Greedy - Road trip

Kingston -> NY
one road, find min no. of times we need to refuel car

Assumptions:
- -> no gap b/w stations so tank is depleted
- -> distance known station 1 -> station 2
- -> We know how much gas left and how much it can cover

Recursive algo:
1. Sort stations dep on how far from kingston
   $\{ s_0, s_1, s_2 \ldots s_n \}$
2. Road trip (s_i): # leave station with full tank
   $t = i + 1$
   while $t < n$ and s_{t+1} reachable:
   $t = t + 1$ # don't stop at s_i
   if $t = n$:
   stop # arrived
   else:
   fill up gas at s_t
   Road trip (s_t)

Complexity:
1. Sort: $O(n \log n)$
2. $O(n)$
∴ $O(n \log n)$

Proof of correctness - by induction

Basis: $n = 1$, to NYC in 1 tank of gas
IH: optimal sol^n when full tank and k stations ahead of us
$k \geq 1$

I/P: full tank and k+1 stations around

Solution: $A = \{a_1, a_2 \ldots a_b\}$

1. Exists at least one optimal sol^n that starts from a_1

2. $A\{a_2, a_3 \ldots a_k\}$ is optimal for reduced problem

3. Combine 1 & 2, we get sol^n to original prob.

Greedy:
1. initialize/sort
2. Pick best local option dep on criteria ts
3. Update state of problem (remove etc)
4. Repeat until no more choices

Recursive format:

function greedyRecursive(problem_state):

Base case: if prob state empty //base condition return result

Best local: choice = selectBestChoice(problem_state)

update prob state: new_prob_state = problem_state − choice

recur on rem state: result = greedyRecursive(new_prob_state)

combine: return combine(choice, result)

We need to find least no. of stations

Proof by induction

Basis: $n = 1$, to NYC in 1 tank of gas

IH: optimal sol^n when full tank and k stations ahead of us $k \geq 1$

IP: show roadtrip can find optimal with full tank if $k+1$ stations ahead.

$A = \{a_1, a_2, a_3 \ldots a_t\}$ solution $t = k+1$ stations

$O = \{o_1, o_2 \ldots o_r\}$ optimal any

we know $a_1 \geq o_1$ as a_1 is the furthest possible station

o_2 is the next, can get there

$O^* = \{a_1, o_2, o_3 \ldots o_r\}$ and $|O^*| = |O|$

So it is further and same no. $\rightarrow$ (same no. of stations) and no loss in optimality

So for $\{a_2, a_3 \ldots a_6\}$ is optimal to reduced problem where there are k stations ahead (IH)

$$|\{a_2, a_3 \ldots a_6\}| \leq |\{o_2, o_3 \ldots o_r\}|$$

(less) (more)

$\therefore |A| \leq |O^*|$ and A is optimal.

Roadtrip to Iterative:

```
sort stations
roadtrip_iter(stations)
    i = 0   (start w full tank)
    while i<n:
        t = i+1
        while t<n and isReachable(s_i, s_t):
            t = t+1
        t = t-1

        if t==n:
            break
        else:
            i = t
```

move as far as possible

adjust t

arrived

stop, fill up, continue

Activity Selection

Given a few activities w start and end times, how many max activities can we do (so they don't overlap).

Pseudo:

```
sort by earliest to latest finish time
Activity select(x):
    select x_1
    current time = x_1.end
    for i = 2 to n:
        if x_i.start >= current time:
            select x_i
            current_time = a_i.finishtime
```

$O(n \log n) + O(n) = O(n \log n)$

* does not work by start time as we don't know if a_1 ends before o_2

Proof by induction:

Base case, ($n = 1$):
if there's only 1 activity then greedy picks it.
it's the most optimal

IH: For $n \leq k$ activities, earliest finish time first greedy algo produces optimal solution

IS: Now for $n = n+1$ activities
$a_1.finish \leq a_2.finish \ldots \leq a_{n+1}.finish$

