Graph Algorithms

- Sets and sequences can only model limited relations between objects, e.g. ordering, overlapping, etc.
- Graphs can model more involved relationships, e.g. road and rail networks
- Graph: $G = (V, E)$, $V$ : set of vertices, $E$ : set of edges
  - Directed graph: an edge is an ordered pair of vertices, $(v_1, v_2)$
  - Undirected graph; an edge is an unordered pair of vertices $\{v_1, v_2\}$
Graph representation

**Adjacency matrix**

Directed graph

\[ V = \{1, 2, 3, 4\} \]

\[ E = \{(1, 2), (1, 3), (1, 4), (2, 3), (3, 4), (4, 2)\} \]

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 \\
1 & 0 & 1 & 1 & 1 \\
2 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 1 \\
4 & 0 & 1 & 0 & 0 \\
\end{array}
\]
Undirected graph

\[ V = \{1, 2, 3, 4\} \]
\[ E = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\} \]

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 \\
1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
3 & 1 & 1 & 0 & 1 & 3 & 0 & 1 & 3 & 0 & 1 \\
4 & 1 & 1 & 1 & 0 & 4 & 0 & 0 & 4 & 0 & 0 \\
\end{array}
\]

Advantage: \(O(1)\) time to check connection.
Disadvantages:

- Space is \(O(|V|^2)\) instead of \(O(|E|)\)
- Finding who a vertex (node) is connected to requires \(O(|V|)\) operations
Adjacency List

Example:

```
2 4 3
1 2 3 4
```

Example:

```
1
2
3
4
```

```
1 2 3 4
2 1 3 4
3 1 2 4
4 1 2 3
```
Advantages:
- easy to access all vertices connected to one vertex
- space is $O(|E| + |V|)$

Disadvantage:
- testing connection in worst case is $O(|V|)$
- space: $|V|$ header, $2|E|$ list nodes $\Rightarrow O(|V| + |E|)$. There might be $|V|^2$ edges ($|E| = |V|^2$) but probably not.
Another representation

Adjacency list with arrays

- For node $i$, use $\text{header}[i]$ and $\text{header}[i + 1] - 1$ as the indices in the list array.
  If $\text{header}[i] > \text{header}[i + 1] - 1$ vertex $i$ is not connected to any node.

- same advantage as adjacency list but save space

- binary search is possible to determine the connection: $O(\log|V|)$

- problem: difficult to update the structure
Traversal of a graph

**Depth First and Breadth First**

**Depth First** (most useful)

var visited[1...|V|]: boolean ← false

Proc DFS(v);
(Given a graph $G = (V, E)$ and a vertex $v$, visit each vertex reachable from $v$)

visited[v] ← true
perform prework on vertex $v$
For each vertex $w$ adjacent to $v$ do
  if not visited[w] then
    DFS(w)
    perform postwork on edge $(v, w)$
(sometimes we perform postwork on all edges out of $v$)

- given a vertex $v$, we need to know all vertices connected to $v$
- stack space $\approx |V| - 1$
Complexity

1) With adjacency list
visited each vertex once
visited each edge twice; once from $v$ to $w$, once from $w$ to $v$.

$O(|V| + |E|)$

2) With adjacency matrix
visited each vertex once
for each vertex, visit all vertices connected to this vertex needs $O(|V|)$ steps

$O(|V|^2)$

Note: In graph, $O(|E|)$ is better than $O(|V|^2)$ in most cases.
1) DFS numbering
Initially $DFS_{num} := 1$
Use DFS with following *prework*
prework
\[ v.DFS := DFS_{num}; \]
\[ DFS_{num} := DFS_{num} + 1; \]

2) DFS tree
Use DFS with following *postwork*
postwork:
\[ \text{add edge } (v, w) \text{ to } T \]
**Topological Sorting**

**Task scheduling**

- A set of tasks. Some tasks depend on other tasks
- Task $a$ depends on task $b$ means that task $a$ cannot be started until task $b$ is finished
- We want to find a schedule for tasks consistent with dependencies

Example: $x \rightarrow y$: $y$ cannot start until $x$ is completed.

A B C E D   A B C D E   A B E C D

are all schedule for tasks $\{A, B, C, D, E\}$.

