Data Structures II : Greedy algorithms

Mauricio Toro
Department of Systems and Informatics
Universidad EAFIT

Cocktail of the day: Grasshoper

Disclaimer: Keep alcohol out of the hands of minors.

Cocktail of the day: Grasshoper

- 30 ml Crème de menthe
- 30 ml Crème de cacao
- 30 ml Fresh cream

Machine improvisation uses string processing

https://www.youtube.com/watch?v=2jFpGQbrcag

Greedy algoritms

- A greedy algorithm is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.
- In many problems, a greedy strategy does not in general produce an optimal solution.

Taken from Wikipedia.

Examples of Greedy Algorithms

- Dijkstra's Algoritm
- Prim's Algoritm
- Approximations:
  - Traveling salesman problem
  - Change-making problem

Change-making problem

- The change-making problem addresses the following question:
  - "how can a given amount of money be made with the least number of coins of given denominations?"
- It is a knapsack type problem, and has applications wider than just currency.

Applications

- Cryptography: Public-key system based on subset-sum
- Cryptography: Computer passwords
- Cryptography: Message verification
- Challenges: Dropbox Challenge
  `http://www.skorks.com/2011/02/algorithms-a-dropbox-challenge-and-dynamic-programming/`

Taken from `www.math.stonybrook.edu/~scott/blair/Subset_Sum_Knapsack_problem.htm`

```
change (n, X[1..k], i):
    if n = 0 then print X[1..i]
    else
        for coin ∈ {100, 200, 500, 100} do
            remainder ← X[i] − coin
            if remainder ≥ 0 then
                    X[i + 1] ← coin
                    change (remainder, X[1..k], i+1)
```

Change-making problem example

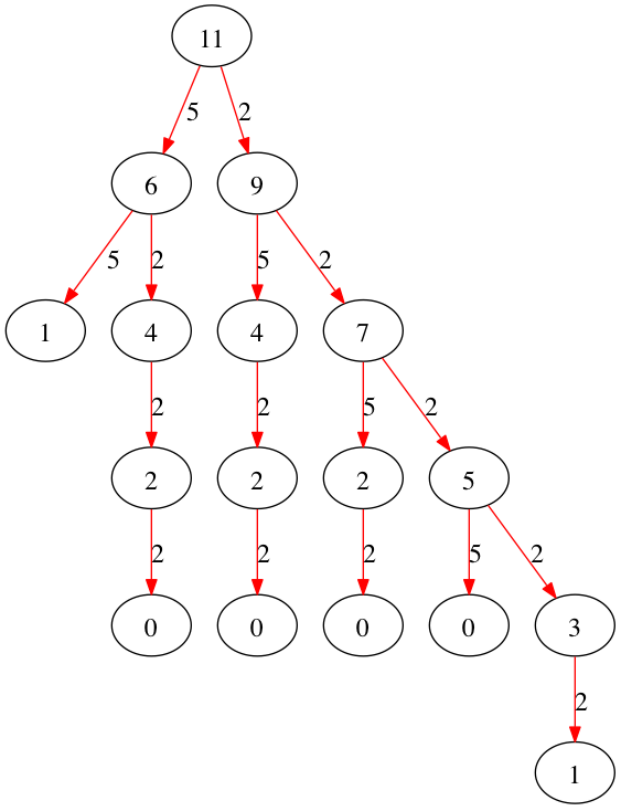


Figure: Change 11 cents with 2-cent and 5-cent coins.

Complexity of the Change-making problem

- The complexity is $O(k^n)$ using brute force, where k is the number of different coins and n the amount of money to be changed.

```
public static final int[] values =
          { 200, 100, 50, 20, 10, 5, 2, 1 };
public static final String coins =
          {"2 euros", "1 euro", "50 cent",...};

public static String coinsChange (int cent) {
   int remainder = cent, count = 0;
   String change = ""+ cent + "cent can be returned as: \n";
   for (int i=0; i<coins.length; i++){
   {    while (remainder > values[i]){
        {  remainder -= values[i]; count ++;  }
       change += "\t" + count +" x" + coins[i] + "\n";
       count = 0;
   }
}
```

will this return the optimum?

Complexity of the Greedy Change-making

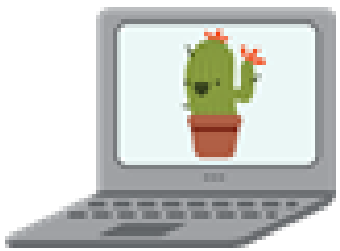
The complexity is $O(k)$.

Single-source shortest path problems

The problem is to determine the cost of the shortest path from the source to every other vertex in V, where the length of a path is just the sum of the costs of the arcs on the path.

Dijkstra's Algorithm

- A greedy algorithm to solve the single-source shortest path problem.
- Dijkstra's algorithm does not work when weights on the arcs are negative numbers.
- Algorithm can be found here: `http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm`



Example of Dijkstra's Algorithm

Simulator:
`https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html`

Running time over adjacency matrices

- Suppose Dijkstra's algorithm operates on a digraph with n vertices and e edges.
- If we use an adjacency matrix to represent the digraph, then the inner loop takes $O(n)$ time, and it is executed n-1 times for a total time of $O(n^2)$.
- The rest of the algorithm is easily seen to require no more time than this.

