GREEDY ALGORITHM

CS6202 COMPUTABILITY, COMPLEXITY AND ALGORITHMS

KWANKAMOL NONGPONG

PROBLEM TYPES

- Decision Problem: Yes/No
  - Given graph G, nodes s; t, is there a path from s to t in G?
- Search Problem
  - Given graph G, nodes s; t, find a path from s to t in G.
- Optimization Problem
  - Given graph G nodes s; t, find the shortest path from s to t.

WHAT IS A GREEDY ALGORITHM?

- A greedy algorithm is an algorithm that follows problem solving heuristic of
  - making locally optimal choice at each stage
  - with the hope of finding a global optimal.
- Make decision incrementally in small steps without backtracking.
- Decisions often based on some fixed and simple priority rules.

ISSUES ON GREEDY ALGORITHMS

- In many problems, a greedy approach does not yield an optimal solution.
- Nonetheless, it may produce locally optimal solutions that approximate a global optimal solution in a reasonable time.

CASE STUDIES

- Activity Selection
- Minimum Max-lateness Scheduling

ACTIVITY SELECTION

ACTIVITY SELECTION

- The problem concerns the selection of non-conflicting activities to perform within a given time frame.
- Given a set of activities each marked by
  - start time (s_i)
  - finish time (f_i)
- Goal
  - Select the maximum number of activities that can be performed by a single person or machine, assuming that a person can only work on a single activity at a time.

EXAMPLE 1

- Consider the following 6 activities:

Activity#	0	1	2	3	4	5
start	1	3	0	5	8	5
finish	2	4	6	7	9	9

- What is the largest set of activities that can be performed by a single person?
- The largest set of activities that can be performed by a single person is {0, 1, 3, 4}.

EXAMPLE 2

- Consider the following 6 activities:

Activity#	0	1	2	3	4	5
start	1	5	3	3	6	0
finish	2	8	6	4	9	1

- The largest set of activities that can be performed by a single person is {5, 0, 3, 1}.

GREEDY ALGORITHM

- Concept
  - Consider an activity with earliest finish time first
  - Based on the sorted order, select an activity whose start time is larger than the finish time of the previous activity (start time and finish time don’t overlap)

ALGORITHM: ACTIVITY SELECTION

```
Sort the set of activities by finishing time f[i]
S = {0}
f = f[0]
for i = 1 to n - 1
  if s[i] ≥ f
   S = S U i
   f = f[i]
  endif
endfor
```

EXAMPLE 1: REVISITED

- Consider the following 6 activities:

Activity#	0	1	2	3	4	5
start	1	3	0	5	8	5
finish	2	4	6	7	9	9

- Sort by finish time

Activity#	0	1	2	3	4	5
start	1	3	0	5	8	5
finish	2	4	6	7	9	9

0, 1, 3, 4

EXAMPLE 2: REVISITED

- Consider the following 6 activities:

Activity#	0	1	2	3	4	5
start	1	5	3	3	6	0
finish	2	8	6	4	9	1

- Sort by finish time

Activity#	5	0	3	2	1	4
start	0	1	3	3	5	6
finish	1	2	4	6	8	9

5, 0, 3, 1

SAMPLE INPUT / OUTPUT

Sample Input	Sample Output
6 1 2 3 4 0 6 5 7 8 9 5 9	0 1 3 4
6 1 2 5 8 3 6 3 4 6 9 0 1	5 0 3 1

MINIMUM MAX-LATENESS SCHEDULING

MINIMUM MAX-LATENESS SCHEDULING

- Different scheduling problem
- Start and end times are not given.
- Instead, requests are of the form (t_i, d_i) where:
  - t_i is the job length, and
  - d_i is the job deadline
- We want to schedule these jobs on a single processor.
  - Schedule all jobs in a sequence.
- Definition
  - The lateness l_i of job i is max{0, f_i- d_i}, i.e. the length of time past its deadline that it finishes
- Goal
  - Minimize the maximum lateness

EXAMPLE: FIRST ATTEMPT

Job No	0	1	2	3	4	5
length	1	2	2	3	3	4
deadline	9	8	15	6	14	9

- ollowing 6 jobs.

lateness = 0

lateness = 2

lateness = 0

lateness = 6

	Job 0	Job 1	Job 2	Job 3	Job 4	Job 5
d	9	8	15	6	14	9

t 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

lateness = 0

lateness = 0

EXAMPLE: SECOND ATTEMPT

Job No	0	1	2	3	4	5
length	1	2	2	3	3	4
deadline	9	8	15	6	14	9

- Consider the following 6 jobs.

