More greed!

10/23/24

Recall: Greedy algorithms

- Use a simple rule to pick part of the solution, generally in a locally-best way
- Then, prune choices this makes impossible and repeat
- Greedy algorithms don’t always work, but they do for some problems

- Proving they work: Suppose the greedy rule doesn’t allow an optimal solution. Take a solution that is optimal and change it to include the greedy choice. Show that this creates an optimal solution that (now) includes the greedy choice, contradicting the original assumption.

Application: Minimizing maximum lateness

- Set of jobs $\{J_1, J_2, \ldots, J_n\}$
  - Job J_i has duration p_i and deadline d_i
- A schedule S assigns jobs to run one at a time, J_i starting at s_i^S and finishing at $f_i^S = s_i^S + p_i$
  - The job's lateness is $\max \{ 0, f_i^S - d_i \}$
- Goal: Construct schedule that minimizes maximum lateness

Application: Minimizing maximum lateness

- Set of jobs $\{J_1, J_2, \ldots, J_n\}$
  - Job J_i has duration p_i and deadline d_i
- A schedule S assigns jobs to run one at a time, J_i starting at s_i^S and finishing at $f_i^S = s_i^S + p_i$
  - The job's lateness is $\max \{ 0, f_i^S - d_i \}$
- Goal: Construct schedule that minimizes maximum lateness

- Earliest Deadline First (EDF): Run the job with minimal d_i

Claim: There is a schedule minimizing max-lateness that starts with the job with earliest deadline

Let J_i be the job with earliest deadline and S be a schedule minimizing max-lateness that doesn't start with J_i.

Claim: There is a schedule minimizing max-lateness that starts with the job with earliest deadline

Let J_i be the job with earliest deadline and S be a schedule minimizing max-lateness that doesn't start with J_i.

WLOG, assume S doesn't include unnecessary idle time.

Claim: There is a schedule minimizing max-lateness that starts with the job with earliest deadline

Let J_i be the job with earliest deadline and S be a schedule minimizing max-lateness that doesn't start with J_i.

WLOG, assume S doesn't include unnecessary idle time.

Construct S' from S as follows:

- Start with J_i
- All jobs running before J_i in S are delayed by p_i
- All jobs running after J_i in S are unchanged

Looking at latenesses

- Jobs running after J_i in S: No change in position so same lateness
- J_i itself: Runs earlier so lateness unchanged or improved

Looking at latenesses

- Jobs running after J_i in S: No change in position so same lateness
- J_i itself: Runs earlier so lateness unchanged or improved
- Jobs running before J_i in S: delayed by p_i

Looking at latenesses

- Jobs running after J_i in S: No change in position so same lateness
- J_i itself: Runs earlier so lateness unchanged or improved
- Jobs running before J_i in S: delayed by p_i

  Let J_j be such a job. Its lateness in S' is either 0 or

  $f_j^{S'} - d_j$

Looking at latenesses

- Jobs running after J_i in S: No change in position so same lateness
- J_i itself: Runs earlier so lateness unchanged or improved
- Jobs running before J_i in S: delayed by p_i

  Let J_j be such a job. Its lateness in S' is either 0 or

  $$f_j^{S'} - d_j \leqslant f_i^{S} - d_j$$

Looking at latenesses

- Jobs running after J_i in S: No change in position so same lateness
- J_i itself: Runs earlier so lateness unchanged or improved
- Jobs running before J_i in S: delayed by p_i

Let J_j be such a job. Its lateness in S' is either 0 or

$$
\begin{aligned}
f_j^{S'} - d_j &\leqslant f_i^{S} - d_j \\
&\leqslant f_i^{S} - d_i
\end{aligned}
$$

Looking at latenesses

- Jobs running after J_i in S: No change in position so same lateness
- J_i itself: Runs earlier so lateness unchanged or improved
- Jobs running before J_i in S: delayed by p_i

Let J_j be such a job. Its lateness in S' is either 0 or

$$\begin{aligned} f_j^{S'} - d_j &\leqslant f_i^{S} - d_j \\ &\leqslant f_i^{S} - d_i \\ &\leqslant \text{lateness of } J_i \text{ in S} \end{aligned}$$

Making change

Given coin denominations d_1, d_2, ..., d_n, what is the fewest coins needed to make change C?

