call: 소프트웨어가 커널의 서비스를 이용하기 위해 사용, 동기적으로 일어남.
  Interrupt: 하드웨어나 소프트웨어에서 비동기적으로 발생.
  인터럽트 발생 -> 현재 상태 (CPU 레지스터) 저장 -> 인터럽트 벡터 테이블 확인 -> 해당되는 핸들러 실행 -> 레지스터 값 복원 후 재실행

<Thread & Process>

- Thread와 process의 차이는 무엇인가?
  Process: 실행 중인 프로그램. 독립적인 주소 공간을 가짐.
  Thread: 프로세스 내에서 실행되는 작업 단위. 메모리와 자원을 공유함.
- IPC가 무엇인가?
  Inter-Process Communication은 프로세스가 서로 데이터를 교환하거나 협력해 작업을 수행할 수 있게 하는 메커니즘임. 메모리 공유, 파이프, 소켓 통신 등의 방법이 있음.
- Thread끼리 context switching 하는 과정에 대해 설명하시오. 같은 프로세스 내부의 thread들끼리 전환되는 것과 다른 프로세스간의 thread 까리 전환되는 것이 어떻게 다른가?
  현재 실행 중인 스레드의 CPU 레지스터 저장 -> OS가 다음 스레드 결정 -> 해당 스레드의 CPU 레지스터 로드 -> 새로운 스레드 실행. Process는 Thread와 달리 주소 공간을 공유하지 않으므로 참조하는 Page Table 변경 등의 추가적인 조치가 필요함.

- Non-preemptive scheduling이란?
  현재 실행되고 있는 Process가 종료될 때까지 다른 Process를 스케줄링하지 않는 방법. FCFS (First-Come, First-Served), SJF (Shortest Job First) 등의 알고리즘이 있음. Context switching으로 인한 오버헤드는 적지만, Starvation 문제가 발생할 수 있음.
- 요즘 많은 사람들이 각각 자신만의 스마트폰을 사용하고 있다. 이처럼 각 개인 스마트폰을 위하여 process system을 design하려고 한다. 스마트폰에서 각 application이 하나의 process로 독립해서 실행하는 것이 나을까? 아니면, 하나의 thread로 만들어져서 실행하는 것이 나을까? 어느 쪽이 나을지 결정하고, 그 이유 를 설명하라.
  애플리케이션 간 메모리 공유가 일어날 경우 개인정보 침해 및 보안 문제가 일어날 수 있다. 따라서 Process로 독립되어 실행되는 것이 더 좋다.

〈Concurrency〉

- Sequential program과 multithread program에서 error detection에 대한 차이점에는 어떤 것이 있나?
  Sequential program에서는 오류 감지가 상대적으로 단순함. 디버깅과 오류 추적이 한 흐름에서 이루어짐. 그러나 Multithread program에서는 Race condition 등으로 인해 오류가 간헐적으로 나타날 수 있고, 오류 발생 시점과 위치를 예측하기 어려움.
- Critical section이란?
  여러 스레드가 동시에 접근해서는 안되는 자원이나 데이터를 포함하고 있는 코드 영역. 한 번에 하나의 스레드만 접근하도록 허용해야 Race condition을 막을 수 있음.
- 세마포어의 개념은? 주요 연산 2가지는? 어떻게 구현해야 하는가?
  여러 프로세스가 공유 자원에 접근 하는 것을 제어하기 위한 동기화 도구. P 연산은 세마포어 값이 1 이상이 될 때 까지 대기한 후 값을 감소시키며, V 연산은 값을 1 증가시키고 대기 중인 프로세스를 깨움. Lock이나 Atomic operation을 이용해 값의 Atomicity를 보장해야 함.

〈Virtual Memory〉

- Virtual Address와 Virtual memory에 대해 설명하시오.
  Virtual Address: 프로세스가 사용하는 가상 주소로, 같은 주소더라도 프로세스에 따른 다른 Physical Address에 매핑됨.
  Virtual Memory: 프로세스가 쓰는 가상 주소 공간. 실제 메모리보다 더 큰 주소 공간을 갖고 있으며, 다른 프로세스에 간섭받지 않고 주소를 쓸 수 있음.
- 어떤 C program으로 작성되어 수행중인 process가 있다고 가정하자. C언어에서는 직 접적으로 주소를 변수에게 지정해 줄 수 있다. 만약 주소 1, 2, 3, 4에 변수를 잡아서 어떤 일을 수행하는 process라고 하자. 이 process를 한 시스템에 동시에 두번 수행 시켰다. 그랬을 때 한 프로세스가 1, 2, 3, 4 주소에 있는 변수를 바꿨을 때 다른 프로세스의 변수들에도 영향을 끼치는가?
  지정한 주소는 Virtual Address이다. 두 프로세스는 주소 공간을 공유하지 않으므로 다른 프로세스에는 영향이 가지 않는다.
- C로 숫자로 직접 입력된 주소를 참조하는 프로그램을 짜서 컴파일한 후, 두 개를 실행시켰다고 하자. 그러면 이 두 프로세스는 물리적으로 같은 곳을 참조하나? 만약 아니라면, 어떻게 서로 다른 물리적 공간을 참조할 수 있나?
  물리적으로 다르다. Virtual Address에 매핑된 Physical Address가 프로세스마다 다르기 때문에 가능하다.
- Logical address와 physical address의 차이는 무엇인가?

Logical address는 프로그램이 실행될 때 CPU에 의해 생성되는 주소로, MMU에서 Physical address로 변환됨. Physical address는 실제 메인 메모리에서 데이터를 저장하고 접근하는데 사용하는 주소.

- Logical address를 physical address로 바꾸어주는 hw가 무엇인가?
  MMU (Memory Management Unit)
- 프로세스는 가상 메모리 주소로 어떻게 실제 메모리 주소를 찾아가는가?
  페이지 번호를 페이지 테이블을 이용해 프레임 번호로 변환. 프레임 번호와 오프셋을 합쳐 회종 물리적인 주소를 만듬.
- OS의 캐시 메모리의 사이즈를 구하기 위한 프로그램을 어떻게 구현하면 되는가?
  배열의 크기를 점진적으로 증가시키며 순차적 메모리 접근 시간을 측정. 접근 시간이 급격히 증가하는 크기가 캐시 크기임.

〈Paging〉

- Paging이 무엇인가? paging을 할 때 어떻게 실제 메모리 주소에 데이터를 전송하는가? page table에는 어떤 항목이 저장되는가? page table은 어디에 저장되는가? page table이 메모리에 저장되면, paging을 할 때 메모리를 두 번 참조해야 되는데, 좀 더 빠르게 하는 방법은 없는가? TLB에는 어떤 항목이 저장되는가?
  가상 메모리를 동일한 크기의 "페이지"로 나누어 관리하는 것. 가상 메모리 주소에서 페이지 번호를 프레임 번호로 변환, 가상 메모리 주소의 오프셋과 합쳐서 물리 메모리 주소로 바꿈.
  Page table에는 페이지 번호와 대응되는 물리적 프레임 번호와 상태 비트 (ex) valid bit)가 있음. 일반적으로 메모리에 저장되며, 자주 참조되는 주소는 TLB(Translation Lookaside Buffer)에 캐시됨.
- Virtual memory에서 page replacement policy에는 어떤 것이 있는가? LRU의 단점은?
  LRU(Least Recently Used), FIFO(First-In, First-Out), Least Frequently Used (LFU), Random, Clock Algorithm 등
- 운영 체제에서 memory management 기법중 하나로 paging이 많이 사용되고 있다. Paging 기법의 장·단점으로는 무엇이 있으며, hierarchical paging 혹은 inverted page table은 어떤 환경에서 유리한가?
  Hierarchical paging: 계층적 페이지 테이블 구조를 이용하면 매핑이 존재하지 않는 주소 범위에 대한 페이지 테이블은 없어도 됨 -> 메모리 공간 절약, 가상 주소 공간이 큰 경우 유리.
  Inverted page table: 존재하는 물리적 페이지에 대한 매핑만 저장하여 페이지 테이블의 크기를 물리적 메모리 크기에 비례하게 줄임. 물리적 메모리가 크고 프로세스 수가 많을 때 유리.

