

Homework 18

Brian Knotten, Brett Schreiber, Brian Falkenstein

October 7, 2018

27

A binary decision tree can be constructed which has the solution to this problem at the leaves. Let a node v at depth i of the decision tree represent a proposed solution for chapters x_1, x_2, \dots, x_i . Let the left child of v represent a proposed solution identical to v 's solution, but considering x_{i+1} in a new volume. Let the right child of v also represent a proposed solution identical to v 's solution, but which appends x_{i+1} into the same volume as x_i . The proposed solutions for all chapters x_1 through x_n will be at the leaves, however, there are 2^n leaves since it is a binary tree of depth n . So the tree must be pruned into a polynomial algorithm, and it also must be pruned for correctness.

The pruning rules are as follows:

1. If a vertex v proposes a solution with more than k volumes, then prune v , since the solution cannot use more than k volumes.

This pruning rule does not definitively reduce the input size, but it makes reasoning about the tree easier.

2. If two nodes u and v are on the same level, have the same number of volumes, and the difference between u 's longest and shortest volumes is less than or equal to the difference between v 's longest and shortest volumes, then prune v . v 's children can only decrease the difference between the longest and shortest volumes if they add to the shortest volume. And anything v 's children can add to the shortest volume, u can also add. So u 's children will always have an equal or better proposed solution than v 's children.

This pruning rule reduces the number of nodes at any given depth to k , since no two nodes at the same depth can use the same number of volumes, since one of them will be pruned. Therefore the tree is of size nk .

With these pruning rules, the tree can be turned into a 2-dimensional array A , where $A[i, j]$ = the least difference in pages between the smallest and largest volumes of a partition of chapters x_1 through x_i into j volumes.

The algorithm to populate A is as follows and runs in $O(nk)$:

```
for i = 1 to n do:
    A[i, 1]                # Base case: having only 1 volume implies 0 distance

for j = 1 to k do:
    A[0, j] = 0            # Base case: 0 pages implies 0 distance

for i = 1 to n do:        # For each chapter x[i]
    for j = 1 to k do:    # For every possible number of volumes:

        A[i, j] = min(    # Take the minimum of either:
            A[i - 1, j],  # appending to the previous volume
            A[i, j]       # or of what is already in the array cell
        )
```

```

if j < n:                # Only create a new volume if it's possible

A[i, j + 1] = min(       # Take the minimum of either:
    A [i - 1, j], # Adding it to its own volume
    A[i, j + 1]    # or of what is already in the array cell
)

```

Output A[n, k] # The solution is the minimum distance between the smallest and
largest volumes considering all n chapters and k volumes

28

a

The decision tree for this problem can be constructed with the following rules:

- Nodes in the tree have n children, where the i 'th child for $1 \leq i \leq n$ corresponds to choosing v_i to be a center point. Thus, the tree has height k and will contain all solutions at the leaves.
- The nodes will store the current aggregate distance, that is, the sum of all the distances from each $v \in V$ to its closest center point $c \in C$.

The total size of the tree is n^k . This can then be pruned with the following rules:

1. If a node at depth i already exists in C , prune it. That is, if that vertex has already been added to C , remove that sub tree, as a valid solution contains k different vertices in C .
2. If two nodes u and v at depth i have the same $c_1 \dots c_{i-1}$ center points, and $D_u < D_v$, prune v (where D_u is the total aggregate distance for u).

The first pruning rule leaves the tree as still roughly n^k . However, under the second pruning rule, every depth i is limited to n nodes. This is because at depth $i - 1$, for some node v , all of v 's children have the same $c_1 \dots c_{i-1}$, and only one will not be pruned, that is the one with the shortest D . This brings the size of the tree down to nk . At each node v , a new aggregate distance must be calculated, which takes nk calculations, which leaves this tree with a run time of $O(n^2k^2)$.

The dynamic algorithm is given as follows:

```

A = [k, n]                # Construct A to be an array of size k by n

A[0, :] = infinity        # Base case: having zero centers gives a
                           # distance of infinity

for i = 1 to k do         # For each row
    last_v = min(A[i-1, :]) # Retrieve the minimum distance vertex from the previous row,
                           # and use it for calculations on the current row.

    for j = 1 to no do     # For each row, check all possible center values
        if(last_v != V[j]) # Disregard already-used vertices
            A[i, j] = new_dist(last_v, V[j]) # Calculate new distance with V[j] as a new center

output min(A[k, :])        # Return the minimum value in the last row

```

Here, A[i, j] represents the minimum aggregate distance using i center points, where v_j is the last vertex to be chosen as a center.