$A = \{a_1, a_2 \ldots a_k\}$ set chosen by greedy
$O = \{o_1, o_2 \ldots o_r\}$ any optimal / any compatible set for $n+1$ activities

We need to show $|A| \geq |O|$ (it's more)

1. Show optimal solution with a_1
$a_1.finish \leq o_1.finish$ as a_1 is the earliest finisher
also we know o_1 and o_2 do not overlap and from o_2, o_2 is after o_1 so
$a_1.finish \leq o_1.finish \leq o_2.start$

Thus we know swapping o_1 for a_1 does not break feasibility,
a_1 is compatible with $o_2, o_3 \ldots o_r$
So $O^* = \{a_1, o_2, o_3 \ldots o_r\}$ is feasible

2. show A is optimal
-> greedy problem works on reduced $\{a_2, a_3 \ldots a_k\}$ to be optimal (IH)
-> in O^*, rest of $\{o_2 \ldots o_r\}$ is solution to same reduced problem so $|\{a_2, a_3 \ldots a_k\}| \geq |\{o_2, o_3 \ldots o_r\}|$

$\therefore$ So $|A| \geq |O^*|$ thus A is optimal sol^n to $k+1$ activities

Coin change

Change with fewest coins

Pseudocode:

sort n coins in decreasing values $v_1, v_2 \dots v_k$

MinCoinChange(m):

r = m -> remaining we need change for

while r > 0:

findmax coin of value $v_i \le r$

select coin of value v_i

$r = r - v_i$

$O(k \log n + m)$

-> doesn't work if no 1 cent coin

-> if we have {9, 4, 1}, m = 12

it will take 9 1 1 1

should take 4 4 4

Proof by induction

e.g coin value, {200, 100, 25, 10, 5, 1}

base case:

$1 \le m < 5$	only m pennies
$m = 5$	1 nickel
$5 < m < 10$	Nickel and m-5 pennies
$m = 10$	dime
$10 < m < 15$	dime and m-10 pennies
$15 \le m < 25$	dime, nickel and m-15 pennies
$m = 25$	quarter
:	
200	

IH: MinCoinCharge provides optimal solution when target value is k and $k \geq 200$

IP: Min coinchange provides optimal solution when target value $k+1$

$A = \{a_1, a_2 \ldots a_k\}$ optimal
$O = \{o_1, o_2 \ldots o_r\}$ any choice

We NTS $|A| \leq |O|$

1. Show optimal solution with a_1, suppose an O doesn't have \$2

O can have at most 1 \$1, or get replaced by \$2
2 \$0.25, or get replaced by \$1
4 \$0.01, or get replaced by \$0.05
2 \$0.1 or \$0.05 + \$0.1
199¢ 194¢
$\therefore$ max 200¢

Coinchange by American coin?

Fractional knap sack

Container of capacity = c
set $X = \{x_1, x_2 \ldots x_n\}$
set of fractions $\{p_1, p_2 \ldots p_n\}$ s.t $0 \leq p_i \leq 1$

$\sum_{i=1}^{n} p_i * x_i$ .mass $= c$ and $\sum_{i=1}^{n} p_i * x_i$ .value is max

How to sort x_i in X?

1. Sort objs $\{x_1, x_2, x_3 \ldots x_n\}$ in $\frac{x_i.value}{x_i.mass}$ order

$O(n \log n)$

```
Greedy_FKP(x) :
    total_mass = 0
    while total_mass < c :
        take as much of next as possible        O(n)
        update total_mass.

∴ O(n log n)
```

