**EE 3980 Algorithms**

Homework 5. Trading Stock

105061110 周柏宇

2020/4/12

1. **Introduction**

*In this homework, we implemented a generic connectivity algorithm to solve the network connectivity problem. In the connectivity algorithm, there are two disjoint set operations: SetFind and SetUnion. We not only implemented the vanilla SetFind and SetUnion, but also implemented their alternatives CollapsingFind and WeightedUnion, respectively. To measure the performance, we execute some combinations of the SetFind and SetUnion to solve graphs with different number of vertices and edges. In the end, we record the result and observe the growth of CPU time to verify our derivation of their time complexity.*

1. **Analysis & Implementation**
   1. **Maximum Subarray – Brute-Force Approach**
2. // Find low and high to maximize ΣA[i], low <= i <= high.
3. // Input: A[1 : n], int n
4. // Output: 1 <= low <= high <= n and max
5. **Algorithm** MaxSubArrayBF(A, n, low, high)
6. {
7. max := 0; // initialize
8. low := 1;
9. high := n;
10. **for** j := 1 **to** n **do** { // try all possible A[j : k]
11. **for** k := j **to** n **do** {
12. sum := 0;
13. // summation of A[j : k]
14. **for** i := j **to** k **do** {
15. sum := sum + A[i];
16. }
17. // record the maximum value and range
18. **if** (sum > max) **then** {
19. max := sum;
20. low := j;
21. high := k;
22. }
23. }
24. }
25. **return** max;
26. }
    1. **Maximum Subarray – Divide-and-Conquer Approach**
27. // Find low and high to maximize ΣA[i],
28. // begin <= low <= i <= high <= end.
29. // Input: A, int begin <= end
30. // Output: begin <= low <= high <= end and max
31. **Algorithm** MaxSubArray(A, begin, end, low, high)
32. {
33. **if** (begin = end) **then** { // termination condition
34. low := begin; high := end;
35. **return** A[begin];
36. }
37. mid := ⌊(begin + end) / 2⌋;
38. // left region
39. lsum := MaxSubArray(A, begin, mid, llow, lhigh);
40. // right region
41. rsum := MaxSubArray(A, mid + 1, end, rlow, rhigh);
42. // cross boundary
43. xsum :=
44. MaxSubArrayXB(A, begin, mid, end, xlow, xhigh);
45. // lsum is the largest
46. **if** (lsum >= rsum **and** lsum >= xsum) **then** {
47. low := llow; high := lhigh;
48. **return** lsum;
49. }
50. // rsum is the largest
51. **else** **if** (rsum >= lsum **and** rsum >= xsum) **then** {
52. low := rlow; high := rhigh;
53. **return** rsum;
54. }
55. low := xlow; high := xhigh;
56. **return** xsum; // cross-boundary is the largest
57. }
58. // Find low and high to maximize ΣA[i],
59. // begin <= low <= mid <= high <= end.
60. // Input: A, int begin <= mid <= end
61. // Output: low <= mid <= high and max
62. **Algorithm** MaxSubArrayXB(A, begin, mid, end, low, high)
63. {
64. lsum := 0; // initialize for lower half
65. low := mid;
66. sum := 0;
67. // find low to maximize ΣA[low : mid]
68. **for** i := mid **to** begin **step** -1 **do** {
69. sum := sum + A[i]; // continue to add
70. **if** (sum > lsum) **then** { //record if larger
71. lsum := sum;
72. low := i;
73. }
74. }
75. rsum := 0; // initialize for upper half
76. high := mid + 1;
77. sum := 0;
78. // find high to maximize ΣA[mid + 1 : high]
79. **for** i := mid + 1 **to** end **do** {
80. sum := sum + A[i]; // continue to add
81. **if** (sum > rsum) **then** { // record if larger
82. rsum := sum;
83. high := i
84. }
85. }
86. **return** lsum + rsum; // overall sum
87. }

Best-case time complexity:

Average time complexity:

Worst-case time complexity:

Space complexity:

1. **Result and Observation**

*To measure the performance, we execute the generic connectivity algorithm with some combinations of SetFind, CollapsingFind, SetUnion and WeightedUnion to solve graphs with different number of vertices and edges. Also, each data point in the table is the average result of 100 executions.*

|  |  |  |
| --- | --- | --- |
| N | Brute-force approach | Divide and conquer approach |
| 16 | 8.82149e-06 | 1.44100e-06 |
| 32 | 4.69685e-05 | 3.59893e-06 |
| 64 | 3.13997e-04 | 8.94403e-06 |
| 128 | 2.47407e-03 | 2.03071e-05 |
| 256 | 1.97191e-02 | 4.32661e-05 |
| 512 | 1.46862e-01 | 9.37350e-05 |
| 1024 | 1.17791e+00 | 1.84481e-04 |
| 2048 | 9.22634e+00 | 3.83117e-04 |
| 3890 | 6.95493e+01 | 7.83225e-04 |

Table 1. Average CPU time [s] vs Input data size