华东师范大学软件学院课程作业

课程名称：软件质量分析	年级：2023 级本科	姓名：张梓卫
作业主题：面向模块的软件可信性度量模型 T_{na}	学号：10235101526	作业日期：2024/11/27
指导老师：陈仪香	组号：	

目录

一 作业一试证明

1 Extensive 结构定义

解答 一 .1: Extensive

Definition (Extensive 结构)
一个经验关系系统 $(A, R, \circ)$ 称为具有 Extensive 结构，若下列各条成立:
1. 弱序性: (A, R) 是一个弱序，即 R 满足传递性和强完备性，强完备性是指对 $\forall a, b \in A$, 有 $(a, b) \in R$ 或 $(b, a) \in R$;
2. 弱结合律: $\forall a, b, c \in A : (a \circ b) \circ c \approx a \circ (b \circ c)$;
3. 单调性: $\forall a, b, c \in A : (a, b) \in R$ 当且仅当 $((a \circ c), (b \circ c)) \in R$ 当且仅当 $((c \circ a), (c \circ b)) \in R$;
4. 阿基米德性: 如果 $\forall a, b, c, d \in A :$ 如果 $(a, b) \in R_s$, 则存在正整数 n 使得 $(na \circ c), (nb \circ d)) \in R_s$ 成立。

2 完整证明

Prove:

需要验证 **4** 条基本性质：弱序性、弱结合律、单调性和阿基米德性。

给定两个模块 m_1 和 m_2，$m_1 \approx m_2$ 当且仅当 $m_1 \succsim_T m_2$ 与 $m_2 \succsim_T m_1$ 都成立。

由关系 $\succsim_T$ 定义可得：$m_1 \approx m_2$ 当且仅当 $\kappa(m_1) = \kappa(m_2)$。

(1) $\succsim_T$ 是弱序的：验证 $\succsim_T$ 是传递的和强完备的。

(a) 传递性：要证明：若 $m_1 \succsim_T m_2$ 且 $m_2 \succsim_T m_3$，则有 $m_1 \succsim_T m_3$。

由假设 $m_1 \succsim_T m_2$，有 $\kappa(m_1) \geq \kappa(m_2)$；同时，假设 $m_2 \succsim_T m_3$，则 $\kappa(m_2) \geq \kappa(m_3)$。因此，$\kappa(m_1) \geq \kappa(m_3)$，由此得出 $m_1 \succsim_T m_3$。

(b) 强完备性：

对于任意 $m_1, m_2 \in A$，根据强完备性定义，要证明 $(m_1 \succsim_T m_2)$ 或 $(m_2 \succsim_T m_1)$ 必定成立。对于任意两元素

m_1, m_2，由于 κ 是一个确定的函数，且总有 $\kappa(m_1)$ 与 $\kappa(m_2)$ 之间的关系，因此总有 $m_1 \succsim_T m_2$ 或 $m_2 \succsim_T m_1$，满足强完备性条件。

(2) 弱结合律：因为 $m_1 \circ_T m_2 = (E_{m_1} \uplus E_{m_2}, R_{m_1} \cup R_{m_2} \cup R_{m_1 m_2})$，所以

$$(m_1 \circ_T m_2) \circ_T m_3 = ((E_{m_1} \uplus E_{m_2}) \uplus E_{m_2}, \\ R_{m_1} \cup R_{m_2} \cup R_{m_3} \cup R_{m_1 m_2} \cup R_{m_1 m_3} \cup R_{m_2 m_3} \cup R_{m_1 m_2 m_3})$$

由定义可知 $(m_1 \circ_T m_2) \circ_T m_3$ 满足结合律，所以有

$$(m_1 \circ_T m_2) \circ_T m_3 \approx m_1 \circ_T (m_2 \circ_T m_3)$$

(3) 单调性：首先证明 $\forall m_1, m_2, m_3 \in MS$，若 $m_1 \succsim_T m_2$，则 $m_1 \circ_T m_3 \succsim_T m_2 \circ_T m_3$。
由 $m_1 \succsim_T m_2$ 得 $\kappa(m_1) \geq \kappa(m_2)$，所以有

$$\kappa(m_1 \circ_T m_3) = \kappa(m_1) + \kappa(m_3) \geq \kappa(m_2) + \kappa(m_3) = \kappa(m_2 \circ_T m_3)$$

