project 2 白盒测试

1. 测试方法

其中语句覆盖是一种最弱的覆盖，判定覆盖和条件覆盖比语句覆盖强，满足判定/条件覆盖标准的测试用例一定也满足判定覆盖、条件覆盖和语句覆盖，条件组合覆盖是除路径覆盖外最强的，路径覆盖也是一种比较强的覆盖，但未必考虑判定条件结果的组合，并不能代替条件覆盖和条件组合覆盖。

2. InfoGain.java

该类共有8个方法：

```
public InfoGain(ArrayList<String[]> trainData, int decatt);
public double getEntropy(Map<String, Integer> attributeNum);
public double getEntropy(ArrayList<Integer> subset, int attributeIndex);
public int getGainRatioMax(ArrayList<Integer> subset, LinkedList<Integer> selatt);
public boolean isPure(Map<String,Integer> targetNum);
public  Map<String,Integer> get_AttributeNum(ArrayList<Integer> subset, int attributeIndex );
public Map<String, ArrayList<Integer>> get_AttributeSubset(ArrayList<Integer> subset, int attributeIndex);
public String get_targetValue(Map<String,Integer> targetNum);
```

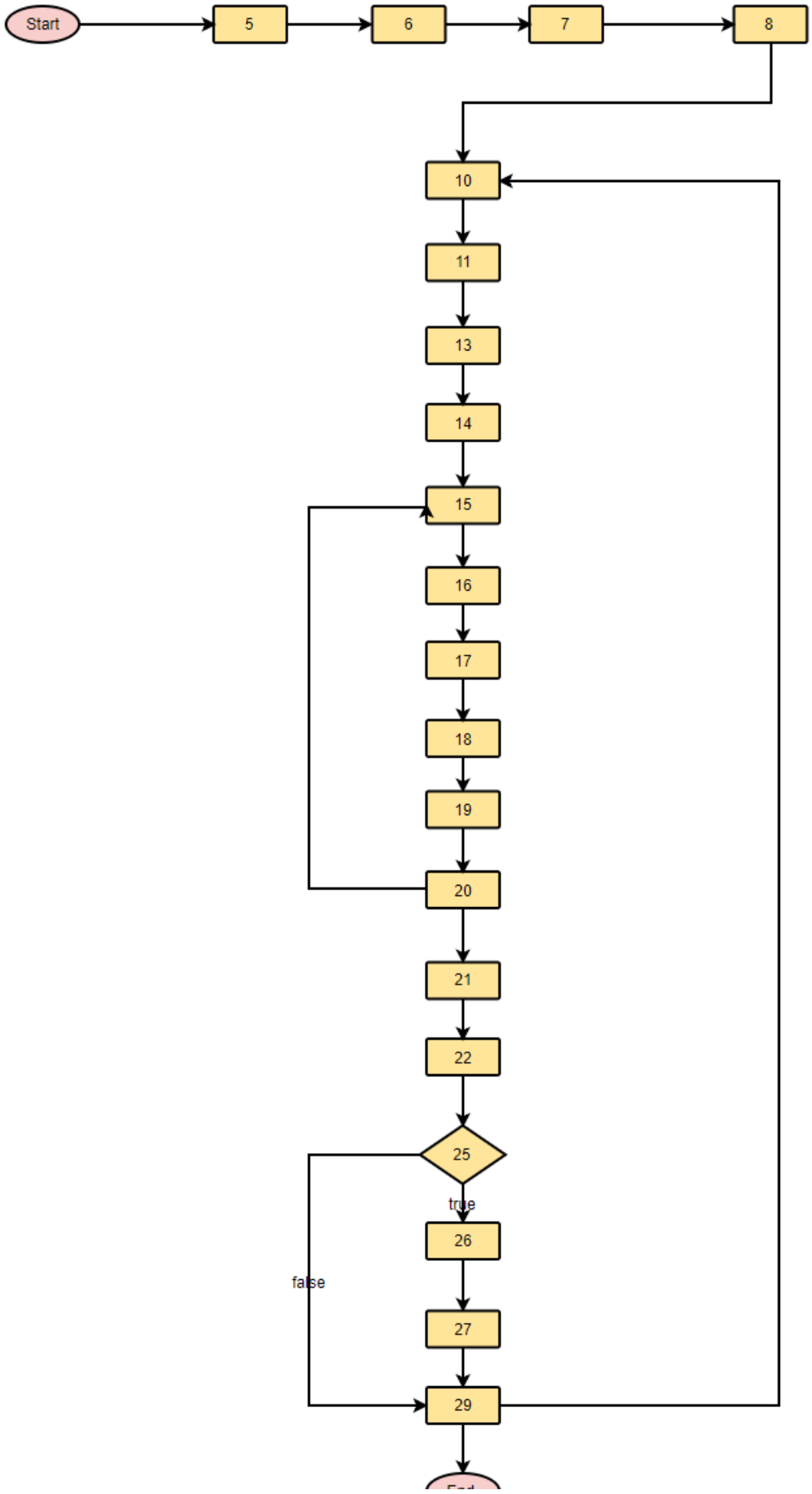
以下选取`getGainRatioMax()`详细介绍。

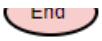
> InfoGain.java `getGainRatioMax`

```
//信息熵增益率相关
public int getGainRatioMax(ArrayList<Integer> subset, LinkedList<Integer> selatt){
    //计算原信息熵

    Map<String, Integer> old_TargetNum = get_AttributeNum(subset, decatt);
    double oldEntropy = getEntropy(old_TargetNum);
    double maxGainRatio=0;
    int maxIndex=decatt;

    for(int attributeIndex: selatt){
        Map<String, ArrayList<Integer>> attributeSubset = get_AttributeSubset(subset, attributeIndex);

        int sum = 0;
        double newEntropy = 0;
        for(ArrayList<Integer> tempSubset: attributeSubset.values()){
            int num = tempSubset.size();
```

