EE 3980 Algorithms

Homework 2. Random Data Searches

105061110 周柏宇

2020/3/22

1. Introduction

In this homework, we implemented 3 search algorithms: linear search, bidirection search and random-direction search. To evaluate the average and worse-case performance, we run each algorithm on lists of randomly-ordered English words with different numbers of entries. In the end, we plot out the CPU time against the input size to observe the trend and to verify our analysis of the complexity of these algorithms.

2. Analysis & Implementation

2.1 Linear search

```
1.  Algorithm Search(word, list, n)         // Linear Search
2.  {
3.      for i := 1 to n do {                // compare all possible entries
4.          if (list[i] = word) return i;
5.      }
6.      return -1;                          // unsuccessful search
7.  }
```

In the linear search, we compare the entry of the list from head to the end. The

best case occurs when the target word is the very first word of the list, while the worst case occurs when the target word is the last word of the list. If we search all words in the list, on average, it requires $\frac{1}{n}\sum_{i=1}^{n} i = \frac{n+1}{2}$ times of comparison. As for the space complexity, we need *n* space for the list.

Best-case time complexity: $\mathcal{O}(1)$

Average time complexity: $\mathcal{O}(n)$

Worst-case time complexity: $\mathcal{O}(n)$

Space complexity: $\mathcal{O}(n)$

2.2 Bidirection search

```
1.  Algorithm BDSearch(word, list, n)       // Bidirection Search
2.  {
3.      for i := 1 to n / 2 do {            // compare entries from both sides
4.          if (list[i] = word) return i;
5.          if (list[n - i - 1] = word) return n - i - 1;
6.      }
7.      return -1;                          // unsuccessful search
8.  }
```

In bidirection search, we compare the entries of the list from both directions. Therefore, the best case occurs when the target word is the first word of the list; the worst case occurs when the target word is in the center of the list. Therefore, we need to compare *n* times before finding it. The average performance is the same as the linear search when we search all words of the list, because we only change the

searching order.

Best-case time complexity: $\mathcal{O}(1)$

Average time complexity: $\mathcal{O}(n)$

Worst-case time complexity: $\mathcal{O}(n)$

Space complexity: $\mathcal{O}(n)$

2.3 Random-direction search

```
1.  Algorithm RDSearch(word, list, n)
2.  {
3.      choose j from {0, 1} randomly;      // randomly select a direction
4.      if (j = 1) then                     // forward linear search
5.          for i := 1 to n do {
6.              if (list[i] = word) return i;
7.          }
8.      else                                // backward linear search
9.          for i := n to 1 do {
10.             if (list[i] = word) return i;
11.         }
12.     return -1;                          // unsuccessful search
13. }
```

In random-direction search, we first choose a direction and then perform the linear search either forward or backward according to the direction. Same as previous search algorithms, we need as least one and at most n comparisons to locate the target word. For the target word being the ith word in the list, assuming the direction is chosen uniformly at random, we expect to compare $0.5 * i + 0.5 * (n - i + 1) = \frac{n+1}{2}$ times.

