[*] Untitled1

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_KEYS 2
typedef struct Node {
    int keys[MAX_KEYS + 1];
    struct Node *children[MAX_KEYS + 2];
    int num_keys;
    int is_leaf;
} Node;
Node* create_node(int is_leaf);
void traverse_tree(Node *root);
Node* insert(Node *root, int key);
void split_child(Node *parent, int index, Node *child);
void insert_non_full(Node *node, int key);
int main() {
    Node *root = create_node(1);
    int keys[] = {10, 20, 5, 6, 15, 30, 25, 40};
    int num_keys = sizeof(keys) / sizeof(keys[0]);
    for (int i = 0; i < num_keys; i++) {
        root = insert(root, keys[i]);
}
    printf("Tree traversal:\n");
    traverse_tree(root)
    return 0;
}
Node* create_node(int is_leaf) {
    Node *new_node = (Node*)malloc(sizeof(Node));
    new_node->is_leaf = is_leaf;
    new_node->num_keys = 0;
    for (int i = 0; i <= MAX_KEYS; i++) {
        new_node->children[i] = NULL;
    }
    return new_node;
}
```

[*] Untitled1

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_KEYS 2
typedef struct Node {
    int keys[MAX_KEYS + 1];
    struct Node *children[MAX_KEYS + 2];
    int num_keys;
    int is_leaf;
} Node;
Node* create_node(int is_leaf);
void traverse_tree(Node *root);
Node* insert(Node *root, int key);
void split_child(Node *parent, int index, Node *child);
void insert_non_full(Node *node, int key);
int main() {
    Node *root = create_node(1);
    int keys[] = {10, 20, 5, 6, 15, 30, 25, 40};
    int num_keys = sizeof(keys) / sizeof(keys[0]);
    for (int i = 0; i < num_keys; i++) {
        root = insert(root, keys[i]);
}
    printf("Tree traversal:\n");
    traverse_tree(root)
    return 0;
}
Node* create_node(int is_leaf) {
    Node *new_node = (Node*)malloc(sizeof(Node));
    new_node->is_leaf = is_leaf;
    new_node->num_keys = 0;
    for (int i = 0; i <= MAX_KEYS; i++) {
        new_node->children[i] = NULL;
    }
    return new_node;
}
void traverse_tree(Node *root) {
    if (root != NULL) {
```

```
    parent->children[index + 1] = new_child;
    parent->keys[index] = child->keys[1];
    parent->num_keys++;
    new_child->num_keys = 1;
    child->num_keys = 1;
    for (int i = 0; i < 2; i++) {
        new_child->keys[i] = child->keys[i + 2];
        new_child->children[i] = child->children[i + 2];
    }
    new_child->children[2] = child->children[4];
}
void insert_non_full(Node *node, int key) {
    int i = node->num_keys - 1;

    if (node->is_leaf) {
        while (i >= 0 && key < node->keys[i]) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
        node->num_keys++;
    } else {
        while (i >= 0 && key < node->keys[i]) {
            i--;
        }
        i++;
        if (node->children[i]->num_keys == MAX_KEYS) {
            split_child(node, i, node->children[i]);
            if (key > node->keys[i]) {
                i++;
            }
        }
        insert_non_full(node->children[i], key);
    }
}
```

[*] Untitled1

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_KEYS 3
#define MIN_KEYS 1
typedef struct Node {
    int keys[MAX_KEYS + 1];
    struct Node *children[MAX_KEYS + 2];
    int num_keys;
    int is_leaf;
} Node;
Node* create_node(int is_leaf);
void traverse_tree(Node *root);
Node* insert(Node *root, int key);
void split_child(Node *parent, int index, Node *child);
void insert_non_full(Node *node, int key);
void free_tree(Node *root);

int main() {
    Node *root = create_node(1);
    int keys[] = {10, 20, 5, 6, 15, 30, 25, 40, 50, 45, 55};
    int num_keys = sizeof(keys) / sizeof(keys[0]);

    for (int i = 0; i < num_keys; i++) {
        root = insert(root, keys[i]);
    }
    printf("Tree traversal:\n");
    traverse_tree(root);
    free_tree(root);

    return 0;
}
Node* create_node(int is_leaf) {
    Node *new_node = (Node*)malloc(sizeof(Node));
    new_node->is_leaf = is_leaf;
    new_node->num_keys = 0;
    for (int i = 0; i <= MAX_KEYS; i++) {
```

[*] Untitled1

```
37          new_node->children[i] = NULL;
38      }
39      return new_node;
40  }
41
42  void traverse_tree(Node *root) {
43      if (root != NULL) {
44          int i;
45          for (i = 0; i < root->num_keys; i++) {
46              if (!root->is_leaf) {
47                  traverse_tree(root->children[i]);
48              }
49              printf("%d ", root->keys[i]);
50          }
51          if (!root->is_leaf) {
52              traverse_tree(root->children[i]);
53          }
54      }
55  }
56  Node* insert(Node *root, int key) {
57      if (root->num_keys == MAX_KEYS) {
58          Node *new_root = create_node(0);
59          new_root->children[0] = root;
60          split_child(new_root, 0, root);
61          insert_non_full(new_root, key);
62          return new_root;
63      } else {
64          insert_non_full(root, key);
65          return root;
66      }
67  }
68  void split_child(Node *parent, int index, Node *child) {
69      Node *new_child = create_node(child->is_leaf);
70      parent->children[index + 1] = new_child;
71      parent->keys[index] = child->keys[1];
72      parent->num_keys++;
```

[*] Untitled1

