**Report**

**Procedure-1:**

convert from Representation 1 to Representation 2. Procedure 1: convert from Representation 1 to Representation 2. The output is an array with the new representation. The address of the new array is returned.

Idea

1-We build a method called base takes parameters (array,size)

2-it intializes two new parameters needed for bfsToDfs method which is (inputIndex,outputIndex) each intialized to 0 , build a new return address inside the base method ($v0)

3-then it enters bfsToDfs in which we get the stack ready taking the ($ra,InputIndex) as inputIndex will change a lot

4-check for the base case to stop recursion which is (i>=size) and go to end case

5-if not stopped multiply 4 to both index then adding to address to get value of both then make the output array to be equal the array given in the certain inputIndex

6-incrementing the outputIndex by 1 so we can get the next value next call

7-next time to get the recursive method working we should change the input given to parameter of inputIndex to be (2\*index+1)

8-jump and link to the same method to call recursive until base case returns it

9-after the (2\*index+1) recursive call is finished we call the same method but index(2\*index+1+1) and jump and link to recursive call so we can switch the array from bfs to dfs perfectly

10-end case closes the stack by loading each ($ra,InputIndex) then closing it and jr $ra

(more knowledge in handMadeReport)

**Procedure-2:**

convert from Representation 2 to Representation 1. The output is an array with the new representation. The address of the new array is returned.

Idea

1-We build a method base called takes parameters (array,size)

2-it intializes two new parameters needed for dfsToBfs method which is (inputIndex,outputIndex) each intialized to 0 , build a new return address inside the base method ($v0)

3-then it enters dfsToBfs in which we get the stack ready taking the ($ra,OutputIndex) as OutputIndex will change a lot

4-check for the base case to stop recursion which is (o>=size) and go to end case

5-if not stopped multiply 4 to both index then adding to address to get value of both then make the output array given in the certain outputIndex to be equal the array in inputindex

6-incrementing the inputIndex by 1 so we can get the next value next call

7-next time to get the recursive method working we should change the input given to parameter of OutputIndex to be (2\*Outputindex+1)

8-jump and link to the same method to call recursive until base case returns it

9-after the (2\*index+1) recursive call is finished we call the same method but index(2\*Outputindex+1+1) and jump and link to recursive call so we can switch the array from dfs to bfs perfectly

10-end case closes the stack by loading each ($ra,OutputIndex) then closing it and jr $ra return output array

(more knowledge in handMadeReport)

**Procedure-3:**

Take a value as an input in addition to the tree with Representation 1 and do breadth- first search.

Idea

1-We build a method called bfsSearch takes parameters (array,size,target)

2-it intializes one new parameters needed for bfsSearchH method which is (index intialized to 0 , and intialize ($v0) to return -1 if target not found

3-then it enters bfsSearchH in which we get the stack ready taking the ($ra)

4-check for the base case to stop recursion which is (index>=size) and go to exit case

5-if not stopped multiply 4 to index then adding to address to get value of it then check if value equal target

6-if not found incrementing the index by 1 so we can get the next value next call recursively

7-if found enters Found case in which we intialize stack to store ($ra,size) and changing size to the index value then incrementing by 1 before starting the loop and loading new variable for (level) = 0 then enters loop

8-if index equals zero the loop ends else dividing index by 2 then adding 1 to the level and keep looping until index == 0

9- endloop case which closes stack by loading ($ra,size) then jr $ra which return level of target

10-exit case closes the stack by loading each ($ra) then closing it and jr $ra and return -1

(more Knowledge in Java Implementation 🡪linear Search)

**Procedure-4:**

Take a value as an input in addition to the tree with Representation 2 and do breadth- first search.

Idea

1-We build a method base called takes parameters (array,size)

2-it intializes two new parameters needed for dfsToBfs method which is (inputIndex,outputIndex) each intialized to 0 , build a new return address inside the base method ($v0)

3-then it enters dfsToBfs in which we get the stack ready taking the ($ra,OutputIndex) as OutputIndex will change a lot

4-check for the base case to stop recursion which is (o>=size) and go to end case

5-if not stopped multiply 4 to both index then adding to address to get value of both then make the output array given in the certain outputIndex to be equal the array in inputindex

6-incrementing the inputIndex by 1 so we can get the next value next call

7-next time to get the recursive method working we should change the input given to parameter of OutputIndex to be (2\*Outputindex+1)

8-jump and link to the same method to call recursive until base case returns it

9-after the (2\*index+1) recursive call is finished we call the same method but index(2\*Outputindex+1+1) and jump and link to recursive call so we can switch the array from dfs to bfs perfectly

10-end case closes the stack by loading each ($ra,OutputIndex) then closing it and jr $ra

11-We build a method called bfsSearch takes parameters (array,size,target) and the array is the array returned from the first method and the size of it

12-it intializes one new parameters needed for bfsSearchH method which is (index intialized to 0 , and intialize ($v0) to return -1 if target not found

13-then it enters bfsSearchH in which we get the stack ready taking the ($ra)

14-check for the base case to stop recursion which is (index>=size) and go to exit case

15-if not stopped multiply 4 to index then adding to address to get value of it then check if value equal target

16-if not found incrementing the index by 1 so we can get the next value next call recursively

17-if found enters Found case in which we intialize stack to store ($ra,size) and changing size to the index value then incrementing by 1 before starting the loop and loading new variable for (level) = 0 then enters loop

18-if index equals zero the loop ends else dividing index by 2 then adding 1 to the level and keep looping until index == 0

19- endloop case which closes stack by loading ($ra,size) then jr $ra which returns level of target

20-exit case closes the stack by loading each ($ra) then closing it and jr $ra and return -1

(more Knowledge in Java Implementation 🡪linear Search)