Solutions for EoPI Variant Problems

December 31, 2022

Chapter 5

5.11 Rectangle Intersection

Variant 1

Given four points in the plane, how would check if they are the vertices of a rectangle?

Solution: solution

Variant 2

How would check if two rectangles, not necessarily aligned with the X and Y axes, intersect?

Solution: solution

Chapter 6

6.4 Advancing Through an Array

Variant 1

Write a program to compute the minimum number of steps needed to advance to the last location.

Solution: Iterate through the array and track the furthest index we know we can advance to at each iteration. The minimum number of steps needed is the number of iterations to have the furthest index at or past the last location.

```
int CanReachEndMinSteps(const vector<int>& max_advance_steps) {
    int max_index = 0, curr = 0, last_index = max_advance_steps.size()-1, steps = 0, lastReach;
    while (curr <= max_index && max_index < last_index) {
        lastReach = max_index;
        while (curr <= lastReach) {
            if (max_advance_steps[curr] + curr > max_index)
                max_index = max_advance_steps[curr] + curr;
            curr++;
        }
        steps++;
    }
    return (max_index < last_index) ? -1 : steps;
}
```

The time complexity is $O(n)$, and the space complexity is $O(1)$.

6.6 Delete Duplicates From a Sorted Array

Variant 1

Write a program which takes as input a sorted array A of integers and a positive integer m, and updates A so that if x appears m times in A it appears exactly $min(2, m)$ times in A. The update to A should be performed in one pass, and no additional storage may be allocated.

Solution: Since the array is sorted, repeated elements must appear one-after-another. We iterate through the array and keep track of the count of repeated integers. At each iteration, if the current integer is different from the previous one, we need to determine how many times the previous integer should repeat in updated array A and fill it in accordingly.

```
int VDeleteDuplicates(int m, vector<int> *A_ptr) {
    vector<int> &A = *A_ptr;
    if (A.empty())
```

```
        return 0;

    int idx = 1, tot_count = 0, count;
    for (int i=1; i<A.size(); i++) {
        if (A[i] == A[i-1])
            tot_count++;
        else if (A[i] != A[i-1]) {
            count = (tot_count == m) ? min(2,m) : tot_count;
            for (int j=1; j<count; j++)
                A[idx++] = A[i-1];
            A[idx++] = A[i];
            tot_count = 1;
        }
    }
    count = (tot_count == m ) ? min(2,m) : tot_count;
    for (int i=1; i<count; i++)
        A[idx++] = A.back();

    return idx;
}
```

The time complexity is $O(n)$, and the space complexity is $O(1)$.