Fundamental Algorithms  
Topic 6: Shortest Paths

By Adam Sheffer

Naïve Path Planning

• **Problem.**
  ◦ We are given a map with cities and non-crossing roads between pairs of cities.
  ◦ Describe an algorithm for finding a path between city A and city B that minimizes the number of roads traveled.
Solution

- Turn the problem into a graph:
  - Every city is a vertex.
  - Every road is an edge.
  - Find a path from vertex $A$ to vertex $B$ that consists of a minimum number of edges.
  - Immediate by running BFS from $A$.

Slightly Less Naïve Path Planning

- **Problem.** Same as previous problem, except that roads are allowed to cross (contain intersections).
  - Minimize the number of road segments that were traveled.
Solution

• Turn the problem into a graph:
  ◦ Every city and every intersection is a vertex.
  ◦ Every road segment is an edge.
  ◦ Find a path from vertex $A$ to vertex $B$ that consists of a minimum number of edges.
  ◦ Immediate by running BFS from $A$.

Path Planning

• Problem. Given a map with roads, find a path from city $A$ to city $B$, that minimizes the distance travelled (or the cost).
Formulating the Problem

• Turn the problem into a graph:
  ◦ Every city and every intersection is a vertex.
  ◦ Every road segment is an edge.
  ◦ A weight function $w: E \rightarrow \mathbb{R}$ that gives every edge its distance.
  ◦ Find a path from vertex $A$ to vertex $B$ that minimizes the sum of its edge weights.

Shortest/Lightest Paths Problem

• Consider a directed graph $G = (V, E)$ and a weight function $w: E \rightarrow \mathbb{R}$.
• The weight of a path $P$ is the sum of the weights of its edges.
• Problem. Given a pair of vertices $s, t \in V$, find a shortest path between $s$ and $t$ (a path of minimum weight).
Confusing Notation

- The term **shortest path** refers to two different things:
  - A path **minimizing the number of edges** (the kind of path the BFS algorithm finds).
  - A path **minimizing the sum of the edge weights**.

- To avoid confusion, we refer to the latter as a **lightest path**.
  - This is not completely standard – be careful when reading a textbook, Wikipedia, etc.

Test Your Intuition

- **Problem.**
  - We are given an efficient algorithm for finding a lightest path in a **directed graph**.
  - Can we use this algorithm to find a lightest path in an **undirected graph**?
Test Your Intuition

- **Problem.**
  - We are given an efficient algorithm for finding a lightest path in a directed graph.
  - Can we use this algorithm to find a lightest path in an undirected graph?
  - Yes! Split every edge in the graph into two opposite directed edges with same weight.

Test Your Intuition #2

- **Bad approach** for finding a lightest path.
  - Go over every path from $s$ to $t$, find weight of each path. Keep a path of min weight.

- **What’s wrong** with this approach?
  - In a complete directed graph there are more than $(|V| - 2)!$ paths to check.
  - Running time would be worse than exponential!
Lightest Paths Are Useful

- Algorithms for finding lightest paths have **surprising applications**.
  - (like most main graph algorithms!)
- We will see two examples:
  - **Finding arbitrage**.
  - **Solving systems of linear inequalities**.

### Accessible Vertices

- Given a directed graph $G = (V, E)$, and vertices $s, t$.
  - We say that $t$ is **accessible** from $s$ if there is a directed path from $s$ to $t$. 

### Accessible Vertices Example

- The blue vertices are accessible from $s$. 

