Readme5

5_1

程序运行

```
python 5_1.py
```

线性搜索定义如下

函数返回值为key在lst中的索引值，若值不存在返回-1

```
def linear_contains(lst, key):
    for idx in range(len(lst)):
        if lst[idx] == key:
            return idx
    return -1
```

二分搜索定义如下

```
def binary_contains(lst, key):
    low = 0
    high = len(lst) - 1
    while low <= high:
        # while there is still a search space
        mid = (low + high) // 2  # 向下取整
        if lst[mid] < key:
            low = mid + 1
        elif lst[mid] > key:
            high = mid - 1
        else:
            return mid
    return -1
```

测试函数如下

整个测试list长度为1000000个元素，其中元素已经有序排列，按照升序排列

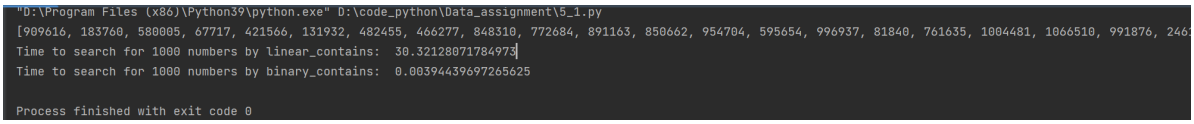
在其中检索1000个元素，待检索元素[0,1100000)中随机取值，也就是说存在检索失败情况，统计1000个元素的总耗时

```
def sort_performance():
    # lst = [random.randint(0, 1000) for i in range(10)]
    position1 = []
    position2 = []
    data_lst = list(range(1000000))
    key_lst = [random.randint(0, 1100000) for i in range(1000)]  # 在[0,1100000)
中随机取值，即存在检索失败情况
    print(key_lst)
    start_time1 = time.time()
    for value in key_lst:
        pos = linear_contains(data_lst, value)
        # position1.append(pos)
```

```
    end_time1 = time.time()

    start_time2 = time.time()
    for value in key_lst:
        pos = binary_contains(data_lst, value)
        # position2.append(pos)
    end_time2 = time.time()
    # print(position1)
    # print(position2)
    # if position1 == position2:
    print("Time to search for 1000 numbers by linear_contains: ", end_time1-
start_time1)
    print("Time to search for 1000 numbers by binary_contains: ", end_time2-
start_time2)



if __name__ == '__main__':
    sort_performance()
```

结果如下

随机选取1000个数的测试结果显示，在大的数据集中，二分搜索的搜索时长和搜索性能（0.0039s）显著优于线性搜索性能（30.3212s）



5_2

程序运行

```
python maze.py
```

以课程中的generic_search.py和maze.py代码为基础进行修改，增加状态计数

主要修改`generic_search.py`中的`dfs`，`bfs`，`astar`函数，在遍历开始时，初始化计数变量count，每一次遍历一个frontier中的结点时，状态发生一次改变，计数器+1

```
def dfs(initial: T, goal_test: Callable[[T], bool], successors: Callable[[T],
List[T]]) -> Optional[Node[T], int]:
    # frontier is where we've yet to go
    frontier: Stack[Node[T]] = Stack()
    frontier.push(Node(initial, None))
    # explored is where we've been
    explored: Set[T] = {initial}
    count = 0  # syzedit

    # keep going while there is more to explore
    while not frontier.empty:
        current_node: Node[T] = frontier.pop()
        count += 1  # syzedit 增加计数器
        current_state: T = current_node.state
        # if we found the goal, we're done
        if goal_test(current_state):
```

```
            return current_node, count
        # check where we can go next and haven't explored
        for child in successors(current_state):
            if child in explored:  # skip children we already explored
                continue
            explored.add(child)
            frontier.push(Node(child, current_node))
    return None, count  # went through everything and never found goal


def bfs(initial: T, goal_test: Callable[[T], bool], successors: Callable[[T],
List[T]]) -> Optional[Node[T], int]:
    # frontier is where we've yet to go
    frontier: Queue[Node[T]] = Queue()
    frontier.push(Node(initial, None))
    # explored is where we've been
    explored: Set[T] = {initial}
    count = 0  # syzedit

    # keep going while there is more to explore
    while not frontier.empty:
        current_node: Node[T] = frontier.pop()
        count += 1  # syzedit 增加计数器
        current_state: T = current_node.state
        # if we found the goal, we're done
        if goal_test(current_state):
            return current_node, count
        # check where we can go next and haven't explored
        for child in successors(current_state):
            if child in explored:  # skip children we already explored
                continue
            explored.add(child)
            frontier.push(Node(child, current_node))
    return None, count  # went through everything and never found goal


def astar(initial: T, goal_test: Callable[[T], bool], successors: Callable[[T],
List[T]],
          heuristic: Callable[[T], float]) -> Optional[Node[T], int]:
    # frontier is where we've yet to go
    frontier: PriorityQueue[Node[T]] = PriorityQueue()
    frontier.push(Node(initial, None, 0.0, heuristic(initial)))
    # explored is where we've been
    explored: Dict[T, float] = {initial: 0.0}
    count = 0  # syzedit

    # keep going while there is more to explore
    while not frontier.empty:
        current_node: Node[T] = frontier.pop()
        count += 1
        current_state: T = current_node.state
        # if we found the goal, we're done
        if goal_test(current_state):
            return current_node, count
        # check where we can go next and haven't explored
        for child in successors(current_state):
            new_cost: float = current_node.cost + 1  # 1 assumes a grid, need a
cost function for more sophisticated apps
```

```
            if child not in explored or explored[child] > new_cost:
                explored[child] = new_cost
                frontier.push(Node(child, current_node, new_cost,
heuristic(child)))
    return None, count  # went through everything and never found goal
```

在`maze.py`中的测试函数如下

```
def method_performance():
    dfs_tol_cnt = 0
    bfs_tol_cnt = 0
    astar_tol_cnt = 0
    for i in range(100):
        # Test DFS
        m: Maze = Maze()
        # print(m)
        solution1, dfs_cnt = dfs(m.start, m.goal_test, m.successors)
        dfs_tol_cnt += dfs_cnt
        # Test BFS
        solution2, bfs_cnt = bfs(m.start, m.goal_test, m.successors)
        bfs_tol_cnt += bfs_cnt
        # Test A*
        # distance, astar_cnt = manhattan_distance(m.goal)
        distance: Callable[[MazeLocation], float] = manhattan_distance(m.goal)
        solution3, astar_cnt = astar(m.start, m.goal_test, m.successors,
distance)
        astar_tol_cnt += astar_cnt
    print("dfs_tol_cnt: ", dfs_tol_cnt)
    print("bfs_tol_cnt: ", bfs_tol_cnt)
    print("astar_tol_cnt: ", astar_tol_cnt)


if __name__ == "__main__":
    method_performance()
```

计数结果如下

在100个随机生成的10*10迷宫中，广度优先搜索的状态最多，搜索最慢，A*方法和深度优先的状态数量相近

```
"D:\Program Files (x86)\Python39\python.exe" D:\code_python\Data_assignment\maze.py
dfs_tol_cnt:  5112
bfs_tol_cnt:  7391
astar_tol_cnt:  5269

Process finished with exit code 0
```