***Github Repository:*** <https://github.com/PashaBarahimi/Software-Testing-Course-Projects>

***Last Commit:*** <hash>

# سوال اول

## Clause-ها

در این predicate، سه clause داریم که a و b و c هستند.

## جدول درستی

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | a | b | c | P |
| 1 | T | T | T | F |
| 2 | T | T | F | T |
| 3 | T | F | T | T |
| 4 | T | F | F | T |
| 5 | F | T | T | F |
| 6 | F | T | F | T |
| 7 | F | F | T | F |
| 8 | F | F | F | T |

محاسبه شرایط major clause بودن سه clause مسئله:

یادآوری:

شرایط a:

شرایط b:

شرایط c:

## GACC (General Active CC)

در GACC، همه جفت‌هایی که مقدار major clause متفاوت دارند را حساب می‌کنیم.

حالت major clause بودن a:

-> rows = [3, 7]

TR = {(3,7)}

حالت major clause بودن b:

-> rows = [1, 3]

TR = {(1,3)}

حالت major clause بودن c:

-> rows = [1, 2, 5, 6, 7, 8]

TR = {(1,2), (1,6), (1,8), (5,2), (5,6), (5,8), (7,2), (7,6), (7,8)}

## RACC (Restricted Active CC)

RACC زیرمجموعه CACC (Correlated Active CC)، و CACC زیرمجموعه GACC می‌باشد.

CACC، جفت‌هایی از GACC است که مقدار predicate متفاوتی دارند.

در RACC، همه minor clause-های جفت‌ها باید مقدار یکسانی داشته باشند.

حالت major clause بودن a:

TR = {(3,7)}

حالت major clause بودن b:

TR = {(1,3)}

حالت major clause بودن c:

TR = {(1,2), (5,6), (7,8)}

## RICC (Restricted Inactive CC)

در ابتدا GICC (General Inactive CC) را محاسبه می‌کنیم.

RICC جفت‌هایی از GICC است که minor clause-های آن مقدار یکسانی دارند.

برای محاسبه GICC، بین سطرها جفت‌هایی را انتخاب می‌کنیم که major clause متفاوت است و predicate یکسانی دارند.

حالت major clause بودن a:

-> rows = [1, 2, 4, 5, 6, 8]

TR = {(1,5), (2,6), (2,8), (4,6), (4,8)}

حالت major clause بودن b:

-> rows = [2, 4, 5, 6, 7, 8]

TR = {(2,4), (2,8), (4,6), (5,7), (6,8)}

حالت major clause بودن c:

-> rows = [3, 4]

TR = {(3,4)}

محاسبه RICC:

حالت major clause بودن a:

TR = {(1,5), (2,6), (4,8)}

حالت major clause بودن b:

TR = {(2,4), (5,7), (6,8)}

حالت major clause بودن c:

TR = {(3,4)}

# سوال دوم

کد به صورت زیر است:

def calculate\_total\_cost(quantity, unit\_price, discount):  
 if quantity <= 0 or unit\_price <= 0:  
 return “Invalid input”  
 else:  
 total\_cost = quantity \* unit\_price \* (1 - discount)  
 return total\_cost

در این کد، 3 پارامتر زیر را داریم:

## quantity

برای این پارامتر می‌توانیم چهار گروه (block) زیر را در نظر بگیریم:

* A0: کوچک‌تر از 0 (مقدار غیر قابل قبول)
* A1: برابر با 0 (مقدار غیر قابل قبول)
* A2: برابر با 1 (مقدار قابل قبول)
* A3: بیشتر از 1 (مقدار قابل قبول)

## unit\_price

برای این پارامتر سه گروه (block) زیر را در نظر می‌گیریم:

* B0: کوچک‌تر از 0 (مقدار غیر قابل قبول)
* B1: برابر با 0 (مقدار غیر قابل قبول)
* B2: بیشتر از 0 (مقدار قابل قبول)

## discount

برای این پارامتر نیز پنج گروه (block) زیر را در نظر می‌گیریم:

* C0: کوچک‌تر از 0 (مقدار غیر قابل قبول)
* C1: برابر با 0 (مقدار قابل قبول)
* C2: بین 0 و 1 (مقدار قابل قبول)
* C3: برابر با 1 (مقدار قابل قبول)
* C4: بیشتر از 1 (مقدار غیر قابل قبول)

با توجه به اینکه روش PWC انتخاب شده، می‌توانیم تست‌های زیر را انتخاب کنیم:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A0, B0, C0 | A0, B2, C1 | A0, B1, C2 | A0, B0, C3 | A0, B0, C4 |
| A1, B1, C0 | A1, B0, C1 | A1, B2, C2 | A1, B1, C3 | A1, B1, C4 |
| A2, B2, C0 | A2, B1, C1 | A2, B0, C2 | A2, B2, C3 | A2, B2, C4 |
| A3, B0, C0 | A3, B2, C1 | A3, B2, C2 | A3, B2, C3 | A3, B1, C4 |

تست‌کیس‌ها به صورت زیر خواهند بود:

import unittest  
  
from main import calculate\_total\_cost  
  
INVALID\_INPUT = "Invalid input"  
  
A = [-1, 0, 1, 2]  
B = [-1, 0, 1]  
C = [-1, 0, 0.5, 1, 2]  
  
  
class TestCalculateTotalCost(unittest.TestCase):  
 def test\_invalid\_inputs\_should\_fail(self):  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[0], B[0], C[0]))  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[0], B[2], C[1]))  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[0], B[1], C[2]))  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[0], B[0], C[3]))  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[0], B[0], C[4]))  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[1], B[1], C[0]))  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[1], B[0], C[1]))  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[1], B[2], C[2]))  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[1], B[1], C[3]))  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[1], B[1], C[4]))  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[2], B[1], C[1]))  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[2], B[0], C[2]))  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[3], B[0], C[0]))  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[3], B[1], C[4]))  
  
 def test\_invalid\_discount\_should\_fail(self): *# not covered in the code*  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[2], B[2], C[0]))  
 self.assertEqual(INVALID\_INPUT, calculate\_total\_cost(A[2], B[2], C[4]))  
  
 def test\_valid\_inputs\_should\_pass(self):  
 self.assertEqual(0, calculate\_total\_cost(A[2], B[2], C[3]))  
 self.assertEqual(2, calculate\_total\_cost(A[3], B[2], C[1]))  
 self.assertEqual(1, calculate\_total\_cost(A[3], B[2], C[2]))  
 self.assertEqual(0, calculate\_total\_cost(A[3], B[2], C[3]))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

همانطور که در کامنت گفته شده، در بازه صحیح بودن تخفیف در کد چک نشده و باگ دارد. به همین دلیل این 2 تست پاس نمی‌شوند:

