~~M-3 assignt~~

* Diction ary

- The dictionary ADT models a searchable collection of key-element items.

→ Dictionary ADT methods:-

1) get (k) :-

- If the dictionary has an item with key k, returns its element, else, returns its element, else, returns NULL.

2) getAll (k) :-

- returns an iterator of entries with key k

3) put (k, o) :-

- inserts item (k, o) into the dictionary

4) remove (k) :-

- If the dictionary has an item with k, removes it from the dictionary and returns its element, else return NULL.

5) removeAll (k) :-

- remove all entries with key k, return an iterator of these entries

6) size() , isEmpty()

chart for class node

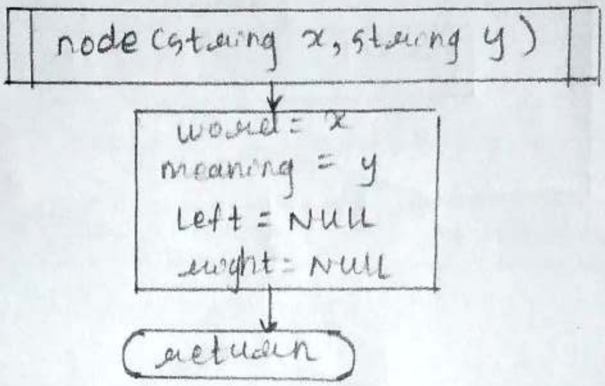
class node

string word
string meaning

node * left=NULL
node * right=NULL

friend class Dictionary

→ Flowchart for node (string x, string y)



→ Flowchart for class Dictionary

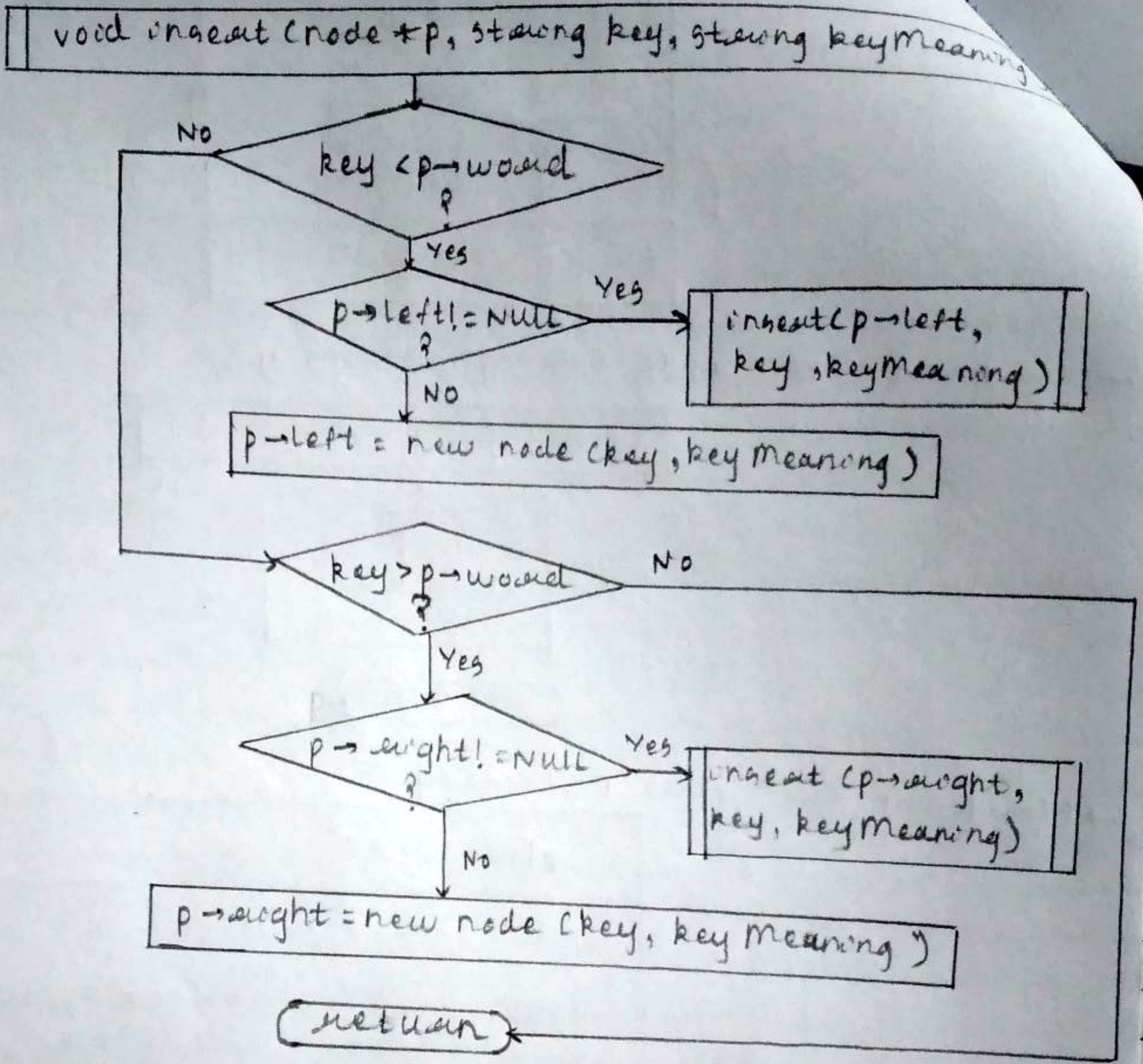
class Dictionary

Dictionary() { root=q=NULL }

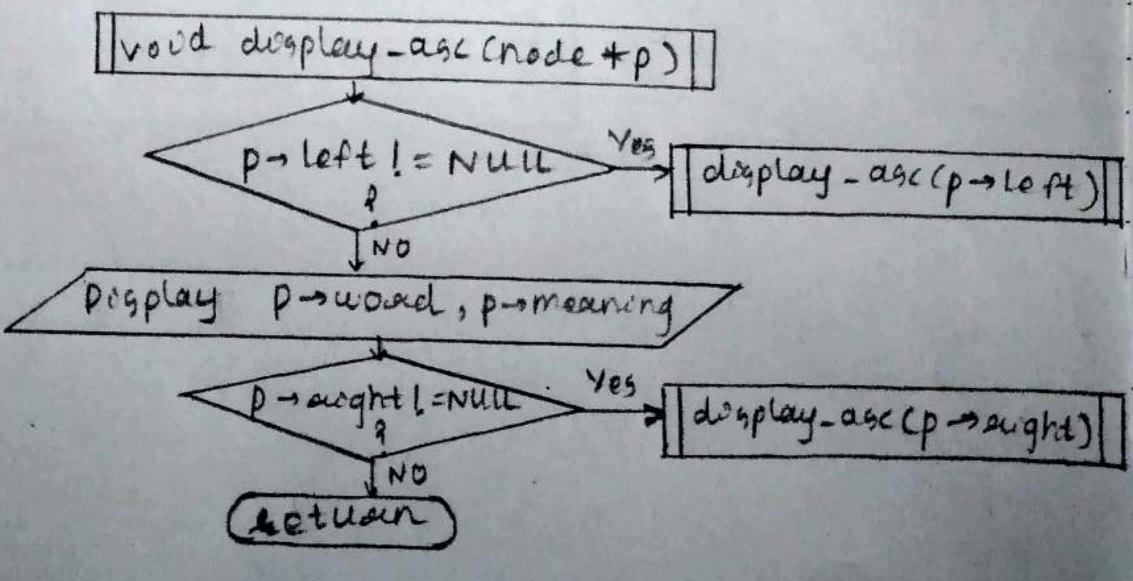
void insert (node *, string, string)
void display_asc (node *)
void display_desc (node *)
void comparisons (node *, string)
void updateWord (node *, string)

node * root, *q

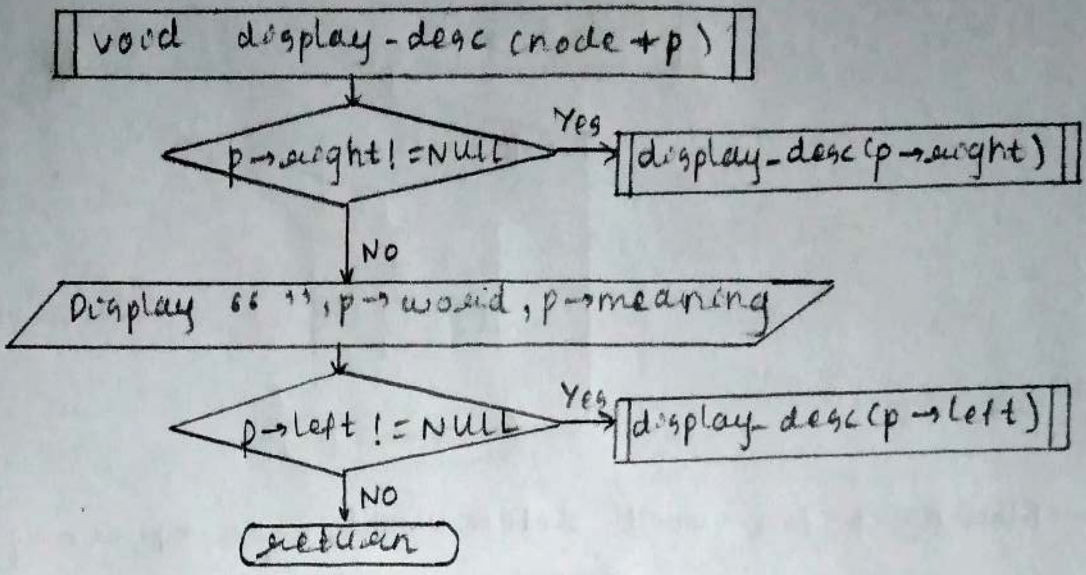
→ Flowchart for void insert (node * p, string key, string keyMeaning)