DB

〈Keywords〉

- Relational Model
- SQL
- ER Model
- Database Design (Normalization)
- Complex Data Types
- Database for Big Data

〈Preliminaries〉

- DBMS와 file management system의 차이점은?
  File management system에서 data는 file에 독립적으로 저장되어 있으며, 질의 처리가 application specific하게 이루어져야 하며, redundancy, inconsistency, integrity 부족 등의 문제가 존재함. 반면, DBMS에서는 data가 특정 구조에 따라 저장되어 있고, 질의 처리가 query language를 통해 systematic하게 이루어지며, redundancy, inconsistency를 줄이고 integrity를 제공하기 위해 다양한 기능이 제공됨.
- Logical database design과 physical database design의 차이점은?
  Logical database design: relation의 attribute, relationship 등을 설계하는 것
  Physical database design: 물리적 저장 구조(어떻게 disk, memory에 저장되는지)를 설계하는 것.
- DBMS의 세 단계 data abstraction은 (physical, logical, view level)?
  Physical level: data가 storage에 저장되어 있는지 나타냄
  Logical level: 어떤 data가 저장되어 있는지와 그들간의 관계성을 나타냄
  View level은 각 user 별로 제공되어야 하는 일부 data를 나타냄.
- Physical data independence란?
  DBMS 내부의 물리적 저장 구조를 바꾼다 하더라도 user 혹은 application에게 노출되는 relation의 구조 등에는 전혀 영향을 끼치지 않는 것을 의미함.

〈Relational Model〉

- *Database schema의 주요 구성 요소는?*
  table 이름, table의 attribute 이름 및 type, table 간의 관계(foreign key relationship) 등
- Primary key와 foreign key의 차이점은?
  Primary key는 table 내 각 tuple의 unique identifier이며, foreign key는 다른 table의 primary key와 연결하기 위한 attribute 값을 가짐.
- Referential integrity는 무엇인가? 이를 violate하지 않도록 제공되는 기능은?
  Referential integrity는 foreign key가 실제로 존재하는 tuple을 reference하고 있어야 함을 뜻함. Reference되는 table에서 tuple을 지우게 되면 referential integrity가 violate될 수 있는데, 이때 "on delete cascade" 옵션을 사용하면 지워진 tuple을 reference하는 foreign key의 tuple도 함께 지워져 referential integrity가 유지됨.
- Relational algebra의 5개 기본 연산자는?
  selection, projection, cartesian product, set union, set difference

〈SQL〉

- DDL과 DML의 차이점과 예제 명령어는?
  DDL은 schema를 정의하기 위한 명령어이며 create table 등이 있음. DML은 data를 변경하거나 query를 처리하기 위한 명령어이며, insert, delete, update, select가 있음.
- Inner join과 outer join의 차이점은? Outer join은 왜 필요한가?
  Inner join은 두 table의 join attribute 값이 일치하는 tuple 쌍만 결과로 나오게 되며, outer join은 join attribute가 일치하지 않는 tuple 쌍도 결과로 나오게 됨. Outer join은 특정 table의 정보를 모두 join 결과에 포함시켜야 할 경우 필요함.
- 인덱싱 알고리즘을 하나 설명하시오.
  B+-Tree 알고리즘이 대표적임. B+-Tree는 하나의 노드에 여러 개의 키를 가지고 있음. Leaf node가 아닌 경우, 자식 노드들은 키 사이의 포인터에 대응되며, 이 자식 노드의 키 값은 두 키 값 사이에 있다는 특징이 있음. Leaf node의 경우 각 포인터가 DB의 튜플에 대응됨. 트리가 unbalance해지는 경우를 막기 위해 한 노드에 키가 너무 많아지면 노드를 분할하거나 키를 다른 노드로 보내는 특성이 있음.
- SQL 질의 처리시 index(예: B+-tree)를 통해 처리 속도가 빨라지는 원리는?
  Where 조건을 만족하는 tuple을 가려내기 위해 table의 모든 tuple을 sequential하게 access할 필요 없이 조건을 만족할 수 있는 일부 tuple만을 access 함.
- Nested (sub) query는 무엇이며 언제 사용하는게 좋은가?
  SQL 질의의 select, from, where 절에 또 다른 select-from-where SQL 질의가 포함되어 있는 것을 의미함. 한 SQL 질의에서 순차적인 step의 logic을 구현하기 용이함 (예: 우선 aggregation을 수행하고 그 결과에 근거하여 data를 filtering 하고자 할 때).
- SQL의 having 절과 where 절의 역할과 차이점은?
  Having 절은 group by의 각 group이 만족해야 하는 조건을 나타내며, where 절은 각 tuple이 만족해야 하는 조건을 나타냄. Where 절은 group by 처리 전에 수행되며, having 절은 group by 처리 후에 수행됨.

〈Database Design〉

- Data inconsistency란 무엇인가?
  동일한 데이터를 다루는 두 테이블에서 값이 일치하지 않게 나타나는 것.
- ER model은 무엇이며 왜 필요한가?
  데이터베이스의 conceptual design에 사용되며, 크게 entity set과 relationship set으로 구성되며, 이들의 attribute를 나타낼 수 있음. ER model은 실세계 requirement를 보다 직관적으로 나타내어 relational schema로의 변환을 용이하게 함.
- Many-to-many relationship으로 연결된 두 entity set을 relational schema로 표현하기 위해 몇 개의 relation이 필요한가?
  Relationship 자체에 별도의 relation이 필요하여 총 3개가 필요함.
- Data normalization은 무엇이며 왜 필요한가?
  Database 내의 relation을 특정 규칙에 의해 여러 개로 쪼개는 과정을 의미하고, 주로 database에 존재하는 중복성(redundancy)를 줄이기 위해 필요함.
- Functional dependency는 무엇인가? 간단한 예시는?
  두 attribute set 간의 관계를 나타내며, 첫번째 attribute set의 값에 의해 두번째 attribute set의 값이 정해짐을 뜻함. 예를 들어, 주민번호와 이름 attribute를 고려하면, 주민번호에 따라 이름이 무조건 정해지므로 functional dependency가 존재함.
- 3NF과 BCNF의 차이점과 각 장단점은?
  3NF에서는 non-key attribute에 의한 functional dependency가 허용되는 반면, BCNF에서는 이것이 전혀 허용되지 않음. BCNF가 data redundancy를 더 많이 줄일 수 있는 반면, dependency preservation을 보장하지 못함. 반대로 3NF는 BCNF에 비해 data redundancy가 더 높을 수 있으나, dependency preservation을 보장함.

〈Database for Big Data〉

- NoSQL의 개념과 예제 시스템은?

  NoSQL은 "Not only SQL"의 약어로, 이름 그대로 SQL과 같은 고정된 스키마를 쓰지 않는다는 특징이 있음. 분산 컴퓨팅 환경에서 대규모 데이터를 처리할 때 특히 효율적임. 대표적인 예시로는 MongoDB가 있음.

- NOSQL database와 relational database의 주요 차이점은?

  Relational database에서는 사전 정의된 엄격한 schema를 요구하지만, NoSQL database에서는 schema가 필요 없거나 flexible함. Relational database는 ACID property를 대체적으로 만족하지만, NoSQL database는 performance를 위해 이를 희생하는 경우가 많음. Relational database는 SQL을 표준으로 채택하지만, NoSQL database는 그렇지 않은 경우가 많음.

- Vector database가 AI 응용에 많이 사용되는 이유는? (2023년 가을 이후 일부 교육)

  AI 응용에서는 데이터를 embedding model에 의해 표현되는 고차원 vector 값으로 나타내기 때문에 유사한 고차원 vector를 빠르게 검색할 수 있는 vector database가 AI 응용에서 매우 중요한 역할을 하기 시작하였음.

