CSE 101 Homework 7

Brian Masse, Taira Sakamoto, Emily Xie, Annabelle Coles

December 4, 2025

1. **Bus routes**

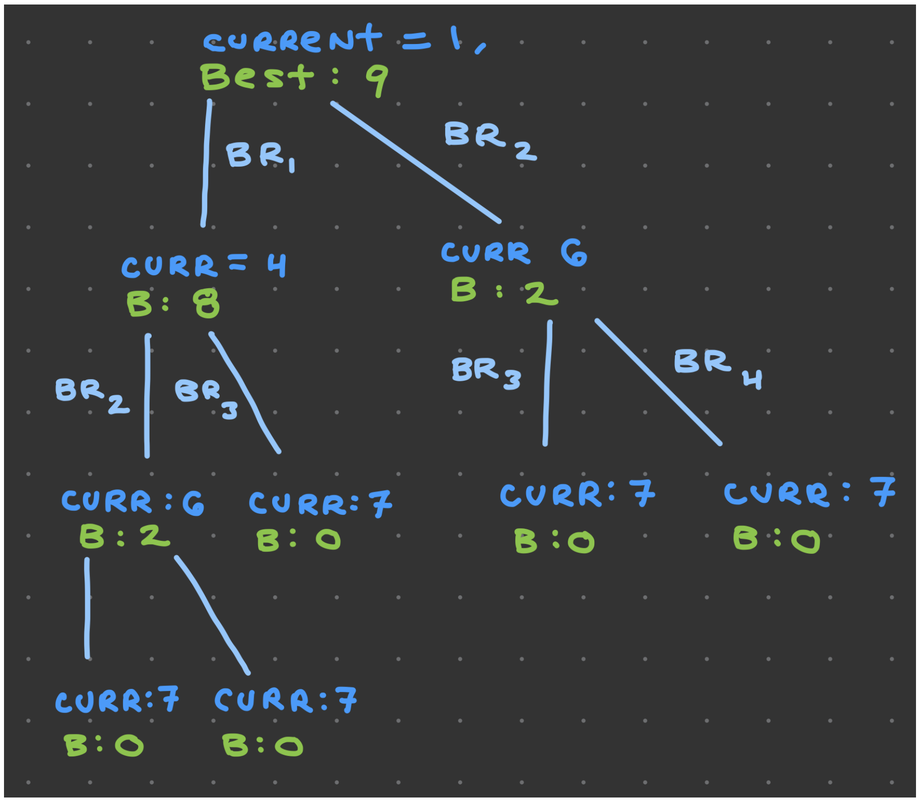
I am going by bus across town, along a path that includes bus stops $1...m$. There are n different bus routes BR_1 ...BR_n, where BR_i goes along my route between stop s_i to stop f_i, and costs c_i to ride. I want to go from stop 1 to stop m paying the smallest total amount as I can. You can assume there will always be a later bus for each route passing each stop.

$BTBusCosts(BR_1..BR_n; current; m)$

```
(a) IF $current \geq m$ return 0.
(b) $Best$ = infinity;
(c) FOR $I = 1$ to $n$ do:
(d)     IF $s_i \leq current < f_i$
(e)         THEN $Best = min(Best, c_i + BTBusCosts(BR_1...BR_n; f_i; m)$.
(f) Return $Best$
```

Questions

(a) Give the tree of recursive calls on the instance $current = 1, m = 7$, $BR_1 = (1, 4, 3), BR_2 = (1, 6, 7), BR_3 = (2, 7, 8), BR_4 = (5, 7, 2)$.



(b) Give an upper bound for the total number of recursive calls for this algorithm.

Note that the worst case has bus routes of the following form, since it maximizes the number of recursive calls at each level.

$$\begin{aligned} BR_1 &= (1,2) \\ BR_2 &= (1,3) \\ &\vdots \\ BR_n &= (1,n+1) \end{aligned}$$

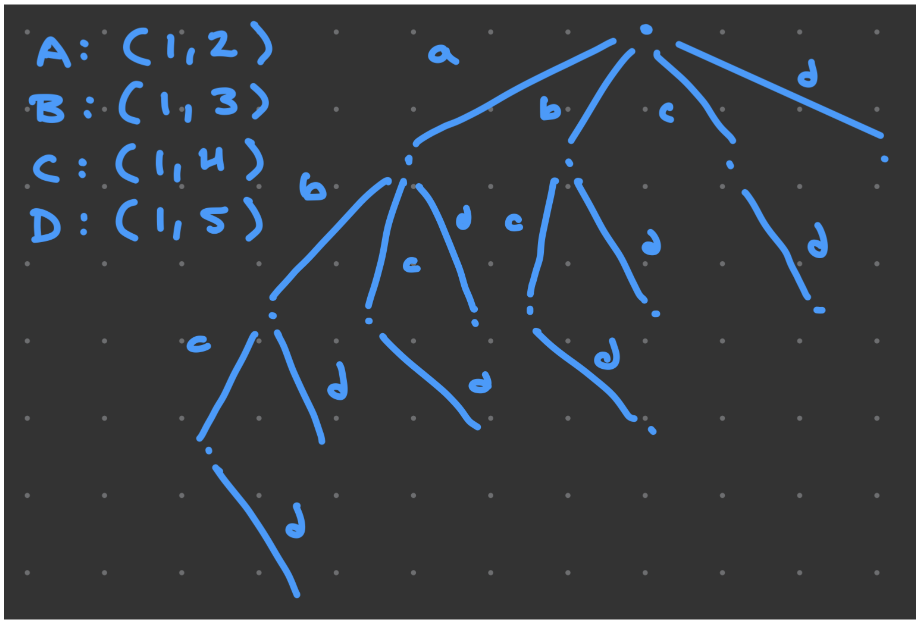
Let x be the distance from current position to position m. Then $T(x)$ is the number of recursive calls at a distance x to the finish:

$$\begin{aligned} T(x) &= T(x-1)+T(x-2)+...T(1) \\ &\leq x \cdot T(x-1) \text{ Since, } T(x-i-1) \leq T(x-i) \forall i \geq 1 \end{aligned}$$

Solving the recurrence relation with base case $T(1) = 1$

$$\begin{aligned} T(x) &\leq x \cdot T(x-1) \\ &\vdots \\ &\leq (x)(x-1)\cdots T(1) \\ &\leq x! \end{aligned}$$

Supplemental Diagram:



(c) Which variables change during this recursion?

Only the value of *current* changes each iteration.

(d) Define sub-problems: Use the above to define a set of sub-problems that can be used to convert the BT algorithm to a DP algorithm, and define an array or matrix to hold the answers to these subproblems.

- Each subproblem starts at some $i \in 1, ...m$ and returns the lowest cost path from i to m. Each operates with the same bus stops and works towards the same final point m. The total number of distinct subproblems is m.
- The solutions to subproblems can be stored in a length m array, where each position is the cost to get from position i to m

(e) What are the base cases?

The singular base case is $i = m$, the subproblem where you start at the end. The cost is 0.

(f) Translate the BT algorithm into a definition of array/matrix values from other values

On iteration k, $k \in \{1, ...m\}$:

```
for $BR_i$ such that $s_i \leq k < f_i$:
    $costs[k] = min(costs[k], c_i + costs[f_i])$
```

(g) In what order should we fill this array or matrix?

This matrix will be filled from the last position $k = m$ to the first position $k = 1$

(h) Assemble these pieces into a DP algorithm.

```
def bus_stops(BR1...BRn, curr, m):

    // there may not be a valid path from every point i to m,
    // so fill all spots except the ending
    cost = [ infinity ]
    cost[m] = 0

    // go from the end of the route to the start
    for k from m - 1 to 1:

        // si, fi, ci represent the start, finish, and cost of a bus route
        // they are defined and accessible from outside the function
        for all BRi in BR1...BRn such that si <= k < fi:
            cost[k] = min(cost[k], ci + cost[fi])

    return cost[1]
```

(i) Give a time analysis for your DP algorithm.

- Assuming accessing costs array is $O(1)$ then the main loop $\in O(m) \cdot O(1) \in O(m)$

- Assume finding all relevant BR_i can be done in worst case $\in O(n)$

Therefore, the total runtime of the algorithm is $\in O(n \cdot m)$

(j) Give the array or matrix for your DP algorithm on the given example.

$m = 7$

$BR_1 = (1, 4, 3)$

$BR_2 = (1, 6, 7)$

$BR_3 = (2, 7, 8)$

$BR_4 = (5, 7, 2)$

$$\begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\infty & \infty & \infty & \infty & \infty & \infty & 0 \\
9 & 2 & 8 & 8 & 2 & 2 & 0
\end{array}$$

2. **Bounded difference sequence**

   We say a sequence of numbers $a_1, ..a_n$ is K-bounded difference if $|a_{i+1} - a_i| \leq K$ for all $1 \leq i \leq n-1$. The maximum bounded difference subsequence problem is, given sequence $a_1, ..a_n$, and a real number $K > 0$, decide the length ℓ of the longest subsequences $a_{i_1}, ..a_{i_\ell}$, $1 = i_1 < i_2 < ...i_\ell$ including position one which is K-bounded difference.

   For example, if $a[1..5] = 2, 7, 1, 4, 3$ and $K = 2$, $2, 1, 3$ is a maximum length K bounded subsequence, as is $2, 4, 3$.

3. **Minimum weight connected subtree of given size**

   You are given a binary tree of size n, where every vertex x has pointers $lc.x$, and $rc.x$ (which could be NIL if those children don't exist) and each vertex has a value, $value.x > 0$. You are also given an integer $1 \leq k \leq n$. You want to find the connected sub-tree with exactly k vertices, which minimizes the total weight of vertices in the sub-tree.

   (a) Characterize Sub Problems

   Find the size and weight of every subtree within the initial tree. If the tree has k nodes, compare the weight of that tree to the running minimum.

   The set of all distinct subproblems, S, = the subtrees starting at every distinct node $v \in V$. This implies there are $|S| = |V| = n$ distinct subproblems

   (b) Base Case

   The root of the subtree has no children (the subtree is 1 node).

