تکلیف اول

بخش اول

تمرین ۴

a) بله. زیرا یک مجموعه فقط میتواند تهی باشد و یا تعدادی بزرگتر مساوی یک عضو داشته باشد و یا null باشد. پس ویژگی completeness را satisfy میکند.

b) بله. هیچ مقداری وجود ندارد که در بیش از یک بلاک قرار بگیرد. پس ویژگی disjointness را نیز satisfy میکند.

c) خیر. زیرا $s1$ و $s2$ میتوانند اشتراک داشته باشند ولی هیچ یک زیرمجوعه‌ی دیگری نباشد. بعنوان مثال اگر داشته باشیم $s1 = \{1, 2, 3\}$ و $s2 = \{3, 4, 5\}$ آنگاه هیچ کدام از بلاک ها شامل این مثال نمیشوند پس ویژگی completeness در این قسمت برقرار نمیباشد.

d) خیر. زیرا اگر $s1$ و $s2$ دو مجموعه‌ی یکسان باشند، آنگاه داخل هر سه بلاک اول قرار میگیرند. زیرا هر دو مجموعه‌ی یکسان، زیرمجموعه‌ی یکدیگر نیز میباشند. بعنوان مثال اگر داشته باشیم $s1 = \{1, 2, 3\}$ و $s2 = \{1, 2, 3\}$ آنگاه هر سه بلاک نخست شامل این حالت میشوند در نتیجه ویژگی disjointness نیز برقرار نمیباشد.

e) اگر پارتیشن‌های characteristic اول را با $a1$ و $a2$ و $a3$ نامگذاری کنیم و همچنین پارتیشن‌های characteristic دوم را با $b1$ و $b2$ و $b3$ و $b4$ نامگذاری کنیم، آنگاه بر اساس case base، criterion تعداد requirement test ها برابر میشود با $1 + (4 - 1) + (3 - 1) = 6$. بعبارتی اگر مثلا case base ها را $a1$ و $b1$ در نظر بگیریم، requirement test ها به این صورت میشوند: $\{a1, b1\}, \{a1, b2\}, \{a1, b3\}, \{a1, b4\}, \{a2, b1\}, \{a3, b1\}$

f) میتوانیم characteristic دوم را به این صورت پارتیشن‌بندی کنیم تا دو ویژگی completeness و disjointness را داشته باشد:

- $s1$ و $s2$ برابر باشند.
- $s1$ و $s2$ هیچ اشتراکی با هم نداشته باشند.
- $s1$ و $s2$ در یک یا بیشتر از یک عضو اشتراک داشته باشند ولی برابر نباشند.

تمرین ۶

a) capacity(C)، size(unnecessary)، front(unnecessary)، elements(E)، X(input element)

b ، c ، d)

- validity of elements
  - $a1$) null $\{null\}$
  - $a2$) empty list $\{[]\}$
  - $a3$) non-empty and non-full list $\{E = [”a”], C = 2\}$
  - $a4$) full list $\{\{E = [”a”, ”b”], C = 2\}\}$
  - $a5$) list has more elements than capacity $\{\{E = [”a”, ”b”, ”c”], C = 2\}\}$

- validity of X
  - $b1$) null $\{null\}$
  - $b2$) has value $\{"y"\}$
- capacity
  - $c1$) negative $\{-1\}$
  - $c2$) 0 $\{0\}$
  - $c3$) 1 $\{1\}$
  - $c4$) more than 1 $\{2\}$
- validity of front (unnecessary)
  - $d1$) null $\{null\}$
  - $d2$) has value $\{"a"\}$
- size (unnecessary)
  - $e1$) negative $\{-1\}$
  - $e2$) 0 $\{0\}$
  - $e3$) 1 $\{1\}$
  - $e4$) more than 1 $\{2\}$

e) طبق معیار BCC ، test requirement ها به این صورت هستند(چون S بر اساس تعداد اعضای elements تعیین میشود، بنابراین نیازی به آوردن مقدار آن در بین تست ها نیست و میتوان آنرا نادیده گرفت. با همین استدلال میتوان از در نظر گرفتن front نیز چشم‌پوشی کرد. همچینین base case ها با رنگ قرمز مشخص شده اند):