### Exchange Rate Table

<table>
<thead>
<tr>
<th>From To</th>
<th>USD</th>
<th>AUD</th>
<th>GBP</th>
<th>EUR</th>
<th>INR</th>
<th>JPY</th>
<th>SGD</th>
<th>CHF</th>
</tr>
</thead>
<tbody>
<tr>
<td>USD</td>
<td>1</td>
<td>1.2910</td>
<td>0.55337</td>
<td>0.80425</td>
<td>43.5000</td>
<td>109.020</td>
<td>1.64790</td>
<td>1.24570</td>
</tr>
<tr>
<td>AUD</td>
<td>0.77720</td>
<td>1</td>
<td>0.42815</td>
<td>0.62225</td>
<td>33.6050</td>
<td>85.0451</td>
<td>1.27498</td>
<td>0.96412</td>
</tr>
<tr>
<td>GBP</td>
<td>1.90710</td>
<td>2.33556</td>
<td>1</td>
<td>1.45335</td>
<td>76.6088</td>
<td>198.636</td>
<td>2.97792</td>
<td>2.25662</td>
</tr>
<tr>
<td>EUR</td>
<td>1.24340</td>
<td>1.60758</td>
<td>0.68096</td>
<td>1</td>
<td>54.0879</td>
<td>136.275</td>
<td>2.04900</td>
<td>3.57263</td>
</tr>
<tr>
<td>INR</td>
<td>0.02990</td>
<td>0.02971</td>
<td>0.01272</td>
<td>0.01849</td>
<td>1</td>
<td>2.5269</td>
<td>0.05788</td>
<td>0.02871</td>
</tr>
<tr>
<td>JPY</td>
<td>0.09910</td>
<td>0.01178</td>
<td>0.03010</td>
<td>0.03732</td>
<td>0.95274</td>
<td>1</td>
<td>0.01494</td>
<td>0.01136</td>
</tr>
<tr>
<td>SGD</td>
<td>0.60600</td>
<td>0.78450</td>
<td>0.33081</td>
<td>0.40084</td>
<td>26.3972</td>
<td>66.7011</td>
<td>1</td>
<td>0.73775</td>
</tr>
<tr>
<td>CHF</td>
<td>0.80083</td>
<td>1.03507</td>
<td>0.44316</td>
<td>0.44407</td>
<td>34.8362</td>
<td>88.0275</td>
<td>1.31949</td>
<td>1</td>
</tr>
</tbody>
</table>

### Example

- For $x_1 - x_2$ is $\leq 0$.
- $x_1 - x_5$ is $\leq -1$.
- $x_2 - x_5$ is $\leq 1$.
- $x_3 - x_1$ is $\leq 5$.
- $x_4 - x_1$ is $\leq 4$.
- $x_4 - x_3$ is $\leq -1$.
- $x_5 - x_3$ is $\leq -3$.
- $x_5 - x_4$ is $\leq -3$. 

---

**Only the blue vertices are accessible from $s$.**
Nonnegative Weights

- We consider the (common) case where the weights are non-negative.
- We first study an algorithm for finding a lightest path from a vertex \( s \in V \) to every other vertex of \( V \).
  - (as BFS does for shortest paths)

Subpaths of a Lightest Path

- Claim.
  - \( G = (V, E) \) – Directed with edge weights.
  - \( P \) – Lightest path from \( s \) to \( t \).
  - Then for any vertex \( v \) in \( P \), the segment of \( P \) between \( s \) and \( v \) is also a lightest path.
Proof of Claim

- Assume for contradiction that there exists \( v \) in \( P \) such that the segment of \( P \) from \( s \) to \( v \) is not a lightest path.
  - \( P_1 \) – portion of \( P \) from \( s \) to \( v \).
  - \( P_2 \) – portion of \( P \) from \( v \) to \( t \).
  - \( w_1, w_2 \) – weights of \( P_1, P_2 \), respectively.

Proof of Claim

- Assume for contradiction that there exists \( v \) in \( P \) such that the segment of \( P \) from \( s \) to \( v \) is not a lightest path.
  - \( P_1 \) – portion of \( P \) from \( s \) to \( v \).
  - \( P_2 \) – portion of \( P \) from \( v \) to \( t \).
  - \( w_1, w_2 \) – weights of \( P_1, P_2 \), respectively.
  - By assumption, exists path \( P'_1 \) from \( s \) to \( v \) of weight \(< w_1 \).
  - Combining \( P'_1 \) and \( P_2 \) yields a path from \( s \) to \( t \) of weight \(< w_1 + w_2 \). Contradiction to \( P \) being a lightest path!
Lightest Paths Tree

• The claim implies: There exists a tree containing a lightest path from \( s \) to every accessible vertex.

• We refer to such a tree as a lightest paths tree from \( s \).

A Lightest Paths Tree of Seattle
Discuss in Groups

- **True** or **False**?
  - There exists a lightest paths tree that is not an MST.
  - There exists an MST that is not a lightest paths tree from any vertex.

Relaxations

- $d[v]$ – weight of the lightest path from $s$ to $v$ that we found **so far**.
- Consider an edge $(u, v)$ in $E$:
  - If $d[v] > d[u] + w(u, v)$,
    then the lightest path we had to $v$ is **heavier than path to** $u + (u, v)$.
Relaxations (cont.)

