Greedy Algorithms

Greed is good.
(Some of the time)

Outline

- Elements of greedy algorithm
  - Greedy choice property
  - Optimal substructures
- Minimum spanning tree
  - Kruskal's algorithm
  - Prim's algorithm
- Huffman code
- Activity selection

Introduction

- Concepts
  - Choosing the best possible choice at each step.
  - This decision leads to the best over all solution.
- Greedy algorithms do not always yield optimal solutions.

Elements of Greedy Algorithms

Greedy-choice property

- A globally optimal solution is derived from a locally optimal (greedy) choice.
- When choices are considered, the choice that looks best in the current problem is chosen, without considering results from subproblems.

Optimal substructures

- A problem has ***optimal substructure*** if an optimal solution to the problem is composed of optimal solutions to subproblems.
- This property is important for both greedy algorithms and dynamic programming.

Greedy Algorithm v.s. Dynamic Programming

Greedy algorithm

- The best choice is made at each step and after that the subproblem is solved.
- The choice made by a greedy algorithm may depend on choices so far, but it *cannot depend on* any future choices or on the solutions to subproblems.
- A greedy strategy usually progresses in a *top-down* fashion, making one greedy choice after another, reducing each given problem instance to a smaller one.

Dynamic programming

- A choice is made at each step.
- The choice made at each step usually *depends on* the solutions to subproblems.
- Dynamic-programming problems are often solved in a *bottom-up* manner.

Steps in Design Greedy Algorithms

- Determine the optimal substructure of the problem.
- Develop a recursive solution.
- Prove that at any stage of the recursion, one of the optimal choices is the greedy choice. Thus, it is always safe to make the greedy choice.
- Show that all but one of the subproblems induced by having made the greedy choice are empty.
- Develop a recursive algorithm that implements the greedy strategy.
- Convert the recursive algorithm to an iterative algorithm.

Shortcuts Design

- Form the optimization problem so that, after a choice is made and there is only one subproblem left to be solved.
- Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe.
- Demonstrate that, having made the greedy choice, what remains is a subproblem with the property that if we combine an optimal solution to the subproblem with the greedy choice we have made, we arrive at an optimal solution to the original problem.