Dijkstra's optimality: Intuition

When \( v \) is deleted from heap, \( v.\text{dist} \) is correct distance.

Why?

(1) Only processed nodes (nodes with \( \text{dist} \leq v.\text{dist} \)) can yield best distance to \( v \). Why?

Since \( v \) is node deleted from heap all non-processed nodes must have \( \text{dist} \geq \text{current } v.\text{dist} \).

(2) Dijkstra's has considered all processed nodes before setting \( v.\text{dist} \).

Question: Does Dijkstra's work if there are negative edge weights? (Assume no neg. cycles)

Which property breaks? (1)

- processed
- unprocessed

First: Applications of neg. edge weights?
Dijkstra's correctness

Proof by induction on the number of processed nodes.

\[ N = \text{set of processed nodes (deleted from heap)}. \]

Notation:
\[ d(u) = \text{final } s-u \text{ distance assigned by Dijkstra's} \]
\[ d^*(u) = \text{optimal } s-u \text{ distance} \]
\[ s \rightarrow u = \text{shortest } s-u \text{ path} \]
\[ l(x) = \text{length of } x, \text{ } x \text{ may be partly edge} \]

Then for \( |N| = 1 \) to \( |N| \), \( d(u) = d^*(u) \) for all \( u \) in \( N \).

Base Case: \( |N| = 1 \), \( N = \{s\} \).
\[ d(s) = d^*(s) = 0 \checkmark \]

Inductive Hypothesis:
For \( |N| = k > 1 \), \( d(u) = d^*(u) \) for all \( u \) in \( N \).

Show true for \( |N| = k+1 \)

\( v \) next \((k+1)^{th}\) node added to \( N \).
Show that when \( \Diamond \) added to \( N \) (deleted from heap), \( d(v) = d^*(v) \)

How does \( \Diamond \) get added to \( N \)?

Some node \( \odot \) in \( N \) was processed, so edges out of \( \odot \) were relaxed, \( \Diamond \) had next smallest distance in heap.

\((uv) \): edge used to add \( \Diamond \) to \( N \)

Claim: Optimal path from \( s \) to \( v \) is \( s \sim u \circ (uv) \)

So \( d(v) = l(s \sim u) + l(u,v) = d^*(v) \).

Suppose there is another \( s-v \) path \( P \) s.t. \( l(P) < d(V) \)

Main idea: if \( (uv) \) is the edge that added \( \Diamond \) to \( N \), optimal \( s-v \) path must go through \( \odot \)

Note that \( P \) must contain at least one node in \( N \).

Let \( (x,y) \) denote first edge in \( P \) that leaves \( N \).
Note: \( v \)'s distance gets updated through an edge \((x,y)\) where \( x \) must be in \( N \). So \( x \) is that node.

Will show that \( l(P) \geq d(v) \)

\[
l(P) \geq l(s \Rightarrow x) + l(x,y) \\
= d(x) + l(x,y) \quad \text{by ind. hyp.} \\
\geq d(y) + l(x,y) \quad \text{defn of } d(y) \\
\geq d(v). \quad \text{Since Dijkstra's chose } v \text{ before } x.
\]

Note that \( x \) may = \( s \) and \( y \) may = \( v \).

\((x,y) = (s,v)\)

In this case, clearly \( l(P) \geq d(v) \) since \((w,v)\) and not \((s,v)\) is the edge that gave \( v \) its final distance.

Question: Does Dijkstra's work for graphs with negative edge weights?

Where does proof break?

Base Case: \(-2\) \(d(s) = 0\)

\(d^*(s) = -1\)
Inductive Step:

\[ d(p) = d(s \sim x) + d(x, y) \]

An unprocessed/unknown vertex (ex. s) can yield a shorter path to a processed/known vertex (ex. v).

One idea: Find the smallest negative number and add this to all edge weights.

Does this work?

Before algorithm for negative weights, applications of negative weights?

Any application where an entity can be gained and lost.

