THINK FAST

Competitive Programming

From Problem 2 Solution in O(1)

Greedy Algorithms

Exchange Argument Technique

Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University

Recall: Proving Greedy

- We may prove greedy properties directly
  - Done
- 2 popular techniques can be used for proving
  - Stay Ahead [Last Session]
  - **Exchange Argument** [Today Session]
- And some other ways (e.g. Matroids)

Recall: General guidelines

- Have to do it Greedy?
  - Brainstorm on all possible heuristics...put in ur ideas' pool
  - Don't think/verify….just brainstorm
- Pick most promising idea
  - Find some example that breaks it...think hard
  - Wrong? think what to fix generally...push idea to the pool
- Good heuristic so far? Can you prove? or informal strong intuition? Yes => do it

Proof technique: Exchange Argument

- Goal: **Convert** your greedy solution to an optimal solution with same value!
- Assume we have Optimal Solution **O**
- And Greedy algorithm solution is **A**
- Do series of exchanges to **O** to convert it to **A** such that both have same min/max answer
- You will need to analyze your greedy properties/nature to determine what to exchange in **O**

Proof technique: Exchange Argument

- 1) Describe the 2 solutions
  - E.g. O = {o1, o2….ok} and A = {a1, a2, …. ak}
- 2) Compare them and see how to convert
  - E.g. Element in O not in A and the reverse
  - E.g. Same elements but in different order. Assume A is sorted. Then e.g. swap 2 consecutive elements in O that are not sorted
- 3) Do Exchange: E.g. Swap **in/out** elements or swap **consecutives** as noted before. Show that exchange step doesn't affect overall answer

Proof technique: Exchange Argument

- You need to do the **exchange** several times to fully convert. Argue that this is possible
- You need to argue about the **existence** of what you are exchanging (e.g. do this case really exists: e.g. in/out or unordered elements)
- Some Greedy solutions uses this technique such as <u>lateness problem, caching and MST</u>

Exchange Argument Vs Greedy properties

- Exchange argument depends on the **output** of your greedy
- Greedy properties proving style depends on your **first** step and nature of **sub-problem**
- Exchange argument is nice to consider when different outputs of optimal solutions have **same length** (e.g. every MST have same cost, but every shortest path may have different edges)

Exchange Argument: MST

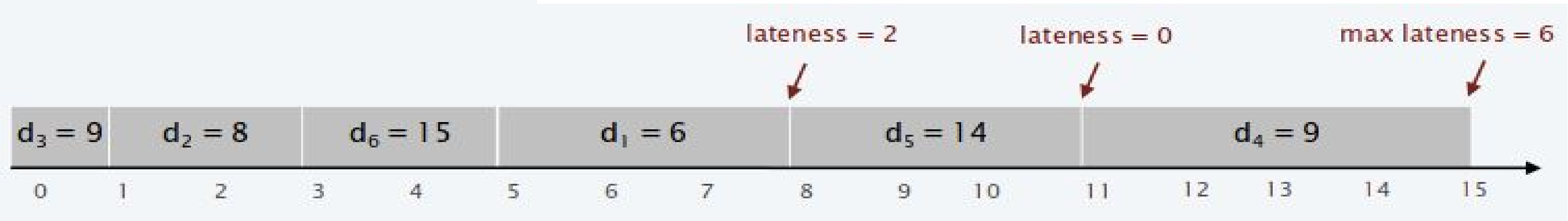
- This is not prove...just big picture outline
- Assume Kruskal gives MST = TK
- And one optimal solution is MST = TO
- Assume edge e1 in TO but not in TK
- We can show that e1 of TO can be **swapped** with some e2 in TK without increasing cost
- Then # of common edges increase
- Keep applying the method to fully convert TO to TK with same cost

Scheduling to minimize lateness

- Jobs: each job = <Processing time, Deadline>
  - J1 (5, 20). If started it at 9, it ends at 14 < 20 (no penalty)
  - J1 (5, 20). If started it at 18, it ends at 23. Penalty 3
  - Penalty (lateness): Extra time used after deadline
- Processing is in non-overlapping manner
- Goal: Schedule them with **min lateness**
  - Find start time of each task / no overlap in processing time
  - Such that **minimize maximum** lateness

Scheduling to minimize lateness

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



- Brainstorm
  - **Sort** incrementally based on **processing time** (logic: removing smaller jobs gives space for next jobs)
  - **Sort** incrementally based on **deadline** (logic: earlier deadline should be done first)

Scheduling: Investigate heuristics

- Processing time heuristics
  - Think for breaking examples
  - Algorithm picks first (1, 100), finish at 100
  - 2nd start at 100 and finish 110. **Penalty 100**
  - The reverse caused 0 lateness
  - Why wrong? Ignored deadline Can we fix?
  - New idea: Sort based on difference |Deadline - Processing|

	1	2
t_j	1	10
d_j	100	10

- Difference heuristics
  - Fails too
  - Why? Lost Deadline positions

    as constraints.

