Lecture09

CSE 417
Algorithms and Complexity

Autumn 2024
Lecture 9 – Greedy Algorithms II

1

Announcements

- Today's lecture
  - Kleinberg-Tardos, 4.2, 4.3
- Wednesday and Friday
  - Kleinberg-Tardos, 4.4, 4.5

2

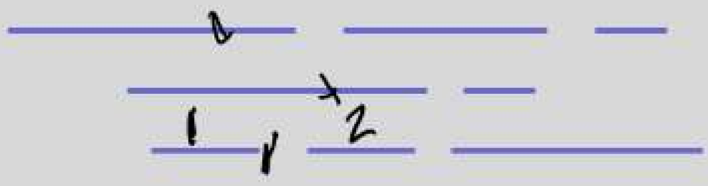
Greedy Algorithms

- Solve problems with the simplest possible algorithm
- The hard part: showing that something simple actually works
- Today's problems (Sections 4.2, 4.3)
  - Graph Coloring
  - Homework Scheduling
  - Optimal Caching

3

Interval Scheduling

- Tasks occur at fixed times, single processor
- Maximize number of tasks completed

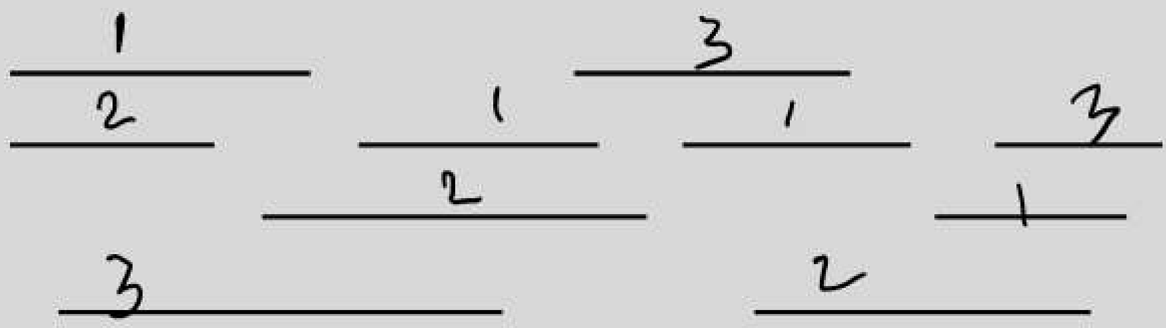


- Earliest finish time first algorithm optimal
- Optimality proof: stay ahead lemma
  - Mathematical induction is the technical tool

4

Scheduling all intervals with multiple processors

- Minimize number of processors to schedule all intervals



Depth: Maximum number of overlapping intervals

5

Runtime ?

$O(n \log n)$

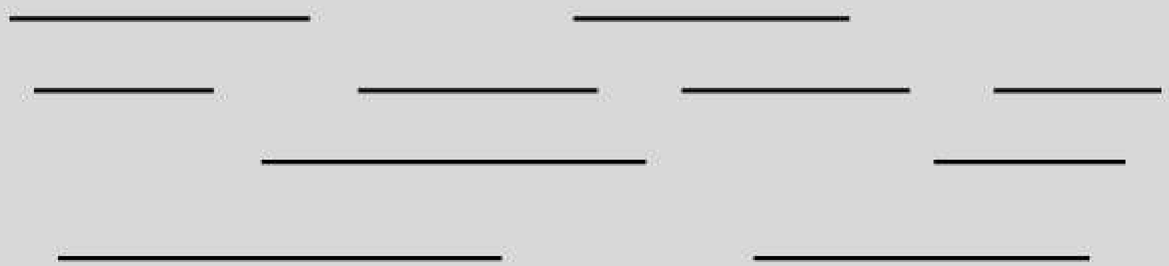
Algorithm

~ $O(n^2)$

$O(n \log n)$ algo: with priority Q.

