2. Assume a situation that the current value of change is v, and a function mincoin(v) will give the minimum number of coins for this value. Approach the following problems in the prescribed order!
   - A coin is to be picked for the change. What are the options (possible coin values) for this coin?
   - What is the remaining value of change after picking each option for that coin?
   - *What **is** the minimum number of coins* for ***each*** *remaining value of change*? (This a critical step that you must learn. The answer is quite philosophical)
   - **THEN**, what would be the value of mincoin(v)?
   - What is(are) the value(s) of change for which the minimum number of coins is obvious?

Ans:

2. **Given a current change value v and a function `mincoin(v)`, answer the following:**
   - **Options for picking a coin:** The possible coin values you have.
   - **Remaining change after picking a coin:** Subtract the coin value from the change value.
   - **Minimum number of coins for each remaining value:** Use the function `mincoin` to get this value.
   - **Value of `mincoin(v)`:** The minimum number of coins needed.
   - **Values for obvious minimum coins:** When change is 0, you need 0 coins. When change is 1, you need 1 coin.

1) Given a rod of length l, list out all the possible ways to cut the rod "one time"? What is the result of each cutting? (To sell a rod at the cut length and keep the rest for the next sells.)

2) Following question 1 above, if the maximum revenue for a rod of length l is determined by function maxRev(l), what is the maximum revenue possible as the result of *each* cutting in question 1?

3) THEN, what would be the value of maxRev(l) ?

Ans:

1. **Given a rod of length l, list all possible ways to cut the rod one time.**

   Answer:

   - Cut the rod into two pieces: length i and $l - i$ for $i = 1, 2, \ldots, l - 1$.

2. **If the maximum revenue for a rod of length l is determined by `maxRev(1)`, what is the maximum revenue for each cutting?**

   Answer:

   - For each cut, the maximum revenue is the sum of the price of the cut piece and the maximum revenue for the remaining piece, i.e., `price[i] + maxRev(l - i)`.

3. **Value of `maxRev(1)`:**

   Answer:

   - The maximum revenue is the maximum of all possible revenues from cutting the rod at different lengths.

4. Therefore, for length of L, at most how many calls to all $\mathrm{maxRev}()$'s should be adequate ?

5. What is the reason that there were too many calls to $\mathrm{maxRev}()$ in the brute-force solution ?

6. Suggest a modification to the brute-force solution such that once a value of an $\mathrm{maxRev}(l)$ is already computed, it will never have to be recomputed again. Only idea is needed in this step.

Ans:

4. Therefore, for length of `L`, at most how many calls to all `maxRev()`'s should be adequate?

- **Answer**: For length `L`, at most `L` calls should be adequate if memoization is used, because each length only needs to be computed once.

5. What is the reason that there were too many calls to `maxRev()` in the brute-force solution?

- **Answer**: The brute-force solution has too many calls because it recalculates the value for the same length multiple times.

6. Suggest a modification to the brute-force solution such that once a value of an `maxRev(1)` is already computed, it will never have to be recomputed again.

- **Idea**: Use a memoization technique by storing the results of `maxRev(1)` in an array and reuse them if they are already computed.

8. Observe how faster the algorithm becomes (comparing the total number of recursive calls).

- **Answer**: By printing `calls`, you can compare the number of calls between the brute-force and memoized versions.

2. **ISSUE:** Based on technique in step 1 above, suppose that items are determined in order from item 0 to n-1, when the algorithm *is deciding* between selecting item i or not, there is no associated information of how the items 0 to i-1 have been selected!
   Therefore, the total number of states that decide on selecting item i is *the total number of ways to select items* 0 *to* i-1,

   which is _____

   - Accordingly, the answers of selecting item i may result in different values, yes or no?
   - Consequently, can we memoize this brute-force code for speed-up?

Ans:

2. Issues with the Brute-Force Code

When deciding on item i, there is no information on how the previous items (0 to $i-1$) were chosen.

The total number of ways to select items 0 to $i-1$ is 2^i.

- Accordingly, the answers of selecting item i may result in different values.
- Therefore, we can memoize this brute-force code for speed-up.

5. Observe. Is each state generated by the brute-force algorithm unique? In other words, are there repeated occurrences of the same state ?

6. Given a problem state, does any recursive call on this state return the same value? Why?

Ans:

5. Is each state unique?

Yes, each state defined by the pair `(i, c)` is unique because it represents a specific item being considered and a specific remaining capacity.

6. Do recursive calls on the same state return the same value?

Yes, because the state `(i, c)` is deterministic in this problem. It always yields the same result for the same inputs.

