Shortest paths with negative arc lengths.

Assume throughout there are no neg. length cycles (algorithms can report such cycles if exist).

So far we've only treated non-negative lengths. Dijkstra's algorithm does not work. In general graphs, the fastest known algorithm is Bellman-Ford.

Recall BF - n iterations. On k'th iteration compute shortest paths using at most k edges by relaxing all edges.

Running time \(O(n \cdot m)\)

We will see today how to do \(O(n \log^2 n)\) for planar graphs.

Remarks: - the fastest known is \(O(n \frac{\log^2 n}{\log \log n})\)

- these results generalize to bounded genus (later if we have time).
Price functions and reduced lengths:

Old idea (best known from Johnson's alg. for APSP in sparse graphs).

A price function is a function $\psi : V \to \mathbb{R}$

the reduced length of arc $uv$ w.r.t. $\psi$ is:

$$\text{len}_\psi(uv) = \text{len}(uv) + \psi(u) - \psi(v)$$

**Lemma:** for any $s$-to-$t$ path $P$,

$$\text{len}_\psi(P) = \text{len}(P) + \psi(s) - \psi(t)$$

$$\text{len}_\psi(P) = \sum_{uv \in P} \text{len}_\psi(uv) = \sum_{uv \in P} \text{len}(uv) + \psi(u) - \psi(v)$$

$$= \text{len}(P) + \psi(s) - \psi(t)$$

Telescope □

**Corollary:** Shortest paths w.r.t. $\text{len}$, $\text{len}_\psi$ are the same.

$\psi$ is called a feasible price function if $\text{len}_\psi(a) \geq 0$ for all arcs $a$. 
Lemma: Let δ(u) denote distances in G from some arbitrary node u. δ(u) is a feasible price function.

Proof: δ(v) ≤ δ(u) + len(uv) (shortest paths inequality)

so \( len(\ uv \) = len(uv) + δ(u) - δ(v) ≥ 0.

So, to compute shortest paths from u in G, suffices to compute shortest paths from any node v and then run Dijkstra.

So far we did not use planarity. Let's combine this approach with small separators.
SP\((G, s)\):

- find a cycle separator \(C \rightarrow G_{\text{in, out}}\)
- pick a node \(r \in C\)
- for \(i \in \text{in, out}\) \(\delta_i := \text{SP}(G_i, r)\)

1. for \(i \in \text{in, out}\) \(A_i := \text{APSP} \) between nodes of \(C\) in \(G_i\) using \(\text{MSSP}\) (use \(\delta_i\) as a feasible price function)

2. \(B := \) distances in \(G\) from \(r\) to nodes of \(C\) using \(\text{BF}\) on the complete graphs defined by \(A_{\text{in}}\) and \(A_{\text{out}}\) called dense dist. graph

3. for \(i \in \text{in, out}\) \(\delta'_i := \text{distances in } G \text{ from } r \text{ to all nodes of } G_i\) using Dijkstra's algorithm initialized with \(B\) and price function \(\delta_i\)

- \(\delta' := \delta'_{\text{in}} \cup \delta'_{\text{out}}\) (distances from \(r\) in \(G\))

4. \(\delta := \text{distances from } s \text{ in } G\) using Dijkstra's algorithm with price function \(\delta'\)

- return \(\delta\)
Analysis:

1. uses 2 calls to MSSP $O(n \log n)$
2. BF in graph with $o(\sqrt{n})$ nodes and $O(n)$ edges $O(\sqrt{n} \cdot n) = o(n^{3/2})$
3. SPF with non-negative lengths $O(n)$

$T(n) \approx 2T(\frac{n}{2}) + o(n^{3/2}) = o(n^{3/2})$
Bottleneck is BF. There are $\sqrt{n}$ iterations.

