Divide & Conquer

split into subproblems

sol = combination of subpb-solutions

Greedy Algorithms

PB

Greedy

- Divide (split) / Decide / Break SUBPB
- Solve SubPb
- Sol = combine (SubPb SOL)

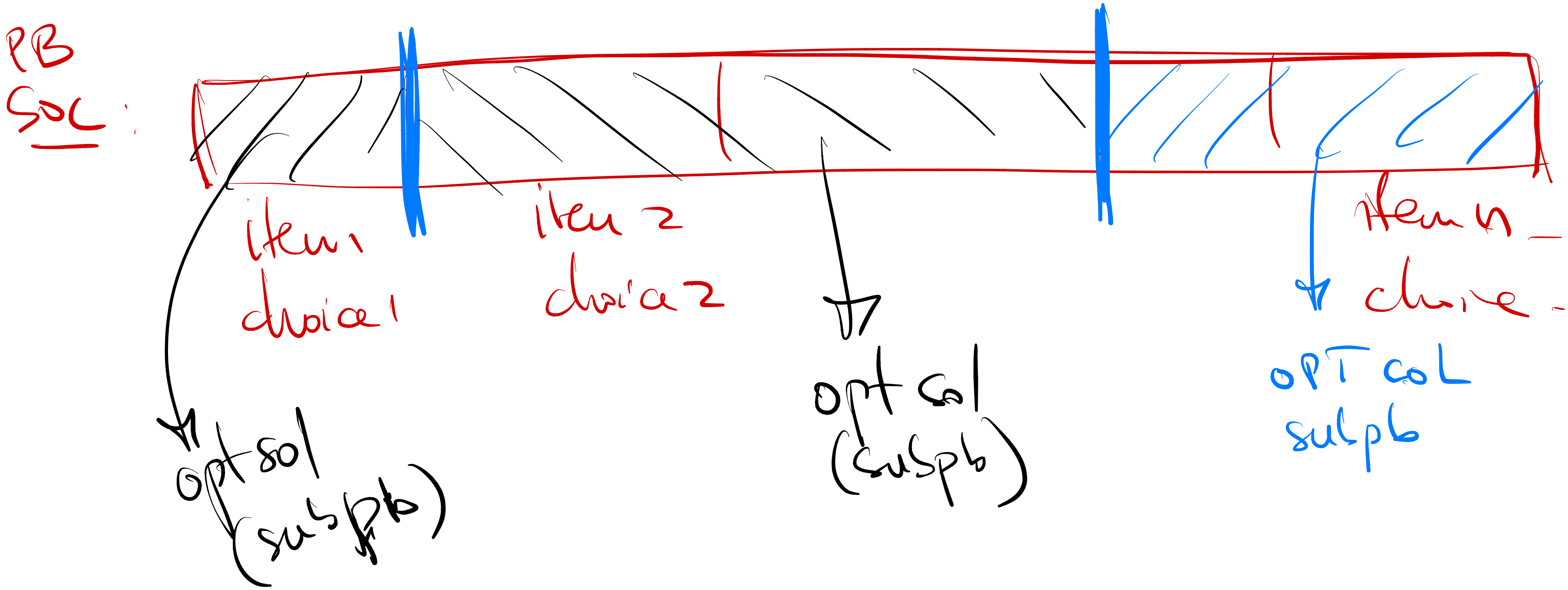
Dynamic Programming.

- Look at all possible SUBPB
  dont know how to break it
- Solve all possible SUBPB
  (even ones we dont need)
- given SOL(subpb) decide the split ⇒ which SUBPB we need
- SOL = comb (selected subpb SOL)

SOL

Sol → Subpb_Sol

$\boxed{\text{Optimal Structure}}$



objective

$$\boxed{C\begin{bmatrix}\text{input}\\ \text{PB}\end{bmatrix} = \text{objective of the pb}} = \text{value}\left(\text{split/decision}\right) + \text{combine function}\left\{C\begin{bmatrix}\text{input}\\ \text{subpb}\end{bmatrix}\right\}$$

Week 5 Objectives

- Subproblem structure
- Greedy algorithm
- Mathematical induction application
- Greedy correctness

Subproblem Optimal Structure

- Divide and conquer - optimal subproblems
- divide PROBLEM into SUBPROBLEMS, solve SUBPROBLEMS
- combine results (conquer)
- critical/optimal structure: solution to the PROBLEM must include solutions to subproblems (or subproblem solutions must be combinable into the overall solution)
- PROBLEM = {DECISION/MERGING + SUBPROBLEMS}

Optimal Structure - GREEDY

- PROBLEM = {DECISION/MERGING + SUBPROBLEMS}
- GREEDY CHOICE: can make the DECISION without solving the SUBPROBLEMS
  - the GREEDY CHOICE looks good at the moment, and it is globally correct
  - example : pick the smallest value
  - solve SUBPROBLEMS after decision is made
- GREEDY CHOICE: after making the DECISION, very few SUBPROBLEMS to solve (typically one)

Optimal Structure - NON GREEDY

dynamic

- Cannot make a choice decision/CHOICE without solving subproblems first
- Might have to solve many subproblems before deciding which results to merge.