void insert (node *p, string key, string keyMeaning)
key < p→word ?
No
Yes
p→left != NULL ?
Yes
insert(p→left, key, keyMeaning)
NO
p→left = new node (key, keyMeaning)
key > p→word ?
No
Yes
p→right != NULL ?
Yes
insert (p→right, key, keyMeaning)
No
p→right = new node (key, keyMeaning)
return

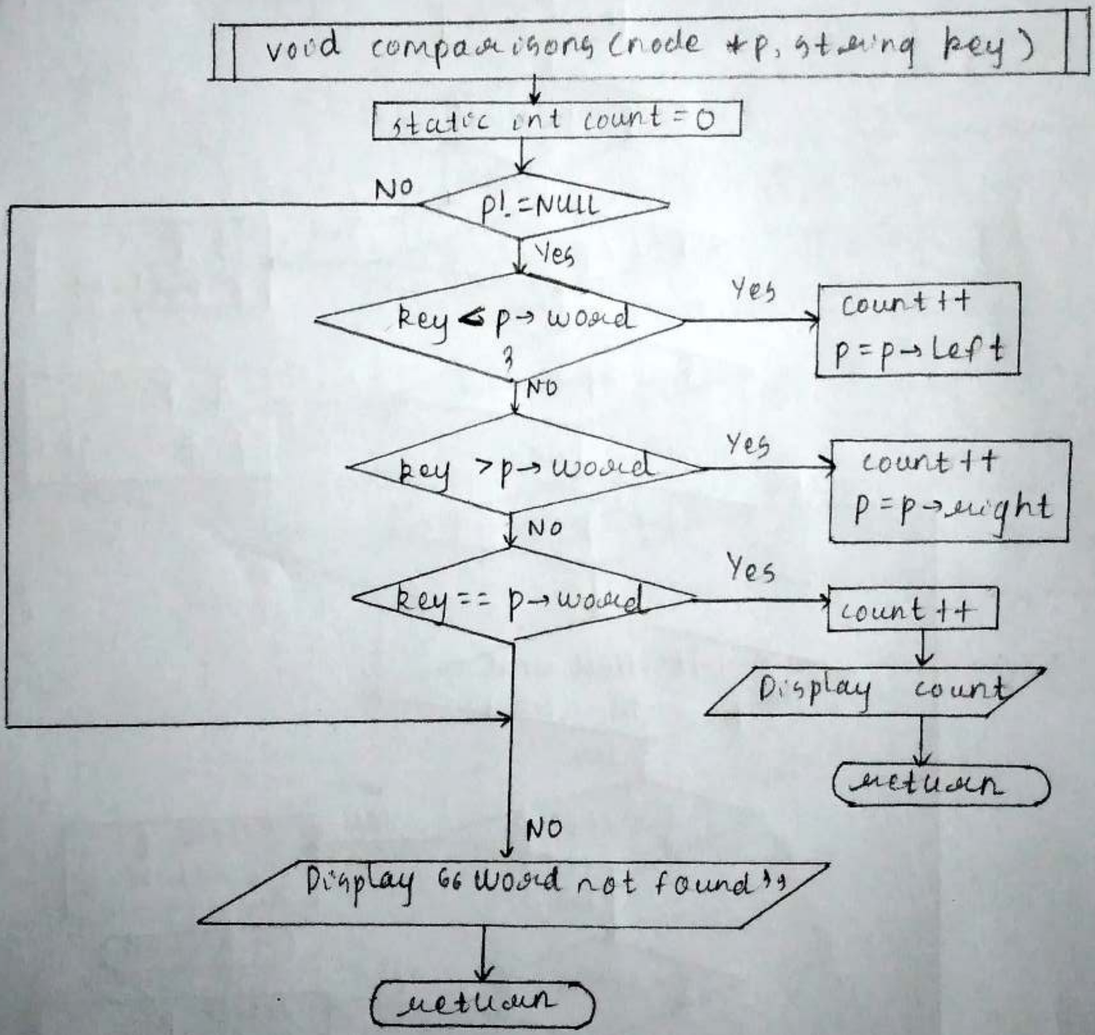
→ Flowchart for void display_asc (node *p)



