CS 577-2: Introduction to Algorithms Fall 2020

Solutions to Alternate Midterm Exam 1

Instructor: Dieter van Melkebeek

Part (a)

The optimal way to clear the subgrid $[n] \times [k]$ using row moves only is to independently optimally clear the prefix of length k of each row $i \in [n]$. The optimal way to clear the prefix of length k of row i is to use a single row move for each maximal substring of occupied cells. When increasing the length of the prefix from $k-1$ to k, the number of row moves remains the same unless the cell in column k starts a new maximal substring of occupied cells, which is the case iff the cell in column k is occupied and the cell in column $k-1$ is not. Aggregating over all rows, it follows that

$$R[k] = R[k-1] + |\{i \in [n] : C(k)_i = \text{'x' and } C(k-1)_i \neq \text{'x'}\}| \,. \tag{1}$$

The relationship (1) holds for $k > 1$. It holds for $k = 1$ as well provided we consider the cells in column 0 to be empty.

The relationship (1) allows us to compute the quantities $R[k]$ for increasing values of k by keeping track of the latest value and the latest two columns of G. For completeness, here is the pseudocode.

Algorithm 1

```
previous ← column consisting of n empty cells
R[0] ← 0
for k ← 1 to m do
    current ← C(k)
    R[k] ← R[k − 1] + |{i ∈ [n] : current_i = 'x' and previous_i ≠ 'x'}|
    previous ← current
return R[0..m]
```

The algorithm consists of m iterations that each take $O(n)$ time, for a total of $O(nm)$ time. As the algorithm only needs store the array $R[0..m]$ and keep track of two columns of n elements each, it runs in space $O(n+m)$.

Part (b)

For any given row $i \in [n]$, the row moves that make up an optimal way to clear the prefix $[\ell]$ can be broken up into:

(1) those that fall entirely within the prefix $[k-1]$,

(2) those that fall entirely within $\{k, \ldots, \ell\}$, and

(3) possibly one that crosses the boundary $[k-1, k]$, namely when the cells in columns $k-1$ and k are both occupied.

An optimal way to clear $[k-1]$ consists of (1) and (3) (restricted to the range $[k-1]$). Similarly, an optimal way to clear $\{k, \ldots, \ell\}$ consists of (2) and (3) (restricted to the range $\{k, \ldots, \ell\}$).

Aggregating over all rows $i \in [n]$, it follows that

$$R(k-1) + S(k,\ell) = R(\ell) + |\{i \in [n] : C(k-1)_i = \text{'x'} = C(k)_i\}| \,. \tag{2}$$

The relationship (2) holds for $k > 1$ as well as for $k = 1$ provided the cells in column 0 are considered empty.

Rearranging (2) yields a way to compute $S(k,\ell)$ with two queries to R, two column queries, and $O(n)$ time, namely as

$$S(k,\ell) \leftarrow R(\ell) - R(k-1) + |\{i \in [n] : C(k-1)_i = \text{'x'} = C(k)_i\}| \,.$$

Part (c)

We can apply the principle of optimality based on the fact that a column move breaks up the problem into two independent smaller subproblems of the same type, namely the subgrid to the left of that column, and the subgrid to the right. This is because the column move prevents any row moves that cross that column.

Moreover, if we consider the *first* column move (if any), then the subproblem on the left subgrid is of simpler type, namely of the type of part (a), as is the remaining problem in case of no column moves. Thus, we have the following cases based on the first column (if any) where we apply a column move:

case 1: If we do not use any column move, then what remains is to solve the problem in part (a) for the entire $[n] \times [m]$ grid. The best we can do in this case is $R[m]$.

case 2: If the first column move happens in column ℓ, then what remains is to solve the problem in part (a) for the subgrid $[n] \times [\ell - 1]$ and the problem in part (c) for the subgrid $[n] \times \{\ell+1, \ldots, m\}$. The best we can do in this case is $R[\ell-1] + 1$ plus the result of the recursive call to the problem of part (c) for the subgrid $[n] \times \{\ell+1, \ldots, m\}$. Note that a column move in column ℓ is only possible if column ℓ is fully occupied.

The result is the minimum over case 1 and case 2 for all $\ell \in [m]$ for which column ℓ is fully occupied.

The calls to problem (c) that arise throughout the recursion all have as argument a subgrid of the form $[n] \times \{k, \ldots, m\}$ for some $k \in [m+1]$. This leads to the following subproblems for $k \in [m+1]$: $\mathrm{OPT}(k)$ denotes the minimum number of moves needed to clear the subgrid $[n] \times \{k, \ldots, m\}$ of G.

Note that $\mathrm{OPT}(1)$ is the final answer. $\mathrm{OPT}(m+1) = 0$ as the empty grid requires no moves.

If we apply the principle of optimality as above to $\mathrm{OPT}(k)$, both in case 1 and in case 2 the instance of problem (a) becomes an instance of problem (b), namely $S(k,m)$ in case 1, and $S(k,\ell-1)$ in case 2. The recursive call to (c) in case 2 becomes the instance $\mathrm{OPT}(\ell+1)$. Thus, we obtain the following recurrence:

$$\mathrm{OPT}(k) = \min\left(S(k,m), \min_{k \le \ell \le m : C(\ell) = \text{'x'}^n} (S(k,\ell-1) + 1 + \mathrm{OPT}(\ell+1))\right)$$

The recurrence allows us to compute the values $\mathrm{OPT}(k)$ from $k = m$ down to $k = 1$ using the initialization $\mathrm{OPT}(m+1) = 0$. We return $\mathrm{OPT}(1)$.

Each application of the recurrence involves taking the minimum over $O(m)$ terms, where each term can be computed in time $O(n)$ using part (b) given access to the array $R[0..m]$. There are m such applications of the recurrence. This yields a contribution of $O(m \cdot n \cdot m) = O(nm^2)$ to the running time. In addition, we need to precompute the array $R[0..m]$, which takes time $O(nm)$ by part (a). This leads to an overall running time of $O(nm^2)$.

The memory space needed is $O(n + m)$ for computing and storing the array $R[0..m]$, $O(n)$ space for evaluating S (reusing that space for each evaluation), and $O(m)$ for storing the OPT array. Thus, the overall space usage is $O(n + m)$.

Alternate solution The problem can be cast as a shortest path problem in the graph $G = (V, E)$, where $V = [m+1]$ and there is an edge $(k, \ell+1)$ with length $S(k, \ell-1)+1$ for each $1 \leq k \leq \ell \leq m$ such that column ℓ is fully occupied, and an edge $(k, m+1)$ with length $S(k, m)$ for $1 \leq k \leq m$. The answer to part (c) is the minimum length of a path from 1 to $m + 1$.

As the graph G is a DAG, the shortest path problem can be solved in time $O(|V|+|E|) = O(m^2)$ multiplied with the time needed to compute the weight of an edge, which is $O(n)$ by part (b), for a total of $O(nm^2)$ time. The amount of space needed is $O(|V|) = O(m)$ plus the space needed to compute the weight of an edge, which is $O(n)$ by part (b), for a total of $O(n + m)$ space.