实验报告

Mutator

我们组实现了七个变异器。

```
        self.mutators = [
            insert_random_character,
            flip_random_bits,
            arithmetic_random_bytes,
            interesting_random_bytes,
            havoc_random_insert,
            havoc_random_replace,
            delete_random_bytes,
        ]
```

前六个给定的`Mutator`如何实现就不展示了，主要是额外实现的`Mutator`。

我们组额外实现的`delete_random_bytes`基于ALF变异算法，随机地挑选一个起始位置后，删除其后的连续N个字节。$N = 1, 2, 4$。通过直接移除部分数据，达到破坏数据结构完整性的目的。

其实现代码为：

```
def delete_random_bytes(s: str) -> str:
    if not s:
        return s
    bytes_arr = bytearray(s.encode())
    if not bytes_arr:
        return s
    possible_n = [n for n in [1, 2, 4] if n <= len(bytes_arr)]
    if not possible_n:
        return s
    n = random.choice(possible_n)
    start_pos = random.randint(0, len(bytes_arr) - n)
    del bytes_arr[start_pos : start_pos + n]

    return bytes_arr.decode(errors="ignore")
```

PowerSchedule

能量调度基于种子的能量选择要执行的种子，代码给定的基类是一个十分简单的随机挑选算法，每个种子有相同的概率被选择。

PathPowerSchedule

实验要求实现`PathPowerSchedule`，它是以稀有路径为高优先级的调度算法，能够触发较少被遍历路径的种子将被分配更多能量。

为了实现这一点，需要两个内部存储变量：

```
def __init__(self) -> None:
    # 路径频率统计，类型为{path : frequency}
    self.path_frequency = {}

    # 种子路径映射图，类型为{seed_id : path}
    self.seed_path_map = {} = {}
```

`path_frequency`以路径为`key`，以其出现次数为`value`，`seed_path_map`以种子id为`key`，将种子映射到其触发的路径上。

对于一个特定的种子，其能量应该与其路径频率成反比，即：

```
# 获取该 seed 触发的路径
path = self.seed_path_map.get(seed.id, None)
# 获取该路径的频率，默认为1
freq = self.path_frequency.get(path, 1)
# 能量分配：基础能量为10，频率越低能量越高，最小为1
seed.energy = max(1, int(10 / freq))req))
```

然后根据能量进行归一化加权即运行基类`PowerSchedule`的`choose()`即可。

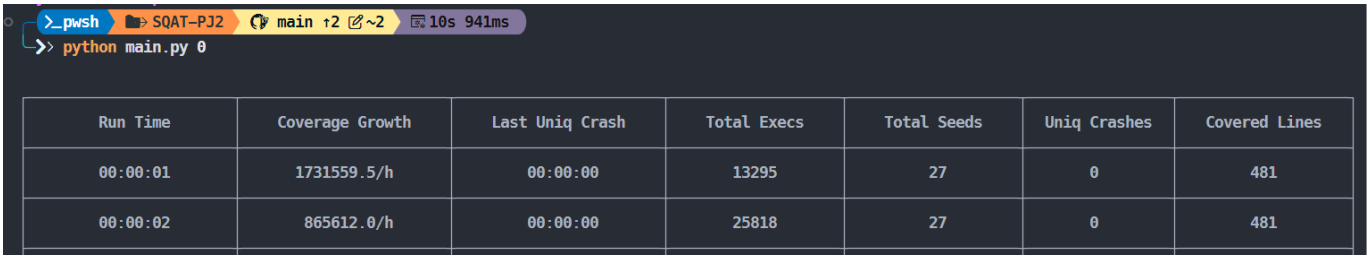
SeedAwarePowerSchedule

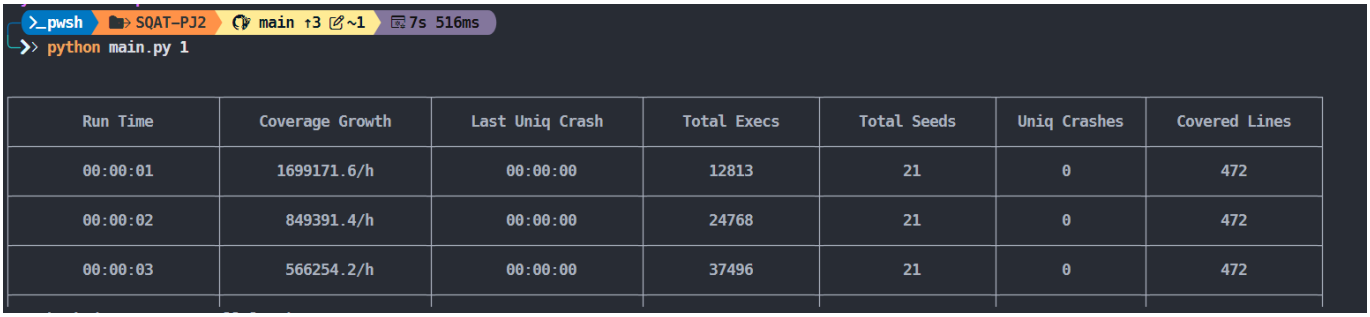
我们组额外实现`SeedAwarePowerSchedule`，种子的年龄低将被优先选择，年龄低、对覆盖率增长贡献大的种子会被分配更多能量，其代码对应于：

```
# 获取种子元信息
default_meta: Tuple[float, float] = (current_time, 0.0)
create_time, coverage_gain = self.seed_metadata.get(seed.id, default_meta)

# 计算年龄因子（单位：小时）
age = max(1.0, (current_time - create_time) / 3600)  # 防止除零

# 能量计算 = 基础能量 * 覆盖增长率 / 年龄
base_energy = 10.0
seed.energy = max(1, int(base_energy * (coverage_gain + 0.1) / age))
```

调用检验

在`main.py`中，输入参数`0`使用路径算法，输入参数`1`使用年龄算法，两者都能正常运行：





Seed本地持久化

我们通过将需要保存的种子进行序列化以提供存储的信息:

```
    @staticmethod
    def _generate_id(data: str) -> int:
        """基于数据内容生成唯一ID"""
        return hash(data)

    def __getstate__(self):
        """控制序列化内容"""
        return (self.data, self.coverage, self.energy, self.id)

    def __setstate__(self, state):
        """反序列化"""
        self.data, self.coverage, self.energy, self.id = state

    def __str__(self) -> str:
        """Returns data as string representation of the seed"""
        return self.data
```

在调度基类中，利用`ObjectUtil`类的操作进行种子保存与加载：

```
    def persist_seed(self, seed: Seed):
        os.makedirs("corpus", exist_ok=True)
        file_path = f"corpus/seed_{seed.id}.pkl"
        # 保存种子数据
        dump_object(file_path, seed.data)
        # 更新注册表...

    def load_seed(self, seed_id: int) -> Seed:
        if seed_id in self.memory_cache:
            return self.memory_cache[seed_id]
        # 从磁盘加载
        if seed_id in self.seed_registry:
            # 加载操作并返回种子
```

最后，在灰盒模糊测试基类初始化时，添加加载种子的方法，即可实现持久化种子与加载的操作：

```
def __init__(
        self, seeds: List[str], schedule: PowerSchedule, is_print: bool
    ) -> None:
        ...
        # 加载持久化的种子
        top_seeds = self._load_top_seeds(50)  # 加载前50个高能量种子
        ...
def _load_top_seeds(self, count: int) -> List[Seed]:
    sorted_seeds = sorted(
        self.schedule.seed_registry.items(),
        key=lambda x: x[1]["energy"],
        reverse=True
    )[:count]
    return [self.schedule.load_seed(sid) for sid, _ in sorted_seeds]
```

测试结果

测试结果在_result文件夹下， 可通过 `python view_results.py` 查看

```
PS D:\课程学习资料\软件质量测试\SQAT-PJ2> python view_results.py

=== Sample-1.pkl ===
Covered Lines: 10
Unique Crashes: 6
Runtime: 7200.99 seconds

=== Sample-2.pkl ===
Covered Lines: 15
Unique Crashes: 4
Runtime: 60.10 seconds

=== Sample-3.pkl ===
Covered Lines: 8
Unique Crashes: 5
Runtime: 600.47 seconds

=== Sample-4.pkl ===
Covered Lines: 604
Unique Crashes: 2
Runtime: 600.02 seconds
```