   - $size = 1$
   - $weight = value.v$

   (c) Recursive Definition of Answers

   If $|v.children| = 0$:

$$\begin{aligned} weight[v] &= value.v \\ size[v] &= 1 \end{aligned}$$

   If $|v.children| = 1$:

$$\begin{aligned} weight[v] &= value.v + weight[v.child] \\ size[v] &= 1 + size[v.child] \end{aligned}$$

   else:

$$\begin{aligned} weight[v] &= value.v + weight[lc.v] + weight[rc.v] \\ size[v] &= 1 + size[lc.v] + size[rc.v] \end{aligned}$$

   Brief Justification: For each subtree subproblem, there are 3 possibilities:

   - The root of the subtree has no children. In this case the size of the subtree is just 1, and its weight is just the weight of its root. There is no recursive relationship, and thus the return values do not depend on the memoized array.

   - The root of the subtree has 1 child. In this case the size of the subtree is 1 + the size of the root's only child, and the weight it the weight of root + the weight of its child. In this case, the current solution depends on the return values of its only child, thus the call to $weight[v.child]$ and $size[v.child]$

- The root of the subtree has 2 children. In this case the size of the subtree is 1 + the size of each of the subtrees rooted at its children, and its weight is the weight of the root + the weight of those subtreets. The solution depends on the return values of both the left and right child.

Therefore, the above recurrence covers the only 3 possibly cases for each subtree in the graph.

(d) Subproblem Order

The subproblem originating at node v must come after the subproblems originating at its children. Therefore, subproblems must be solved in a post-order traversal (both children before their parents.)

(e) Form of Output

While calculating the weight of each subtree, keep track of the smallest weight. For those with node count $= k$, check if its weight it lower than the current smallest. After scanning through all subtrees, return the smallest.

Note: Alternatively, keep track of node v where the smallest subtree occurs to return not just the smallest weight, but the root of the subtree too.

(f) Pseudocode

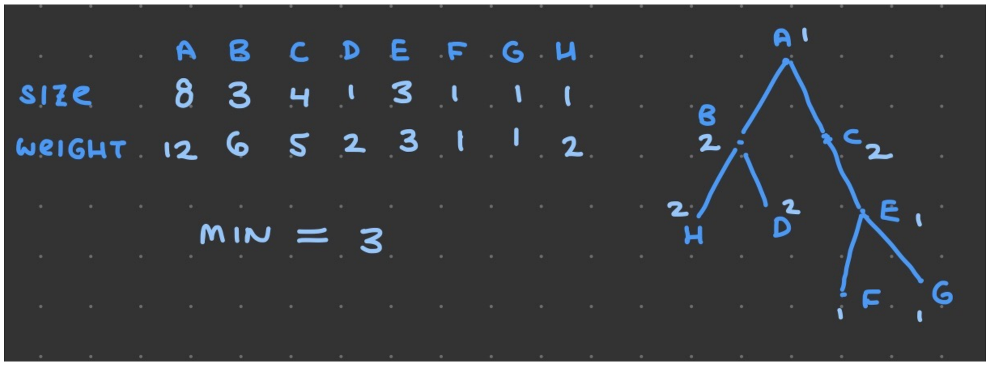
```
def MWCS(root, k) -> smallest:

// if a subtree does not contain k nodes (ie. does not satisfy constraints of
   potential solution)
// return infinity
smallest = infinity

for v in (post-order traverse root):
    if |v.children| == 0:
        weight[v] = v.value
        size[v] = 1

    if |v.children| == 1:
        weight[v] = v.value + weight[v.child]
        size[v] = 1 + size[v.child]

    else:
        weight[v] = v.value + weight[lc.v] + weight[rc.v]
        size[v] = 1 + size[lc.v] + size[rc.v]

    // attempt to update the variable for subtrees satisfying problem
       constraints
    if size[v] == k:
        smallest = min(smallest, weight[v])
    return smallest
```

(g) Runtime Analysis

- Post-traversal $\in O(n)$
- Runtime for each node: $\in O(1)$

Thus, total runtime $\in O(n)$

(h) Small Example



	A	B	C	D	E	F	G	H
SIZE	8	3	4	1	3	1	1	1
WEIGHT	12	6	5	2	3	1	1	2

MIN = 3

4. **Road trip**

   You are going on a long trip. You start on the road at mile post 0. Along the way there are n hotels, at mile posts $a_1 < a_2 < \cdots < a_n$, where each a_i is measured from the starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance a_n), which is your destination.

   You'd ideally like to travel 200 miles a day, but this may not be possible (depending on the spacing of the hotels). If you travel x miles during a day, the *penalty* for that day is $(200 - x)^2$. You want to plan your trip so as to minimize the total penalty—that is, the sum, over all travel days, of the daily penalties.

5. **Knapsack variant**

   Consider the variant of knapsack where you are allowed to choose items more than once. You still have n items each with values v_i and costs c_i, and a budget U. If you pick item i p_i times, you must have $\sum_{1 \leq i \leq n} p_i c_i \leq U$, and your objective is to maximize $\sum_{1 \leq i \leq n} p_i v_i$. Each p_i must be a non-negative integer.