CSE 417
Algorithms and Complexity

Richard Anderson

Autumn 2020

Lecture 10 – Greedy Algorithms III

Announcements

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

Midterm

- Pro:
  - Feedback on understanding, incentive for mastering material, complementary assessment to homework, established part of course
- Con:
  - Administrative difficulties in ensuring "exam conditions", time zones, extra work in multiple versions
- Approach:
  - Include a midterm in the homework problems, encourage students to first do midterm under exam conditions, but then redo problems under homework conditions. Count as a regular assignment.

Greedy Algorithms

- Solve problems with the simplest possible algorithm
- Today's problems (Sections 4.2, 4.3)
  - Homework Scheduling
  - Optimal Caching
- Start Dijkstra's shortest paths algorithm

Homework Scheduling

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

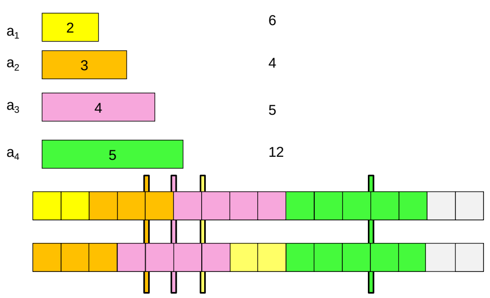
- Can I get all my work turned in on time?
- If I can't get everything in, I want to minimize the maximum lateness

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 $f_i >= d_i$

Determine the minimum lateness



Greedy Algorithm

- Earliest deadline first
- Order jobs by deadline

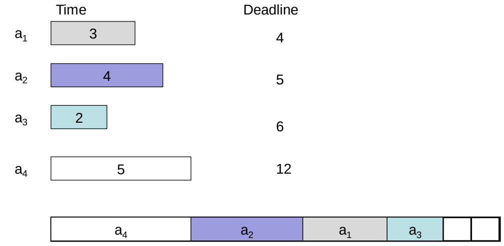
- This algorithm is optimal

Analysis

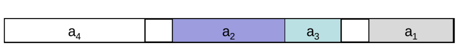
- Suppose the jobs are ordered by deadlines, $d_1 <= d_2 <= . . . <= d_n$
- A schedule has an *inversion* if job j is scheduled before i where j > i

- 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

List the inversions



Lemma: There is an optimal schedule with no idle time

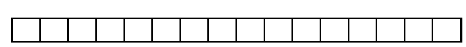


- It doesn't hurt to start your homework early!

- 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

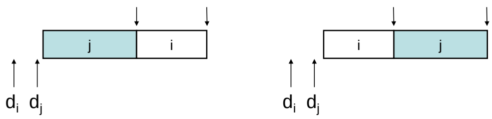
Lemma

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

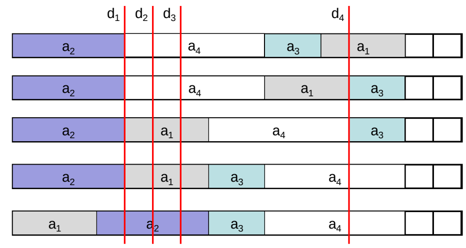


Interchange argument

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



Proof by Bubble Sort



Determine maximum lateness

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

Result

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

Homework Scheduling

- How is the model unrealistic?

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?

Optimal Caching

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

Caching example

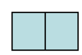
A, B, C, D, A, E, B, A, D, A, C, B, D, A

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

Farthest in the future algorithm

- Discard element used farthest in the future



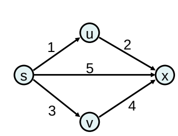
A, B, C, A, C, D, C, B, C, A, D

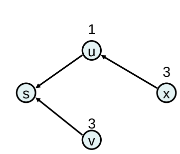
Correctness Proof

- 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 . . .

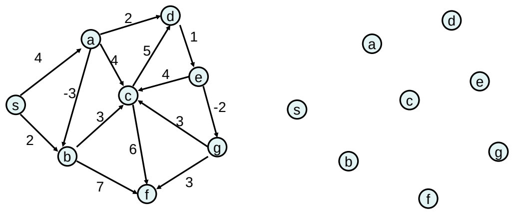
Single Source Shortest Path Problem

- Given a graph and a start vertex s
  - Determine distance of every vertex from s
  - Identify shortest paths to each vertex
    - Express concisely as a “shortest paths tree”
    - Each vertex has a pointer to a predecessor on shortest path



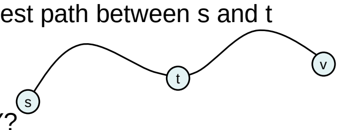


Construct Shortest Path Tree from s



Warmup

- If P is a shortest path from s to v, and if t is on the path P, the segment from s to t is a shortest path between s and t



- WHY?

Assume all edges have non-negative cost

Dijkstra's Algorithm

S = { }; d[s] = 0; d[v] = infinity for v != s

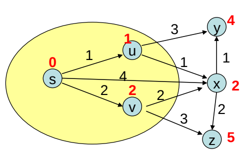
While S != V

Choose v in V-S with minimum d[v]

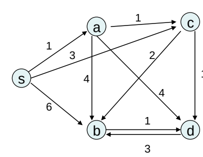
Add v to S

For each w in the neighborhood of v

d[w] = min(d[w], d[v] + c(v, w))



Simulate Dijkstra's algorithm (starting from s) on the graph



Round	Vertex Added	s	a	b	c	d
1						
2						
3						
4						
5						