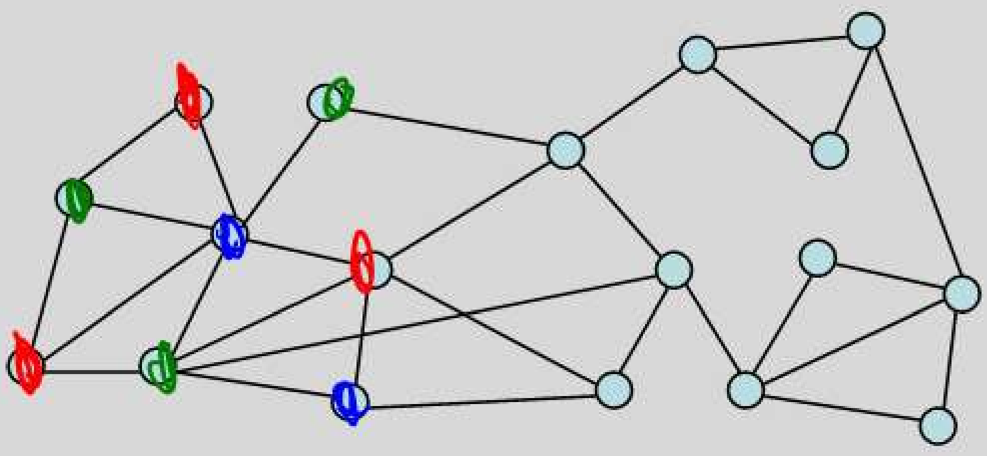
Sort intervals by start time

for i = 1 to n
    Assign interval i to the lowest numbered idle processor



Greedy Graph Coloring

Theorem: An undirected graph with maximum degree K can be colored with K+1 colors



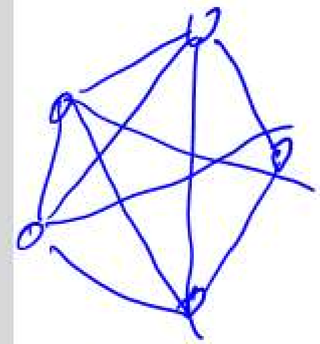
7

Greedy Coloring Algorithm

- Assume maximum degree K
- Pick a vertex v, and assign a color not in N(v) from [1, . . . , K + 1]
- Always an available color

- In the worst case, this algorithm cannot be improved
  - There exists a graph of degree K requiring K+1 colors

8



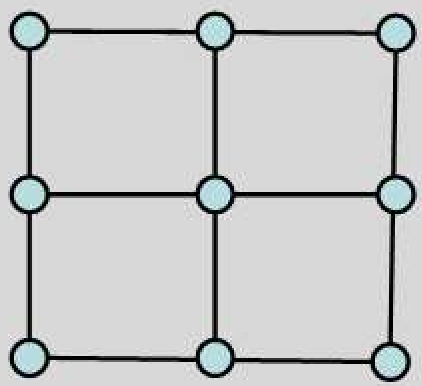
K_5

Coloring Algorithm, Version 1

```
Let k be the largest vertex degree
Choose k+1 colors

for each vertex v
       Color[v] = uncolored

for each vertex v
       Let c be a color not used in N[v]
       Color[v] = c
```