Ex: Fractional Knapsack

Greedy

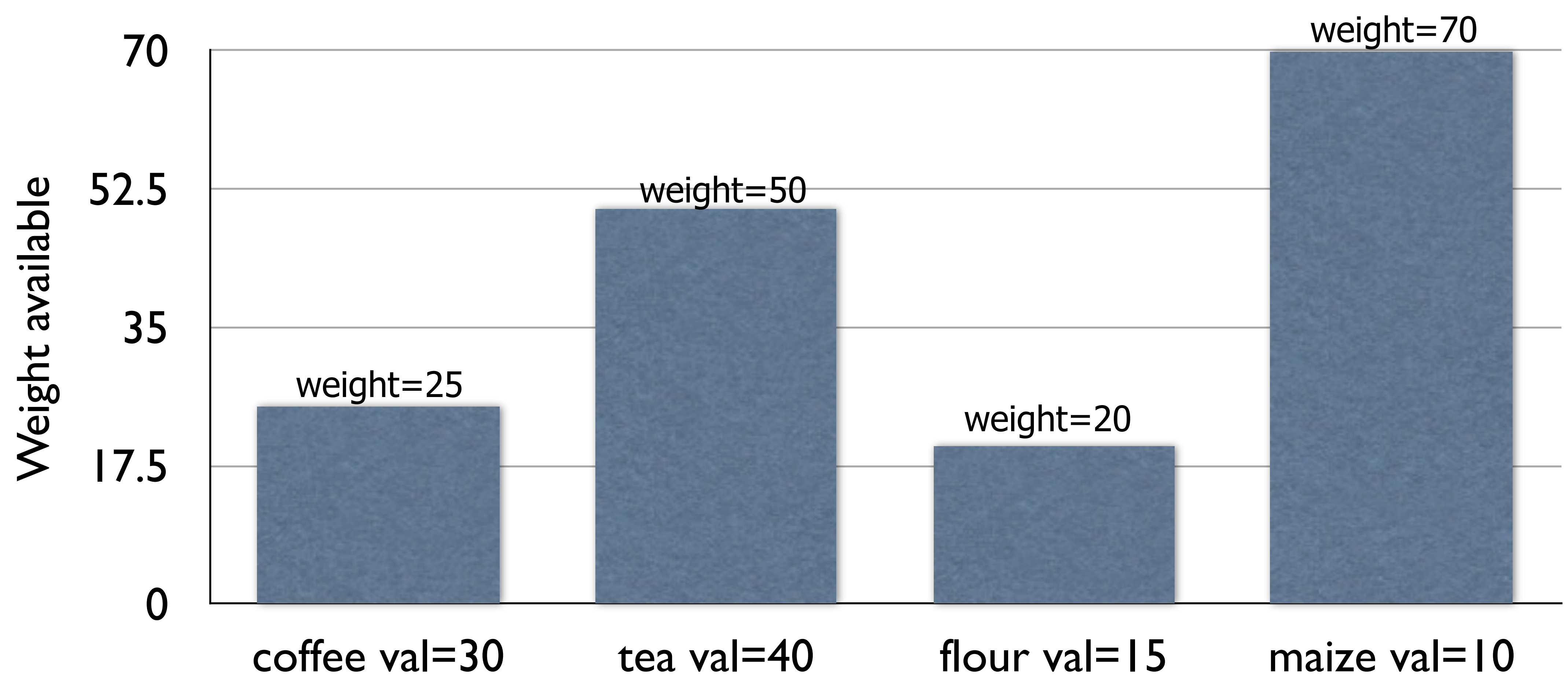
discrete ⇒ DP

discrete : paintings in museum } take it / leave it

- fractional goods (coffee, tea, flour, maize...) sold by weight
- supply (weights/quantities available) w1,w2,w3,w4...
- values (totals) v1,v2,v3,v4...
  - ex: coffee w1=10pounds; coffee overall value v1=$40
- knapsack capacity (weight) = W

  weight constraint
- task : fill the knapsack to maximize value

Ex: Fractional Knapsack

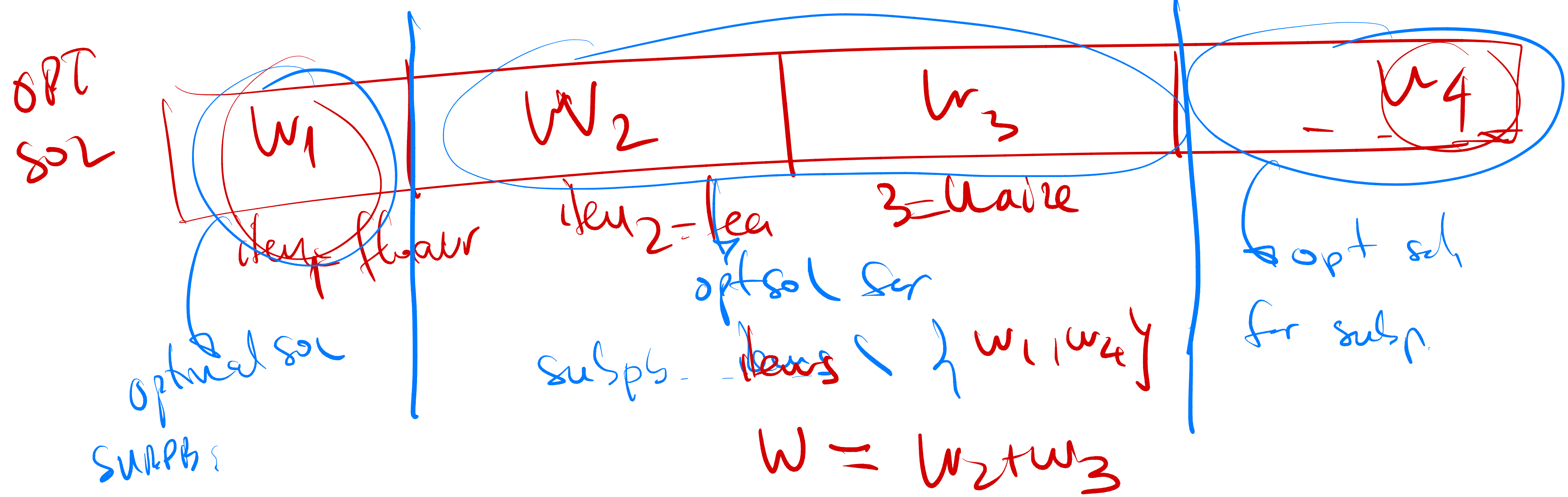


- naive approaches may lead to a bad solution
  - choose by biggest value - tea first
  - choose by smallest quantity - flour first
- choose by quality is correct- Greedy coffee first
  - q_{coffee}=30/25; q_{tea}=40/50; q_{flour}=15/20; q_{maize}=10/70

$\frac{\text{value}}{\text{weight}}$

Fraction vs 0/1b.