- LLM의 RAG에서 사용되는 벡터DB의 개념과 예제 시스템은?

  벡터 형식의 데이터를 처리하는 데에 특화가 되어 있는 DB 시스템. 벡터 유사도를 기반으로 한 검색과 같은 편리한 기능을 제공하여 인공지능 분야에서 자주 쓰임. 예시로는 Pinecone, Elasticsearch 등이 있음.

- Vector database가 LLM의 RAG(retrieval-augmented generation)에 사용되는 원리는?

  외부 data가 embedding model에 의해 이미 vector로 변환되어 vector database에 저장되어 있음. 주어진 질문을 동일한 embedding model을 통해 vector로 변환한 후 역시 vector database를 통해 저장되어 있는 유사한 vector를 검색함. 이렇게 retrieve된 vector는 주어진 질문과 관련성이 높은 정보로서 query prompt에 추가함으로써 query prompt를 augment 할 수 있음.

이산 수학, 확률, 통계

〈Keywords〉

- Propositional Logic, Predicate Logic
- Program Verification
- Methods of Proof
- Sets
- Relations: Equivalence Relation, Order Relation
- Function
- Mathematical Induction
- Recursive Definition, Algorithms & Relations
- Counting
- Discrete Probability
- Graphs, Trees
- Algorithms

〈Program Verification〉

- Loop Invariant란 무엇인가?
  알고리즘의 루프 내에서 항상 참인 조건. 루프가 시작되기 전, 진행되는 동안, 그리고 끝날 때 모두 참이여야 함.
- Partial correctness와 Total correctness의 차이는 무엇인가?
  Partial correctness과 Total correctness는 프로그램이 반드시 종료됨을 보장함. Total correctness는 Loop invariant와 더불어 termination condition도 사용해서 증명함.
- Precondition과 Postcondition은 무엇인가?
  Precondition: Procedure가 시작되기 전에 만족되어야 하는 조건.
  Postcondition: Procedure가 완료된 후 반드시 만족해야 하는 조건.

〈Relation〉

- Relation의 정의는?
  특정 조건을 만족하는 원소 순서쌍의 집합. 두 집합의 원소들 사이의 관계를 나타낸다고 볼 수 있다.
- ∀a ∈ A, ∀b ∈ B 일 때 aRb <-> (a, b) ∈ R 이 뭔 뜻인지 설명하시오. 여기서 왼쪽 R와 오른쪽의 R이 같은 것인가, 다른 것인가? 다르다면 어떻게 다른가?
  "a와 b가 관계 R에 있다"는 "(a, b) 순서쌍이 관계 R가 나타내는 집합의 원소이다"와 동치이다. 왼쪽은 관계, 오른쪽은 집합에 더 가까운 의미.
- Equivalence relation이란?
  반사적(reflexive), 대칭적(symmetric), 추이적(transitive) 성질을 모두 만족하는 Relation.
- Partial order relation이란? Total order relation이란?
  Partial order relation: 집합의 원소들 중 일부분만 순서가 정해진 관계. 반사적(reflexive), 비대칭적(antisymmetric), 추이적(transitive)이어야 함.
  Total order relation: 집합에 있는 모든 원소에 대해 순서가 정해진 관계. 위의 조건에 추가적으로

완전성 (Totality)를 만족해야 함.

〈Function〉

- Bijection의 정의는?
  두 집합의 모든 원소를 중복 없이 일대일로 대응시키는 함수.
- Bijective function은 반드시 total function인가?
  Total function은 정의역의 모든 요소가 공역의 한 요소와 대응하는 함수임. 따라서 Bijective function은 항상 total function임.
- Partial function과 total function의 정의를 말하고 그 예를 드시오.
  Partial function: 정의역의 일부 요소에 대해서만 값이 정의된 함수. f(x) = sqrt(x)
  Total function: 정의역의 모든 요소에 대해서 값이 정의된 함수. 다항함수.

〈Mathematical Induction〉

- Mathematical induction이란?
  어떤 명제가 모든 자연수에 참임을 보이기 위해 사용하는 증명 방법. n = 1인 경우에 대해 명제를 증명하고 n = k에서 명제가 성립한다면 n = k + 1에서도 명제가 성립함을 보임.
- Mathematical induction의 한 종류로서 Strong induction은 무엇인가?
  Mathematical induction과 목적 및 과정이 유사하나, Strong induction에서는 P(1) ~ P(k)가 참이면 P(k+1)도 참임을 보여 P가 모든 자연수에 대해 참임을 보임.

〈Recursive Definition〉

- Structural induction이란?
  재귀적으로 정의된 구조에서 어떤 명제가 참임을 성립하기 위해 쓰는 증명 방법. 기저 사례에서 명제가 참임을 보이고 substructure에서 명제가 참이라면 이를 포함하는 한 단계 위의 structure에서도 명제가 참임을 보임.
- Binary tree의 recursive 정의를 제시하시오.
  노드가 없는 상태. 혹은 왼쪽 및 오른쪽 자식으로 Binary tree의 root를 갖는 노드.

〈Counting〉

- f: A->B 함수이며, |A|=n 이고 |B|=m일때, 몇개의 onto function 을 정의 할 수 있는가?
  Inclusion-Exclusion 원리를 이용해 공역의 모든 원소가 정의역에 대응되는 경우의 수를 계산하면,

$$m^n - mC1\,(m-1)^n + mC2\,(m-2)^n - \ldots = \sum_{k=0}^{m} mCk\,(-1)^k (m-k)^n$$

〈Discrete Probability〉

- Random variable이 무엇인가?
  어떤 사건에서 나올 수 있는 모든 결과를 각기 다른 수치로 표현한 것.
- Sample space란 무엇인가?
  발생 가능한 모든 사건 결과의 집합
- Sample space가 갖는 조건이 무엇인가?
  가능한 모든 결과를 포함해야 하고 (Exhaustiveness) 각 원소는 배타적 이어야 한다 (Mutual Exclusiveness).
- Exponential distribution이 무엇인가?
  연속 확률 분포의 일종으로, Random variable의 값이 커질 수록 확률이 지수적으로 감소한다. 일정 시간 동안 사건이 발생하는 횟수가 포아송 분포를 따른다면, 사건 사이의 시간 간격은 Exponential distribution을 따르는 특성이 있다.

- Poisson distribution이 무엇인가?
  이산 확률 분포의 일종으로, 일반적으로 단위 시간 동안 사건이 발생하는 횟수를 모델링하기 위해 사용.

〈Graph, Trees〉

- Dijkstra의 Shortest path algorithm의 전제조건은?
  가중치가 양수여야 함. 그렇지 않은 경우 플로이드-워셜 알고리즘을 이용해 최단거리를 구할 수 있음.
- MST가 의미하는 것은?
  그래프의 모든 정점을 포함하는 subgraph 중 순환 경로가 없고 간선 가중치의 합이 최소인 것.

네트워크

〈Keywords〉

- Delay, Loss, Throughput
- HTTP, DNS, Video Stream
- UDP
- TCP, Congestion Control
- Network layer; IPv4, IPv6,
- Generalized forwarding, Routing
- Link Layer, wireless/mobile networks
- Security in computer networks

〈Preliminary〉

- OSI 7계층에 대해 설명하시오.
  네트워크 통신을 7개의 계층으로 나누어 각 계층의 기능과 역할을 정의한 것.
- Transport, network, link 계층이 모두 연결에 관한 것인데, 어떤 특징과 차이가 있는가?
  Transport: e2e session을 관장하는 프로토콜, end host에서 demux를 위한 identifier가 필요함.
  Network: Internet wide한 routing을 통해 end host간의 통신을 책임짐.
  Link: 물리적 인터페이스 간의 single-hop에 적용되고, network layer와는 별도의 addressing scheme을 가짐.