This graph must be acyclic!
The problem

Given a directed acyclic graph $G = (V, E)$ with $n$ vertices, label the vertices from 1 to $n$ such that, if $v$ is labelled $k$, then all vertices that can be reached from $v$ by a directed path are labelled with labels $> k$.

In other words, label vertices from 1 to $n$ such that for any edge $(v, w)$ the label of $v$ is less than the label of $w$.

**Lemma.** A directed acyclic graph always contains a vertex with in-degree 0.

**Proof.** If all vertices have positive in-degrees, starting from any vertex $v$, traverse the graph ”backward”. We never have to stop. But we only have a finite number of vertices! Consequently, there must be a cycle in the graph – a contradiction! (pigeonhole principle).
Algorithm:

By induction:
find one vertex with in-degree 0. Label this vertex 1, and delete all edges from this vertex to other vertices.
Now the new graph is also acyclic and is of size $n - 1$. By induction we know how to label it.

Implementation.
1. Initialize in-degree of all vertices
2. Put all vertices with 0 in-degree into a queue or stack
   \[ l \leftarrow 0 \]
3. dequeue \( v \); \( l \leftarrow l + 1 \); \( v.label \leftarrow l \);
   for all edge \( (v, w) \)
     decrease in-degree of \( w \) by 1
     if degree of \( w \) is now 0 enqueue \( w \)
until queue is empty

\textit{Time: } \( O(|E| + |V|) \)
Single-Source Shortest-Paths

- Weighted graph
  \( G = (V, E) \) directed graph with weights associated with the edges
- The weight of an edge \((u, v)\) is \(w(u, v)\).
  The weight of a path \(p = < v_0, v_1, \cdots v_k >\) is the summation of the weights of its edges
  \[
  w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i).
  \]
- We define the shortest-path weight from \(u\) to \(v\) by
  \[
  \delta(u, v) = \begin{cases} 
  \min \{w(p) : p \text{ is a path from } u \text{ to } v\} \\
  \infty \text{ if there is no path from } u \text{ to } v
  \end{cases}
  \]
• The shortest path from $u$ to $v$ is defined as any path $p$ from $u$ to $v$ with weight $w(p) = \delta(u, v)$.

• The problem: Given the directed graph $G = (V, E)$ and a vertex $s$, find the shortest paths from $s$ to all other vertices.

• For undirected graphs, change edge $\{u, v\}$ with weight $w$ to a pair of edges $(u, v)$ and $(v, u)$ both with weight $w$.

Example:
• Negative weight cycle
  In some instances of the single-source shortest-paths problem, there may be edges with negative weights.

  † If there is no negative cycle, the shortest path weight \( \delta(s, v) \) is still well defined.
  † If there is negative cycle reachable from \( s \), then the shortest path weight from \( s \) to any vertex on the cycle is not well defined.
  † A lesser path can always be found by following the proposed ”shortest path” and then traverse the negative weight cycle.

• Cycles in shortest path?
  † A shortest path cannot contain a negative cycle.
    Shortest path weight is not well defined.
  † A shortest path cannot contain a positive cycle.
    Removing the positive cycle will produce a path with lesser weight.
  † How about 0-weight cycle?
    We can remove all 0-weight cycles and produce a shortest path without cycle.

• We can assume that shortest paths we are looking for contain no cycle.
  Therefore any shortest path contains at most \(|V| - 1\) edges.
For each vertex $v$, we maintain two attributes, $\pi[v]$ and $d[v]$.

- $d[v]$ is an upper bound on the weight of a shortest path from source $s$ to $v$.
  † During the execution of a shortest-path algorithm, $d[v]$ may be larger than the shortest-path weight.
  † At the termination of a shortest-path algorithm, $d[v]$ is the shortest-path weight from $s$ to $v$.

- $\pi[v]$ is used to represent the shortest paths.
  † During the execution of a shortest-path algorithm, $\pi[]$ need not indicate shortest paths.
  † $\pi[v]$ is the last edge of a path from $s$ to $v$ during the execution of a shortest-path algorithm.
  † At the termination of a shortest-path algorithm, $\pi[v]$ represent the last edge of a shortest path from $s$ to $v$.
  † Since sub-path of a shortest path is itself shortest path, therefore $
\langle v, \pi[v], \pi[\pi[v]], \cdots, s \rangle$ is the shortest path from $s$ to $v$ in reverse order.
• Initialization

\texttt{Initialize\_Single\_Source}(G, s)

1 For each vertex \( v \in V[G] \) do
2 \( d[v] = \infty; \)
3 \( \pi[v] = nil; \)
4 \( d[s] = 0; \)

• Relaxation

\texttt{Relax}(u, v, w)

1 if \( d[v] > d[u] + w(u, v) \) then
2 \( d[v] = d[u] + w(u, v); \)
3 \( \pi[v] = u; \)

\texttt{Relax}(u, v, w) \) tests if we can improve the shortest path to \( v \) found so far by going through \( u \).

