Lecture 12 – Course Review

AAA705: Software Testing and Quality Assurance

Jihyeok Park

PLRG

2024 Spring

Contents

Contents

- **Combinatorial testing (CT)** or **combinatorial interaction testing (CIT)** constructs test cases by considering the **interactions** between the parameters.
- We need to find a **covering array** to ensure that all interactions are covered (e.g., constraint mixed covering array (CMCA) as follows).

A	B	C
0	0	0
1	1	1
		2

$P(x, y) = x + y > 0$

$\Rightarrow$

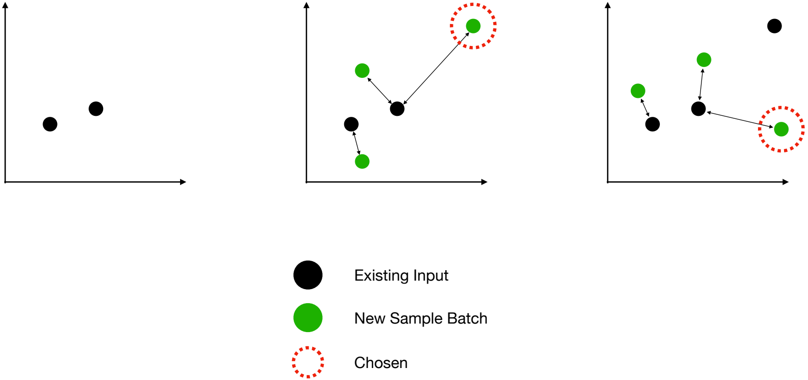
$CA(N = 5; t = 2, k = 3, v = (2, 2, 3), P)$

A	B	C
0	1	1
1	0	2
1	1	0
1	0	1
0	1	2

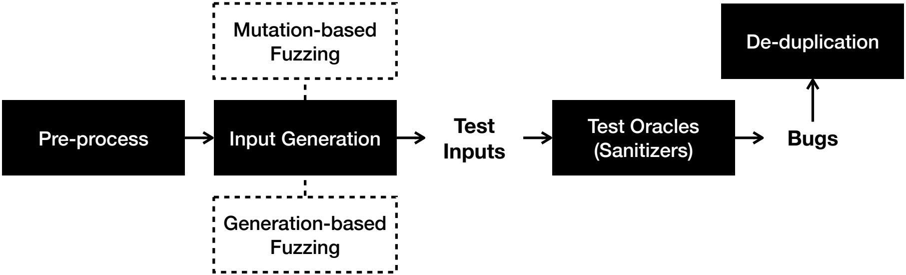
Black-box Testing – Adaptive Random Testing

- The **diversity** of a test suite is defined as the **sum of distances** between all pairs of test inputs.