- $\{a3, b2, c4\} \rightarrow \{C = 2, E = ["a"], X = "y"\}$
  - assert(isFull() == False)
  - assert(isEmpty() == False)
  - assert(deQueue() == "a")
- $\{a1, b2, c4\} \rightarrow \{C = 2, E = null, X = "y"\}$
  - assert(isFull() == NullPointerException)
  - assert(isEmpty() == NullPointerException)
  - assert(deQueue() == NullPointerException)
  - assert(enQueue("y") == NullPointerException)
- $\{a2, b2, c4\} \rightarrow \{C = 2, E = [], X = "y"\}$
  - assert(isFull() == False)
  - assert(isEmpty() == True)
  - assert(deQueue() == IllegalStateException)

- $\{a4, b2, c4\} \rightarrow \{C = 2, E = [\text{"}a\text{"}, \text{"}b\text{"}], X = \text{"}y\text{"}\}$
  - assert(isFull() == True)
  - assert(isEmpty() == False)
  - assert(deQueue() == "a")
  - assert(enQueue("y") == IllegalStateException)
- $\{a5, b2, d4\} \rightarrow \{C = 2, E = [\text{"}a\text{"}, \text{"}b\text{"}, \text{"}c\text{"}], X = \text{"}y\text{"}\}$ (invalid)
- $\{a3, b1, d4\} \rightarrow \{C = 2, E = [\text{"}a\text{"}], X = null\}$
  - assert(enQueue(null) == NullPointerException)
- $\{a3, b2, c1\}$ ($a3$ is incompatible with $c1$)
- $\{a3, b2, c2\}$ ($a3$ is incompatible with $c2$)
- $\{a3, b2, c3\}$ ($a3$ is incompatible with $c3$)

میتوانستیم تعدادی requirement test دیگر که در BCC پوشش داده نشده بودند را هم اضافه کنیم تا بتوانیم چیزهای بیشتری را تست کنیم:

- $\{a2, b2, c2\} \rightarrow \{C = 0, E = [], X = \text{"}y\text{"}\}$
  - assert(isFull() == True)
  - assert(isEmpty() == True)
  - assert(deQueue() == NullPointerException)
  - assert(enQueue("y") == IllegalStateException)
- $\{a2, b2, c1\} \rightarrow \{C = -1, E = [], X = \text{"}y\text{"}\}$
  - assert(boundedQueue(-1) == IllegalArgumentException)

بخش دوم

متغیرهای این بخش را میتوان به این صورت تعریف کرد: id(I)، name(N)، phone(P)، contacts(C)
که در آن contacts لیستی از مخاطبین به صورت $(id, name, phone)$ میباشد.

پرسش اول

هر نمونه تست باید شامل یک لیستی از مخاطبین به صورت $(id, name, phone)$ باشد و همچنین دو رشته برای phone و name باشد. رشته phone باید تنها از رقم ها ایجاد شده باشد و رشته ی name باید از حروف ایجاد شده باشد. لیست مخاطبین برای تست کردن محدودیت unique بودن phone و name برای این عمل مورد نیاز است. در صورتی که محدودیت های دیگر روی phone (مثل طول رشته و شروع با "09") را میتوان به صورت مستقل بر اساس مقدار phone آزمود.