If so, we update \( d[v] \) and \( \pi[v] \).
• Each algorithm for single-source shortest-path will begin by calling Initialize_Single_Source($G, s$).
• And then Relax($u, v, w$) will be repeatedly applied to edges.
• The algorithms differ in how many times they relax each edge and the order in which they relax edges.
The Bellman-Ford Algorithm

Bellman-Ford algorithm solves the single-source shortest-path problem in general case where graph may contains cycles and edge weights may be negative.

- If there is no negative cycle, the algorithm will compute the shortest-paths and their weights.
- If there is negative cycle, the algorithm will report no solution exists.
- The idea is to repeatedly use the following procedure to progressively decrease an estimate $d[v]$ of the weight of shortest path from $s$ to $v$.

```
Relax_All(G, s)
1    For each edge $(u, v) \in E$ do
2        Relax$(u, v, w)$;
```
Lemma: Let \( p = <s = v_0, v_1, \ldots, v_k = v> \) be a path from \( s \) to \( v \) of length \( k \) and weight \( w(p) \), then after \( k \) applications of Relax_All(\( G, s \)), \( d[v] \leq w(p) \).

Proof:
Prove by induction on \( k \).

- \( k = 1 \).
  In this case, \( p = <s, v> \) and \( w(p) = w(s, v) \). After Relax(\( s, v, w \)) is applied, \( d[v] \leq d[s] + w(s, v) = w(s, v) = w(p) \).

- \( k > 1 \).

  † Let \( p_1 = <v_0, v_1, \ldots, v_{k-1}> \), then \( p_1 \) is a path of length \( k - 1 \).
  † Therefore after \( k - 1 \) applications of Relax_All(\( G, s \)), we have \( d[v_{k-1}] \leq w(p_1) \).
  † After another application of Relax_All(\( G, s \)),
    \[ d[v] \leq d[v_{k-1}] + w(v_{k-1}, v_k) \leq w(p_1) + w(v_{k-1}, v_k) = w(p) \].

□

Since shortest paths have lengths less than \( |V| \), what we need to do is to apply Relax_All(\( G, s \)) \( |V| - 1 \) times.
Bellman_Ford($G, w, s$)
1 Initialize_SingleSource($G, s$)
2 for $i := 1$ to $|V| - 1$ do
3 for each edge $(u, v) \in E$ do
4 Relax($u, v, w$);
5 for each edge $(u, v) \in E$ do
6 if $d[v] > d[u] + w(u, v)$ then
7 return False;
8 return True;
Lines 5-7 test if the graph contains negative cycle reachable from $s$.

- If there is no such cycle, then there is no edge $(u, v) \in E$ such that $d[v] > d[u] + w(u, v)$ since otherwise $d[v]$ is not the shortest-path weight from $s$ to $v$.

- If there is such a cycle $c = \langle v_0, v_1, \cdots, v_k \rangle$ where $v_0 = v_k$ and $\sum_{i=1}^k w(v_{i-1}, v_i) < 0$.

  † Suppose that (for the purpose of contradiction) for each edge $(u, v) \in E$, $d[v] \leq d[u] + w(u, v)$.

  † Then $d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$ for $1 \leq i \leq k$.

  † And $\sum_{i=1}^k d[v_i] \leq \sum_{i=1}^k d[v_{i-1}] + \sum_{i=1}^k w(v_{i-1}, v_i)$.

