Greedy Algorithm

"Optimization Problem"

Greedy algorithm generate a solution

which are { 1. feasible (satisfying the Conserains)
2. Local optimal (" maybe not global optimal)
3. Irrevocable (i.e. no Back tracking. Once a part of solution is generated it remains unchanged)

General Template:

1. Sorting Based on some criterion.
2. Choose the optimal one at current stage

"Greedy Algorithm may not be globally optimal"

Example: Find the subset with maximum sum of non-adjacent elements

Algorithm:

```
S := empty set
While some elements remain in A:
    Pick the largest A[i]
    S. append (A[i])
    delete A[i] and its neighbours
return S
```

Consider a counter example (non-global-optimal one)

A = {1, 4, 6, 4} —algorithm→ S = {1, 6}, sum = 7.

Actual Optimal One: S' = {4, 4}, sum = 8

Interval Scheduling

Interval scheduling.
- Job j starts at s_j and finishes at f_j.
- Two jobs are compatible if they don't overlap.
- Goal: find maximum size subset of mutually compatible jobs.

find subset with max # jobs.

Greedy Template

1. Consider jobs in some orders
   { Start time?
   Interval length?
   # Conflicts?
   ...
2. Choose the one which is compatible with the ones already taken && is optimal base on criterion at current stage

The algorithm:

"We sort jobs in order of finish time"

```
Sort jobs by finish time so that f1 ≤ f2 ≤ ··· ≤ fn.
A ← φ , last ← 0
for j ← 1 to n
  if sj (start time of job j) ≥ last then:
      A ← A ∪ {j} , last ← fj (finish time of job j)
return A
```

Running time is dominated by cost of sorting

∴ T(n) = O(nlogn)

Optimality: This algorithm is global optimal!

proof:

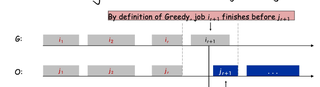
Let the Greedy Algorithm generated solution be G

the Optimal Solution is O (i.e. with maximum # compatible jobs)

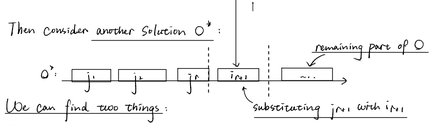
Assume: G is different from O

{ G = {$i_1, i_2, i_3, \cdots, i_k$}.
O = {$j_1, j_2, j_3, \cdots, j_m$}

Find the largest r, so that $i_1 = j_1, i_2 = j_2 \cdots, i_r = j_r$



Then consider another solution O*:



We can find two things:

① jobs in O* is still compatible with each others
i.e. Still no time conflicts

Since { By Greedy Definition, the finish time of j_{r+1} is later than i_{r+1}
the finish time of j_r is equal to i_r

⇒ There is enough space for i_{r+1}!

② O* is still optimal, Since # jobs doesn't change.

Then, We can repeat the above process with O ← O*

Until O becomes the same as G!

⇒ Proof finished! G is also optimal as O!

The fraction Knapsack Problem:

Input: Set of n items: item i has weight w_i and value v_i, and a knapsack with capacity W.

Goal: Find $0 \le x_1, \ldots, x_n \le 1$ such that $\sum_{i=1}^{n} x_i w_i \le W$ and $\sum_{i=1}^{n} x_i v_i$ is maximized.

(Sum of fractions of items; Sum of weighted values)

{ fraction knapsack: $0 \le x_1, \ldots x_n \le 1$
0/1 knapsack: $x_1, x_2 \cdots, x_n \in \{0, 1\}$.

Algorithm:

```
Sort items so that v1/w1 ≥ v2/w2 ≥ ··· ≥ vn/wn. ("value-per-pound")
w ← W
for i ← 1 to n:
  if wi ≤ w then:
     xi ← 1
     w ← w - wi
  else:
     xi ← w/wi   (i.e. fill up the remaining space)
     return
```

Running time: O(nlogn) (dominated by sorting)

Correctness (Optimality) Proof:

We assume $\sum w_i \ge W$, Since if $\sum w_i < W$ the algorithm is trivially optimal

Let greedy solution G = ($x_1, x_2, \cdots, x_k, 0, \ldots, 0$) (since $\sum w_i < k$)

Consider optimal solution

O = ($y_1, y_2, \cdots, y_n$)

Look at the first item i where the two solutions differ.

G = $x_1\ x_2\ x_3\ \ldots\ x_{i-1}\ x_i \ldots x_k \ldots\ 0\ \ 0$
O = $x_1\ x_2\ x_3\ \ldots\ x_{i-1}\ y_i \ldots y_k \ \ldots\ y_{n-1}\ y_n$
(same)

By definition of Greedy: $x_i \ge y_i$

let $x = x_i - y_i \ge 0$

We then do: ① let $y_i = x_i$; ② remove a total weight of xw_i items from ($y_{i+1}, \cdots y_n$)

(This is always doable: since $\sum_{j=i}^{n} x_i w_i = \sum_{j=i}^{n} y_i w_i$ the is always enough items to fill the gaps)

After the modification:

1. The total value is not decreased

Since we: ① add weight xw_i items with value-per-pound $\frac{v_i}{w_i}$ to O
② remove weight xw_i items with value $\frac{V}{W}$ from O.
($= \frac{\sum_{j=i}^{n} v_j}{\sum_{j=i}^{n} w_j}$)

Since the items is sorted based on $\frac{v_i}{w_i}$

The removed items have lesser or equal value compared to added items.

2. The total value cannot be larger

Since: O is already optimal

∴ The modified O (O') is still optimal

Then we do: ① O ← O', ② repeat the process

until O = G ⇒ G is also optimal!

Interval Partitioning

- Lecture j starts at s_j and finishes at f_j.
- Goal: find the minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

Algorithm

```
Sort intervals by starting time so that s1 ≤ s2 ≤ ··· ≤ sn.
d ← 0   // # classrooms.
for i ← 1 to n:
  if lecture i is compatible with some classroom k:
     Schedule lecture i in classroom k.
  else:
     add a new classroom d+1.
     Schedule lecture i in classroom d+1
     d ← d+1
```

Optimality:

Define: Depth = the maximum number of intervals that exist at any instant of time

Observation: "# classroom needed ≥ Depth"

∴ if solution ⇒ # classroom needed = Depth then the solution is optimal

↓ Proof of the optimality of the algorithm:

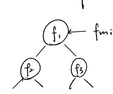
Let d = number of classroom opened by greedy algorithm

Note that: the reason why opened the d-th classroom:
is we need to schedule a lecture j, which is incompatible with all d-1 other classrooms
which means: at time s_j (starting time of lecture j)
there are d-1 lectures that are proceeding
∴ depth ≥ d (depth: the maximum number of intervals that exist at any instant of time)

Since depth ≤ d by observation
∴ depth = d ⇒ Algorithm is optimal

Running time:

Using a min-heap to store classrooms




```
for i ← 1 to n:
  if lecture i is compatible with some classroom k:
     Schedule lecture i in classroom k.
  else:
     add a new classroom d+1.
     Schedule lecture i in classroom d+1
     d ← d+1
```

"Extract-min" ⇒ fmin ⇒ Whether fmin ≤ s_j

f = f_j ⇒ "Insert"

"Insert"

∴ Running Time: n intervals, each one with Θ(logn) time.
⇒ T(n) = θ(nlogn)