Algorithm Design and Analysis (Fall 2022) Assignment 4

Xiangyuan Xue (521030910387)

1. (a) Recall the state transition equation

$$f(i,w) = \max\{f(i-1,w), f(i-1,w-c_i)+v_i\}$$

where $c_i \le w \le W$.

Note that $f(i,w)$ only depends on $f(i-1,w)$, which implies that we only need to record the last states within a linear space. A simple way is using two lists and update the results alternately. Actually, only one list is required in following pseudo-codes.

Algorithm 1 Knapsack Problem

```
Input: $n$ items with cost $c_i$ and value $v_i$ and a capacity $W$
Output: Maximum total value $V$ with total cost $C \le W$
1: Let $f$ be a list with indices $\{0,1,2,\dots,W\}$
2: for $j \leftarrow 0$ to $W$ do
3:     $f(j) \leftarrow 0$
4: for $i \leftarrow 1$ to $n$ do
5:     for $j \leftarrow W$ down to $c_i$ do
6:         $f(j) \leftarrow \max\{f(j), f(j-c_i)+v_i\}$
7: return $\max_{0 \le j \le W} f(j)$
```

Updating in reversal order promises the topological order. We can see its time complexity is still $O(nW)$, but it needs only $O(W)$ space.

(b) Recall the state transition equation

$$f(i,j) = \min\begin{cases} f(i-1,j)+1 \\ f(i,j-1)+1 \\ f(i-1,j-1)+1_{x_i \ne y_j} \end{cases}$$

If we think of f as an $n \times m$ matrix, $f(i,j)$ only depends on $f(i-1,j)$, $f(i,j-1)$ and $f(i-1,j-1)$. Similarly, only linear space is required in following pseudo-codes.

Algorithm 2 Edit Distance

```
Input: Two strings $X = \{x_1, x_2, \dots, x_n\}$ and $Y = \{y_1, y_2, \dots, y_m\}$
Output: Edit distance between $X$ and $Y$
 1: if $n < m$ then
 2:     Swap($X$, $Y$)
 3:     Swap($n$, $m$)
 4: Let $f$ be a list with indices $\{0, 1, 2, \dots, m\}$
 5: for $j \leftarrow 0$ to $m$ do
 6:     $f(j) \leftarrow j$
 7: for $i \leftarrow 1$ to $n$ do
 8:     $temp \leftarrow f(0)$, $f(0) \leftarrow i$
 9:     for $j \leftarrow 1$ to $m$ do
10:         $last \leftarrow temp$, $temp \leftarrow f(j)$
11:         if $x_i = y_j$ then
12:             $f(j) \leftarrow last$
13:         else
14:             $f(j) \leftarrow \min\{last, f(j-1), f(j)\} + 1$
15: return $\min_{0 \le j \le m} f(j)$
```

Note that $f(j)$ should be temporarily stored before updating. This is because $f(j+1)$ depends on the original $f(j)$ instead of the updated $f(j)$. This extra operation promises the correctness on the topological order. We can see its time complexity is still $O(nm)$, but it needs only $O(\min\{n, m\})$ space.

2. (a) Since $L = R = 1$, we have $\forall 1 \le j \le l-1 : i_{j+1} - i_j = 1$. In other words, the sequence is successive. Let $f(k)$ denote the maximum revenue of a subsequence ending with a_k. The state transition equation is as follow

$$f(k) = \max\{a_k, f(k-1) + a_k\}$$

The answer is $\max_{1 \le k \le n} f(k)$. When updating, we record the initial position p_k for $f(k)$. In this way, we can find out the exact subsequence required.

We prove the correctness by induction on k.

- For $k = 1$, a_1 is exactly the subsequence, so $f(1) = a_1$.
- Suppose that for any $k = m$, $f(k)$ is the maximum revenue.
- For $k = m + 1$, there are two possible cases
  - The subsequence contains only one element, $f(k) = a_k$
  - The subsequence contains more than one element, $f(k) = f(k-1) + a_k$

  Thus, all the subproblems are calculated and $f(k)$ given by the state transition equation is the maximum revenue.

Therefore, this algorithm can always find the correct answer.

Consider its running time. Since $f(k)$ only depends on $f(k-1)$, the procedure is linear on the sequence, so its time complexity is also linearly $O(n)$.

(b) Similarly, let $f(k)$ denote the maximum revenue of a subsequence ending with a_k. The state transition equation is as follow

$$f(k) = \max\left\{a_k, \max_{k-R\le p\le k-L} f(p) + a_k\right\}$$

Note that when we enumerate p, we should promise that $p > 0$. Similarly, The answer is $\max\limits_{1\le k\le n} f(k)$. When updating, we record the previous position p_k for $f(k)$. In this way, we can trace back to find out the exact subsequence required.

We prove the correctness by induction on k.

