CS 331: Algorithms and Complexity (Spring 2024)

Unique numbers: 50930/50935/50940/50945

Discussion Section: 6

Maximum sum of a substring

We are given a sequence of numbers $A = [a_1, a_2, ..., a_n]$. Now we need to compute the largest sum of a substring. More formally we need to calculate

$$S = \max_{0 \le i \le j \le n} \sum_{k=i+1}^{j} a_k.$$

Note: here a substring can be empty, i.e. $i = j$ in equation above.

Example: $A[1:8]$ = [1, -4, 2, 3, -1, 2, -3, 2], then the maximum sum of substring is $sum(A[3:6])$ = 2+3-1+2 = 6.

Idea to approach:

1. **Brute force:** Enumerate over i, j, calculate $\sum_{k=i+1}^{j} a_k$ and pick the largest one, which take $O(n^3)$. A cleverer way can be done in $O(n^2)$.

2. **Divide and Conquer:** Solve for maximum sum of substring in $A[0, n/2]$ and $A[n/2, n]$. And also we need to calculate the maximum sum of substring that contains $a_{n/2-1}, a_{n/2}$, which can be solved in each direction for n times, by enumerating all the summation of substring from $n/2 - 1$ to 1.

   We need to be a little tricky here to be done in n times, where if we need to calculate $a_1, a_1 + a_2, a_1 + a_2 + a_3, ..., a_1 + a_2 + ... + a_k$ in k time, we need to solve $S(i) = a_1 + ... + a_i$ and use that to calculate $S(i+1) = S(i) + a_{i+1}$.

   The recurrence can be solved by $T(n) = 2T(n/2) + n$, where we can solve in time $T(n) = O(n \log n)$.

3. **DP:** Many ways to solve it.

   (a) Design a subproblem. says $f(j)$ as maximum sum of substring ends in j, could be empty if we pick $i = j$.

   (b) Find the recurrence. For $f(j)$, either it is empty, the sum would be 0, or it contain at least a_j, then the maximum sum of substring containing a_j at right end is actually

$a_j + f(j-1)$. Thus the recurrence can be written as:

$$f(j) = \max\{0, a_j + f(j-1)\}$$

(c) Pick a style to write out your algorithm: top-down (memoization) vs bottom-up (iterative). We here use iterative style, where we loop j from 0 to n, with $f(-1) = 0$, to apply the recurrence equation above. And we maintain a variable S, initial as 0, to save the currently best $f(i)$ before iteration j: $S = \max_{i \leq j} f(i) = \max\{S_{old}, f(j)\}$. Finally we output S.

(d) Time complexity. We can see each iteration we only take constant time (one for calculate $f(j)$, one for update S) to update, so the time complexity is $O(n)$.