$$diversity(T) = \sum_{(t_1,t_2)\in T\times T} d(t_1, t_2)$$



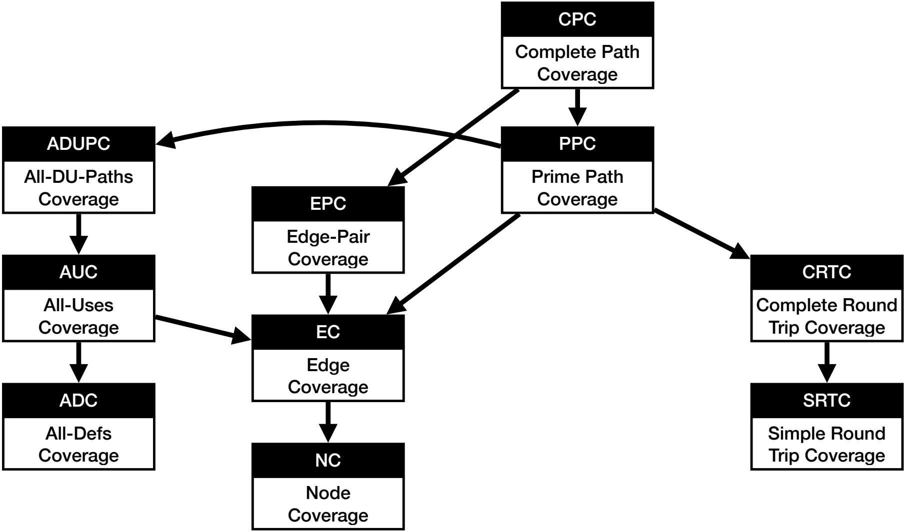
Black/White-box Testing – Fuzz Testing



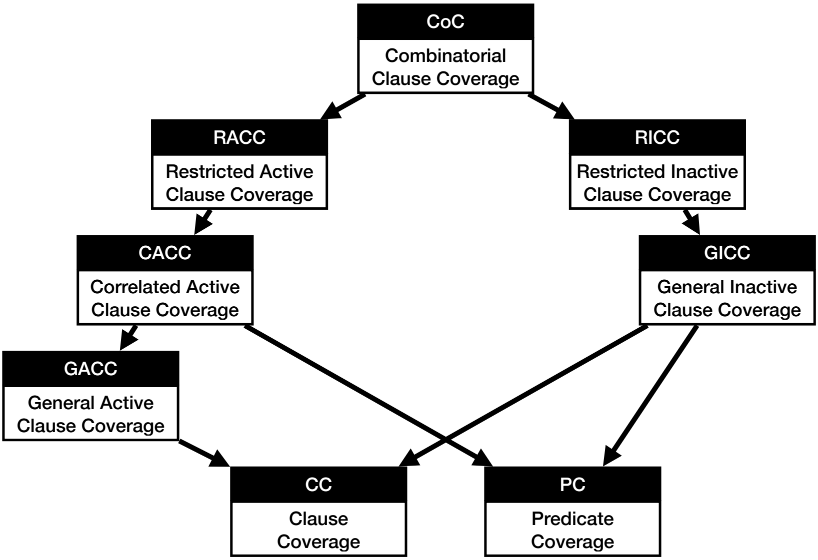
- **Pre-process** – prepare the SUT for fuzz testing
- **Input Generation** – generate test inputs
  - **Mutation-Based Fuzzing** – modify existing test inputs
  - **Generation-Based Fuzzing** – generate new test inputs
- **Test Oracles (Sanitizers)** – detect exceptional outcomes
- **De-duplication** – remove duplicate test inputs

Contents

Coverage – Graph Coverage

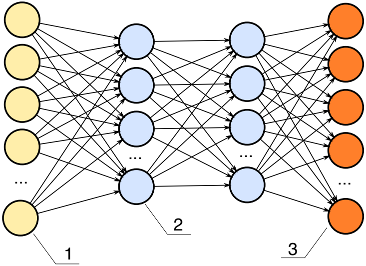


Coverage – Logic Coverage



Coverage – Neuron Coverage

PLRG



$$NC(T,t) = \frac{|\{n \mid \exists x \in T.\ f_\theta(n,x) > t\}|}{|N|}$$

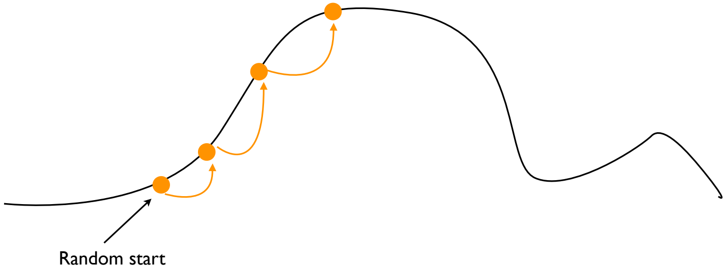
$$KMNC = \frac{\sum_{n \in N} |\{S^n_m \mid \exists x \in T.\ f_\theta(n,x) \in S^n_m\}|}{|N|}$$

$$UNC = \frac{|\{n \mid \exists x \in T.\ f_\theta(n,x) > up_n\}|}{|N|} \quad LNC = \frac{|\{n \mid \exists x \in T.\ f_\theta(n,x) < low_n\}|}{|N|}$$

