SINGLE-SOURCE SHORTEST PATHS
Part II of the course:
optimization problems on graphs

single-source shortest paths
   Find shortest path from source vertex to all other vertices

all-pairs shortest paths
   Find shortest paths between all pairs of vertices

maximum flow
   Find maximum flow from source vertex to target vertex

maximum bipartite matching
   Find maximum number of disjoint pairs of vertices with edge between them
Graph terminology

graph $G = (V,E)$

- $V =$ set of vertices (or: nodes)
- $E =$ set of edges (or: arcs)
- (edge-)weighted graph: every edge $(u,v)$ has a weight $w(u,v)$
- directed graph vs. undirected graph
- paths and cycles
Representation of graphs, I

adjacency-matrix representation

\[ M[i,j] = \begin{cases} 
0 & \text{if } i = j \\
w(v_i, v_j) & \text{if } i \neq j \text{ and edge } (v_i, v_j) \text{ exists; } w(v_i, v_j) \text{ is weight of } (v_i, v_j) \\
\infty & \text{if edge } (v_i, v_j) \text{ does not exist}
\end{cases} \]
Representation of graphs, II

adjacency-list representation

there is an edge from $v_1$ to $v_6$ with weight 2

today we assume adjacency-list representation

weighted, directed graph
Shortest paths

weighted, directed graph $G = (V, E)$

- weight (or: length) of a path = sum of edge weights
- $\delta (u, v)$
  = distance from $u$ to $v$
  = min weight of any path from $u$ to $v$
- shortest path from $u$ to $v$
  = any path from $u$ to $v$ of weight $\delta (u, v)$

Is $\delta (u, v)$ always well defined?

No, not if there are negative-weight cycles.

If $\delta (u, v)$ well defined, then there is a shortest path with at most $|V| - 1$ edges
Shortest-paths problem

**single-pair**
- find shortest path for single pair $u,v$ of vertices
- can be solved using single-source problem with source $= u$ and no better solutions are known

**single-source**
- find shortest path from given source to all other vertices
  - this week

**single-destination**
- find shortest paths to given destination from all other vertices
  - same as single-source for undirected graphs, for directed graphs just reverse edge directions

**all-pairs**
- find shortest paths between all pairs of vertices
  - can be solved by running single-source for each vertex, but better solutions are known: next week
The Single-Source Shortest-Paths Problem

Input:
- graph \( G = (V,E) \)
- source vertex \( s \in V \)

Output
- for each \( v \in V \): distance \( \delta (s,v) \)
  store distance in attribute \( d(v) \)
  if \( v \) not reachable: \( \delta (s,v) = \infty \)
- a shortest-path tree that encodes all shortest paths from \( s \):
  for \( v \neq s \) the predecessor \( \pi(v) \) on a shortest path from \( s \) to \( v \)

NB if negative-weight cycle is reachable from \( s \), then we only need to report this fact

book (3rd ed) uses \( v.d \) and \( v.\pi \) instead of \( d(v) \) and \( \pi(v) \)
The SSSP problem on unweighted graphs

**Breadth-First Search (BFS)**

running time: \( \Theta(|V| + |E|) \)

round \( i \):

**for** each vertex \( u \) at distance \( i-1 \) from \( s \) (that is, with \( d(v)=i-1 \))

**do for** all neighbors \( v \) of \( u \)

**do if** \( v \) unvisited **then** set \( d(v)=i \)

implementation: maintain queue \( Q \)

initial part of \( Q \): nodes with \( d(v)=i-1 \) that have not been handled yet

rest of \( Q \): nodes with \( d(v)=i \) with a neighbor that has been handled
The SSSP problem on weighted graphs

Dijkstra’s algorithm
- works only for non-negative edge weights
- running time $\Theta (|V| \log |V| + |E|)$

Bellman-Ford algorithm
- can handle negative edge weights, works even for negative-weight cycles
- running time $\Theta (|V| \cdot |E|)$

Special case: directed acyclic graph (DAG)
- can handle negative edge weights
- running time $\Theta (|V| + |E|)$
Framework for solving SSSP problems

For each vertex \( v \), maintain value \( d(v) \) that is current “estimate” of \( \delta(s, v) \) and pointer \( \pi(v) \) to predecessor in current shortest-path tree

- **Invariant:** \( d(v) \geq \delta(s, v) \)
  - if \( d(v) < \infty \): there exists \( s \)-to-\( v \) path of weight \( d(v) \) in which \( \pi(v) \) is predecessor of \( v \)

