CSE 417
Algorithms and Complexity

Richard Anderson

Autumn 2020

Lecture 8

Announcements

- Reading
  - For today, sections 4.1, 4.2,
  - For next week sections 4.4, 4.5, 4.7, 4.8
- Homework 3 is available
  - Random Graphs

Stable Matching Results

- Averages of 5 runs
- Much better for M than W
- Why is it better for M?
- What is the growth of m-rank and w-rank as a function of n?

n	m-rank	w-rank
500	5.10	98.05
500	7.52	66.95
500	8.57	58.18
500	6.32	75.87
500	5.25	90.73
500	6.55	77.95
1000	6.80	146.93
1000	6.50	154.71
1000	7.14	133.53
1000	7.44	128.96
1000	7.36	137.85
1000	7.04	140.40
2000	7.83	257.79
2000	7.50	263.78
2000	11.42	175.17
2000	7.16	274.76
2000	7.54	261.60
2000	8.29	246.62

Coupon Collector Problem

- n types of coupons
- Each round you receive a random coupon
- How many rounds until you have received all types of coupons
- p_i is the probability of getting a new coupon after i-1 have been collected
- t_i is the time to receive the i-th type of coupon after i-1 have been received

$$p_i = \frac{n-(i-1)}{n} = \frac{n-i+1}{n}$$

t_i has geometric distribution with expectation

$$\frac{1}{p_i} = \frac{n}{n-i+1}$$

$$\begin{aligned} \mathrm{E}(T) &= \mathrm{E}(t_1 + t_2 + \cdots + t_n) \\ &= \mathrm{E}(t_1) + \mathrm{E}(t_2) + \cdots + \mathrm{E}(t_n) \\ &= \frac{1}{p_1} + \frac{1}{p_2} + \cdots + \frac{1}{p_n} \\ &= \frac{n}{n} + \frac{n}{n-1} + \cdots + \frac{n}{1} \\ &= n \cdot \left(\frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}\right) \\ &= n \cdot H_n. \end{aligned}$$

$$\mathrm{E}(T) = n \cdot H_n = n \log n + \gamma n + \frac{1}{2} + O(1/n).$$

Stable Matching and Coupon Collecting

- Assume random preference lists
- Runtime of algorithm determined by number of proposals until all w's are matched
- Each proposal can be viewed[1] as asking a random w
- Number of proposals corresponds to number of steps in coupon collector problem

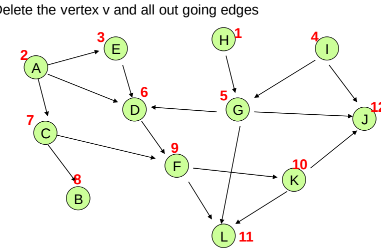
[1]There are some technicalities here that are being ignored

Highlight from last lecture: Topological Sort Algorithm

While there exists a vertex v with in-degree 0

Output vertex v

Delete the vertex v and all out going edges



Greedy Algorithms

Greedy Algorithms

- Solve problems with the simplest possible algorithm
- The hard part: showing that something simple actually works
- Pseudo-definition
  - An algorithm is Greedy if it builds its solution by adding elements one at a time using a simple rule

Scheduling Theory

- Tasks
  - Processing requirements, release times, deadlines
- Processors
- Precedence constraints
- Objective function
  - Jobs scheduled, lateness, total execution time

Interval Scheduling

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

- Tasks {1, 2, . . . N}
- Start and finish times, s(i), f(i)

What is the largest solution?

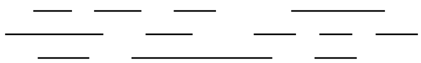
Greedy Algorithm for Scheduling

Let T be the set of tasks, construct a set of independent tasks I, A is the rule determining the greedy algorithm

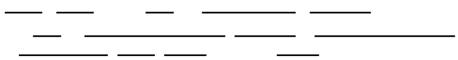
```
I = { }
While (T is not empty)
        Select a task t from T by a rule A
        Add t to I
        Remove t and all tasks incompatible with t from T
```