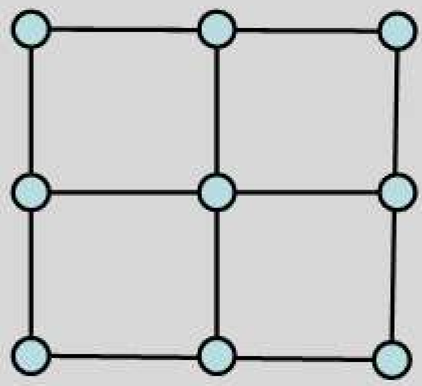


9

Coloring Algorithm, Version 2

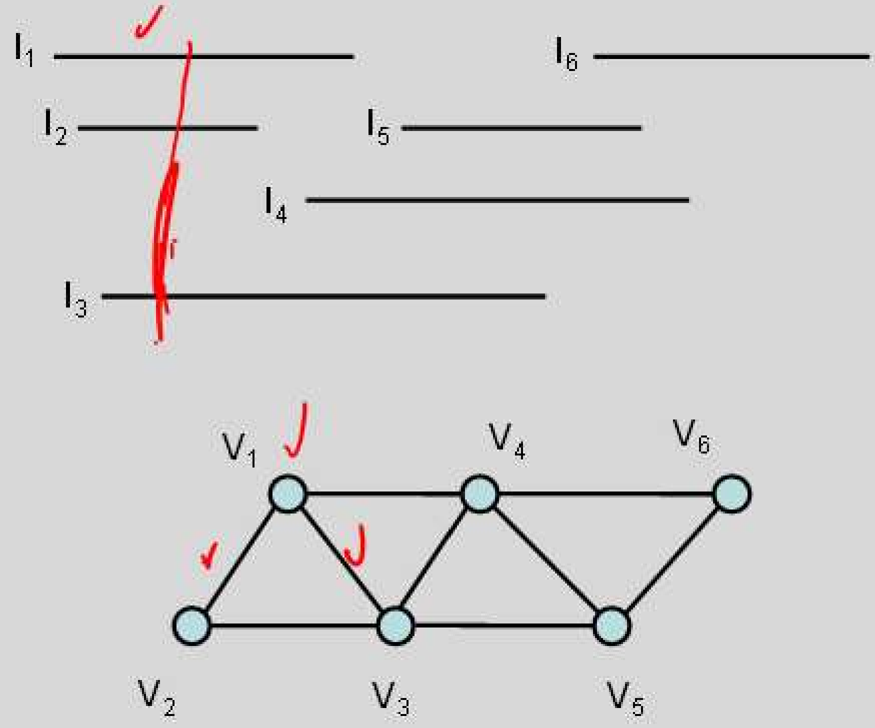
```
for each vertex v
       Color[v] = uncolored

for each vertex v
       Let c be the smallest color not used in N[v]
       Color[v] = c
```



10

Interval scheduling is graph coloring



11

Homework Scheduling

- Tasks to perform
- Deadlines on the tasks
- Freedom to schedule tasks in any order

- Can I get all my work turned in on time?
- If I can't get everything in, I want to minimize the maximum lateness

12

Scheduling tasks

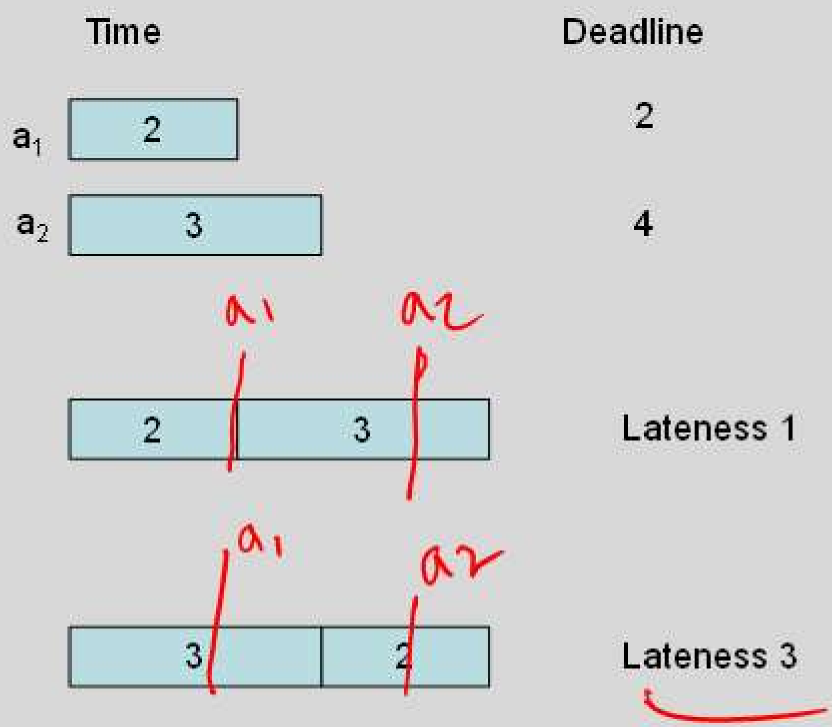
- Each task has a length t_i and a deadline d_i
- All tasks are available at the start
- One task may be worked on at a time
- All tasks must be completed

- Goal minimize maximum lateness
  - Lateness: $L_i = f_i - d_i$ if $f_i \geq d_i$