  † Therefore $\sum_{i=1}^k w(v_{i-1}, v_i) \geq 0$

- Time complexity: $O(|V||E|)$.
Acyclic Graph

- Suppose that graph $G$ has no cycle.
- We first use topological sorting to order the vertices of $G$.
  - If $s$ has label $k$, then for any vertex $v$ with label $< k$, there is NO PATH from $s$ to $v$, so $d[v] = \infty$.
  - We then consider each vertex with label $> k$ in the order of $k + 1, k + 2, \cdots, |V|$.
  - Consider a vertex $v$ in the above order (with label $> k$).
    We want to compute $d[v]$ and $\pi[v]$.
    We need only consider those vertices $u$ such that $(u, v)$ is an edge in $G$.
    For each $(u, v) \in E[G]$ do
      Relax($u, v, w$)

- This is correct since for any $(u, v) \in E$, label for $u$ is less the label for $v$.
- Complexity: $O(|V| + |E|)$
Non-Negative Weights

• General graph with no negative weight edge.
• Graph now is not acyclic. Therefore there is no topological order.
• What is the main idea from acyclic case?
  
  \textit{When we consider shortest path from } s \textit{ to } v \textit{, the topological order enables us to ignore all vertices after } v. 

• Could we define an order for general graphs to do similar things?
• For general graphs,

  \textit{Order the vertices by the weights of their shortest paths from } s. 

  Unlike topological order, \textit{we do not know this order} before we find shortest paths.
• We will find the order during the process of finding shortest paths.

• Can we first find the closest vertex $w_1$?
  Yes! $w_1$ is the vertex satisfying following:

$$w(s, w_1) = \min_v w(s, v)$$

Why?

Consider the shortest path from $s$ to $w_1$.
It must consist of only two vertices $s$ and $w_1$.
Otherwise if

$$s \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k \rightarrow w_1$$

is the shortest path from $s$ to $w_1$, then $d[v_1] = w(s, v_1) \leq \delta(s, w_1) = d[w_1]$

- either $w_1$ is not closest – contradiction!
- or $\delta(s, w_1) = \delta(s, v_1)$, we can choose $v_1$ to be the closest vertex.
- therefore we can determine $d[w_1]$ and find $w_1$ this way.
• Can we find the second closest vertex \( w_2 \)?

YES! The only paths we need to consider are the edges from \( s \) (except \((s, w_1)\)) and paths of two edges, the first one being \((s, w_1)\), and the second one being from \( w_1 \).

– Why? Again, consider a shortest path from \( s \) to \( w_2 \)

\[
s \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k \rightarrow w_2
\]

– Consider the first vertex (from \( s \) to \( w_2 \)) that is not \( s \) and \( w_1 \).
– It is either \( v_1 \) or \( v_2 \) (and in this case \( v_1 = w_1 \)).
– Therefore we choose the minimum of
  \[
  w(s, v) \ (v \neq w_1) \text{ or } d[w_1] + w(w_1, v) \ (v \neq s).
  \]
– this give us \( w_2 \) and \( d[w_2] \).
Induction hypothesis:
Give graph $G$ and a vertex $s$, we know the $k - 1$ vertices that are closest to $s$ and we know the weights of the shortest paths to them.

Base case: done!

Inductive Step: We want to find the $k$th ($w_k$) closest vertex and the weight of shortest path to it.
Let the $k - 1$ closest vertices be $w_1, w_2, \ldots, w_{k-1}$.
Let $V_{k-1} = \{s, w_1, w_2, \ldots, w_{k-1}\}$
The shortest path from $s$ to $w_k$ can go only through vertices in $V_{k-1}$.

(If it goes through a vertex not in $V_{k-1}$, this vertex is closer than $w_k$)

Therefore $w_k$ is the vertex satisfying the following:

$w_k \not\in V_{k-1}$ and the shortest path from $s$ to $w_k$ through $V_{k-1}$ is less or equal to the shortest path from $s$ to any other vertex $v \not\in V_{k-1}$ through $V_{k-1}$.
For $v \not\in V_{k-1}$, let
\[ d[v] = \min_{u \in V_{k-1}} (d[u] + w(u, v)). \]

$d[v]$ is the shortest path from $s$ to $v$ through $V_{k-1}$.

Therefore $w_k$ is a vertex such that
\[ w_k \not\in V_{k-1} \text{ and } d[w_k] = \min_{v \not\in V_{k-1}} \{d[v]\}. \]

- Adding $w_k$ does not change the weights of the shortest paths from $s$ to $u$, $u \in V_{k-1}$, since $u$ is closer than $w_k$.

- The Algorithm is complete now. We should consider how to implement it efficiently.