⇒ subpb $W-10$; items: all except coffee + coffee ($w_{coff}-10$)



$$C[W, \text{items}] = \boxed{\begin{array}{l} V_{best} \left\{ \underset{}{\text{arg max}} \ \dfrac{v}{w} \right\} \\ W_{best} \ \text{weight} \end{array}}$$

$$+\, C\left[\begin{array}{l} W - w_{best}, \\ \text{items} - \{\text{item best}, w_{best}\} \end{array} \right]$$

• greedy choice criteria

• greedy proof

Ex: Fractional Knapsack

- solution: compute item quality (value/weight)
- $q_i = v_i / w_i$
- sort items by quality q1>q2>q3>…
- LOOP
  - take as much as possible of the best quality
  - if knapsack full, STOP
  - if stock depletes (knapsack not full), move on to the next quality item, repeat
  - END LOOP

Proofs

• exchange

global sol exchange => better value

• induction

after k Greedy steps => part of OPT SOL

• stay ahead

after k Greedy steps I have the same choices (or better) as other solutions

Fractional Knapsack - greedy proof

- proving now that the greedy choice is optimal
  - meaning that the solution includes the greedy choice.
- greedy choice: take as much as possible form best quality (below item with quality q1)
  - items available sorted by quality: q1>q2>q3>..., greedy choice is to take as much as possible of item 1, that is quantity w1
- contradiction/exchange argument
  - suppose that best solution doesnt include the greedy choice: SOL=(r1,r2,r3,...) quantities chosen of these items, and that r1 is not the max quantity available (of max quality item), r1<w1
  - create a new solution SOL' from SOL by taking more of item 1 and less of the others
    - e=min(r2,w1-r1); SOL'=(r1+e,r2-e,r3,r4...)
    - value(SOL') - value(SOL) = e(q1-q2)>0 which means SOL' is better than SOL: CONTRADICTING that SOL is best solution

Fractional Knapsack - greedy proof

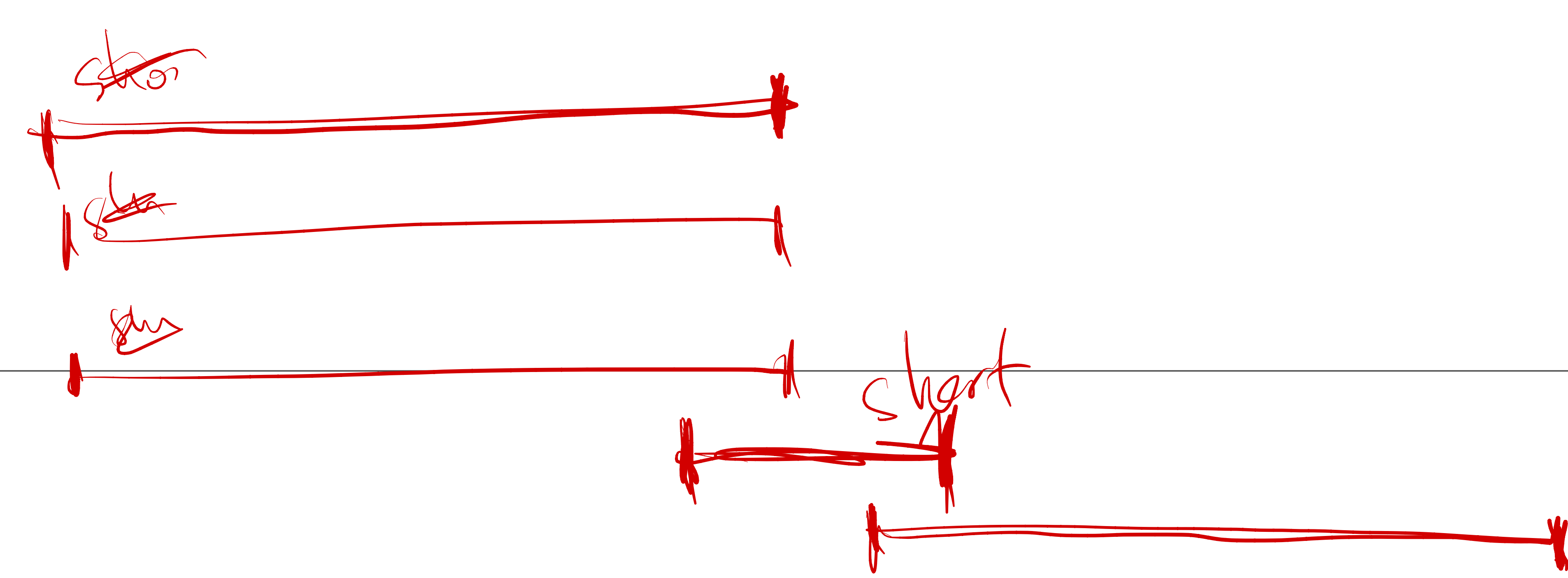
- english explanation:
  - say coffee is the highest quality,
  - the greedy choice is to take max possible of coffee which is w1=10pounds
- contradiction/exchange argument
  - suppose that best solution doesnt include the greedy choice: SOL=(8pounds coffee, r2 of tea, r3 flours,...) r1=8pounds<w1=10pounds
  - create a new solution SOL' from SOL by taking out 2pounds of tea and adding 2 pounds of coffee; e=2pounds
    - e=min(r2,w1-r1); SOL'=(r1+e,r2-e,r3,r4...)
    - value(SOL') - value(SOL) = e(q1-q2)>0 which means SOL' is better than SOL: CONTRADICTING that SOL is best solution

