4.2 Scheduling to Minimize Lateness
Minimizing lateness problem.

- Single resource processes one job at a time.
- Job $j$ requires $t_j$ units of processing time and is due at time $d_j$.
- If $j$ starts at time $s_j$, it finishes at time $f_j = s_j + t_j$.
- Lateness: $\ell_j = \max \{ 0, f_j - d_j \}$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max \ell_j$.

**Ex:**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_j$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$d_j$</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

lateness $= 2$
lateness $= 0$
max lateness $= 6$

$d_3 = 9$
d$_2 = 8$
d$_6 = 15$
d$_1 = 6$
d$_5 = 14$
d$_4 = 9$
Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- **[Shortest processing time first]** Consider jobs in ascending order of processing time $t_j$.

- **[Earliest deadline first]** Consider jobs in ascending order of deadline $d_j$.

- **[Smallest slack]** Consider jobs in ascending order of slack $d_j - t_j$. 
Minimizing Lateneness: Greedy Algorithms

**Greedy template.** Consider jobs in some order.

- **[Shortest processing time first]** Consider jobs in ascending order of processing time $t_j$.

  
  \[
  \begin{array}{c|c|c}
  & 1 & 2 \\
  \hline
  t_j & 1 & 10 \\
  d_j & 100 & 10 \\
  \end{array}
  \]

  counterexample

- **[Smallest slack]** Consider jobs in ascending order of slack $d_j - t_j$.

  
  \[
  \begin{array}{c|c|c}
  & 1 & 2 \\
  \hline
  t_j & 1 & 10 \\
  d_j & 2 & 10 \\
  \end{array}
  \]

  counterexample
Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

Sort \( n \) jobs by deadline so that \( d_1 \leq d_2 \leq \ldots \leq d_n \)

\[
\begin{align*}
  &t \leftarrow 0 \\
  &\text{for } j = 1 \text{ to } n \\
  &\quad \text{Assign job } j \text{ to interval } [t, t + t_j] \\
  &\quad s_j \leftarrow t, f_j \leftarrow t + t_j \\
  &\quad t \leftarrow t + t_j \\
  &\text{output intervals } [s_j, f_j]
\end{align*}
\]

<table>
<thead>
<tr>
<th>( d_1 = 6 )</th>
<th>( d_2 = 8 )</th>
<th>( d_3 = 9 )</th>
<th>( d_4 = 9 )</th>
<th>( d_5 = 14 )</th>
<th>( d_6 = 15 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

max lateness = 1
Minimizing Lateness: No Idle Time

**Observation.** There exists an optimal schedule with no *idle time*.

![Diagram showing schedules for different values of d](image)

**Observation.** The greedy schedule has no idle time.
**Minimizing Lateness: Inversions**

**Def.** An *inversion* in schedule $S$ is a pair of jobs $i$ and $j$ such that: $i < j$ but $j$ scheduled before $i$.  

![Diagram showing an inversion in schedule with jobs $j$ and $i$ scheduled out of order.]  

**Observation.** Greedy schedule has no inversions.

**Observation.** If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.
Minimizing Lateness: Inversions

Def. An inversion in schedule S is a pair of jobs i and j such that: i < j but j scheduled before i.

Claim. Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

Pf. Let $\ell$ be the lateness before the swap, and let $\ell'$ be it afterwards.

- $\ell'_k = \ell_k$ for all $k \neq i, j$
- $\ell'_i \leq \ell_i$
- If job j is late:

$$
\begin{align*}
\ell'_j &= f'_j - d_j \quad \text{(definition)} \\
&= f_i - d_j \quad \text{((j finishes at time $f_i$))} \\
&\leq f_i - d_i \quad \text{(i < j)} \\
&\leq \ell_i \quad \text{(definition)}
\end{align*}
$$
Minimizing Lateness: Analysis of Greedy Algorithm

Theorem. Greedy schedule $S$ is optimal.

Pf. Define $S^*$ to be an optimal schedule that has the fewest number of inversions, and let's see what happens.

- Can assume $S^*$ has no idle time.
- If $S^*$ has no inversions, then $S = S^*$.
- If $S^*$ has an inversion, let $i-j$ be an adjacent inversion.
  - swapping $i$ and $j$ does not increase the maximum lateness and strictly decreases the number of inversions
  - this contradicts definition of $S^*$.


Greedy Analysis Strategies

**Greedy algorithm stays ahead.** Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.

**Exchange argument.** Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

**Structural.** Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.
Selecting Breakpoints
Selecting Breakpoints

Selecting breakpoints.

- Road trip from Princeton to Palo Alto along fixed route.
- Refueling stations at certain points along the way.
- Fuel capacity = C.
- Goal: makes as few refueling stops as possible.

**Greedy algorithm.** Go as far as you can before refueling.
Selecting Breakpoints: Greedy Algorithm

Truck driver's algorithm.

Sort breakpoints so that: $0 = b_0 < b_1 < b_2 < \ldots < b_n = L$

$S \leftarrow \{0\}$  —— breakpoints selected
$x \leftarrow 0$  —— current location

while $(x \neq b_n)$

let $p$ be largest integer such that $b_p \leq x + C$

if $(b_p = x)$

return "no solution"

$x \leftarrow b_p$

$S \leftarrow S \cup \{p\}$

return $S$

Implementation.  $O(n \log n)$

- Use binary search to select each breakpoint $p$. 
Selecting Breakpoints: Correctness

**Theorem.** Greedy algorithm is optimal.

**Pf.** (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let \(0 = g_0 < g_1 < \ldots < g_p = L\) denote set of breakpoints chosen by greedy.
- Let \(0 = f_0 < f_1 < \ldots < f_q = L\) denote set of breakpoints in an optimal solution with \(f_0 = g_0, f_1 = g_1, \ldots, f_r = g_r\) for largest possible value of \(r\).
- Note: \(g_{r+1} > f_{r+1}\) by greedy choice of algorithm.