〈HTTP, DNS, Video Stream〉

- HTTP/2에서 persistent connection이란?
  HTTP 문서를 로드할 때 매 object마다 연결을 새로 하는 대신, 여러 object를 한 번의 connection에서 가져옴으로서 TCP 연결로 인한 오버헤드를 줄임. 또한 Object들을 쪼개서 보내 HoL blocking을 mitigate시킴.
- HTTP/3가 HTTP/2에 비해 바뀐 점은?
  구글의 QUIC을 채택함. 암호화를 제공하며 UDP 위에서 동작함. HTTP/2의 per-connection congestion control을 per-object로 바꿔 HoL blocking을 더 완화시킴.
- DNS에서 주소에 대응되는 entry는 어떻게 찾아지는가?
  클라이언트가 Hierarchical한 분산 시스템에서 local DNS server를 통해 root / TLD / authoritative servers를 대개 iterative하게 접근함. 보통은 Local DNS server에 lookup된 entry들이 캐싱되어 있으므로 대부분의 DNS lookup은 local DNS server에서 서비스됨.
- Video streaming에 쓰이는 프로토콜 중 하나를 설명하시오.
  DASH (Dynamic Adaptive Streaming over HTTP): 서버가 하나의 video file을 여러 encoding 레벨과 chunk 단위로 쪼갬. 해당 file을 어느 서버가 가지고 있는지에 대한 정보를 manifest file에 담음. 클라이언트가 꾸준히 server-to-client bandwidth를 측량하고, manifest file을 참고하여 어느 chunk를 어떤 서버로부터 다운로드 받을지를 결정.

〈TCP〉

- TCP와 같이 protocol을 reliable하게 설계 하려면 무엇이 필요한가?
  데이터 재전송이 필요한지 트래킹하기 위한 Seq. #, 전송 확인을 위한 ACK, 패킷 손실을 감지하기 위한 timestamp 등
- Flow control이란 무엇인가? Congestion control과 flow control의 차이점은?
  송신 측이나 수신 측이 데이터 전송 속도를 조절하여 데이터 손실 및 시스템 과부하를 막는 것. Flow control이 각 peer의 상황에 따라 이루어진다면, Congestion control은 peer 사이의 network의 상황에 따라 조절이 이루어짐.
- TCP와 UDP의 차이점은?
  TCP: Reliable하지만 헤더가 크고 전송 속도가 느림.
  UDP: Reliable하지 않지만 오버헤드가 적고 데이터를 빠르게 송신 가능.
- Congestion이 발생했다는 것을 어떻게 알 수 있는가?
  Timeout이 일어난 경우, 서로 다른 데이터 전송 후에 동일한 ACK가 오는 경우 등
- 3-way handshaking이란?
  TCP에서 두 호스트 간 연결을 설정하기 위해 사용하는 프로토콜의 일부. 상호 간 연결 확인 및 시퀀스 번호 확인이 이루어짐.
- TCP congestion control은 어떤 원리로 동작하는가?
  각 호스트가 AIMD (Additive Increase, Multiplicative Decrease)를 기반으로 해서 congestion control을 시행함. 초기 flow가 달라도 점근적으로 모든 호스트가 bottleneck bandwidth의 fair share를 갖게 됨.
- TCP는 어떻게 발전하였는가?
  TCP Tahoe: fast retransmit
  TCP Reno: loss detection에서 3 duplicate acks와 timer expiration를 구분
  CUBIC: initial window growth = exponential

〈Generalized forwarding, Routing〉

- Generalized forwarding에 대해 설명하라.
  기존의 IP routing은 라우터들이 독립적으로 라우팅 프로토콜 연산을 하였으나 SDN에서는 중앙집중적으로 controller에서 모든 라우팅 연산을 하여 SDN 스위치에 전달하는 방식. OpenFlow의 match + action abstraction으로 스위치 연산을 정의. MPLS 등의 추가적인 TE 방식 채택없이 중앙집중적으로 TE이 가능해짐. Firewall, IDS 등의 다양한 보안 규칙도 추가의 middlebox 구성없이 가능해짐.
- Internet routing은 어떻게 일어나는가?
  Inter-AS와 intra-AS routing으로 hierarchical하게 구성됨. Inter-AS routing은 policy-based BGP routing, Intra-AS routing은 shortest path routing (OSPF, RIP). OSPF의 경우 area를 나눠 hierarchical routing을 지원함.
- BGP가 어떻게 작동하는지 설명하여라.
  eBGP와 iBGP로 구성됨. eBPG를 통해 이웃 AS로부터의 라우팅 정보 교환하고 iBGP를 통해 이웃 AS로부터 알게된 라우팅 정보를 AS내의 다른 라우터들에게 배포함. BGP route에는 AS-PATH가 거쳐가야하는 AS들과 다음 AS로 나가기 위해 거쳐야하는 intra-AS내의 gateway 정보가 있음. 여러 route가 있을 경우 Local preference (policy-based routing), shortest AS path, closest NEXT-HOP 순으로 적합도를 따짐.

〈Link Layer〉

- CSMA/CD란?
  이더넷 네트워크에서 사용되는 프로토콜. 여러 장치가 동일한 전송 매체를 공유할 때 쓰임. 매체가 쓰이고 있는지, 데이터 전송 중 충돌이 발생했는지 알 수 있는 기능을 포함.

- MAC address란? IP address가 MAC address에 비해 갖는 장점은?
  네트워크 인터페이스 카드에 할당된 고유의 하드웨어 주소. IP 주소는 MAC 주소와 달리 동적 할당, 라우팅이 가능하며, 이를 이용하면 물리적으로 직접 연결되어 있지 않더라도 다른 기기를 거쳐 통신이 가능함.
- Wireless network에서는 왜 Collision avoidance를 채택하는가? 어떻게 하면 효율성을 증가시킬 수 있는가?
  Hidden terminal이 있어 collision detection이 안 될 수 있음. Data 보낸 후 ack을 기다리기 보다 RTS/CTS를 쓰면 더 효율적.
- 802.11 프로토콜의 특징을 설명하여라.
  Access point를 포함해 3개의 MAC address가 관여되어 있음. Base station와 client가 BER를 고려해 rate adaptation을 함. node가 power management를 위해 base station에 next beacon frame 전까지 sleep을 알림.

〈Security in Network〉

- Network security의 4개 구성요소는 무엇인가?
  Confidentiality, Authentication, Message integrity, Access and availability
- TLS에 대해 설명하여라.
  Confidentiality, authentication, message integrity를 모두 만족함. RSA를 connection setup 및 master key generation에 쓰고, 그 이후는 symmetric cypher suite을 통해 효율을 높임.
- Network에서 checksum을 위해 사용하는 hash function과 security에서 data integrity를 위해 사용하는 hash function이 서로 interchangeable한가?
  Network에서 사용하는 hash function은 속도가 중요시 되며 악의적인 공격에 의한 데이터 변조를 고려하지 않음. Security에서 사용하는 hash function은 데이터 위변조를 막기 위해 충돌 저항성이 요구됨. 즉, interchangeable하지 않음.

Software Engineering

〈Keywords〉

- Process Model: Heavy-weight, Prototyping, Agile, DevOps, …
- Software Modeling: UML (Unified Modeling Language)
- Software Requirements Engineering
- Software Engineering Principles: Separation of Concerns, Abstraction, Modularity, …
- Software Quality: Correctness, Robustness, Safety, Security, …
- Software Design: Architectural Patterns, Resilient Design Patterns
- Software Testing: Black-box & white-box testing

〈Software Development Life Cycle and Process Model〉

- 주어진 프로젝트에 적합한 프로세스모델을 선택하기위해 고려해야 할 사항은?
  프로젝트 규모, 요구 사항의 명확상, 프로젝트 기간, 리스크, 프로젝트의 품질 요구 조건 등
- Evolutionary Model (Prototyping)과 Incremental Model의 차이점에 대해 설명하고 어떠한 프로젝트에 적합한지?
  Evolutionary Model: 요구 사항이 불명확한 상황에서 프로토타입을 반복적으로 개선해 프로젝트를 완성시킴. UI/UX가 중요한 프로젝트 및 새로운 기술을 사용하는 프로젝트에 사용함.
  Incremental Model: 요구 사항이 명확한 상황에서 전체 시스템을 증분으로 나누어 빠르게 개발하고 이후 기능이 점진적으로 보완됨. 초기 버전을 빠르게 제공해야 하는 프로젝트, 대규모 프로젝트에 사용함.
- 기존의 Heavy process model의 단점을 해결하기위해 나온 Agile Methods들 중에 하나를 선택해 프로젝트에 적용할때 선제 조건은?
  Scrum. 요구사항이 명확하지 않거나 자주 변경될 가능성이 높은 프로젝트. 소규모 팀. 고객이 지속적으로 피드백을 해 줄 수 있어야 함. 짧은 개발 주기와 빠른 배포가 요구되는 프로젝트.

〈Software Modeling〉

- UML 에서 시퀀스 다이어그램이란 무엇인가?
  시스템 내 오브젝트들 간의 상호 작용을 시간 순서에 따라 시각적으로 표현한 것.
- 모델간의 추적성이란 무엇이고 추적성을 유지하는 것이 왜 중요한가?
  소프트웨어 개발 과정에서 다양한 단계와 산출물 간의 관계를 식별하고 추적할 수 있는 능력. 추적성을 유지하면 요구 사항에 변경이 생겼을 때 이를 용이하게 처리할 수 있고, 전체 프로젝트 관리에 도움이 됨.
- 추적성 유지와 관리를 위해 가장 중요한 모델은 무엇인가?
  요구사항 관리 모델.
- UML에서 개발하고자하는 시스템의 동적, 정적, 인터액션을 나타낼 수 있는 다이어그램에대해 예를 들어 설명하면?
  로그인을 예시로 들면,
  정적 요소: 클래스 다이어그램. 사용자 클래스, ID 및 비밀번호 속성, 비밀번호 변경 메소드.
  동적 요소: 시퀀스 다이어그램. 사용자가 서버에 로그인 요청을 보내면 서버가 이를 처리하고 응답 반환.
  인터랙션 요소: 인터랙션 다이어그램. 사용자는 아이디 및 비밀번호를 보내고 서버는 JSON을 반환.

- 유즈케이스 모델(Use Case Model)은 무엇을 표현하기 위한 것인가?
  Actor와 시스템이 상호작용하는 방식을 표현하기 위해 사용됨. 이를 통해 시스템이 제공해야 하는 기능 및 범위를 정의할 수 있음.
- 디자인의 평가 메트릭인 Cohesion 과 Coupling 에 대해 설명?
  Cohesion: 한 모듈이 얼마나 단일한 목적을 가지고 있는지 나타냄. 높을수록 좋음.
  Coupling: 한 모듈이 다른 모듈에 얼마나 의존적이고 연결되어 있는지 나타냄. 낮을수록 좋음.
- Software architecture란? 그 역할은?
  소프트웨어 시스템의 구조와 구성 요소 간의 관계를 정의하고, 시스템의 설계와 구현을 위한 고수준의 계획을 제공하는 개념.

<Software Engineering Principles>

- SW에서 모듈화란? 모듈화를 잘 하기 위해 중요한 것은?
  소프트웨어를 설계 할 때 독립적이고 재사용할 수 있는 모듈로 나누어 기획하는 것. 재사용성과 범위를 적절히 고려하여야 하며, 상호작용을 위한 인터페이스에 대한 규약이 필요함.
- DevOps는 어떠한 종류의 소프트웨어 개발과 배포에 효과적인가?
  지속적인 업데이트가 필요한 소프트웨어, 마이크로서비스 아키텍쳐, 대규모 시스템
- AI 시대의 개발자의 가장 중요한 역할이 무엇이라고 생각하나?
  정확한 요구 사항의 수집, 소프트웨어 아키텍처 설계, 효율적인 개발 모델 선택 등

<Software Requirements Engineering>

- 요구 공학에서는 어떻게 요구사항을 수집하는가?
  인터뷰, Ethnography, 브레인스토밍 워크숍, 스토리보딩, 롤플레잉, 프로토타이핑 등
- 올바른 요구사항 파악을 위해 가장 중요한 것은?
  Stakeholder의 식별, 다양한 요구사항 수집 기법 사용, 실제적인 상황을 관찰하고 수집하기 위한 커뮤니케이션 등
- 요구사항 관리(Requirements Management)를 왜 해야 하는가?
  요구사항 변경에 따른 요구사항 충돌을 막고, 변경점으로 인해 수정되야 하는 다른 요구사항을 추적해 요구사항 시스템을 안정적으로 유지하기 위함.
- 품질속성(Quality Attributes)는 어떻게 도출하고 왜 중요한가?
  유틸리티 트리, Facilitated Brainstorming 등의 방법이 있음. Quality Attribute가 중요한 이유는 이를 도출하여 측정 가능한 criteria를 만들 수 있고, 설계와 테스팅 단계에서 큰 도움이 되기 때문.
- Requirement Engineering Process에서 Requirement Analysis의 역할에 대해 설명?
  Elicitation 단계에서 도출된 요구 사항을 조직화, 상충되는 요구를 정리하고 UML 등을 통해 모델링 하는 단계. 시스템 모델링이 stakeholder의 요구사항과 잘 부합하도록 유도한다.
- Volatile requirement 원인과 해결방법은?
  비즈니스 환경 변화, stakeholder의 요구 변화, 기술의 발전, 사용자 피드백 등으로 인해 요구 사항이 변할 수 있음. 초기에 요구사항 관리 프로세스를 명확하게 설정해 이러한 변화에 대비해야 함.

<Software Testing>

- 테스트 기법으로 화이트 박스와 블랙 박스 테스트가 있는데, 그 둘의 차이점은? 화이트 박스 테스팅이 에러가 없는 프로그램이라고 보장해 주지 않는 이유는?
  화이트 박스: 내부 구조를 고려해 테스트
  블랙 박스: 내부 구조를 고려하지 않고 입력과 출력만 이용해 테스트
  화이트 박스 테스트를 하더라도 모든 코드 경로를 검증하기 어렵고, 테스트 케이스가 다양한 입력에 대한 결과를 커버하지 못할 수 있음.

- Verification Testing과 Validation Testing의 차이는 무엇인가?
  Verification Testing: 개발 과정에서 SW가 주어진 요구 사항에 들어맞는지 테스트.
  Validation Testing: 개발된 SW가 stakeholder와 사용자의 실제 기대에 부합하는지 테스트.
- Test Oracle 문제란 무엇인가?
  기술적 한계나 요구사항 단계에서의 문제 등으로 인해 테스트 결과의 정확성을 판단하기 어렵거나 불가능한 상황.
- Testing을 통해 결함이 없는 소프트웨어를 만들 수 있는가?
  큰 규모의 프로젝트에서는 사실상 불가능. 휴먼 에러의 가능성, Test Oracle 문제 등으로 인해 testing의 무결성을 보장할 수 없음. 따라서 SW의 완전한 무결성도 테스트 불가함.

〈기타〉

- CMMI 레벨이란?
  소프트웨어 개발 프로세스의 성숙도를 평가하고 개선하기 위한 모델. 개발 성숙도에 따라 5단계로 분류됨.

AI / ML

<Keywords>

- Problem Solving: Searching, Constraint Satisfaction Problem
- First-Order Logic, Knowledge representation, Automated Planning
- Machine Learning: Decision Trees, Linear Regression, …
- Deep Learning: FFN, CNN, RNN
- Reinforcement Learning
- Natural Language Processing, Computer Vision

<Problem Solving>

- A* 알고리즘에서는 goal까지의 거리를 예측하는 휴리스틱 함수가 쓰인다. 이 휴리스틱 함수는 어떤 조건을 만족해야 하는가?
  허용성 (Admissible): 모든 n에 대해 h(n)은 실제 비용과 같거나 더 작아야 한다.
  일관성 (Consistency): 모든 n에 대해 h(n)은 이웃 노드까지의 이동 비용과 이웃 노드에서 목적지까지의 휴리스틱의 합보다 작거나 같아야 한다.

<Machine Learning>

- K-Means 목적함수와 알고리즘을 각각 설명하시오.
  목적함수: 각 점에서 할당된 cluster의 중심 (cluster에 할당된 점들의 평균)까지 거리의 합
  알고리즘: 초기화 단계에서 점들을 임의의 cluster에 할당. cluster의 중심을 계산. 각 점들을 중심이 가장 가까운 cluster에 할당. 2번째와 3번째 단계를 할당이 변하지 않을 때까지 반복.
- Degree 4의 Polynomial Regression 모델이 오버피팅하고 있다면 모델의 Degree를 어떻게 바꾸어야하는지, 왜 그렇게 하여야하는지 설명하시오.
  현재 모델의 capacity가 높아 testing 데이터에 대한 학습이 지나치게 잘 되고 있는 상황. 모델의 Degree를 낮추어 capacity를 낮추고, 보다 일반적인 feature에 대한 학습이 일어나도록 유도해야 함.
- 20세~40세 성인의 얼굴 사진을 보고 나이를 예측하는 모델을 만들려고 한다. Classification과 Regression 중 어떤 접근법이 적합한가? 왜 그런지 설명하시오.
  나이는 연속적인 값이므로 Regression을 이용하는 것이 적합함. Classification의 경우 두 bucket 중간에 위치한 데이터가 input으로 들어왔을 때 두 분류에 대한 확률이 모두 높게 나타나 학습이 부정확하게 일어날 수 있음.
- MNIST 데이터셋의 10-way classification 문제를 베이지안 방법론으로 모델링할 경우, 사전확률(Prior Distribution), 우도함수(Likelihood Function), 사후확률(Posterior Distribution)을 각각 어떻게 정의할지 설명하시오.
  사전확률: 데이터셋에 0~9까지의 수가 균일하게 있다고 가정하면 $P(C) = 1/10$
  우도함수: $P(A_1, A_2, \ldots, A_n|C) = f(C)$, *f는 class가 C일 때 픽셀의 확률 분포 함수*
  사후확률: 베이즈 정리에 따라 $P(C|A_1, A_2, \ldots, A_n) = \frac{P(A_1, A_2, \ldots, A_n|C)\,P(C)}{\sum_{c=0}^{9} P(A_1, A_2, \ldots, A_n|c)\,P(c)}$

<Deep Learning>

- 인공신경망을 학습할 때 사용하는 Dropout 테크닉이 무엇인지, 그리고 왜 필요한지 설명하시오.
  신경망에서 확률적으로 뉴런을 무시하는 (가중치 0을 부여하는) 방법. 이를 통해 모델이 다양한 feature를 학습하게 하고 overfitting을 막을 수 있음.

<Reinforcement Learning>

- 강화학습에서 Exploration과 Exploitation의 개념이 무엇인지 각각 설명하시오.
  Exploration: 에이전트가 시도해 보지 않은 행동을 하거나 새로운 정보를 얻기 위해 어떤 행동을 선택하는 것. 장기적으로 이득이 될 수 있는 정보를 학습하기 위함.
  Exploitation: 이미 알고 있는 정보와 경험을 바탕으로 가장 높은 보상을 받을 가능성이 높은 행동을 선택하는 것.

알고리즘

〈Keywords〉

- (Time Complexity)
- Sorting
- Divide-and-conquer
- Dynamic Programming
- Greedy algorithms
- Graph algorithms
- Advanced data structures
- Complexity Theorem: NP-completeness

〈Sorting〉

- Sorting algorithm 들에는 어떤 것들이 있는가? Insertion sort, heap sort, selection sort, quick sort 중 optimal 한 것과 optimal 하지 않은 알고리즘은? 그 이유는?
  Optimal: Quick sort, Merge sort, Heap sort
  Not Optimal: Insertion sort, Selection sort, Bubble sort
- Partially ordered set이란?
  자신의 원소 중 일부만 순서가 정해진 집합.
- Quick sort 에 대해 설명하시오.
  피벗 선택 -> 피벗을 기준으로 더 작은 원소들은 왼쪽, 더 큰 원소들은 오른쪽으로 이동 -> 재귀적으로 피벗 좌측과 우측을 정렬.
- Show how you can make Quicksort to have O(n lg n) worst-case running time.
  Median-of-Medians: 원소를 5개씩 묶고 각 그룹에서 중앙값 계산. 그룹의 중앙값 중에서 중앙값 계산.
- Optimal한 정렬 알고리즘의 예를 들어보세요. 그 알고리즘의 complexity를 분석하고 optimal하다는 것을 증명해보세요.
  Quick sort, Merge sort, Heap sort 등. 모든 비교 정렬 알고리즘은 Ω(n lg n) 보다 빠를 수 없으므로 정렬 알고리즘의 복잡도가 O(n lg n)임을 증명.
- n개의 숫자가 array로 주어졌습니다. 이 중에 Median을 찾는 알고리즘을 설명해보세요. 설명한 알고리즘의 complexity는 무엇입니까? Median을 찾는 optimal한 알고리즘의 complexity는 무엇입니까? optimal하다는 것을 어떻게 증명할 수 있나요? Median을 찾는 알고리즘을 이용하여 n개의 숫자를 정렬해보세요. 이 정렬 알고리즘 의 complexity는 무엇입니까? 두 가지 문제를 통해 Reduction이라는 개념을 설명해 보세요.
  Medians-of-Medians 알고리즘을 응용하면 항상 O(n) 안에 Median을 찾을 수 있음. 이를 Quicksort에 적용하여 worst case에도 시간 복잡도가 O(n lg n)이 되도록 할 수 있음.
  Reduction: 문제를 해결하기 위해 이를 다른 문제로 변환하는 것. 첫 번째는 중앙값 찾기 문제를 어느 정도의 균등성이 보장되는 피벗 찾기 문제로, 두 번째는 정렬 문제를 중앙값 찾기 문제로 Reduction 한 것으로 볼 수 있음.

<Dynamic Programming>

- Dynamic Programming이란?
  문제를 하위 문제로 나누고, 하위 문제의 해결 결과를 저장해 두었다가 재사용해 최종 문제를 해결하는 알고리즘의 분류.
- Dynamic programming과 Greedy algorithm에 대하여 각각이 무엇인지 설명하고, 공 통점과 차이점을 이야기하세요. 구체적으로 Dynamic programming으로 풀 수 있는데 Greedy algorithm으로는 풀 수 없는 문제의 예를 들어보세요. 그 문제가 dynamic programming으로 풀 수 있다는 것을 증명해보세요.
  Greedy Algorithm: 매 단계에서 근시안적으로 가장 최적이라고 생각되는 선택을 하여 문제를 해결하는 알고리즘의 분류.
  공통점: 문제를 하위 문제로 나누어 해결함 / 차이점: Dynamic programming은 하위 단계들의 결과를 고려해 최적해를 찾음.
  0-1 Knapsack 문제. 아이템의 고려 범위와 배낭 용량을 변수로 해 하위 문제로 나눌 수 있음.