Activity Selection Problem

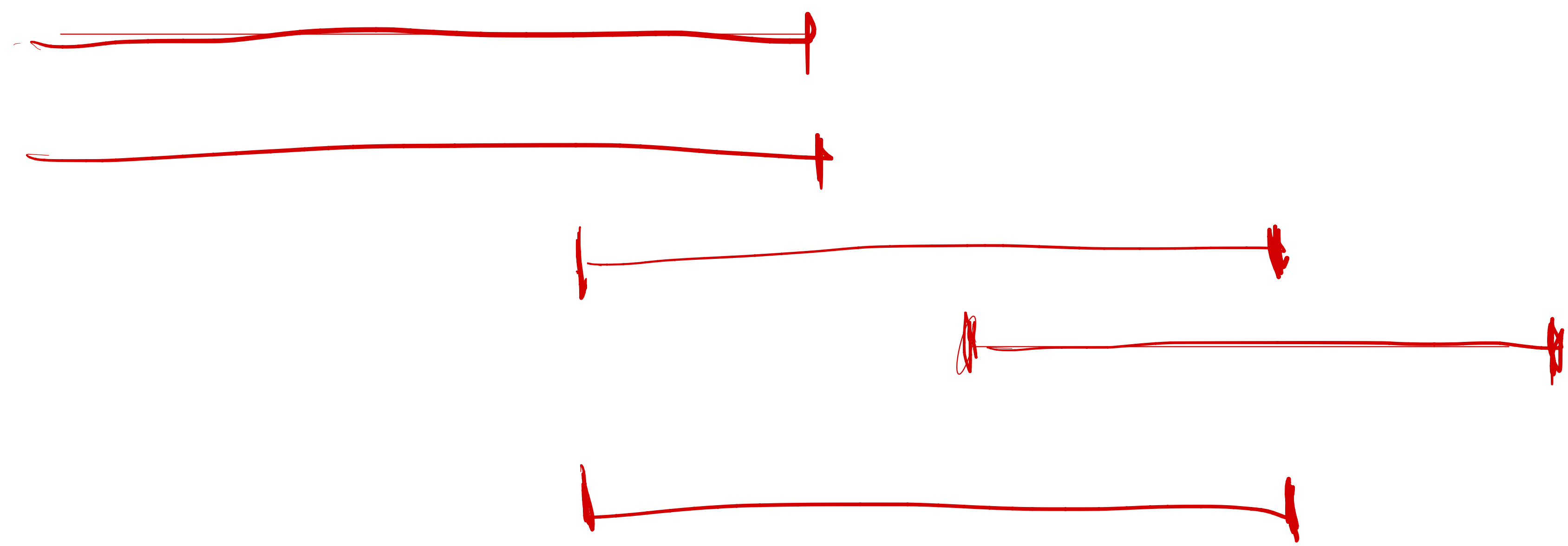
- S=set of n activities given by start and finish time $a_i = (s_i, f_i)$ $i=1:n$, $f_i > s_i$
- Determine a selection that gives a maximal set
  - select maximum number of activities
  - no overlapping activities can be selected

Criteria: shortest

INCORRECT



criteria: least overlap (# of ∩ activities)



Activity Selection Problem

- Greedy solution: sort activities by their finishing time
  - f1<f2<f3...
  - select the activity that finishes first $a = (s_1, f_1)$
  - discard all overlapping activities with selected one : discard all activities with starting time $s_i < f_1$
  - repeat
- intuition: activity that finishes first is the one that leaves as much time as possible for other activities

Activity Selection Problem

- Proof of greedy choice optimality
  - activities sorted by finishing time f1<f2<f3…
  - greedy choice pick the activity a with earliest finishing time f1
  - want to show that activity a is included in one of the best solutions (could be more than one optimal selection of activities)
- Exchange argument
  - SOL a best solution.
  - if SOL includes a, done.
  - suppose the best solution does not select a, SOL= (b,c,d,…) sorted by finishing time $f_b<f_c<f_d$. Then create a new solution that replaces b with a SOL'=(a, c, d,…).
    - This solution SOL' is valid, a and c dont overlap: $s_c>f_b>f_a$
    - SOL' is as good as SOL (same number of activities) and includes a

Mathematical Induction

- property $P(n) = \{TRUE, FALSE\}$ for n=integer
  - want to prove P(n)=TRUE for all n
- Base cases: P(n)=TRUE for any $n \leq n_0$
- Induction Step: prove P(n+1) for next value n+1
  - if P(t)=TRUE for certain values of t<n+1 then prove by mathematical derivation/arguments than P(n+1)=TRUE
- Then P(n) = TRUE for all n

