Greedy Algorithm

Bibhudatta Sahoo,
National Institute of Technology Rourkela

Introduction

- **Optimization problem**: there can be many possible solution. Each solution has a value, and we wish to find a solution with the optimal (minimum or maximum) value

- **Greedy algorithm**: an algorithmic technique to solve optimization problems
  - Always makes the choice that looks best at the moment.
  - Makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution

Optimization Problems

A problem that may have many feasible solutions.

Each solution has a value

In maximization problem, we wish to find a solution to maximize the value

In the minimization problem, we wish to find a solution to minimize the value

Greedy algorithms

An optimization problem is one in which you want to find, not just *a* solution, but the *best* solution

A "greedy algorithm" sometimes works well for optimization problems

A greedy algorithm works in phases: At each phase:

- You take the best you can get right now, without regard for future consequences
- You hope that by choosing a *local* optimum at each step, you will end up at a *global* optimum

Example: Counting money

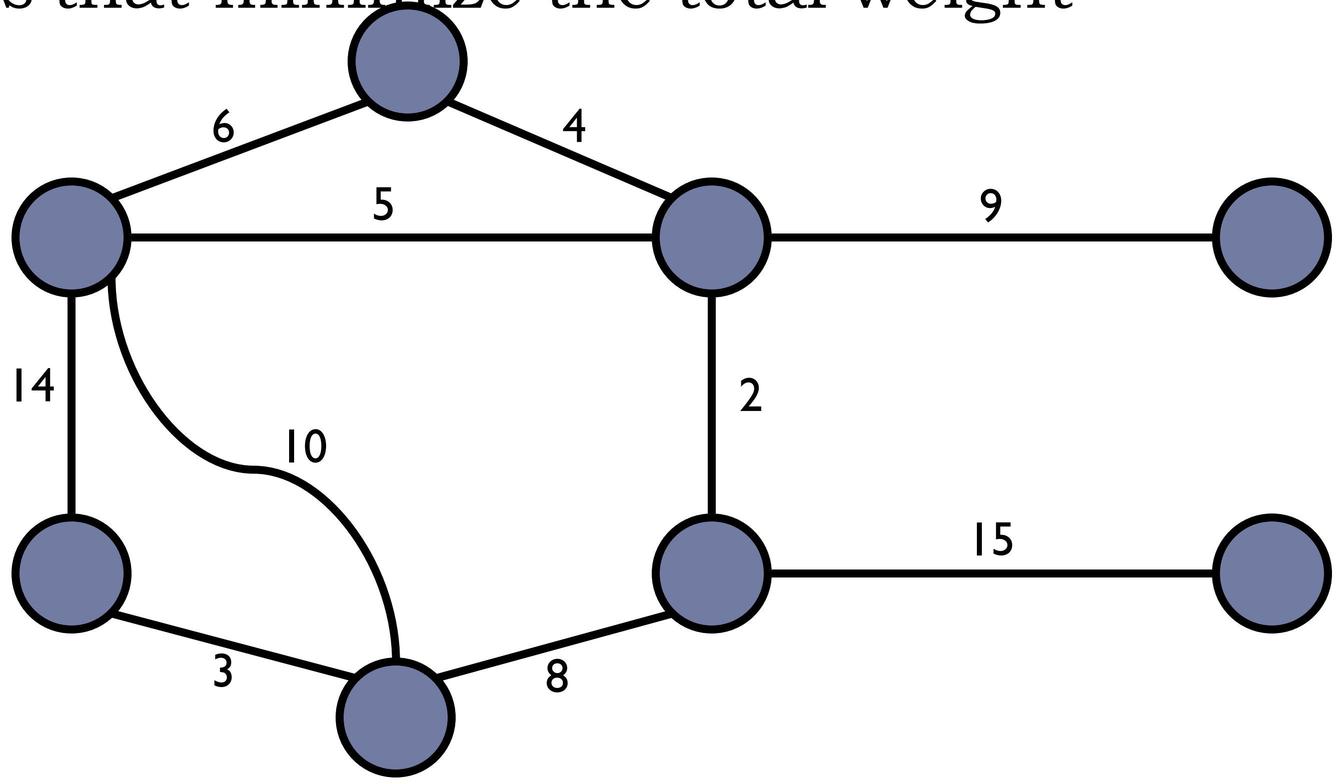
- Suppose you want to count out a certain amount of money, using the fewest possible coins
- A greedy algorithm would do this would be: At each step, take the largest possible coin that does not overshoot
  - Example: To make Rs6.50, you can choose:
    - a Rs. 5 coin
    - a Rs. 1 coin , to make Rs. 6
    - a 50 paisa coin, to make Rs.6.50
- **For INDIAN money, the greedy algorithm always gives the optimum solution**

Minimum Spanning tree

- A *spanning tree* of a graph is just a sub-graph that contains all the vertices and is a tree.

