Greedy Algorithms

Make the local optimal choice w/o considering all options AND it works!

Tricky: Proving that they are correct.
If one exists, finding it can be difficult.
Knowing when no greedy solution exists is tricky.
Implementation tends to be easy + fast.
IMPORTANT: TESTING

Single Room Scheduling

list

#	S	e	
1	20	30	
2	10	15	
3	5	25	✓
4	25	40	✓
5	5	18	
6	19	24	

] example schedule

① Sort Data
② Loop Data
Add event if we can

Idea: Start Time → this example proves wrong

5 150
6 7
7 8 etc.
8 9

Idea: Sort by length

Our example

10 15	5	✓
19 24	5	✓
20 30	10	X
5 18	13	X
25 40	15	✓
5 25	20	X

for this one, length work

Counter-ex

150	152
5	151
151	1000

Can't sort by length

Correct Alg: Sort by end time

10	15	✓
5	18	X
19	24	✓
5	25	X
20	30	X
25	40	✓

my ans Alg A

Alg A' better ans

Proof by contradiction

If A' beats A, then there's an earliest time when A' has more events completed

But in beginning A' has not beat us.

After the 1st event that A schedules A' isn't ahead.

Assume we're (A) equal or ahead after k events, prove it's equal or ahead after k+1 events.

I can schedule anything for my k+1st event that A' can. I take the one that finishes first so A' can't get ahead of me.

Coin Changing

Coins ~~demo~~ denominations are each multiples of the previous

1, 3, 6, 24, 48, ~~240~~

make change 700¢ →

$2 \times 240¢$
$4 \times 48¢$
$+1 \times 24¢$
$+1 \times 3¢$
$+1 \times 1¢$

9 coins

700
-480
220
192
28
24
4

True for US coins
1, 5, 10, 25 to
prove brute force to 50

Find a denomination set that doesn't satisfy this property for which the greedy alg fails

1¢, 7¢, 8¢

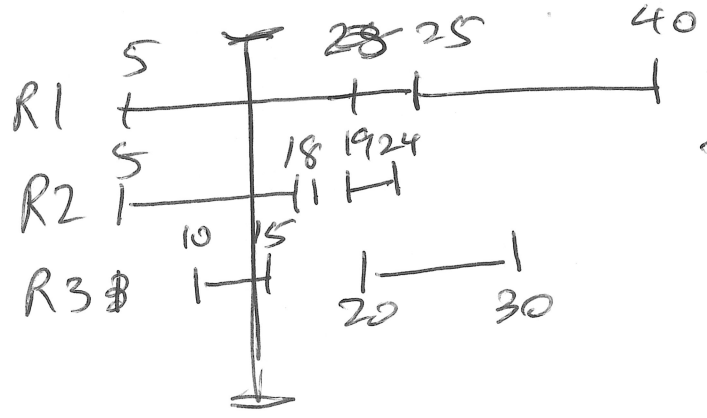
$21¢ = 2 \times 8¢ + 5 \times 1¢ = 7$ coin
best = $3 \times 7¢$ ✓

Multiple Room Scheduling

Same Input as Single Room.
Access to unlimited # rooms.
Minimize # rooms you use, schedule all events.

① Sort by Start Time
② for each event if a room is open, put the event in that room
OR if not open new room

5 25 40
5 18
10 15
19 24
20 30
25 40



3 rooms

Pf: look @ time when last room opened. At that time k events were simultaneously running

In code: (15, -1)

(5,1), (5,1), (10,1), (18,-1), (19,1), (20,1), (25,-1)
(25,-1), (25,1), (30,-1), (40,-1)

```
res = 0
for (x : Obj)
    cur += x.code; //(1 or -1)
    res = max(cur, res)
```

Implementation Trick: Mult by 2 make start times even, end times odd. (To address off by 1 issues.)

fractional knapsack

knapsack capacity W (max weight can carry)

#	W	value/unit
1	10	$25
2	5	$6
3	8	$100
4	25	$5
5	6	$75

W = 22

14
8

8 x 100 = $800
6 x 75 = $450
8 x $25 = $200
$1450

Algorithm: Sort by descending value/unit. Take as much as you can in order.

