

CPSC 335 – Project 3

Project Report

Members

- Andrew Lee / aslee@csu.fullerton.edu
- Michael Bui / buimichael@csu.fullerton.edu
- Natalia Garcia / natgarcia@csu.fullerton.edu

Explanation

Algorithm 1:

This algorithm makes use of the divide-and-conquer approach to efficiently determine the visibility of the buildings from the left and right by first dividing a given array in two halves. We implemented two functions, `recursiveLeft()` and `recursiveRight()`, that recursively find the visible buildings from either side using one of the two halved arrays. The lists returned consisted of the indices with the heights of the buildings visible from the left and right, following smaller to greater values for the right and vice versa for the left.

Algorithm 2:

For this algorithm, the greedy approach is utilized so that each step attempts to find the best overall optimal choice. Our design features a user-defined data type called `Activity` that represents the tuples inside a given list, which is then used to create the `greedySelect` function, taking in the list of activities from start to finish. First, a given list is compared and sorted by their activities' finish times. The list is then iterated, setting time equal to the finish time of the first tuple. The finish time of a tuple is compared to the start time of the next tuple. If the start time is greater or equal than the finish time, the tuple with the greater time is pushed into the selected list. Finally, only the tuples pushed into the list of selected activities are returned.

Pseudocode

Algorithm 1:

```
vector recursiveLeft(arr, start, end) {
    if start == end return vector<int>{start};

    mid = (start + end) / 2;

    left = recursiveLeft(arr, start, mid);
    right = recursiveLeft(arr, mid + 1, end);

    max = 0;
    for(int i = 0; i < left.length(); i++) {
        if max < arr[left[i]] {max = arr[left[i]]};
    }

    vector combined = left;
    for(int i = 0; i < right.length(); i++) {
        if arr[right[i]] > max {combined.append(right[i])};
    }

    return combined;
}

vector recursiveRight(arr, start, end) {
    if start == end return vector<int>{start};

    mid = (start + end) / 2;

    left = recursiveRight(arr, start, mid);
    right = recursiveRight(arr, mid + 1, end);
```

```

    max = 0;
    for(int i = 0; i < right.length(); i++) {
        if max < arr[right[i]] {max = arr[right[i]}};
    }

    vector combined = right;
    for(int i = 0; i < left.length(); i++) {
        if arr[left[i]] > max {combined.append(left[i])}};
    }

    return combined;
}

```

Algorithm 2:

```

greedySelect(list) {
    sorted = sortList(list);

    selected;
    time = 0

    for each item in sorted {
        if(time <= item.start) {
            selected.push_back(item);
            time = item.finish;
        }
    }

    return selected;
}

```

Screenshots

Algorithm 1:

Input

```
1      [3, 7, 8, 3, 6, 1]
```

Output

```
1      Left view --> [0, 1, 2]
2      Right view --> [5, 4, 2]
```

Algorithm 2:

Input

```
1      [(1, 3), (2,5), (4,6), (6,7), (5,9) , (8,9)]
```

Output

```
$ ./algo2.out
```

```
$ cat output2.txt
[(1, 3), (4, 6), (6, 7), (8, 9)]
```