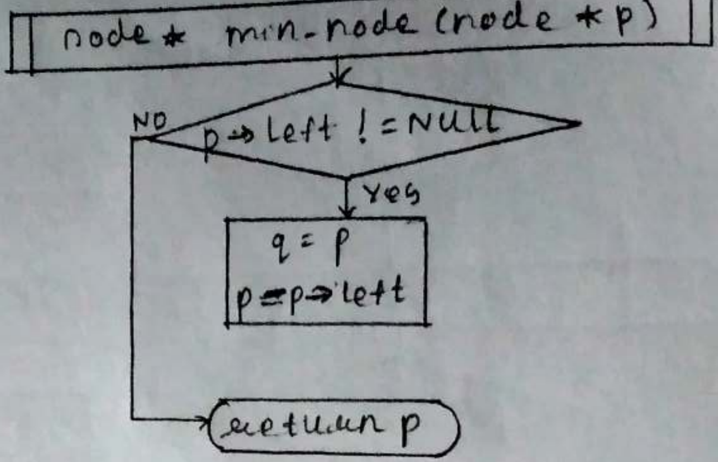
…rt for void display_desc (node *p)



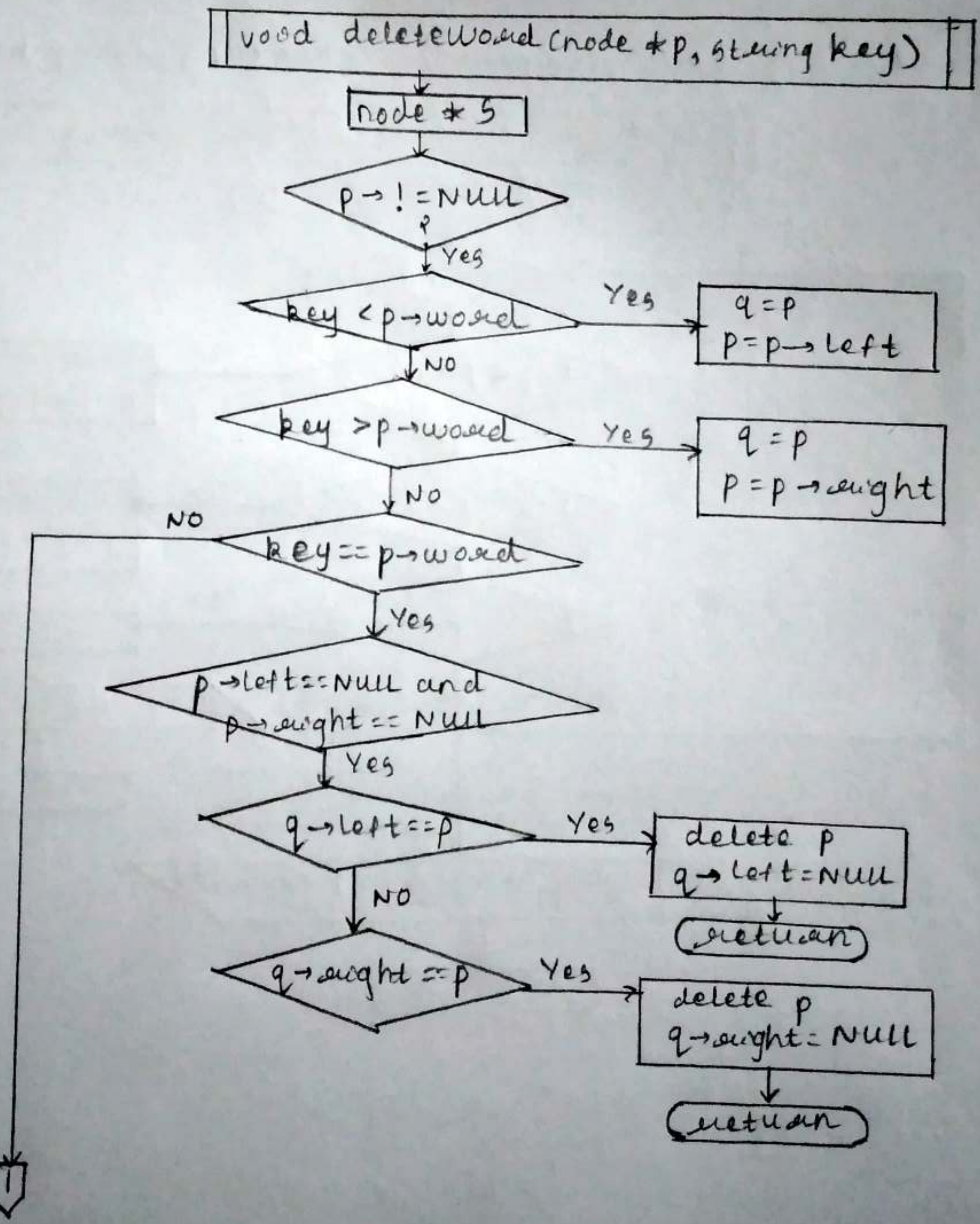
→ Flowchart for void comparisons(node* p, string key)