Example

$c = 15$ $\quad x_1 + x_2 + \frac{1}{10}x_3 \quad 15$ $\quad$ total_value =

7 + 6 = 13 $\quad$ 15 − 13 = 2 $\quad \frac{2}{20} = 10\%$

	x_1	x_2	x_3	x_4	x_5	x_6
value	15	14	30	100	2	100,000
mass	6	7	20	80	2	100,001
value / mass	2.5	2	1.5	1.25	1	0.99999

high $\qquad$ low

Here we take x_1 as it is 6, x_2 as 7, and 10% of x_3 = 15

Proof

Base case : $n = 0$, no objs so no pick

IH : Greedy chooses optimal solution with k objects

IP : Greedy chooses optimal solution with $k+1$ objs

Greedy FKP soln: $A = \{p_1, p_2, p_3 \cdots p_{k+1}\}$

any $O = \{q_1, p_2, q_3 \cdots q_{k+1}\}$

1. Optimal solution with p_1
2. A is optimal

∴ Contradiction

we don't have a sol^n with p_1

O's first fraction is not p_1, $q_1 < p_1$

Total mass(A) = Total mass(O) = c

Since $q_i < p_1$, has to be a x_i where $q_i > p_i$

$O^* = \{p_1, q_2, q_3, \ldots q_i' \ldots, q_{k+1}\}$ → extra as this is $q_i > p_i$

make sure total_mass$(O_i^*) = c$ as $p_i > q_1$

$(p_1 - q_1) x_i.mass = (q_i - q_i') x_i.mass \quad q_i' < q_i$

$total\_val(O^*) - total\_val(O) = (p_1 - q_1) x_1.value - (q_i - q_i') x_i.value$

$= (p_1 - q_1) x_1.value - \frac{(p_1 - q_1) \cdot x_i.mass}{x_i.mass} \times x_i.value$

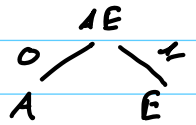
$= (p_1 - q_1) \times x_1.mass \left(\frac{x_1.value}{x_1.mass} - \frac{x_i.val}{x_i.mass} \right)$

> 0

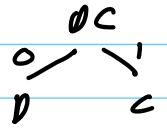
Huffman coding

-> build binary tree from bottom up.
-> start with least freq letters and connect through shared parent node, 0 and 1
-> parent = single character joining two daughter
-> repeat until all added.

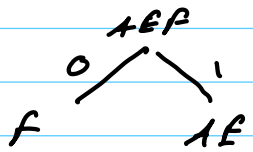
cj:
B: 16
F: 11
C: 9
D: 8
E: 8
A: 7



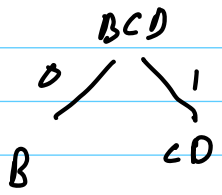
B: 16
AE: 15
F: 11
C: 9
D: 8



CD: 17
B: 16
AE: 15
F: 11

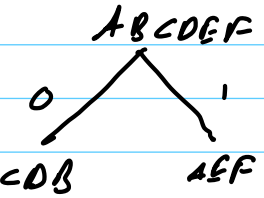


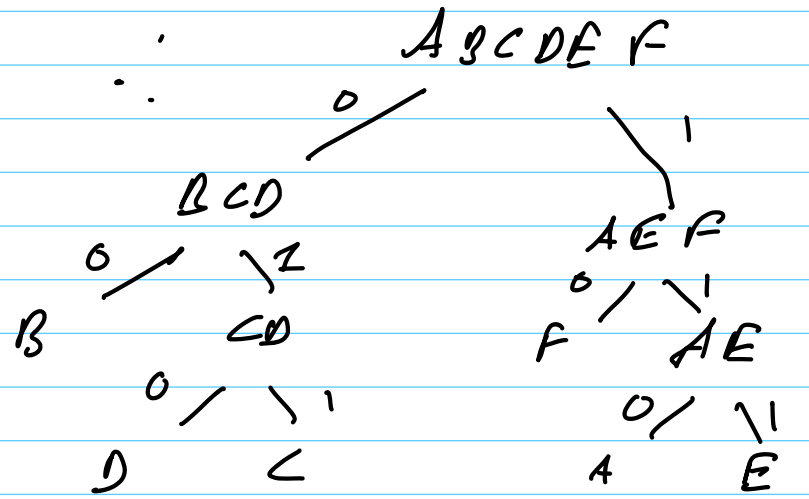
AEF: 26
CD: 17
B: 16



3

AEF: 26
CDB: 33





$O(n \log n)$

Greedy:

sort
run to find local best
update problem statement
run on reduced problem
combine local best and reduced