    $0 \quad f_i < d_i$

13

Example


Time
Deadline
a1
2
2
a2
3
4
a1
a2
2
3
Lateness 1
a1
a2
3
2
Lateness 3

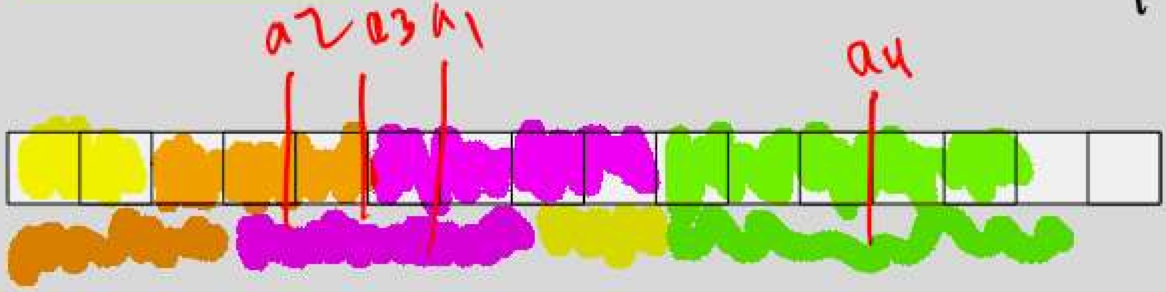
14

$a_2 a_1 a_3 a_4$, $a_2 a_3 a_1 a_4$

Determine the minimum lateness

max L_i

	Time	Deadline	Finish	Lates	
			2	0	3
a_1	2	6	5	1	0
a_2	3	4	9	4	2
a_3	4	5	14	2	2
a_4	5	12			



15

Greedy Algorithm

- Earliest deadline first
- Order jobs by deadline

- This algorithm is optimal

16

Analysis

- Suppose the jobs are ordered by deadlines, $d_1 \leq d_2 \leq . . . \leq d_n$
- A schedule has an *inversion* if job j is scheduled before i where j > i

- The schedule A computed by the greedy algorithm has no inversions.
- Let O be the optimal schedule, we want to show that A has the same maximum lateness as O

17

List the inversions

	Time	Deadline
a_1	3	4
a_2	4	5
a_3	2	6
a_4	5	12

(a_4, a_2)
(a_4, a_1)
(a_4, a_3)
(a_2, a_1)

18

Lemma: There is an optimal schedule with no idle time

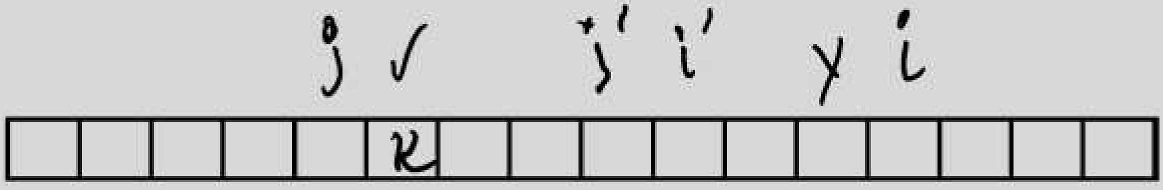
- It doesn't hurt to start your homework early!
- Note on proof techniques
  - This type of can be important for keeping proofs clean
  - It allows us to make a simplifying assumption for the remainder of the proof

19

Lemma

j > i

- If there is an inversion i, j, there is a pair of adjacent jobs i', j' which form an inversion



