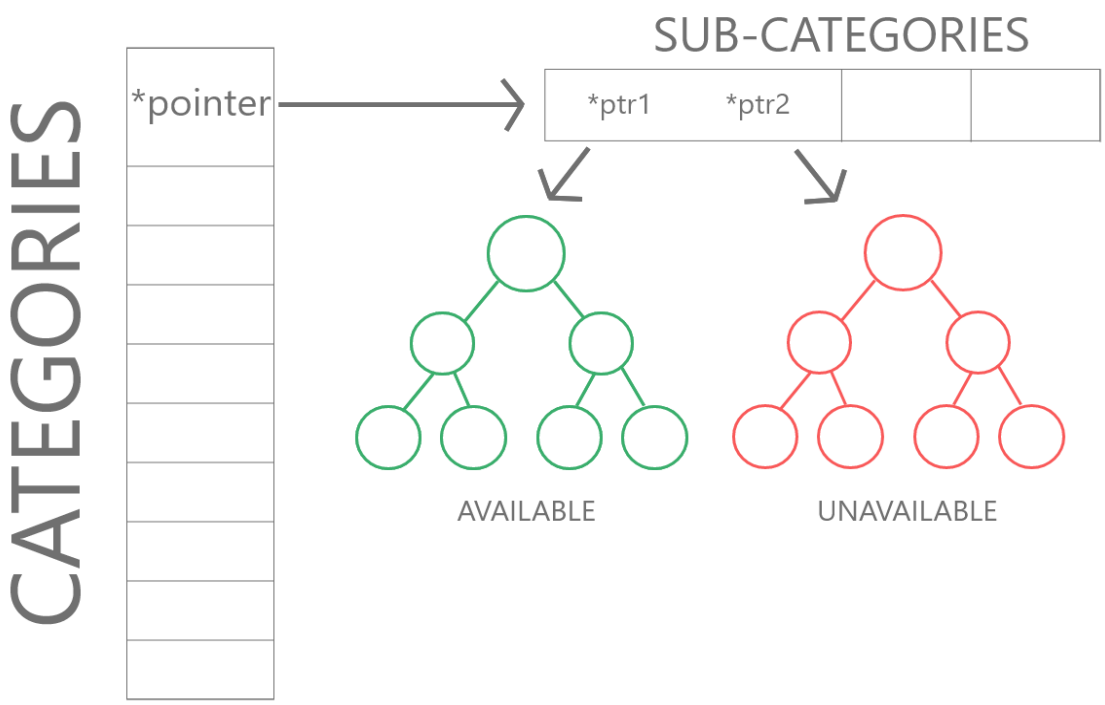
The Rural Company

Rural Company is an online home services platform. Launched never, Rural Company today operates worldwide. The platform helps customers book reliable & high quality services like beauty treatments, massages, haircuts, home cleaning, handymen, appliance repair, painting, pest control and more – delivered by trained professionals conveniently at home.
The moto is to provide options of selecting different workers in categories and sub-categories.

Structure Used to Store Workers



Data Structures Used

1) Multi Lists – List of Lists
2) Trees

* The index used in order to insert the workers in the tree is the rating of the worker. By default we have increased the rating from 0 to n as we take n workers as input. The rating can be changed if required and specified.

Code (Important Bits)

1.

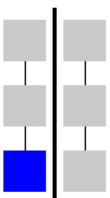
```
// An AVL tree node
class Node
{
    public:
        string name;
        string phone_number;
        int key; //Relative rating of the person
        Node *left;
        Node *right;
        int height;

};
```

2.

```
// Recursive function to insert a key in the subtree rooted with
node and returns the new root of the subtree.
Node* insert(Node* node, int key, string name, string number)
{
    /* 1. Perform the normal BST insertion */
    if (node == NULL)
        return(newNode(key, name, number));
    if (key < node->key)
        node->left = insert(node->left, key, name, number);
    else if (key > node->key)
        node->right = insert(node->right, key, name, number);
    else // Equal keys are not allowed in BST
        return node;
    /* 2. Update height of this ancestor node */
    node->height = 1 + max(height(node->left),
                           height(node->right));
    /* 3. Get the balance factor of this ancestor
        node to check whether this node became
        unbalanced */
    int balance = getBalance(node);
    // If this node becomes unbalanced, then there are 4 cases
    // Left Left Case
    if (balance > 1 && key < node->left->key)
        return rightRotate(node);
    // Right Right Case
    if (balance < -1 && key > node->right->key)
        return leftRotate(node);
    // Left Right Case
    if (balance > 1 && key > node->left->key)
    {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    // Right Left Case
    if (balance < -1 && key < node->right->key)
    {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    /* return the (unchanged) node pointer */
    return node;
}
```

