Algorithm Design and Analysis

Assignment 4

Wu Jiabao, 523030910241

1. Construct $f(i,j)$: the function returns a bool value to show whether $C[1:i+j]$ is a shuffle of $A[1:i]$ and $B[1:j]$.

$$f(i,j) = f(i-1,j)_{A[i]=C[i+j]} \vee f(i,j-1)_{B[j]=C[i+j]} \vee false_{A[i]\neq C[i+j] \wedge B[j]\neq C[i+j]}$$

Algorithm 1 Determine whether C is a shuffle of A and B

Input: strings $A,\ B,\ C$

Output: Whether C is a shuffle of A and B.

$f(0,0) \leftarrow true$
for i from 1 to n:
    for j from 1 to m:
        $f(i,j) \leftarrow f(i-1,j)_{A[i]=C[i+j]} \vee f(i,j-1)_{B[j]=C[i+j]} \vee false_{A[i]\neq C[i+j] \wedge B[j]\neq C[i+j]}$
return $f(n,m)$

Its time complexity is $O(nm)$.

2. (a) Construct $f(x,y)$: the maximal profit players get from $(1,1)$ to (x,y).

$$f(x,y) = \max\{f(x-1,y)+v(x,y)-1,\ f(x,y-1)+v(x,y)-1\}$$

Algorithm 2 Maximize the player's profit

Input: n gifts (x_i,y_i), m

Output: the maximum profit.

$v(x,y) \leftarrow 0$ if gifts are not on (x,y)
$v(x_i,y_i) \leftarrow v_i$
$f(1,1) \leftarrow 0,\ f(i,0) = f(0,j) \leftarrow -\infty$
for i from 1 to m:
    for j from 1 to m:
        $f(i,j) \leftarrow \max\{f(i-1,j)+v(i,j)-1,\ f(i,j-1)+v(i,j)-1\}$
return $\max\{f(i,j)\}$.

Prove of correctness: Assume we move further than 1 cost from (a,b) to (c,d). Then the cost will be greater than $(c-a)+(d-b)$, which is the cost of going step by step. Since we can only receive $v_{(c,d)}$ rather than all gifts along the way from (a,b) to (c,d), going step by step is an optimal choice.

There are many ways to go to (i,j) from $(1,1)$ step by step, and $\max\{f(i,j)\}$ selects the one with maximal profit.

(b) Construct $f(i)$: the maximal profit we can get from the first i gifts.

$$f(i) = \max\{f(i), (f(j) + v_i - (x_i - x_j) - (y_i - y_j))_{x_j \le x_i \wedge y_j \le y_j}\},\ j = 1, \cdots, i-1$$

Algorithm 3 Maximize the player's profit

Input: n gifts (x_i, y_i), m

Output: the maximum profit.

```
for i from 1 to n:
    f(i) ← v_i − x_i − y_i + 2
for i from 1 to n:
    for j from 1 to i − 1:
        f(i) = max{f(i), (f(j) + v_i − (x_i − x_j) − (y_i − y_j))_{x_j ≤ x_i ∧ y_j ≤ y_j}}
return max{f(i)}
```

Prove of correctness: Assume we always have the maximal profit during $i-1$-th iteration. Suppose in i-th iteration, if (x_j, y_j) is in the $Upper-left$ or $Lower-right$ of (x_i, y_i), $f(j)$ will keep its value; If it is in the $Lower-left$ of (x_i, y_i), assume $f(i) < f_{OPT}(i)$. Then we know that (x_j, y_j) causes the update of $f_{OPT}(i)$. Since we cannot go from (x_i, y_i) to (x_j, y_j), $f_{OPT}(j)$ is still maximal. Thus:

$$f_{OPT}(i) = \max\{f(i), (f(j) + v_i - (x_i - x_j) - (y_i - y_j))\}$$

Thus it causes a contradiction. The algorithm always finds the maximal profit of (x_i, y_i).

(c) I guess the answer is false. It is difficult enough to solve $Upper-right$ problem in polynomial-time.

3. (a) Construct $f(i,j)$: whether there is a subset $S = \{a_1, \cdots, a_i\}$ with sum exactly j.

Algorithm 4 Determine the subset

Input: T, k

Output: whether there is a subset $S = \{a_1, \cdots, a_i\}$ with sum exactly j.

$f(0,j) \leftarrow false$ for all $j = 1, \cdots, k$

$f(0,0) \leftarrow true$

for i from 1 to n:

  for j from 1 to k:

    $f(i,j) \leftarrow f(i-1,j) \vee (f(i-1,j-a_i) \wedge (j \geq a_i))$

return $f(n,k)$

Its time complexity is $O(kn)$.