故 $m_1 \circ_T m_3 \succsim_T m_2 \circ_T m_3$。
同理，$\forall m_1, m_2, m_3 \in MS$，若 $m_1 \succsim_T m_2$，则 $m_3 \circ_T m_1 \succsim_T m_3 \circ_T m_2$。
这样有 $m_1 \circ_T m_3 \succsim_T m_2 \circ_T m_3$。
接下来要证明 $\forall m_1, m_2, m_3 \in MS$，若 $m_1 \circ_T m_3 \succsim_T m_2 \circ_T m_3$，
则 $m_1 \succsim_T m_2$。

(4) 阿基米德性：
设 $m_1 \succsim_T m_2$ 且 $m_2 \not\succsim_T m_1$，则 $\kappa(m_1) \geq \kappa(m_2)$ 且 $\kappa(m_1) \neq \kappa(m_2)$。
取正整数 n 使得 $n \geq \frac{\kappa(m_4)-\kappa(m_3)}{\kappa(m_1)-\kappa(m_2)}$ 成立。进而有

$$n\kappa(m_1) + \kappa(m_3) \geq n\kappa(m_2) + \kappa(m_4)$$

$$nm_1 \circ_T m_3 \succsim_T nm_2 \circ_T m_4$$

综上所述 $(MS, \succsim_T, \circ_T)$ 具有 Extensiveness 结构。 ■

二 作业二：面向模块的软件可信性度量模型 T_{na}

1 分析代码行数与期望变量数量

我们首先把需要进行分析的代码贴上来：

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
typedef struct {
    int nodeU;
    int nodeV;
    int weight;
} Edge; // 以上的代码头文件以及 Edge 结构体定义在后面各模块都要使用到，故不记载至模块中
```

kruskal.c

```c
// 此模块共有 9 行代码，定义为 EM1，含有 2 个变量 {edgeA, edgeB}
int compare(Edge edgeA, Edge edgeB) {
    if(edgeA.weight > edgeB.weight) {
```

```
        return 1;
    } else if(edgeA.weight < edgeB.weight) {
        return -1;
    } else {
        return 0;
    }
}
```

EM1.c

此模块中，变量 edgeA, edgeB 均只有当前一个用途，且具有自描述性，是用户所期望的。

因此，$\#\ (E_{m_1})_{Except} = 2$。

```
// 此模块共有 22 行代码，定义为 EM2, 含有 9 个变量 {edgeArray, firstIndex, midIndex, lastIndex, X, i, j, k, q}
void merge(Edge edgeArray[], int firstIndex, int midIndex, int lastIndex) {
    int q = lastIndex - firstIndex + 1;
    Edge *X = (Edge *)malloc(q * sizeof(Edge));
    int i = firstIndex, j = midIndex + 1, k = 0;
    while(i <= midIndex && j <= lastIndex) {
        if(compare(edgeArray[i], edgeArray[j]) < 0) {
            X[k++] = edgeArray[i++];
        } else {
            X[k++] = edgeArray[j++];
        }
    }
    while(i <= midIndex) {
        X[k++] = edgeArray[i++];
    }
    while(j <= lastIndex) {
        X[k++] = edgeArray[j++];
    }
    for(k = 0; k < q; k++){
        edgeArray[firstIndex + k] = X[k];
    }
    free(X);
}
```

EM2.c

此模块中，变量 edgeArray[], firstIndex, lastIndex, i 在其他函数中也被使用, 其余变量 midIndex, X, j, k, q 均只有当前一个用途，且具有自描述性，是用户所期望的。

因此，$\#\ (E_{m_2})_{Except} = 5$。

