دانشگاه صنعتی اصفهان

دانشکده مهندسی برق و کامپیوتر

عنوان: تکلیف دوم درس آزمون نرم افزار

نام و نام خانوادگی: علیرضا ابره فروش

شماره دانشجویی: ۹۸۱۶۶۰۳

نیم سال تحصیلی: بهار ۱۴۰۱/۱۴۰۲

مدرّس: دکتر الهام محمودزاده

۱

۱.۱ سوال ۶ فصل هفتم صفحه‌ی ۱۷۸

۱.۱.۱ a

- **Node coverage**:

$(1), (2), (3), (4), (5), (6), (7), (8), (9), (10)$

- **Edge coverage**:

$(1,4), (1,5), (2,5), (6,2), (3,6), (3,7), (4,8), (5,8), (5,9), (9,6), (6,10), (7,10)$

- **Prime path coverage**:

$(1,4,8), (1,5,8), (3,6,10), (3,7,10), (1,5,9,6,2), (1,5,9,6,10), (2,5,9,6,10), (2,5,9,6,2), (3,6,2,5,8), (3,6,2,5,9), (5,9,6,2,5), (6,2,5,9,6), (9,6,2,5,8), (9,6,2,5,9)$

۲.۱.۱ b

$(1,4,8), (2,5,9), (3,6,10), (3,7,10)$

۳.۱.۱ c

$(1,4,8), (3,6,10), (3,7,10), (1,5,9,6,2,5,8)$

۲.۱ سوال ۷ فصل هفتم صفحه‌ی ۱۷۹

۱.۲.۱ a

p2 و p3 مسیر تست هستند. اما از آنجایی که p1 به نود پایانی ختم نشده است نمی‌تواند مسیر تست باشد. همچنین p4 نیز مسیر تست نیست چون با نود شروع آغاز نشده است. و در نهایت p5 نیز مسیر تست نیست چون یال $(3,2)$ وجود ندارد.

۲.۲.۱ b

- **Edge-pair coverage**:

$(1,2,1), (1,2,3), (1,3,1), (2,1,2), (2,1,3), (2,3,1), (3,1,2), (3,1,3)$

۳.۲.۱ c

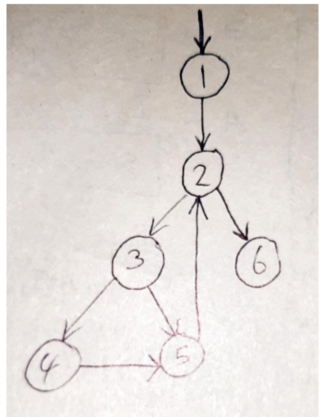
هیچ یک از مسیرهای p2 و p3 از جفت یال‌های $(3,1,3)$ و $(2,1,2)$ عبور نمی‌کند. سایر موارد کاندید هم مسیر تست نیستند.

۴.۲.۱ d

p3 مستقیما از این prime path عبور نمی‌کند. اما p3 از prime path با sidetripِ $(1,2,1)$ عبور می‌کند.

۲ سوال اول صفحه‌ی ۱۸۷

۱.۲ a



شکل ۱

۲.۲ b

$(1, 2, 6), (3, 5, 2, 6), (1, 2, 3), (3, 4, 5, 2, 3), (3, 5, 2, 3), (3, 4, 5, 2, 6)$

۳.۲ c

t1:	$(1, 2, 6)$
t2:	$(3, 5, 2, 6), (3, 4, 5, 2, 3), (1, 2, 3)$
t3:	$(3, 5, 2, 3), (3, 4, 5, 2, 6), (1, 2, 3)$
t4:	$(3, 5, 2, 6), (1, 2, 3)$

۴.۲ d

def(1) توسط همه‌ی مسیرهای تست به کار برده می‌شود و def(3) توسط مسیرهای $\{t2\}$ و $\{t3\}$ و $\{t4\}$ به کار برده می‌شود پس هر یک از مجموعه‌های $\{t2\}$ و $\{t3\}$ و $\{t4\}$ می‌تواند یک مجموعه‌ی مینیمال باشد که همه‌ی defها را پوشش می‌دهد.

