GREEDY

Definition:

Greedy Algorithms are used to solve Optimization Problems.

Problem

solution-1
solution-2
.
.
.

} All of them are correct in the sense that they satisfy the problem constraints

The multiple valid solutions are called as feasible solution.

One out of the feasible solution becomes the optimal solution

Optimization

Objective: It is the criterion/metric that quantifies how good a solution is

What is the objective that you are trying to maximize or minimize.

Constraints: These are conditions or rules that a problem must satisfy.

What type of Problems Greedy algorithm solve?

TYPES OF PROBLEMS

Normal Problem

A problem has one specific answer ie right/correct answer

OPTIMIZATION Problem:

A problem can have more than one right answer

But I am not interested in the correct answer

Sequential Decision Making:

These are problems, which involves a series of decisions to be made to achieve the end goal.

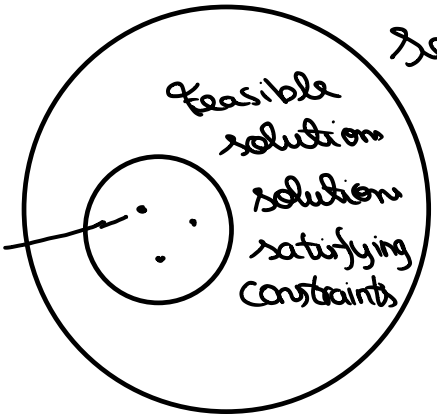
In simple words allows you to build the solution in incremental steps.

Example:

Finding the shortest path.
Minimum Denomination Problem
Video games.

Greedy Algorithms are suitable for:

Optimization Problems.

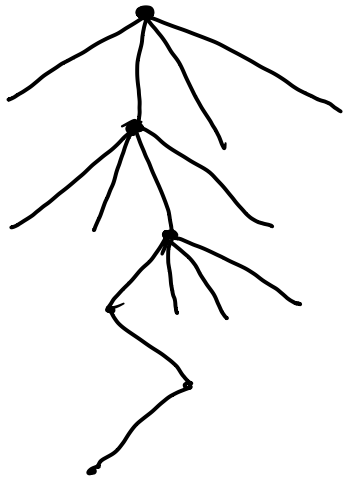


set of all solutions

OPTIMAL SOLUTION

Solutions which maximize my objective

Sequencial Decision making



Algorithms for optimization problems typically go through a sequence of steps with set of choices at each step.

At every step, you simply ignore all other options and blindly chose the immediate best. (Locally optimal)

Optimal substructure property:

If An optimal solution can be constructed from optimal solutions of its sub problems. then the problem is said to have optimal substructure property

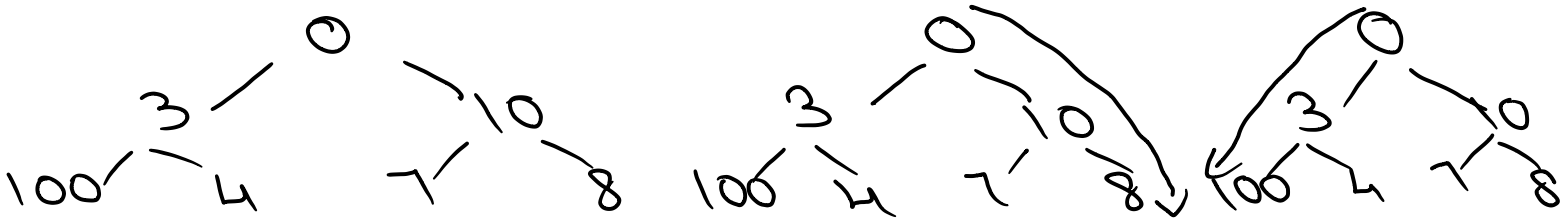
The best solution to a big problem can be constructed from the best solutions to its smaller parts.

The global optimum comes from optimal solutions of sub problems.

When to Consider greedy strategy?

* If the question involve
  1. Minimize / Maximize
  2. Series of Decisions to be made
  3. Has the optimal substructure property.

A graph Greedy Optimal



Container Loading Problem:

A large ship is to be loaded with Cargo containers

All containers are of same size

Different containers may have Different weights.

Let W_i be the weight of the i^{th} container.

The cargo capacity of the ship is C.

Let x_i be a variable whose value can be either 0 or 1

0 → not loaded

1 → loaded

Objective function: Maximize $\sum_{1}^{n} x_i$

Constraints { subject to $\sum_{1}^{n} w_i x_i \leq C$ &

$x_i \in \{0, 1\}$

Knapsack Problem:

Imagine you are thief and you want to fill as much as valuable items in your bag. Two Types:

i) Fractional knapsack (Can fill fraction of items)

ii) 0/1 knapsack (Can't break items, either take completely or don't).

Problems in this chapter

Represent it formally as an optimization problem

Identify the greedy choice (what I should be greedy about?)

Prove why being Greedy works

The greedy choice might not always be obvious.

Fractional knapsack works with being Greedy.

What about 0/1 knapsack:

*Being greedy about the value of the items works, but not in all situations.