```
// 此模块共有 9 行代码，定义为 EM3, 含有 4 个变量 {edgeArray[], firstIndex, lastIndex, a}
void mergeSort(Edge edgeArray[], int firstIndex, int lastIndex) {
    if(firstIndex >= lastIndex) {
        return;
    }
    int a = (firstIndex + lastIndex) / 2;
    mergeSort(edgeArray, firstIndex, a);
    mergeSort(edgeArray, a + 1, lastIndex);
    merge(edgeArray, firstIndex, a, lastIndex);
}
```

EM3.c

此模块中，变量 edgeArray[], firstIndex, lastIndex 在其他函数中也被使用, a 变量只有当前一个用途，且具有自描述性，是用户所期望的。

因此，$\#\ (E_{m_3})_{Except} = 1$。

```
// 此模块共有 5 行代码，定义为 EM4, 含有 3 个变量 {parent[], numberOfNodes, i}
int initializeParent(int parent[], int numberOfNodes) {
    for(int i = 0; i < numberOfNodes; i++) {
        parent[i] = i;
```

```
    }
}
```

EM4.c

此模块中，变量 parent[], numberOfNodes, i 在其他函数中也被使用。

因此，$\#\ (E_{m_4})_{Except} = 0$。

```
// 此模块共有 7 行代码，定义为 EM5，含有 2 个变量 {parent[], node}
int findRoot(int parent[], int node) {
    if(parent[node] == node) {
        return node;
    } else {
        return parent[node] = findRoot(parent, parent[node]);
    }
}
```

EM5.c

此模块中，变量 parent[], node 在其他函数中也被使用。

因此，$\#\ (E_{m_5})_{Except} = 0$。

```
// 此模块共有 10 行代码，定义为 EM6，含有 5 个变量 {parent[], nodeU, nodeV, rootOfNodeU, v}
bool join(int parent[], int nodeU, int nodeV) {
    int rootOfNodeU = findRoot(parent, nodeU);
    int v = findRoot(parent, nodeV);
    if(rootOfNodeU != v) {
        parent[v] = rootOfNodeU;
        return true;
    } else {
        return false;
    }
}
```

EM6.c

此模块中，变量 parent[] 在其他函数中也被使用，其余变量 nodeU, nodeV, rootOfNodeU, v 均只有当前一个用途，且具有自描述性，是用户所期望的。

因此，$\#\ (E_{m_6})_{Except} = 4$。

```
// 此模块共有 17 行代码，定义为 EM7，含有 7 个变量 {numberOfNodes, edgeArray[], numberOfEdges, resultEdgeArray
    [], parent[], i, y}
bool kruskal(int numberOfNodes, Edge edgeArray[], int numberOfEdges, Edge resultEdgeArray[]) {
    int *parent = (int *)malloc(numberOfNodes * sizeof(int));
    initializeParent(parent, numberOfNodes);
    mergeSort(edgeArray, 0, numberOfEdges - 1);
    int y = 0;
    for (int i = 0; i < numberOfEdges; i++) {
        if(join(parent, edgeArray[i].nodeU, edgeArray[i].nodeV)) {
            resultEdgeArray[y++] = edgeArray[i];
        }
    }
    free(parent);
    if(y == numberOfNodes - 1) {
        return true;
    } else {
        return false;
    }
}
```

EM7.c

此模块中，变量 numberOfNodes, edgeArray[], numberOfEdges, parent[], i 在其他函数中也被使用，其余变量 resultEdgeArray[], y 均只有当前一个用途，且具有自描述性，是用户所期望的。

因此，$\#\ (E_{m_2})_{Except} = 2$。

