CSC373

Weeks 2 & 3: Greedy Algorithms

Nisarg Shah

Recap

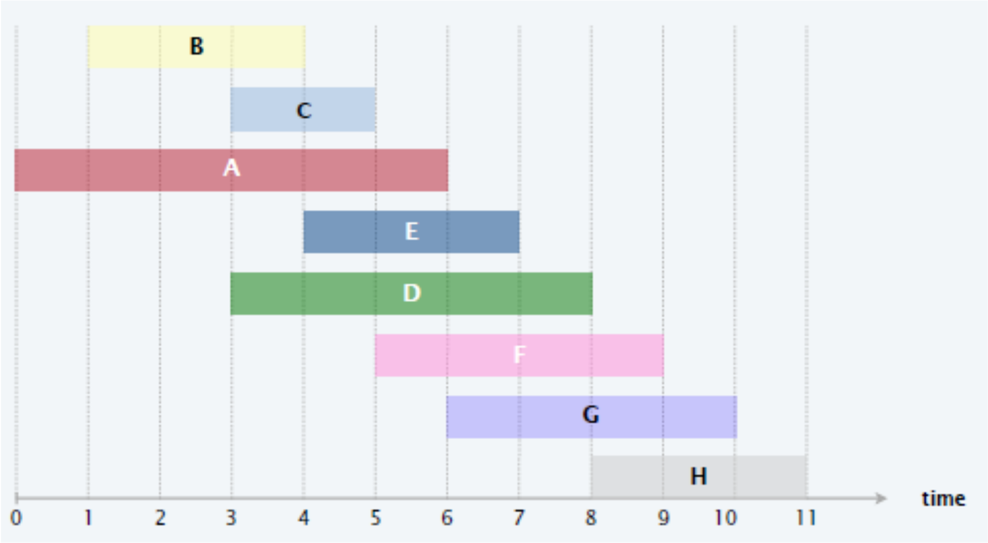
- Divide & Conquer
  - Master theorem
  - Counting inversions in $O(n \log n)$
  - Finding closest pair of points in $\mathbb{R}^2$ in $O(n \log^2 n)$
    - Can be improved to $O(n \log n)$
  - Fast integer multiplication in $O\left(n^{\log_2 3}\right)$
  - Fast matrix multiplication in $O\left(n^{\log_2 7}\right)$
  - Finding k^{th} smallest element in $O(n)$
    - Can be used for finding the median in $O(n)$ time

Greedy Algorithms

- Greedy/myopic algorithm outline
  - Goal: find a solution x maximizing/minimizing objective function f
  - Challenge: space of possible solutions x is too large
  - Insight: Computing x requires taking several decisions (e.g., decide to either keep or discard each element of a set)
  - Approach: Instead of taking all the decisions together, take them one at a time
    - Take the next decision "greedily" to maximize the immediate "benefit" without knowing how you'll take future decisions
    - Most greedy algorithms trivially run in polynomial time, but require a proof that they will always return an optimal solution

Interval Scheduling

- Problem
  - Job j starts at time s_j and finishes at time f_j
  - Two jobs i and j are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ don't overlap
    - Note: we allow a job to start right when another finishes
  - Goal: find maximum-size subset of mutually compatible jobs



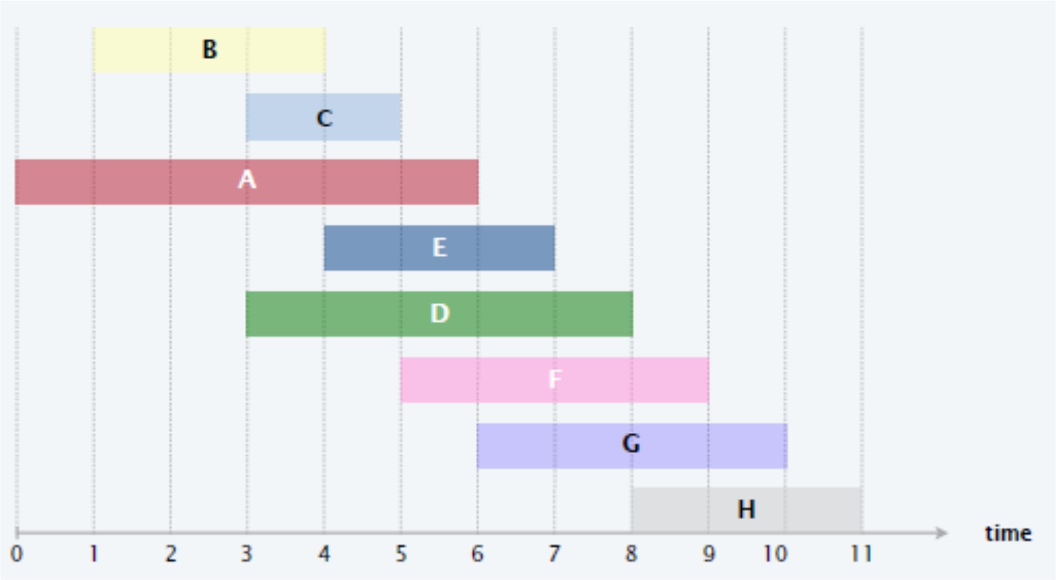
Interval Scheduling

- Greedy template
  - Consider the jobs one-by-one in some "natural" order
  - For each job being considered, take it if it's compatible with the ones already taken

- Question: In what order should we consider the jobs?

Possible Orders

- Earliest start time: ascending order of s_j
- Earliest finish time: ascending order of f_j
- Shortest interval: ascending order of $f_j - s_j$
- Fewest conflicts: ascending order of c_j, where c_j is the number of remaining jobs that conflict with j



Interval Scheduling

- Counterexamples

earliest start time

shortest interval

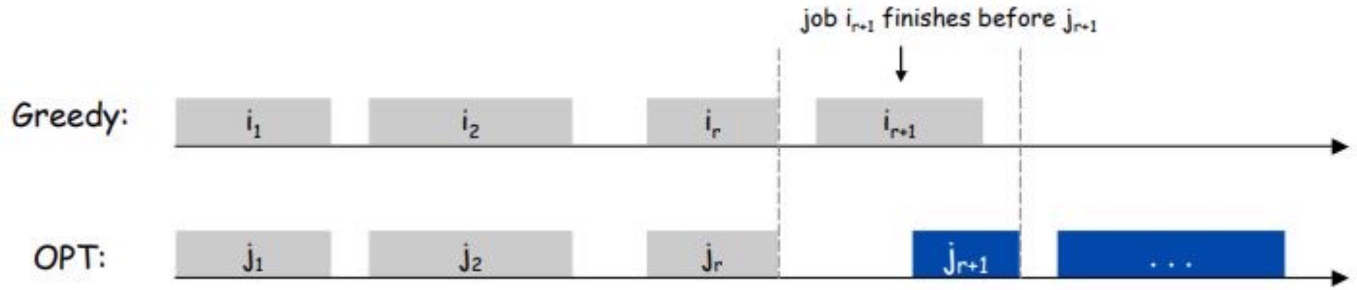
fewest conflicts

Interval Scheduling

- Implementing greedy with earliest finish time (EFT)
  - Sort jobs by finish time, say $f_1 \leq f_2 \leq \cdots \leq f_n$
    - $O(n \log n)$
  - For each job j, we need to check if it's compatible with *all* previously chosen jobs
    - Naively, this can take $O(n)$ time per job j, so $O(n^2)$ total time
    - We only need to check if $s_j \geq f_{i^*}$, where i^* is the *last added job*
      - For any jobs i added before i^*, $f_i \leq f_{i^*}$
      - By keeping track of f_{i^*}, we can check job j in $O(1)$ time
  - Total running time: $O(n \log n)$

Interval Scheduling

- Proof of optimality by contradiction
  - Suppose for contradiction that greedy solution is not optimal
  - Say greedy selects jobs $i_1, i_2, \dots, i_k$ sorted by finish time
  - Consider an optimal solution $j_1, j_2, \dots, j_m$ by finish time which matches greedy for as many indices as possible
    - That is, $j_1 = i_1, \dots, j_r = i_r$ for the greatest possible r



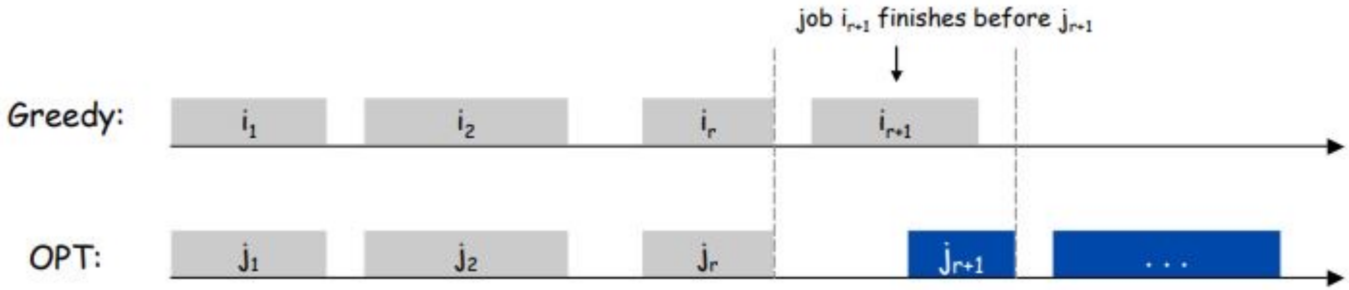
Interval Scheduling

- Proof of optimality by contradiction
  - Claim: $r < k \leq m$
  - Proof:
    - If $r = k$, then OPT selects every job selected by GRD
    - But since we assumed GRD is not optimal, OPT must select at least one more job, which doesn't conflict with any jobs selected by GRD
    - But then GRD would have selected this job too, a contradiction!