Why greedy strategy works for fractional knapsack, but doesn't work for 0/1 knapsack?

Excludes/Limits subsequent filling

*A simple change in problem constraint renders the greedy algorithm useless.

*Greedy fails when you have to worry about the future.

your current decision might have negative consequence in the future, for that you need to have a look ahead or backtracking

Greedy algorithm never backtrack.

*Brute forcing 0/1 knapsack takes $\Omega(2^n)$

Given 'n' objects and a sack capacity

Pg 198 Horowitz. Sahni.

$x_i \in [0,1] \rightarrow x_i \in \{0,1\}$

It changes everything renders the greedy algorithm useless.

```
F_KnaSack(W, P, C, n):
    # (i=1 to N
    val[i] = Pi / Wi

    sort by val
    currLoad = 0
    for (i = 0 to n-1):
        If W[i] ≤ C - currLoad
            Take all of item, Load += W[i]
        else
            Take (C - currLoad) of item i
            break
```

Fix the greedy criterion
Sort by the greedy criterion
chosenSet = ϕ
for each item
    If feasible add to the chosen set

Return set

Knapsack Correctness / Proof:

Let $a = (a_1, a_2 \ldots a_n)$ be the solution to some Knapsack problem.

(0.5, 1, 0.75) Take half of the first item the complete second item and so on.

CLAIM: If a^* is optimal solution then a^* contains the first greedy choice.

makes out the most valuable item

First let prove this claim, and use it as a stepping stone to derive the complete correctness of Knapsack.

$$P \rightarrow Q$$
$$\neg Q \rightarrow \neg P$$

ie If a^* does not contain the first greedy choice, then a^* is not optimal.

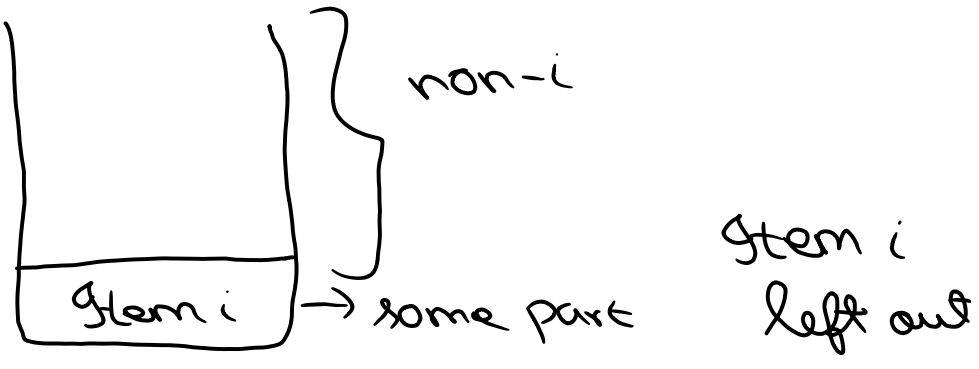
*If the solution can't utilize the most valuable item, then it can't be the (most) optimal solution

Suppose $\tilde{a}$ doesn't max out the most valuable item (say item i)

CASE-1: The knapsack is not full yet.

- Add more of item i. This will improve the quality of the solution
$\Rightarrow \tilde{a}$ is not optimal.

CASE 2: The knapsack is full (but still some amount of i is left out)
WKT i is the most valuable
still have some i left out
so we can replace something from knapsack with i
This will improve the quality of the solution.



swap something of same size as of left out i.

$$\frac{P_i}{w_i} > \frac{P_k}{w_k} \quad \forall \; k \neq i$$

If the solution does not contain the first greedy choice, then it is not optimal.

Activity Selection Problem:

scheduling several competing activities that are overlapping.

Objective: Pick the max # of non conflicting meeting

Given a set of n intervals $S = \{(s_i, f_i)\}$

We want to find the biggest subset S' of S such that no pair of activities in S' overlaps/conflicts.

Mutually compatible Activities.

Activities a_i and a_j are compatible if

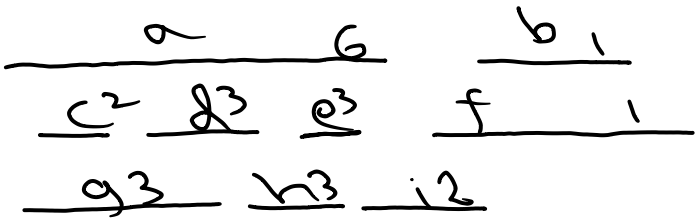
$$f_i \leq s_j \quad \text{or} \quad f_j \leq s_i$$

Greedy Algorithm for Activity Selection Problem

```
G ← ϕ
Sorts activities by [criterion]
for activities in sorted order:
        if activity compatible with G:
                Add activity to G
        end if
end for
```

Parameters / Constraints: Criterion

1. Starting time {a, f}
2. Finish time {c, d, e, f}
3. Duration $f_i - s_i$ {c, e, b}
4. Minimum Conflicting {



* Fractional Knapsack
* 0/1 Knapsack
* Activity Selection Problem
* Container Loading Problem.