- For $k = 1$, a_1 is exactly the subsequence, so $f(1) = a_1$.
- Suppose that for any $k \le m$, $f(k)$ is the maximum revenue.
- For $k = m + 1$, there are two possible cases
  - The subsequence contains only one element, then $f(k) = a_k$
  - The subsequence contains more than one element, then its last element can be $a_{k-R}, a_{k-R+1}, \dots, a_{k-L}$, so $f(k) = \max\limits_{k-R\le p\le k-L} f(p) + a_k$

  Thus, all the subproblems are calculated and $f(k)$ given by the state transition equation is the maximum revenue.

Therefore, this algorithm can always find the correct answer.

Consider its running time. For each k, we need $O(R-L)$ time for the inner loop to calculate the second case. Since $R - L \le n$ and the length of sequence is n, we can conclude that its time complexity is $O((R-L)\cdot n) = O(n^2)$.

(c) The updating procedure in 2b is essentially to find the maximum element in a successive sequence $f(k-R), f(k-R+1), \dots, f(k-L)$ which has been figured out before. This procedure can be accelerated by PLL (Potential Largest List). Details are described in following pesudo-codes.

Algorithm 3 Maximum (L, R)-step Subsequence

Input: Integer sequence $a_1, a_2, \dots, a_n$, lower bound L and upper bound R

Output: Maximum revenue from a (L, R)-step subsequence

```
 1: function UPDATE($k$)
 2:     while $Q.front().index < k - R$ do
 3:         $Q.pop\_front()$
 4:     while $Q.back().value \leq f(i - L)$ do
 5:         $Q.pop\_back()$
 6:     $Q.push\_back(f(i - L))$
 7:     return $Q.front()$
 8: Construct a double-ended queue $Q$
 9: for $k \leftarrow 1$ to $n$ do
10:     $f(k) \leftarrow \max\{a_k, \text{UPDATE}(k)\}$
11: return $\max_{1 \leq k \leq n} f(k)$
```

When updating, if we record the previous position p_k for $f(k)$, we can trace back to find out the exact subsequence required.

As the discussion in slides, we know that PLL records all the potential largest value in descending order. So the first element in PLL is the maximum value we are looking for. Except for the updating procedure, the algorithm is exactly the same with 2b. Therefore, this algorithm can always find the correct answer.

Consider its running time. If we charge the cost of updating to $a_1, a_2, \dots, a_n$, we can notice that each a_i will be pushed and popped only once, implying that the updating procedure is $O(1)$. The updating procedure happens exactly n times. Thus, its total time complexity is $O(n)$.

3. Let $f(i, j)$ denote the minimum total number of comparison for the best binary search tree containing words $a_i, a_{i+1}, \dots, a_j$. If we take a_k as the root of T and suppose L and R are subtrees of T, the total number of comparison is

$$\begin{aligned}\sum_{p=i}^{j} w_p l_p(T) &= w_k l_k(T) + \sum_{p=i}^{k-1} w_p l_p(T) + \sum_{p=k+1}^{j} w_p l_p(T) \\ &= w_k + \sum_{p=i}^{k-1} w_p \left[1 + l_p(L)\right] + \sum_{p=k+1}^{j} w_p \left[1 + l_p(R)\right] \\ &= \sum_{p=i}^{j} w_p + \sum_{p=i}^{k-1} w_p l_p(L) + \sum_{p=k+1}^{j} w_p l_p(R)\end{aligned}$$

If we have found the best binary search tree in $a_i, a_{i+1}, \dots, a_{k-1}$ and $a_{k+1}, a_{k+2}, \dots, a_j$ for any $i \leq k \leq j$, we have following state transition equation

$$f(i, j) = \min_{i \leq k \leq j} s(i, j) + f(i, k-1) + f(k+1, j)$$

where $s(i,j) = \sum_{p=i}^{j} w_p$, which has been figured out by preprocessing. Details are described in following pseudo-codes.

Algorithm 4 Best Binary Search Tree

```
Input: Number of searching $w_1, w_2, \dots, w_n$ for words $a_1, a_2, \dots, a_n$
Output: Minimum total number of comparison
 1: for $i \leftarrow 1$ to $n$ do
 2:     for $j \leftarrow i$ to $n$ do
 3:         $s(i,j) \leftarrow s(i,j-1) + w_j$
 4: for $i \leftarrow 1$ to $n$ do
 5:     for $j \leftarrow 1$ to $n$ do
 6:         if $i \le j$ then
 7:             $f(i,j) \leftarrow \infty$
 8:         else
 9:             $f(i,j) \leftarrow 0$
10: for $l \leftarrow 1$ to $n$ do
11:     for $i \leftarrow 1$ to $n - l + 1$ do
12:         $j \leftarrow i + l - 1$
13:         for $k \leftarrow i$ to $j$ do
14:             $f(i,j) \leftarrow \min\{f(i,j), s(i,j) + f(i,k-1) + f(k+1,j)\}$
15: return $f(1,n)$
```

We prove the correctness by induction on interval length $l = j - i + 1$.