```c
// 此模块共有 27 行代码，定义为 EM8
// 含有 6 个变量 {numberOfNodes, numberOfEdges, edgeArray[], resultEdgeArray, i, weightSum}
int main() {
    int numbrOfNodes, numberOfEdges;
    printf("Input number of nodes: ");
    scanf("%d", &numbrOfNodes);
    printf("Input number of edges: ");
    scanf("%d", &numberOfEdges);
    Edge *edgeArray = (Edge *)malloc(numberOfEdges * sizeof(Edge));
    printf("Input edges: \n");
    for(int i = 0; i < numberOfEdges; i++) {
        scanf("%d %d %d", &edgeArray[i].nodeU, &edgeArray[i].nodeV, &edgeArray[i].weight);
    }
    Edge *resultEdgeArray = (Edge *)malloc((numbrOfNodes - 1) * sizeof(Edge));
    if(kruskal(numbrOfNodes, edgeArray, numberOfEdges, resultEdgeArray)) {
        printf("Edges in MST: \n");
        int weightSum = 0;
        for(int i = 0; i < numbrOfNodes - 1; i++) {
            printf("%d %d %d\n", resultEdgeArray[i].nodeU, resultEdgeArray[i].nodeV, resultEdgeArray[i].weight);
            weightSum += resultEdgeArray[i].weight;
        }
        printf("Sum of weights: %d", weightSum);
    } else {
        printf("No Answer");
    }
    free(resultEdgeArray);
    free(edgeArray);
    return 0;
}
```

EM8.c

此模块中，变量 numbersOfNodes, numberOfEdges, edgeArray[], resultEdgeArray, i 在其他函数中也被使用, 其余变量 weightSum 只有当前一个用途，且具有自描述性，是用户所期望的。

因此，$\#\ (E_{m_2})_{Except} = 1$。

接下来要对每个模块的 $\#\ (E_{m_i})_{Except}, i \in 1,2,3,4,5,6,7,8$ 值进行分析：

模块	代码行数	变量数量 (except)
EM1	9	2
EM2	22	5
EM3	9	1
EM4	5	0
EM5	7	0
EM6	10	4
EM7	17	2
EM8	27	1
总计	106	38(Total)

表 1: 各模块代码行数与变量数量的 except 值

2 计算 $\kappa(m_i)$ 值与 $\beta(m_i)$ 的值

具有用户期望机制实现程序元素个数与 E 中所有程序元素个数的比值:

$$\kappa(m) = \frac{\#(E_m)_{\text{expect}}}{\#(E)}$$

$$\beta(m) = \frac{\text{模块}m\text{ 所含有的代码函数}}{\text{代码中参与划分模块的总行数}}$$

模块	except 值	$\kappa(m_i)$	代码行数	$\beta(m_i)$
EM1	2	0.053	9	0.085
EM2	5	0.132	22	0.208
EM3	1	0.026	9	0.085
EM4	0	0.000	5	0.047
EM5	0	0.000	7	0.066
EM6	4	0.105	10	0.094
EM7	2	0.053	17	0.160
EM8	1	0.026	27	0.255
总计	38(E_{QS})	-	106	-

表 2: 各模块 $\kappa(m_i)$ 和 $\beta(m_i)$ 值计算表（保留三位有效数字）

3 实值测量函数计算

给定一个程序系统 $S=(E,R)$，令 MS 为该系统中所有模块组成的集合，$\mathbb{R}^+$ 为正实数组集，$m=(E_m,R_m)\in MS$, $0<\rho<1$。则模块可信性经验关系系统 $(MS,\succeq_T,\circ_T)$ 的实值测量函数 μ_{na} 定义如下：

$$\mu_{\mathrm{na}}(m)=(\kappa(m))^{\rho}$$

```python
def mu_na(kappa, rho):
    return kappa ** rho
```

mu_na.py

在计算时，我们采用放大比值的方法，根据以下的映射关系进行计算：

$$\mu(m)=\begin{cases}1, & \mu_{\mathrm{na}}(m)<\frac{1}{10}\\ 10\cdot\mu_{\mathrm{na}}(m), & \text{其它情况}\end{cases}$$

代码实现：

```python
def measure(kappa, rho):
    if kappa < 0.1:
        return 1
    else:
        return 10 * kappa ** rho
```

measure.py

4 计算 T_{na} 值

$$T_{\mathrm{na}}(S)=\left(\sum_i \beta(m_i)\cdot(\mu(m_i))^{\rho}\right)^{\frac{1}{\rho}}$$

代码实现：