  - Hence, both greedy and optimal select at least one job each after their (common) r^{th} job $i_r = j_r$
    - Both i_{r+1} and j_{r+1} must be compatible with the previous selection ($i_1 = j_1, \ldots, i_r = j_r$)

Interval Scheduling

- Proof of optimality by contradiction
  - Consider a new solution $i_1, i_2, \dots, i_r, i_{r+1}, j_{r+2}, \dots, j_m$
    - We have replaced j_{r+1} by i_{r+1} in our optimal solution
    - This is still feasible because $f_{i_{r+1}} \le f_{j_{r+1}} \le s_{j_t}$ for $t \ge r+2$
    - This is still optimal because m jobs are selected
    - But it matches the greedy solution in $r+1$ indices
      - This is the desired contradiction



Interval Scheduling

- Proof of optimality by induction
  - Let G_j be the subset of jobs picked by greedy after considering the first j jobs by increasing finish time
  - If greedy solution is G, then $G_j = G \cap \{1, \dots, j\}$
  - Note that $G_0 = \emptyset$ and $G_n = G$
  - We call G_j *promising* if some optimal solution O_j "extends it"
    - $\exists T \subseteq \{j+1, \dots, n\}$ such that $O_j = G_j \cup T$ is optimal
  - Inductive claim: For all $t \in \{0,1, \dots, n\}$, G_t is promising
  - If we prove this, then we are done since $G = G_n$ is promising, which is the same as $G = G_n$ being optimal (Why?)

Interval Scheduling

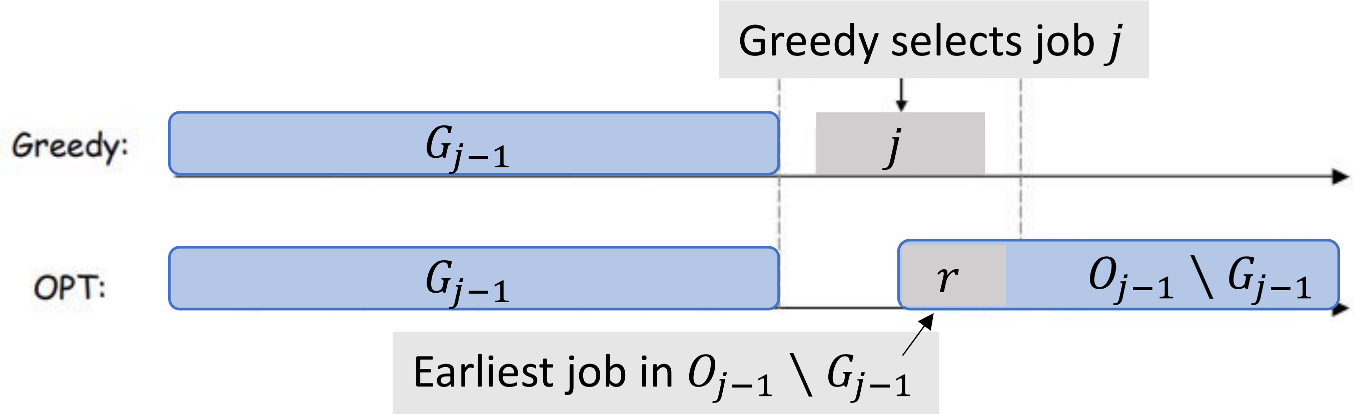
- Proof of optimality by induction
  - Inductive claim: For all $t \in \{0,1,\dots,n\}$, G_t is promising
  - Base case: For $t = 0$, $G_0 = \emptyset$ is trivially promising (Why?)
  - Induction hypothesis: Suppose that for $t = j - 1$, G_{j-1} is promising and optimal solution O_{j-1} extends G_{j-1}
  - Induction step: At $t = j$, we have two possibilities:

    1) Greedy did not select job j, so $G_j = G_{j-1}$
       - Job j must have had a conflict with some job in G_{j-1}
       - Since $G_{j-1} \subseteq O_{j-1}$, O_{j-1} also cannot include job j
       - Hence, $O_j = O_{j-1}$ also extends $G_j = G_{j-1}$

Interval Scheduling

- Proof of optimality by induction
  - Induction step: At $t = j$, we have two possibilities:

    2) Greedy did select job j, so $G_j = G_{j-1} \cup \{ j \}$
      - Consider the earliest job r in $O_{j-1} \setminus G_{j-1}$
      - Note that $f_j \leq f_r \leq s_\ell$ for any job $\ell \in O_{j-1} \setminus (G_{j-1} \cup \{r\})$
      - So $O_j = O_{j-1} \cup \{j\} \setminus \{r\}$ is optimal and extends G ∎



Contradiction vs Induction

- Both methods make the same claim
  - “$\forall j$, the greedy solution after j iterations can be extended to some optimal solution”
  - Proof by induction explicitly proves this inductively
  - Proof by contradiction…
    - Supposes that this is not true
    - Considers the smallest $r + 1$ such that the greedy solution after $r + 1$ iterations cannot be extended to an optimal solution
      - Same as finding an optimal solution that matches greedy for the maximum possible number of iterations r
    - Derives a contradiction by showing that greedy after $r + 1$ can still be extended to some optimal solution
      - Equivalent to the induction step

Contradiction vs Induction

- Choose the method that feels natural to you

- It may be the case that…
  - For some problems, a proof by contradiction feels more natural
  - But for other problems, a proof by induction feels more natural
  - No need to stick to one method

- As we saw for interval partitioning, sometimes you may require an entirely different kind of proof

Interval Partitioning

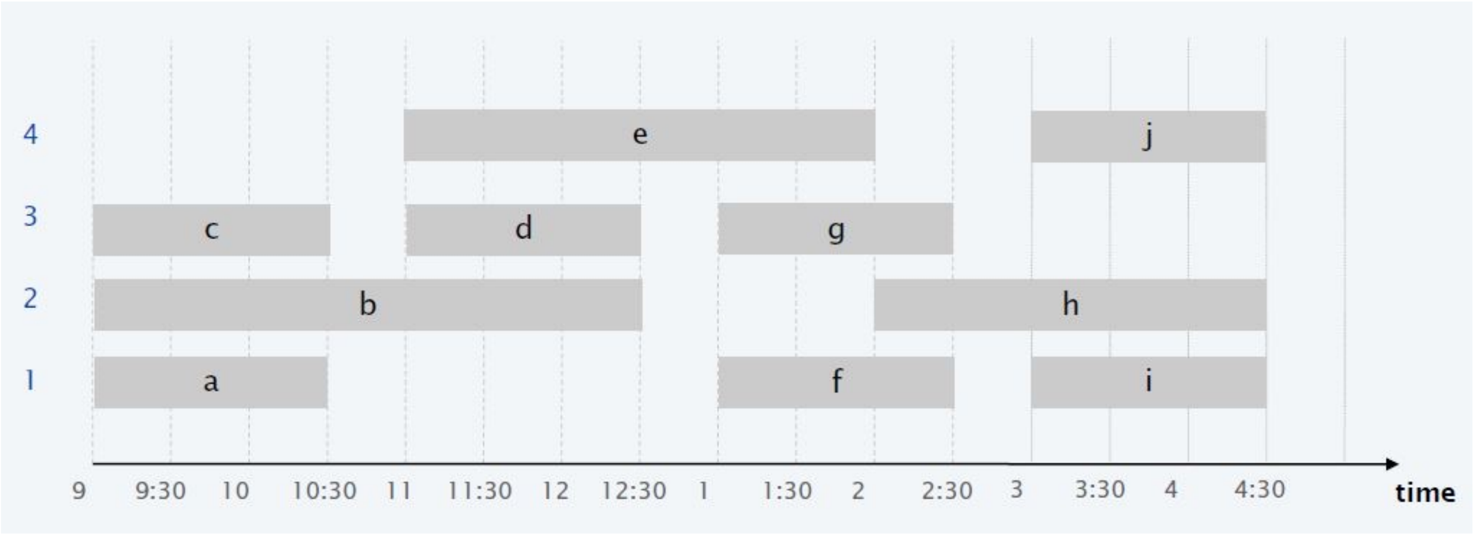
- Problem
  - Job j starts at time s_j and finishes at time f_j
  - Two jobs are compatible if they don't overlap
  - Goal: group jobs into fewest partitions such that jobs in the same partition are compatible