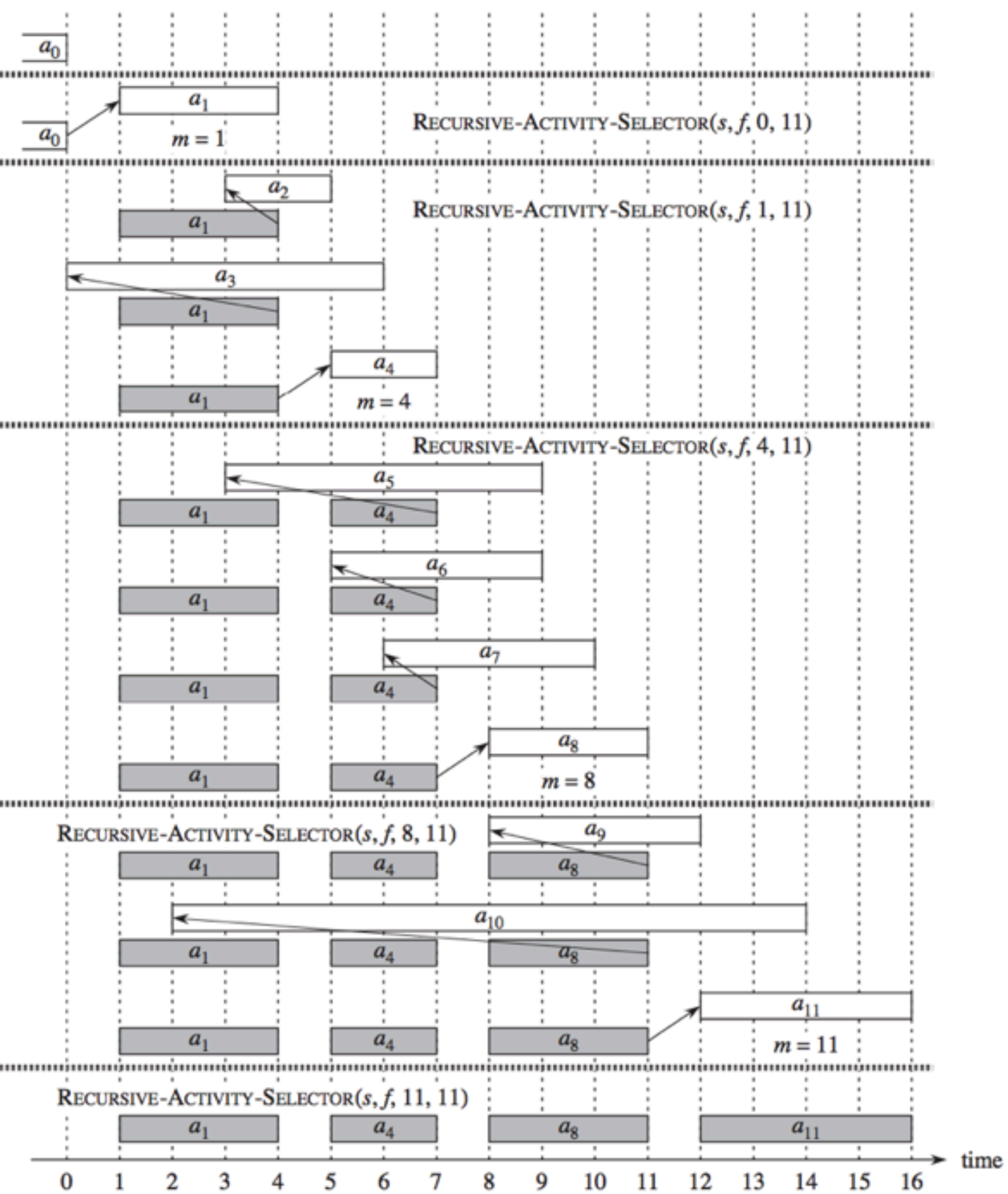
Mathematical Induction- Example

- $P(n)$: $1+2+3+\dots+n = n(n+1)/2$
- base case $n=1$: $1=1*2/2$ - correct
- induction step : lets prove $P(n+1)$ assuming $P(n)$
  - $P(n+1)$: $1+2+3+\dots+n + (n+1) = (n+1)(n+2)/2$.
  - assuming $P(n)$ TRUE : $1+2+3\dots+(n+1) = [1+2+3+\dots+n] + (n+1) = n(n+1)/2 + (n+1) = (n+1)(n+2)/2$; so $P(n+1)$ TRUE
- thus $P(n)$ TRUE for all $n>0$

Activity Selection - Induction Argument

- $s(a)$= start time; $f(a)$=finish time
- SOL=$\{a_1,a_2,\ldots,a_k\}$ greedy solution
  - chosen by earliest finishing time
- OPT = $\{b_1,b_2,\ldots,b_m\}$ optimal solution, sorted by finishing time; optimal means m max possible
- prove by induction that $f(a_i)\leq f(b_i)$ for all i=1:k
  - base case $f(a_1)\leq f(b_1)$ because $f(a_1)$ smallest in the whole set
  - inductive step: assume $f(a_{n-1})\leq f(b_{n-1})$. Then b_n is a valid choice for greedy at step n because $f(a_{n-1})\leq f(b_{n-1})\leq s(b_n)$. Since greedy picked a_n over b_n, it must be because an fits the greedy criteria $f(a_n)\leq f(b_n)$
- so $f(a_k)\leq f(b_k)$. If m>k then any b_{k+1} item would also fit into greedy solution (CONTRADICTION) thus m=k

k	s_k	f_k
0	–	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



Q1*: OBJ = max total act time

idea: greedy: earliest start time

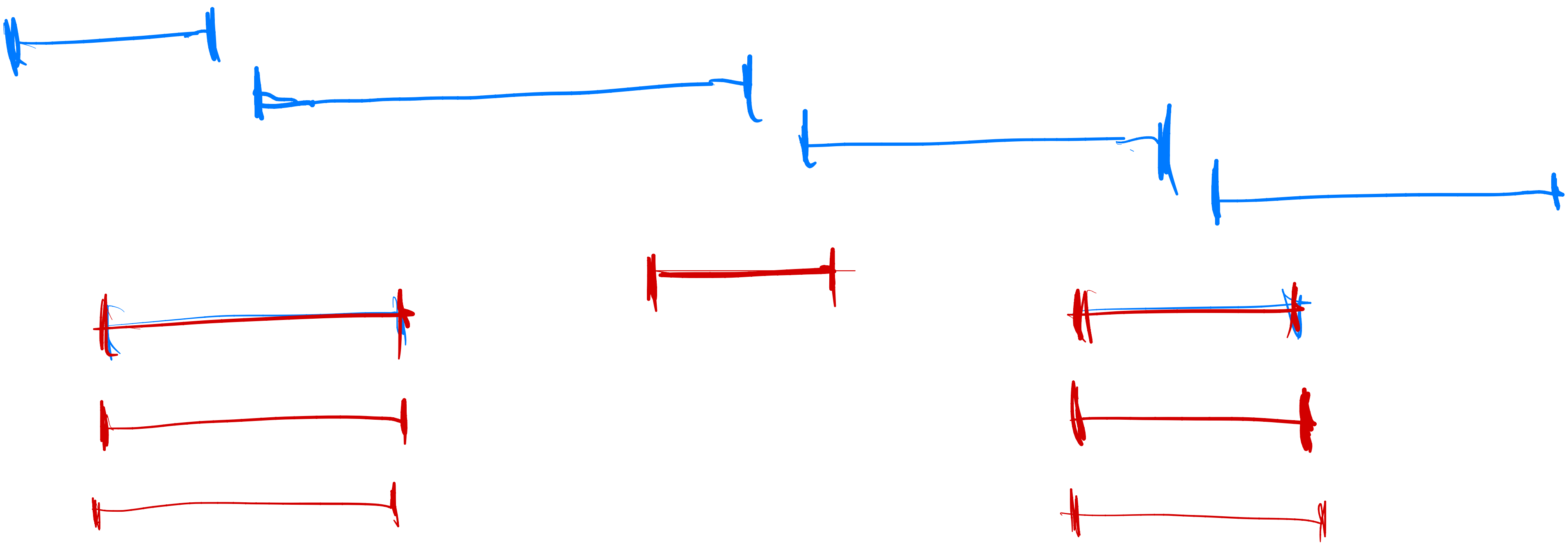
Q2**: at any moment max 3 activities can overlap

RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)

```
m = k + 1
while m ≤ n and s[m] < f[k]        // find
    m = m + 1
if m ≤ n
    return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
else return Ø
```

$C\begin{bmatrix} s_i : s_n \\ f_i : f_n \end{bmatrix} = \underset{\text{value}}{\underbrace{i = \text{greedy choice } \arg\min(f)}} \; 1 + C\begin{bmatrix} s_i : s_n \setminus \{s_i\} \\ f_i : f_n \setminus \{f_i\} \end{bmatrix} \setminus \text{overlapping activ. with } i$

Criteria: min overlap



Coin Change

Denominations = $\{d_1, d_2, \ldots d_n\}$

$\boxed{\text{example } \{1, 5, 10, 25\}}$

infinite supply...

T cents,

Task: min # coins sum = T

T = 24 Greedy: 10, 10, 1, 1, 1, 1 OBJ = 6

Greedy: choose highest coin that fits in T

$$C[D, T] = \underset{\text{Greedy } d_i}{1} + C[D, T - d_i]$$

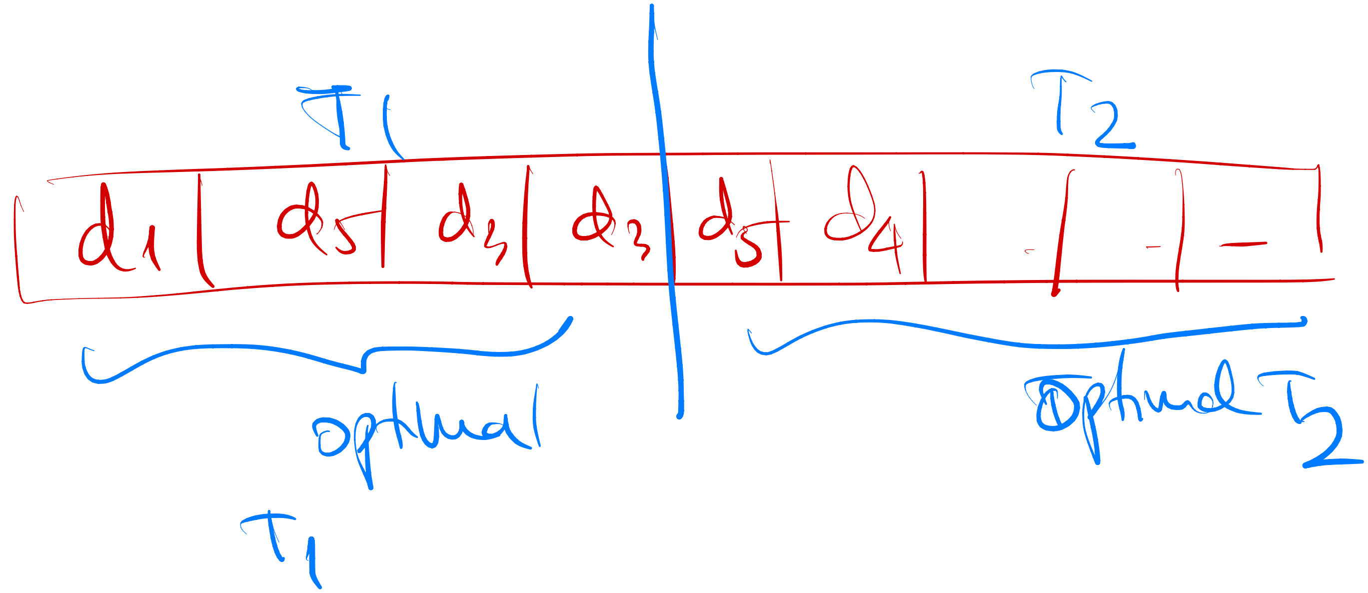
$D = \{1, \cancel{3}, 4, 5, 7, 10, 25\}$