<Graph Algorithm>

- 그래프의 정의는? 트리의 정의는? 트리와 그래프의 관계는?
  그래프: 정점과 간선으로 이루어진 구조 / 트리: 사이클이 없는 무방향 그래프.
- Topological Sorting을 설명하시오
  방향 그래프의 정점을 간선의 방향을 거스르지 않도록 나열하는 것.
  In-degree가 0인 정점을 큐에 넣음. -> 해당 정점이 가리키고 있던 정점들의 In-degree 1 감소 -> 반복
- 오일러 사이클이란?
  그래프에서 모든 간선을 한 번만 지나면서 시점과 종점이 같은 경로.
- Tree traversal의 3 가지 방식을 설명하시오.
  전위 순회, 중위 순회, 후위 순회. 부모 노드를 언제 방문하는지가 다름.
- Transitive closure란?
  그래프에서 경로가 존재하는 두 정점의 순서쌍의 집합.
- Finite state automata란?
  유한한 수의 상태를 가진 기계. 읽어들인 문자열과 현재 상태를 기반으로 다른 상태로 전이하는 특성을 가지고 있음.
- Binary tree란? binary tree에서 각 node는 두 개의 children을 갖거나 혹은 leaf node 이거나 둘 중에 하나라고 할 때, non-leaf node와 leaf node 개수의 관계식은 어떻게 되는가?
  각 노드가 최대 2개의 자식을 갖는 트리. (Non-leaf node) + 1 = (Leaf-node)
- Minimum spanning tree를 구하는 알고리즘과, 그 알고리즘의 복잡도는?
  크루스칼 알고리즘, 프림 알고리즘. O(E ln V)
- Let T be a minimum spanning tree of G. Then, for any pair of vertices s and t, is the shortest path from s to t in G the path from s to t in T? Justify your answer.
  아니요. 반례: A -(2)- B -(1)- C -(1)- D -(1)- A, A->B 경로.
- Describe Dijkstra's algorithm and analyze its running time.
  그래프에서 임의의 시작 정점에서 다른 정점까지 최단 경로를 찾는 알고리즘.
  초기화: 시작 정점까지의 거리를 0으로 설정.
  반복: 방문하지 않은 노드 중 방문한 노드들에서 가장 가까운 노드(A)를 방문. A에 연결되어 있는 노드들까지의 거리를 확인. A를 거쳐 가는 것이 기존 경로보다 비용이 적게 든다면 업데이트.
  시간복잡도: O(E log V) (Priority Queue 사용시)

- Describe Floyd-Warshall algorithm and analyze its running time.
  그래프에서 모든 정점에서 다른 모든 정점까지의 최단 거리를 찾는 알고리즘.
  모든 정점 쌍에 대해 모든 정점을 중간 정점으로 고려해 비용 계산 (중간 정점까지 경로 비용 + 중간 정점에서 도착 정점까지 경로 비용). 이 정점을 거쳐 가는 것이 더 적은 비용이 드는 경우 비용 업데이트.
  시간복잡도: O(V^3)

〈Advanced data structures〉

- Priority queue란? Heap이란?
  Priority queue: 우선순위가 가장 높은 원소가 먼저 추출되는 자료구조.
  Heap: 모든 노드가 자신의 자식 노드보다 큰 (최대 힙) / 작은 (최소 힙) 값을 가지는 완전 이진 트리.

〈Complexity Theorem〉

- NP Complete의 정의는?
  NP: 답을 다항 시간 내에 검증할 수 있는 문제들의 집합
  NP-Hard: NP에 속한 모든 문제가 문제 P로 Reduction 될 수 있다면, P는 NP-Hard에 속함.
  Np-Complete: NP이면서 NP-Hard인 문제.

그래픽

〈Keywords〉

- Geometric Objects and Transformation
- Rotation
- Interpolation
- Viewing
- Rasterization
- Ray Tracing

〈Transformation〉

- Affine transform과 Rigid body transform에 대해서 차이를 설명하시오.
  Affine transform: 이동, 기울이기, 회전, 확대/축소를 포함. 물체의 기하학적 구조는 유지되지만 각도와 길이가 달라질 수 있음.
  Rigid body transform: 위치와 방향 변경만 가능. 각도와 길이가 유지되어 기하학적 구조가 유지됨.
- Affine transform은 크게 두 부분으로 분리가 가능하다. 분리가 가능한 두 부분에 대해서 설명하시오.
  선형 변환: 원래 좌표에 선형 변환 행렬을 곱함. 기울이기, 회전, 확대/축소를 포함.
  평행 이동: 원래 좌표에 평행 이동 벡터를 더함.

〈Rotation〉

- XYZ-Euler angle rotation의 Gimbal lock problem에 대해서 설명하시오.
  3차원에서 물체의 회전을 표현하기 위해 3개의 각을 사용하는데, 이때 각도에 따라 회전축이 겹쳐지면서 회전의 자유도를 잃을 수 있음. 이를 해결하기 위해 쿼터니언과 같은 방법을 사용함.
- Rotation transform은 Quaternion multiplication으로 표현이 가능하다. Quaternion에 대해서 설명하시오.
  Quaternion은 4개의 성분으로 이루어져 있음. 1개의 실수부와 서로 직교하면서 제곱하면 -1이 되는 3개의 허수부가 있음. 허수부를 묶어서 벡터부분, 실수부를 스칼라 부분이라고도 함.

〈Shading〉

- 여러 shading 모델들과, 차이점을 설명하시오
  Flat shading: 각 폴리곤에 대해 단일 색상을 적용하는 방법.
  Gouraud shading: 각 정점에서 색상을 계산한 후 폴리곤 안에서 색상을 선형 보간해 적용하는 방법.
  Phong shading: 법선 벡터를 사용해 정점 색상 값을 보간.

〈Rendering〉

- OpenGL rendering에서 model view matrix 에 대해서 설명하시오.
  3D 장면의 모델링과 뷰잉 변환을 결합한 행렬. 객체의 이동, 회전, 확대/축소를 담당하는 모델 변환 행렬과 카메라의 위치와 방향을 설정하는 뷰 변환 행렬의 곱으로 구할 수 있음.

- OpenGL Shading Language는 크게 fragment shader와 vertex shader로 나누어 질 수 있다. 둘의 차이점에 대해서 설명하시오.
  Fragment shader: 보간된 정점 속성을 바탕으로 최종적으로 렌더링 되는 픽셀값을 계산. 텍스처 매핑, 복잡한 조명 적용, 후처리에 사용.
  Vertex shader: 정점 속성을 받아 변환된 정점의 위치와 추가로 계산된 속성을 출력함. 기하학적 변환 및 기본적인 조명 계산에 사용.
- 3차원 기하정보를 2차원 영상으로 표현하는 과정을 렌더링이라고 한다. 이를 위해서는 perspective projection이 필수적으로 요구되는데, 이에 대해 설명하시오.
  관찰자는 일정 거리만큼 떨어진 평면에 투영된 상을 통해 3차원 물체를 시각적으로 받아들인다고 가정하고 이를 이용해 3차원 물체를 2차원에 표현하는 방법.

<Ray Tracing>

- 물체의 realistic 한 appearance를 렌더링하기 위해서는 물체의 표면 반사계수를 4차원 함수로 표현한 표면 모델(BRDF)이 필요하다. BRDF에 대해 설명하시오.
  BDRF는 4차원 함수로 정의되며, 입사광과 반사광의 방향을 나타내는 단위벡터가 주어졌을 때 입사각과 반사각에서의 반사 비율을 값으로 가짐. 이 함수는 항상 0보다 크거나 같으며, 헴홀츠의 상호성, 에너지 보존의 성질을 가짐.
- Recursive ray tracing에 대해서 간단히 설명하시오.
  반사광 및 굴절광이 다른 물체에 부딪히는 것을 고려해 렌더링하는 기법. 주 광선이 물체에 도달하면 반사광(또는 굴절광)의 방향 및 색을 계산하고, 이 광선을 다시 scene에 발사해 재귀적으로 빛의 영향을 계산. 최대 재귀 깊이에 도달하거나 광선의 세기가 일정 수준 이하로 낮아질 때 까지 반복.