Chemistry - heat gain/loss in chemical reaction
- nodes: compounds
- edge \((u,v)\): if compound \(u\) can be reduced to compound \(v\)
- weights: amount of heat gain/loss

Finance - cost of a financial transaction
- nodes: financial state (e.g. bought stock, sold shares)
- edge \((u,v)\): if can get from state \(u\) to \(v\)
- weight: cost incurred/refunded
Actually in Dijkstra's updating occurs per edge.

Can replace steps 9-11 with:

```
update (v, u) // (v, u) ∈ E
if (!u.processed and v.dist + w_{v,u} < u.dist)
    u.dist = v.dist + w_{v,u}
    u.pred = v.
```

Specifically, we update edges out of the most recently processed node (v). Since we know that v's distance cannot be improved.

What is different when there are negative weights?
Most recently processed node is irrelevant since it's distance may be updated later.

With negative edge weights:
So we should update not just edges out of most recent processed node but all edges.

Order doesn't matter, so just use alphabetic order (a, b), (a, c), (b, c), ...

But how many times to update all edges?
After one update, which node's distance is correct? Remember alphabetical order: (a, c) (b, a) (s, a) (s, b) (s, c)  

Final dist's should be:  
- s: dist = 0  
- b: dist = 1  
- a: dist = -3  
- c: dist = -2  

1st: b dist correct  
2nd: a dist correct  
3rd: c dist correct  

After i updates, all nodes whose shortest paths have i edges have correct distance.

How many times should we update to ensure we get shortest distances for all possible nodes?

Max # of edges in a path: |V| - 1

This is actually more than we need \(\Rightarrow\) HW gives:

A finds shortest path distances from s to every other node in \(G^+\)/Bellman Ford (s, v)  
\(v = (v, E), (u, v) \in E, w_{uv} \text{ may be } < 0, s \neq v\)
1. For all \(v \in V\), \(v\).dist = \(\infty\), \(v\).pred = null (no more \(v\).processed!)
2. s.dist = 0
3. For \(i = 1\) to \(|V| - 1\)
   //Check all edges (to see if end vertex can be updated)
   4. For (each edge \((v, u) \in E\))
   5. \(\text{Update } (v, u)) \)
Bellman Ford (G, s)

A \rightarrow (5) \rightarrow \text{M} \rightarrow 3 \rightarrow \text{N}

IVI - 1 = 5 - 1 = 4

Alphabetic order:
- (a, b), (a, c), (a, e)
- (c, a)
- (e, b), (e, c)

\$ d_{ib} = 75 \$
- (s, a), (s, e)

1st check: \( a = 9 \), \( e = 6 \)
2nd: \( b = 13, 7 \), \( c = 14, 3 \)
3rd: \( a = 1 \)
4th: \( b = 5 \)

What could go wrong?

What:

\( \begin{align*}
9 & \rightarrow (a) \\
5 & \rightarrow (c) \\
\end{align*} \)

Negative cycle!

Cannot find shortest path!

How to detect at the end of algorithm?

If any edge \((v, u)\) can improve \( u.dist \), then there must
d.dist can be improved to 12

\[
\begin{align*}
\text{Add to code:} & \\
\text{C. For each edge } (v, u) \in E: & \\
\quad \text{if } (u, \text{dist} > v, \text{dist} + w_{v, u}) & \\
\quad \quad \quad \quad \quad \quad \text{print("Error! Negative cycle!");}
\end{align*}
\]

Run Time: (main loop): 
- Check each edge \( |V| - 1 \) times
- \( \Rightarrow O(|V| \cdot |E|) \) 
  - Note: Much slower than Dijkstra's.

Correctness: \<SKIP>\ 

Proof: by induction on distance found using some \# of edges.

Let \( d_i(v) \): S-u distance using \( \leq i\) edges found by BF

Claim: For all \( i \leq |V| - 1 \), \( d_i(v) \) is optimal distance using \( \leq i \) edges.

Base Case: \( i = 1 \) \( d_1(v) \) is best distance using \( \leq 1 \) edges. \checkmark

Ind. Hyp: \( d_k(v) \) is optimal for all \( u \in V \)

Ind. Step: Show \( d_{k+1}(v) \) is optimal.