	1	2
t_j	1	10
d_j	2	10

Scheduling: Investigate heuristics

- Earliest Deadline heuristics (right one)
  - But, it ignores processing time...you should be afraid :)

EARLIEST-DEADLINE-FIRST $(n, t_1, t_2, \ldots, t_n, d_1, d_2, \ldots, d_n)$

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

$t \leftarrow 0$

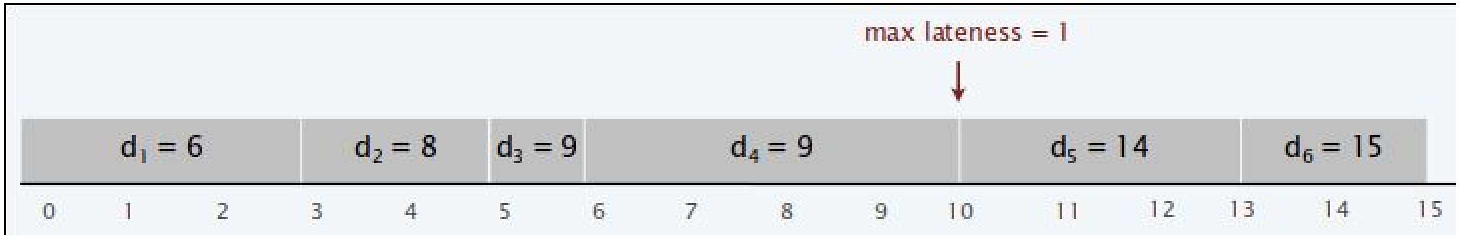
FOR $j = 1$ TO n

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

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

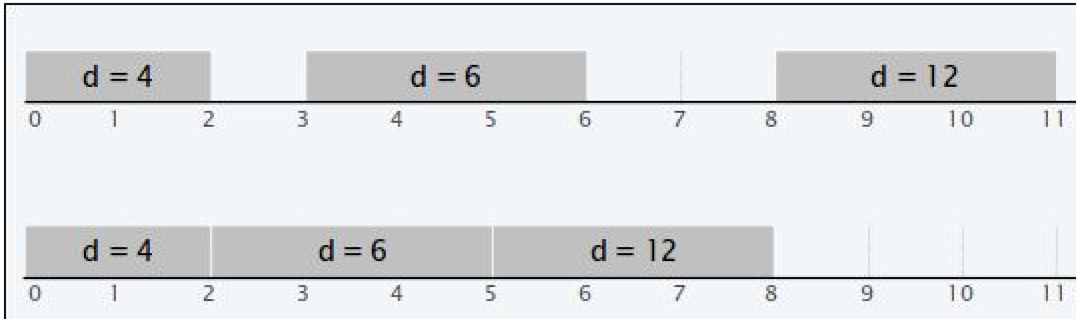
$t \leftarrow t + t_j$

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



Scheduling to minimize lateness

- Observe that
  - Our optimal solution just stacks the jobs, e.g. **no idle time**
  - Any optimal solution with idle time can be compressed



  - Our algorithm has k items sorted by deadline
  - Optimal answer will have **SAME** k items, **NOT** ordered
  - So one can think in doing **series of swaps** to prove

Scheduling to minimize lateness

- If we can do series of swaps that don't affect overall answer, then greedy is optimal
  - Swap = Exchange
- Sorting on unsorted arrays <=> **inversions**
- Array Inversion: if $j < i$ and $A[j] > A[i]$
  - A = [2, 4, 1, 3, 5] has 3 inversions (2, 1), (4, 1), (4, 3).
  - If swapping consecutive elements only, we can do maximum nC2 swaps (e.g. for A = [5, 4, 3, 2, 1])

Schedules & Inversions

- Recall we have d1 <= d2 <= … dn
- Schedule has an inversion if job j before i AND Dj > Di
  - assume: d1 = 5, d2 = **10**, d3 = **10**, d4 = 20
  - S1 = {d1, d2, d3, d4} => No inversions .. our greedy
  - S2 = {d1, **d3, d2**, d4} => No inversions, duplicates sorted
  - S3 = {**d4, d1**, d2, d3} => Has Inversion
- Q: given set of deadlines, how many schedules with no inversions?