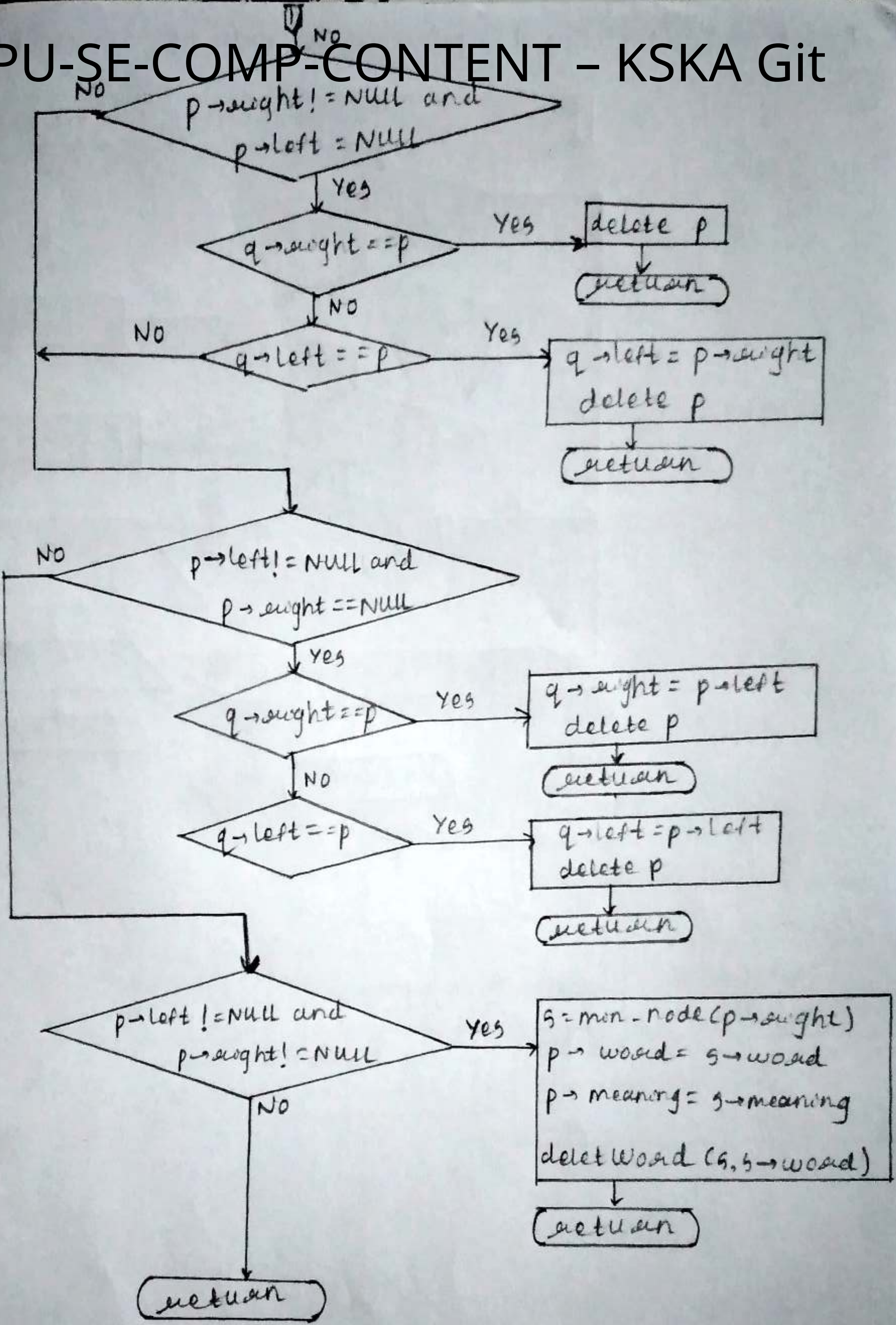


→ Flowchart for node* min-node (node *p)