- at end of algorithm: \( d(v) = \delta(s, v) \)
  - and so pointers \( \pi(v) \) encode shortest-path tree

```
Initialize-Single-Source \((G, s)\)
1. \textbf{for} each vertex \( v \) do \( d(v) \leftarrow \infty \); \( \pi(v) \leftarrow \text{NIL} \)
2. \( d(s) \leftarrow 0 \)
```
Framework for solving SSSP problems

For each vertex \( v \), maintain value \( d(v) \) that is current “estimate” of \( \delta(s,v) \) and pointer \( \pi(v) \) to predecessor in current shortest-path tree

- Invariant: \( d(v) \geq \delta(s,v) \)
  - if \( d(v) < \infty \): there exists s-to-v path of weight \( d(v) \) in which \( \pi(v) \) is predecessor of \( v \)
- at end of algorithm: \( d(v) = \delta(s,v) \)
  - and so pointers \( \pi(v) \) encode shortest-path tree

Estimates \( d(v) \) are updated using relaxation:

\[
\text{Relax}(u,v) \quad // (u,v) \text{ is an edge}
\]

1. if \( d(u) + w(u,v) < d(v) \)
2. then \( d(v) \leftarrow d(u) + w(u,v) \)
   \( \pi(v) \leftarrow u \)

(if \( d(u) = \infty \) then nothing happens)
All algorithms we discuss today use this framework

- algorithms differ in how they determine which edge to relax next
- NB: a single edge can be relaxed multiple times

Lemma (path-relaxation property):
Let $s,v_1,v_2,...,v_k$ be a shortest path. Suppose we relax the edges of the path in order, with possibly other relax-operations in between:

$$\ldots, \text{Relax}(s,v_1), \ldots, \text{Relax}(v_1,v_2), \ldots, \text{Relax}(v_2,v_3), \ldots, \ldots, \ldots, \ldots, \ldots, \ldots, \text{Relax}(v_{k-1},v_k), \ldots$$

Then after these relax-operations we have $d(v_k) = \delta(s,v_k)$.

Proof: Induction on $k$. Recall that we always have $d(v_k) \geq \delta(s,v_k)$.

- base case, $k=1$: after Relax($s,v_1$) we have $d(v_1) \leq d(s) + w(s,v_1) = \delta(s,v_1)$
- $k>1$: just before Relax($v_{k-1},v_k$) we have $d(v_{k-1}) = \delta(s,v_{k-1})$ by induction.
  after Relax($v_{k-1},v_k$) we thus have $d(v_k) \leq d(v_{k-1}) + w(v_{k-1},v_k)$
  \[ = \delta(s,v_{k-1}) + w(v_{k-1},v_k) \]
  \[ = \delta(s,v_k) \]
Dijkstra's algorithm

Input:
- weighted graph $G(V,E)$ with non-negative edge weights
- source node $s$

Required output:
- for each vertex $v$:
  - value $d(v) = \delta(s,v)$
  - pointer $\pi(v)$ to parent in shortest-path tree
Let’s see if we can adapt BFS

round $i$:

for each vertex $u$ at distance $i-1$ from $s$ (that is, with $d(v)=i-1$)

  do for all neighbors $v$ of $u$

    do if $v$ unvisited then set $d(v)=i$  \ this is a Relax-operation

when edges have weights, distances can “jump” and we cannot handle vertices in rounds as in BFS …

… but we can still try to handle vertices in order of increasing distance to $s$
Dijkstra’s algorithm
- handle vertices in increasing distance to $s$
- handling a vertex = relax all outgoing edges

Implementation: maintain priority queue $Q$
- $Q$ contains vertices that have not been handled yet; key = $d(v)$
Dijkstra \((G,s)\)

// \(G = (V,E)\) is a weighted directed graph with no negative edge weights

1. Initialize-Single-Source \((G,s)\)
2. \(S \leftarrow \) empty set // \(S = \) set of vertices that have been handled
3. Put all vertices \(v\) in \(V\) in min-priority queue \(Q\), with \(d(v)\) as key
4. \textbf{while} \(Q\) not empty
5. \hspace{1em} \textbf{do} \(u \leftarrow \text{Extract-Min}(Q)\)
6. \hspace{2em} \(S \leftarrow S \cup \{u\}\)
7. \hspace{2em} \textbf{for} each outgoing edge \((u,v)\)
8. \hspace{3em} \textbf{do} \(\text{Relax}(u,v)\)

if \(d(v)\) changes, we have to update \(Q\):

\begin{align*}
\text{Relax}(u,v) & \quad // (u,v) \text{ is an edge} \\
1. \quad \textbf{if} & \quad d(u) + w(u,v) < d(v) \\
2. & \quad \textbf{then} \quad d(v) \leftarrow d(u) + w(u,v) \\
3. & \quad \pi(v) \leftarrow u \\
4. & \quad \text{Decrease-Key}(Q,v,d(v)) \\
\end{align*}

change key of \(v\) to (new, lower value of) \(d(v)\)
Dijkstra (G,s)

