

**Breadth-First Search**

At each step, visit, then add all children to queue Children will be visited after the parent

|  |  |
| --- | --- |
|  |  |
| A | Visit A  Add children (B, F) to q  q: [B, F] |
| B | Visit B  Add children (C, D, E) to q  q: [F, C, D, E] |
| F | Visit F  Add children (G) to q  q: [C, D, E, G] |
| C | Visit C  Add children (none) to q  q: [D, E, G] |
| D | Visit D  Add children (none) to q  q: [E, G] |
| E | Visit E  Add children (none) to q  q: [G] |
| G | Visit G  Add children (H, J, K) to q  q: [H, J, K] |
| H | Visit H  Add children (none) to q  q: [J, K] |
| J | Visit J  Add children (none) to q  q: [K] |
| K | Visit K  Add children (none) to q  q: [] |
| Now q is empty, so exit from the while loop.  Order: A B F C D E G H J K | |

Depth-first Search

|  |  |
| --- | --- |
|  |  |
|  | Visit A  A has no sibling, no push to  A.firstChild B |
|  | Visit B  B has a sibling (F), push to  B.firstChild C |
|  | Visit C  C has a sibling (D), push D to  C.firstChild = null Exit from inner while loop |
|  | Visit D  D has sibling (E), push E to  D.firstChild = null Exit from inner while loop |
|  | Visit E  E has no sibling, no push  E.firstChild = null Exit from inner while loop |
|  | Visit F  F has no sibling, no push  F.firstChild = G |
|  | Visit  G has no sibling, no push  G.firstChild = H |
|  | Visit H  H has a sibling (J), push to  H.firstChild = null Exit from inner while loop |
|  | J has a sibling (K), push to  J.firstChild = null Exit from inner while loop |
|  | K has no sibling, no push  K.firstChild = null Exit from inner while loop |
| Since is empty after the inner while loop, it also terminating the outer while loop.  Order: A B C D E F G H J K | |

**Question 3: Cat Pedigree**

Principle of coverage: the tests should visit all lines in the code.

**Testing a data class:**

* Normal case, excluding boundary normal case
* Boundary normal case
* Exceptional case, excluding boundary exceptional case
* Boundary exceptional case

*Cat.IsValidName*

|  |  |  |
| --- | --- | --- |
| **Case** | **Input Data** | **Expected Output** |
| IsValidName(null) should return false | null | false |
| IsValidName(“”) – empty string should return false | “” | false |
| IsValidName(“Fluffiest”) – string of 9 letters should return false | “Fluffiest” | false |
| IsValidName(“EvenFluffier” – string of 12 letters should return false | “EvenFluffier” | false |
| IsValidName(“A”) – string of 1 letter should return true | “A” | true |
| IsValidName(“Salem”) – string of 5 letters should return true | “Salem” | True |
| IsValidName(“Fluffier”) – string of 8 letters should return true | “Fluffier” | True |

*Cat.IsValidAge*

|  |  |  |
| --- | --- | --- |
| **Case** | **Input Data** | **Expected Output** |
|  |  |  |

**Testing a collection:**

* Normal case: When collection has multiple element
* Boundary case: When collection has one element
* Empty case: When collection has no elements

*Pedigree.AddCat*

This class has the following behaviours:

* Normal case: Add a cat normally (valid mothers, no mother so go to end of tree)
* Exceptional case: Add a cat with invalid mother (go to end of tree), Add cat with duplicate name
* Boundary case: Add a cat to empty collection (replace root)

|  |  |  |
| --- | --- | --- |
| **Case** | **Input Data** | **Expected Output** |
| AddCat add one cat to empty pedigree should make a pedigree of one cat | new Pedigree, Add (“Paws”) | True  pedigree.ToArray returns  [“Paws”] |
| AddCat add multiple cats to empty pedigree should make a pedigree of multiple cats | new Pedigree, Add (“Paws”, “Fluffy”, “Bunny”, “Cloud”) | True  pedigree.ToArray returns  [“Paws”, “Fluffy”, “Bunny”, “Cloud”] |
| AddCat add cats to valid mothers | new Pedigree,  Add (“Paws”)  Add (“Fluffy” as child of “Paws”)  Add(“Bunny” as child of “Fluffy”)  Add(“Cloud” as child of “Paws”) | True  Pedigree.ToArray returns 4 cats: [“Paws”, “Fluffy”, “Bunny”, “Cloud”]  “Paws” has “Fluffy” as first child  “Fluffy” has “Cloud” as first sibling and “Bunny” as first child |
| AddCat add cat to invalid mothers should add the new cat as sibling to the last cat | new Pedigree, with “Paws”  Add(“Cloud” as child of “Fluffy”) | True  Pedigree.ToArray returns 2 cats: [“Paws”, “Fluffy”]  “Paws” has “Fluffy” as first sibling |
| AddCat with duplicate name should return false | new Pedigree with “Paws”  Add(“Paws”) | False  Pedigree.ToArray return 1 cat: [“Paws”] |

*Pedigree.FindCat*

|  |  |  |
| --- | --- | --- |
| **Case** | **Input Data** | **Expected Output** |
| FindCat find cat in empty pedigree | new Pedigree,  Find(“Paws”) | null |
| FindCat find cat in pedigree of one cat | new Pedigree with Paws  Find(“Paws”) | “Paws” |

*Pedigree.ToArray*

|  |  |  |
| --- | --- | --- |
| **Case** | **Input Data** | **Expected Output** |
| ToArray empty pedigree should return array of length 0 | new Pedigree | Array length = 0 |