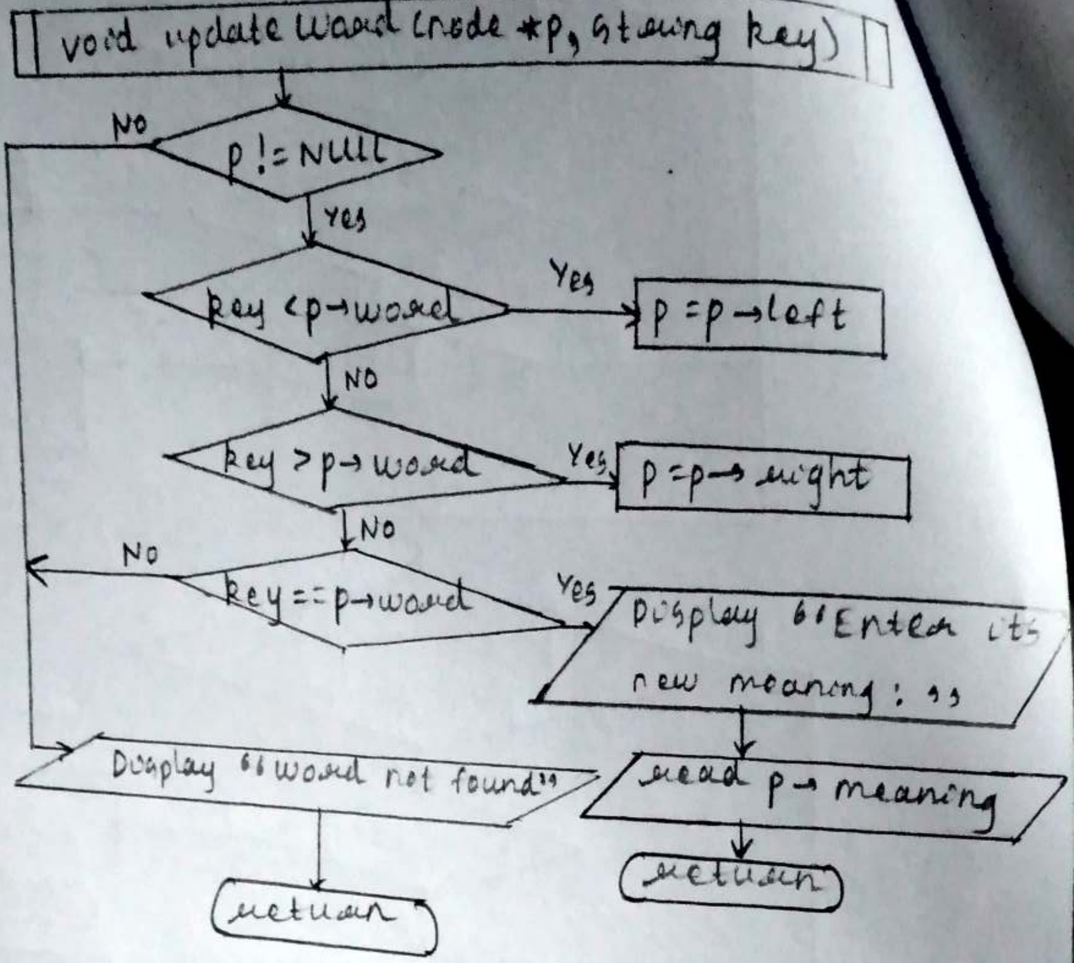


→ Flowchart for void deleteWord (node *p, string key)

