

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <limits.h>
4
5 #define SKIPLIST_MAX_LEVEL 6
6
7 typedef struct snode {
8     int key;
9     int value;
10    struct snode **forward;
11 } snode;
12
13 typedef struct skiplist {
14     int level;
15     int size;
16     struct snode *header;
17 } skiplist;
18
19 skiplist *skiplist_init(skiplist *list) {
20     int i;
21     snode *header = (snode *) malloc(sizeof(struct snode));
22     list->header = header;
23     header->key = INT_MAX;
24     header->forward = (snode **) malloc(
25         sizeof(snode*) * (SKIPLIST_MAX_LEVEL + 1));
26     for (i = 0; i <= SKIPLIST_MAX_LEVEL; i++) {
27         header->forward[i] = list->header;
28     }
29
30     list->level = 1;
31     list->size = 0;
32
33     return list;
34 }
35
36 static int rand_level() {
37     int level = 1;
38     while (rand() < RAND_MAX / 2 && level < SKIPLIST_MAX_LEVEL)
39         level++;
40     return level;
41 }
42
43 int skiplist_insert(skiplist *list, int key, int value) {
44     snode *update[SKIPLIST_MAX_LEVEL + 1];
45     snode *x = list->header;
46     int i, level;
47     for (i = list->level; i >= 1; i--) {
48         while (x->forward[i]->key < key)
49             x = x->forward[i];
50         update[i] = x;
51     }
52     x = x->forward[1];
53
54     if (key == x->key) {
55         x->value = value;
56         return 0;
57     } else {
58         level = rand_level();
59         if (level > list->level) {
60             for (i = list->level + 1; i <= level; i++) {
61                 update[i] = list->header;
62             }
63             list->level = level;
64         }
65
66         x = (snode *) malloc(sizeof(snode));
67         x->key = key;
68         x->value = value;
69         x->forward = (snode **) malloc(sizeof(snode*) * (level + 1));
70         for (i = 1; i <= level; i++) {
71             x->forward[i] = update[i]->forward[i];
72             update[i]->forward[i] = x;
73         }
74     }
75     return 0;
76 }
77
78 snode *skiplist_search(skiplist *list, int key) {
79     snode *x = list->header;
80     int i;
81     for (i = list->level; i >= 1; i--) {
82         while (x->forward[i]->key < key)
83             x = x->forward[i];
84     }
85     if (x->forward[1]->key == key) {
86         return x->forward[1];
87     } else {
88         return NULL;
89     }
90     return NULL;
91 }
92
93 static void skiplist_node_free(snode *x) {
94     if (x) {
95         free(x->forward);
96         free(x);
97     }
98 }
99
100 int skiplist_delete(skiplist *list, int key) {
101     int i;
102     snode *update[SKIPLIST_MAX_LEVEL + 1];
103     snode *x = list->header;
104     for (i = list->level; i >= 1; i--) {
105         while (x->forward[i]->key < key)
106             x = x->forward[i];
107         update[i] = x;
108     }
109
110     x = x->forward[1];
111     if (x->key == key) {
112         for (i = 1; i <= list->level; i++) {
113             if (update[i]->forward[i] != x)
114                 break;
115             update[i]->forward[i] = x->forward[i];
116         }
117         skiplist_node_free(x);
118
119         while (list->level > 1 && list->header->forward[list->level]
120             == list->header)
121             list->level--;
122         return 0;
123     }
124     return 1;
125 }
126
127 static void skiplist_dump(skiplist *list) {
128     snode *x = list->header;
129     while (x && x->forward[1] != list->header) {
130         printf("%d[%d]->", x->forward[1]->key, x->forward[1]->value);
131         x = x->forward[1];
132     }
133     printf("NIL\n");
134 }
135
136 int main() {
137     int arr[] = { 3, 6, 9, 2, 11, 1, 4 }, i;
138     skiplist list;
139     skiplist_init(&list);
140
141     printf("Insert:-----\n");
142     for (i = 0; i < sizeof(arr) / sizeof(arr[0]); i++) {
143         skiplist_insert(&list, arr[i], arr[i]);
144     }
145     skiplist_dump(&list);
146
147     printf("Search:-----\n");
148     int keys[] = { 3, 4, 7, 10, 11 };
149
150     for (i = 0; i < sizeof(keys) / sizeof(keys[0]); i++) {
151         snode *x = skiplist_search(&list, keys[i]);
152         if (x) {
153             printf("key = %d, value = %d\n", keys[i], x->value);
154         } else {
155             printf("key = %d, not found\n", keys[i]);
156         }
157     }
158
159     printf("Search:-----\n");
160     skiplist_delete(&list, 3);
161     skiplist_delete(&list, 9);
162     skiplist_dump(&list);
163
164     return 0;
165 }

```

x Terminal

Insert:-----

1[1]->2[2]->3[3]->4[4]->6[6]->9[9]->11[11]->NIL

Search:-----

key = 3, value = 3

key = 4, value = 4

key = 7, not found

key = 10, not found

key = 111, not found

Search:-----

1[1]->2[2]->4[4]->6[6]->11[11]->NIL