Proof:

1. Base case
2. IH: Greedy finds optimal for k
3. IP: Greedy finds optimal for $k+1$

$A = \{p_1, p_2, p_3 \ldots p_t\}$

$O = \{o_1, o_2 \ldots o_r\}$

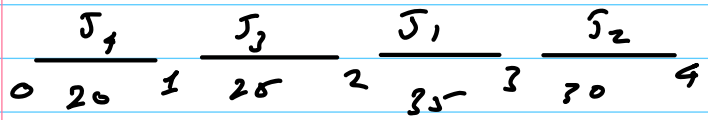
1. Show p_1 is optimal
2. Show A is optimal (reduced)
3. Say combined is optimal

Job sequencing problem

Job Sequencing with Deadlines

n=7

Jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7
profits	35	30	25	20	15	12	5
deadlines	3	4	4	2	3	1	2
					x	x	x



Pseudocode :

```
I. Sort depending on profit, in X as (profit, deadline)
n = max deadline number from X
Jobsequencing(X)
    sequence = array with max size of deadline
    for i in sequence :
        if len(sequence) == n :
            return sequence
        else :
            add i to sequence deadline position
            if it is filled then put in deadline-1
            position in sequence (Loop till
            free position found)
```

Proof :

Base case : (n = 0 or n = 1)
if there's no jobs then will return
will return only job possible if it is
the only one that needs to be done.

IH : Greedy algorithm provides optimal
solution when there are k jobs
where $k \geq 1$

IP: We need to show that greedy algo finds the most optimal solution when there are k+1 jobs

$A = \{J_1, J_2, J_3, \ldots J_r\}$ by greedy

$O = \{O_1, O_2, O_3, \ldots O_k\}$ any optimal

1. Show that J_1 is optimal

Here we know that J_1 is a tuple (profit, deadline) and is more optimal than O_2., the profit for J_1 is larger than for O_1 so we can replace O_1 to make $O^* = \{J_1, O_2, O_3, \ldots O_{k+1}\}$ such that $J_1.profit > O_1.profit$.
Here J_1 is given the latest spot for its deadline but O_1 may be given spot depending on if its deadline is different.

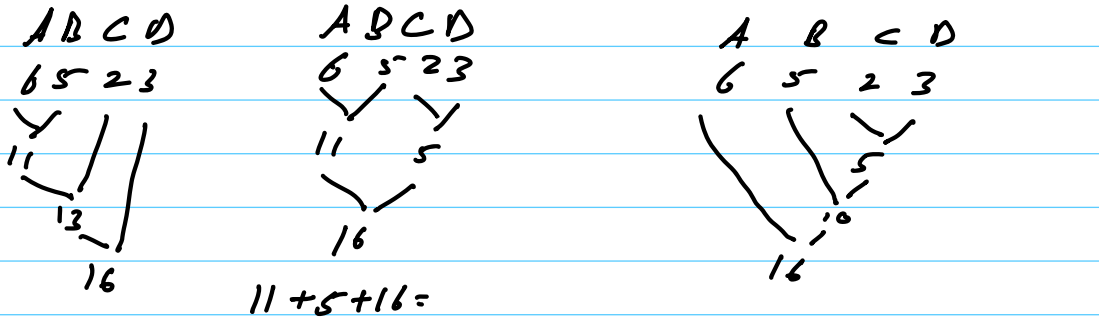
Since the J_1 is now in O^*, it has a larger profit than O.

2. Show that A is optimal
The greedy algo carries out the algo on rest of $O^* = \{O_2, O_3 \ldots O_k\}$ and we know by IH that it finds the most optimal from k jobs, thus we get total profit A ≥ total profit O

O^* ends up to be A

Thus we see by combining the first step (J_1) and rest with IH we get the most optimal solution.