<기타>

- Edge detection에 대해 설명하시오.
  이미지에서 경계를 추출하는 과정. 일반적으로 Sobel Operator, Prewitt Operator와 같은 Kernel Matrix와 이미지의 Convolution 연산을 통해 이루어짐.
- Massive한 데이터를 그래픽으로 출력하려면, 어떻게 해야 하는가?
- 3차원의 가상공간에 놓여 있는 3차원 물체들이 2차원의 그림으로 렌더링되어 우리의 화면에 보여진다. 이를 우리는 2차원 윈도우를 통해서 상호작용을 하게 되는데 이때 간단하게는 마우스를 이용하여 3차원 공간에 있는 물체를 선택하게 된다. 이때 마우스 input은 (x,y) 2차원 좌표인데 어떻게 3차원 공간에 있는 물체를 선택 (selection) 할 수 있을까? 이를 가능하게 할 수 있는 Object Selection 방법을 제시하라.
  Ray Casting. 뷰 변환의 역행렬을 이용하여 마우스 좌표를 2차원 뷰 좌표로 변환. 모델 변환의 역행렬을 이용하여 이 뷰 좌표를 3차원 공간 좌표로 변환. 관찰자가 마우스를 통해 가리키는 점으로 광선을 발사 후 물체와 충돌 검사.
- 물체의 빠른 충돌 감지를 위해서 물체를 감싸는 간단한 도형인 bounding volume 을 hierarchical 하게 구성하여 사용한다. 이의 간단한 예를 만들어 얼마나 빠르게 처리를 할 수 있는지 설명하라.
  최상위 볼륨이 32개의 물체를 포함하고, 이진 트리 형태로 BVH(Bounding Volume Hierarchy)가 구성되어 있다고 가정. 광선 충돌 검사를 할 때 BVH가 없다면 32개의 물체를 모두 검사해야 하지만, BVH를 이용하면 6회만에 가능.

정보 보호

<Keywords>

- Confidentiality
- Integrity
- Public-key cryptosystem
- Blockchain
- Access Control
- Web Security
- Software Error & Vulnerabilities
- Program Analysis

<Confidentiality>

- 암호 운영 모드에서 IV 또는 넌스를 사용하는 이유를 설명하시오.
  같은 평문에 대해서도 서로 다른 암호문이 출력되도록 하기 위함. 즉, 암호문에 가변성(variability)을 제공함.

<Integrity>

- 해시함수의 중요한 안전성 요구 조건을 세가지 설명하시오.
  역상저항성, 제2역상저항성, 충돌저항성
- 웹서버에서 사용자 패스워드를 안전하게 저장하고 사용하는 방법에 대해 설명하시오. 이에 필요한 암호학적 특성에 대해 논하시오.
  사용자가 입력한 암호 평문에 솔트를 추가 후 해시로 변환하여 저장. 이후 암호 평문 입력하면 동일한 솔트와 해시 함수를 적용 후 비교. 해시 함수는 역상저항성, 제2역상저항성, 충돌저항성을 요함.

<Availability>

- SYN Cookie 에 대해서 설명하시오.
  클라이언트에 부여되는 고유한 해시 값으로, 3-way handshake에서 서버가 클라이언트에게 SYN-ACK 단계에서 보냄. 클라이언트가 ACK와 함께 SYN Cookie를 다시 보내면, 서버는 이를 검증하고 연결을 수립함. 서버의 자원을 소모하지 않고 연결을 수립할 수 있어 SYN Flood 공격을 막을 수 있음.
- DDoS 공격에 대응하기 위한 방어 기술에 대하여 설명하시오.
  방화벽, Challenge-Response mechanism (ex) CAPTCHA), IPS, 로드 밸런서 등

<Public-key crpytosystem>

- 대칭키 암호와 공개키 암호의 장단점을 비교하시오.
  대칭키는 속도가 빠르고 (동일 안전성 수준에서) 키 크기가 상대적으로 작음. 반면, 공개키 암호는 효율성이 떨어지지만 암복호화 키가 다르므로 키 교환 및 관리가 용이함.
- 이산대수(discrete log) 문제를 설명하시오.
  군(group) G의 원소 g에 의하여 생성되는 부분군(subgroup) <g>의 원소 y가 주어졌을때, g^x=y인 x를 찾는 문제임.

- Diffie-Hellman 키교환 문제를 설명하고 어떻게 중간자공격(man-in-the-middle)에 취약한지 설명하시오.
  암호키를 교환하는 방법. 큰 소수 p와 generator g는 public key. A와 B는 서로에게 y_x = g^x log p를 전송 (x는 암호키). y_a와 y_b를 곱하여 공유된 암호키 생성.
  중간자 공격: 중간자가 y_a와 y_b를 y_p로 변조. A와 B가 y_a y_p와 y_b y_p를 각각 암호키로 사용하므로 중간자는 A와 B가 보낸 암호문을 모두 해독할 수 있음.

<Trust Model>

- 네트워크에서 정보를 전송할 때 데이터를 보호하기 위해 암호화하는 방법에는 어떤 것이 있는가?
- Transport Layer Security(TLS)가 보증하는 3가지 Security Properties에 대해서 설명하시오.
  Confidentiality: 데이터가 암호화되어 보내짐.
  Integrity: MAC와 해시함수를 이용해 데이터가 원문대로 보내졌음을 확인.
  Authentication: CA가 발급한 디지털 인증서를 통해 신뢰할 수 있는 엔티티인지 확인.
- PKI 시스템에서 CA(Certificate Authority)가 침해되었을때 발생할 수 있는 문제를 설명하시오.
  누구나 "신뢰 가능한 인증서"를 발급할 수 있게 됨. 공격자는 이를 이용해 자신의 mockup 홈페이지나 안전하지 않은 사이트에 인증서를 부여할 수 있음.

<Web security>

- Cross-site Scripting 공격에 대해서 설명하고 이를 방어하기 위한 방법들에 대해서 설명하시오.
  유저가 웹사이트에 의도하지 않은 스크립트를 넣어 공격하는 것. 유저 입력의 정제 (Sanitization)
  CSP(Content Security Policy) 선언을 통해 막을 수 있음.
- Content-Security Policy를 이용하면 XSS 공격을 완벽히 방어하는것이 가능한가? 가능하지 않다면 그 이유에 대해서 설명하시오.
  CSP를 통해 서버에서 만들어지지 않은 스크립트의 실행을 완전히 막을 수 있으므로 가능.

<Software Error & Vulnerabilities>

- Process 와 Thread에 대해서 설명하고 두개의 다른점을 설명하시오.
  Process: 실행 중인 프로그램. 독립적인 주소 공간을 가짐.
  Thread: 프로세스 내에서 실행되는 작업 단위. 메모리와 자원을 공유함.
- Process 들이 이용하는 가상 메모리(Virtual Memory)가 서로 다른 Process간 공유된다면 장점과 단점은 무엇인가?
  장점: 프로세스 간 정보 공유가 매우 쉬워짐.
  단점: 다른 프로세스에 의한 정보 탈취, 메모리 침범에 의한 공격에 취약해짐 등
- Buffer overflow 공격에 대해서 설명하시오.
  지정된 크기보다 메모리를 더 많이 읽거나 메모리에 더 많이 써서 하는 공격. 이를 통해 데이터를 읽고 조작하거나, 프로그램의 control flow를 바꿀 수 있음.
- Big endian and little endian에 대해서 설명하고 서로 다른 2개의 방식이 Buffer overflow 공격에 미치는 영향에 대해서 설명하시오.
  Big endian은 MSB가 가장 낮은 주소에, little endian은 가장 높은 주소에 저장됨. 데이터를 해석하거나 주입할 때 바이트 순서에 차이가 있음.

<Program Analysis>

- 대표적인 정적, 동적 프로그램 테스팅 기술에 대해 설명하고, 장단점을 논하시오.

정적 기술: 코드 리뷰, 정적 분석 도구, 형식 검증 등
동적 기술: 단위 테스트, 통합 테스트, 시스템 테스트, 회귀 테스트 등

<기타>

- 브라우저를 켜서 카이스트 웹사이트에 접속하기 까지 발생하는 모든 보안 기술들을 순서대로 논하고 간단하게 설명하시오.
  카이스트 웹서버와 연결: Syn Cookie
  오브젝트 로드: CA 인증서, TLS
  클라이언트에 표시: CSRF 토큰, CSP 등