Simulate the greedy algorithm for each of these heuristics

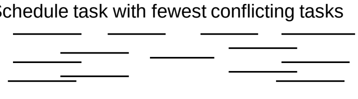
Schedule earliest starting task



Schedule shortest available task

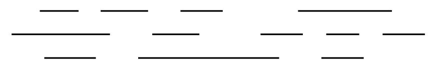


Schedule task with fewest conflicting tasks

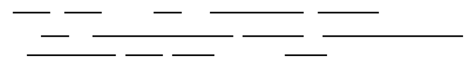


Greedy solution based on earliest finishing time

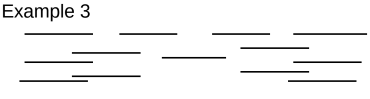
Example 1



Example 2



Example 3



Theorem: Earliest Finish Algorithm is Optimal

- Key idea: Earliest Finish Algorithm stays ahead
- Let A = $\{i_1, . . ., i_k\}$ be the set of tasks found by EFA in increasing order of finish times
- Let B = $\{j_1, . . ., j_m\}$ be the set of tasks found by a different algorithm in increasing order of finish times
- Show that for $r <= \min(k, m)$, $f(i_r) <= f(j_r)$

Stay ahead lemma

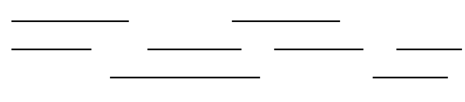
- A always stays ahead of B, $f(i_r) <= f(j_r)$
- Induction argument
  - – $f(i_1) <= f(j_1)$
  - – If $f(i_{r-1}) <= f(j_{r-1})$ then $f(i_r) <= f(j_r)$

Completing the proof

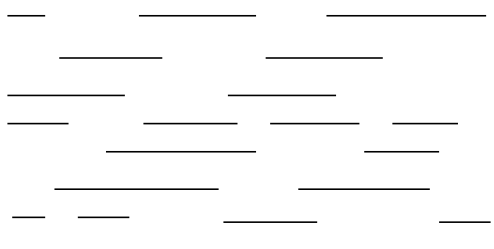
- Let A = $\{i_1, . . ., i_k\}$ be the set of tasks found by EFA in increasing order of finish times
- Let O = $\{j_1, . . ., j_m\}$ be the set of tasks found by an optimal algorithm in increasing order of finish times
- If $k < m$, then the Earliest Finish Algorithm stopped before it ran out of tasks

Scheduling all intervals

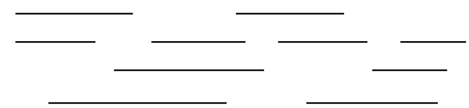
- Minimize number of processors to schedule all intervals



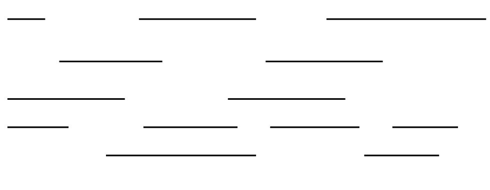
How many processors are needed for this example?



Prove that you cannot schedule this set of intervals with two processors



Depth: maximum number of intervals active



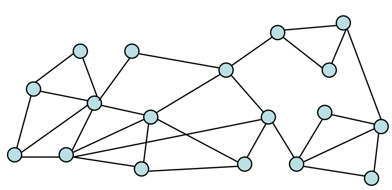
Algorithm

- Sort by start times
- Suppose maximum depth is d, create d slots
- Schedule items in increasing order, assign each item to an open slot

- Correctness proof: When we reach an item, we always have an open slot

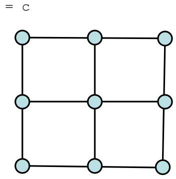
Greedy Graph Coloring

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



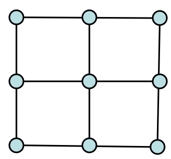
Coloring Algorithm, Version 1

```
Let k be the largest vertex degree
Choose k 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
```



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
```



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 = $f_i - d_i$ if $f_i >= d_i$

Example

Time	Deadline
2	2
3	4

2 | 3 — Lateness 1

3 | 2 — Lateness 3

Determine the minimum lateness

Time	Deadline
2	6
3	4
4	5
5	12