```
71      parent->keys[index] = child->keys[1];
72      parent->num_keys++;
73      new_child->num_keys = 1;
74      child->num_keys = 1;
75      for (int i = 0; i < 2; i++) {
76          new_child->keys[i] = child->keys[i + 2];
77          new_child->children[i] = child->children[i + 2];
78      }
79      new_child->children[2] = child->children[4];
80  }
81  void insert_non_full(Node *node, int key) {
82      int i = node->num_keys - 1;
83      if (node->is_leaf) {
84          while (i >= 0 && key < node->keys[i]) {
85              i--;
86          i++;
87          if (node->children[i]->num_keys == MAX_KEYS) {
88              split_child(node, i, node->children[i]);
89              if (key > node->keys[i]) {
90                  i++;
91              }
92          }
93          insert_non_full(node->children[i], key);
94      }
95  }
96  void free_tree(Node *root) {
97      if (root != NULL) {
98          if (!root->is_leaf) {
99              for (int i = 0; i <= root->num_keys; i++) {
100                 free_tree(root->children[i]);
101             }
102         }
103         free(root);
104     }
105 }
106
```

[*] Untitled1

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_KEYS 4
#define MIN_KEYS 2
#define MAX_CHILDREN (MAX_KEYS + 1)
#define MIN_CHILDREN (MIN_KEYS + 1)
typedef struct Node {
    int keys[MAX_KEYS];
    struct Node *children[MAX_CHILDREN];
    int num_keys;
    int is_leaf;
} Node;
Node* create_node(int is_leaf);
void traverse_tree(Node *root);
Node* insert(Node *root, int key);
void split_child(Node *parent, int index, Node *child);
void insert_non_full(Node *node, int key);
void free_tree(Node *root);
int main() {
    Node *root = create_node(1);
    int keys[] = {10, 20, 5, 6, 15, 30, 25, 40, 50, 45, 55, 60, 70, 80};
    int num_keys = sizeof(keys) / sizeof(keys[0]);
    for (int i = 0; i < num_keys; i++) {
        root = insert(root, keys[i]);
    printf("Tree traversal:\n");
    traverse_tree(root);
    free_tree(root)
    return 0;
Node* create_node(int is_leaf) {
    Node *new_node = (Node*)malloc(sizeof(Node));
    new_node->is_leaf = is_leaf;
    new_node->num_keys = 0;
    for (int i = 0; i < MAX_CHILDREN; i++) {
        new_node->children[i] = NULL;
    }
    return new_node;
```

[*] Untitled1

```
37  }
38  void traverse_tree(Node *root) {
39      if (root != NULL) {
40          int i;
41          for (i = 0; i < root->num_keys; i++) {
42              if (!root->is_leaf) {
43                  traverse_tree(root->children[i]);
44              }
45              printf("%d ", root->keys[i]);
46          }
47          if (!root->is_leaf) {
48              traverse_tree(root->children[i]);
49          }
50      }
51  }
52  Node* insert(Node *root, int key)
53      if (root->num_keys == MAX_KEYS) {
54          Node *new_root = create_node(0);
55          new_root->children[0] = root;
56          split_child(new_root, 0, root);
57          insert_non_full(new_root, key);
58          return new_root;
59      } else {
60          insert_non_full(root, key);
61          return root;
62      }
63  }
64  void split_child(Node *parent, int index, Node *child) {
65      Node *new_child = create_node(child->is_leaf);
66      parent->children[index + 1] = new_child;
67      parent->keys[index] = child->keys[2];
68      parent->num_keys++;
69      new_child->num_keys = 2;
70      child->num_keys = 2;
71      for (int i = 0; i < 2; i++) {
72          new_child->keys[i] = child->keys[i + 3];
```

[*] Untitled1

```
73          new_child->children[i] = child->children[i + 3];
74      }
75      new_child->children[2] = child->children[5];
76  }
77  void insert_non_full(Node *node, int key) {
78      int i = node->num_keys - 1;
79      if (node->is_leaf) {
80          while (i >= 0 && key < node->keys[i]) {
81              node->keys[i + 1] = node->keys[i];
82              i--;
83          }
84          node->keys[i + 1] = key;
85          node->num_keys++;
86      } else {
87          while (i >= 0 && key < node->keys[i]) {
88              i--;
89          }
90          i++;
91          if (node->children[i]->num_keys == MAX_KEYS) {
92              split_child(node, i, node->children[i]);
93              if (key > node->keys[i]) {
94                  i++;
95              }
96          }
97          insert_non_full(node->children[i], key);
98      }
99  }
100 void free_tree(Node *root) {
101     if (root != NULL) {
102         if (!root->is_leaf) {
103             for (int i = 0; i <= root->num_keys; i++) {
104                 free_tree(root->children[i]);
105             }
106         }
107         free(root);
108     }
```