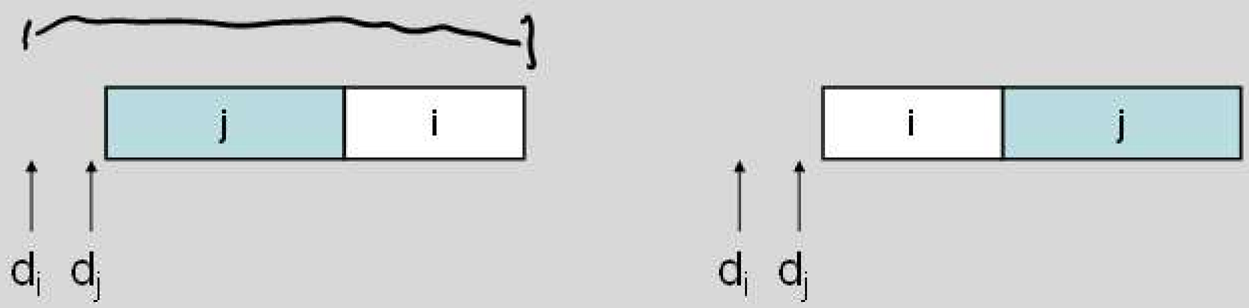
if k < j, adj inversion

k > j

20

Interchange argument

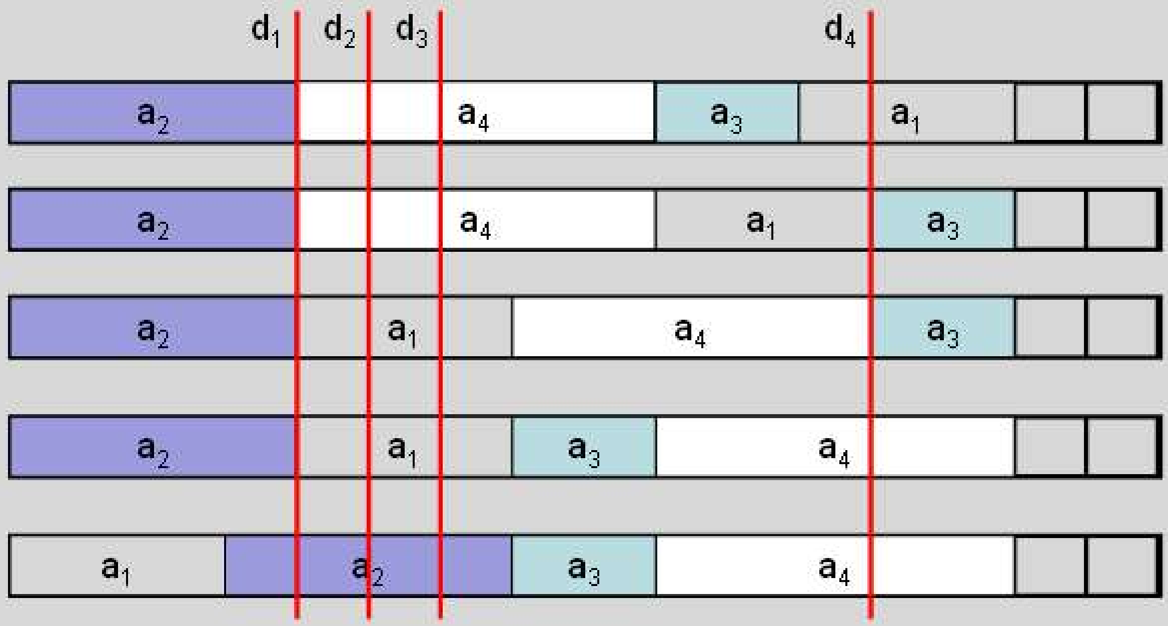
- Suppose there is a pair of jobs i and j, with $d_i \leq d_j$, and j scheduled immediately before i. Interchanging i and j does not increase the maximum lateness.



21

$O(n^2)$

Proof by Bubble Sort



Determine maximum lateness

22

Real Proof

- There is an optimal schedule with no inversions and no idle time.
- Let O be an optimal schedule k inversions, we construct a new optimal schedule with k-1 inversions
- Repeat until we have an optimal schedule with 0 inversions
- This is the solution found by the earliest deadline first algorithm

23

Result

- Earliest Deadline First algorithm constructs a schedule that minimizes the maximum lateness

24

Homework Scheduling

- How is the model unrealistic?

25

Extensions

- What if the objective is to minimize the sum of the lateness?
  - EDF does not work
- If the tasks have release times and deadlines, and are non-preemptable, the problem is NP-complete
- What about the case with release times and deadlines where tasks are preemptable?

26

Optimal Caching

- Caching problem:
  - Maintain collection of items in local memory
  - Minimize number of items fetched

27

Caching example

A, B, C, D, A, E, B, A, D, A, C, B, D, A

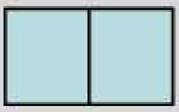
Optimal Caching

- If you know the sequence of requests, what is the optimal replacement pattern?
- Note – it is rare to know what the requests are in advance – but we still might want to do this:
  - Some specific applications, the sequence is known
    - Register allocation in code generation
  - Competitive analysis, compare performance on an online algorithm with an optimal offline algorithm

29

Farthest in the future algorithm

- Discard element used farthest in the future



A, B, C, A, C, D, C, B, C, A, D

30

Correctness Proof

- Sketch
- Start with Optimal Solution O
- Convert to Farthest in the Future Solution F-F
- Look at the first place where they differ
- Convert O to evict F-F element
  - There are some technicalities here to ensure the caches have the same configuration . . .

31

Later this week

32