Schedules & Inversions

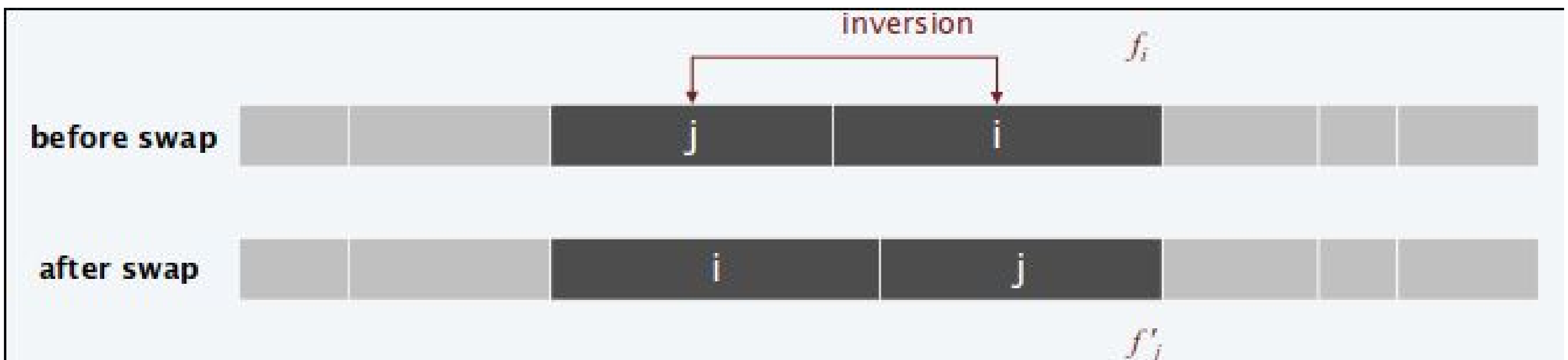
- Assume our greedy solution is A and one of Optimal schedules is O (no idle time)
  - Either A = O, we are ok
  - A != O, but both has no inversions
  - A != O, but O has inversions
- We need to prove the last 2 cases: E.g. O can be converted to A with same final lateness

Schedules with no inversion

- Lemma: All schedules with **no inversions** and no idle time have the **same** maximum **lateness**
  - Such schedules have different jobs order
  - But they differ only in the order of identical deadlines
  - E.g. {d1, **d2, d3**, d4} = {d1, **d3, d2**, d4} = {5, **10, 10**, 20}
  - Assume k tasks with deadline d in some permutation
  - The last one will have **maximum** lateness
  - its lateness will be based on accumulation of t jobs, **regardless** of their order

Schedules with inversions

- Can we prove that swapping **consecutives** inverted jobs doesn't increase max lateness?
- Assume: job j before i AND Dj > Di [inversio]



- Swapping doesn't affect next jobs deadlines

Schedules with inversions

- Observe: i now is shifted earlier.
  - Then it can't have bigger lateness
- If j has no lateness, we are ok...otherwise
- Let lateness Li = F - Di and Lj = F - Dj
  - Recall: Di <= Dj, then
  - Lj = F - **Dj** <= F - **Di** <= Li
  - As Lj <= Li, swap won't increase lateness
- Q: Can we find example of 2 jobs where
  - D1 != D2, Optimal lateness > 0, Schedule {1, 2} as optimal as {2, 1}

Summary

- Our last intuition (first deadline) was good, but **suspicious …** prove was critical
- **Exchange argument** in lateness problem
  - Consecutive swaps to convert unsorted array to sorted one
  - We put limit on # of swaps (nC2)
  - Proved a consecutive swap doesn't worse solution
- Visualization helps
- Some Exchange argument problems are challenging and need much analysis for its differnt cases (e.g. Optimal Offline Caching)

Your turn

- In a marathon, each of n players will swim Si minutes, then run Ri minutes.
- **One** player can be at swimming pool at a time
- Find permutations of the players to minimize the marathon finish time
- E.g. p1 = <10, 5> and p2 = <12, 7>.
  - Use order P2 P1
  - P2: swim for 12 m, then run for 7 m. Finish 19 m
  - P1: start at **12,** swim for 10 min, run for 5 m. Finish **27 m**
- Find algorithm. Prove it.

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً