|  |
| --- |
|  |
| Discrete Structures and Algorithms |
| Project Assignment |

|  |
| --- |
| By: Jake McDonnell and Joshua Wilkinson  1-27-2022 |

**Table of Contents**

[**Introduction** 2](#_Toc95981771)

[**Description of the Problem** 3](#_Toc95981772)

[**Pseudocode** 4](#_Toc95981773)

[**Data Structures** 8](#_Toc95981774)

[**Functions** 10](#_Toc95981775)

[**Extra Methods/Operations** 11](#_Toc95981776)

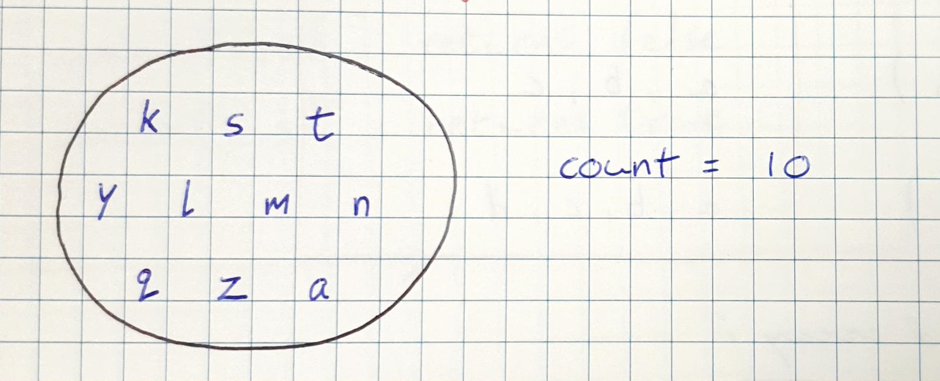
[**Code Present** 11](#_Toc95981777)

[**Test Data** 12](#_Toc95981778)

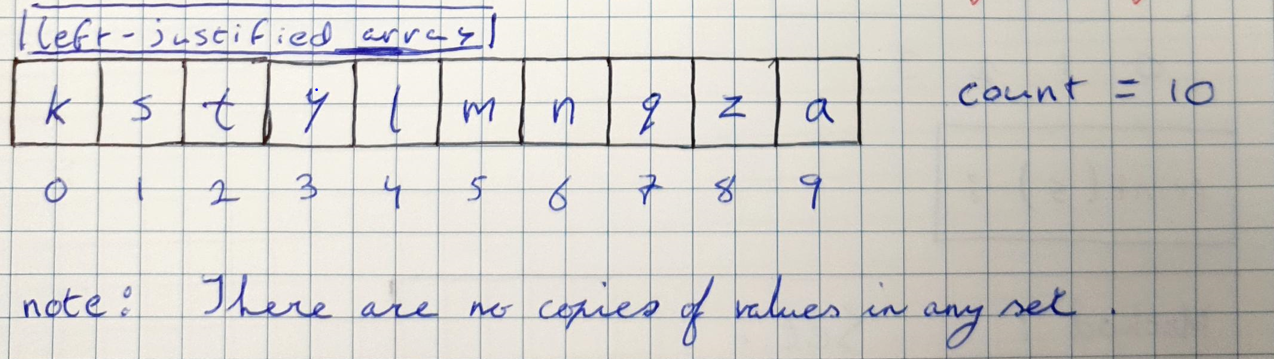
# **Introduction**

The goal of this project is to create a menu driven application, to implement a toolbox for the Set data type and its operations. A set is a collection of distinct elements. The elements that make up a set in java can be of any data type, so for this project we have chosen the 'char' data type. We will also be using a left-justified array to represent the set's data.

Here's an example of a set represented in maths:



And here's how it would be represented in the java program:



# **Description of the Problem**

The project will include the methods for the Sets, which are as follows:

* **Add(k ,s):** Inserts a new entry with key k into the set S and return the new entry. It must check that the item is not already in the set (in other words, it must check that there are no duplicates or allow for duplicates).
* **Contains(k , s):** Return true if this set S contains the specified element k. (If k is a member of the set)
* **RemoveAll(remSet , S):** Remove from this set S all of its elements that are contained in the specified set remSet. This operation effectively modifies this set so that its new value is the set difference of the two sets.
* **Print(S):** Print the contents of set S.
* **Union(unionSet, S):** Checks for a union between two sets. If a union is present, it will output the union.
* **IsEmpty(S):** Checks to set if the set is empty. If empty returns true, otherwise it returns as false.

Each operation should be accessible from a menu, that allows the user to pick and choose which operation to go with. The user should also have the option to terminate the program when they are finished with it.

# **Pseudocode**

**//** **add(k, S)**

public void add(char k, char S[])

{

boolean valid = true

for (int i = 0; i < count; i++)

{

if (k == S[i])

{

println("No Duplicates! No value added.")

valid = false

}

}

if (valid)

{

S[count] = k

println("Added" + S[count])

count+1

}

}

**//** **contains(k, s)**

public boolean contains(char k, char s[])

{

boolean contains = false

for (int i = 0; i < count; i++)

{

if (s[i] == k)

{

contains = true

}

}

return contains

}

**// print(s)**

public void print(char S[])

{

for (int i = 0; i < count; i++)

println(S[i])

}

**//is Empty**

public static boolean isEmpty()

{

if(count == 0)

return true;

else

return false;

}

**//Union**

public static void union(char unionSet[], char s[])

{

boolean valid;

for(int i = 0; i < unionCount; i ++)