8. Comparing the Number of Recursive Calls

By memoizing, we reduce the number of recursive calls significantly because we avoid recalculating results for the same states.

2. Beginning State

The beginning state is when both strings are at their first character (`i = 0`, `j = 0`).

3. If `A` runs out (`i == len(A)`), but `B` has not (`j < len(B)`):

The additional edit distance required is the number of remaining characters in `B`.

4. If `B` runs out, but `A` has not:

The additional edit distance required is the number of remaining characters in `A`.

1. **Given v1 ≥ v2:**
   1.1. **Which recursive call is made first**: `mincoin(v1)` or `mincoin(v2)`?
   - The call order depends on the implementation, but typically the smaller value (`v2`) is processed first to minimize the amount more efficiently.
   1.2. **Which function returns first**: `mincoin(v1)` or `mincoin(v2)`?
   - The function for the smaller amount (`v2`) is likely to return first as it's quicker to compute.
   1.3. **Which `mm` entry obtains its final value first**: `mm[v1]` or `mm[v2]`?
   - `mm[v2]` obtains its final value first because it's computed first.
2. **Non-recursive Solution**:
   - Use dynamic programming to fill an array `mm` where `mm[i]` represents the minimum number of coins needed for amount `i`.

Questions and Answers

1. **Observation of Recursive Calls**:
   - The function `maxVal(i, C)` requires the return values of `maxVal(i+1, C)` and `maxVal(i+1, X)` where `X < C`.
   - This shows the dependencies among the subproblems, suggesting a way to solve the problem using a bottom-up approach with dynamic programming.
2. **Non-recursive Version Using Dynamic Programming**:
   - By looping through the items and capacities in a specific order, we can fill up a table of solutions to subproblems, ensuring all dependencies are pre-computed before they are needed.

Understanding the Brute Force Approach

Definition

The brute force approach, in the context of problem-solving and algorithms, refers to a straightforward and simple method for solving a problem. It systematically explores all possible solutions to find the correct one. This method is often easy to implement but can be inefficient and impractical for large input sizes due to its high time complexity.

Characteristics

1. **Exhaustive Search**: Brute force algorithms explore every possible combination or permutation to find the solution.

2. **Simplicity**: These algorithms are usually straightforward and easy to understand. They do not require complex data structures or advanced algorithms.

3. **Guaranteed Correctness**: Since brute force checks all possibilities, it guarantees finding the correct solution if one exists.

4. **Inefficiency**: The main drawback of brute force methods is their inefficiency, especially with large datasets. They often have exponential time complexity (e.g., $O(n!)$, $O(2^n)$), making them impractical for real-world applications with large input sizes.

Example

Consider the problem of finding the maximum sum of a sub-rectangle in a 2D array. A brute force approach would involve:

1. Generating all possible sub-rectangles.

2. Calculating the sum of elements in each sub-rectangle.

3. Keeping track of the maximum sum found.

This approach, while guaranteed to find the correct solution, involves a large number of iterations. For an $N \times N$ matrix, there are $O(N^4)$ possible sub-rectangles, resulting in an extremely high computational cost for larger N.

Steps in Brute Force

1. **Generate All Possibilities**: Identify and enumerate all potential solutions to the problem.

2. **Evaluate Each Solution**: Check each possible solution to see if it meets the criteria or constraints of the problem.

3. **Select the Best Solution**: Compare all evaluated solutions and select the one that optimizes the required outcome (e.g., maximum sum, shortest path).

Applications

1. **Search Algorithms**: Finding an element in an unsorted list.

2. **Optimization Problems**: Knapsack problem, traveling salesman problem.

3. **Pattern Matching**: Searching for a substring in a string.

Advantages and Disadvantages

Advantages:

- **Simplicity**: Easy to implement and understand.

- **General Applicability**: Can be applied to a wide range of problems without the need for problem-specific optimizations.

- **Guaranteed Solution**: Always finds a solution if one exists.

Disadvantages:

- **Inefficiency**: High time complexity makes it impractical for large datasets.

- **Resource Intensive**: Consumes significant computational resources, both in terms of time and memory.

- **Not Scalable**: Does not scale well with input size, leading to exponential growth in computation time.

Conclusion

The brute force approach is a fundamental method in problem-solving that guarantees finding the correct solution by exhaustively exploring all possibilities. While it is simple and

guarantees correctness, its inefficiency limits its use to small or moderately sized problems. Understanding the brute force method is crucial as it provides a baseline against which more sophisticated and efficient algorithms can be compared and developed. In an academic context, demonstrating knowledge of brute force techniques shows a foundational understanding of algorithmic problem-solving, which is essential for grasping more advanced concepts.