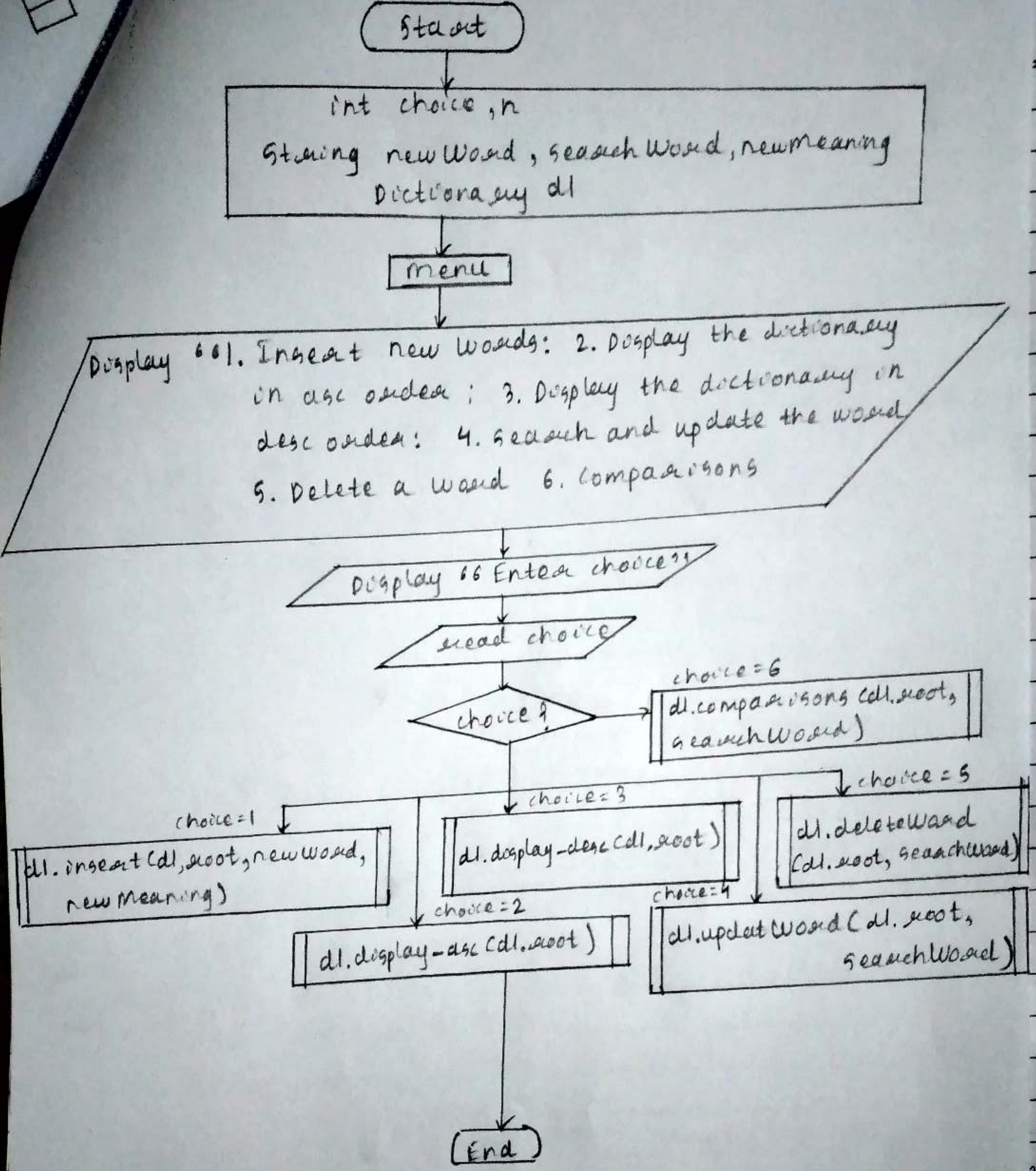



NO
NO
p→right != NULL and p→left = NULL
Yes
q→right == p
Yes
delete p
return
NO
NO
q→left == p
Yes
q→left = p→right
delete p
return
NO
p→left != NULL and p→right == NULL
Yes
q→right == p
Yes
q→right = p→left
delete p
return
NO
q→left == p
Yes
q→left = p→left
delete p
return
p→left != NULL and p→right != NULL
Yes
s = min_node(p→right)
p→ word = s→word
p→ meaning = s→meaning
deletWord (s, s→word)
return
NO
return

→ Flowchart for void updateWord (node *p, string key)



t for int main()


Start
int choice, n
String newWord, searchWord, newMeaning
Dictionary d1
menu
Display "1. Insert new words: 2. Display the dictionary in asc order ; 3. Display the dictionary in desc order: 4. Search and update the word 5. Delete a word 6. Comparisons
Display "Enter choice"
read choice
choice?
choice=6
d1.comparisons (d1.root, searchWord)
choice=1
d1.insert (d1, root, newWord, newMeaning)
choice=3
d1.display-desc (d1, root)
choice=5
d1.deleteWord (d1.root, searchWord)
choice=2
d1.display-asc (d1.root)
choice=4
d1.updatWord (d1.root, searchWord)
End

→ Pseudocode for class node

```
1. Declare string word
           string meaning
2. Initialize node * left = NULL
              node * right = NULL
3. Declare ~~node~~ friend class Dictionary
```

→ Pseudocode for node (string x, string y)

```
1. Initialize word = x
              meaning = y
2. Initialize left = NULL
              right = NULL
```

→ Pseudocode for class Dictionary

```
1. Declare node * root, *q
2. create function void insert (node *, string, string)
                   void display_asc (node *)
                   void display_desc (node *)
                   void comparisons (node *, string)
                   void updateWord (node *, string)
```

→ Pseudocode for Dictionary()

```
1. Initialize root = NULL
              q = NULL
```

→ Pseudocode for void ~~Dict~~ insert (node *p, string key,
                                      string keymeaning)

```
1. if key < p→word then
      if p→left != NULL
            call function insert (p→left, key,
                                  keymeaning)
      else
            initialize p→left = new node (key, keymeaning)
```

```
else if (key > p→word) then
        if p→right != NULL then
                call function insert (p→right, key, keymean-
                                                         ing)

        else
                initialize p→right = new node (key, keyMea-
                                                        ning)

2   return
```

→ Pseudocode for void display_asc (node *p)

```
1.  if p→left != NULL then
          call function display_asc (p→left)
2   Display p→word , p→meaning
3.  if p→right != NULL then
          call function display_asc (p→right)
4.  return
```

→ Pseudocode void display_desc (node *p)

```
1.  if p→right != NULL then
          call function display_desc (p→right)
2   Display p→word , p→meaning
3.  if p→left != NULL then
          call function display_desc (p→left)
4.  return
```