non-greedy

***: What D (denominations sets) allow greedy solutions?

General case

OPT sol T



Gas Stations

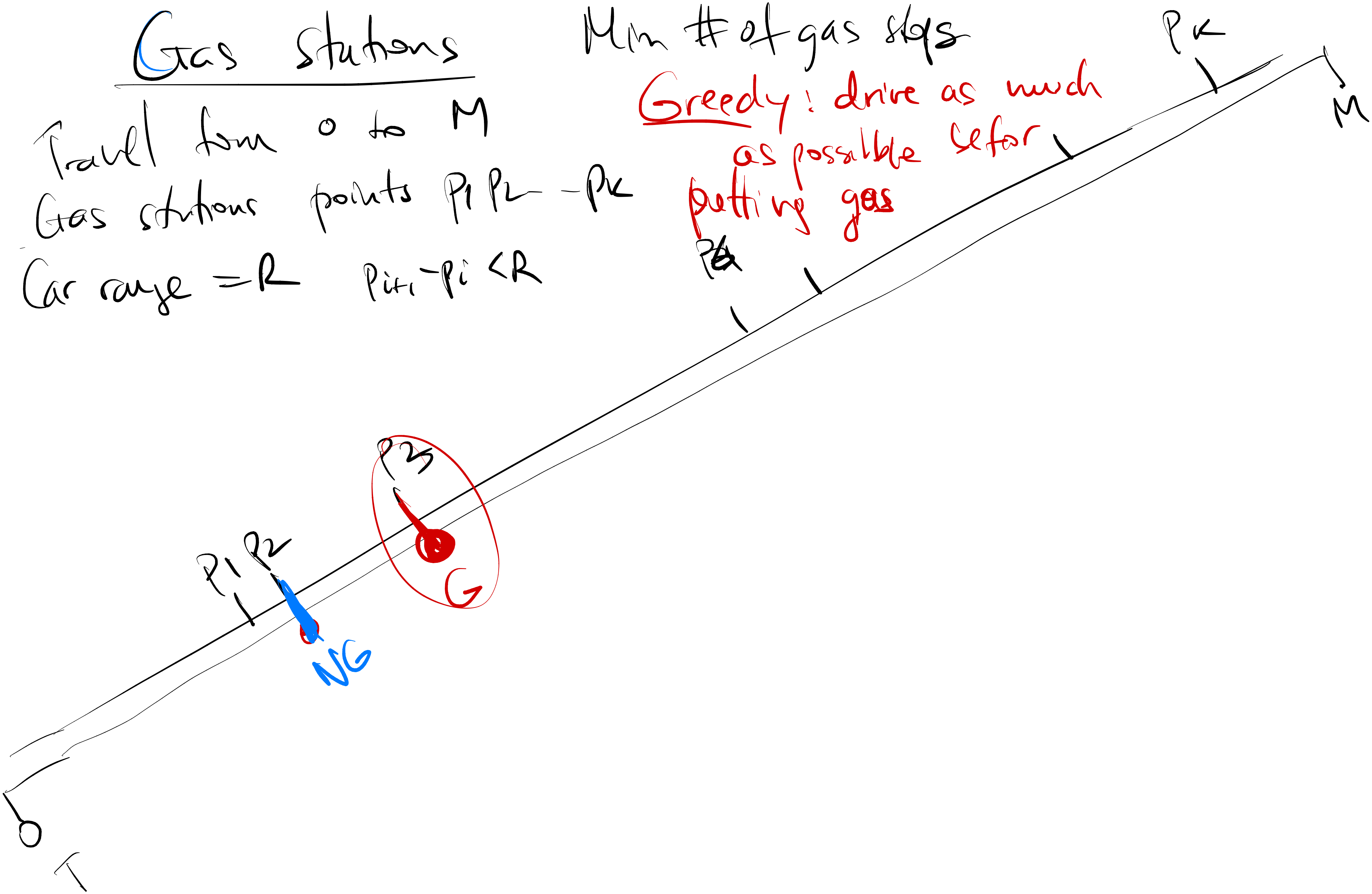
Min # of gas stops

Travel from 0 to M

Gas stations points $p_1\ p_2 - - p_k$

Car range $= R$ $\quad p_{i+1} - p_i < R$

Greedy: drive as much as possible before putting gas



Th Any fraction $\in \mathbb{Q} = \sum$ egyptian fractions

egypt. fraction $= \dfrac{1}{n_{\in \mathbb{N}}}$

CS task: decompose fraction $\dfrac{a}{b} = \sum_{i=1}^{k} \dfrac{1}{n_i}$

$$\frac{5}{6} = \frac{1}{2} + \frac{1}{3}$$

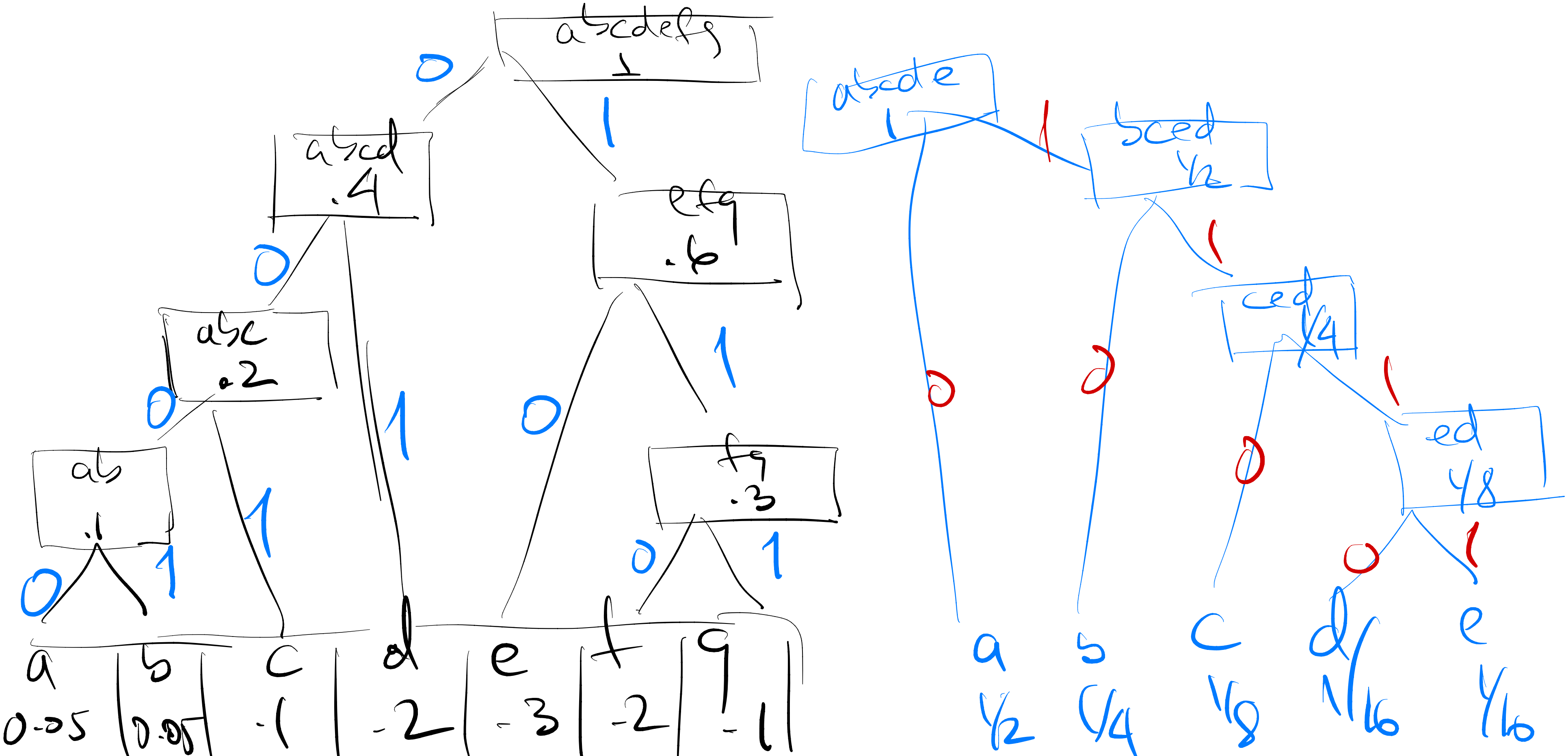
$$\boxed{\frac{7}{15}} = \frac{1}{3} + \boxed{\frac{2}{15}} = \frac{1}{3} + \frac{1}{8} + \frac{1}{120}$$ ✓