- For $l = 1$, we have $i = j$ and a_i is the only element in the binary search tree. The total number of comparison is exactly w_i.
- Suppose that for any $l \le m$, $f(i,j)$ is the minimum number of comparison for the words $a_i, a_{i+1}, \dots, a_j$.
- For $l = m + 1$, if we take a_k as the root, the interval length of both sides will be no more than l and their minimum numbers of comparison have been figured out. If we try $k = i, i+1, \dots, j$, all the subproblems will be calculated. By former deduction, we know that the state transition equation calculates $f(i,j)$ correctly.

Therefore, this algorithm can always find the correct answer.

Consider its running time. The number of valid interval is $O(n^2)$. For each interval, the updating procedure enumerates the split point, which takes $O(n)$ time. Therefore, the total time complexity is $O(n^3)$. Note that preprocessing takes $O(n^2)$ time, which can be ignored.

4. **Lemma** Suppose we are at (x,y), if we move to (x',y') where $x' \ge x$ and $y' \ge y$, the lowest cost is exactly $|x' - x| + |y' - y|$.

Proof Since x, y, x', y' are integers, the inequality holds that

$$\left(x'-x\right)^2+\left(y'-y\right)^2 \geq \left|x'-x\right|+\left|y'-y\right|$$

which gives the lower bound of the cost. Actually, this lower bound can be reached by always moving to an adjacent point, namely $(x,y) \to (x+1,y) \to \cdots \to (x',y) \to (x',y+1) \to (x',y+2) \to \cdots \to (x',y')$. The cost of each step is 1, so the lowest cost is exactly $|x'-x|+|y'-y|$.

(a) Let $f(i,j)$ denote the maximum profit when the player is at (i,j). The state transition equation is as follow

$$f(i,j)=\max\{f(i-1,j), f(i,j-1)\}+v_{ij}-1$$

where v_{ij} is the value of the gift at (i,j), which can be 0 if there is no gift at (i,j). The answer is $\max\limits_{1\leq i,j\leq m} f(i,j)$. When updating, we record the previous point p_{ij} for $f(i,j)$. In this way, we can trace back to specify the movement.

We prove the correctness by induction on i and j.

- For $i=1$ or $j=1$, the best choice is always moving to adjacent point, because the sum of value is maximized and by lemma, the cost is minimized. So $f(i,j)$ is the maximum profit.
- Suppose that for any $i\leq a$ or $j\leq b$, $f(i,j)$ is the maximum profit.
- For $i=a+1$ or $j=b+1$, if the profit is maximized by moving directly from (i',j') such that $|i-i'|+|j-j'|\geq 2$, another path $(i',j') \to (i'+1,j') \to \cdots \to (i,j') \to (i,j'+1) \to \cdots \to (i,j)$ will be no worse, because the sum of value is non-decreasing and by lemma, the cost is non-increasing. So the profit is maximized by moving from either $(i-1,j)$ or $(i,j-1)$. Since all the subproblems have been calculated before, $f(i,j)$ is the maximum profit.

Therefore, this algorithm can always find the correct answer.

Consider its running time. The algorithm sequentially traverses the whole grid, whose size is $m\times m$. Therefore, its time complexity is $O(m^2)$.

(b) We divide the movement into stages. In each stage, the player collects a single gift. Let $f(i,j)$ denote the maximum profit in stage i and the player is at (x_j,y_j). The state transition equation is as follow

$$f(i,j)=\max_{x_k\leq x_j, y_k\leq y_j} f(i-1,k)+v_j-d(k,j)$$

where v_j is the value of the j-th gift and $d(k,j)=|x_k-x_j|+|y_k-y_j|$. Note that for the states $f(i,j)$ with no transition, we simply let $f(i,j)=-\infty$.

The answer is $\max\left\{0, \max\limits_{1\leq i,j\leq n} f(i,j)\right\}$. When updating, we record the previous gift p_j for $f(i,j)$. In this way, we can trace back to specify the movement.

We prove the correctness by induction on the stage i.

- For $i = 1$, if we collect the gift j, the value is v_j and by lemma, the minimum cost is $|x_j - 1| + |y_j - 1|$, so $f(1, j) = v_j - (|x_j - 1| + |y_j - 1|)$.
- Suppose that for any $i = p$, $f(i, j)$ is the maximum profit.
- For $i = p+1$, the player collects another gift j by moving from (x_k, y_k) to (x_j, y_j), which requires $x_k \leq x_j$ and $y_k \leq y_j$. The maximum profit comes from all the possible k, which are enumerated in the state transition equation. Since all the subproblems have been calculated before, $f(i, j)$ is the maximum profit.

Therefore, this algorithm can always find the correct answer.

Consider its running time. In the outer loop, this algorithm repeats n times for n stages. In the inner loop, this algorithm repeats n times for n gifts. Therefore, its time complexity is $O(n^2)$.

5. (a) It takes me about 8 hours to finish this assignment.
   (b) I prefer a 3/5 score for its difficulty.
   (c) I have no collaborators.