```python
    def T_na(modules, rho):
    """
    计算 T_na 的值
    :param modules: 包含模块 (beta, kappa) 的列表，例如 [(beta1, kappa1), (beta2, kappa2), ...]
    :param rho: 控制公式计算的参数 rho
    :return: T_na 的值
    """
    total = 0
    for beta, kappa in modules:
        mu_value = measure(kappa, rho)  # 使用 measure 计算 (m)
```

```
        total += beta * (mu_value ** rho)
    return total ** (1 / rho)
```

T_{na}.py

传入 ρ 值列表，计算 T_{na} 值：

```
rho_values = [0.8, 0.6, 0.4, 0.2, 0.1, 0.01, 0.001]
for rho in rho_values:
    T_na_value = T_na(modules, rho)
    print(f"rho = {rho}, T_na(S) = {T_na_value:.6f}")
```

main.py

结果如下所示:

```
D:\Python\python.exe F:\Project\Python\SoftwareQuality\Main8.py
rho = 0.8, T_na(S) = 1.252422
rho = 0.6, T_na(S) = 1.477098
rho = 0.4, T_na(S) = 1.713976
rho = 0.2, T_na(S) = 1.909806
rho = 0.1, T_na(S) = 1.972840
rho = 0.01, T_na(S) = 2.002815
rho = 0.001, T_na(S) = 2.004322
```

图 1: 不同 ρ 值下的 T_{na} 值

5 δ_{ij} 的值

在 PPT 中还提到了 δ_{ij} 的值，为可信性替代性度，当模块间的可替代性减少时，软件可信度会增加。

```
def delta_ij(rho):
    """
    计算模块间的可信性替代性度 _ij
    :param rho: 参数 rho 值
    :return: _ij 的值
    """
    if rho == 1:  # 避免除零错误
        raise ValueError("rho = 1 时，delta_ij 无法计算（除零错误）")
    return 1 / (1 - rho)
```

delta.py

制作成表格如下所示：

```
D:\Python\python.exe F:\Project\Python\SoftwareQuality\Main8.py
rho     T_na(S)         delta_ij
---------------------------------
0.800   1.252422        5.00
0.600   1.477098        2.50
0.400   1.713976        1.67
0.200   1.909806        1.25
0.100   1.972840        1.11
0.010   2.002815        1.01
0.001   2.004322        1.00
```

图 2: δ_{ij} 值

将结果制作成 LaTeX 表格如下所示：

ρ	T_{na}	δ_{ij}
0.800	1.252422	5.00
0.600	1.477098	2.50
0.400	1.713976	1.67
0.200	1.909806	1.25
0.100	1.972840	1.11
0.010	2.002815	1.01
0.001	2.004322	1.00

表 3: 不同 ρ 值对应的 T_{na} 和 δ_{ij} 计算结果

三 附录

1 完整可运行代码

```python
def delta_ij(rho):
    if rho == 1:  # 避免除零错误
        raise ValueError("rho = 1 时，delta_ij 无法计算（除零错误）")
    return 1 / (1 - rho)

def mu_na(kappa, rho):
    return kappa ** rho

def measure(kappa, rho):
    if kappa < 0.1:
        return 1
    else:
        return 10 * kappa ** rho

def T_na(modules, rho):
    total = 0
    for beta, kappa in modules:
        mu_value = measure(kappa, rho)  # 使用 measure 计算 (m)
        total += beta * (mu_value ** rho)
    return total ** (1 / rho)

modules = [
    # (beta, kappa) for EMi
    (0.085, 0.053),
    (0.208, 0.132),
    (0.085, 0.026),
    (0.047, 0.000),
    (0.066, 0.000),
    (0.094, 0.105),
    (0.160, 0.053),
    (0.255, 0.026)
]

rho_values = [0.8, 0.6, 0.4, 0.2, 0.1, 0.01, 0.001]

print(f"{'rho':<8}{'T_na(S)':<15}{'delta_ij':<10}")
print("-" * 35)

for rho in rho_values:
    T_na_value = T_na(modules, rho)
    delta_value = delta_ij(rho)
    print(f"{rho:<8.3f}{T_na_value:<15.6f}{delta_value:<10.2f}")
```

main.py