lateness = 1

lateness = 0

lateness = 0

	Job 3	Job 5	Job 1	Job 0	Job 4	Job 2
d	6	9	8	9	14	15

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

lateness = 0

lateness = 0

lateness = 1

EXAMPLE: THIRD ATTEMPT

Job No	0	1	2	3	4	5
length	1	2	2	3	3	4
deadline	9	8	15	6	14	9

- Consider the fo

lateness = 0

lateness = 1

lateness = 0

lateness = 0

	Job 3	Job 1	Job 0	Job 5	Job 4	Job 2
d	6	8	9	9	14	15

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

lateness = 0

lateness = 0

POSSIBLE GREEDY ALGORITHMS

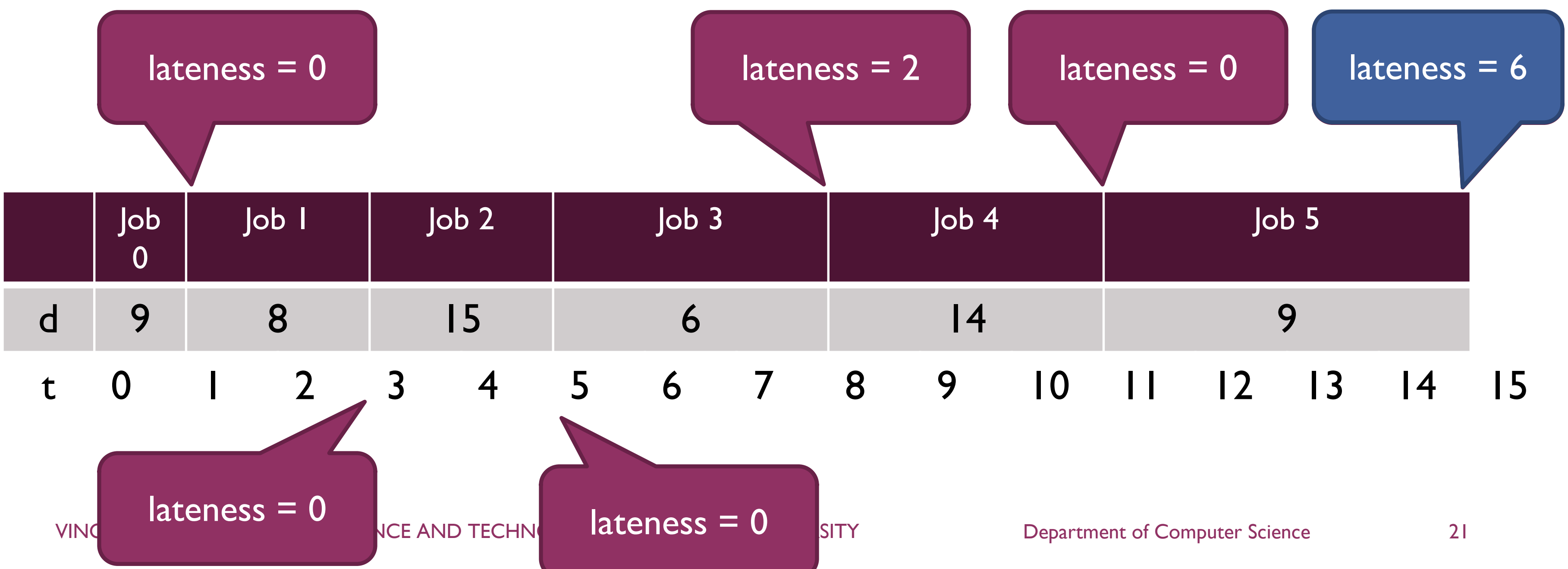
- Concepts:
  - Shortest jobs first t_i
  - Smallest slack time $d_i - t_i$

- Will any of these algorithms work?

SHORTEST JOBS FIRST T_l

That's basically our first attempt!

Job No	0	1	2	3	4	5
length	1	2	2	3	3	4
deadline	9	8	15	6	14	9



SMALLEST SLACK TIME $D_I - T_I$

Job No	0	1	2	3	4	5
length	1	2	2	3	3	4
deadline	9	8	15	6	14	9
slack time	8	6	13	3	11	5

- Sort by slack time

Job No	3	5	1	0	4	2
length	3	4	2	1	3	2
deadline	6	9	8	9	14	15
slack time	**3**	**5**	**6**	**8**	**11**	**13**

SMALLEST SLACK TIME $D_I - T_I$ (CONT

That's basically our second attempt!

Job No	3	5	1	0	4	2
length	3	4	2	1	3	2
deadline	6	9	8	9	14	15
slack time	**3**	**5**	**6**	**8**	**11**	**13**

lateness = 1

lateness = 0

lateness = 0

	Job 3	Job 5	Job 1	Job 0	Job 4	Job 2
d	6	9	8	9	14	15

t 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

lateness = 0

lateness = 0

lateness = 1

CORRECT GREEDY ALGORITHM

- Earliest deadline first

```
Sort the job by increasing deadline d[i]
f = s
for i = 1 to n
    Schedule job i starting from time f to f + t[i]
    f = f + t[i]
endfor
```

EXAMPLE: REVISITED

Job No	0	1	2	3	4	5
length	1	2	2	3	3	4
deadline	9	8	15	6	14	9

- Consider the fo

lateness = 0

lateness = 1

lateness = 0

lateness = 0

	Job 3	Job 1	Job 0	Job 5	Job 4	Job 2
d	6	8	9	9	14	15

t 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

lateness = 0

lateness = 0

EXERCISE 2

Sample Input	Sample Output
6 1 9 2 8 2 15 3 5 3 14 4 9	3 1 0 5 4 2 1
5 1 1 4 5 2 3 3 3 5 5	0 2 3 1 4 10

EXERCISE 3: CLASSROOM SCHEDULING

- The problem concerns the assignment of non-conflicting class time to a number of classrooms.
- Given a set of classes/lectures each marked by
  - start time (s_i)
  - finish time (f_i)
- Goal
  - Find the minimum number of classrooms that are needed for all the given classes such that two lectures do not occur at the same time in the same room.

EXERCISE 3: INPUT & OUTPUT FORMAT

- Input
  - Line 1: The number of classes/lectures to be conducted (n)
  - Line 2 to n+1: The start and finish time of class i (separated by a space)
- Output
  - Line 1: The number of classrooms that are needed (c).
  - Line 2 to c+1: lectures number to be conducted in each classroom.

EXERCISE 3: SAMPLE INPUT/OUTPUT

Sample Input	Sample Output
3 9:00 12:00 12:00 13:30 13:30 16:30	1 1 2 3
5 9:00 12:00 13:30 16:30 10:30 13:30 13:30 16:30 10:30 12:00	3 1 2 3 4 5