Data Structures and Algorithms

LECTURE 07: GREEDY ALGORITHMS

UNIVERSITY of GREENWICH
Alliance with FPT Education

Pearson BTEC

Contents

- Greedy Algorithms
- Greedy Failure Cases
- Optimal Greedy Algorithms

Greedy Algorithms

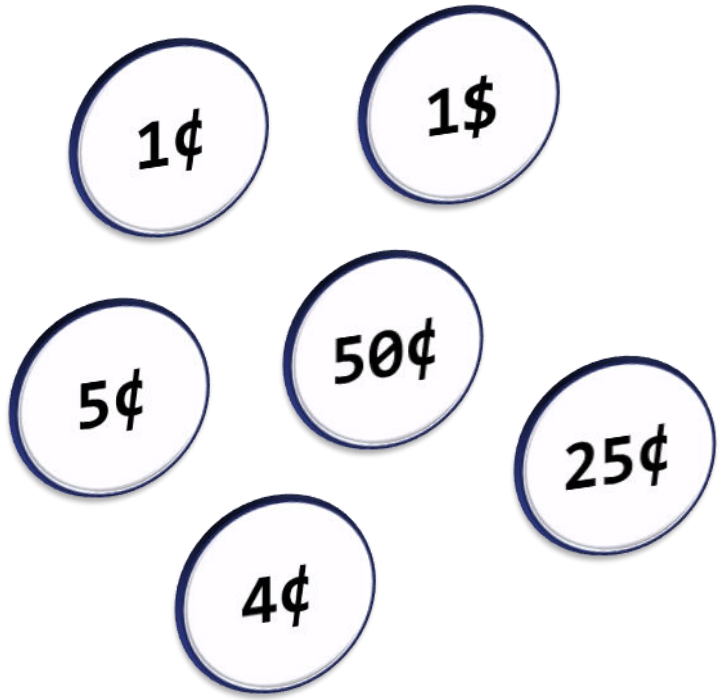
- Used for solving optimization problems
- Usually more efficient than the other algorithms
- Can produce a **non-optimal** (incorrect) result
- Pick the **best local** solution
  - The optimum for a **current** position and point of view
- Greedy algorithms assume that always choosing a **local** optimum leads to the **global** optimum

Optimization Problems

- Finding the best solution from all possible solutions
- Examples:
  - Find the **shortest** path from Sofia to Varna
  - Find the **maximum increasing subsequence**
  - Find the shortest route that visits each city and returns to the origin city

Problem: Sum of Coins

- Write a program, which gathers a sum of money, using the least possible number of coins
- Consider the US **currency coins**
  - **0.01**, **0.02**, **0.05**, **0.10**
- **Greedy algorithm** for "Sum of Coins":
  - Take the largest coin while possible
  - Then take the second largest
  - Etc.