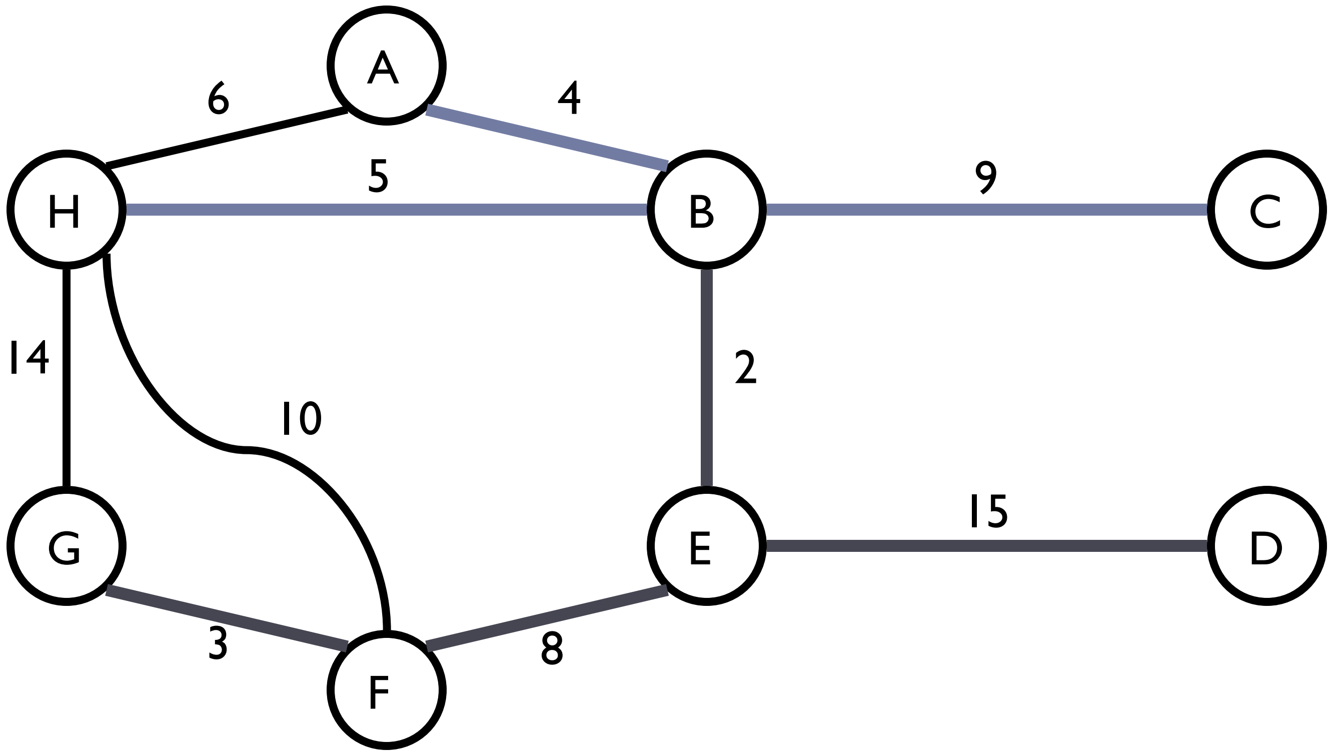
Example: Minimum Spanning Tree

- Problem: given a connected, undirected, weighted graph, find a *spanning tree* using edges that minimize the total weight



Example: Minimum Spanning Tree

- Answer:



Greedy Method: Overview

- Optimization problems: terminology
  - Solutions judged on some criteria:

    Objective function

  Example: Sum of edge weights in path is smallest
  - A solution must meet certain constraints

    A solution is *feasible*

  Example: All edges in solution are in graph, form a simple path
  - One (or more) feasible solutions that scores highest (by the objective function) is the *optimal solution(s)*