$$TKNC = \frac{|\bigcup_{x \in T} \bigcup_{1 \le l \le L} top_k(x,l)|}{|N|}$$

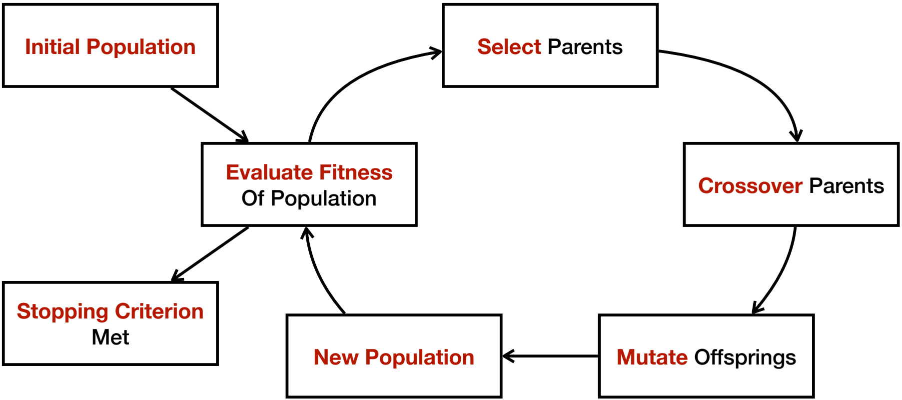
Contents

Search-Based Software Testing – Local Search

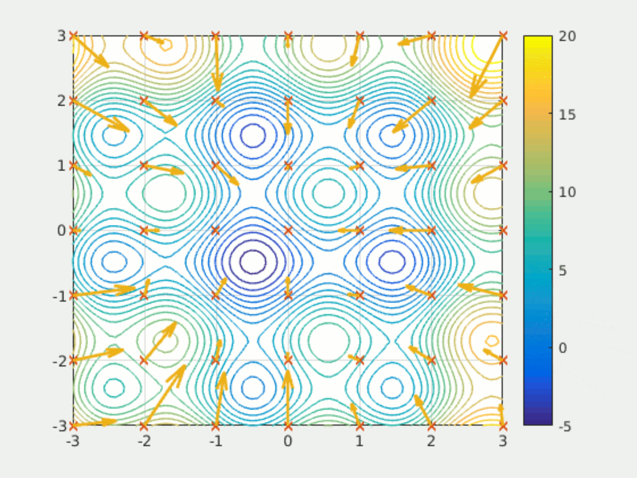


- **Simulated Annealing** – introduce a temperature and gradually decrease it to reduce the probability of accepting worse solutions.
- **Tabu Search** – keep track of the last few moves and avoid revisiting them.

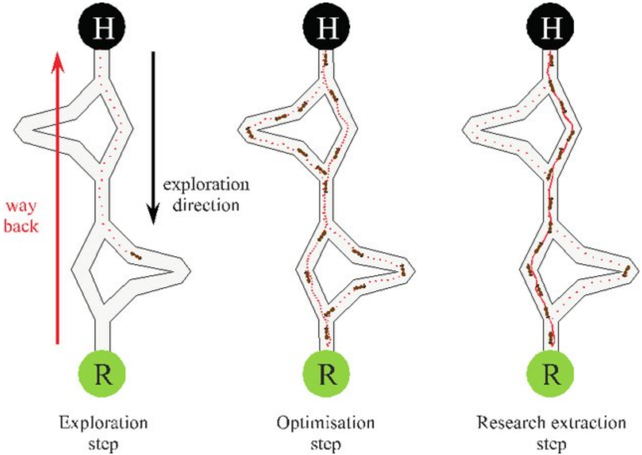
Search-Based Software Testing – Genetic Algorithm



Particle Swarm Optimization (PSO)



Ant Colony Optimization (ACO)

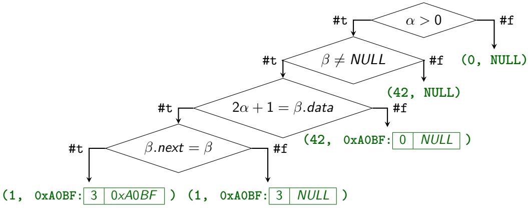


Dynamic Symbolic Execution (DSE)