- One idea
  - Find the maximum compatible set using the previous greedy EFT algorithm, call it one partition, recurse on the remaining jobs.
  - Doesn't work (check by yourselves)

Interval Partitioning

- Think of scheduling lectures for various courses into as few classrooms as possible

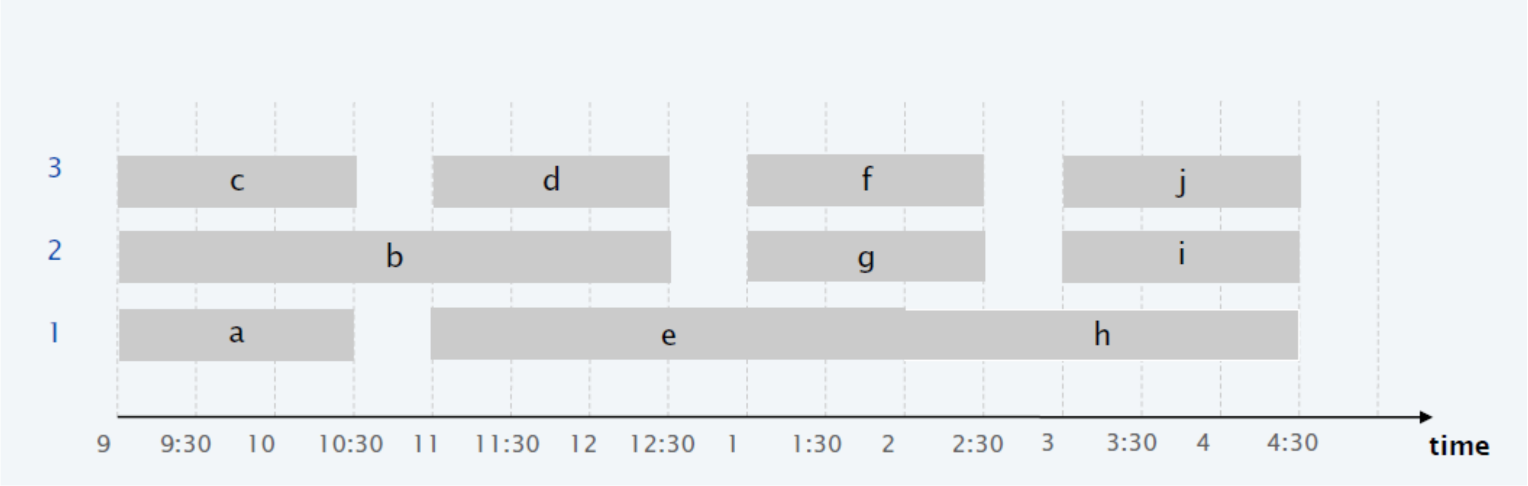
- This schedule uses 4 classrooms for scheduling 10 lectures



Interval Partitioning

- Think of scheduling lectures for various courses into as few classrooms as possible

- This schedule uses 3 classrooms for scheduling 10 lectures

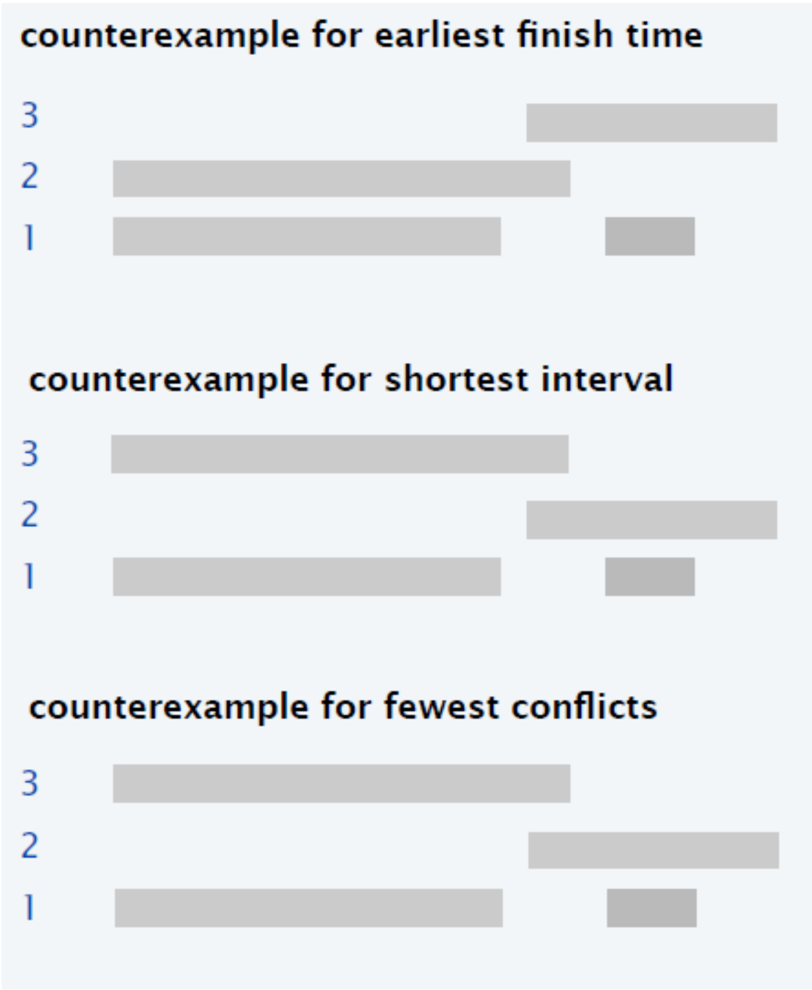


Interval Partitioning

- Let's go back to the greedy template!
  - Go through lectures in some "natural" order
  - Assign each lecture to a used classroom that is compatible (what if there are several?), and use a new classroom if the lecture conflicts with every used classroom

- Order of lectures?
  - Earliest start time: ascending order of s_j
  - Earliest finish time: ascending order of f_j
  - Shortest interval: ascending order of $f_j - s_j$
  - Fewest conflicts: ascending order of c_j, where c_j is the number of remaining jobs that conflict with j

Interval Partitioning



- At least when you assign each lecture to an arbitrary compatible classroom, three of these heuristics do not work.
- The fourth one works! (next slide)

Interval Partitioning

EARLIESTSTARTTIMEFIRST(n, s_1, s_2, ..., s_n, f_1, f_2, ..., f_n)

SORT lectures by start time so that $s_1 \leq s_2 \leq \ldots \leq s_n$.

$d \leftarrow 0$ ← number of allocated classrooms

FOR $j = 1$ TO n

  IF lecture j is compatible with some classroom

    Schedule lecture j in any such classroom k.

  ELSE

    Allocate a new classroom $d + 1$.

    Schedule lecture j in classroom $d + 1$.