- Consider an edge \((u, v)\) in \(E\):
  - If \(d[v] > d[u] + w(u, v)\),
    - then the lightest path we found to \(v\) is heavier than path to \(u + (u, v)\).
  - In this case we set \(d[v] = d[u] + w(u, v)\).
  - Such a step is called relaxation of \((u, v)\).

Dijkstra’s Algorithm

- \((d[v] – \text{weight of lightest path from } s \text{ to } v \text{ found so far})\)
- First, set \(d[s] = 0\), and \(d[v] = \infty\) for every \(v \neq s\).
- Grow a lightest paths tree in steps:
  - At each step, out of the vertices that are not in the tree, we add one with a min \(d[u]\).
  - After adding a vertex \(u\) to the tree, relax every edge that leaves \(u\).
Dijkstra Example

Dijkstra Example (cont.)
Run Dijkstra on Your Own

- Run Dijkstra from $s$.

Dijkstra Running Time

- Setting $d[s] = 0$, and $d[v] = \infty$ for every $v \neq s$: $O(|V|).$

- Grow tree in steps:
  - At each step, add vertex with min $d[u]$.
  - After adding $u$, relax every edge $(u, v)$.

- Discuss in groups.
  - Complete the running time analysis.
  - Hint: You also need to choose a data structure.
The Data Structure

- **Recall.** A *priority queue* stores “objects” with priorities.
  - Supports *insertion* of new elements, *removal* of element with max priority, and *priority updates*.
  - When queue contains $n$ elements, removal takes $O(\log n)$, and other operations $O(1)$.
- Our Dijkstra alg uses a priority queue:
  - The *elements* of the queue are the *vertices*.
  - The *priority* of an element $u$ is $d[u]$.

Dijkstra Running Time

- First, adding $|V|$ vertices to queue: $O(|V|)$.
- At each step we remove one vertex from the queue. In total: $O(|V| \log |V|)$.
- After adding vertex $v$ to tree, we perform several relaxations.
  - In total: $O(|E|)$.
- **Dijkstra running time:**
  $$O(|V| \log |V| + |E|).$$
Dijkstra Correctness Intuition

- Dijkstra clearly returns a tree of paths from \( s \) to every accessible vertex.
- **Claim.** The paths chosen by Dijkstra are ordered by increasing weight.
  - At each step, the algorithm chooses the lightest path it is aware of.
  - Every path that is discovered later is at least as heavy as the ones that were previously chosen. (Since every new path is a previously chosen path + additional edges.)

Test Your Intuition

- **Both Prim and Dijkstra:**
  - Gradually grow a tree.
  - Use a priority queue to repeatedly remove a vertex with minimum value.
  - After removing a vertex from the queue, we add it to tree and check the adjacent edges.
- **What is the difference** between the two algorithms?
  - The priority of a vertex is an edge weight in Prim and \( d[\nu] \) in Dijkstra.
Prim VS Dijkstra

- In this graph, both Prim and Dijkstra first check the edges leaving $s$ and get the priority queue:
  - $a$ with priority 6, $b$ with priority 4.
- Then, both add $b$ to the tree and relax the edge from $b$ to $a$.
- What happens next in each algorithm?

Discuss in Groups

- Problem.
  - Directed $G = (V, E)$ with edge weights. Every edge is colored red or blue.
  - Find a lightest path from $s$ to $t$ only out of paths passing through red edges an even number of times (ignore paths passing reds an odd number of times)
  - If a path passes through the same red edge twice, this counts as two red edges.
  - (a hard problem!)
Example: Even Number of Reds

What is the weight of the lightest path between \( s \) and \( t \) with an even number of red edges?

Answer: 20

Solution

- **Split every vertex** \( v \) into vertices \( v_o \) and \( v_e \). Intuition:
  - We would like to **get to** \( v_e \) only after visiting an even number of red edges.
  - Similarly, get to \( v_o \) only after visiting an odd number of red edges.
- **How can we do such a thing?!**
Edges in the New Graph

- A red edge changes parity and a blue does not.
  - The new edges have no color.

Graph Split Example
Completing the Solution

• New graph has $2|V|$ vertices and $2|E|$ edges.
  ◦ Asymptotically the same graph size.
• Run Dijkstra from $s_e$ to $t_e$.