پرسش دوم

a ، b)

- length of phone
  - $a1$) 0 {""}
  - $a2$) 1 {"0"}
  - $a3$) more than 1 and less than 11 {"0912"}
  - $a4$) 11 {"09123456789","01122334455"}
  - $a5$) more than 11 {"091122334455"}
- first two characters of phone (for phones with length more than 1)
  - $b1$) "09" {"09123456789"}
  - $b2$) other {"01122334455"}
- number of contacts
  - $c1$) 0 {[]}
  - $c2$) more than 0 $\{C = [\text{"}ali\text{"}, \text{"09987654321"}]\}$
- relation between contacts and phone
  - $d1$) contacts contains phone $\{C = [\{\text{"}ali\text{"}, \text{"09123456789"}\}], P = \text{"09123456789"}\}$
  - $d2$) contacts doesn't contains phone $\{C = [\{\text{"}ali\text{"}, \text{"09987654321"}\}], P = \text{"09123456789"}\}$
- relation between contacts and name
  - $e1$) contacts contains name $\{C = [\{\text{"}ali\text{"}, \text{"09987654321"}\}], N = \text{"}ali\text{"}\}$
  - $e2$) contacts doesn't contains name $\{C = [\{\text{"}ali\text{"}, \text{"09987654321"}\}], N = \text{"}reza\text{"}\}$

c) ابتدا test requirement های هریک از روش‌ها را مینویسیم و سپس تحلیل میکنیم (جفت‌های $\{a1, b1\}$، $\{a1, b2\}$، $\{a2, b1\}$، $\{a2, b2\}$ و همچنین جفت‌های $\{c1, d1\}$، $\{c1, e1\}$ با همدیگه سازگار نیستند. در نتیجه test requirement هایی که شامل هر یک از این جفت های ناسازگار میشوند نیز incompatible هستند. test requirement هایی که incompatible هستند را با رنگ قرمز مشخص کرده ایم):

- ECC = $\{a1, b1, c1, d1, e1\}, \{a2, b2, c2, d2, e2\}, \{a3, b1, c1, d2, e1\}, \{a4, b2, c2, d1, e2\}, \{a5, b1, c1, d1, e1\}$
- BCC = $\{a4, b1, c2, d2, e2\}, \{a1, b1, c2, d2, e2\}, \{a2, b1, c2, d2, e2\}, \{a3, b1, c2, d2, e2\}, \{a5, b1, c2, d2, e2\}$
  $\{a4, b2, c2, d2, e2\}, \{a4, b1, c1, d2, e2\}, \{a4, b1, c2, d1, e2\}, \{a4, b1, c2, d2, e1\}$
- PWC = $\{a1, b1, c1, d1, e1\}, \{a1, b2, c2, d2, e2\}, \{a2, b1, c1, d2, e1\}, \{a2, b2, c2, d1, e2\}, \{a3, b1, c2, d1, e2\},$
  $\{a3, b2, c1, d2, e1\}, \{a4, b1, c2, d2, e1\}, \{a4, b2, c1, d1, e2\}, \{a5, b1, c1, d2, e2\}, \{a5, b2, c2, d1, e1\}$

در روش ECC تعداد test requirement ها خیلی کم هستند و از همان تعداد کم هم، تعداد زیادی incompatible هستند. پس این روش مناسب نمیباشد. زیرا حالت های خیلی کمی را test میکند.
در روش PWC تعداد test requirement ها زیاد هستند، ولی بسیاری از آن‌ها incompatible هستند و نیازی نیست در نظر گرفته شوند. همچنین در این مساله ی خاص، تنها دو characteristic اول و سه characteristic آخر با یکدیگر ارتباط دارند و از یکدیگر مستقل هستند و مستقلا میتوانند تست شوند (به عنوان مثال characteristic اول با

characteristic چهارم، هیچ ارتباطی ندارند و مستقلا باید تست شوند). برای همین نیازی نیست جفت بلاک های بین دو characteristic اول و سه characteristic آخر همگی در نظر گرفته شوند. برای همین این روش نیز در این مساله به نظر مناسب نمیباشد.

در روش BCC از هر characteristic یک بلاک مهم و پایه در نظر گرفته میشود و کافیست با ثابت نگه داشتن سایر بلاک‌ها، در هر مرحله تغییرات یک characteristic را در نظر بگیرید که با توجه به مساله، به نظر این روش مناسب میباشد زیرا هم تعداد test requirement ها و همچنین حالت‌های incompatible خیلی زیاد نمیباشد و اکثر حالت های ممکن مساله را که باید تست شوند را cover میکند. در واقع جفت‌هایی که incompatible هستند با احتمال خیلی کمی در بین test requirement ها قرار میگیرند. زیرا حالت‌های پایه و مهم خیلی تست های incompatible را ایجاد نمیکنند.

d) برای هر کدام از test requirement های روش BCC که incompatible نیستند، تست هایی تعریف میکنیم تا همه‌ی آن‌ها پوشش داده شوند.