PLRG

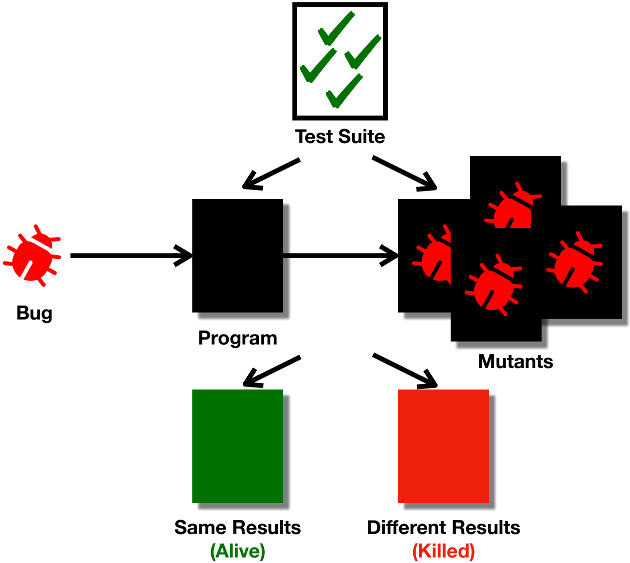
```
class Node {
  int data;
  Node* next;
};

void f(int x, Node *p) {

  if (x > 0)

    if (p != NULL)

      if (x*2+1 == p->data)

        if (p->next == p)
          *
          ERROR;

  return 0;
}
```

$\mathbb{X}$	$\mathbb{V}$	Ω
x	1	α
p	0xA0BF : [3 | 0xA0BF]	β
Φ	$(\alpha > 0) \land (\beta \neq NULL) \land (2\alpha + 1 = \beta.data) \land (\beta.next = \beta)$	



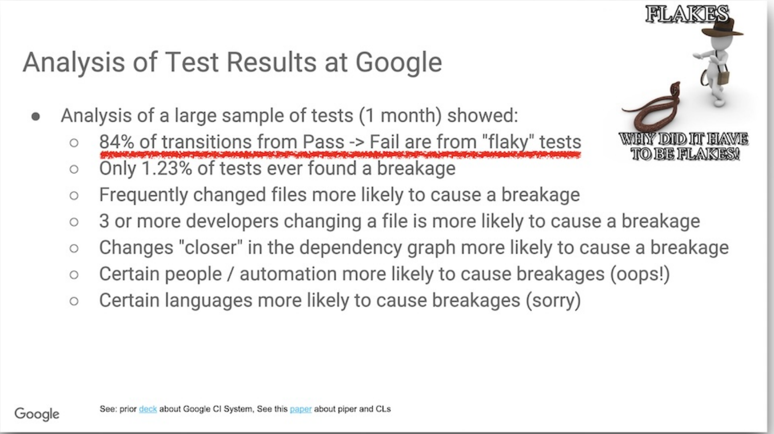
Contents

Testing Adequacy Criteria – Mutation Testing



Testing Adequacy Criteria – Flakiness

PLRG



"The State of Continuous Integration Testing at Google", John Micco, ICST 2017 Keynote (https://research.google/pubs/pub45880/)

Contents

Algorithm Greedy Minimization with Cost

function GreedyMinimization(T, R)
    $T' \leftarrow \emptyset$
    while `true` **do**
        $t \leftarrow \text{argmax}_{t \in T} \frac{|R \cap \text{TR}(t)|}{\text{Time}(t)}$
        if $|R \cap \text{TR}(t)| = 0$ **then**
            break
        $T' \leftarrow T' \cup \{t\}$
        $R \leftarrow R \setminus \text{TR}(t)$
    return T'

TC	r_1	r_2	r_3	r_4	*Time*
t_1	✓	✓			3
t_2		✓		✓	5
t_3		✓	✓	✓	10
t_4			✓		2
t_5			✓	✓	8

$T' = \varnothing$ $R = \{r_1, r_2, r_3, r_4\}$

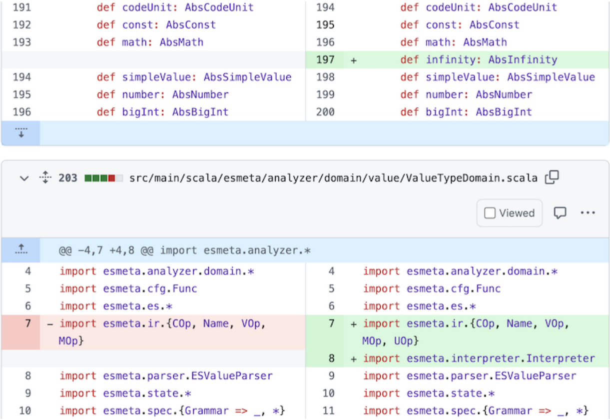
$T' = \{t_1\}$ $R = \{r_3, r_4\}$

$T' = \{t_1, t_4\}$ $R = \{r_4\}$

$T' = \{t_1, t_4, t_2\}$ $R = \varnothing$

Regression Testing – Test Case Selection

- Most of the modern software are developed using the **version control system** (e.g., Git, SVN, etc.).
- The most easiest way is to use the **diff** command provided by the version control system to know the **changed parts** of the program.

