Trees in C

Ying Pei Lin

Fall 2024

Zip Code Table

At `read_postcodes` function, we store the zip codes in the format of char and after we use the `strcmp` function to compare the zip codes in the search function. Down below is the code snippet of the search function.

```
area* linear_search_char(codes *postnr, const char *zip) {
  for (int i = 0; i < postnr->n; i++) {
    if (strcmp(postnr->areas[i].zip_char, zip) == 0) {
      return &postnr->areas[i];
    }
  }
  return NULL;
}

area* binary_search_char(codes *postnr, const char *zip) {
  int left = 0;
  int right = postnr->n - 1;

  while (left <= right) {
    int mid = (left + right) / 2;
    int cmp = strcmp(postnr->areas[mid].zip_char, zip);

    if (cmp == 0) {
      return &postnr->areas[mid];
    }
    if (cmp < 0) {
      left = mid + 1;
    } else {
      right = mid - 1;
    }
  }
  return NULL;
}
```

This is not efficient because the `strcmp` function compares the zip codes character by character. We can improve the search function by converting the zip codes to integers and compare them directly. The Table 1 shows the time taken for linear search with different data types while Table 2 shows the time taken for binary search with different data types.

ZIP Code	Search Type	Time (ns)
"111 15"	Linear Search	3 ns
"111 15"	Binary Search	39 ns
"984 99"	Linear Search	25037 ns
"984 99"	Binary Search	36 ns

Table 1: Comparison the Search Times for Char Data Type ZIP Codes in Linear and Binary Searches

ZIP Code	Search Type	Time (ns)
111 15	Linear Search	0 ns
111 15	Binary Search	18 ns
984 99	Linear Search	5076 ns
984 99	Binary Search	32 ns

Table 2: Comparison the Search Times for Integer Data Type ZIP Codes in Linear and Binary Searches

Before comparing the results, we should note that these two cases are the edge cases. The first case 111 15 is the first element in the array, and the second case 984 99 is the last element in the array.

For the first case, the linear search is faster than the binary search because the linear search can find the element in the first iteration. For the second case, the binary search is faster than the linear search because it does not have to iterate through all the elements to find the last element. Additionally, the data type does affect the search time. The integer data type is faster than the character data type in both linear and binary searches.

Direct Indexing

The direct indexing method is a way to improve the search time by using the zip code as an index to the array. To achieve this, I add a new `areas_direct` array is an array of pointers to the `area` struct and use the zip code as the index to the array. The modified `codes` struct is as follows:

```
typedef struct codes {
  area *areas;
  area **areas_direct; // Array of pointers to areas
```

```
  int n;
} codes;
```

To initialize the direct indexing, I create a new function `init_direct` to convert the zip code to an integer and use it as the index to the array after store the post codes using the original `read_postcodes` function.

```
codes *init_direct(codes *postnr) {
  postnr->areas_direct = (area**)malloc(sizeof(area*)*100000);
  for(int i = 0; i < postnr->n; i++) {
    int zip = zip_to_int(postnr->areas[i].zip_char);
    postnr->areas_direct[zip] = &postnr->areas[i];
  }
  return postnr;
}
```

Comparing the search time for the direct indexing method with the binary search methods, the direct indexing method is faster than the binary search method as shown in Table 3. The direct indexing method is faster because it uses the zip code as an index to the array, which is a constant time operation.

ZIP Code	Search Type	Time (ns)
111 15	Direct lookup	0 ns
111 15	Binary Search	18 ns
984 99	Direct lookup	0 ns
984 99	Binary Search	20 ns

Table 3: Comparison the Search Times for Integer Data Type ZIP Codes in Direct lookup and Binary Searches