3.

```
// Recursive function to delete a key from the subtree rooted with node
and returns the new root of the subtree.
Node* deleteNode(Node* root, int key)
{  // STEP 1: PERFORM STANDARD BST DELETE
    if (root == NULL)
        return root;
    // If the key to be deleted is smaller  than the root's key, then it
lies in left subtree
    if ( key < root->key )
        root->left = deleteNode(root->left, key);
    // If the key to be deleted is greater than the root's key, then it
lies in right subtree
    else if( key > root->key )
        root->right = deleteNode(root->right, key);
    // if key is same as root's key, then This is the node to be deleted
    else
    { // node with only one child or no child
        if( (root->left == NULL) ||
            (root->right == NULL) )
        {  Node *temp = root->left ? root->left : root->right;
            // No child case
            if (temp == NULL)
            {   temp = root;
                root = NULL;
            }
            else // One child case
            *root = *temp; // Copy the contents of the non-empty child
            free(temp);}
        else
        { //node with two children: Get the inorder successor (smallest
in the right subtree)
            Node* temp = minValueNode(root->right);
            // Copy the inorder successor's data to this node
            root->key = temp->key;
            // Delete the inorder successor
            root->right = deleteNode(root->right,
                                     temp->key);}}
    // If the tree had only one node then return
    if (root == NULL)
    return root;
    // STEP 2: UPDATE HEIGHT OF THE CURRENT NODE
    root->height = 1 + max(height(root->left), height(root->right));
    // STEP 3: GET THE BALANCE FACTOR OF THIS NODE (to check whether
this node became unbalanced)
    int balance = getBalance(root);
    // If this node becomes unbalanced,
    // then there are 4 cases
    // Left Left Case
    if (balance > 1 && getBalance(root->left) >= 0)
        return rightRotate(root);
    // Left Right Case
    if (balance > 1 && getBalance(root->left) < 0)
    {   root->left = leftRotate(root->left);
        return rightRotate(root);
    }
    // Right Right Case
    if (balance < -1 && getBalance(root->right) <= 0)
        return leftRotate(root);
    // Right Left Case
    if (balance < -1 && getBalance(root->right) > 0)
    {   root->right = rightRotate(root->right);
        return leftRotate(root);}
    return root; }
```



4.

```
//Creating an array of Node* and initializing it to NULL
line->55-63
    vector<Node*> roots;
    vector<Node*>::iterator it;
    Node * p = NULL;
    for(int i = 0; i < 1000; i++){
        roots.push_back(new Node());
    }
    for(it = roots.begin(); it < roots.end(); it++){
        *it = NULL;
    }
```

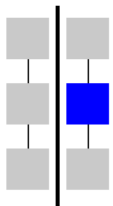
5.