    $d \leftarrow d + 1$

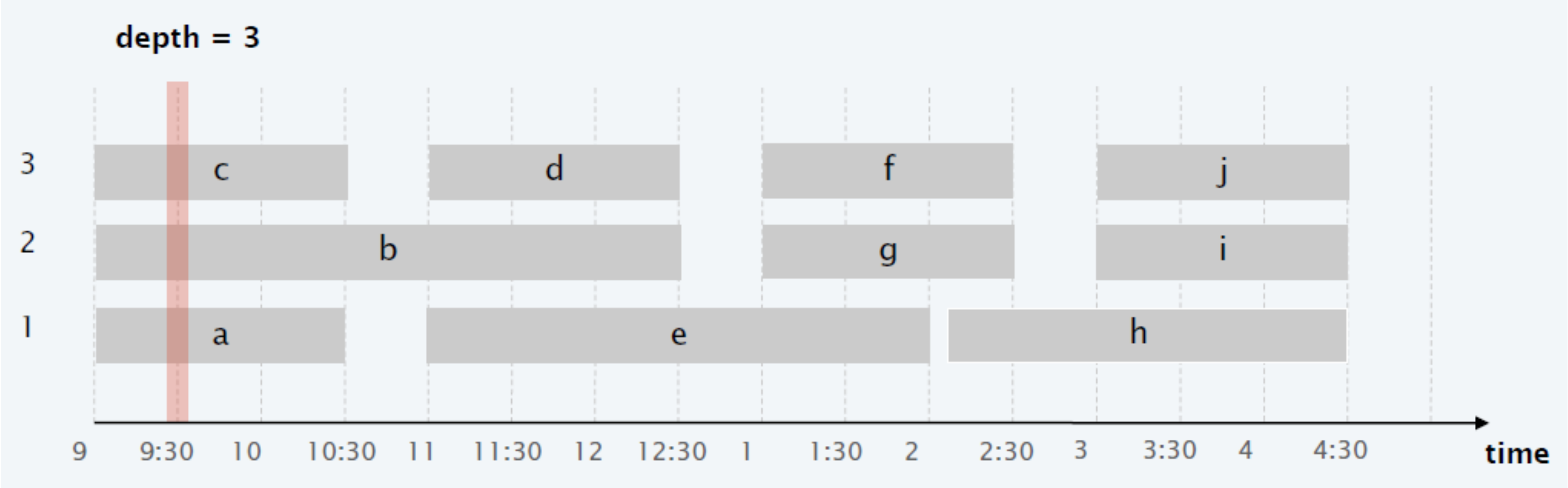
RETURN schedule.

Interval Partitioning

- Running time
  - Key step: check if the next lecture can be scheduled at some classroom
  - Store classrooms in a priority queue / min heap
    - key = latest finish time of any lecture in the classroom
  - Is lecture j compatible with some classroom?
    - If $s_j \geq$ smallest key (say of classroom k), add lecture j to classroom k & update its key to f_j
    - Otherwise, create a new classroom, add lecture j, set key to f_j
  - $O(n \log n)$ for sorting, $O(n \log d)$ for priority queue operations (if greedy ends up using d classrooms)
  - Since $d \leq n$, total time is $O(n \log n)$

Interval Partitioning

- Proof of optimality (lower bound)
  - Easy claim: # classrooms needed in any schedule $\geq$ "depth"
    - depth = maximum number of lectures running at any time
    - Recall, as before, that job i runs in $[s_i, f_i)$
  - Difficult claim: # classrooms needed by greedy $\leq$ depth



Interval Partitioning

- Proof of optimality (upper bound)
  - Let d = # classrooms used by greedy
  - Classroom d was opened because each classroom $k \in \{1, \dots, d-1\}$ had a lecture ℓ_k that was in conflict with lecture j
  - Consider the set of d lectures $\{\ell_1, \dots, \ell_{d-1}, j\}$
  - *Since we sorted by start time*, each lecture in this set starts at/before s_j and ends after s_j (since it conflicts with lecture j)
  - So, at time s_j, there are at least d mutually conflicting lectures
  - Hence, depth $\geq d =$ #classrooms used by greedy ■

Interval Graphs

NOT IN SYLLABUS

- Interval scheduling and interval partitioning can be seen as graph problems

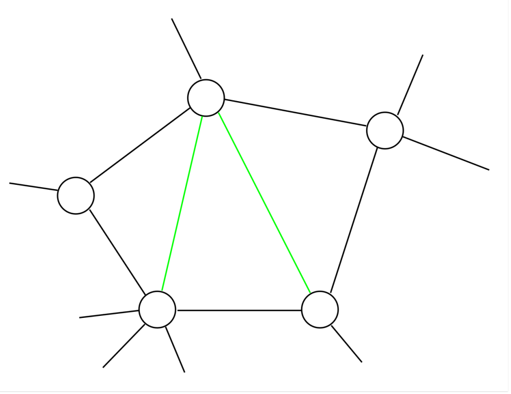
- Input
  - Graph $G = (V, E)$
  - Vertices V = jobs/lectures
  - Edge $(i, j) \in E$ if jobs i and j are incompatible

- Interval scheduling = maximum independent set (MIS)
- Interval partitioning = graph coloring

Interval Graphs

NOT IN SYLLABUS

- MIS and graph coloring are NP-hard for general graphs
- But they're efficiently solvable for "interval graphs"
  - ➢ Graphs which can be obtained from incompatibility of intervals
  - ➢ In fact, this holds even when we are not given an interval representation of the graph
- Can we extend this result further?
  - ➢ Yes! Chordal graphs
    - o Every cycle with 4 or more vertices has a chord



Minimizing Lateness

- Problem
  - We have a single machine
  - Each job j requires t_j units of time and is due by time d_j
  - If it's scheduled to start at s_j, it will finish at $f_j = s_j + t_j$
  - Lateness: $\ell_j = \max\{0, f_j - d_j\}$
  - Goal: minimize the maximum lateness, $L = \max_j \ell_j$

- Contrast with interval scheduling
  - We can decide the start time
  - There are soft deadlines

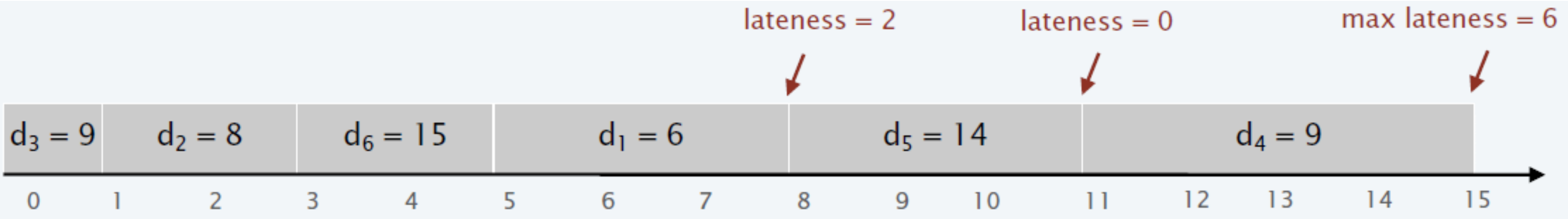
Minimizing Lateness

- Example

Input

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

An example schedule



Minimizing Lateness

- Let’s go back to greedy template
  - Consider jobs one-by-one in some “natural” order
  - Schedule jobs in this order (nothing special to do here, since we have to schedule all jobs and there is only one machine available)
- Natural orders?
  - Shortest processing time first: ascending order of processing time t_j
  - Earliest deadline first: ascending order of due time d_j
  - Smallest slack first: ascending order of $d_j - t_j$

Minimizing Lateness

- Counterexamples

  - Shortest processing time first
    - Ascending order of processing time t_j

	1	2
t_j	1	10
d_j	100	10

  - Smallest slack first
    - Ascending order of $d_j - t_j$

	1	2
t_j	1	10
d_j	2	10

Minimizing Lateness

- By now, you should know what's coming…
- We'll prove that earliest deadline first works!

EARLIESTDEADLINEFIRST($n, t_1, t_2, \ldots, t_n, d_1, d_2, \ldots, d_n$)

SORT n jobs so that $d_1 \leq d_2 \leq \ldots \leq d_n$.

$t \leftarrow 0$

FOR $j = 1$ TO n

  Assign job j to interval $[t, t + t_j]$.

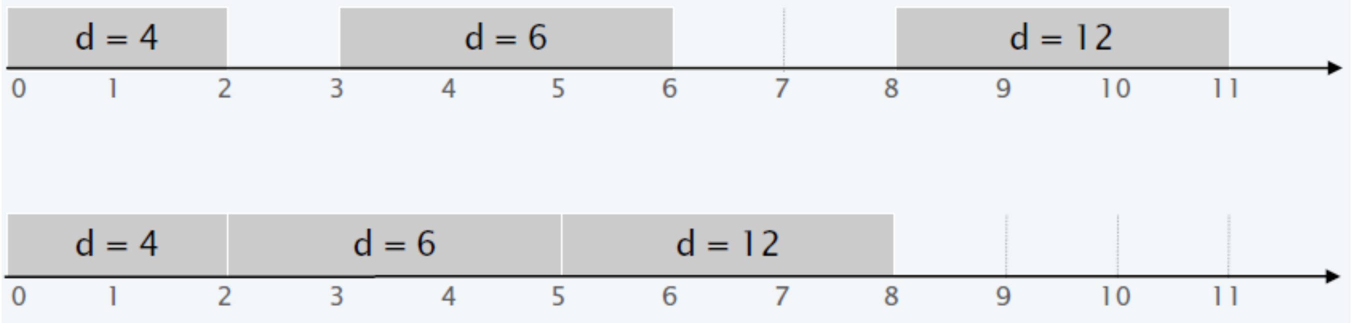
  $s_j \leftarrow t$; $f_j \leftarrow t + t_j$

  $t \leftarrow t + t_j$

RETURN intervals $[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]$.