At the beginning of $k$'th iteration we have an estimate \( d(u) \) for the distance to each node $u$ of $C$. We update the estimate by

$$
\forall u \in C \quad d(u) := \min_{u \in C} \left\{ d(u) + A_{\text{in}}(u,v) \right\}
$$

Equivalently, want to find all column minima of the matrices $A_i^v (u,v) = d(u) + A_i (u,v)$ ($i \in \{\text{in}, \text{out}\}$)

Naively this takes $O(n)$ time (the number of elements in $A$)

using the Monge property we can do it in $\tilde{O}(\sqrt{n})$

We say a matrix $A$ is **Monge** if for $i < j$ and $k < l$ we have

$$
A_{ik} + A_{jl} \leq A_{il} + A_{jk}
$$
in fact, we will only use a weaker property. A matrix $A$ is totally monotone if for $i<j$, $k<l$

$$A_{ik} > A_{il} \implies A_{jk} > A_{jl}$$

clearly, if $A$ is Monge, then both $A$ and $A^t$ are totally monotone.

We will show how to find row minima of a totally monotone $n \times n$ matrix in $O(n)$ time (hence row and column minima of a Monge matrix in $O(n)$ time).

For simplicity will assume minima are unique

**Claim:** the sequence of column indices of row minima of a totally monotone matrix is monotonically increasing.

Suppose $k$ is minimum of row $j$ then $A_{jk} < A_{jl} \quad \forall l > k$, so for $i<j$, $A_{ik} < A_{il} \quad \forall l > k$ so the column index of the min of row $i$ is at most $k$.
Here is how we use this lemma in finding the row minima of a $n \times n$ totally monotone matrix.

First find the row minima of, say, just the even numbered rows. Then deduce in $O(n)$ time the minima of the even numbered rows.

So let's focus on $m \times n$ matrices with $m < n$.

Call an element **dead** if it is not a row minimum.

Lemma: Let $A$ be a TM matrix. Let $k < l$ be column indices.

1. If $A_{ik} \geq A_{il}$ then $\forall j \geq i$ $A_{jk}$ is dead.
2. If $A_{ik} < A_{il}$ then $\forall j \leq i$ $A_{jl}$ is dead.

**Pf:**

1. $A_{ik} \geq A_{il} \implies A_{jk} \geq A_{jl}$
2. Suppose $A_{jk} > A_{jl} \implies A_{ik} \geq A_{il}$, contradiction.

![Diagram](image1)

(1)

![Diagram](image2)

(2)
We use this observation in a procedure that reduces a \( m \times n \) matrix \( (m \leq n) \) into a \( m \times m \) submatrix \( Q \) that include all columns with non-dead elements.

Define the index of \( Q \) to be the largest \( k \) s.t.

\[
\forall j \leq k \; \forall i < j \quad Q_{ij} \text{ is dead}
\]

Note: the index of every \( m \times n \) matrix \( (m \leq n) \) is at least 1 and at most \( m \)

Suppose we know the index of \( Q \) is at least \( k \). Compare \( Q_{kk} \) with \( Q_{kk+1} \)

- if \( Q_{kk} \) < \( Q_{kk+1} \) then by the previous lemma, all elements in column \( k+1 \) at or above \( k \) are dead.
- if \( k = m \) then column \( m+1 \) is entirely dead. Otherwise the index of \( Q \) is at least \( k+1 \).
- if \( Q_{k,k} \geq Q_{k,k+1} \), then by the lemma, all elements of column \( k \) at or below \( k \) are dead so column \( k \) is entirely dead.

The reduction procedure is therefore:

\[
\text{Reduce}(A):
\]
\[
Q := A \\
k := 1 \\
\]
while \( Q \) has more than \( m \) columns:
\[
\text{if } Q_{k,k} < Q_{k,k+1} \\
\text{if } k < n \quad k := k+1 \\
\text{else delete column } m+1 \text{ of } Q \\
\text{else} \\
\text{delete column } k \text{ of } Q \\
\text{if } k > 1 \quad k := k-1 \\
\]
return \( Q \)

Analysis:

\[
\#	ext{ of deletions} \leq n - m \\
\#	ext{ of times } k \text{ is decreased} \leq \#	ext{ of deletions} \leq n - m \\
\#	ext{ of times } k \text{ is increased} \leq m + \#	ext{ of times } k \text{ decreased} \\
\leq m + n - m = n \\
\#	ext{ of steps} \leq \#	ext{ of increases} + \#	ext{ of deletions} \leq 2n - m = \Theta(n)
\]
it is easy to implement each operation in Reduce in constant time, so time for reducing an \( m \times n \) matrix into a \( m \times m \) one is \( O(n) \).