The main computation is for $d[v]$ for $v \not\in V_{k-1}$. 
• We do not have to compute all $d[v]$ for each $V_k$.

*Most of $d[v]$ for $V_k$ are equal to $d[v]$ for $V_{k-1}$.
We only need to update a few $d[v]$ when we add $w_k$.*

• When we add $w_k$
For $v$, such that $v \not\in V_k$ and $(w_k, v)$ is an edge.

$$d[v] = \min\{d[v], \; d[w_k] + w(w_k, v)\}$$

(Note: this is the same as Relax($w_k, v, w$).)

Consider a shortest path from $s$ to $v$ through $V_k$.
If the last edge is $(w_i, v)$, $i < k$, then there is no change to $d[v]$.
If the last edge is $(w_k, v)$ then $d[v] = d[w_k] + w(w_k, v)$. 

Blue: $V_{k-1}$
Green: $V_k$
What data structure should we use?

**Heap** is a good choice!

- We can keep $d[v]$ in a min_heap. Then we can find $w_k$ in $O(1)$ time.
- After we find $w_k$, we update $d[v]$.
  - Delete $w_k$ from heap.
  - For each $v$ in the heap such that $(w_k, v)$ is an edge, change its key from $d[v]$ to
    $$\min\{d[v], d[w_k] + w(w_k, v)\} \text{ (Relax}(w_k, v, w))\.$$
- We need to use the heap with element locations (see notes for heap)!
Dijkstra’s Algorithm

The above analysis gives us the Dijkstra’s algorithm.

Dijkstra(G, w, s)
1    Initialize_Single_Source(G, s);
2    S := ∅;
3    Q := V[G];
4    while Q ≠ ∅ do
5       u := Extract_Min(Q);
6       S := S ∪ {u};
7       for each (u, v) ∈ E and v ∈ Q do
8          Relax(u, v, w);
9          Update v in Q;
**Time Complexity**

With a binary heap:
- $|V|$ delete min operations: $O(|V| \log(|V|))$
- $|E|$ update operations: $O(|E| \log(|V|))$

TOTAL $O((|V| + |E|) \log(|V|))$

With a Fibonacci heap:
- $|V|$ delete min operations: $O(|V| \log(|V|))$
- $|E|$ update operations: $O(|E|)$

TOTAL $O(|V| \log(|V|) + |E|)$

Without a heap:
- $|V|$ delete min operations: $O(|V||V|)$
- $|E|$ update operations: $O(|E|)$

TOTAL $O(|V|^2 + |E|) = O(|V|^2)$

(Compare with acyclic case $O(|V| + |E|)$)
(Compare with Bellman-Ford algorithm $O(|V||E|)$)
Minimum Spanning Trees

- Consider an undirected weighted graph $G = (V, E)$.
- A spanning tree of $G$ is a connected subgraph that contains all vertices and no cycles.
- Minimum spanning tree of $G$: a spanning tree $T$ of $G$ such that the sum of the weights of edges in $T$ is minimum.
- Applications:
  - computer networks (e.g. broadcast path)
  - there is a cost for sending a message on the link.
  - broadcast a message to all computers in the network from an arbitrary computer
  - want to minimize the cost
The Problem

Given an undirected connected weighted graph $G = (V, E)$, find a spanning tree $T$ of $G$ of minimum cost.

Idea.
Extend tree: always choose to extend tree by adding cheapest edge.

For simplicity, we assume all costs (weights) are distinct!

Base case: Let $r$ be an arbitrarily chosen root vertex. The minimum-cost edge incident to $r$ must be in the minimum spanning tree (MST)

† Suppose this edge is $\{r, s\}$

† if $\{r, s\}$ is not in MST, add $\{r, s\}$ to MST

† Now we have a cycle

† Delete the MST edge incident to $r$ from the cycle. We have a new tree.
† the cost of this new tree is less than the cost of MST. Contradiction!
Induction hypothesis

Given a connected graph \( G = (V, E) \), we know how to find a subgraph \( T \) of \( G \) with \( k \) edges, such that \( T \) is a tree and \( T \) is a subgraph of the MST of \( G \).

Extend \( T \):

† Find the cheapest edge from a vertex in \( T \) to a vertex not in \( T \). Let it be \( \{u, v\} \), such that \( u \in T \) and \( v \notin T \).

† Add \( \{u, v\} \) to \( T \).