Greedy Method: Overview

- Greedy strategy:
  - Build solution by stages, adding one item to partial solution found so far
  - At each stage, make locally optimal choice based on the *greedy rule* (sometimes called the *selection function)*
    - Locally optimal, I.e. best given what info we have now
  - Irrevocable, a choice can't be un-done
  - Sequence of locally optimal choices leads to globally optimal solution (hopefully)
    - Must prove this for a given problem!
    - Approximation algorithms, heuristics

Greedy algorithm for optimization problem

Greedy algorithms **do not always** yield a genuinely optimal solution. In such cases the greedy method is frequently the basis of a *heuristic approach*.

Even for problems which can be solved exactly by a greedy algorithm, establishing the **correctness** of the method may be a non-trivial process.

In order to give a precise description of the greedy paradigm we must first consider a more detailed definition of the environment in which typical optimization problems occur.

What an optimisation problem will have , in the context of greedy algorithms

- A collection (set, list, etc) of **candidates**, e.g. nodes, edges in a graph, etc.
- A set of candidates which have already been `used'.
- A **predicate** (*solution*) to test whether a given set of candidates give a *solution* (not necessarily optimal).
- A predicate (*feasible*) to test if a set of candidates can be **extended** to a (not necessarily optimal) solution.
- A **selection function** (*select*) which chooses some candidate which h as not yet been used.
- An **objective function** which assigns a *value* to a solution.

Finding Solution to optimization Problem

- An optimisation problem involves finding a subset, *S*, from a collection of candidates, *C*; the subset, *S*, must satisfy some specified criteria, i.e. be a solution and be such that the *objective function* is optimised by *S*.
- *`Optimised'* may mean **Minimised or Maximised**
- Depending on the precise problem being solved. Greedy methods are distinguished by the fact that the **selection function** assigns a *numerical value* to each candidate, *x*, and chooses that candidate for which:
  - ***SELECT(x)*** **is largest**
  - or *SELECT(x)* is **smallest**

Elements of Greedy Strategy

- An greedy algorithm makes a sequence of choices, each of the choices that seems best at the moment is chosen
  - NOT always produce an optimal solution
- Two ingredients that are exhibited by most problems that lend themselves to a greedy strategy
  - Greedy-choice property
  - Optimal substructure

Greedy-Choice Property

- A globally optimal solution can be arrived at by making a locally optimal (greedy) choice
  - Make whatever choice seems best at the moment and then solve the sub-problem arising after the choice is made
  - 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 sub-problems
- Of course, we must prove that a greedy choice at each step yields a globally optimal solution

Optimal Substructures

- A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to sub-problems
  - If an optimal solution A to S begins with activity 1, then A' = A – {1} is optimal to S'={i ∈S: $s_i \geq f_1$}

is

```
--**************************************************
function greedy (C : candidate_set) return candidate_set
is
x : candidate;
S : candidate_set;
begin
  S := {};
  while (not solution(S)) and C /= {} loop
    x := select( C );
    C := C - {x};
    if feasible( S union {x} ) then
      S := S union { x };
    end if;
  end loop;
  if solution( S ) then
    return S;
  else
    return es;
  end if;
end greedy;
```

function select (C : candidate_set)
return candidate;
function solution (S : candidate_set)
return
boolean;
function feasible (S : candidate_set)
return
boolean;

Knapsack Problem

- One wants to pack *n* items in a luggage
  - The *i*th item is worth p_i rupees and weighs w_i *kg.*
  - Maximize the value but cannot exceed *W* kg
  - p_i , w_i, *W* are integers
- 0-1 knapsack → each item is taken or not taken
- Fractional knapsack → fractions of items can be taken
- Both exhibit the optimal-substructure property
  - 0-1: If item *j* is removed from an optimal packing, the remaining packing is an optimal packing with weight at most $W\text{-}w_j$
  - Fractional: If *w pounds* of item *j* is removed from an optimal packing, the remaining packing is an optimal packing with weight at most *W-w* that can be taken from other *n-1* items plus $w_j - w$ of item *j*

Greedy Algorithm for Fractional Knapsack problem

- Fractional knapsack can be solvable by the greedy strategy
  - Compute the value per pound v_i/w_i for each item
  - Obeying a greedy strategy, take as much as possible of the item with the greatest value per pound.
  - If the supply of that item is exhausted and there is still more room, take as much as possible of the item with the next value per pound, and so forth until there is no more room
  - $O(n \lg n)$ (we need to sort the items by value per pound)
  - Greedy Algorithm?
  - Correctness?