To complete the picture, here is the algo for computing all row minima of a totally monotone \( m \times n \) matrix (\( m \leq n \))

\[
\text{FindMin}(A):
\]
1. if \( A \) has just one row, return the min column
2. \( Q = \text{Reduce}(A) \)
3. \( Q' = \text{submatrix of } Q \text{ induced by even rows} \)
4. \( \text{FindMin}(Q') \)
5. find the row minima of the odd rows of \( Q \)
6. return row minima of \( Q \)

Step 2 takes \( O(n) \) time.
Steps 3,5 take \( O(m) \) time.

Let \( T(m,n) \) denote the running time of \( \text{FindMin} \) on a \( m \times n \) matrix

\[
T(m,n) \leq c_1 n + c_2 m + T\left(\frac{m}{2}, m\right) = O(n)
\]

SMAWK - Aggarwal, Klawe, Moran, Shor, Wilber
Back to the BF step of the shortest paths algorithm. There are $O(n)$ iterations, in each we compute

$$\forall v \in V \quad d(v) := \min_{u \in C} \left\{ \begin{array}{ll}
d(u) + A_{in}(u,v) \\
d(u) + A_{out}(u,v)
\end{array} \right\}$$

that is, find all column minima of $A^i$ defined by

$$A^i_{u,v} = d(u) + A_{in}(u,v)$$

$$d(i) + A_{in}(i,k) + d(j) + A_{in}(j,l) \geq d(i) + A_{in}(i,l) + d(j) + A_{in}(j,k)$$

Monge property!
no munge property

We saw one solution when discussed FR - Dijkstra:
matrix view:

$\Rightarrow$

BF step in
$O(n \log n)$ time

Another solution: adapt SMAWK to triangular matrices. Klawe, Kleitman - $O(n \alpha(n))$

$\Rightarrow$ BF step in $O(n \alpha(n))$ time

Fastest known result for SP uses this. Time $O(n \frac{\log n}{\log \log n})$
Extension to genus $g$:
Running time: $O(g^2 n \log^2 n)$ (roughly)

Can find non-separating cycle $C$ with $O(\sqrt{\frac{n}{g}} \log g)$ nodes in $O(g n \log n)$ time.
Cut the graph open along $C$, duplicating $C$

This reduces the genus by one.
Repeat until graph is planar. It has $2g$ boundary cycles. Each pair of boundary cycles corresponds to some non-separating cycle in the original graph.
Compute $SP$ in the planar graph.
use 2g calls to MSSP to compute all-pair distances between all nodes of all cycles. Set the distances between the two copies of each boundary cycle node to zero. Now run BF with these distances.

For each pair of boundary cycles \((C_1, C_2)\) (possibly \(C_1 = C_2\)), relax the corresponding edges in the dense distance graphs together, using the Monge property.

- Zero length edges (between two copies of a cycle) are few.
- We know how to relax edges between nodes of the same cycle.
- How about edges between two different cycles?

You will show in the problem set how to handle those.
Running time:

2g cycles, each with $O\left(\sqrt{\frac{n}{g}} \log g\right)$ nodes.

Total # of nodes (number of iterations for BF) is $O\left(\sqrt{ng} \log g\right)$

Number of pairs $O(g^2)$

Relaxing all edges between a pair of different cycles takes $O\left(\sqrt{\frac{n}{g}} \log g\right)$ time.

⇒ time for a single iteration of BF:

$O\left(g^2 \cdot \sqrt{\frac{n}{g}} \log g\right)$

⇒ total time for BF:

$O\left(g^2 n \log^2 g\right)$

Time for SP in planar graph is $O(n \log^2 n)$

So overall time is bounded by $O\left(g^2 n \log^2 n\right)$. 