Running time.
  ◦ Creating the new graph: $O(|V| + |E|)$.
  ◦ Running Dijkstra on new graph: $O(|V|\log|V| + |E|)$
  ◦ Total: $O(|V|\log|V| + |E|)$.

Test Your Intuition

• What changes if we want the lightest path with an ODD number of red edges?
  ◦ In the new graph, this is the lightest path from $s_e$ to $t_e$. 
Discuss in Groups

- **Problem.**
  - Directed graph $G = (V, E)$. Edge weights are positive integers.
  - Find a **lightest path** from $s$ to $t$ **only out of the paths having an even weight**.
  - (ignore paths of odd weight)
  - Find an efficient alg for the problem.

Adding Negative Weights

- Is Dijkstra’s algorithm still correct when allowing **negative weights**?
  - No! Dijkstra will find the path of weight 0.
Adding Negative Weights (cont.)

- Is the problem well defined when allowing negative weights?
  - No! What is the weight of the lightest path from $s$ to $t$.

  \[ s \rightarrow -5 \rightarrow t \]

Negative Cycles

- A negative cycle is a cycle whose sum of edge weights is negative.
  - We can use such cycles to get paths with an arbitrarily large negative weight.
  - When there are no negative cycles, the lightest paths problem is well-defined.
The Bellman-Ford Algorithm

- The **Bellman-Ford algorithm** receives a weighted directed graph, possibly with negative weights, and a vertex $s$.
  - If there is a path from $s$ to a **negative cycle**, the alg reports this.
  - Otherwise, alg **finds lightest paths** from $s$ to every accessible vertex.

Recall: Relaxations

- Consider an **edge** $(u, v)$ in $E$:
  - If $d[v] > d[u] + w(u, v)$, then the **lightest path we found to $v$ is heavier than path to $u + (u, v)$**.
  - In this case we set $d[v] = d[u] + w(u, v)$.
  - Such a step is called **relaxation of $(u, v)$**.
The Bellman-Ford Algorithm

- First, set $d[s] = 0$, and $d[v] = \infty$ for every $v \neq s$.
- Repeat $|V| - 1$ times:
  - For every edge $(u, v)$ in $E$ (in arbitrary order):
    - Relax $(u, v)$.

(Writing this as code, we would have a function relax(u,v) and two nested loops.)

Bellman-Ford Example
Run Bellman-Ford

- Run Bellman-Ford from $s$.
- Use the **worst possible edge order**. What happens after each round?
Correctness intuition

• Repeat $|V| - 1$ times:
  ◦ For every edge $(u, v)$ in $E$ (in arbitrary order):
    • Relax $(u, v)$.

• Intuition.
  ◦ After first round of relaxations, we found all of the lightest paths of length one.
  ◦ After second round of relaxations, we found all of the lightest paths of length two.
  ◦ And so on...
  ◦ Longest cycle-free path is of length $|V| - 1$.

Bellman-Ford Running Time

• Repeat $|V| - 1$ times:
  ◦ For every edge $(u, v)$ in $E$ (in arbitrary order):
    • Relax $(u, v)$.

• Initialization of $d[v]$ values: $O(|V|)$.
• A relaxation takes $O(1)$ time.
• We perform $O(|V| \cdot |E|)$ relaxations.
• Total running time: $O(|V| \cdot |E|)$. 
Can We Do Better?

- $O(|V| \cdot |E|)$ is a bad running time.
  - The Bellman-Ford algorithm is from the 1950s.
  - Many people tried to find a faster algorithm, without any success.
  - A major open problem!

Discuss in Groups

- **Problem 1.**
  - Make a small change in Bellman-Ford so that it also checks if there is a negative cycle.

- **Problem 2.**
  - Change our Bellman-Ford algorithm, so that it also saves a lightest path to each vertex.
  - After running the alg, we should be able to obtain a lightest path to any vertex $v$, in a time linear in the length of this path.
  - Hint: There is no need to save $|V| - 1$ full paths.
Arbitrage

- **Problem.** We have \( n \) types of currency and the conversion rates between them.
  - Describe an efficient algorithm that checks whether there exists a series of conversions that starts with 1 unit of a certain currency and ends up with more.