// G = (V,E) is a weighted directed graph with no negative edge weights
1. Initialize-Single-Source ( G,s )
2. S ← empty set // S = set of vertices that have been handled
3. Put all vertices v in V in min-priority queue Q, with d(v) as key
4. while Q not empty
5. do u ← Extract-Min(Q)
6. S ← S U { u }
7. for each outgoing edge (u,v)
8. do Relax(u,v)

Invariant: at start of each iteration of while-loop: d(v) = δ(s,v) for all v in S.

Initialization: S ← empty set, so Invariant holds before first iteration
Invariant: at start of each iteration of while-loop: \( d(v) = \delta(s,v) \) for all \( v \) in \( S \).

Maintenance: must prove \( d(u) = \delta(s,u) \) for selected vertex \( u \)

- \( u \) was chosen
- if \( y = u \): okay
- otherwise: \( d(u) \leq d(y) = \delta(s,y) \leq \delta(s,u) \)

\( d(u) \geq \delta(s,u) \) by property of Relax

\( \Rightarrow (x,y) \) has been relaxed

\( \Rightarrow d(y) = \delta(s,y) \)

no negative edge-weights

first vertex on shortest path to \( u \) that has not yet been handled

\( x \) has been handled

\( \Rightarrow d(y) = \delta(s,y) \)
Invariant: at start of each iteration of while-loop: \( d(v) = \delta(s,v) \) for all \( v \) in \( S \).

Termination:

- initially \( Q \) contains \(|V|\) vertices
- at every iteration size of \( Q \) decreases by one
- termination after \(|V|\) iterations

- at end of algorithm: \( Q \) is empty
- we always have \( Q = V - S \)
- Invariant: \( d(v) = \delta(s,v) \) for all \( v \) in \( S \)
- algorithm is correct
Dijkstra \((G,s)\)

// \(G = (V,E)\) is a weighted directed graph with no negative edge weights

1. Initialize Single-Source \((G,s)\)
2. \(S \leftarrow \) empty set // \(S = \) set of vertices that have been handled
3. Put all vertices \(v\) in \(V\) in min-priority queue \(Q\), with \(d(v)\) as key
4. while \(Q\) not empty
5. do \(u \leftarrow\) Extract-Min\((Q)\)
6. \(S \leftarrow S \cup \{u\}\)
7. for each outgoing edge \((u,v)\)
8. do Relax\((u,v)\)

Running time:

- number of Extract-Min operations = \(|V|\)
- number of Relax-operations = \(|E|\)
  
  Relax involves Decrease-Key-operation

normal heap: Extract-Min and Decrease-Key both \(O(\log n)\) \(\text{O}(V+E) \log V\)

Fibonacci heap: Extract-Min \(O(\log n)\), Decrease-Key \(O(1)\) on average

\(O(V \log V + E)\)
Theorem: Dijkstra’s algorithm solves the Single-Source Shortest-Path Problem for graphs with non-negative edge weights in $O(|V| \log |V| + |E|)$ time.
Bellman-Ford algorithm

Input:
- weighted graph $G(V,E)$, can have negative edge weights
- source node $s$

Required output:
- for each vertex $v$:
  - value $d(v) = \delta(s,v)$
  - pointer $\pi(v)$ to parent in shortest-path tree
- or report there is a negative-weight cycle reachable from $s$
Recall path-relaxation property

Lemma:
Let $s,v_1,v_2,...,v_k$ be a shortest path. Suppose we relax the edges of the path in order, with possibly other relax-operations in between:

$$..., \text{Relax}(s,v_1), ..., \text{Relax}(v_1,v_2), ..., \text{Relax}(v_2,v_3), ...., ..., ..., \text{Relax}(v_{k-1},v_k), ...$$

Then after these relax-operations we have $d(v_k) = \delta(s,v_k)$.