Sum of Coins Visualization

Target: 18

Actual: 0

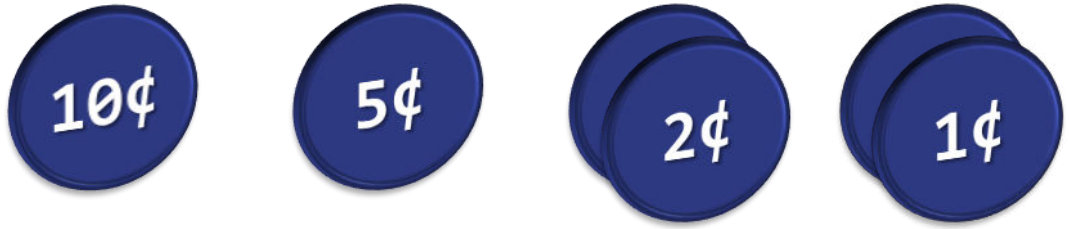
Sum of Coins Visualization

Target: 18

Actual: 10

Sum of Coins Visualization

Target: 18



Actual: 15

Sum of Coins Visualization

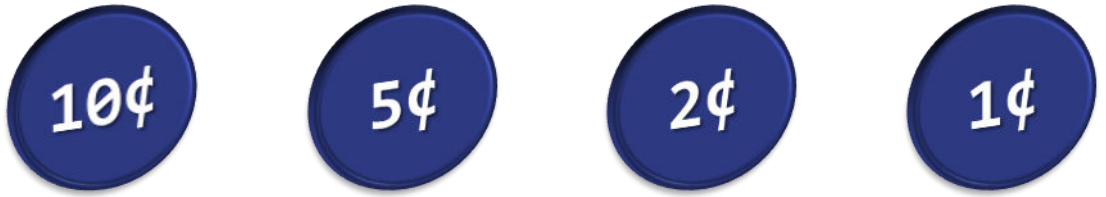
Target: 18

Actual: 17

Sum of Coins Visualization

Target: 18

Actual: 18



Solution: Sum of Coins (1)

```
public static Map<Integer, Integer>
                        chooseCoins(int[] coins, int targetSum) {
  List<Integer> sortedCoins = Arrays.stream(coins).boxed()
              .sorted(Collections.reverseOrder())
              .collect(Collectors.toList());
  Map<Integer, Integer> chosenCoins = new LinkedHashMap<>();
  int currentSum = 0; int coinIndex = 0;
  // Next slide
  if (currentSum != targetSum)
    throw new IllegalArgumentException();
  return chosenCoins;
}
```

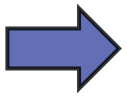
Solution: Sum of Coins (2)

```
while (currentSum != targetSum && coinIndex < sortedCoins.size()) {
  int currentCoin = sortedCoins.get(coinIndex);
  int remainder = targetSum - currentSum;
  int numberOfCoins = remainder / currentCoin;
  if (currentSum + currentCoin <= targetSum) {
    chosenCoins.put(currentCoin, numberOfCoins);
    currentSum += numberOfCoins * currentCoin;
  }
  coinIndex++;
}
```

Problem: Set Cover

- Write a program that finds the smallest subset of S, the union of which = **U** (if it exists)
- You will be given a **set** of integers **U** called "**the Universe**"
- And a set **S** of **n** integer sets whose union = **U**

```
Universe: 1, 2, 3, 4, 5
Number of sets: 4
1
2, 4
5
3
```



```
Sets to take (4):
{ 2, 4 }
{ 1 }
{ 5 }
{ 3 }
```

Solution: Set Cover (1)

```
public static List<int[]> chooseSets(
                List<int[]> sets, List<Integer> universe) {
  List<int[]> selectedSets = new ArrayList<>();
  Set<Integer> universeSet = new HashSet<>();
  for (int element : universe) { universeSet.add(element);}
  while (!universeSet.isEmpty()) {
    // Next Slide
  }
  return selectedSets;
}
```

Solution: Set Cover (2)

```
int notChosenCount = 0;
int[] chosenSet = sets.get(0);
for (int[] set : sets) {
  // Next slide
}
selectedSets.add(chosenSet);
for (int elem : chosenSet) {
  universeSet.remove(elem);
}
```

Solution: Set Cover (3)

```
int count = 0;
for (int elem : set) {
  if (universeSet.contains(elem)) {
    count++;
  }
}
if (notChosenCount < count) {
  notChosenCount = count;
  chosenSet = set;
}
```

Check your solution here: https://judge.softuni.bg/Contests/2454/Searching-Sorting-and-Greedy-Algorithms-Lab