Minimizing Lateness

- Observation 1
  - There is an optimal schedule with no idle time



- Observation 2
  - EDF has no idle time

- To prove:
  - EDF is at least as good as any schedule (even that optimal schedule) with no idle time

Minimizing Lateness

- Consider any schedule with no idle time
  - It can be represented as a permutation $(q_1, q_2, \dots, q_n)$ of $(1,2, \dots, n)$
- Define an inversion:
  - Any pair of jobs (i, j) such that $i < j$ but i is scheduled after j
- Observation 3
  - EDF has zero inversions
  - Every other schedule with no idle time has at least one inversion

Minimizing Lateness

- Observation 4
  - If a no-idle-time-schedule $(q_1, q_2, \dots, q_n)$ has at least one inversion, then it has at least one inversion in an adjacent pair of jobs (q_i, q_{i+1})

- Proof:
  - If not, then each of the pairs $(q_1, q_2), (q_2, q_3), \dots, (q_{n-1}, q_n)$ is not an inversion
  - Then, $q_1 < q_2,\ q_2 < q_3,\ \dots,\ q_{n-1} < q_n$
  - Then, $q_1 < q_2 < \cdots < q_n$
  - The only such schedule is $(1,2, \dots, n)$, which has zero inversions

Minimizing Lateness

- Observation 5
  - Swapping adjacently scheduled inverted jobs doesn't increase lateness but reduces #inversions by one
- Proof
  - Check that swapping an adjacent inverted pair reduces the total #inversions by one

Minimizing Lateness

- Observation 5
  - Swapping adjacently scheduled inverted jobs doesn't increase lateness but reduces #inversions by one
- Proof
  - Let ℓ_k and ℓ'_k denote the lateness of job k before & after swap
  - Let $L = \max_k \ell_k$ and $L' = \max_k \ell'_k$
  - 1) $\ell_k = \ell'_k$ for all $k \neq i, j$ (no change in their finish time)
  - 2) $\ell'_i \leq \ell_i$ (i is moved early)

Minimizing Lateness

- Observation 5
  - Swapping adjacently scheduled inverted jobs doesn't increase lateness but reduces #inversions by one
- Proof
  - 3) $\ell'_j = f'_j - d_j = f_i - d_j \leq f_i - d_i = \ell_i \quad (\because i < j \Rightarrow d_i \leq d_j)$
  - Hence, $L' = \max\left\{\ell'_i, \ell'_j, \max_{k \neq i,j} \ell'_k\right\} \leq \max\left\{\ell_i, \ell_i, \max_{k \neq i,j} \ell_k\right\} \leq L$
  - Alternatively:
    - $\ell'_k = \ell_k \leq L$ for all $k \neq i, j$
    - $\ell'_i \leq \ell_i \leq L$
    - $\ell'_j \leq \ell_i \leq L$
    - Hence, $L' = \max\left\{\ell'_i, \ell'_j, \max_{k \neq i,j} \ell'_k\right\} \leq L$

Minimizing Lateness

Observations 1 & 2:
Greedy EDF and some optimal schedule OPT have no idle time (thus, they're permutations of jobs)

Observation 3:
EDF permutation has 0 inversions, every other permutation has at least 1 inversion.

Observations 4 & 5:
If OPT has $r \geq 1$ inversions, there is another optimal permutation that has $r - 1$ inversions.

Proof by contradiction/induction that there is an optimal permutation with 0 inversions

- ➢ Must be the EDF permutation

Minimizing Lateness

- Proof of optimality by contradiction
  - Suppose for contradiction that the greedy EDF permutation is not optimal
  - Among all optimal permutations with no idle time (these exist by Observation 1), consider OPT* which has the fewest inversions
  - Because EDF permutation is the only one with zero inversions (Observation 3) and it is not optimal, OPT* has $r \geq 1$ inversions
  - By Observation 4, it has an adjacent inversion (i, j)
  - By Observation 5, swapping the adjacent pair produces a new permutation (no idle time) that is optimal and has $r - 1$ inversions
  - Contradiction! ■

Minimizing Lateness

- Proof of optimality by (reverse) induction
  - Claim: For each $r \in \left\{0,1,\dots,\binom{n}{2}\right\}$, there is an optimal permutation (no idle time) with *at most* r inversions
  - Base case of $r = \binom{n}{2}$: Use any optimal permutation (Observation 1)
  - Induction hypothesis: Suppose the claim holds for $r = t + 1$
  - Induction step:
    - Let OPT* be an optimal permutation with at most $t + 1$ inversions
    - If it has at most t inversions, we're done!
    - If it has exactly $t + 1 \geq 1$ inversions, find and swap an adjacent inverted pair to get a new optimal permutation with t inversions (Observations 4 & 5)
  - QED!
  - Claim for $r = 0$ shows optimality of the EDF permutation (Observation 3)

Lossless Compression

- Problem
  - We have a document that is written using n distinct labels
  - Naïve encoding: represent each label using $\log n$ bits
  - If the document has length m, this uses $m \log n$ bits

- English document with no punctuations etc.
- $n = 26$, so we can use 5 bits
  - $a = 00000$
  - $b = 00001$
  - $c = 00010$
  - $d = 00011$
  - …

Lossless Compression

- Is this optimal?
  - What if a, e, r, s are much more frequent in the document than x, q, z?
  - Can we assign shorter codes to more frequent letters?

- Say we assign…
  - $a = 0, b = 1, c = 01, \ldots$
  - See a problem?
    - What if we observe the encoding '01'?
    - Is it 'ab'? Or is it 'c'?

Lossless Compression

- To avoid conflicts, we need a *prefix-free encoding*
  - Map each label x to a bit-string $c(x)$ such that for all distinct labels x and y, $c(x)$ is not a prefix of $c(y)$

  - Then it's impossible to have a scenario like this

$$\underbrace{\underbrace{\ldots\ldots\ldots\ldots\ldots\ldots}_{c(x)}\ldots\ldots\ldots\ldots\ldots}_{c(y)}$$

  - Now, we can read left to right
    - Whenever the part to the left becomes a valid encoding, greedily decode it, and continue with the rest