→ Pseudocode for void comparisons (node *p, string key)

```
1.  initialize static int count = 0
2.  while p != NULL do
    begin
          if (key < p→word) then
                increment count
                initialize p = p→left
```

```
            else if (key > p->word) then
                    increment count
                    p = p->right
            else  if  (key == p->word) then
                    increment count
                    Display "Number of comparisons to
                            find the word : ", count

                    return

   end
3. Display "word Not found"
4. return

→ Pseudo code for node* ~~dictionary~~ min_node(node *p)
1. while p->left != NULL do
   begin
           initialize q = p
                      p = p -> left
   end
2. return p

→ Pseudocode for void deleteWord(node *p, string key)
1. declare node *s
2. while p != NULL do
   begin
        if key < p->word then
               initialize  q = p
                           p = p-> left
        else if key > p->word then
               initialize  q = p
                           p = p-> right
        else if key == p->word
               if p->left == NULL and p->right = NULL then
            if q->left == p then
                delete p
                initialize q->left = NULL
                return
            if q->right == p then
                delete p
                initialize q->right = NULL
                return
        if p->right != NULL and p->left == NULL then
            if q->right == p then
                initialize q->right = p->right
                delete p
                return
            else if q->left == p then
                initialize q->left = p->right
                delete p
                return
        else if (p->left != NULL and p->right == NULL) then
            if q->right == p then
                initialize q->right = p->left
                delete p
                return
            else if q->left == p then
                initialize q->left = p->left
                delete p
                return
    else if p->left != NULL and p->right != NULL then
            initialize s = min_node(p->right)
            store p->word = s->word
            store p->meaning = s->meaning
            call function deleteWord(s, s->word)
            return
```

```
   end
3. Display "word not found"
4. return
```

→ Pseudocode for void updateWord(node *p, string key)

```
1. while p! = NULL ~~then~~ do
   ~~do~~ begin
          if key < p→word then
              store p=p→left
          else if (key > p→word) then
              store p = p→ ~~left~~ right
          else if key = =p→word then
              Display "Enter its new meaning! "
              read p→meaning
              return
   end
2. Display "word not found! "
3. return
```

→ Pseudocode for int main()

```
1. Start
2. declare int choice, n
           string newWord, searchWord, newMeaning
3. Create Dictionary d1
4. menu:
       Display "Dictionary: "
       Display "1. Insert new word: 2. Display the
                dictionary in asc order: 3. Display
                the dictionary in des order: 4. Search
                and update the word: 5. Delete a word
                6. Comparisons "
       Display "Enter your choice: "
```

```
~~read~~
read choice
switch (choice)
        case 1:
                dl.insert (dl.root, new Word, new Meaning)
                break
        case 2:
                call function dl.display_asc (dl.root)
                break
        case 3:
                call function dl.display_desc (dl.root)
                break
        case 4:
                dl.updateWord (dl.root, searchWord)
                break
        case 5:
                call function dl.deleteWord (dl.root,
                                                searchWord)
                break
        case 6:
                call function dl.comparisons (dl.root,
                                                searchWord)
        default:
        ~~case 7:~~
                Display "Invalid input
        if choice != 7 then
                goto menu
5.  End
```

Q1. Discuss about various operations of binary search tree.

Soln: various operations that can be performed on binary search tree are:-

1) Insertion of a node in a binary tree

→ Algorithm:-

1. Read value for the ~~ete~~ node which is to be created and store it in a node called New.
2. Initially if (root != NULL) then root = New
3. Again read the next value of node created in New.
4. If (New→value < root→value) then attach New node as a left child of root otherwise attach New node as a right child ~~the~~ of root.

2) Deletion of an element from the binary tree

→ The node to be deleted may be a leaf node:-

- In this case simply delete a node and set null pointer to it's parents those side at which this deleted node exist.

→ The node to be deleted has one child:-

- In this case the child of the node to be deleted is appended to its parent node.

→ The node to be deleted has two children:-

- In this case node to be deleted is replaced by its in-order successor node.

3) Searching through the BST:-

→ Compare the target value with the element in the root node:-

- if the target value is equal, the search ~~the left subtree~~ is successful.
- if target value is less, search the left subtree.

- if the target value is greater, search the right subtree.
- if the subtree is empty, the search is unsuccessful.

Q2. What is Dictionary ADT and its operations?

Ans. The dictionary ADT models a searchable collection of key-element items.

→ Dictionary ADT methods:

1) get (k) :-
- If the dictionary has an item with key k, returns its element, else, returns its element, else, returns NULL.

2) getALL (k) :-
- returns an iterator of entries with key k

3) put (k,o) :-
- inserts item (k,o) into the dictionary

4) remove (k) :-
- if the dictionary has an item with k, removes it from the dictionary and returns its element, else return ~~of~~ NULL.

5) removeALL (k) :-
- remove all entries with key k, return an iterator of these entries

6) size (), is Empty()

* Conclusion :-
- Successfully operated on binary search tree data structure
- Applied binary search tree for dictionary operations.