Rhythm Flow

At first blush this problem looks like it's a vanilla assignment problem (or maximum-weight bipartite matching). But a naive implementation using e.g. the Hungarian algorithm won't pass the samples: a maximum-weight matching might violate the constraint that if two actual button presses are matched to expected button presses, the earlier actual press must match the earlier expected press.

But this same restriction allows us to formulate a problem solution using dynamic programming. Let `dp[i][j]` be the maximum possible score after matching the first i actual presses to the first j expected presses. We can fill in this DP table row-by-row:

for $j = 0 \dots n$ **do**
  $\mathrm{dp}[0][j] = 0$
end for
for $i = 1 \dots m$ **do**
  for $j = 0 \dots n$ **do**
    $\mathrm{dp}[i][j] = \mathrm{dp}[i-1][j]$ ▷ Do not match actual press i to any expected press
    for $k = j; k \geq 1; k = k - 1$ **do**
      $\mathrm{score} = s(a_i, e_k) + \mathrm{dp}[i-1][k-1]$ ▷ Match actual press i to expected press k
      $\mathrm{dp}[i][j] = \max(\mathrm{dp}[i][j], \mathrm{score})$
    end for
  end for
end for

where $s(a, e)$ is the score listed in the table. The answer to the problem is then `dp[m][n]`.

Improving the Time Complexity

The above algorithms runs in $O(n^2 m)$ time, which is too slow. But notice that the above DP can be improved to remove the inner loop completely. Instead of searching over all possible matches for actual press i, we can reuse `dp[i][j-1]`, which already computes the best possible score when matching all actual presses up to press i with all expected presses $0 \dots j - 1$:

for $j = 0 \dots n$ **do**
  $\mathrm{dp}[0][j] = 0$
end for
for $i = 1 \dots m$ **do**
  $\mathrm{dp}[i][0] = 0$
  for $j = 1 \dots n$ **do**
    $\mathrm{score1} = s(a_i, e_j) + \mathrm{dp}[i-1][j-1]$ ▷ Match actual press i to expected press j
    $\mathrm{score2} = \mathrm{dp}[i][j-1]$ ▷ Match actual press i to some expected press $< j$
    $\mathrm{score3} = \mathrm{dp}[i-1][j]$ ▷ Don't match actual press i at all
    $\mathrm{dp}[i][j] = \max(\mathrm{score1}, \mathrm{score2}, \mathrm{score3})$
  end for
end for

This solution runs in time $O(nm)$. (Note also that it's not necessarily to keep the entire DP table in memory: you only need the current and previous rows, though this optimization is not needed to fit within the problem memory limit.)

Alternate Solution

Instead of removing the inner loop of the $O(n^2 m)$ solution, it's also possible to optimize it by noticing that it never makes sense to match actual button press i to expected button press k once the distance between a_i and e_k is greater than 102 milliseconds:

for $j = 0 \dots n$ **do**

```
    dp[0][j] = 0
  end for
  for i = 1...m do
    for j = 0...n do
      dp[i][j] = dp[i-1][j]                 ▷ Do not match actual press i to any expected press
      for k = j; k ≥ 1; k = k-1 do
        if a_i - e_k > 102 then
          break
        end if
        score = s(a_i, e_k) + dp[i-1][k-1]  ▷ Match actual press i to expected press k
        dp[i][j] = max(dp[i][j], score)
      end for
    end for
  end for
```

The time complexity is now $O(nm)$ (with a worst-case $102\times$ constant factor due to the inner loop), which is enough to solve the problem within the time limit.