![Diagram showing greedy and optimal selections]

In the diagram:
- **Greedy:** \(g_0, g_1, g_2, g_r, g_{r+1}\)
- **OPT:** \(f_0, f_1, f_2, f_r, f_{r+1}, \ldots, f_q\)

*why doesn't optimal solution drive a little further?*
Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)
- Assume greedy is not optimal, and let's see what happens.
- Let \( G = g_0 < g_1 < \ldots < g_p = L \) denote set of breakpoints chosen by greedy.
- Let \( O = f_0 < f_1 < \ldots < f_q = L \) denote set of breakpoints in an optimal solution with \( f_0 = g_0, f_1 = g_1, \ldots, f_r = g_r \) for largest possible value of \( r \).
- Note: \( g_{r+1} > f_{r+1} \) by greedy choice of algorithm.

Another optimal solution has one more breakpoint in common ⇒ contradiction.
4.4 Shortest Paths in a Graph

shortest path from Princeton CS department to Einstein's house
Edsger W. Dijkstra

The question of whether computers can think is like the question of whether submarines can swim.

Do only what only you can do.

In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind.

The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence.

APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.
Shortest path network.
- Directed graph \( G = (V, E) \).
- Source \( s \), destination \( t \).
- Length \( \ell_e \) = length of edge \( e \).

**Shortest path problem:** find shortest directed path from \( s \) to \( t \).

Cost of path = sum of edge costs in path

Cost of path \( s-2-3-5-t \):

\[
= 9 + 23 + 2 + 16 = 48.
\]
Dijkstra's algorithm.

- Maintain a set of explored nodes $S$ for which we have determined the shortest path distance $d(u)$ from $s$ to $u$.
- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose unexplored node $v$ which minimizes

$$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e,$$

add $v$ to $S$, and set $d(v) = \pi(v)$.

shortest path to some $u$ in explored part, followed by a single edge $(u, v)$
Dijkstra's Algorithm

Dijkstra's algorithm.

- Maintain a set of explored nodes $S$ for which we have determined the shortest path distance $d(u)$ from $s$ to $u$.
- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose unexplored node $v$ which minimizes $\pi(v)$:

$$
\pi(v) = \min_{e = (u, v) : u \in S} d(u) + \ell_e,
$$

add $v$ to $S$, and set $d(v) = \pi(v)$.

![Diagram of Dijkstra's Algorithm](image)
Dijkstra's Algorithm: Proof of Correctness

Invariant. For each node $u \in S$, $d(u)$ is the length of the shortest $s$-$u$ path.

Pf. (by induction on $|S|$)

Base case: $|S| = 1$ is trivial.

Inductive hypothesis: Assume true for $|S| = k \geq 1$.

- Let $v$ be next node added to $S$, and let $u$-$v$ be the chosen edge.
- The shortest $s$-$u$ path plus $(u, v)$ is an $s$-$v$ path of length $\pi(v)$.
- Consider any $s$-$v$ path $P$. We'll see that it's no shorter than $\pi(v)$.
- Let $x$-$y$ be the first edge in $P$ that leaves $S$, and let $P'$ be the subpath to $x$.
- $P$ is already too long as soon as it leaves $S$.

\[
\ell(P) \geq \ell(P') + \ell(x,y) \geq d(x) + \ell(x,y) \geq \pi(y) \geq \pi(v)
\]

- nonnegative weights
- inductive hypothesis
- defn of $\pi(y)$
- Dijkstra chose $v$ instead of $y$
Dijkstra's Algorithm: Implementation

For each unexplored node, explicitly maintain $\pi(v) = \min_{e = (u,v): u \in S} d(u) + \ell_e$.

- Next node to explore = node with minimum $\pi(v)$.
- When exploring $v$, for each incident edge $e = (v, w)$, update

$$\pi(w) = \min \{ \pi(w), \pi(v) + \ell_e \}.$$

Efficient implementation. Maintain a priority queue of unexplored nodes, prioritized by $\pi(v)$.

<table>
<thead>
<tr>
<th>PQ Operation</th>
<th>Dijkstra</th>
<th>Array</th>
<th>Binary heap</th>
<th>d-way Heap</th>
<th>Fib heap $\dagger$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>$n$</td>
<td>$n$</td>
<td>$\log n$</td>
<td>$d \log_d n$</td>
<td>$1$</td>
</tr>
<tr>
<td>ExtractMin</td>
<td>$n$</td>
<td>$n$</td>
<td>$\log n$</td>
<td>$d \log_d n$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>ChangeKey</td>
<td>$m$</td>
<td>$1$</td>
<td>$\log n$</td>
<td>$\log_d n$</td>
<td>$1$</td>
</tr>
<tr>
<td>IsEmpty</td>
<td>$n$</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>Total</td>
<td>$n^2$</td>
<td>$m \log n$</td>
<td>$m \log_{m/n} n$</td>
<td>$m + n \log n$</td>
<td></td>
</tr>
</tbody>
</table>

$\dagger$ Individual ops are amortized bounds