(b) Construct $a_i' = \lfloor \frac{a_i}{K} \rfloor$ for all a_i, $k' = \lfloor \frac{k}{K} \rfloor$, where $K \in \mathbb{N}$ is big enough.

Then we employ the algorithm in (a) again and we receive an approximate solution S'.

Prove of correctness: Suppose there is a subset S such that $\sum_{a_i \in S} a_i = k$. Then we have:

$$k' - n \leq \sum_{a_i \in S} (\frac{a_i}{K} - 1) \leq \sum_{a_i \in S} a_i' \leq \sum_{a_i \in S} \frac{a_i}{K} \leq k' + 1$$

Thus $\sum_{a_i' \in S'} a_i' \in [k' - n, k' + 1]$.

$$k - (n+1)K \leq K(k' - n) \leq \sum_{a_i' \in S'} a_i \leq \sum_{a_i' \in S'} K a_i' + nK \leq k + (n+1)K$$

Hence let $\epsilon = \frac{(n+1)K}{k}$, we have $\sum_{a_i' \in S'} a_i \in [(1-\epsilon)k, (1+\epsilon)k]$.

Time complexity: $O(n \cdot \frac{k}{K}) = O(\frac{n^2}{\epsilon})$.

4. (a) Construct $f(v, true)$: the number of independent sets including v in the subtree of v.

$f(v, false)$: the number of independent sets not including v in the subtree of v.

$$f(v, true) = \prod_{u \in children(v)} f(u, false)$$

$$f(v, false) = \prod_{u \in children(v)} (f(u, false) + f(u, true))$$

Algorithm 5 Find the number of independent sets in G

Input: G, n

Output: the number of independent sets in G

$f(u, false) \leftarrow 1,\ f(u, true) \leftarrow 1$ for all u is a leaf.
for all internal vertice v in level order:
    $f(v, true) \leftarrow \prod_{u \in children(v)} f(u, false)$
    $f(v, false) \leftarrow \prod_{u \in children(v)} (f(u, false) + f(u, true))$
return $f(r, true) + f(r, false)$ where r is the root of G.

Prove of correctness: For the leaves the number of independent sets is 2.

Assume after $i-1$-th iteration all vertice with $i-1$ distance from leaves have got the number of independent sets.

In i-th iteration, choose a leaf's i-th ancestor u. Since the subtrees rooted in its children are independent, we only need to condier whether choose u or not. If u is selected, all its children are not. Otherwise its children can be selected or not.

Time complexity: Since each vertex v is processed once and each $f(u, true)$ and $(u, false)$ is looked up for once: from its parent. The overall complexity is $O(n)$.

(b) Construct $f(v, true)$: size of maximum independent sets in $subtree(v)$ including v.
$f(v, false)$: size of maximum independent sets in $subtree(v)$ not including v.
$g(v, true)$: number of maximum independent sets in $subtree(v)$ including v.
$g(v, false)$: number of maximum independent sets in $subtree(v)$ not including v.

Algorithm 6 Find the number of maximum independent sets in G

Input: G, n

Output: the number of independent sets in G

1: $f(u, false) \leftarrow 0,\ f(u, true) \leftarrow 1$ for all u is a leaf.
2: $g(u, false) \leftarrow 1,\ g(u, true) \leftarrow 1$ for all u is a leaf.
3: **for** all internal vertice v in descending-level-order:
4: $\quad f(v, true) \leftarrow 1 + \sum_{u \in children(v)} f(u, false)$
5: $\quad f(v, false) \leftarrow \sum_{u \in children(v)} \max\{f(u, true), f(u, false)\}$
6: $\quad g(v, true) \leftarrow \prod_{u \in children(v)} g(u, false)$
7: $\quad g(v, false) \leftarrow \prod_{u \in children(v)} \begin{cases} g(u, false) & f(u, false) > f(u, true) \\ g(u, false) + g(u, true) & f(u, false) = f(u, true) \\ g(u, true) & f(u, false) < f(u, true) \end{cases}$
8: **return** $g(r, true) + g(r, false)$ where r is the root of G.

Prove of correctness: If v is in the maximum independent set, its children must not be chosen. If v is not, we should consider its child.
Since we need to maximize the size of independent set, we should always choose vertice with maximal f.

Time complexity: $O(n)$ since all vertice are visited only once.

5. It takes me 12 hours to finish it. Difficulty is 4. Collaborators: Li Haochen, Wang Kun.