0-1 knapsack is harder!

- 0-1 knapsack cannot be solved by the greedy strategy
  - Unable to fill the knapsack to capacity, and the empty space lowers the effective value per pound of the packing
  - We must compare the solution to the sub-problem in which the item is included with the solution to the sub-problem in which the item is excluded before we can make the choice
  - Dynamic Programming

Typical Steps

- Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.
- Prove that there's always an optimal solution that makes the greedy choice, so that the greedy choice is always safe.
- Show that greedy choice and optimal solution to subproblem $\Rightarrow$ optimal solution to the problem.
- Make the greedy choice and **solve top-down**.
- May have to preprocess input to put it into greedy order.
  - **Example:** Sorting the edges by weight to find MST.

Elements of Greedy Algorithms

- Greedy-choice Property.
  - A globally optimal solution can be arrived at by making a locally optimal (greedy) choice.
- Optimal Substructure.

Elements of Greedy Algorithms

- Computer Scientists consider Greedy paradigm as a general design technique despite the fact that it is applicable to optimization problem.
- The Greedy techniques suggests constructing a solution to an OPTIMIZATION problem through a sequence of steps, each steps expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached.
-
- **The choice made at each step must be:**
-
- # **Feasible**: It has to satisfy the problem's constraints
-
- # **Locally optimal**: It has to be the best local choice among all feasible choices available on that step.
-
- # **Irrevocable**: once made, it cannot be changed on subsequent steps of the algorithm

Form of Optimization problem in the context of Greedy algorithm

- A collection (set, list, etc) of **candidates**, e.g. nodes, edges in a graph, etc.
- A set of candidates which have already been `used'.
- A **predicate** (*solution*) to test whether a given set of candidates give a *solution* (not necessarily optimal).
- A predicate (*feasible*) to test if a set of candidates can be **extended** to a (not necessarily optimal) solution.
- A **selection function** (*select*) which chooses some candidate which h as not yet been used.
- An **objective function** which assigns a *value* to a solution.

Defining Optimization problem

- **Defination²**: An optimization problem involves finding a subset, *S*, from a collection of candidates, *C*; the subset, *S*, must satisfy some specified criteria, i.e. be a solution and be such that the *objective function* is optimized by *S*. *`Optimized'* may mean ***minimized*** **or** ***maximized*** depending on the precise problem being solved.
- Greedy methods are distinguished by the fact that the **selection function** assigns a *numerical value* to each candidate, *x*, and chooses that candidate for which:
  - ***SELECT(x)* is largest**
  - **or**
  - ***SELECT(x)* is smallest**

Greedy Paradigm

- All Greedy Algorithms have exactly the same general form.
- A Greedy Algorithm for a particular problem is specified by describing the predicates `*solution*' and `*feasible*'; and the selection function `*select*'.
- Consequently, Greedy Algorithms are often very easy to design for optimisation problems.

Generic procedure for Greedy Method

- **Procedure GREEDY(C)**

- General frame work for greedy algorithm to find the optimal solution to the optimization problem. **C**: The set of all candidates to be the solution; **S**: The set of candidate that represent a solution, i.e. $\mathbf{S} \subseteq \mathbf{C}$

```
1.  S = ∅;
2.  While not solution (S) and C ≠ ∅ do
3.  Select  x ∈ C based on the selection criterion
4.  C ← C  - { x };
5.  If feasible ( S ∪ { x}) then
6.  S ← S ∪ { x };
7.  End if;
8.  End while;
9.  If Solution(S) then
10. return(S);
11. Else
12. return (“ No Solution”);
13. End if;
14. End greedy;
```

Greedy Method with SUBSET Paradigm

- **Algorithm GREEDY(a, n)**
- This algorithm is based on the selection of next feasible component for a solution by using a selection function SELECT() from the remaining list of candidates
- a[1:n] contains n inputs representing candidate set C

```
S = $\emptyset$; //Initializing the solution //
for i=1 to n  do
{
x := SELECT(a);
if  FEASIBLE (S, x) then
S := S $\cup$ { x };
}
Return S;
End greedy;
```