```
            sum += num;
            double tempEntropy = getEntropy(tempSubset,decatt);
            newEntropy += num * tempEntropy;
        }
        newEntropy /= sum;
        double tempGainRatio = (oldEntropy - newEntropy)/getEntropy(subset,
attributeIndex);  //计算信息增益率

        //如果信息增益率为负，应该停止分支，此处避免麻烦没有做进一步讨论。
        if(tempGainRatio > maxGainRatio){
            maxGainRatio = tempGainRatio;
            maxIndex = attributeIndex;
        }
    }
    return  maxIndex;
}
```

这段代码的程序流程图如图所示：(代码行数从`//信息熵增益率相关`开始，并且从1开始计数，本文件的pdf版中的代码包含行数)。


Start
5
6
7
8
10
11
13
14
15
16
17
18
19
20
21
22
25
true
26
false
27
29



程序路径有2条：

1. 5, 6, 7, 8, 10, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 25, 26, 27, 29
2. 5, 6, 7, 8, 10, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 25, 26, 27, 29

对于测试的代码来说，语句覆盖率必须要大于95%，然后依据设计的测试用例严格程度可以分为6种：

1. 语句覆盖测试用例设计： 设计合适的测试用例使得所有的语句都被覆盖。

```
@Test
void TestGetGainRatioMax(){
    InfoGain ig=initInfoGain2();
    ArrayList<Integer> idxlist=new ArrayList<>();
    idxlist.add(0);idxlist.add(1);
    LinkedList<Integer> ll=new LinkedList<Integer>();
    ll.add(0);ll.add(1);ll.add(2);ll.add(3);
    assertEquals(0,ig.getGainRatioMax(idxlist,ll));
    idxlist.add(2);idxlist.add(3);
    assertEquals(0,ig.getGainRatioMax(idxlist,ll));
    InfoGain ig2=initInfoGain3();
    assertEquals(1,ig2.getGainRatioMax(idxlist,ll));
}
```

2. 判定覆盖测试用例设计： 设计合适的输入使得判断语句（本函数之中只有line 25是判断语句）的每个取值分支都至少经历一次。 也即使得 `tempGainRatio > maxGainRatio` 和 `tempGainRatio <= maxGainRatio` 都至少经历一次。

