Greedy Algorithm

Topics Covered:

Greedy Algorithm

Implementation and Analysis

Main Incredient of greedy algorithm

1. Greedy Algorithm:

Date ______
Page ______

- Toy problem: what is the largest number that consist of digit 9, 8, 1, 9, 6, 1 ? Use all the digit

  ex. 16899 , 69891 , 98961 ...

  * 98961 → $\underbrace{99861}$ will be correct answer

* Greedy Search Strategy:

{9, 8, 9, 6, 1} :- 9 9 8 6 1 {Append}

Remove → Find max digit

→ Append it to the number

→ Remove it from the list of digit

→ Repeat while there are digits in the list

Q2. Queue of Patient:

Input: n patient have come to the doctor's office at 9:00AM. They can be treated in any order. For i-th patient, the time needed for treatment is t_i. You need to arrange the patient in such a queue that the total waiting time is minimized.

Output: The minimum total waiting time.

example: Suppose $t_1 = 15$ $t_2 = 20$ and $t_3 = 10$ and if suppose we arranged (1, 2, 3)

- First patient doesn't wait
- Second patient wait for 15 min
- Third patient wait for 15 + 20 = 35 min

⇒ Total waiting time 15 + 35 = 50 min.

Second arrangement: (3, 1, 2)

- First patient doesn't wait
- Second patient wait for 10 min
- Third → 25 min {10 + 15}

∴ ⇒ Total waiting time ⇒ 10 + 25 = 35 min

this arrangement is better.

Greedy choice:

→ first patient with the maximum treatment time
→ first treat patient with minimum treatment time
→ first treat the patient with average treatment

⇒ • first treat the patient with the minimum treatment time.

→ Remove this patient from the queue

→ Treat all the remaining patient in such order as to minimize their total waiting time.

Subproblem:

• Minimum total waiting time for

n patient $= (n-1) \cdot t_{min}$ + minimum total waiting time for $n-1$ patient without t_{min}

Implementation and Analysis

MinTotalWaitingTime (t, n) Pseudo code.

```
waitingTime ← 0
treated ← array of n zeros
for i from 1 to n:
    tmin ← +∞
    minIndex ← 0
    for j from 1 to n:
        if treated [j] == 0 and t[j] < tmin:
            tmin ← t[j]
            minIndex ← j
    waitingTime ← waitingTime + (n-i)·tmin
    treated [minIndex] = 1
return waitingTime.
```

Lemma: The running time of MinTotalWaitingTime (t, n) is $O(n^2)$

Actually this problem can be solved in time $O(n \log n)$

- Instead of choosing the patient with minimum treatment time out of remaining ones n times, sort patient by increasing treatment time
- This sorted arrangement is optimal

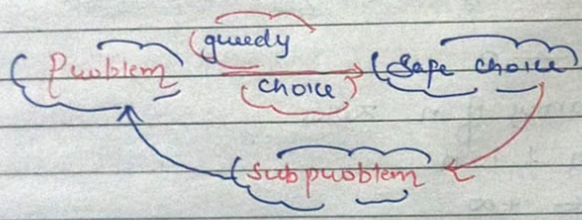
Main ingredient of greedy search

- Make some first choice of optimal s
- Then solve a problem of the same kind
- Smaller : fewer digits, fewer patients
- This is called a "subproblem".

* A choice is called safe if there is optimal solution

→ Not all choice are safe

General Strategy:



Q3: Celebration Party Problem.

Naive Algorithm

- Try all possible distribution of children into one or more group
- For each distribution check whether any two children in any group differ by at most 2 year of age
- Return the minimum number of groups among valid distribution.

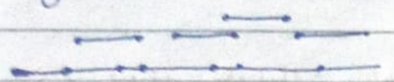
Lemma: The running time of the naive algorithm is at least 2^n, where n is the number of children

Greedy Algorithm:

Input: A set of n points $x_1, \ldots, x_n \in R$

Output: "The minimum number of segments of length at most d needed to cover all the path points"

Safe choice: Cover the leftmost point with a segment of length d which starts in the points



- cover the leftmost point with a segment of length 2
- Remove all points within this segment
- Solve the same problem with the remaining points

Implementation and Analysis

Assuming $x_1 < x_2 \cdots x_n$

Pseudo code: PointsCoverSorted $(x_1, \ldots x_n)$

```
segments ← empty list
left ← 1
while left ≤ n:
    (l,r) ← (x_left, x_left + 2)
    segments.append (l,r)
    left ← left + 1
    while left ≤ n and x_left ≤ r:
        left ← left + 1
return segments
```

~~Lemma~~:

The running time of This is $O(n)$ linear

Proof

- As left changes from 1 to n
- For each left, append at most 1 new segment of to solution
- overall running time $O(n)$

* Total Running Time:

→ PointsCover ⇒ $O(n)$

Sort $\{x_1, x_2 \cdots x_n\}$ then call PointsCover:

Sort + PointCover

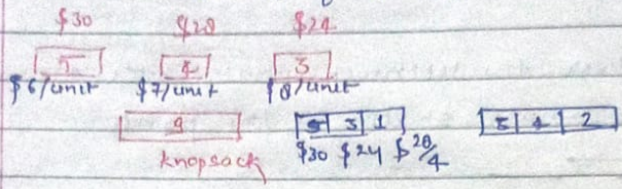
$O(n \log n)$

Managing Loot:

Input: Weight $w_1 \ldots w_n$ and value $v_1 \ldots v_n$ of n items; Capacity W

Output: The maximum total value of fraction of item that fit into a knapsack of capacity W.

Ex



Lemma: There exist an optimal solution that uses as much as possible of an item with with the maximum per unit of weight. {safe choice}

Greedy Algorithm

- While knapsack is not full
- choose item i with maximum $\frac{v_i}{w_i}$
- If item fits into knapsack, take all of it.
- Otherwise take so much to fill the knapsack.
- Return total value and amounts taken

Pseudo Code:

```
maxValuePerWeight ← 0
bestItem ← 0
for i from 1 to n:
    if wi > 0:
        if vi/wi > maxValuePerWeight:
            maxValuePerWeight ← vi/wi
            bestItem ← i
return bestItem.
```

knapsack($W, w_1, v_1, \ldots w_n, v_n$)

```
amount ← [0, 0 ... 0]
totalValue ← 0
repeat n times:
    if W = 0:
        return (totalValue, amounts)
```

$i \leftarrow$ BestItem $(w_1, v_1, \ldots, w_n, v_n)$

$a \leftarrow \min(w_i, W)$

$totalValue \leftarrow totalValue + a \frac{v_i}{w_i}$

$w_i \leftarrow w_i - a$

$amounts[i] \leftarrow amount[i] + a$

$W \leftarrow W - a$

return (totalValue, amounts)

Lemma: The running time of Knapsack $O(n^2)$

KnapsackFast $(W, w_1, v_1, \ldots, w_n, v_n)$ {when already sorted}

amount $\leftarrow [0, 0 \ldots 0]$

totalValue $\leftarrow 0$

for i from 1 to n:

if $W = 0$:

return (totalValue, amounts)

$a \leftarrow \min(w_i, W)$

$totalValue \leftarrow totalValue + a \frac{v_i}{w_i}$

$w_i \leftarrow w_i - a$

$amount[i] \leftarrow amounts[i] + a$

$W \leftarrow W - a$

return (totalValue, amounts)

→ knapsack after sorting $O(n)$

Sort + knapsack $O(n \log n)$