greedy criteria

subpb = smaller numerator

$$\frac{5}{121} = (\text{best})\ \frac{1}{33} + \frac{1}{121} + \frac{1}{363}$$

Greedy: $\dfrac{5}{121} = \dfrac{1}{25} + \dfrac{1}{757} + \dfrac{1}{763309} + \dfrac{1}{5\ldots} + \dfrac{1}{5\ldots}$



abcdefg
1
abcd
.4
efg
.6
abc
.2
ab
.1
fg
.3
a b c d e f g
0.05 0.05 .1 .2 .3 .2 .1
abcde
1
bced
1/2
ced
1/4
ed
1/8
a b c d e
1/2 1/4 1/8 1/16 1/16

Huffman Codes : Task encode char → string of bits

var. length

given char probas

a	b	c	d	e	f	g
.05	.05	.1	.2	.3	.2	.1

ex2

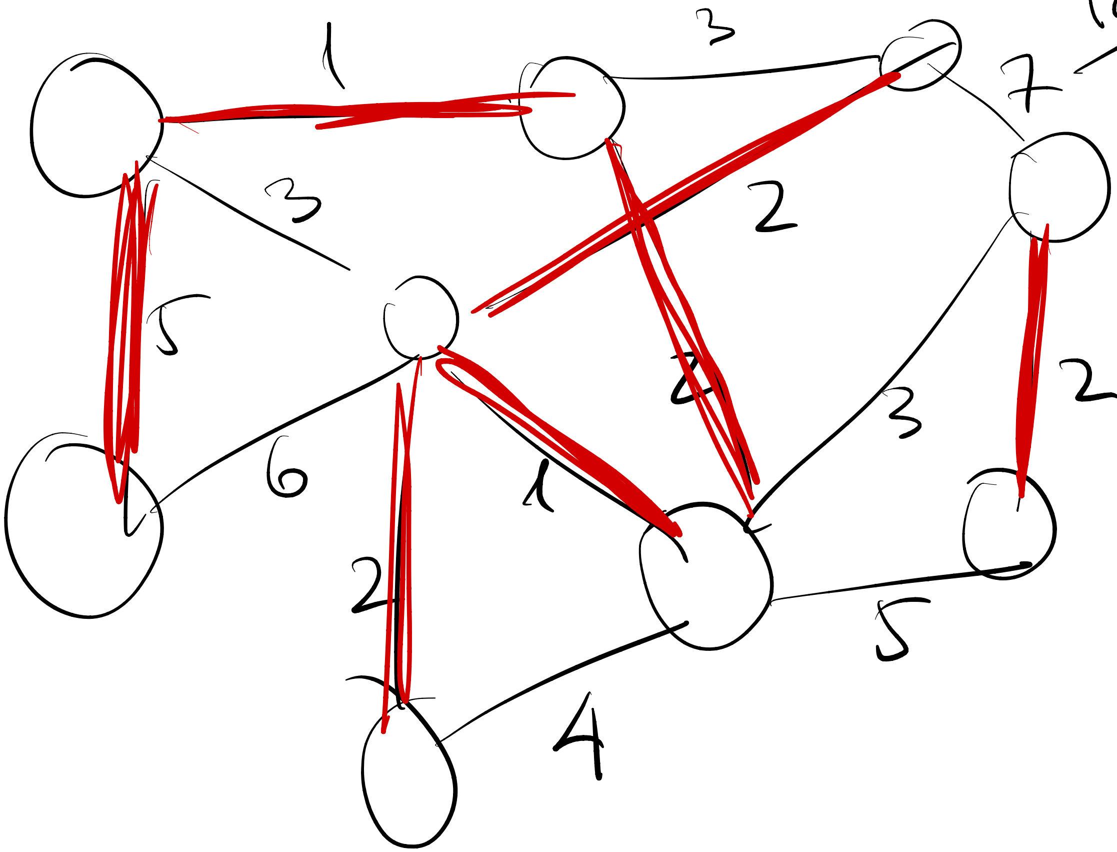
a	b	c	d	e
1/2	1/4	1/8	1/16	1/16

OBJ : minimize E[# of bits/char] in encoding.

$$= \sum_i p(c_i) \cdot |\text{encode\_length}_{c_i}|$$

constraint: possible to decode "prefix-free code"

MST : Undirected graph $G=(V,E)$ edge weight



Task: find subset of edges $\subset E$

- connects all vertice
- no cycles
  $\Rightarrow$ tree
- total sum min