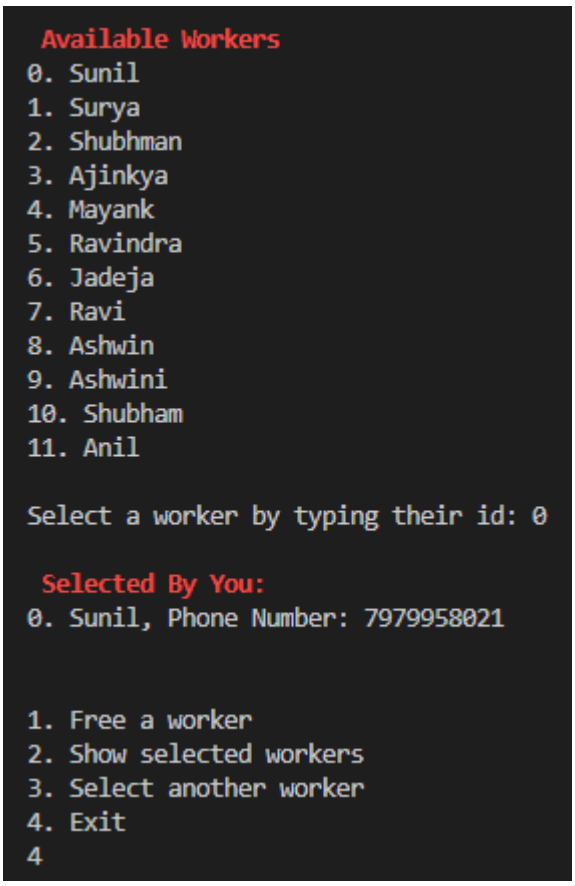
```
//Building the complete data structure by taking input from
input.txt, line->66-102
    while (file_input) {
        file_input >> line;

        numberOfCategories = stoi(line);


        for(int i = 1; i <= numberOfCategories; i++){
            file_input >> nameOfCategory;
            file_input >> line;

            numberOfSubcategories = stoi(line);

            vector<pair<string, pair<Node*, Node*>>>
temp_subcategoriesAndWorkers;

            for(int j = 0; j < numberOfSubcategories; j++){
                file_input >> nameOfSubcategory;
                file_input >> line;

                numberOfWorkersInACategory = stoi(line);
                vector<pair<Node*, Node*>> temp_workers;

                for(int k = 0; k < numberOfWorkersInACategory; k++){
                    file_input >> nameOfWorker;
                    file_input >> phoneNumber;
                    roots[index] = insert(roots[index], k,
nameOfWorker, phoneNumber);

                }

                temp_subcategoriesAndWorkers.push_back(make_pair(nam
eOfSubcategory, make_pair(roots[index], roots[1000-index-1])));
                index++;
            }
            categoriesAlongWithSubcategoriesAndWorkers.push_back(mak
e_pair(nameOfCategory,temp_subcategoriesAndWorkers ));
        }
        break;
    }

    file_input.close();
```

Output

```
 Categories
0. Applicance-Repair
1. Cleaning-and-Pest-Control
2. Grooming
3. Housekeeping
Enter your Choice: 1

 Sub-categories
0. Bathroom-and-kitchen
1. Sofa-and-Carpet-Cleaning
2. Pest-Control
3. Full-Home-Cleaning
Enter your Choice: 1

 Available Workers
0. Sunil
1. Surya
2. Shubhman
3. Ajinkya
4. Mayank
5. Ravindra
6. Jadeja
7. Ravi
8. Ashwin
9. Ashwini
10. Shubham
11. Anil

Select a worker by typing their id: 2

 Selected By You:
2. Shubhman, Phone Number: 7326096860


1. Free a worker
2. Show selected workers
3. Select another worker
4. Exit
3


 Categories
0. Applicance-Repair
1. Cleaning-and-Pest-Control
2. Grooming
3. Housekeeping
Enter your Choice: 1

 Sub-categories
0. Bathroom-and-kitchen
1. Sofa-and-Carpet-Cleaning
2. Pest-Control
3. Full-Home-Cleaning
Enter your Choice: 1
```

```
 Categories
0. Applicance-Repair
1. Cleaning-and-Pest-Control
2. Grooming
3. Housekeeping
Enter your Choice: 1

 Sub-categories
0. Bathroom-and-kitchen
1. Sofa-and-Carpet-Cleaning
2. Pest-Control
3. Full-Home-Cleaning
Enter your Choice: 1
 Worker already selected from the category,
 select another category or free the worker
 first!

1. Free a worker
2. Show selected workers
3. Select another worker
1

1

0. Shubhman
Enter the key to be deleted
0

1. Free a worker
2. Show selected workers
3. Select another worker
4. Exit
3


 Categories
0. Applicance-Repair
1. Cleaning-and-Pest-Control
2. Grooming
3. Housekeeping
Enter your Choice: 1

 Sub-categories
0. Bathroom-and-kitchen
1. Sofa-and-Carpet-Cleaning
2. Pest-Control
3. Full-Home-Cleaning
Enter your Choice: 1

 Available Workers
0. Sunil
1. Surya
2. Shubhman
3. Ajinkya
```





* If a worker is selected from a specific sub-category then another worker cannot be selected from the same sub-category, the selected worker needs to be freed first.