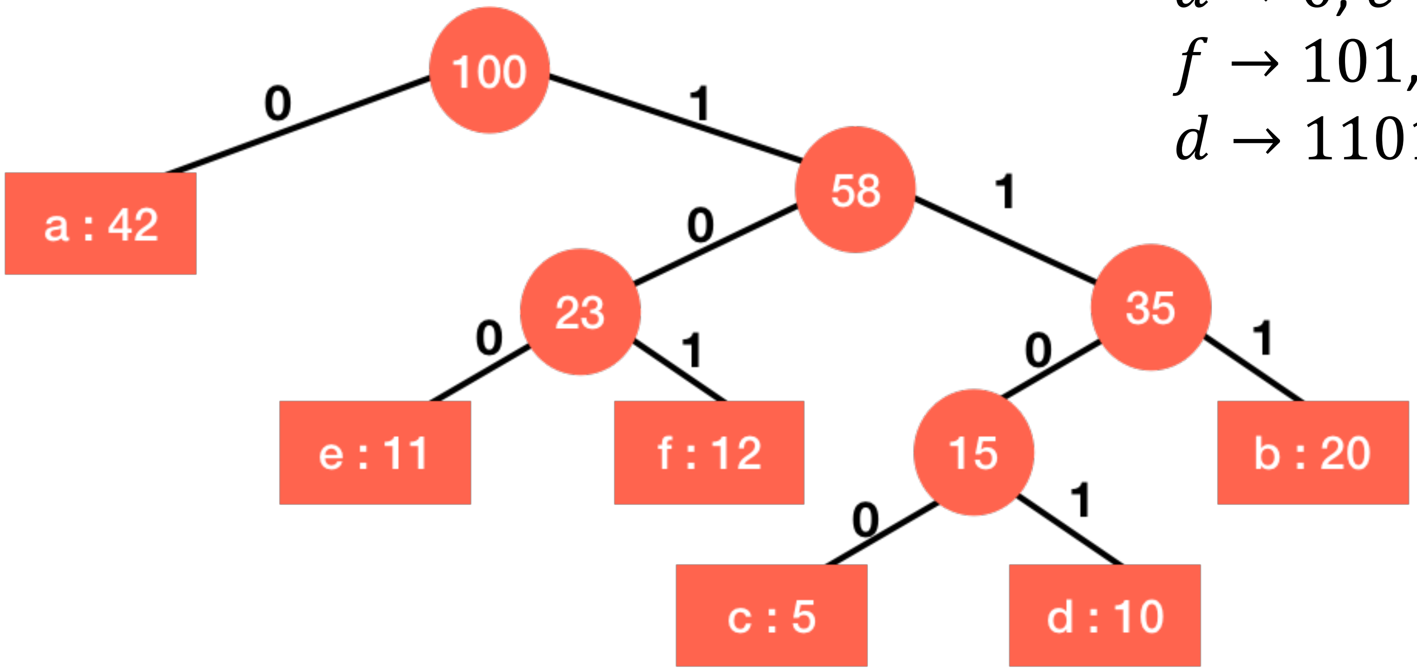
Lossless Compression

- Formal problem
  - Given n symbols and their frequencies $(w_1, \ldots, w_n)$, find a prefix-free encoding with lengths $(\ell_1, \ldots, \ell_n)$ assigned to the symbols which minimizes $\sum_{i=1}^{n} w_i \cdot \ell_i$
    - Note that $\sum_{i=1}^{n} w_i \cdot \ell_i$ is the length of the compressed document

- Example
  - $(w_a, w_b, w_c, w_d, w_e, w_f) = (42,20,5,10,11,12)$
  - No need to remember the numbers ☺

Lossless Compression

- Observation: prefix-free encoding = tree



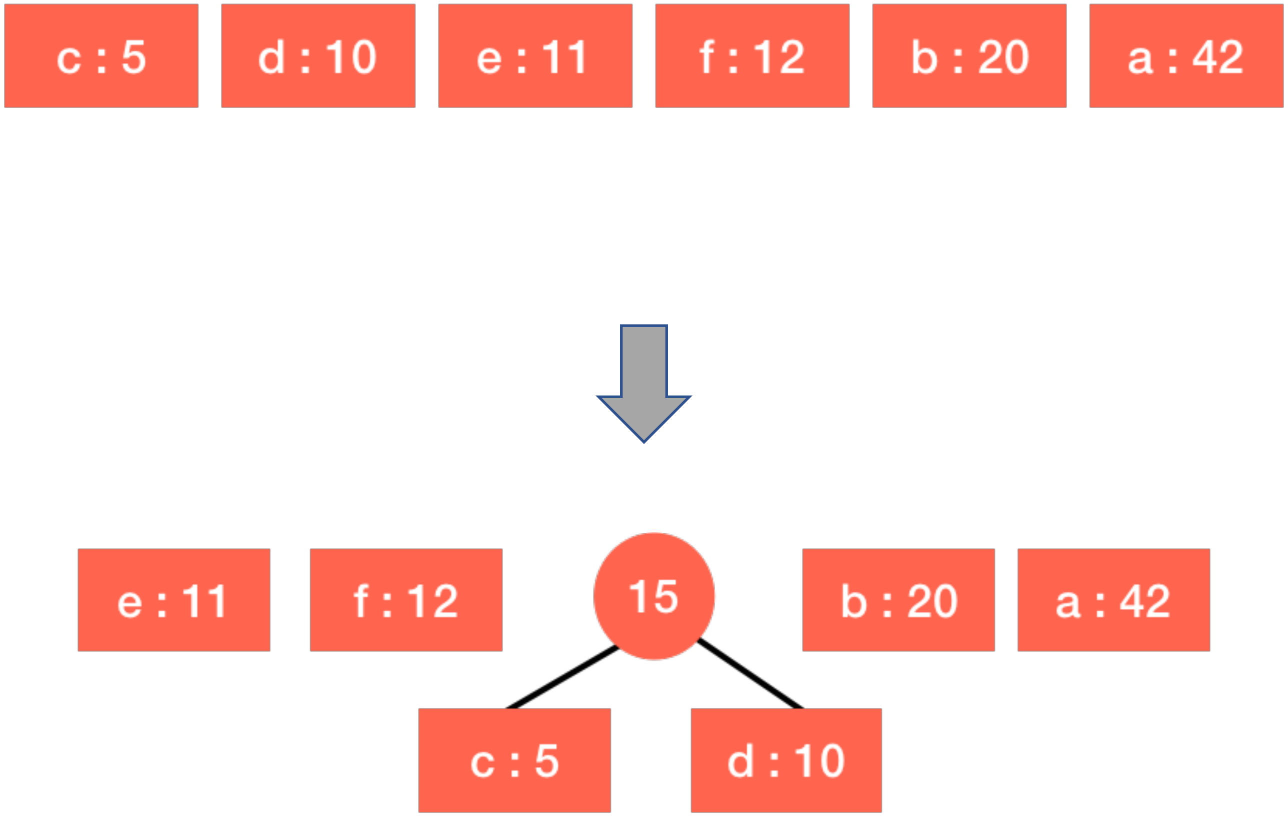
$a \to 0, e \to 100,$
$f \to 101, c \to 1100,$
$d \to 1101, b \to 111$

Lossless Compression

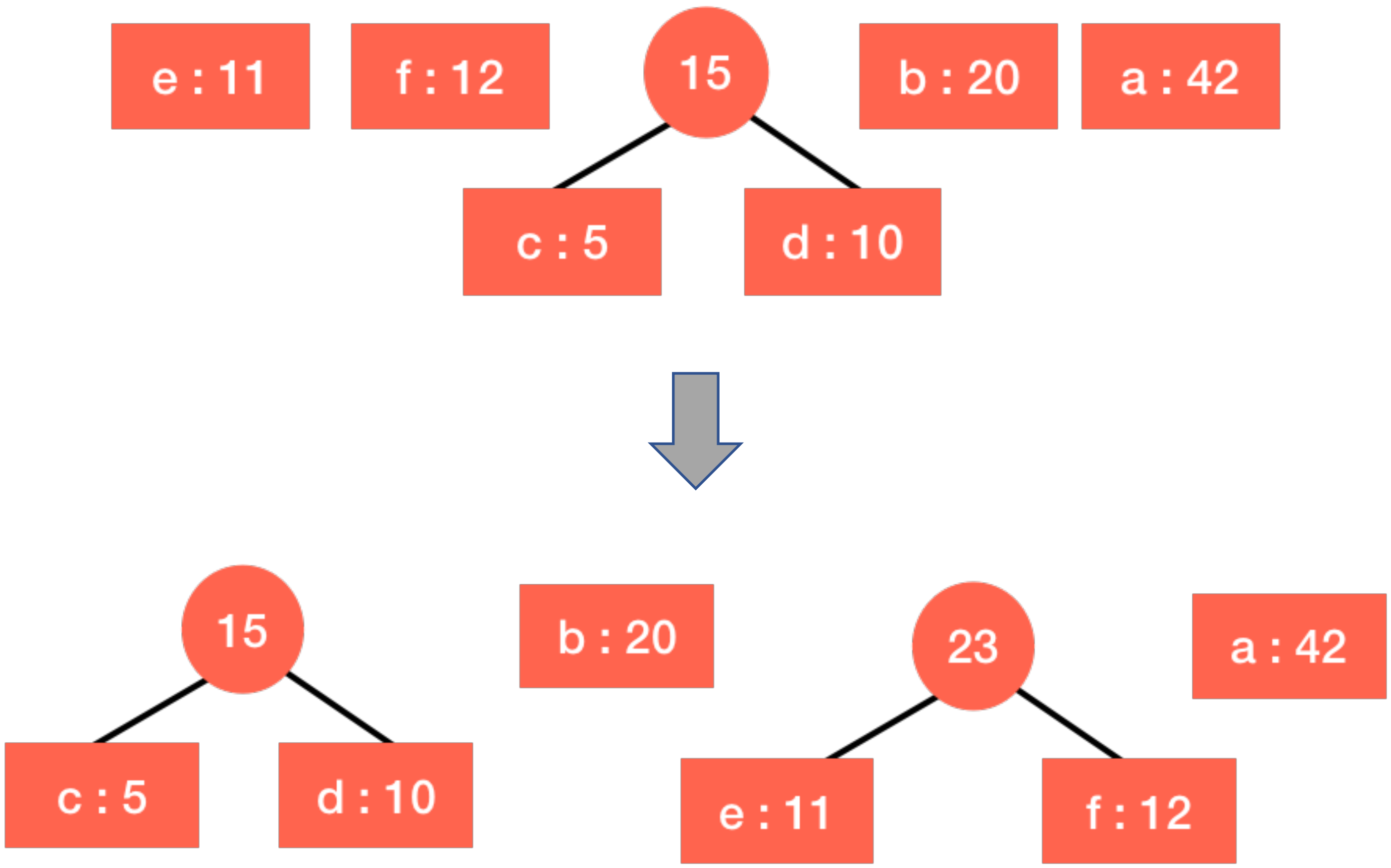
- Huffman Coding
  - Build a priority queue by adding (x, w_x) for each symbol x
  - While $|\text{queue}| \geq 2$
    - Take the two symbols with the lowest weight (x, w_x) and (y, w_y)
    - Merge them into one symbol with weight $w_x + w_y$

