Algorithm Experiment
Longest Subsequence

李胜锐 2017012066

2020 年 3 月 27 日

Date Performed: 2020 年 3 月 27 日

1 Experiment Environment

Operating System: macOS 10.15.2
Processor: 2.6 GHz 6-Core Intel Core i7
Memory: 16GB
Language: python 3.7
IDE: PyCharm

2 Algorithm Analysis

2.1 Binary Search

```
def bisect_left(a, x, lo=0, hi=None):
    if lo < 0:
        raise ValueError('lo must be non-negative')
    if hi is None:
        hi = len(a)
    while lo < hi:
        mid = (lo + hi) // 2
        if a[mid] < x:
            lo = mid + 1
        else:
            hi = mid
    return lo
```

The simple binary search algorithm.
With a input array and a target number, the algorithm can find **the largest number in the array that less or equal to the target**, and return its location.
Hence, the time complexity of one search in the whole array of size n will be:

$$T(n) = \Theta(\log n)$$

2.2 Get Longest Subsequence

```
def get_longest_substr(whole_arr):
    least_last = [math.inf for _ in range(len(whole_arr) + 1)]
    longest_record = [[] for _ in range(len(whole_arr))]
```

```python
    for num in whole_arr:
        loc = bisect_left(least_last, num)
        least_last[loc] = num
        longest_record[loc].clear()
        longest_record[loc] += longest_record[loc - 1][:] if loc > 0 else []
        longest_record[loc].append(num)
    for index, number in enumerate(least_last):
        if least_last[index + 1] == math.inf:  # the last number
            return longest_record[index]
```

We have three data structures here:

1) **whole_arr**: array of input, denoted as **W**.
2) **least_last**: array of the least last number of different length, denoted as **L**.
3) **longest_record**: array of array, record the whole subsequence with different length, denoted as **R**.

The algorithm is descripted as followed:

1) Initialize every elements in **L** as ∞
2) Iterate every number **m** of **W** in order.
3) Search in **L**: For each **m**, use binary search to find the appropriate location of **m** in **L**.
4) Modify **L**: Replace the element in the searched location with **m**.
5) Modify **R**: If the location of **m** is $\mathbf{L}[0]$, let $\mathbf{R}[0] = [m]$. Else, if the location is i, then $i-1$ must have been handled before. Thus, let $\mathbf{R}[i] = \mathbf{R}[i-1].append(\mathbf{m})$.
6) Get result: Find the largest index l of modified elements in **L**, in other word, the location of the last element in **L** that is not ∞. Thus, $l+1$ means the lenghth of the lognest subsequence. And $\mathbf{R}[l]$ is exactly this subsequence.

Each iteration needs a binary search, and it requires n iterations:

$$T(n) = \Theta(n \log n)$$

3 Result Analysis

3.1 Time complexity

表 1: Time complexity

n	Time(s)
10	6.509e-05
100	0.0003771
1000	0.004551
10000	0.05056
100000	0.4987
1000000	12.28

1) Obviously the time complexity is less than $\Theta(n^2)$ and is larger than $\Theta(n)$.

2) Thus, the result $T(n) = \Theta(n \log n)$ is appropriate.

4 Instruction

This test process have three main functions:

1) test – type in your own array divided by blank space and find the longest monotonically increasing subsequence of it

2) rand – generate a random array and find the longest monotonically increasing subsequence of it

3) time – compute the time cost of this method

Follow the instruction. And it is easy to conduct the experiment with it.