Homework 15

Brian Knotten, Brett Schreiber, Brian Falkenstein

October 1, 2018

23

This problem is a variation of the set partition problem where we are partitioning a set of size n (the n request times) into k subsets (each subset is the collection of requests satisfied if the information is sent at one of the k broadcasts). The decision tree for this problem can be thought of as:

- Each node has a branching factor of n and stores the cumulative waiting times given the partitions applied up until that point.
- The children of each node represent placing a partition at that position in R. So, the first child represents placing a partition at R_0, the second child a partition at R_1, etc.
- At depth i, i partitions have been placed.

Thus, the root of the tree would be null, and the first level would contain the waiting times resulting from placing a partition at $R_0 ... R_n$. Note that the sum of the waiting times in solutions that do not broadcast pages to users (that is, there are users requesting the page after the last partition) is ∞, because a valid solution must have every request met.
This will cover all $\binom{n}{k}$ possible ways to partition R k times. This tree can then be pruned and turned into a dynamic programming algorithm with the following rules:

1. If a node has more than k partitions, prune it. (this limits the height of the tree to k).
2. For nodes at the same depth (IE have made the same number of partitions), prune all but the one with the shortest cumulative wait time.

This limits the height of the tree to k, and the number of nodes at each depth to n (the sums must be calculated, and then pruned). The solution would then be found at the leaves, and would be the minimum sum over all the leaves.
This tree can then be imagined as an $n \times k$ table, where the rows represent the number of partitions (the depth of the tree) and the columns represent the places you can place the next partition (the children of each node in the tree). $A[i, j] =$ The total waiting time of adding the jth partition at time i.
A supplemental array $Broadcasts[n]$ is needed, where $Broadcasts[i] = 1$ if the optimal solution should broadcast at time $i + 1$, otherwise, $Broadcasts[i] = 0$. This algorithm is polynomial, as it takes time nk to traverse the table, and n time to compute the sum at each index, resulting in a runtime of $O(n^3)$.

```
for i = 1 to n do:
    Broadcasts[i] = 0                          # Initially, no broadcasts have been made

for j = 1 to k do:                             # For each new opportunity to broadcast (level of the tree)
    minimum_time = n                           # Keep track of the broadcast time which minimizes cost

    for i = n to 1 do:                         # For each time that can broadcast (node in the tree level)

        sum = 0                                # Calculate the wait time with the current broadcasts

        time_waiting = infinity                # Initially set the wait time to infinity so that unresolved
                                               # are waiting forever.
```

```
        for s = n to 1 do:                      # For each time interval
            if Broadcasts[s] == 1 or s == i:    # If this time has a previous or currently-considered broadca
                time_waiting = 1                # Reset the wait-time

            sum = sum + R[s] * time_waiting     # Sum up the wait times of each requester
            time_waiting = time_waiting + 1     # As it loops backwards in time, increase the wait time multi

        A[i, j] = sum

        if A[i, j] < A[i, minimum_time]:        # If a smaller wait time is found, redefine the minimum time
            minimum_time = j

        Broadcasts[minimum_time] = 1            # After all wait times have been considered,
                                                # broadcast at the time which minimizes wait time

Output Broadcasts                               # Output the whole binary Broadcasts array,
                                                # which has a 1 in the best times to broadcast.
```

26

The decision tree in this problem is similar to that in the previous problem, but instead of having a branching factor of just n, it will have a branching factor of $n \times j$, in order to account for placing the partition at any one of the n times for all j pages. The table will also be similar, but will now need a 3rd dimension to account for each page. That is, $A[p, q, r]$ represents: the minimum total waiting time of broadcasting Page r at time $p + 1$, having made q broadcasts in total.

The broadcast array will also get another dimension. Now, $Broadcasts[p, r] = 1$ if page r is broadcast at time $p + 1$, otherwise, $Broadcasts[p, r] = 0$.

Moreover, the increasing waiting for time must be considered for each page, so now $TimeWaiting$ is an array of size j, where $TimeWaiting[r] =$ the current amount of time waiting for page r.

The same pruning rules in 23 apply to 26.

```
for p = 1 to n do:
    for r = 1 to j do:
        Broadcasts[p, r] = 0 # Initially, no broadcasts have been made for any page

for q = 1 to k do:
    minimum_time = n
    minimum_page = j

    for p = n to 1 do:
        for r = 1 to n do:                          # For each page
            sum = 0

            TimeWaiting[r] = infinity               # Reset time_waiting to infinity so that unresolved req

            for s = n to 1 do:                      # For each time interval

                if Broadcasts[s, r] == 1 or s == p: # If this page at this time has a previous or currently
                    TimeWaiting[r] = 1              # Reset the time waiting for that particular page

                sum = sum + R[s, r] * TimeWaiting[r]
                TimeWaiting[r] = TimeWaiting[r] + 1

            A[p, q, r] = sum
```

```
            if A[p, q, r] < A[p, minimum_time, minimum_page]:
                minimum_time = p
                minimum_page = r

    Broadcasts[minimum_time, q, minimum_page] = 1

Output Broadcasts                                    # Output the whole binary Broadcasts array,
                                                     # which has a 1 in the best times to broadcast
                                                     # for each page.
```