- Let's see this on the previous example

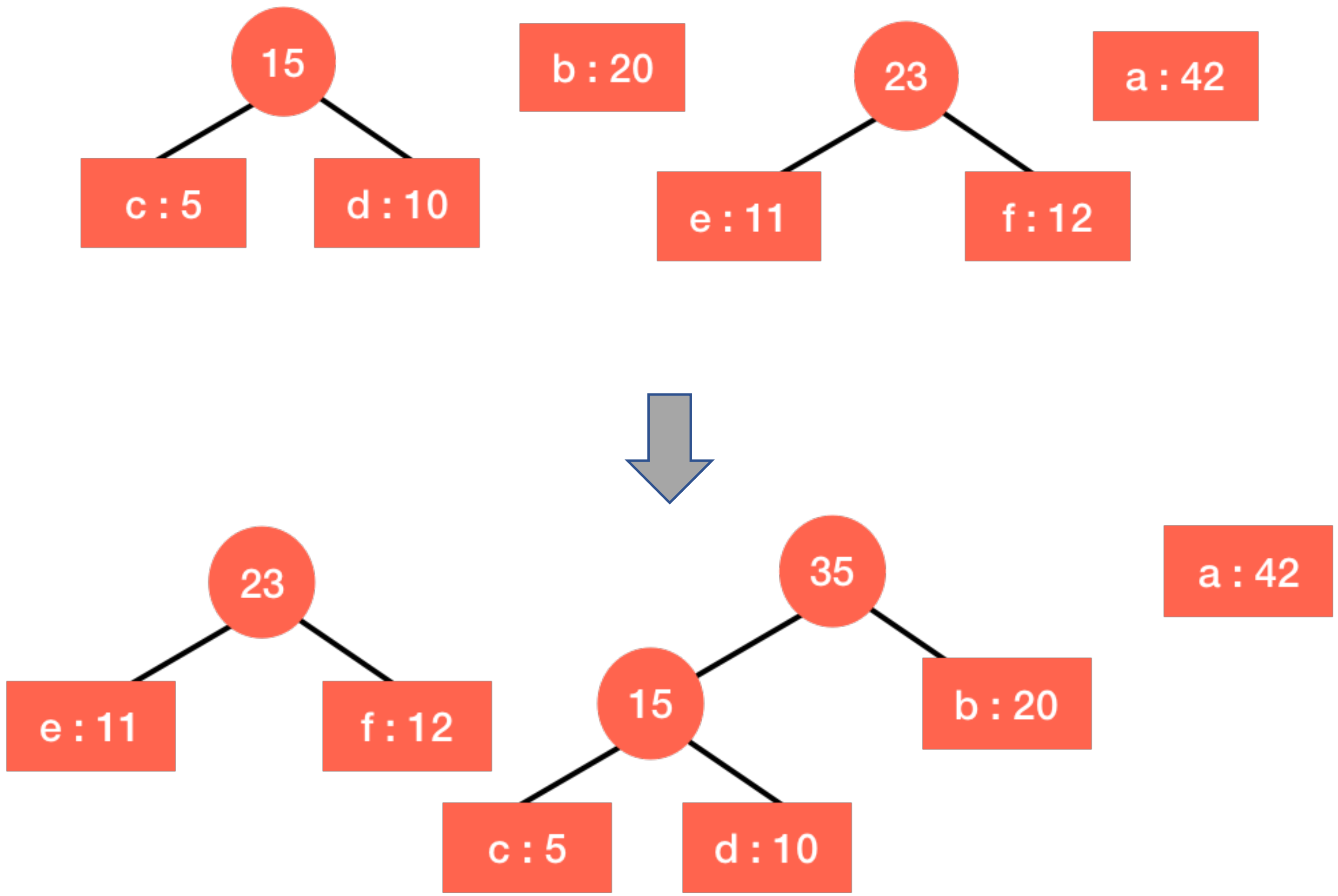
Lossless Compression



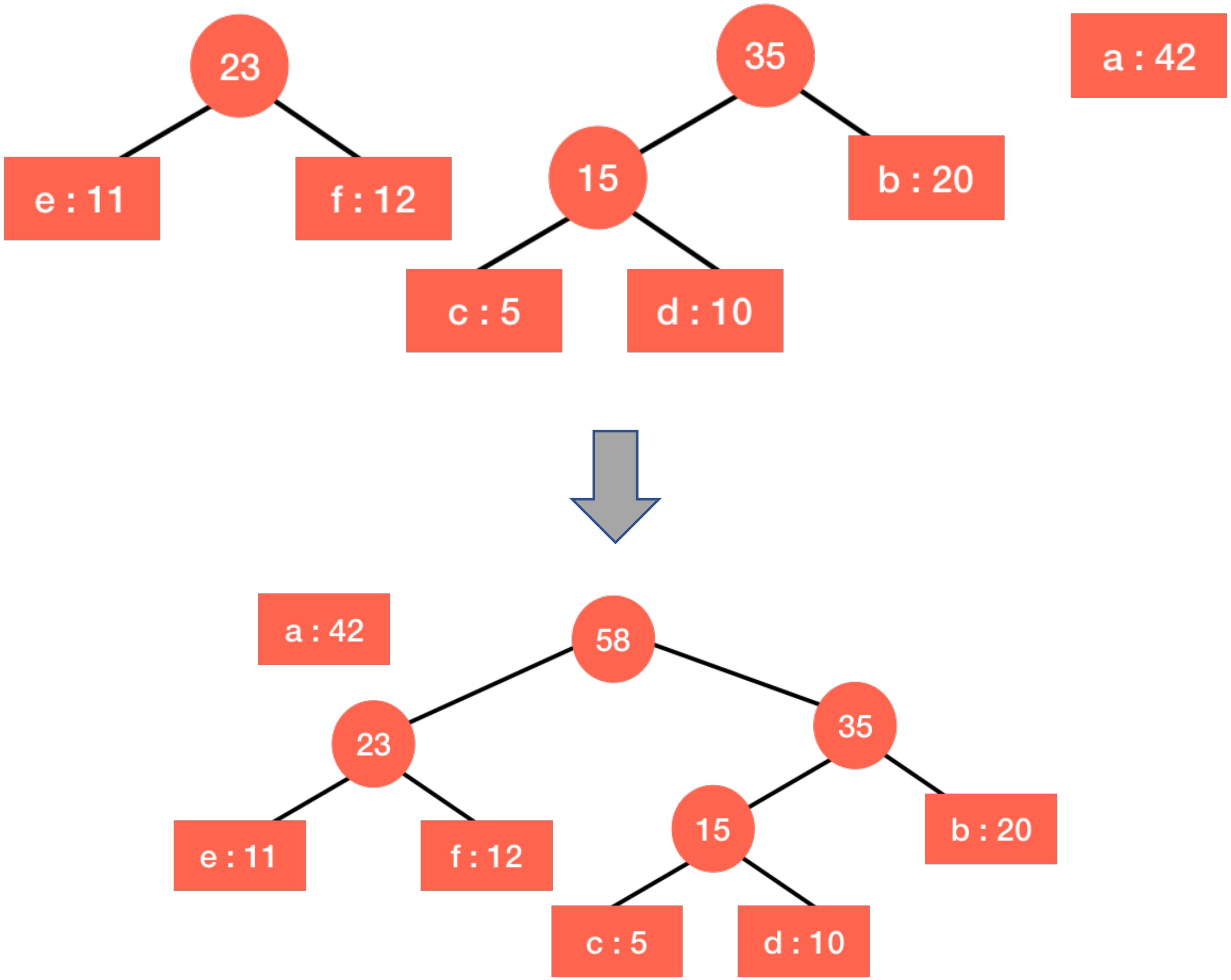
Lossless Compression



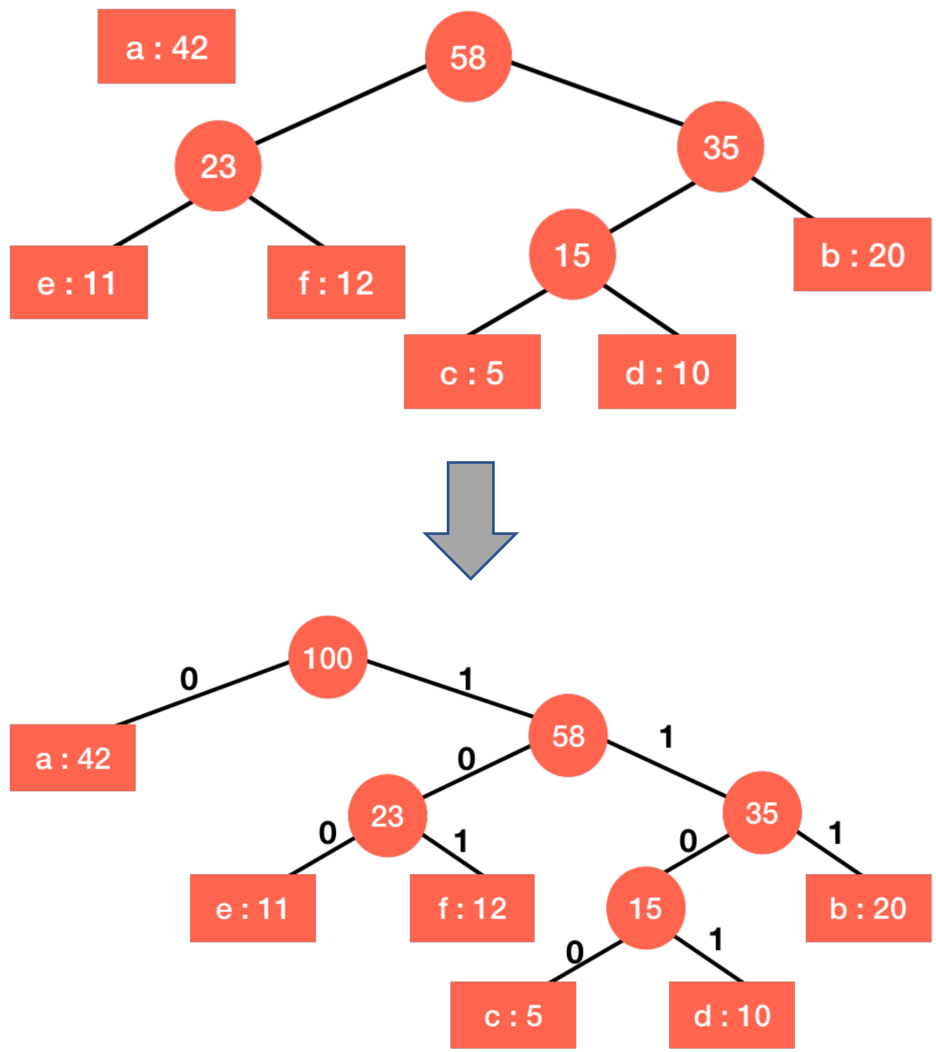
Lossless Compression



Lossless Compression



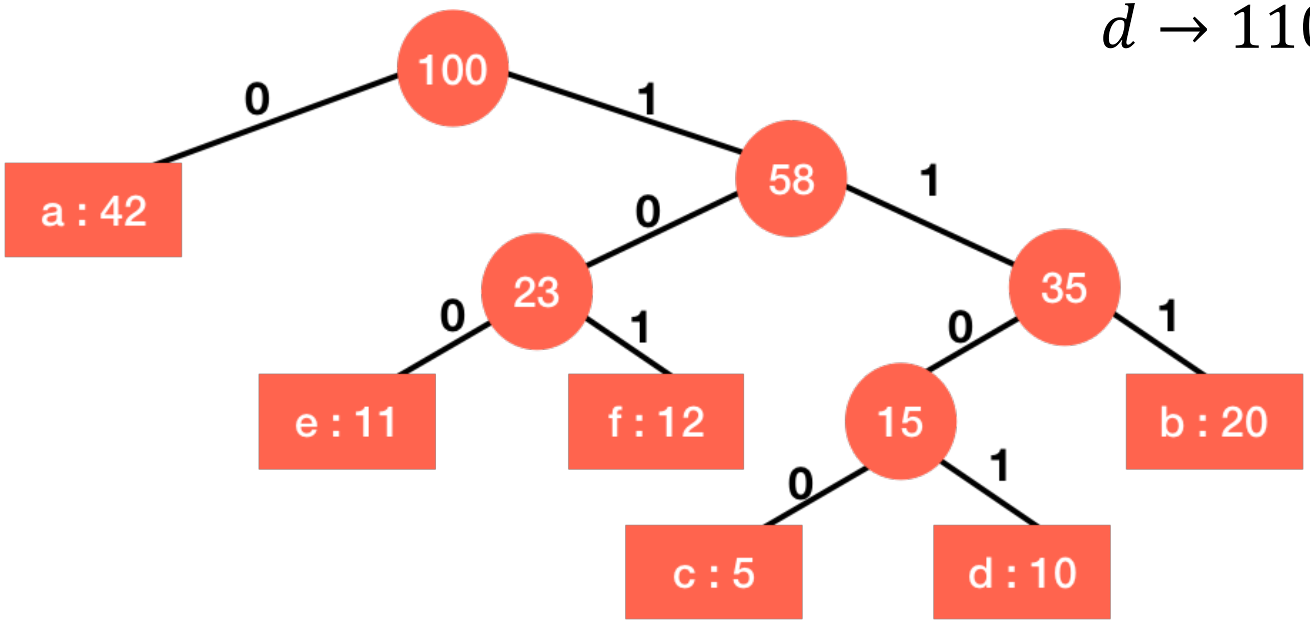
Lossless Compression



Lossless Compression

- Final Outcome

$a \to 0, e \to 100,$
$f \to 101, c \to 1100,$
$d \to 1101, b \to 111$



Lossless Compression

- Running time
  - $O(n \log n)$
  - Can be made $O(n)$ if the labels are given to you sorted by their frequencies
    - Exercise! Think of using two queues…

- Proof of optimality
  - Induction on the number of symbols n
  - Base case: For $n = 2$, both encodings which assign 1 bit to each symbol are optimal
  - Hypothesis: Assume it returns an optimal encoding with $n - 1$ symbols and consider the case of n symbols.

Lossless Compression

- Lemma 1: If $w_a \leq w_b$ but $\ell_a \geq \ell_b$, then swapping the encodings of a and b does not make the objective any worse.

- Proof:
  - We simply need to check that the given inequalities imply
    $$w_a \cdot \ell_b + w_b \cdot \ell_a \leq w_a \cdot \ell_a + w_b \cdot \ell_b$$
  - QED!

Lossless Compression

- Let x, y be the first two symbols in Huffman priority queue