Problem with negative weights: we don’t know in which order to relax edges

Idea of Bellman-Ford algorithm
- after relaxing all edges we have relaxed the first edge on any shortest path
- after relaxing all edges once more, we have relaxed the second edge
- etc.
Bellman-Ford \((G, s)\)

// \(G = (V,E)\) is a weighted directed graph; edge weights may be negative

1. Initialize-Single-Source \((G, s)\)
2. \(\text{for } i \leftarrow 1 \text{ to } |V| - 1\)
3. \(\text{do for each edge } (u, v) \text{ in } E\)
4. \(\text{do Relax}(u, v)\)

// Check for negative length cycles:
5. \(\text{for each edge } (u, v) \text{ in } E\)
6. \(\text{do if } d(u) + w(u, v) < d(v)\) // there is an edge that could still be relaxed
7. \(\text{then report there is a negative-weight cycle reachable from } s\)

\(d(s) = 0\)

\[
\begin{array}{c}
\text{Node} \\
-1 \\
0 \\
2 \\
1
\end{array}
\]

\[
\begin{array}{c}
w(e_1) = 1 \\
w(e_2) = -3 \\
w(e_4) = 2 \\
w(e_5) = 4 \\
w(e_3) = 1
\end{array}
\]
Bellman-Ford \((G,s)\)

// \(G = (V,E)\) is a weighted directed graph; edge weights may be negative

1. Initialize-Single-Source (\(G,s\))
2. \(\text{for } i \leftarrow 1 \text{ to } |V| - 1\)
3. \(\text{do for } \text{each edge } (u,v) \in E\)
4. \(\text{do Relax}(u,v)\)

   // Check for negative length cycles:
5. \(\text{for each edge } (u,v) \in E\)
6. \(\text{do if } d(u) + w(u,v) < d(v) \quad \text{// there is an edge that could still be relaxed}\)
7. \(\text{then report there is a negative-weight cycle reachable from } s\)

Correctness, in case of no reachable negative-weight cycles:

Invariant of the loop in lines 2 – 4 (follows from path-relaxation property):

for each \(v\) with shortest path from \(s\) consisting of \(\leq i\) edges, we have \(d(v) = \delta(s,v)\)

So after the loop, we have: \(d(v) = \delta(s,v)\) for all \(v\) in \(V\) and algorithm will not report a negative-weight cycle in line 7
Correctness in case of reachable negative-weight cycle:

$\mathbf{v_1, v_2, \ldots, v_k, v_1}$: reachable negative-weight cycle

Assume for contradiction that $d(v_i) + w(v_i, v_{i+1}) \geq d(v_{i+1})$ for all $i=1, \ldots, k$

Then:

$\begin{align*}
    d(v_1) + w(v_1, v_2) & \geq d(v_2) \\
    d(v_2) + w(v_2, v_3) & \geq d(v_3) \\
    & \vdots \\
    d(v_{k-1}) + w(v_{k-1}, v_k) & \geq d(v_k) \\
    d(v_k) + w(v_k, v_1) & \geq d(v_1)
\end{align*}$

$+ \quad \sum_{1 \leq i \leq k} d(v_i) + \left( \sum_{1 \leq i \leq k} w(v_i, v_{i+1}) + w(v_k, v_1) \right) \geq \sum_{1 \leq i \leq k} d(v_i)$

So $\sum_{1 \leq i \leq k} w(v_i, v_{i+1}) + w(v_k, v_1) \geq 0$, contradicting that cycle has negative weight.
Bellman-Ford \((G,s)\)

\[
\text{\texttt{// } G = (V,E) \text{ is a weighted directed graph; edge weights may be negative}}
\]

1. Initialize-Single-Source \(( G,s )\)
2. \textbf{for } \(i \leftarrow 1 \) \textbf{ to } |V| - 1
3. \textbf{do for } each edge \((u,v)\) in \(E\)
4. \textbf{do } Relax\((u,v)\)

\[
\text{\texttt{// Check for negative length cycles:}}
\]

5. \textbf{for } each edge \((u,v)\) in \(E\)
6. \textbf{do if } \(d(u) + w(u,v) < d(v)\) \text{ // there is an edge that could still be relaxed}
7. \textbf{then } report there is a negative-weight cycle reachable from \(s\)

Running time: \(O( |V| \cdot |E| )\)
Theorem: The Bellman-Ford algorithm solves the Single-Source Shortest-Path Problem (even in case of negative-weight cycles) in $O(|V| \cdot |E|)$ time.
The SSSP problem on weighted graphs: summary

Bellman-Ford algorithm
- can handle negative edge weights, works even for negative-weight cycles
- running time $\Theta(|V| \cdot |E|)$

Dijkstra’s algorithm
- works only for non-negative edge weights
- running time $\Theta(|V| \log |V| + |E|)$

Special case: directed acyclic graph (DAG): see the book
- can handle negative edge weights
- running time $\Theta(|V| + |E|)$

Next week: the All-Pairs Shortest-Path Problem