Homework 4

Haoyu Zhen

May 3, 2022

Problem 1

(a)

Intuitively, I design the algorithm 1. Ostensively, the complexity of the algorithm is $\mathcal{O}(n)$.

Algorithm 1 Find the maximum $(1,1)$-step subsequence

Input: A sequence of integers $a_1, \cdots, a_n$
Output: The maximu revenue we can get from a $(1,1)$-step subsequence

$\mathcal{R} \leftarrow 0$, $M_{\text{cur}} \leftarrow 0$, $i \leftarrow 1$
while $i \le n$ **do**
  $M_{\text{cur}} \leftarrow M_{\text{cur}} + a_i$
  if $M_{\text{cur}} < 0$ **then**
    $M_{\text{cur}} \leftarrow 0$
  end if
  if $M_{\text{cur}} > \mathcal{M}$ **then**
    $\mathcal{R} \leftarrow M_{\text{cur}}$
  end if
  $i \leftarrow i + 1$
end while
return $\mathcal{R}$

The soundness of Algo. 1:

- The interpretation of M_{cur}^i at a_i: the maximum ravenue which **ends** at a_i.
- The propostion above holds by mathematical induction. $M_{\text{cur}}^i = \max(0, M_{\text{cur}}^{i-1} + a_i)$.
- Thus $\mathcal{R} = \max_i M_{\text{cur}}^i$ which is what we want.

(b)

Similarly, we have algorithm 2.
The correctness of Algo 2 is almost same with that of Algo 1. The complexity is obviously $\mathcal{O}(n^2)$.

(c)

Design the algorithm 3 with the auxiliary of the *Priority Queue*.
The complexity is $\mathcal{O}(n)$ because every element in a will be inserted into PLL once.
The soundness of Algo 3 intrinsicly hold by the fact that PLL.Front at i is $\max_{j\in\{i-R,\cdots,i-L\}} M_j$ and M_i is the maximum ravenue which **ends** at a_i.

Algorithm 2 Find the maximum (L, R)-step subsequence

Input: A sequence of integers $a_1, \cdots, a_n$
Output: The maximu revenue we can get from a (L, R)-step subsequence

$\boldsymbol{M} \triangleq (M_1, M_2, .., M_n) \leftarrow \mathbf{0}$
$M_i \leftarrow \max(a_i, 0)$ for all $i \in [L]$
for $i = (L+1, L+2, \cdots, n)$ **do**
  $M_i = a_i + \max_{j \in \{i-R, \cdots, i-L\}} M_j$ *#Complexity* $\mathcal{O}(R-L) = \mathcal{O}(n)$
  $M_i \leftarrow \max(M_i, 0)$
end for
return $\max_{i \in [n]} M_i$

Algorithm 3 Find the maximum (L, R)-step subsequence (**Pro**)

Input: A sequence of integers $a_1, \cdots, a_n$
Output: The maximu revenue we can get from a (L, R)-step subsequence

$\boldsymbol{M} \triangleq (M_1, M_2, .., M_n) \leftarrow \mathbf{0}$
$M_i \leftarrow \max(a_i, 0)$ for all $i \in [L]$
PLL = NULL
for $i = (L+1, L+2, \cdots, n)$ **do**
  if PLL.front.index$\leq i - R$ **then**
    PLL.PopFront
  end if
  while PLL.back.value$\leq M_{i-L}$ **do**
    PLL.PopBack
  end while
  PLL.PushBack(M_{i-L})
  $M_i = a_i +$ PLL.Front
  $M_i \leftarrow \max(M_i, 0)$
end for
return $\max_{i \in [n]} M_i$

Problem 2

First we define the function $dp(i,j)$: the total number of comparisions for words $a_i, \cdots, a_j$. The transition equation is:

$$dp(i,j) = \sum_{k=i}^{j} w_i + \max_{i+1 \le k \le j-1} dp(i,k-1) + dp(k+1,j).$$

Proof: if k is the best BST's root for i to j, we obtain cost $= \sum_{k=i}^{j} w_i + dp(i,k-1) + dp(k+1,j)$.
Then we have the naive algorithm:

Algorithm 4 Find the best BST for the n words with the minimum number of comparisons

Input: A sequence $\boldsymbol{w} = (w_1, w_2, \cdots, w_n)$
Initiate two 2D arrays: $\boldsymbol{n}$ and **idx**
$n_{i,i} = w_i$ and $\text{idx}_{i,i} = i$
for $i = n-1, n-2 \cdots, 1$ **do**
  for $j = i+1, i+2, \cdots, n$ **do**
    $i \leftarrow t$ and $j \leftarrow k+t-1$
    $\boldsymbol{n}(i,j) = \sum_{k=i}^{j} w_i + \min_{[i+1 \le k \le j-1]} [\boldsymbol{n}(i,k-1) + \boldsymbol{n}(k+1,j)]$
    $\mathbf{idx} \leftarrow \arg\min_{[i+1 \le k \le j-1]} \boldsymbol{n}(i,k-1) + \boldsymbol{n}(k+1,j))$
  end for
end for

Now we could construct the tree with **idx**:

1. Let the root be $\mathbf{idx}[1,n]$.
2. If k is the root of the BST for $a_i, \cdots, a_j$, then k's left child is $\mathbf{idx}[i,k-1]$ and its right child is $\mathbf{idx}[k+1,j]$.
3. Do step 2 **recusively**.

The complexity is $\mathcal{O}(n^3)$. The soundness of the algorithm depends on that of the transition equation.

Problem 3

For given string s: $\bar{s}_{i,j} \triangleq s[i:j]$ which is a consecutive subsequencee of s. Then we define $f(i,j)$ as the longest palindrome with length $l(i,j)$ that is a subsequence of $\bar{s}_{i,j}$. Then

$$l(i,j) = \max\left[l(i+1,j),\, l(i,j-1),\, 2 \times 1(s_i = s_j) + l(i+1,j-1)\right] \tag{1}$$

And we update $f(i,j)$ correspondingly. The correctness of EQ 1 is trivial. Here's the *story*:

- If $s_i \neq s_j$, then the lonest palindrome is in $\bar{s}_{i,j-1}$ or in $\bar{s}_{i+1,j}$.
- If $s_i = s_j$, we have a new choice: $\text{append}(s_i, s', s_j)$ where s' is in $\bar{s}_{i+1,j-1}$.

Hence we design the algorithm 5.
The running time is $\mathcal{O}(n^2)$.

Algorithm 5 find the longest palindrome that is a subsequence of a string

Input: A string s

Initiate 2 2D arrays: $\boldsymbol{l}$ and $\boldsymbol{f}$
$l_{i,i} = 1$, $f_{i,i} = s_i$ and $l_{i,j} = 0$, $f(i,j) = \varnothing$ for all $i > j$
for $i = n-1, n-2 \cdots, 1$ **do**
    for $j = i+1, i+2, \cdots, n$ **do**
        Update l
    end for
end for
return $f(1, n)$

Problem 4

For any given tree G with root r, denote $f(r)$ as the number of independent sets in G. Then

$$f(r) = \prod_{v \in \{r\text{'s children}\}} f(v) + \prod_{v \in \{r\text{'s grandchildren}\}} f(v).$$

The first term of LHS represents the number of independent sets under the case that r is not in these sets. Choosing r, we will obtain the latter term.
Now we design the algorithm 6.

Algorithm 6 Count the number of independent sets in a tree

Input: A tree G with a arbitrary root r

Implement BFS from r. Assign the depth, parent and children to every vertex. And we could get a sequence $\boldsymbol{v} = (v_1, \cdots, v_n)$ where vertices is sorted by their depth.
for $u = (v_1, v_2, \cdots, v_n)$ **do**
    if u is a leaf **then**
        $f(u) = 1$
    else if u has no grandchild **then**
        $f(u) = \prod_{v \in \{u\text{'s children}\}} f(v)$
    else
        Update $f(u)$ by the transition EQ.
    end if
end for
return

The complexity is $\mathcal{O}(n)$ because every vertex will be mutiplied at most twice.

Problem 5

About a day. Difficulty 3. Collaborators: Yinlin Sun.