

Leetcode minstack

Problem List < > 🔍

Description Editorial Solutions Submissions

155. Min Stack Solved

Medium Topics Companies Hint

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with $O(1)$ time complexity for each function.

Example 1:

Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[-3],[],[-3]]

Output
[null,null,null,null,-3,null,0,-2]

Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top(); // return 0
minStack.getMin(); // return -2

Constraints:

- $-2^{31} \leq val \leq 2^{31} - 1$

13.5K 81 ☆

Code

C Auto

```
1
2
3
4 typedef struct {
5     int size;
6     int top;
7     int *s;
8     int *minstack;
9
10
11 } MinStack;
12
13
14 MinStack* minStackCreate() {
15     MinStack *st=(MinStack*) malloc(sizeof(MinStack));
16     if(st==NULL)
17     {
18         printf("memory allocation failed");
19         exit(0);
20     }
21     st->size=5;
22     st->top=-1;
23     st->s=(int*) malloc (st->size*sizeof(int));
24     st->minstack = (int*) malloc (st->size * sizeof(int));
25     if(st->s==NULL)
26     {
27         printf("memory allocation failed");
28         free(st->s);
29         free(st->minstack);
30         exit(0);
31     }
32     return st;
33 }
34
35 void minStackPush(MinStack* obj, int val) {
36     if(obj->top==obj->size-1)
37     {
38         printf("stack is overflow");
39     }
40     else{
41         obj->top++;
42         obj->s[obj->top]=val;
43         if (obj->top == 0 || val < obj->minstack[obj->top - 1]) {
44             obj->minstack[obj->top] = val;
45         } else {
46
47
```

Saved to local

Testcase Test Result

Ln 27, Col 44

Problem List

DescriptionEditorialSolutionsSubmissions

155. Min Stack

Solved

MediumTopicsCompaniesHint

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

`MinStack()` initializes the stack object.

`void push(int val)` pushes the element `val` onto the stack.

`void pop()` removes the element on the top of the stack.

`int top()` gets the top element of the stack.

`int getMin()` retrieves the minimum element in the stack.

You must implement a solution with $O(1)$ time complexity for each function.

Example 1:

Input

`["MinStack","push","push","push","getMin","pop","top","getMin"]`
`[[[]],[-2],[-3],[-3],[[],[],[]]]`

Output

`[null,null,null,null,-3,null,0,-2]`

Explanation

`MinStack minStack = new MinStack();`
`minStack.push(-2);`
`minStack.push(-3);`
`minStack.push(-3);`
`minStack.getMin();` // return -3
`minStack.pop();`
`minStack.top();` // return 0
`minStack.getMin();` // return -2

Constraints:

- $-2^{31} \leq val \leq 2^{31} - 1$

13.5K81

Code

C

Auto

```
32 return st;
33 }
34 }
35
36 void minStackPush(MinStack* obj, int val) {
37     if(obj->top==obj->size-1)
38     {
39         printf("stack is overflow");
40     }
41     else{
42         obj->top++;
43         obj->xs[obj->top]=val;
44         if (obj->top == 0 || val < obj->minstack[obj->top - 1]) {
45             obj->minstack[obj->top] = val;
46         } else {
47             obj->minstack[obj->top] = obj->minstack[obj->top - 1];
48         }
49     }
50 }
51
52 }
53
54 void minStackPop(MinStack* obj) {
55     int value;
56     if(obj->top==--1)
57     {
58         printf("underflow");
59     }
60     else
61     {
62         value=obj->xs[obj->top];
63         obj->top--;
64         printf("%d is popped\n",value);
65     }
66 }
67
68 }
69 }
70
71 int minStackTop(MinStack* obj) {
72     int value=-1;
73     if(obj->top==--1)
74     {
```

Saved to local

Ln 1, Col 1

Testcase

Test Result

Accepted Runtime: 0 ms

Case 1

Problem List

Description

Editorial

Solutions

Submissions

155. Min Stack

Solved

MediumTopicsCompaniesHint

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with $O(1)$ time complexity for each function.