Show that giving the largest denomination possible is optimal if $d_1 = 1$, $d_2 = 5$, $d_3 = 10$, and $d_4 = 25$ ($n=4$).

Making change

Claim: Giving the largest denomination possible is optimal (i.e. uses the fewest coins) if $d_1 = 1$, $d_2 = 5$, $d_3 = 10$, and $d_4 = 25$ (n=4).

Proof: Suppose the optimal solution doesn't use the largest possible denomination X. We break into cases based on X.

It can't be d_1=1 since there isn't anything smaller. Can't be d_2=5 since any solution for C >= 5 cents using only d_1s can be made better by replacing 5 of them with a d_2. Similarly, any solution for C >= 10 using only d_1s and d_2s has coins totaling 10 that can be replaced with a d_3.

Thus, suppose X is d_4. By the reasoning above, the optimal solution must use as many d_3s as possible. There can't be 3 of these or replacing them with a d_4 and a d_2 would reduce the number of coins. Therefore C must be in the range 25-29. Each of these can be eliminated by case analysis.

Application: Minimizing total flow

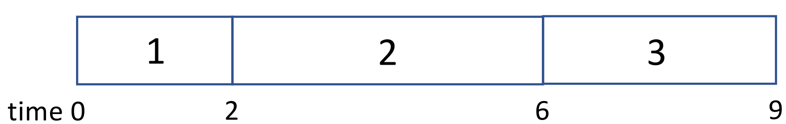
- Set of jobs $\{J_1, J_2, \ldots, J_n\}$
  - Job J_i has duration p_i and a release time r_i
- A schedule S assigns jobs to run one at a time, J_i starting at $s_i^S \geq r_i$ and finishing at $f_i^S = s_i^S + p_i$
  - The job's flow is $f_i^S - r_i$
- Goal: Construct schedule that minimizes the total flow (sum of every job's flow) when every job's release time is 0 ($r_i = 0$ for all i)

Application: Minimizing total flow

- Set of jobs $\{J_1, J_2, \ldots, J_n\}$
  - Job J_i has duration p_i and a release time r_i
- A schedule S assigns jobs to run one at a time, J_i starting at $s_i^S \geq r_i$ and finishing at $f_i^S = s_i^S + p_i$
  - The job's flow is $f_i^S - r_i$
- Goal: Construct schedule that minimizes the total flow (sum of every job's flow) when every job's release time is 0 ($r_i = 0$ for all i)

i	$\mathbf{r_i}$	$\mathbf{p_i}$
1	0	2
2	0	4
3	0	3

What is the total flow for this schedule?



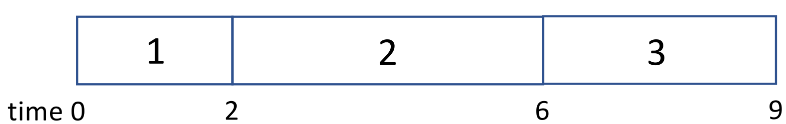
A. 9
B. 12
C. 17
D. 25
E. None of the above

Application: Minimizing total flow

- Set of jobs $\{J_1, J_2, \ldots, J_n\}$
  - Job J_i has duration p_i and a release time r_i
- A schedule S assigns jobs to run one at a time, J_i starting at $s_i^S \geq r_i$ and finishing at $f_i^S = s_i^S + p_i$
  - The job's flow is $f_i^S - r_i$
- Goal: Construct schedule that minimizes the total flow (sum of every job's flow) when every job's release time is 0 ($r_i = 0$ for all i)

i	**r_i**	**p_i**
1	0	2
2	0	4
3	0	3

What is the total flow for this schedule?



A. 9
B. 12
C. 17
D. 25
E. None of the above

Application: Minimizing total flow

- Set of jobs $\{J_1, J_2, \ldots, J_n\}$
  - Job J_i has duration p_i and a release time r_i
- A schedule S assigns jobs to run one at a time, J_i starting at $s_i^S \geq r_i$ and finishing at $f_i^S = s_i^S + p_i$
  - The job's flow is $f_i^S - r_i$
- Goal: Construct schedule that minimizes the total flow (sum of every job's flow) when every job's release time is 0 ($r_i = 0$ for all i)

What algorithm should we try for this?

