

## Lab2

### Multi-Choices

1. A, 2.B, 3.C

### True or False

1. T, 2.T, 3.F

### Evaluation

7.

```
slidev_to_pdf.py lab2_补测.py > ...
E: > 2025fall > DS > lab2_补测.py > ...
1 import time
2 import timeit
3
4 # 1. 准备数据
5 N = 100000
6 name = "ZaiZhang"
7 print(f"Generating data with N={N}...")
8
9 lst = [(i, i) for i in range(N)]
10 d = dict(lst)
11
12 lst.append((name, 42))
13 d[name] = 42
14

15 # 定义测试函数
16 def search_in_list():
17     for j, k in lst:
18         if j == name:
19             return k
20     return None
21
22 def search_in_dict():
23     # 哈希
24     return d[name]
25
26 # 3. 运行 1000 次取平均
27 print("Running benchmarks...")
28 t_list = timeit.timeit(search_in_list, number=1000)
29 t_dict = timeit.timeit(search_in_dict, number=1000)
30
31 print(f"List search time (1000 runs): {t_list:.5f} seconds")
32 print(f"Dict search time (1000 runs): {t_dict:.5f} seconds")
33
34 # 4. 计算差异
35 if t_dict > 0:
36     print(f"Conclusion: Dict is {t_list / t_dict:.2f} times faster than List.")
37 else:
38     print("Conclusion: Dict is infinitely faster (time too small to measure).")
```

```
PS C:\Users\zhangzai> & C:/Users/zhangzai/AppData/Local/Programs/Python/Python311/python.exe e:/2025fall/DS/lab2_补测.py
Generating data with N=100000...
Running benchmarks...
List search time (1000 runs): 1.96597 seconds
Dict search time (1000 runs): 0.00005 seconds
Conclusion: Dict is 41215.24 times faster than List.
PS C:\Users\zhangzai>
```

8.

The screenshot shows a code editor with two tabs: 'slidev\_to\_pdf.py' and 'lab2\_补测.py'. The current file is 'lab2\_补测.py'. The code implements a Binary Search Tree (BST) with Node and BST classes. The Node class has an \_\_init\_\_ method that initializes a key and left/right pointers. The BST class has an \_\_init\_\_ method that initializes the root pointer to None. The get method returns the value of a node with a specific key. The \_get method is a recursive helper that returns None if the node is None, or the node's key if it matches the target key, or the result of a recursive call on the left or right child if the target key is less than or greater than the node's key respectively.

```
E: > 2025fall > DS > lab2_补测.py > ...
1  import random
2  import sys
3  import heapq
4  import timeit
5
6  # 增加递归深度限制, 防止深度过大的 BST 导致报错
7  sys.setrecursionlimit(200000)
8
9  # Class Definitions (Fixed from PDF)
10
11 class Node:
12     def __init__(self, key, left=None, right=None):
13         self.key = key
14         self.left = left
15         self.right = right
16
17 class BST:
18     def __init__(self):
19         self._root = None
20
21     def get(self, key):
22         return self._get(self._root, key)
23
24     def _get(self, x, key):
25         if x is None:
26             return None
27         if key == x.key:
28             return x.key
29         elif key < x.key:
30             return self._get(x.left, key)
31         else:
32             return self._get(x.right, key)
```

```
34     # 插入元素
35     def put(self, key):
36         self._root = self._put(self._root, key)
37
38     def _put(self, x, key):
39         if x is None:
40             return Node(key)
41         if key < x.key:
42             x.left = self._put(x.left, key)
43         elif key > x.key:
44             x.right = self._put(x.right, key)
45         return x
46
47     # 计算树的高度
48     def height(self):
49         return self._height(self._root)
50
51     def _height(self, x):
52         if x is None:
53             return 0
54         return 1 + max(self._height(x.left), self._height(x.right))
55
56 class MaxPQ:
57     def __init__(self):
58         self._pq = []
59
60     def insert(self, key):
61         # 存储 -key 来模拟最大堆
62         heapq.heappush(self._pq, -key)
63
64     def insert(self, key):
65         # 存储 -key 来模拟最大堆
66         heapq.heappush(self._pq, -key)
67
68     def contains(self, key):
69         return -key in self._pq
70
71     # 获取最大值 (堆顶)
72     def get_max(self):
73         if not self._pq: return None
74         return -self._pq[0]
75
76     # Experiment Setup
77
78     my_id = 42453034
79     N = 100000
80     print(f"Building data structures with N={N} random integers...")
81
82     bst = BST()
83     pq = MaxPQ()
84
85     # 1. 插入数据
86     for item in lst:
87         bst.put(item)
88         pq.insert(item)
```

```
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
```

```
89 # 2. 插入目标最大值 (my_id)
90 # my_id 远大于 range(100000), 所以它一定是最大值
91 bst.put(my_id)
92 pq.insert(my_id)
93
94 print("Data structures built. Starting performance test...")
95
96 # 3. 定义测试操作
97 def get_max_from_bst():
98     bst.get(my_id)
99
100 def get_max_from_pq():
101     # 在 MaxPQ 中, 最大值就在数组索引 0 的位置
102     pq.get_max()
103
104 # 4. 执行测试
105 t_bst = timeit.timeit(get_max_from_bst, number=1000)
106 t_pq = timeit.timeit(get_max_from_pq, number=1000)
107
108 print(f"BST get_max time (1000 runs): {t_bst:.6f} seconds")
109 print(f"PQ get_max time (1000 runs): {t_pq:.6f} seconds")
110
111 if t_pq > 0:
112     print(f"Conclusion: MaxPQ is {t_bst / t_pq:.2f} times faster than BST.")
113
114 # 回答 height
115 print(f"The height of the BST is: {bst.height()}")
```

```
PS C:\Users\zhangzai> & C:/Users/zhangzai/AppData/Local/Programs/Python/Python311/python.exe e:/2025fall/DS/lab2_补测.py
Building data structures with N=100000 random integers...
Data structures built. Starting performance test...
BST get_max time (1000 runs): 0.000531 seconds
PQ get_max time (1000 runs): 0.000058 seconds
Conclusion: MaxPQ is 9.20 times faster than BST.
The height of the BST is: 40
```