### First Ideas

- \( a_{ij} \) – convergence rate from currency \( c_i \) to currency \( c_j \).
- Build a weighted directed graph:
  - A vertex for every type of currency.
  - An edge from every vertex to every vertex.
  - The edge from \( c_i \) to \( c_j \) is of weight \( a_{ij} \).
- A cycle \( c_{i_1} \rightarrow c_{i_2} \rightarrow c_{i_3} \rightarrow \cdots \rightarrow c_{i_1} \) is of weight \( a_{i_1i_2} + a_{i_2i_3} + \cdots + a_{i_ki_1} \).
  - But we are looking for \( a_{i_1i_2} \cdot a_{i_2i_3} \cdots a_{i_ki_1} > 1 \).
Addressing the Problem

- We are looking for a cycle $c_1 \rightarrow c_2 \rightarrow \ldots \rightarrow c_k \rightarrow c_1$ such that
  \[ a_{i_1i_2} \cdot a_{i_2i_3} \cdots a_{i_ki_1} > 1 \]

\[
\log a_{i_1i_2} + \log a_{i_2i_3} + \cdots + \log a_{i_ki_1} > 0 \\
- \log a_{i_1i_2} - \log a_{i_2i_3} - \cdots - \log a_{i_ki_1} < 0
\]

Example

<table>
<thead>
<tr>
<th></th>
<th>£</th>
<th>$</th>
<th>€</th>
</tr>
</thead>
<tbody>
<tr>
<td>£</td>
<td>1</td>
<td>0.2</td>
<td>0.15</td>
</tr>
<tr>
<td>$</td>
<td>4.5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>€</td>
<td>7</td>
<td>0.5</td>
<td>1</td>
</tr>
</tbody>
</table>
The Full Solution

- **The algorithm.**
  - Build a graph as before, but instead of weights of $a_{ij}$, use $-\log a_{ij}$.
  - Run Bellman-Ford to check whether there is a negative cycle.
  - A negative cycle corresponds to a profitable series of conversions.
  - **What is missing?** Bellman-Ford requires a vertex to start from.
  - We can choose any vertex because the graph is complete.

A Moment of Nonesense

The Flood has receded and the ark is safely aground atop Mount Ararat. Noah tells all the animals to go forth and multiply. Soon the land is teeming with every kind of living creature in abundance, except for snakes. Noah wonders why.

One morning two miserable snakes knock on the door of the ark with a complaint. “You haven’t cut down any trees.” Noah is puzzled, but does as they wish.

Within a month, you can’t walk a step without treading on baby snakes. With difficulty, he tracks down the two parents. “What was all that with the trees?” “Ah,” says one of the snakes, “you didn’t notice which species we are.” Noah still looks blank. “We’re adders, and we can only multiply using logs.”
Discuss in Groups

- We ran Bellman-Ford and the algorithm stated that a negative cycle exists.
- **How can we easily find the negative cycle?**
  - (If there are many negative cycles, we only want to find one.)

![Graph showing a negative cycle](image)

Systems of Linear Equations

- We have:
  - Variables $x_1, x_2, \ldots, x_n$.
  - A set of **linear inequalities** of the form $x_i - x_j \leq b_{ij}$.
    - (where $b_{ij}$ is a number).
- **Problem.**
  - Write an algorithm that checks whether there is a solution to the system.

| $x_1 - x_2 \leq$ 0, |
| $x_1 - x_5 \leq$ -1, |
| $x_2 - x_3 \leq$ 1, |
| $x_3 - x_1 \leq$ 5, |
| $x_4 - x_1 \leq$ 4, |
| $x_4 - x_3 \leq$ -1, |
| $x_5 - x_3 \leq$ -3, |
| $x_5 - x_4 \leq$ -3. |
Examples

• Is there a solution to
  \[ x_1 - x_2 \leq 1, \]
  \[ x_2 - x_3 \leq 2. \]

• Is there a solution to
  \[ x_1 - x_2 \leq -5, \]
  \[ x_2 - x_1 \leq 2. \]

When there is no Solution

• The only way not to have a solution is to have a “cycle”
  \[ x_1 - x_2 \leq b_{12}, \]
  \[ x_2 - x_3 \leq b_{23}, \]
  \[ \ldots \]
  \[ x_{k-1} - x_k \leq b_{k-1,k}, \]
  \[ x_k - x_1 \leq b_{k1}, \]

  where the sum of the \[ b \text{'s} \] is negative.