Note: if you have to take whole item, this doesn't work any more.

Containers

Receiving containers: A, B, C, ..., Z
Stack containers but can't place a larger container on top of a smaller one.

B D A C B C B A C
✓ ✓ = =

B
A B
A C C
B D C

When placing an item place on smallest possible top of stack.

P
EG

D
EG

Pic1 Pic2

Pic1 equal or better Pic2 (all freedom + bit more)

Huffman Coding

Compression Scheme for a file that uses fixed length encoding to change it to use variable length encoding where no code is a prefix of another code (prefix free code)

Reg Ascii

'A' : 01000001
'Z' : 01011010

1000 As
5 Zs

~~neco~~ net savings = 4x1000 - 2x5
= 3990 bits!

Variable Length Code

'A' : 1011
'Z' : 0010010110

each 'A' we save 4 bits
each 'Z' lose 2 bits

Note if A = 1011, no code can start 1011
all codes that start w/101 must be 1010...

Use Huffman on an individual file

Step 1: count up each symbol's frequency

Step 2: Use those frequencies to create our binary code.

Step 3: Store the code in some fashion (in zipfile) followed w/ the Huffman encoding.

How to Make Huffman Tree

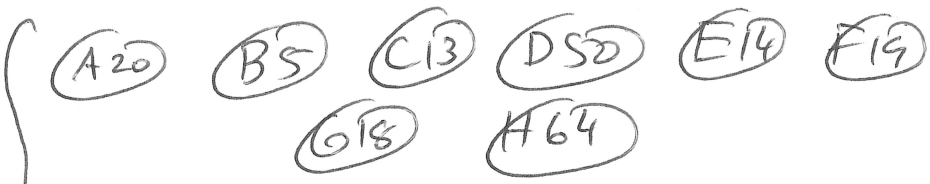
88
30
118

82

000	A	20
001	B	5
010	C	13
011	D	50
100	E	14
101	F	19
110	G	18
111	H	64

orig:
200×3
600 bits

(1) Create a node for each letter w/ its freq

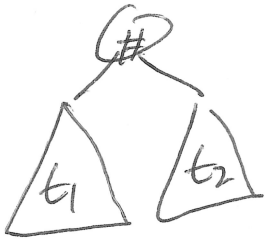


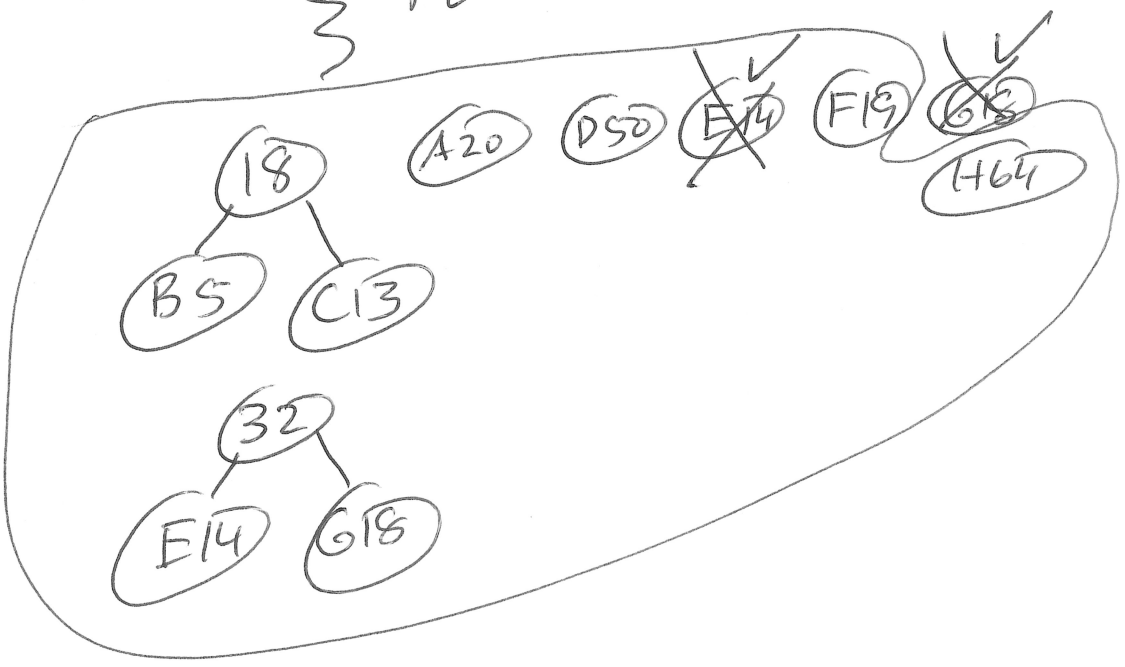
Put these in a PQ by freq of trees.