- $\{a4, b1, c2, d2, e2\} \rightarrow \{P = \text{"09123456789"}, N = \text{"}reza\text{"}, C = [\{\text{"}ali\text{"}, \text{"09987654321"}\}]\}$
  - assert(add("reza"، "09123456789") == True)
- $\{a3, b1, c2, d2, e2\} \rightarrow \{P = \text{"09123"}, N = \text{"}reza\text{"}, C = [\{\text{"}ali\text{"}, \text{"09987654321"}\}]\}$
  - assert(add("reza"، "09123") == False)
- $\{a5, b1, c2, d2, e2\} \rightarrow \{P = \text{"091122334455"}, N = \text{"}reza\text{"}, C = [\{\text{"}ali\text{"}, \text{"09987654321"}\}]\}$
  - assert(add("reza"، "091122334455") == False)
- $\{a4, b2, c2, d2, e2\} \rightarrow \{P = \text{"01122334455"}, N = \text{"}reza\text{"}, C = [\{\text{"}ali\text{"}, \text{"09987654321"}\}]\}$
  - assert(add("reza"، "01122334455") == False)
- $\{a4, b1, c1, d2, e2\} \rightarrow \{P = \text{"09123456789"}, N = \text{"}reza\text{"}, C = []\}$
  - assert(add("reza"، "09123456789") == True)
- $\{a4, b1, c2, d1, e2\} \rightarrow \{P = \text{"09123456789"}, N = \text{"}reza\text{"}, C = [\{\text{"}ali\text{"}, \text{"09123456789"}\}]\}$
  - assert(add("reza"، "09123456789") == False)
- $\{a4, b1, c2, d2, e1\} \rightarrow \{P = \text{"09123456789"}, N = \text{"}ali\text{"}, C = [\{\text{"}ali\text{"}, \text{"09987654321"}\}]\}$
  - assert(add("ali"، "09123456789") == False)

پرسش سوم

برای تابع find مراحل پرسش قبل را پاسخ میدهیم (در این قسمت فرض کرده‌ایم اگر str خالی باشد، لیست خالی برمیگردد. همچنین برای راحتی نوشتار، شماره تلفن ها را یازده رقمی در نظر نگرفته ایم.):

- value of str
  - $a1$) only include digits {"12"}
  - $a2$) only include characters {"al"}
  - $a3$) include digits and characters {"$r2$"}
- length of str
  - $b1$) 0 {""}
  - $b2$) more than 0 {"12"}
- length of contacts:
  - $c1$) 0 {$C = []$}
  - $c2$) 1 {$C = [\{"ali","091122"\}]$}
  - $c3$) more than 1 {$C = [\{"12","ali","091122"\},\{"13","alireza","091234"\}]$}
- number of contacts that str is substring of their phone
  - $d1$) 0 {$str = "12", C = [\{"12","ali","095566"\},\{"13","alireza","095678"\}]$}
  - $d2$) 1 {$str = "12", C = [\{"12","ali","091122"\},\{"13","alireza","095678"\}]$}
  - $d3$) more than 1 {$str = "12", C = [\{"12","ali","091122"\},\{"13","alireza","091234"\}]$}
- number of contacts that str is substring of their name
  - $e1$) 0 {$str = "12", C = [\{"12","hamid","091122"\},\{"13","reza","091234"\}]$}
  - $e2$) 1 {$str = "al", C = [\{"12","ali","091122"\},\{"13","reza","091234"\}]$}
  - $e3$) more than 1 {$str = "al", C = [\{"12","ali","091122"\},\{"13","alireza","091234"\}]$}
- number of contacts that str is substring of their id
  - $f1$) 0 {$str = "12", C = [\{"34","ali","091122"\},\{"45","alireza","091234"\}]$}
  - $f2$) more than 0 {$str = "12", C = [\{"12","ali","091122"\},\{"13","alireza","091234"\}]$}