Optimal merge pattern



11+13+16 = 40 (time) (largest)

11+5+16 = 21+11 = 32

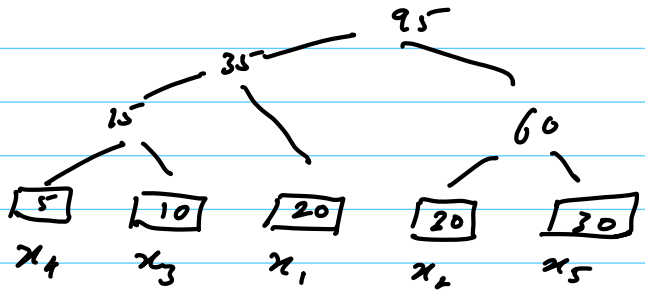
5+10+16 = 31 (smallest)

Greedy: always merge small lists, total merging time reduced

example

lists -> x_1 x_2 x_3 x_4 x_5

sizes -> 20 30 10 5 30



when we get 35, we have 35, 30, 30

We do 20, 30 = 60, then we do 35, 60 = 95

15 + 35 + 95 + 60 = 205

(Used in huffman)

Practice problems week 6

n trains
arrival
departure
can arrive only when previous departed
(similar to activity select)
k platforms, always enough for n trains

x = [(930, 1020), (1000, 1340), (1030, 1560), (1350, 1600)
10 9 10 9

min_platforms (x)
sort dep on dep time [(830, 920), (700, 840), (930, 1100)]
tuple = [ ]
current_finish = None
dictionary_stations = { }
for i in x:

sort
use list platform available to
assign platform # with available
time
platform 0 to train 0 arrive time
for i in x:
search for platform that's
avail time is before start
if not avail:
add new platf

see the dep time
compare with last arrival time
if it is after then assign it its
station
otherwise go back and check arrival
of the one before, if after assign
it its station.
if we reach the first and one
before, create a new station and
assign
return assignment

Practice prob 7 Pseudo

Lamp posts
start at city hall (pos 0)
to Out & East (position n)
total length = n
n+1 equally spaced lamp posts located at 0, 1, 2, ... n on the street

ith lamp post illuminates (i - r[i], i + r[i])

min_lamp(r) r: [0, 0, 1, 1, 0, 2]

sort r = [0, 0, 0, 1, 1, 2]
(indices 0, 1, 2, 3, 4, 5)

[(0,0), (1,1), (2,2), (3,4), (3,5), (3,7)]

[1 1 1 2 2 3

[(-2,2), (-1,3), (1,3), (2,4), (4,5), (2,8)]

1 - 0,
3 - 1, 3 + 1
2, 4
5 - 2, 5 + 2
3, 7

r = (1, 1, 2)
(indices 0, 1, 2)
= [(1,1), (0,2), (0,4)]

[(-1,1), (0,2), (1,3)]

min_lamp(r):
get new array with left and right cover of lamps
arr = [(left, right), (left, right) ...]
sort arr by left endpoints, leftmost to rightmost

for i in arr:
if leftmost lamp left end > 0:
return None

general format:

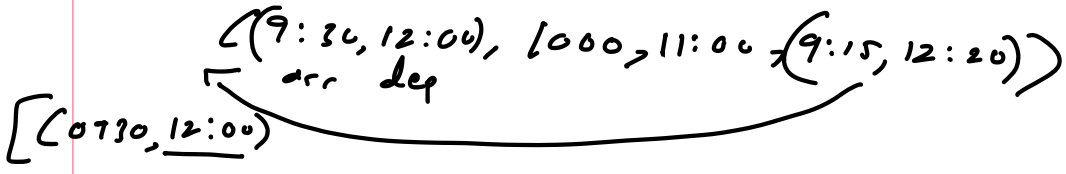
1. sort(data)