A. First-in first-out (min r_i first)
B. Last-in first-out (max r_i first)
C. Shortest processing time (min p_i first)
D. Longest processing time (max p_i first)
E. Let's end class now and just do them all for HW

Application: Minimizing total flow

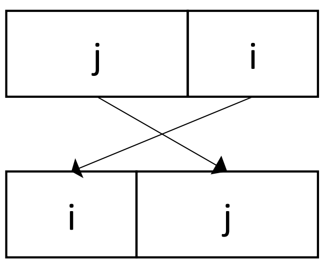
- Set of jobs $\{J_1, J_2, \ldots, J_n\}$
  - Job J_i has duration p_i and a release time r_i
- A schedule S assigns jobs to run one at a time, J_i starting at $s_i^S \geq r_i$ and finishing at $f_i^S = s_i^S + p_i$
  - The job's flow is $f_i^S - r_i$
- Goal: Construct schedule that minimizes the total flow (sum of every job's flow) when every job's release time is 0 ($r_i = 0$ for all i)

What algorithm should we try for this?

A. First-in first-out (min r_i first)
B. Last-in first-out (max r_i first)
C. Shortest processing time (min p_i first)
D. Longest processing time (max p_i first)
E. Let's end class now and just do them all for HW

Shortest Processing Time (SPT)

- Suppose SPT doesn't give an optimal schedule
- Consider an optimal schedule and consider a place where it runs a longer job J_j immediately before a shorter job J_i
- Swap the order of those jobs

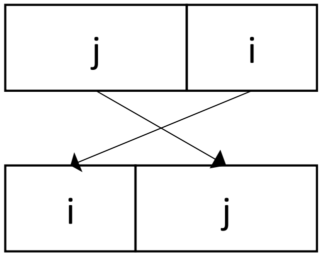


$p_i < p_j$
so change in flow is
$p_i - p_j < 0$

- Repeating this gives SPT while improving at every step

Shortest Processing Time (SPT)

- Suppose SPT doesn't give an optimal schedule
- Consider an optimal schedule and consider a place where it runs a longer job J_j immediately before a shorter job J_i
- Swap the order of those jobs



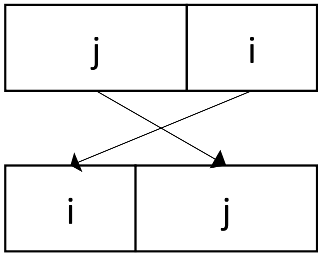
$p_i < p_j$
so change in flow is
$p_i - p_j < 0$

> Does it still work if we remove the requirement that $r_i = 0$ for all i? (Whenever idle, start the shortest job that has been released.)
> A. Yes
> B. No
> Either adapt the proof or give a counterexample

- Repeating this gives SPT while improving at every step

Shortest Processing Time (SPT)

- Suppose SPT doesn't give an optimal schedule
- Consider an optimal schedule and consider a place where it runs a longer job J_j immediately before a shorter job J_i
- Swap the order of those jobs



$p_i < p_j$
so change in flow is
$p_i - p_j < 0$

Does it still work if we remove the requirement that $r_i = 0$ for all i? (Whenever idle, start the shortest job that has been released.)

A. Yes
B. <u>No</u>

Either adapt the proof or give a counterexample

- Repeating this gives SPT while improving at every step

Application: Encoding data

- How many bits does it take to encode 100 Java chars?

Application: Encoding data

- How many bits does it take to encode 100 Java chars?
  - Suppose you know they are all a thru e? How many bits per char?

A. 1
B. 2
C. 3
D. 5
E. None of the above

Application: Encoding data

- How many bits does it take to encode 100 Java chars?
  - Suppose you know they are all a thru e? How many bits per char?

A. 1
B. 2
C. 3
D. 5
E. None of the above

Application: Encoding data

- How many bits does it take to encode 100 Java chars?
  - Suppose you know they are all a thru e? How many bits per char?
  - What if you know they have a specific frequency?

char	a	b	c	d	e
#times	47	11	20	5	17

Huffman encoding

char	a	b	c	d	e
#times	47	11	20	5	17
encoding	0	1111	10	1110	110

- Length: 47 + 2*20 + 3*17 + 4*(11+5) = 202

Each term: length of encoding * #times

Huffman encoding

char	a	b	c	d	e
#times	47	11	20	5	17
encoding	0	1111	10	1110	110

- Length: 47 + 2*20 + 3*17 + 4*(11+5) = 202

Each term: ~~length of encoding~~ * #times
depth in tree