(2) while (pq.size() > 1) {

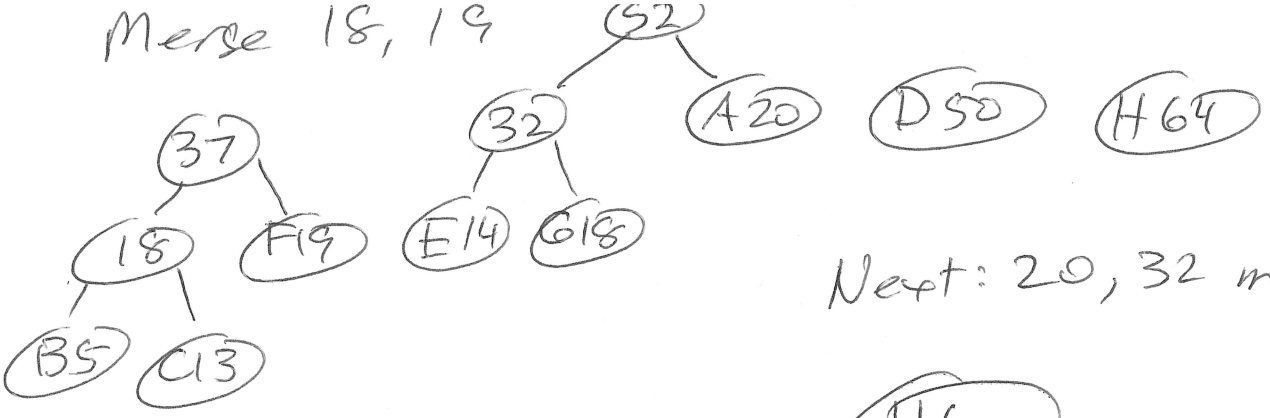
```
tree t1 = pq.poll();
tree t2 = pq.poll();
tree t3 = merge(t1, t2);
pq.offer(t3);
```

