Five Kinds of Algorithms

CMPT 145

Algorithms

- An algorithm is a sequence of instructions that accomplish a stated task.
- Example tasks:
    - Calculate the average of a collection of numbers
    - Calculate the square root of a number
    - Check if a binary tree is ordered.

How do you design an algorithm
if you do not already know
how the algorithm should work?

Study algorithms designed by someone else.

Algorithms Unit Overview

1. Tasks: What kinds of tasks do we write algorithms for?
2. Algorithm Styles: What kinds of algorithms are there?
3. Examples: We study example algorithms for a variety of tasks.

Solutions are constructed by making choices

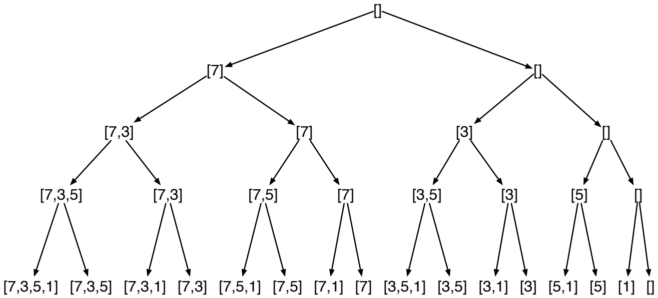
- Average of a list: choose all numbers to construct a sum
- Itinerary: choose flights to construct a connecting itinerary.
- Subset Sum: choose some values to construct a sum
- Maximum Slice: choose indices to construct a slice
- Making Change: choose coins to construct list
- Leap line: Choose to step or jump to construct a sequence.

Algorithms construct a solution by exploring the possible choices.

Exploring choices

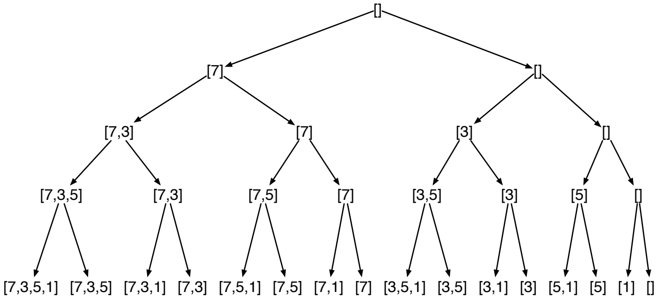
- To solve a search task, our algorithms have to explore possible choices.
- Exploring choices is like exploring a tree.
  - The root is where no choices have been made yet.
  - The leaf nodes are where all choices have been made; these are all the possibilities.
  - Other nodes are intermediate stages where some choices have been made, other choices remain to be made.
- This is not always a tree we construct in the heap.
- This is often a conceptual tree that our algorithms explore!

Example: Tree of Choices Subset Sum



A tree of all the possible subsets of the list $[1, 3, 5, 7]$. Each left branch includes one of the elements; each right branch leaves it out. The number of levels is $N + 1$. The number of leaf nodes is 2^N.

Example: Tree of Choices Subset Sum



An algorithm to solve Subset Sum could traverse the entire tree, looking for one leaf node whose sum is T. Generating all possible options is called Brute Force

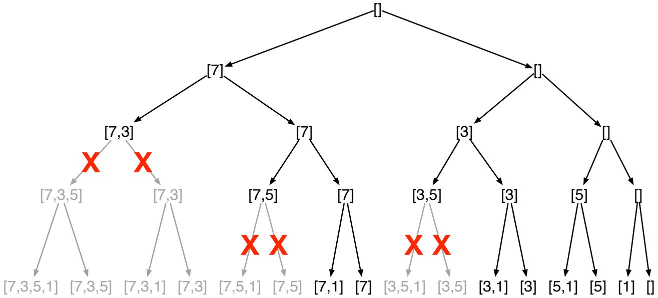
Brute Force

- Generate all possible values one at a time.
- Stop when you find one that satisfies the requirements.
- Examples:
  - Subset Sum: Generate all possible subsets, one at a time, and check.
  - Maximum Slice: Generate all slices, and check.
  - Make Change: Generate all possible coin sets, one at a time, and check.
  - Leap Line: Generate all possible sequences of step/jump, and check.

Brute Force takes too long!

- Generate all possible values one at a time.
- Stop when you find one that satisfies the requirements.
- Problem: All possible is too many.
- Examples:
  - Subset Sum: How many possible subsets?
  - Maximum Slice: How many slices?
  - Make Change: How many coin sets?
  - Leap Line: How many sequences?
- Even if checking each possible value is $O(1)$, the total time to check them all is too high!

Example: Tree of Choices Subset Sum

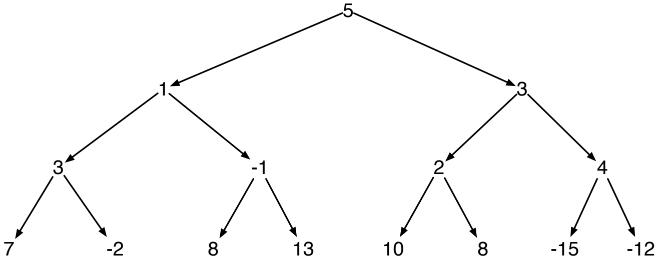


Some choices invalidate all future choices. You can prevent exploration of all choices by checking before you reach a leaf node. This is called Backtracking.