  - w_x is the lowest, w_y is the second lowest
  - They become siblings in the Huffman tree from the first iteration

- Lemma 2: $\exists$ optimal tree T in which x and y are siblings.
- Proof:
  1. Take any optimal tree
  2. If ℓ_x isn't the longest encoding, swapping x with a symbol that has the longest encoding keeps the tree optimal (Lemma 1)
  3. In this optimal tree, x must have a sibling (check!)
  4. If it's not y, swapping it with y keeps the tree optimal (Lemma 1)
  5. Now we have an optimal tree where x and y are siblings. ■

Lossless Compression

- Proof of optimality
  - Let H be the Huffman tree
  - Let T be an optimal tree in which x and y are siblings (Lemma 2)
  - Let H' and T' be obtained from H and T by treating 'xy' as one symbol with frequency $w_x + w_y$
  - Note that
    - $Length(H) = Length(H') + \left(w_x + w_y\right) \cdot 1$
    - $Length(T) = Length(T') + \left(w_x + w_y\right) \cdot 1$
  - But due to the induction hypothesis, $Length(H') \leq Length(T')$
  - Hence, $Length(H) \leq Length(T)$ ■

Other Greedy Algorithms

- If you aren't familiar with the following algorithms, spend some time checking them out!
  - Dijkstra's shortest path algorithm
  - Kruskal and Prim's minimum spanning tree algorithms