Running time over adjacency lists

http://www.geeksforgeeks.org/
greedy-algorithms-set-7-dijkstras-algorithm-for-adjace

Traveling salesman problem (TSP)

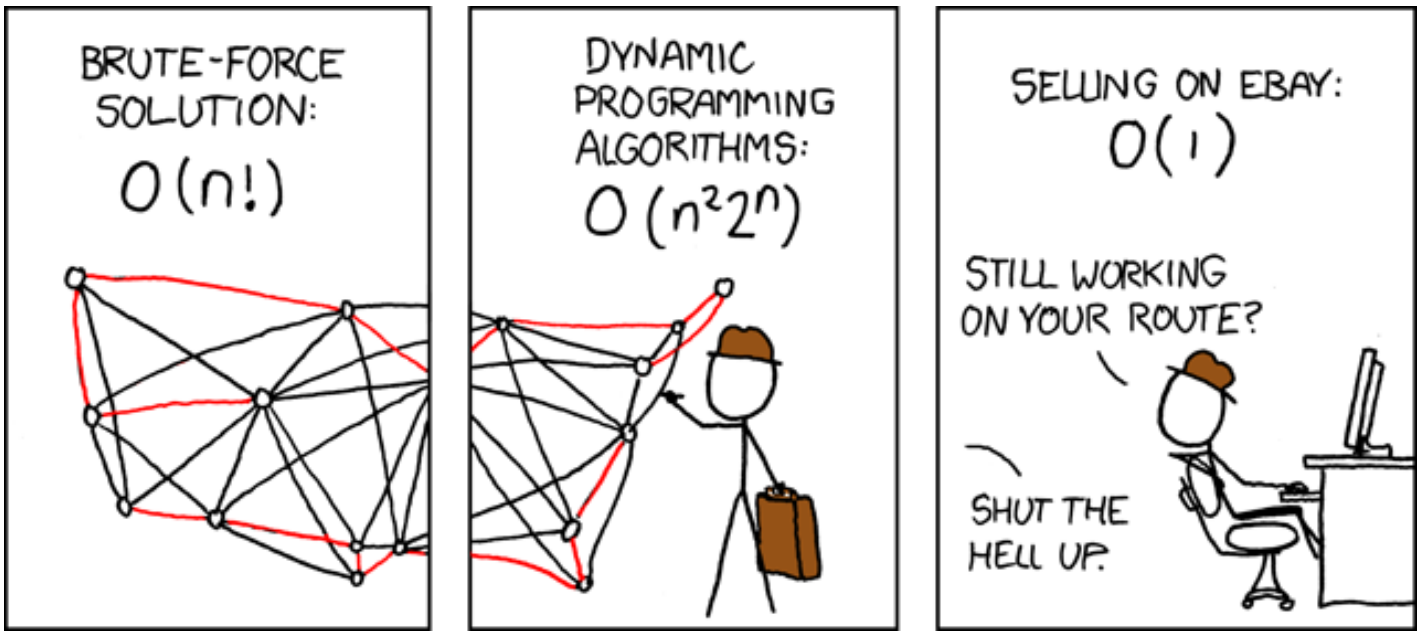


Figure: Traveling salesman comic

TSP's applications

- Electronic circuit design
- Collection of coins from payphones
- Vehicle routing problems

Taken from `http://www.math.uwaterloo.ca/tsp/apps/`

Traveling salesman problem (2)

- For example, a greedy strategy for the traveling salesman problem is the following heuristic:
  - "At each stage visit an unvisited city nearest to the current city".
- This heuristic need not find a best solution, but terminates in a reasonable number of steps; finding an optimal solution typically requires unreasonably many steps.

References

- Please how to reference images, trademarks, videos and fragments of code.
- Avoid plagiarism

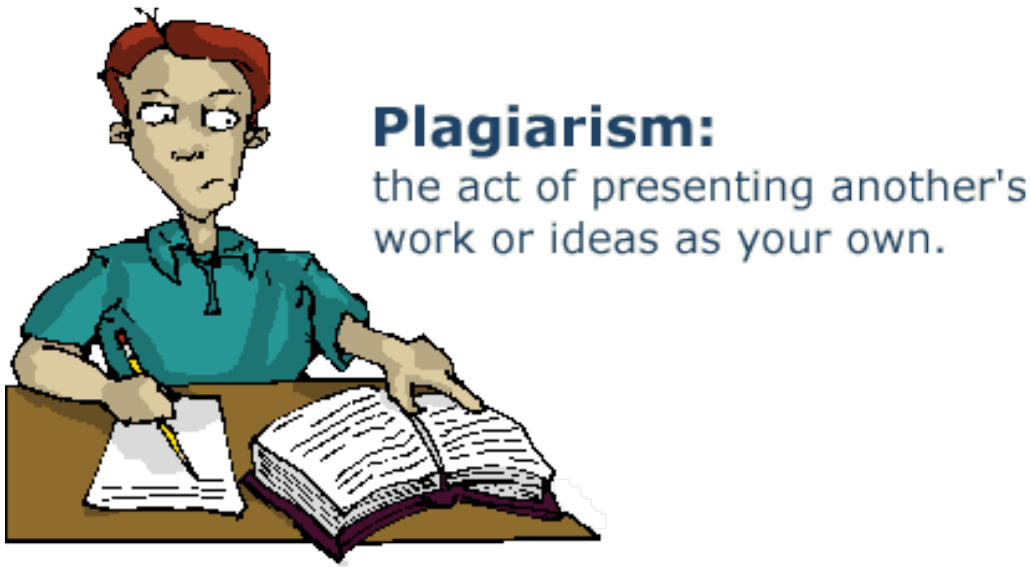


Figure: Figure about plagiarism, University of Malta [Uni09]

References

University of Malta.

Plagarism — The act of presenting another's work or ideas as your own, 2009.

[Online; accessed 29-November-2013].

Further reading

- R.C.T Lee, Introduction to the analysis and design of algorithms, Chapter 3, Pages 71 - 115.