Greedy Failure Cases

- Greedy Algorithms Often Fail

Sum of Coins Failure

Target: 18

Actual: 0

Sum of Coins Failure

Target: 18

Actual: 10

Sum of Coins Failure

Target: 18

Actual: 15

Sum of Coins Failure

Target: 18

Actual: 16

Sum of Coins Failure

Target: 18

Actual: 17

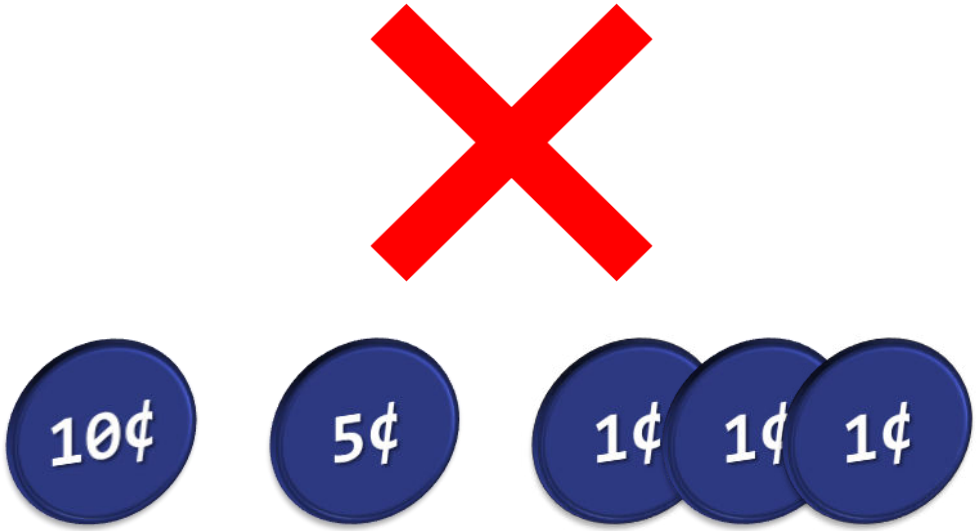
Sum of Coins Failure

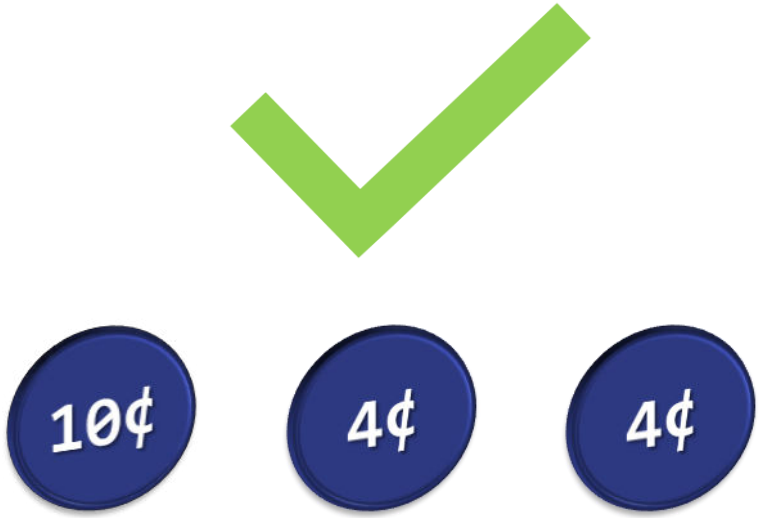
Target: 18

Actual: 18

Sum of Coins Failure

Target: 18





UNIVERSITY of GREENWICH
Alliance with FPT Education

Pearson BTEC

Optimal Greedy Algorithms

- Optimal Substructure and Greedy Choice Property

UNIVERSITY of GREENWICH
Alliance with FPT Education

Pearson BTEC

Optimal Greedy Algorithms

- Suitable problems for greedy algorithms have these properties:
  - **Greedy choice property**
  - **Optimal substructure**
- Any problem having the above properties is guaranteed to have an optimal greedy solution

Greedy Choice Property

- **Greedy choice** property
  - **A global optimal solution** can be obtained by greedily selecting a **locally optimal** choice
  - Sub-problems that arise are solved by consequent greedy choices
    - Enforced by optimal substructure

Optimal Substructure Property

- **Optimal substructure** property
  - After each greedy choice the problem remains an optimization problem of the same form as the original problem
  - **An optimal global solution contains the optimal solutions of all its sub-problems**

Greedy Algorithms: Example

- The "**Max Coins**" game
  - You are given a set of coins
  - You play against another player, alternating turns
  - Per each turn, you can take up to three coins
  - Your goal is to have as many coins as possible at the end

Max Coins – Greedy Algorithm

- A simple **greedy strategy** exists for the "Max Coins" game

```
At each turn take the maximum number of coins
```

- Always choose the local maximum (at each step)
  - You don't consider what the other player does
  - You don't consider your actions' consequences
- The **greedy algorithm** works optimally here
  - It takes as many coins as possible

Summary

- Greedy Algorithms
- Optimal Greedy Algorithms