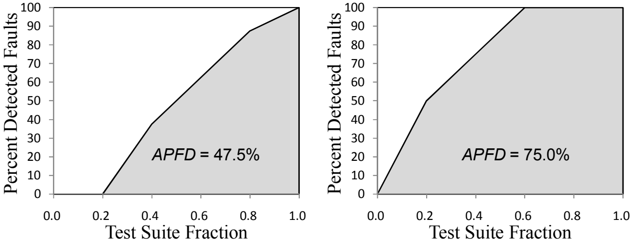


Regression Testing – Test Prioritization

- We can measure the effectiveness of the test case prioritization based on the **average percentage of faults detected** (APFD).
- Intuitively, APFD evaluates the effectiveness of the test case based on the **area** under the curve of the **fault detection rate**.

Example on test suite and faults exposed

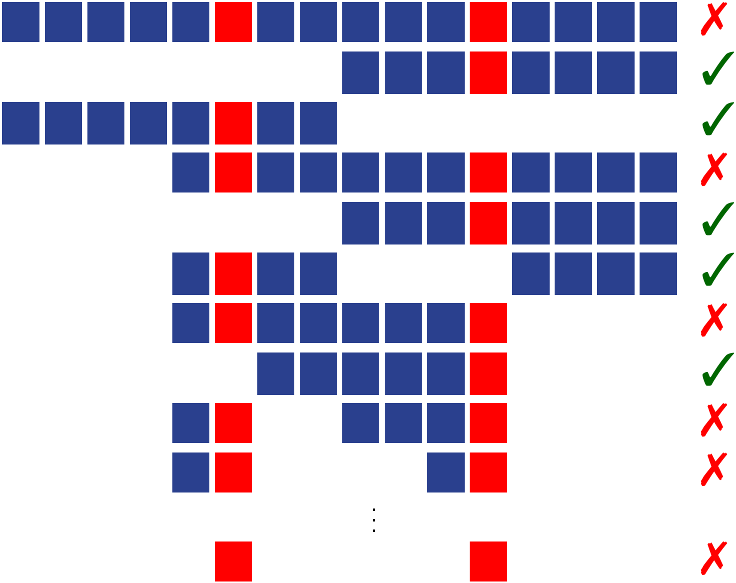
Test Case	Fault							
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
t_A	•		•					•
t_B								
t_C		•		•	•			•
t_D	•	•				•		
t_E			•				•	



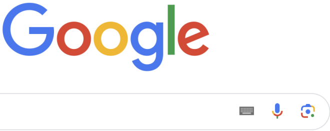
(a) *APFD* for test suite T_1 (b) *APFD* for test suite T_2

Contents

Fault Localization – Delta Debugging



Fault Localization – Information Retrieval



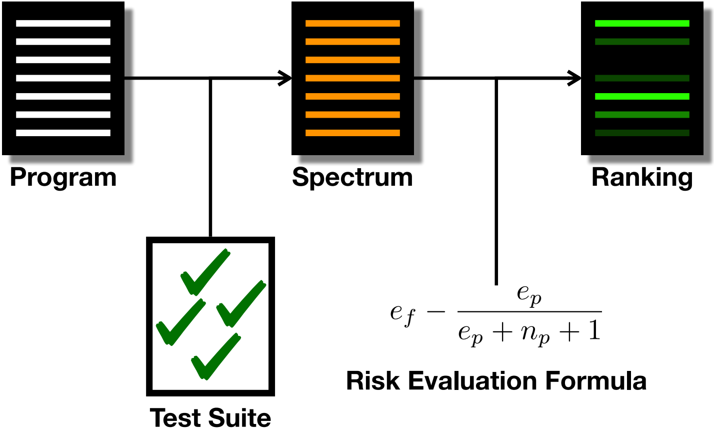
- **Term-Frequency Inverse Document Frequency (Tf-Idf)** is a numerical statistic that reflects how important a term is to a document in a collection or corpus.
- **Term Frequency (tf)**