Best-case time complexity: $\mathcal{O}(1)$

Average time complexity: $\mathcal{O}(n)$

Worst-case time complexity: $\mathcal{O}(n)$

Space complexity: $\mathcal{O}(n)$

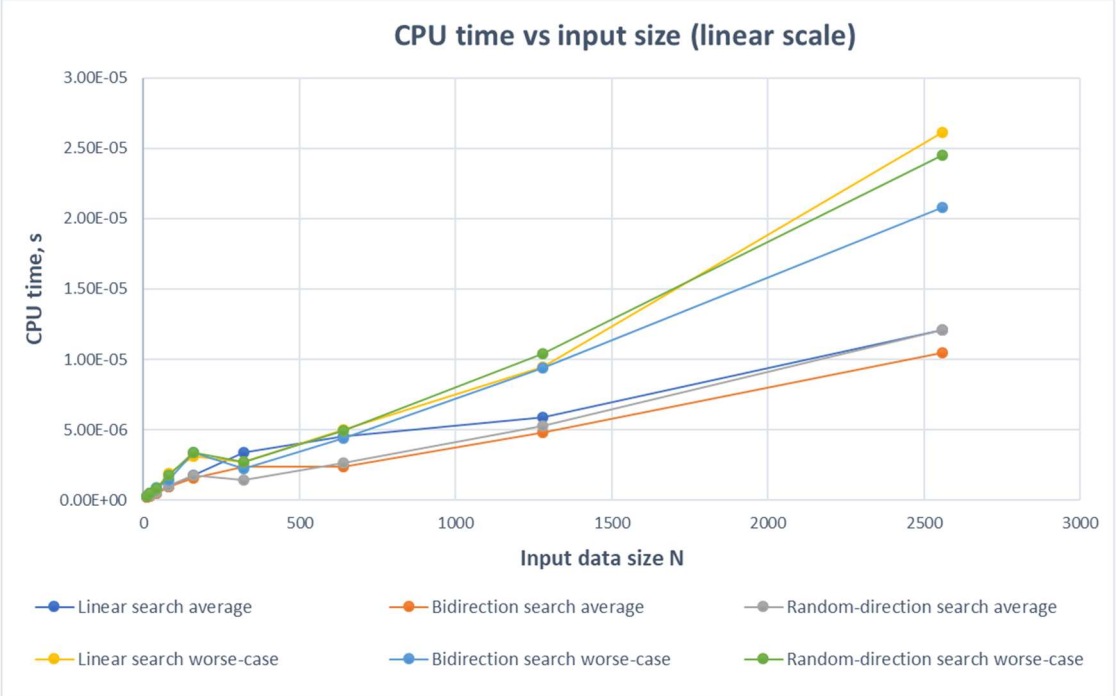
3. Result and Observation

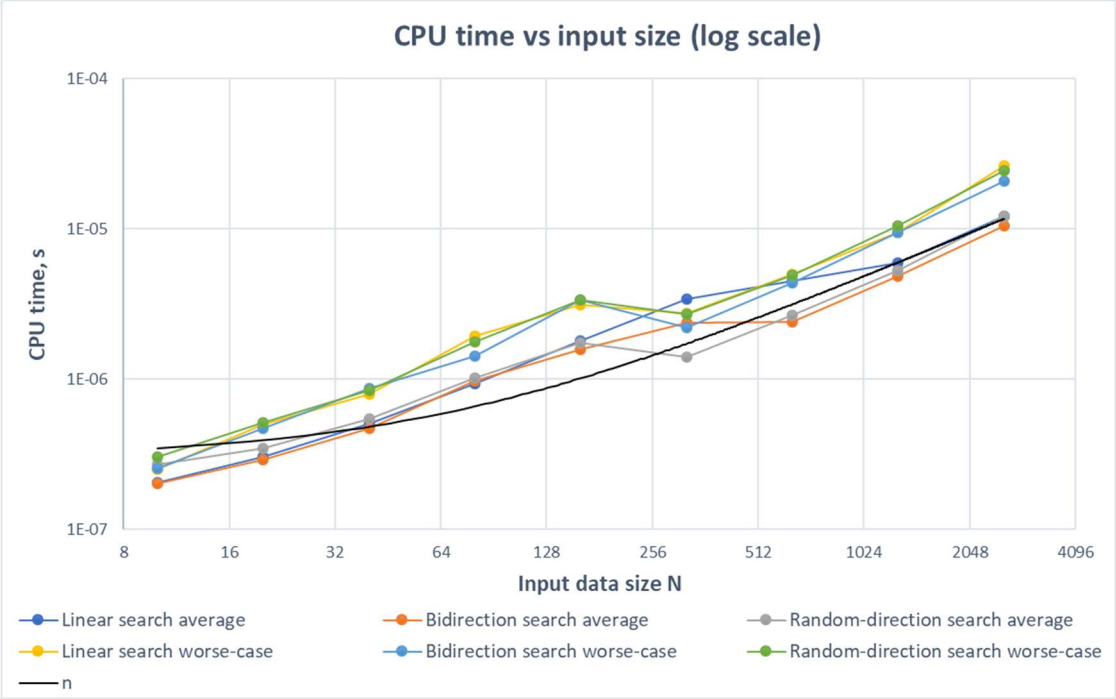
To measure the average performance, we search all possible words in the list and average the CPU time over the number of words as average performance. As for the worse-case performance, we choose to search the word that requires most comparisons for each search algorithm. To exhibit and compare the growth of CPU time, we run three search algorithms with different numbers of input for several times. Specifically, the average performance is averaged over 500 trials and the worse-case performance is averaged over 5000 trials.

N	*Average*			*Worse-case*		
	Linear search	*Bidirection search*	*Random-direction search*	*Linear search*	*Bidirection search*	*Random-direction search*
10	0.206375	0.200796	0.270987	0.250006	0.25382	0.304413
20	0.305009	0.292397	0.346279	0.501442	0.469017	0.512171
40	0.506604	0.473344	0.548053	0.80018	0.861788	0.837994

80	0.930452	0.97487	1.01422	1.93582	1.41358	1.78337
160	1.78876	1.56901	1.75746	3.11975	3.37939	3.3812
320	3.40464	2.36826	1.39842	2.74181	2.21658	2.68579
640	4.51523	2.40808	2.66012	4.99859	4.39959	4.93641
1280	5.89816	4.81463	5.28454	9.4604	9.413	10.4376
2560	12.0937	10.4677	12.0814	26.1076	20.804	24.4856

Table 1. CPU time [μ s] vs input data size N





According to our analysis, we expect the performance to be the same in terms of the average and worst case since the only difference between them is the order of searching, and it roughly holds. The bidirection search seems to be a little bit faster due to our implementation which requires less operation at loop comparison. We also expect the worse-case performance to be around two times slower than that of the average case. In Table 1, for data set with a larger number of words, we certainly can see the relationship.

Since the average and worse-case time complexity are $\mathcal{O}(n)$ for all three search algorithms, in the linear scale scatter plot, we can observe the linear trend for all the lines if we ignore the fluctuation at small input sizes. In the log scale scatter plot, we can compare the slope with the linear line to show that they are indeed linear, and the difference between average and worse-case lines is only the bias.