Backtracking

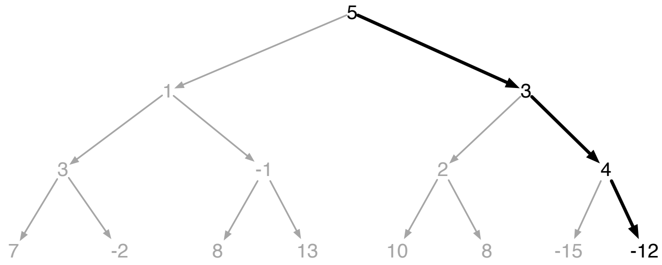
- Generate all possibilities, except those that won't satisfy the requirements.
- Key: We can generate possibilities by making choices one at a time.
  - If a single choice invalidates the possibility, throw it away, and make a different choice.
- Examples:
  - Subset Sum: Build M adding values one at a time from L. Discard M if $sum(M) > T$.
  - Maze Solver: Try all possible open locations. Stop if you are blocked in.

Example: Maximum Path



An algorithm to solve Maximum Path could traverse the entire tree, looking for the path with the highest sum. Looking at all the paths is called Brute Force

Example: Maximum Path



An algorithm to solve Maximum Path could try to make a smart choice, and ignore other options. This is called Greedy. It doesn't work well here, but it can be very good with other problems.

Greedy

- Make choices that seem pretty good.
- Examples:
  - Make change: try high value coins first.
  - Huffman Tree Construction: Choose the two lowest frequency trees.
- Greedy algorithms do not always find the right answer, but when they do, it's great!

Example: Sorting a list

- Can you think of a Brute Force algorithm to sort a list of numbers?
- Backtracking?
- Good algorithms for sorting are Divide and Conquer algorithms.

Example: Sorting a list

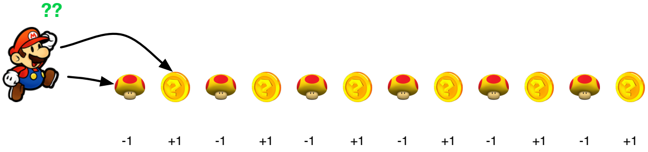
```python
def quiksort(alist):
    if len(alist) == 0:
        return []
    else:
        pivot = alist[0]
        smaller = [x for x in alist if x <  pivot]
        equal   = [x for x in alist if x == pivot]
        greater = [x for x in alist if x >  pivot]
        return quiksort(smaller) + equal + quiksort(greater)
```

The problem is divided into sub problems, and solved. The solutions are combined. The result is a sorted list. This is Divide and Conquer

Divide and Conquer

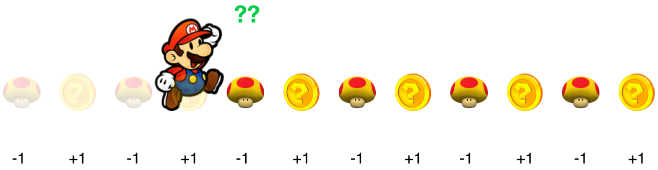
- Split the task into smaller sub-tasks.
- Examples:
  - Sorting a list.
  - Binary search in a list.
  - Looking for a value in a binary search tree.

Example: Leap Line



Mario has to choose the best option now. If he explores both, it's Brute Force.

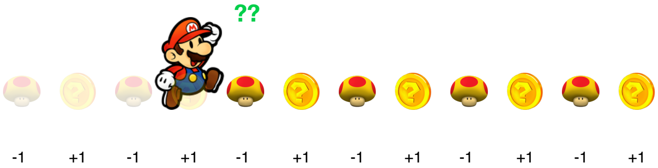
Example: Leap Line



At some time in the future, Mario will face a similar choice.

As Mario explores the choices for the best sequence, he will arrive at this choice point 4 different ways from the start.

Example: Leap Line



Mario should calculate the best sequence of choices from here to the end once, and then save it. Every other time he explores this sub-problem, he can look up the answer he saved. This is called Dynamic programming.

Dynamic Programming

- Avoid repeated exploration of future choices by solving the problem once, and saving the result.
- The data structure used to save the results is called a memo. In Python, use a dictionary!
- Dynamic programming is often as simple as Backtracking + memo.
- Examples:
  - Leap Line: remember the optimal sequence for shorter lines.
  - Maximum Slice: Remember the sums of slices you're exploring; reuse the values, don't recalculate.

Algorithm Styles

- Brute Force: Generate all possibilities one at a time. Stop when you find one that satisfies the requirements.
- Backtracking: Generate all possibilities, except those that won't satisfy the requirements.
- Greedy: Make choices that seem pretty good, but don't try alternatives.
- Divide and Conquer: Split the task into smaller sub-tasks.
- Dynamic Programming: Backtracking with memo-ization.

All these styles can be applied to search, decision, and optimization tasks.