$$tf(t, d) = \frac{f_{t,d}}{|d|}$$

- **Inverse Document Frequency (idf)**

$$idf(t, D) = \log\left(\frac{|D|}{|\{d \in D \mid t \in d\}|}\right)$$

Fault Localization – Spectrum-Based



$$e_f - \frac{e_p}{e_p + n_p + 1}$$

Fault Localization – Mutation-Based

Proportion of test cases that mutant *m* turns from fail to pass

$$\mu(s) = \boxed{\frac{1}{|mut(s)|} \sum_{m \in mut(s)}} \left(\boxed{\frac{|f_P(s) \cap p_m|}{|f_P|}} - \alpha \cdot \boxed{\frac{|p_P(s) \cap f_m|}{|p_P|}} \right)$$

Proportion of test cases that mutant *m* turns from pass to fail

Proportion of test cases that mutant *m* turns from pass to fail

where α is the **balancing factor**:

$$\alpha = \frac{f2p}{|mut(P)| \cdot |f_P|} \cdot \frac{|mut(P)| \cdot |p_P|}{p2f}$$

Contents

Testing Oracle – Metamorphic Testing

PLRG

Definition (Metamorphic Relationship)

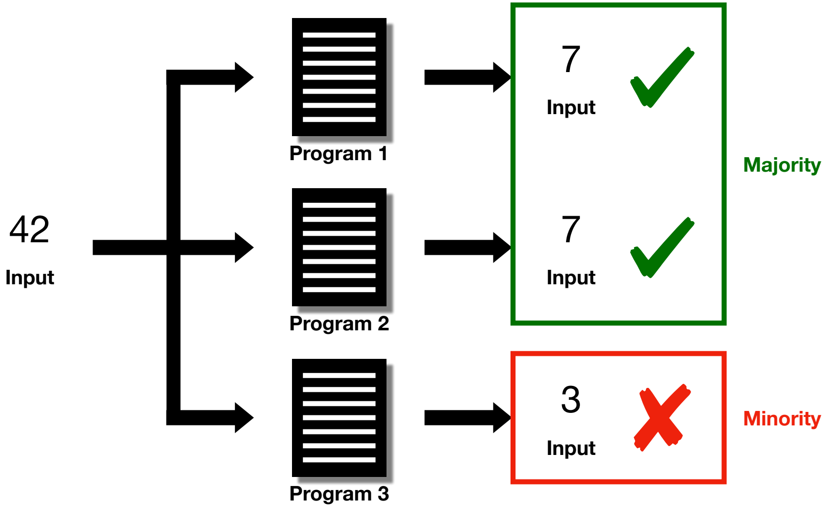
A program $p : X \to Y$ has a **metamorphic relationship** $f : X \to X$ with a relation $R \subseteq Y \times Y$ if and only if:

$$\forall x \in X.\ (p(x), p \circ f(x)) \in R$$

For example, the sin function has the following metamorphic relationships:

f	**Relationship R**
$f(x) = \pi - x$	$\sin(x) = \sin(\pi - x)$
$f(x) = x + \pi$	$\sin(x) = -\sin(x + \pi)$
$f(x) = x + \frac{\pi}{2}$	$\sin^2(x) + \sin^2(x + \frac{\pi}{2}) = 1$

Testing Oracle – Differential Testing



Testing Oracle – Property-based Testing

- A traditional example-based oracle requires **input-output pairs**.

```python
def abs(x):
  if x < 0:
    return -x
  return x

def test_abs():
  assert abs(0) == 0
  assert abs(1) == 1
  assert abs(-1) == 1
```

- A **property-based oracle** requires the **properties** of a given input.

```python
def abs(x):
  if x < 0:
    return -x
  return x

def test_abs(x):
  assert abs(x) >= 0
  assert abs(x) == abs(-x)
  assert abs(abs(x)) == abs(x)
```

Contents

Course Project

The last part of the course is the **course project (Due: 05/20)**.

There are two presentation sessions:

- **05/20 (Mon.) – Project Presentation I**
- **05/22 (Wed.) – Project Presentation II**

The grading criteria are as follows:

- **30% – Topic Selection**
- **30% – Results**
- **40% – Presentation**

Let's decide the **topic** of the course project!

Please send me an email with the **GitHub repository link** for your project.

- I hope you enjoyed the class!

Jihyeok Park
jihyeok_park@korea.ac.kr
https://plrg.korea.ac.kr