۵.۲ e

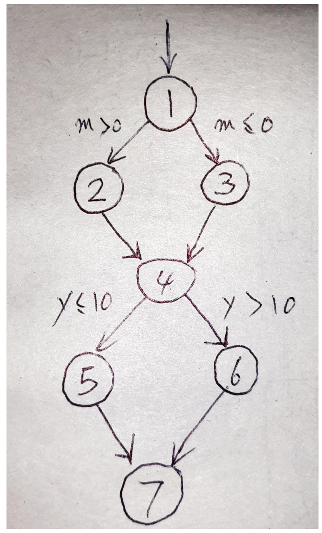
$\{t1, t2\}$ و $\{t1, t3\}$

۶.۲ f

$\{t1, t2, t3\}$

۳ سوال اول صفحات ۲۰۲ و ۲۰۳

۱.۳ a



شکل ۲

۲.۳ b

۱ و ۲ و ۳

۳.۳ c

۲ و ۳ و ۷

۴.۳ d

خیر-از آنجایی که در نودهای ۲ و ۳ متغیر def w شده است و نتیجتا از نود ۱ تا نود ۷ مسیر def-clear نداریم. پس du-path نیز نداریم.

۵.۳ e

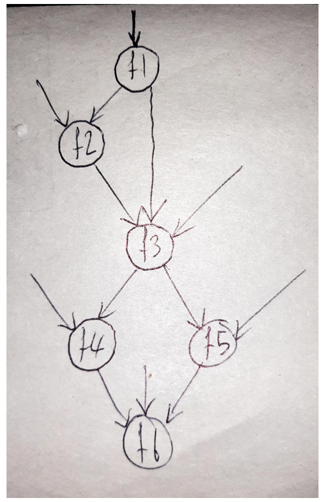
$w:\quad (2,4,5,7),(3,4,5,7),(2,4,6,7),(3,4,6,7)$

$x:\quad (5,7),(6,7)$

۴ سوال‌های ۳ و ۴ صفحات ۲۱۸ و ۲۱۹

۱.۴ سوال ۳

۱.۱.۴ a



شکل ۳

۲.۱.۴ b

$t1: \quad [f1, f3, f5, f6]$

$t2: \quad [f1, f3, f4, f6]$

$t3: \quad [f1, f2]$

$t4: \quad [f1, f3, f4, f6]$

$t5: \quad [f1, f2, f3, f4, f6]$

۳.۱.۴ c

$\{t1, t5\}$

۴.۱.۴ d

$\{t1, t5\}$

۵.۱.۴ e

$\{[f1, f3, f4, f6], [f1, f3, f5, f6], [f1, f2, f3, f4, f6], [f1, f2, f3, f5, f6]\}$

هیچ یک از مسیرهای تست $[f1, f2, f3, f5, f6]$ را پوشش نمی‌دهند.

۲.۴ سوال ۴

۱.۲.۴ a

- Line 12: Calling `takeOut()` method in the `trash()` method.
- Line 20: Calling `takeOut()` method in the `takeOut()` method.

خط ۱۲ و خط ۱۳

۲.۲.۴ b

$last-defs: \quad (x,1),(n,9),(n,11),(m,5),(m,7),(e,21),(e,23),(d,19),(a,15),(b,15)$

$first-uses: \quad (o,13),(a,19),(b,21),(b,23),(x,6),(m,9),(m,11),(n,12),(e,24),(d,21),(d,23)$

$allpairs: \quad (trash(),n,9),(trash(),n,9),(takeOut(),b,21),(trash(),m,5),(takeOut(),a,19)(trash(),n,11),(takeOut(),b,21),(takeOut(),b,23),(trash(),m,7),(takeOut(),a,19),(takeOut(),e,21),(trash(),o,13),(trash(),n,11),(takeOut(),b,23),(takeOut(),e,23),(trash(),o,13)$ For the `trash()` method:

- Last-def: Line 7 (`m = 4`)
- First-use: Line 9 (`n = 3m`)
- Last-def: Line 11 (`n = 4m`)
- First-use: Line 12 (`takeOut(m, n)`)

For the `takeOut()` method:

- Last-def: Line 19 (`d = 42a`)
- First-use: Line 21 (`e = 2b+d`)
- Last-def: Line 23 (`e = b+d`)
- First-use: Line 24 (`return e`)

۳.۲.۴ c

- Input: $x = 10$
- Explanation: The condition $x > 0$ is true, so $m = 4$. Then the nested condition $x > 5$ is also true, so $n = 3m = 12$. The call to `takeOut(m, n)` will pass $m = 4$ and $n = 12$ as arguments. The expected output will depend on the implementation of the `takeOut()` method.

۵ سوال اول صفحات ۲۳۳ و ۲۳۴

5.1 a

The variable ”elements” in the `BoundedQueue2` class represents the elements present in the queue. Since we do not care about the specific objects, there are four useful values for this variable:

1. `null`: Represents an empty position in the queue.
2. `obj`: Represents a non-null object stored in the queue.
3. `obj1`: Represents a non-null object different from `obj` stored in the queue.
4. `obj2`: Represents another non-null object different from `obj` and `obj1` stored in the queue.

5.2 b

To determine the number of states, we need to consider all possible combinations of the representation variables `[elements, size, front, back]`. Let’s analyze each variable:

- `elements`: There are 4 useful values for this variable (as discussed in part (a)).
- `size`: The size can range from 0 to nn (the maximum capacity of the queue).
- `front`: The front can range from 0 to $n1$.
- `back`: The back can range from 0 to $n1$.

Considering these ranges, the total number of states is $(4(n+1)nn)$, where nn is the maximum capacity of the queue.

5.3 c

The reachable states are those that can be reached through valid method calls and operations on the `BoundedQueue2` object. To determine the number of reachable states, we need to identify the valid transitions between states based on the methods `enQueue()` and `deQueue()`.

5.4 d

Since the exact maximum capacity (n) of the queue is not specified, I cannot provide a specific drawing of the reachable states without this information.

5.5 e

Adding edges for the `enQueue()` and `deQueue()` methods:

- `enQueue()`: This method adds an element to the back of the queue if it is not full. It updates the `elements`, `size`, and `back` variables accordingly.
  1. If the queue is not full:
     - Transition: `[elements, size, front, back]` →→ `[updated elements, updated size, front, updated back]`
  2. If the queue is full (Exceptional Return):
     - Transition: `[elements, size, front, back]` → `[elements, size, front, back]`
- `deQueue()`: This method removes the element at the front of the queue if it is not empty. It updates the `elements`, `size`, and `front` variables accordingly.
  1. If the queue is not empty:
     - Transition: `[elements, size, front, back]` →→ `[updated elements, updated size, updated front, back]`
  2. If the queue is empty (Exceptional Return):
     - Transition: `[elements, size, front, back]` → `[elements, size, front, back]`

5.6 f

To achieve Edge Coverage, we need to design a test set that covers all possible edges or transitions between states. Since the specific maximum capacity (n) of the queue is not provided, it's challenging to provide a complete test set without this information. However, here's an example of a small test set:

- Create an empty queue with a capacity of 3.
- Call enQueue(obj1).
- Call enQueue(obj2).
- Call deQueue().
- Call enQueue(obj3).

This test set covers the following edges:

- Initial state to state with one element: [[null, null], 0, 0, 0] -> [[obj1, null], 1, 0, 1]
- State with one element to state with two elements: [[obj1, null], 1, 0, 1] -> [[obj1, obj2], 2, 0, 2]

- State with two elements to state with one element: [[obj1, obj2], 2, 0, 2] -> [[null, obj2], 1, 1, 2]
- State with one element to state with two elements again: [[null, obj2], 1, 1, 2] -> [[obj3, obj2], 2, 1, 0]