2. count = 0
   current_state = start

3. for choice in data:

```
        if condition_met(choice, current_state):
            update_state(choice)
            count += 1
```

return count or final_state




```
def platform_assignment(trains):
        sort (trains) by arr time
        new list stations with av time
[(0, 12:00)  ( added first train dep to station 0 in list
        for i in rest of trains:

                if train arrival time is after any stat
                    avail time:
                        add train, station in list

                else:
                        create new station,
                        assign
```

Proof by induction:

Base case ($n=1$ / $a=0$):
if there are no trains, there will not be any scheduling
if there is 1 train, it will be assigned one station, $st = 0$

IH: Greedy method finds the most optimal number of stations required when there are k trains.
$k \geq 1$

IP: We want to show greedy finds the most optimal no. of stations when no. of trains is $(k+1)$

Assume $A = \{ s1, s2, s3 \ldots s_t \}$ stations found by greedy
Assume $O = \{ o1, o2, o3 \ldots o_r \}$ found by any optimal

1. We show $s1$ is optimal: . . . .
we take A and O so that
we know $s1$ has more trains than o_1

if one train arrives, the size of O will increase. However if we use O^* with sl, $O^* = \{s_1, o_2, o_3 \ldots - o_r\}$

O^* will assign optimally s.t. $|O^*| \leq |O|$

(9-10) 8-9 (11)-12

s_1 s_2 s_3

nxt: 10 nxt: 9 nxt: 12

(9-10) 8-9 (11)-12

s_1 s_2

nxt: 10 nxt: 9

Step: Show A is optimal.

k, IH

O^* will same to A

$\therefore A \subseteq O$ //

$[\underset{0}{1}, \underset{1}{1}, \underset{2}{2}]$ index - magnitude

$-1 > 1+2 = 2$

$[\underset{0}{0}, \underset{1}{0}, \underset{2}{1}, \underset{3}{1}, \underset{4}{0}, \underset{5}{2}]$

$[(-1,1), (0,2), (0,4)]$

$[(0,0), (1,1),$
$(1,3), (2,4)$
$(4,4), (3,7)]$

-1, 0, 1, 2, 4

if:
return N

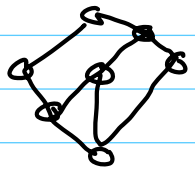
0, 1, 2, 3, 4, 7

5, 6

Damn

MST (connected)

Prims:
connected tree
connect by smallest



Kruskals:
does not make a cycle, always choose smallest edge

$\Theta(|V| \cdot |E|)$
$\Theta(n \cdot c)$ $\Theta(a^c)$

Dijkstra Algo

shortest path by greedy
each vertice given score from home vertice
move to vertice with relatively less score. Move to new home vertice and give score to neighboring. (not connected given ∞)

T/F:
1. F
2. T
3. T
4. T
5. F
6. F
7. T
8. T^ F
9. T
10. F

d_3 d_1 d_2 d_4
p_3 p_2 p_4 p_1

B 1

0 — 1 — 2 — 3 — 4

1. sort jobs according to profit descending
let job_assignment(X):
create new list for slot assignment
set current as first in sorted list
for i in X:
if (current.d_i - 1, current.d_i) not in list:
assign slot to i
current = current.next

Chatgpt question

T/F

1. F ✓
2. T ✓
3. T ✗
4. T ✓
5. T ✓
6. T —
7. F —
8. T —
9. T —
10. T ✗

B. Huffman Coding (symbols, freqs)
create priority queue with asc freq dep on freq
priority is the freq $n \log n$
for i in i=1 to n-1: $O(n)$
left = symbol of first in Q
right = symbol of second in Q
joined = left + right
insert node joined into Q
leftover is the root
assign lefts and rights values of 0s and 1s by traversing
return set of (symbol, code) pairs
$O(n \log n)$

Task scheduling:

weight w_i and time t_i

So we are give w_i and t_i for each i

we find t_i / w_i for each i

```
def scheduling (X):
    sort X by t_i/w_i in ascending order
    current_time = 0
    for i sorted:
        1. schedule i to run from
           current_time
        2. time[i] = current_time + p_i
        3. current_time = current_time + p_i
    return schedule
```