حال بر اساس معیار BCC تست هایی تولید میکنیم تا تمامی test requirement هارا پوشش دهد:

- $\{a1, b2, c3, d3, e1, f2\} \rightarrow \{str = "12", C = [\{"12","ali","091122"\},\{"13","alireza","091234"\}]\}$
  - assert(find("12") == $[\{"12","ali","091122"\},\{"13","alireza","091234"\}]$)
- $\{a2, b2, c3, d3, e1, f2\} \rightarrow \{str = "al", C = [\{"12","ali","091122"\},\{"13","alireza","091234"\}]\}$
  - assert(find("al") == $[\{"12","ali","091122"\},\{"13","alireza","091234"\}]$)
- $\{a3, b2, c3, d3, e1, f2\} \rightarrow \{str = "r2", C = [\{"12","ali","091122"\},\{"13","alireza","091234"\}]\}$
  - assert(find("$r2$") == [])

- $\{a1, b1, c3, d3, e1, f2\}$ (incompatible)
- $\{a1, b2, c1, d3, e1, f2\} \rightarrow \{str = "12", C = []\}$
  - assert(find("12") == [])
- $\{a1, b2, c2, d3, e1, f2\} \rightarrow \{str = "12", C = [\{"12", "ali", "091122"\}]\}$
  - assert(find("12") == [{"12", "ali", "091122"}])
- $\{a1, b2, c3, d1, e1, f2\} \rightarrow \{str = "12", C = [\{"12", "ali", "095566"\}, \{"13", "alireza", "095678"\}]\}$
  - assert(find("12") == [])
- $\{a1, b2, c3, d2, e1, f2\} \rightarrow \{str = "12", C = [\{"12", "ali", "091122"\}, \{"13", "alireza", "095678"\}]\}$
  - assert(find("12") == [{"12", "ali", "091122"}])
- $\{a1, b2, c3, d3, e2, f2\}$ (incompatible)
- $\{a1, b2, c3, d3, e3, f2\}$ (incompatible)
- $\{a1, b2, c3, d3, e1, f1\} \rightarrow \{str = "34", C = [\{"12", "ali", "091122"\}, \{"45", "alireza", "091234"\}]\}$
  - assert(find("12") == [{"34", "ali", "091122"}, {"45", "alireza", "091234"}])

حال مراحل بالا را برای تابع remove مجددا تکرار میکنیم:

- value of id
  - $a1$) negative $\{-1\}$
  - $a2$) non-negative $\{1\}$
- length of contacts
  - $b1$) 0 $\{[]\}$
  - $b2$) more than 0 $\{[\text{"1"}, \text{"}ali\text{"}, \text{"091234"}]\}$
- relation between contacts and input id
  - $c1$) there is no contact that input id is substring of its id. $\{id = \text{"1"}, C = [\{\text{"2"}, \text{"}ali\text{"}, \text{"091234"}\}]\}$
  - $c2$) there is no contact that input id is equal to its id but there is a contact that its id is substring of input id. $\{id = \text{"1"}, C = [\{\text{"12"}, \text{"}ali\text{"}, \text{"091234"}\}]\}$
  - $c3$) there is a contact that its id is equal to input id but the input id isn't substring of its phone. $\{id = \text{"1"}, C = [\{\text{"1"}, \text{"}ali\text{"}, \text{"095566"}\}]\}$
  - $c4$) there is a contact that its id is equal to input id and input id is substring of its phone. $\{id = \text{"1"}, C = [\{\text{"1"}, \text{"}ali\text{"}, \text{"091234"}\}]\}$

حال بر اساس معیار BCC تست هایی تولید میکنیم تا تمامی test requirement هارا پوشش دهد:

- $\{a2, b2, c4\} \rightarrow \{id = \text{"1"}, C = [\{\text{"1"}, \text{"}ali\text{"}, \text{"091234"}\}]\}$
  - assert(remove("1") == True)
- $\{a1, b2, c4\} \rightarrow \{id = \text{"} - 1\text{"}, C = [\{\text{"1"}, \text{"}ali\text{"}, \text{"091234"}\}]\}$
  - assert(remove("-1") == False)
- $\{a2, b1, c4\} \rightarrow \{id = \text{"1"}, C = []\}$
  - assert(remove("1") == False)
- $\{a2, b2, c1\} \rightarrow \{id = \text{"1"}, C = [\{\text{"2"}, \text{"}ali\text{"}, \text{"091234"}\}]\}$
  - assert(remove("1") == False)
- $\{a2, b2, c2\} \rightarrow \{id = \text{"1"}, C = [\{\text{"12"}, \text{"}ali\text{"}, \text{"091234"}\}]\}$
  - assert(remove("1") == False)
- $\{a2, b2, c3\} \rightarrow \{id = \text{"1"}, C = [\{\text{"1"}, \text{"}ali\text{"}, \text{"095566"}\}]\}$
  - assert(remove("1") == True)

پرسش چهارم

به طور شهودی بنظر میرسد که تست کردن یکجای این عمل ها کار مناسبی نمیباشد، زیرا طبق صحبت های اولیه درس، حتی الامکان بهتر است تست ها در مرحله test unit انجام شوند. چرا که سرعت پاسخ دهی و همچنین تعداد حالت هایی که باید بررسی و تست شوند و پیچیدگی آن‌ها کاهش میابد. حال اگر بخواهیم هر سه عمل را به صورت یکجا تست کنیم، باید characteristic هایی تعریف کنیم که هر سه عمل و محدودیت هایی که دارند را پوشش دهند. از آن‌جایی که هر سه عمل از یکدیگر مستقل انجام میشوند و هیچ ارتباطی بین دو به دوی آن‌ها وجود ندارد، و همچنین هیچ محدودیتی بین دو به دوی آن‌ها تعریف نشده است که باید مورد تست قرار گیرد، میتوان بدون تعریف کردن characteristic جدیدی، از همان characteristic هایی که برای هر یک از عمل ها در بخش های قبلی تعریف کردیم استفاده کرد. بعنوان مثال، برای تست کردن تابع find لزومی ندارد تست ما حتما شامل مخاطبینی باشد که شماره تلفن آن‌ها حتما 11 رقمی باشد و با 09 شروع شده باشد. میتوانیم از هر عدد دیگری برای تست کردن اینکه آیا str ورودی زیررشته ای در داخل لیست دارد و یا خیر استفاده کرد. یا بعنوان مثال دیگر، شرط unique بودن شماره و نام ها برای تابع find نیازی نیست. چرا که کافیست همه‌ی اعضایی که با str ورودی اشتراک دارند را برگرداند. چه unique باشند چه نباشند. به این ترتیب نیازی نیست که characteristic جدیدی تعریف کرد. اما اگر قرار باشد همه‌ی characteristic های قبلی را به صورت یکجا استفاده کنیم، این باعث میشود تعداد test requirement های غیرضروری که در واقع رابطه‌ی بین characteristic های اعمال متفاوت را تست میکند افزایش یابد. بعنوان مثال اگر از روش PWC استفاده کنیم، جفت هایی که بین characteristic های مختلف از دو عمل متفاوت قرار میگیرند(مثلا جفت length of phone در characteristic اول از عمل add و only include digits از characteristic اول از عمل find) هیچ ارتباطی با یکدیگر ندارند در نتیجه لزومی ندارد حتما در تست های ما cover شوند. پس بنظر میرسد تست کردن یکجای این اعمال، باعث افزایش test requirement های غیرضروری و غیر مرتبط و در نتیجه کاهش سرعت عمل بدون تاثیری در بهبود نتیجه خواهد داشت و بنابراین بنظر میرسد راه حل مناسبی برای این مساله نمیباشد.