优点：较语句覆盖可以有更强的覆盖，简单易理解，只需要关注单个判定，无需细分即可得到测试用例。

缺点：大部分的判定条件都是多个组合起来的（or/and/case），若仅判断整个条件的结果，会遗漏部分数据类型测试路径。

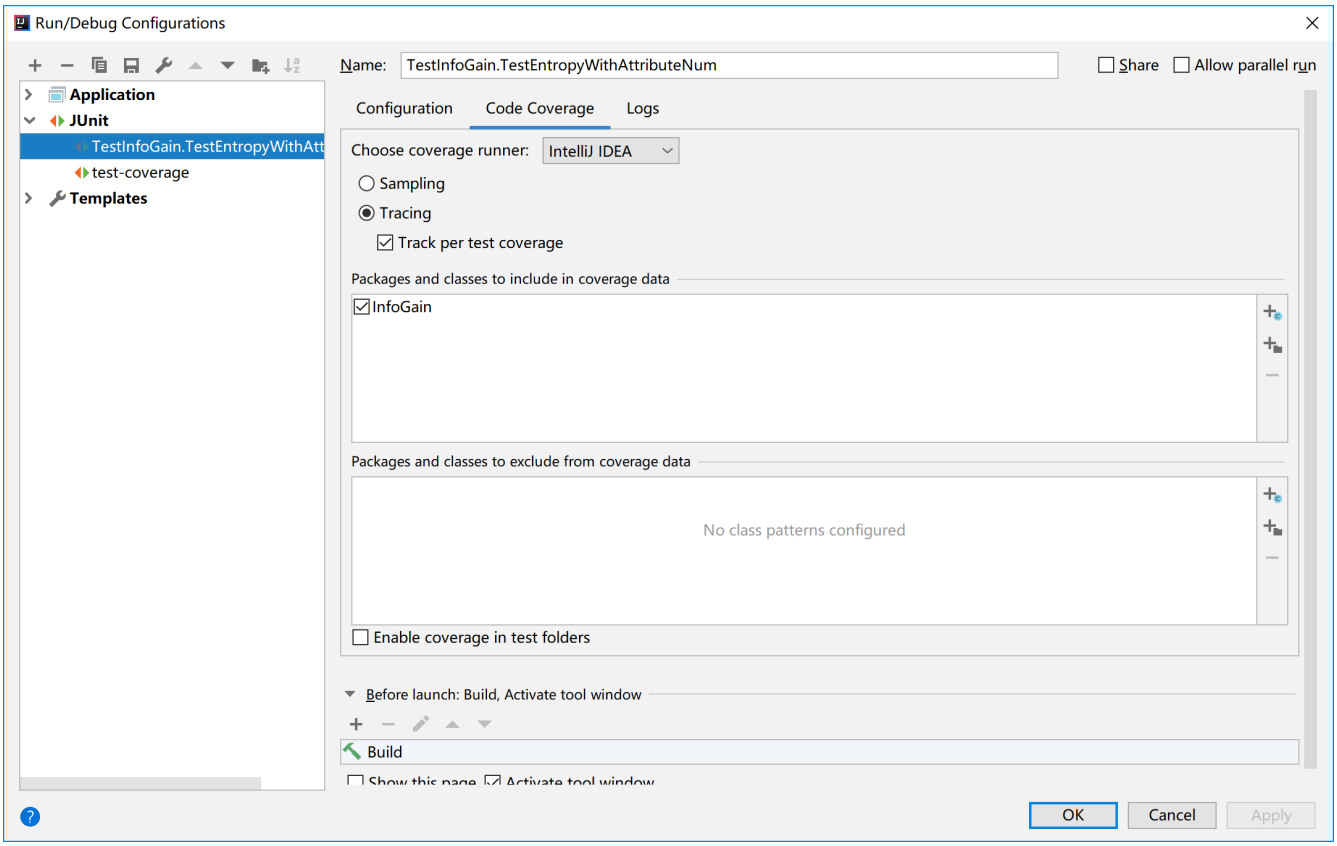
3. 路径覆盖测试用例设计： 保证每条可能执行到的路径至少执行一次。

4. 条件覆盖测试用例设计： 选择足够的测试用例，使得运行这些测试用例时，判定中每个条件的所有可能结果至少出现一次，但未必能覆盖全部分支。

条件覆盖要检查每个符合谓词的子表达式值为真和假两种情况，要独立衡量每个子表达式的结果，以确保每个子表达式的值为真和假两种情况都被测试到。

5. 判定条件覆盖测试用例设计： 判定-条件覆盖就是设计足够的测试用例，使得判断中每个条件的所有可能取值至少执行一次，同时每个判断的所有可能判断结果至少执行，即要求各个判断的所有可能的条件取值组合至少执行一次。
6. 条件组合覆盖设计用例设计： 使所有判定中各条件判断结果的所有组合至少出现一次，满足这种覆盖标准成为条件组合覆盖。这是算是覆盖最全的了。

通过设置Junit测试类的运行configuration：



并且选择`Run with coverage`就可以查看测试用例的覆盖率。（这里提一下助教的要求：代码覆盖率不低于95%）