}

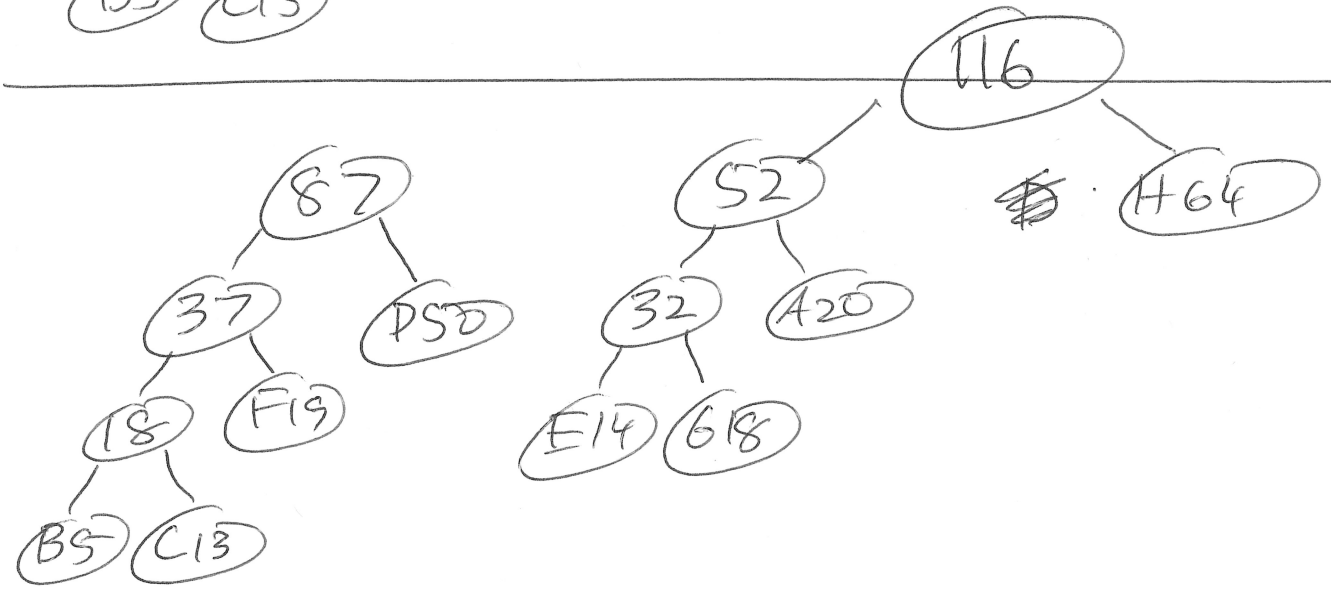




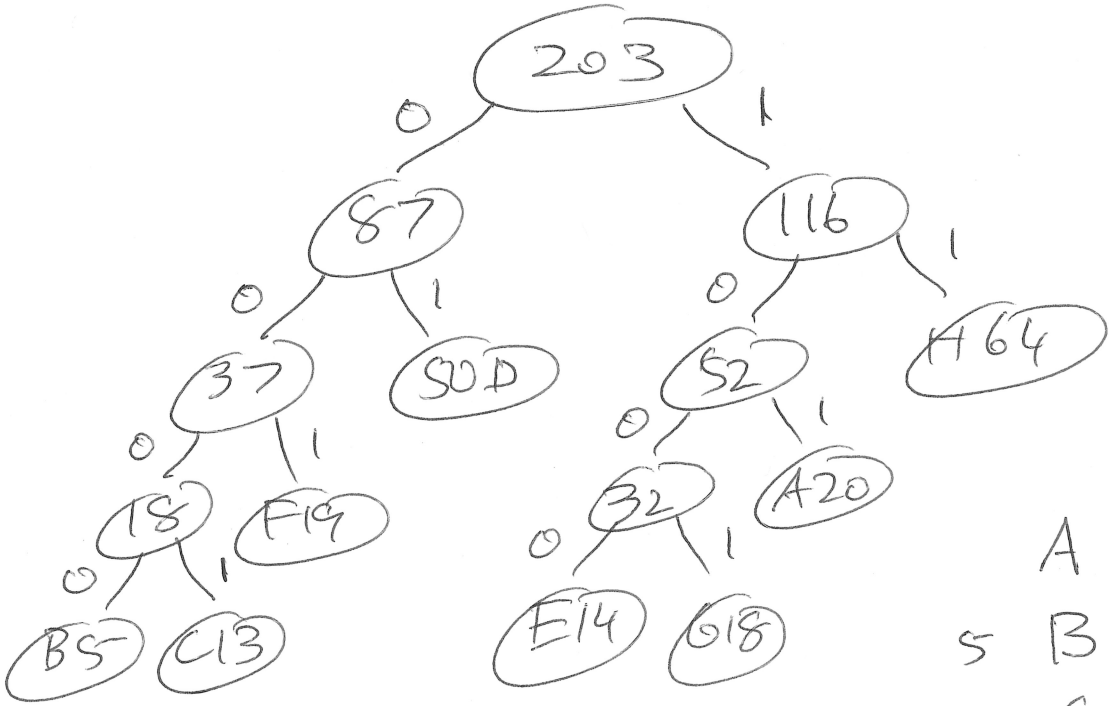
Merge 18, 19



Next: 20, 32 merge



116
87
203



	A	101
5	B	0000
13	C	0001
50	D	01
14	E	1000
	F	001
18	G	1001
64	H	11

Save: 50+64-5-13-14-18

114 - 50

64