CSE 6331 HOMEWORK 6

BRENDAN WHITAKER

1. We wish to compute the weight of the minimum weight sequence of moves by which a pawn can go from first to last row. Let i the the index of the current row of the pawn. Then define $f(i,j)$ the be the weight of the minimum weight sequence from position (i,j) to row m. Then we have the recurrence relation:

$$f(i,j) = \text{weight}(i,j) + \min \begin{cases} f(i+1, j-1) \\ f(i+1, j) \\ f(i+1, j+1) \end{cases}. \tag{1}$$

where $1 \leq i, j \leq m$. And our boundary conditions are:

$$\begin{aligned} f(m+1, j) &= 0 \\ f(i, 0) &= \infty \\ f(i, m+1) &= \infty. \end{aligned} \tag{2}$$

Our goal is to compute $f(1,1)$. We give a non-recursive algorithm to compute $f(1,1)$ by the array $F[1..m+1, 0..m+1]$:

Algorithm 1: Chessboard-Array-F

Data: The weight matrix $Weight[1..m, 1..m]$.
Result: The array $F[i,j]$ computed for $1 \leq i, j \leq m$.

begin
  global array $F[1..m+1, 0..m+1]$;
  initialize $F[i, m+1] \longleftarrow 0, F[i, 0] \longleftarrow 0$ for $1 \leq i \leq m+1$;
  initialize $F[m+1, j] \longleftarrow 0$ for $0 \leq j \leq m+1$;
  for $i \leftarrow m$ **to** 1 **do**
    for $j \leftarrow m$ **to** 1 **do**
      $F[i,j] \longleftarrow Weight[i,j] + \min \{ F[i+1, j-1], F[i+1, j], F[i+1, j+1] \}$;
    end
  end
end

The asymptotic running time is $\Theta(n^2)$. The problem statement says it is not necessary to actually print the sequence, but if we wanted to, we would simply use a while loop, saying while $i \neq m+1$ starting at $i = 1$ and initializing $j = 1$, print (i,j), then compute $F[i,j] - Weight[i,j]$ and then we know that one of the elements of $\{ F[i+1, j-1], F[i+1, j], F[i+1, j+1] \}$ must be equal in value to this quantity. So we use an if-elseif-else statement to find the first of these which has the desired value, and then update i, j values to match the chosen next move.

2. We wish to compute the minimum number of operations needed to transform $A = a_1 \cdots a_m$ into $B = b_1 \cdots b_n$. We let X_i be the decision to delete/add/change/not change the character a_i such that it matches b_i, where we have at most $\max\{ m, n \}$ such decisions. Then let $f(i,j)$ be the minimum

Date: SP18.

number of operations needed to transform $A_i = a_i \cdots a_m$ into $B_i = b_i \cdots b_n$. Then we have:

$$f(i,j) = \begin{cases} 1 + \min \begin{cases} f(i+1,j) & \text{(delete } a_i\text{)} \\ f(i,j+1) & \text{(add new } a_i\text{)} \\ f(i+1,j+1) & \text{(replace } a_i \text{ with } b_j\text{)} \end{cases} & \text{if } a_i \neq b_j \\ f(i+1,j+1) & \text{if } a_i = b_j \end{cases}, \tag{3}$$

where $1 \leq i \leq m$ and $1 \leq j \leq m$. And our boundary conditions are:

$$\begin{aligned} f(n+1,m+1) &= 0, \\ f(i,n+1) &= 1, \\ f(m+1,j) &= 1, \end{aligned} \tag{4}$$

where $1 \leq i \leq m$ and $1 \leq j \leq n$. Our goal is to compute $f(1,1)$. We represent the values of $f(i,j)$ in an array $F[1..m+1, 1..n+1]$, which we compute in the following algorithm:

Algorithm 2: String-Array-F

Data: Two strings $A = a_1 \cdots a_m$ and $B = b_1 \cdots b_n$.
Result: The array $F[i,j]$ computed for $1 \leq i \leq m, 1 \leq j \leq n$.

begin
  global array $F[1..m+1, 1..n+1]$;
  initialize $F[m+1, n+1] \longleftarrow 0$;
  initialize $F[i, n+1] \longleftarrow 1$ for $1 \leq i \leq m$;
  initialize $F[m+1, j] \longleftarrow 1$ for $1 \leq j \leq n$;
  for $i \leftarrow m$ **to** 1 **do**
    for $j \leftarrow n$ **to** 1 **do**
      if $a_i = b_j$ **then**
        $F[i,j] \longleftarrow F[i+1, j+1]$;
      else
        $F[i,j] \longleftarrow 1 + \min \{ F[i+1,j], F[i,j+1] F[i+1,j+1] \}$;
      end
    end
  end
end

The asymptotic running time is $\Theta(mn)$. The problem statement does not ask us to print out the actual number of operations, but if we wanted to, we would loop while $i \neq m+1$ and $j \neq n+1$ starting from $i = j = 1$, and check if $a_i = a_j$. If it is, we increment both i, j and go to next iteration, otherwise, we increment our operation counter, and find the minimum of $\{ F[i+1,j], F[i,j+1] F[i+1,j+1] \}$ and set i, j to its indices, then go to next iteration.

3. We have a set J of N jobs with corresponding processing time t_i for each, rewards p_i for finishing by time T, and penalties q_i for not finishing by T. We wish to compute a subset of $S \subseteq J$ s.t.:

$$\sum_{i \in S} t_i \leq T, \tag{5}$$

and we maximize:

$$f(S) = \sum_{i \in S} p_i - \sum_{i \notin S} q_i. \tag{6}$$

So let the i-th subproblem denote our decision of whether or not to include job i in S. So define $g(i,t)$ to be the maximum profit of any subset S_i of jobs in J with index $\geq i$ s.t. $\sum_{k \in S_i} t_k \leq t$. We give a recurrence relation:

$$g(i,t) = \begin{cases} \max \begin{cases} p_i + g(i+1, t+t_i) \\ -q_i + g(i+1,t) \end{cases} & \text{if } t < T \\ g(i,t) = -q_i + g(i+1,t) & \text{if } t \geq T \end{cases}. \tag{7}$$

where $1 \leq i \leq N$. Our boundary condition is:

$$g(N+1, t) = 0. \tag{8}$$

Our goal is to compute $g(1,0)$. We give a non-recursive algorithm:
Define $t' = t - t_i$.

Algorithm 3: Job-Array-G

Data: The set J of N jobs, sets P, Q of rewards and penalties for each job, and the set $\mathcal{T}$ of times t_i for each job.

Result: The array $G[i,t]$ computed for $1 \leq i \leq N, 1 \leq j \leq T$.

```
begin
    global array G[1..N+1, 1..T];
    initialize G[N+1, t] ⟵ 0;
    for i ← N to 1 do
        for t ← T to 0 do
            int t' ⟵ t + t_i;
            if t' ≤ T then
                G[i,t] ⟵ max { P[i] + G[i+1, t'], −Q[i] + G[i+1, t] };
            else
                G[i,t] ⟵ −Q[i] + G[i+1, t];
            end
        end
    end
end
```

The running time is $O(NT)$.