{  
 valid = true;

}

for(int j = 0; j < count; j++)

{

If(s[j] == unionSet[i])

{

valid = false;

}

If(valid)

{

s[count] = unionSet[i];

count ++;

}

}

}

**//Remset**

public static void removeAll(char remSet[], char s[])

{  
 for(int i = 0; i< remSetCount; i ++)

{

for( int j = 0; j < count; j ++)

{

If(s[j] == remSet[i])

{

for(int index = j; index < count; index ++)

{

s[index] = s[index + 1];

}

count --;

}

}

}

}

# **Data Structures**

|  |  |
| --- | --- |
| **Add (k , s)** |  |
| **Method:** | **Set:** |
| Add(‘a’ , s) | a |
| Add(‘b’ , s) | a , b |
| Add(‘c’ , s) | a, b, c |
| Add(‘d’ , s) | a, b, c, d |

|  |  |
| --- | --- |
| **Contains (k , s)** |  |
| **Method:** | **Set:** |
| Add(‘a’ , s) | a |
| Add(‘b’ , s) | a , b |
| contains(‘a’ , s) | Returns True |
| contains(‘q’ , s) | Returns False |

|  |  |  |
| --- | --- | --- |
| **Remove(remset[] , s[])** |  |  |
| **Method:** | **Set:** | **RemSet:** |
| Add(‘j’ , s) | j | a, b, c, d, e |
| Add(‘a’ , s) | j, a | a, b, c, d, e |
| Add(‘k’ , s) | j, a , k | a, b, c, d, e |
| Add(‘e’ , s) | j, a, k, e | a, b, c, d, e |
| Remove | j, k |  |

|  |  |  |
| --- | --- | --- |
| **Union(remset[] , s[])** |  |  |
| **Method:** | **Set:** | **RemSet:** |
| Add(‘j’ , s) | j | k, e |
| Add(‘a’ , s) | j, a | k, e |
| Union | j, a, k, e |  |

|  |  |
| --- | --- |
| **isEmpty()** |  |
| **Method:** | **Set:** |
| isEmpty() | True |
| Add(‘j’ , s) | j |
| isEmpty() | False |
|  |  |

# **Functions**

**Menu:** The menu displays the list of methods the user can invoke by pressing the corresponding button. After each method has been carried out it returns the user to the menu, where they can choose another option or terminate the program.

**Add:** Adds the element (K) to the array (S). If there is a duplicate of the character k, it will not be taken into the array and will instead output “No Duplicates! No value has been added!”.

**Contains:** Allows the user to enter a character (k) and check if its currently an element in the array (s). If so, it will return true, if not, it will return false.

**RemSet:** Removes elements from the array (S) using elements from another array (remSet). If elements that are in remSet are in array S, then they will be removed, and a left shift will be performed to move the elements into their new places in the index.

**Print:** Will output all elements currently present in the array (s).

# **Extra Methods/Operations**

**IsEmpty:** If the count of the array is 0, the method will return true, showing the user that the array is empty. If the count is greater than 0, it will return false, meaning the array is not empty.

**Union:** Takes in another array (unionSet) and compares it against the main array (S). If the same char is present at the same index in both arrays, valid will be set to false, and not add duplicates. Otherwise, both arrays will be joined together.

# **Other Methods Used**

clear(): Simply clears the screen by print 100 new lines.

enter(): Prompts the user for input to continue with the program.

# 

# **Code Present**

**Java Code (Double Click on the text box to view all of the code)**

****

# **Test Data**

**Menu:**

**Shape, rectangle

Description automatically generated**

Simple menu, easy to navigate options.

**Add:**

**Shape, rectangle

Description automatically generated**

In this scenario, we chose to add character ‘j’ to the set. Once the element was added, the program outputs the element that was added.

**Contains:**

Text

Description automatically generated

Having previously added character ‘j’ to the set, we can now choose to see if the value has actually been entered into the set. This method prompts the user to enter the value they want to check is in the set S, it returns true in this situation because ‘j’ is in the set S.

Text

Description automatically generated

Here is what it looks like when an entered value is not in set S.

**RemoveAll:**

**Shape, rectangle

Description automatically generated**

Having previously added character ‘j’ to the set, we chose to remove it using the set remSet[]. The program prompts the user as to how many elements they would like to remove. Here we chose only 1, and then entered the character we wanted to remove. In this case ‘j’.

**Print (When set isn’t empty):**

**Shape, rectangle

Description automatically generated**

After adding ‘j’ back to the set S, we need to confirm that it has been added back. Using the print method, we can see that by the output, ‘j’ has been added back to the array.

**Print (When set is empty):**

**A picture containing shape

Description automatically generated**

With ‘j’ removed from the set, we can see that the ouput doesn’t print any elements in the set, as it is empty.

**Union:**

**Shape

Description automatically generated**

In set S we have ‘j’ as our element. After opting to use the Union method, we entered that we wanted to add two new elements to unionSet, those being ‘a’ and ‘k’.

**Shape, rectangle

Description automatically generated**

After enter is pressed, the two sets are combined, meaning ‘a’ and ‘k’ are now part of set S (if any of the union set values matched the ones in set S, they would not be added).

**isEmpty:**

**Shape, rectangle

Description automatically generated**

With set S containing the previously listed elements, after calling the isEmpty method, the program returns false, to show that set S is NOT empty.

**Shape, rectangle

Description automatically generated**

After all elements of set S have been removed, calling the isEmpty method makes the program returns true. This confirms that the set is empty.