STARTING TIME:

Duration

Minimum Conflicting :

While (S is not Empty)
{
  select the interval I that overlaps the
    least number of other intervals
  Add I to final solution set S'
  Remove all activity from S that conflicts I
}

Approach - 4:

Sort the intervals based on end time
  for every consecutive interval E
    if the left most end is after
    - the last selected 's right most
    - end then we select
    otherwise we skip it

We always pick the activity with least ending time.

Why being greedy about Minimum Finish Time works

If I select the activity that ends first, it will leave the longest possible period of time free for scheduling other activities.

Proof:

1. GREEDY CHOICE PROPERTY:

   ∃ atleast 1 optimal solution that start with the greedy choice

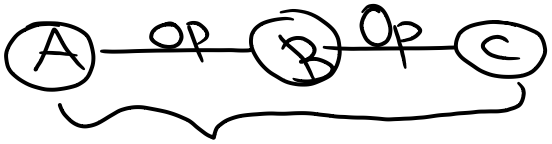
2. OPTIMAL SUBSTRUCTURE:

   If take/consider an optimal sol upto the first greedy

$a_1, a_2, \ldots a_n$ ordered by f_i —→①

$o_1, o_2, \ldots o_n$ ordered by f_i —→②

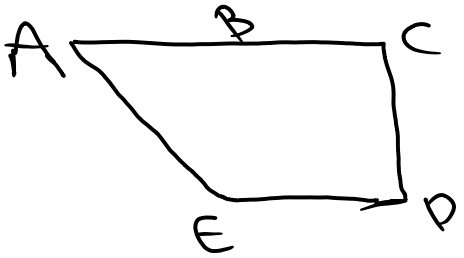
An optimal solution that doesn't start with greedy choice can be safely swapped with 1 & 2



optimal

Principle of Optimality:

We can solve the larger problem given the solution to it's smaller subproblem



Largest Path is not optimal substructure

$L(B,D) = \langle B,A,E,D \rangle$

Can't be combined for global optimal solution.

$\begin{cases} L(B,A) = \langle B,C,D,E,A \rangle \\ L(A,D) = \langle A,B,C,D \rangle \end{cases}$

Greedy A

* Build solution in steps
* At each step make a choice that is
  1. Feasible (Satisfy problem constraints)
  2. Irrevocable (Can't revise/undo) you make a choice and commit yourself to it
  3. Locally optimal (Pretend like the world going to end)

you hope that by being greedy at every step you might maximize the final objective.

Class Scheduling (SCHED):

Input: $C = \{(s_i, f_i) \mid 1 \le i \le n\}$ the start and finish times for n classes

Output: m a minimum number of classroom used and P is an m-partitioning of C into m non-overlapping sets.

RSP($L = \{(s_i, f_i) \mid 1 \le i \le n\}$)

Sort L in ascending order of
- start time

start with d=0 classroom

For each class l_i in L:
if lecture l_i is compatible with
- one of the d classroom
- schedule l_i in this classroom
else:
Allocate a new lecture hall d+1
schedule l_i in this classroom
End for.

If an old classroom is available at the start of the current lecture, schedule it over there.

How much time will it take to check compatible
Should I go through all d at once, every time the loop runs?
O(n*d) (d could become n)
in worst case
Using a max-heap or priority-queue to sort it based on time which will reduce the running time to nlogn.

The depth of a set of intervals is defined as the maximum number of intervals that overlap with each other.

Theorem:

In any instance of Interval Partitioning the number of lecture halls needed is atleast the depth of the set of lectures.

Proof:

Suppose a set of lectures has depth d and let $L_1, \ldots L_d$ all pass over a common point on the time-line. Then each of these lectures must be scheduled in a

different lecture hall, so the whole instance needs atleast 3 lecture halls.

Theorem:

The greedy algorithm never schedules two overlapping lectures in the same lecture hall.

Proof:

Consider any two lectures L and L' that overlaps.

Suppose L precedes L' in the sorted order.

Then when L' is given to the algorithm, L is in the set of lectures whose lecture hall won't be compatible with lecture L'.

Consequently the algorithm will not assign lecture L' to the lecture hall that is used for L.

As this holds for any randomly chosen overlapping lectures, this holds for every combination of two overlapping lecture.

The first ending activity determines the first available Room

By using a heap, you are picking up a room that has the longest possible period of time free for scheduling other activities

Implicitly you are applying ASP within RSP to schedule your lectures.

Theorem

The greedy algorithm for Interval Partitioning is optimal.

Proof:

If greedy allocates d lecture halls, then the depth is d.

Since w.k.t.

d is the lower bound

greedy is optimal.

Lecture hall d opened because lecture j overlapped with d-1 others

Since we sorted by start time, all these overlapping lectures are caused by lectures that start earlier than or at s_j.

Thus we have d lectures overlapping at time s_j.

This holds whenever you open a new lecture hall.

Consequently also for the last

one opened.
Thus greedy opens d lecture halls and d is the minimum that we needed based on theorem-1.

Proof by structural bound:

One finds a simple, "structural bound" asserting every problem.

COIN CHANGE PROBLEM:

Greedy works for [10, 5, 2]
Example Change = ₹12
But doesn't work for [8, 6, 2]