• Summing up the inequalities gives
  \[ 0 \leq b_{12} + b_{23} + \ldots + b_{k1} < 0. \]
Solution

- Build a graph:
  - **Vertex** for each variable $x_i$.
  - An inequality of the form $x_i - x_j \leq b_{ij}$ leads to an edge from $x_j$ to $x_i$ of weight $b_{ij}$.
  - We need to check if there is a negative cycle.

  $\begin{align*}
x_1 - x_2 & \leq 0, \\
x_1 - x_5 & \leq -1, \\
x_2 - x_5 & \leq 1, \\
x_3 - x_1 & \leq 5, \\
x_4 - x_1 & \leq 4, \\
x_4 - x_3 & \leq -1, \\
x_5 - x_3 & \leq -3, \\
x_5 - x_4 & \leq -3.
\end{align*}$

- **Bellman-Ford** can find if there is a negative cycle.
  - **Issue.** No matter what vertex we start from, the cycle might not be accessible from it!
  - **Add vertex $s$, and edges of weight 0 from $s$ to every other vertex.**
  - **Run Bellman-Ford from $s$.**

  $\begin{align*}
x_1 - x_2 & \leq 0, \\
x_1 - x_5 & \leq -1, \\
x_2 - x_5 & \leq 1, \\
x_3 - x_1 & \leq 5, \\
x_4 - x_1 & \leq 4, \\
x_4 - x_3 & \leq -1, \\
x_5 - x_3 & \leq -3, \\
x_5 - x_4 & \leq -3.
\end{align*}$
Running Time

- $m$ equations in $n$ variables.
- Building the graph: $O(n + m)$.
- Running Bellman-Ford:
  \[ O(|V||E|) = O((n + 1)(m + n)) = O(mn + n^2). \]
- Total running time: $O(mn + n^2)$.

Discuss in Groups

- **Problem.**
  - Improve the running time of the previous problem from $O(mn + n^2)$ to $O(mn)$.
  - Hint: Where in Bellman-Ford does the extra $n^2$ come from? Can you remove this?

- Repeat $|V| - 1$ times:
  - For every edge $(u, v)$ in $E$ (in arbitrary order):
    - Relax $(u, v)$. 
Paths Ending in $v$

• Problem.
  ◦ Directed $G = (V, E)$ with edge weights.
  ◦ There are negative weights in the graph, but **no negative cycles**.
  ◦ For a vertex $v$ in $V$, let $\delta^*(v)$ be the weight of the **lightest path ending in $v$**.
  ◦ How can we efficiently compute $\delta^*(v)$ for one specific $v$?

Example

\[
\begin{align*}
\delta^*(a) &= -2 & \delta^*(d) &= -1 \\
\delta^*(b) &= -2 & \delta^*(e) &= 0 \\
\delta^*(c) &= -3
\end{align*}
\]
Solution

- **Problem.**
  - For a vertex $v$ in $V$, let $\delta^*(v)$ be the weight of the lightest path ending in $v$.

- **Solution.**
  - **Reverse the direction of every edge** of $G$, to obtain a new graph $G^R$.
  - **Run Bellman-Ford from $v$ in $G^R$**.
  - The lightest path starting in $v$ in $G^R$ is the lightest path ending in $v$ in $G$.
  - Running time: $O(|V| \cdot |E|)$.

Discuss in Groups

- **Problem.**
  - We are given a directed graph $G = (V, E)$ with edge weights.
  - There are negative weights in the graph, but **no negative cycles**.
  - For every vertex $v$ in $V$, find the lightest path ending in $v$. 
Solution

- Add a **new vertex** \( s \).
  - Add an **edge from \( s \) to every vertex** in \( V \).
  - Every new edge has **weight 0**.
- Run Bellman-Ford from \( s \).

A lightest path ending in \( v \) is also a lightest path from \( s \) to \( v \).

Running Time

- Revising the graph: \( O(|V|) \).
- Running Bellman-Ford: \( O(|V| \cdot |E|) \).
- Total running time: \( O(|V| \cdot |E|) \).
The End

Warning: Don't let this class affect you too much!

Staring at the ceiling, she asked me what I was thinking about.

I should have made something up.

The Bellman-Ford algorithm makes terrible pillow talk.