Greedy Method with ODERING Paradigm

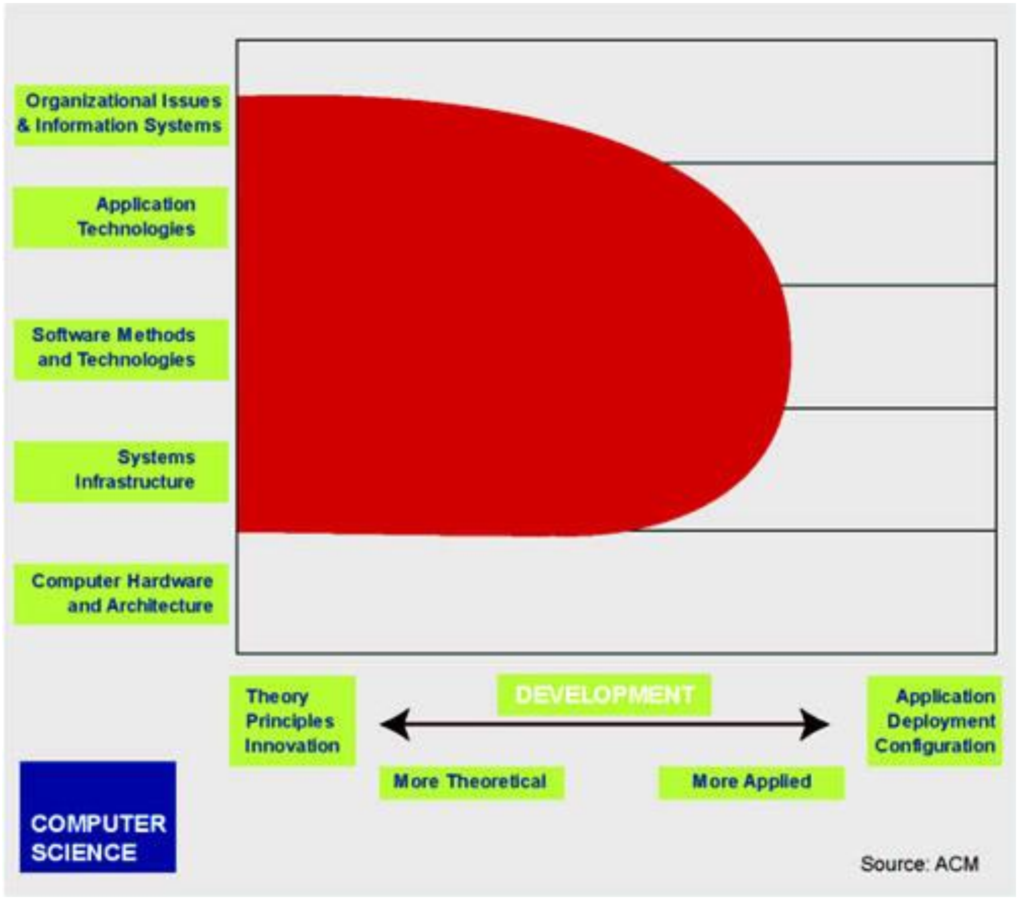
- **Algorithm GREEDY(a, n)**
- This algorithm is based on the selection of next feasible component for a solution by using the order from the a[1:n]
- a[1:n] contains n inputs in some order based upon the requirement by objective function

```
1. S = ∅; //Initializing the solution //
2. for i=1 to n  do
3. {
4. if  FEASIBLE (S, a_i) then
5. S := S ∪ { a_i};
6. }
7. Return S;
8. End greedy;
```

Greedy Paradigm

- At every step choose the best candidates remaining
- While solving a problem, Heap structure to be preferred so that, for all $x \in C$,can be arranged so that the selection process will be O(1).
- A greedy algorithm builds solution to a problem in steps.
- At each step best available remaining candidates are selected.
- A greedy algorithm may or may not be optimal. If it is not optimal it is sometimes close to the optimal and so yields a solution that in many applications is good enough.

Thank you


Organizational Issues & Information Systems
Application Technologies
Software Methods and Technologies
Systems Infrastructure
Computer Hardware and Architecture
Theory Principles Innovation
DEVELOPMENT
Application Deployment Configuration
More Theoretical
More Applied
COMPUTER SCIENCE
Source: ACM