Example 1:

Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[[],[-2],[0],[-3],[],[],[],[]]]

Output
[null,null,null,null,-3,null,0,-2]

Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top(); // return 0
minStack.getMin(); // return -2

Constraints:

- $-2^{31} \leq val \leq 2^{31} - 1$

13.5K81

Code

```
69 }
70
71 int minStackTop(MinStack* obj) {
72     int value=-1;
73     if(obj->top==-1)
74     {
75         printf("underflow\n");
76         exit(0);
77     }
78 }
79 else
80 {
81     value=obj->s[obj->top];
82     return value;
83 }
84 }
85
86
87
88 }
89
90 int minStackGetMin(MinStack* obj) {
91
92
93     if(obj->top==-1)
94     {
95         printf("underflow\n");
96         exit(0);
97     }
98 }
99 else
100 {
101     return obj->minstack[obj->top];
102 }
103 }
104
105 }
106
107 void minStackFree(MinStack* obj) {
108     free(obj->s);
109     free(obj->minstack);
110     free(obj);
111 }
112
113
114 /**
115  * Your MinStack struct will be instantiated and called as such:
116  */
117 }
```

TestcaseTest Result

Output

Problem List

Description

Editorial

Solutions

Submissions

155. Min Stack

Solved

MediumTopicsCompaniesHint

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with $O(1)$ time complexity for each function.

Example 1:

Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[[],[-2],[0],[-3],[],[],[],[]]]

Output
[null,null,null,null,-3,null,0,-2]

Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top(); // return 0
minStack.getMin(); // return -2

Constraints:

- $-2^{31} \leq val \leq 2^{31} - 1$

13.5K81

Code

```
108 free(obj->s);
109 free(obj->minstack);
110 free(obj);
111 }
112 }
113
114 /**
115  * Your MinStack struct will be instantiated and called as such:
116  * MinStack* obj = minStackCreate();
117  * minStackPush(obj, val);
118  * minStackPop(obj);
119  * minStackTop(obj);
120  * int param_3 = minStackTop(obj);
121  * int param_4 = minStackGetMin(obj);
122  */
123 }
```

TestcaseTest Result

Accepted

Runtime: 2 ms

Case 1

Input

["MinStack","push","push","push","getMin","pop","top","getMin"]

[[[],[-2],[0],[-3],[],[],[],[]]]

Stdout

-3 is popped

Output

[null,null,null,null,-3,null,0,-2]

Expected

[null,null,null,null,-3,null,0,-2]

Contribute a testcase

Code

```
typedef struct {

    int size;
    int top;
    int *s;
    int *minstack;

} MinStack;

MinStack* minStackCreate() {
    MinStack *st=(MinStack*) malloc(sizeof(MinStack));
    if(st==NULL)
    {
        printf("memory allocation failed");
        exit(0);
    }
    st->size=5;
    st->top=-1;
    st->s=(int*) malloc (st->size*sizeof(int));
    st->minstack = (int*) malloc (st->size * sizeof(int));
    if(st->s==NULL)
    {
        printf("memory allocation failed");
        free(st->s);
        free(st->minstack);
        exit(0);
    }
    return st;
}

void minStackPush(MinStack* obj, int val) {
    if(obj->top==obj->size-1)
    {
        printf("stack is overflow");

    }
    else{
        obj->top++;
        obj->s[obj->top]=val;
        if (obj->top == 0 || val < obj->minstack[obj->top - 1]) {
            obj->minstack[obj->top] = val;
        } else {
            obj->minstack[obj->top] = obj->minstack[obj->top - 1];
        }
    }
}
```

```

void minStackPop(MinStack* obj) {
    int value;
    if(obj->top== -1)
    {
        printf("underflow");

    }
    else
    {
        value=obj->s[obj->top];
        obj->top--;
        printf("%d is popped\n",value);
    }
}

int minStackTop(MinStack* obj) {
    int value=-1;
    if(obj->top== -1)
    {
        printf("underflow\n");
        exit(0);

    }
    else
    {
        value=obj->s[obj->top];
        return value;
    }
}

int minStackGetMin(MinStack* obj) {
    if(obj->top== -1)
    {
        printf("underflow\n");
        exit(0);

    }
    else
    {
        return obj->minstack[obj->top];
    }
}

void minStackFree(MinStack* obj) {
    free(obj->s);
    free(obj->minstack);
}

```

```
    free(obj);  
  
}  
  
/**  
 * Your MinStack struct will be instantiated and called as such:  
 * MinStack* obj = minStackCreate();  
 * minStackPush(obj, val);  
  
 * minStackPop(obj);  
  
 * int param_3 = minStackTop(obj);  
  
 * int param_4 = minStackGetMin(obj);  
  
 * minStackFree(obj);  
 */
```