† Claim: We now have a tree with \( k + 1 \) edges which is a subgraph of the MST of \( G \).

  • Again add \( \{u, v\} \) to the MST
  • Consider the path from \( u \) to \( v \) in MST
  • There must be an edge \( e = \{u_1, v_1\} \) in this path such that \( u_1 \in T \) and \( v_1 \notin T \).
  • Delete edge \( e \)
  • Since \( \text{weight}(e) > \text{weight}(\{u, v\}) \), the new tree has a cost less than the MST
  • Contradiction
Implementation

• Similar to the implementation of single-source shortest-path algorithm

• Choose an arbitrary vertex as the root

• For each iteration we need to find the minimum cost edge connecting $T$ to vertices outside of $T$.

• We again use a heap.
  For each vertex $w$ not in $T$, we use the minimum-cost of the costs of the edges going into $w$ from a vertex in $T$ as the key.

• For each iteration we delete min from the heap. Suppose $u$ is the new vertex.
  Update the keys for vertex $v$ not in $T$ by cost of edge $\{u, v\}$.

• Time: $|V|$ delete min: $O(|V| \log(|V|))$
  $|E|$ update operations: $O(|E| \log(|V|))$
  Total: $O((|V| + |E|) \log(|V|))$

• This is called PRIMS algorithm
Prim’s Algorithm

The above analysis gives us the Prim’s algorithm.

MST_Prim(G, w, r)
  1 for each u ∈ V[G] do
  2     key[u] := ∞;
  3     π[u] := NIL;
  4     key[r] := 0;
  5     Q := V[G];
  6 while Q ≠ ∅ do
  7     u := Extract_Min(Q);
  8     for each v ∈ Adj[u] do
  9         if v ∈ Q and w(u, v) < key[v] then
 10             π[v] := u;
 11             key[v] := w(u, v);
 12             update key[v] in Q
**Kruskal’s MST**

**Idea:** Choose cheapest edge in a graph.

**Algorithm:**

put all edges in a heap, put each vertex in a set by itself;

while not found a MST yet do begin
    delete min edge, \( \{u, v\} \), from the heap;
    if \( u \) and \( v \) are not in the same set
        mark \( \{u, v\} \) as tree edge;
        union sets containing \( u \) and \( v \);
    if \( u \) and \( v \) are in the same set
        do nothing;
end

**Time:**

\( O((|V| + |E|) \log(|V|)) \) for heap operation.

\( O(|E| \log^* (|V|)) \) for union-find operation.

Total: \( O((|V| + |E|) \log(|V|)) \) time.
**All-Pair Shortest-Paths Problem**

- **The problem:** Given a weighted graph $G = (V, E)$, find the shortest paths between all pairs of vertices.

- We can call single-source shortest-paths algorithm $|V|$ times
  - † If there is no negative cycle.
    Complexity: $O(|V|^2|E|)$
  - † If there is no negative weight edge.
    Complexity: $O(|V|^2 \log(|V|) + |V||E|)$ or $O(|V|(|V| + |E|) \log(|V|))$
    If $G$ is not dense, this is a good solution.

- We consider to use induction to design a direct solution.
• We can use induction on the vertices.
• We know the shortest paths between a set of $k$ vertices ($V_k$).
• We want to add a new vertex $u$
• We can find the shortest path from $u$ to all the vertices in $V_k$

shortest-path($u, w$) =
\min_{v \in V_k, (u,v) \in E} \{ w(u, v) + \text{shortest-path}(v, w) \} (*)

Shortest-path($w, u$) can be computed similarly!

We update shortest-path($w_1, w_2$), $w_1, w_2 \in V_k$

shortest-path($w_1, w_2$) = \min \{ \text{shortest-path}($w_1, u$) + \text{shortest-path}($u, w_2$),
shortest-path($w_1, w_2$) \} (**)  

\textbf{Time:} (**) can be done in $|V|^2$
\textbf{(*)&} can be done in $|V|^2$

Total: $O(|V|^3)$. 

41
A better solution

- **Idea:** Number of vertices is fixed.
  Induction puts restrictions on the type of paths allowed

- We label vertices from 1 to $|V|$
  A path from $u$ to $w$ is called a $k$-path if, except for $u$ and $w$, the highest-labelled vertex on the path is labelled by $k$.
  A 0-path is an edge

- **Induction hypothesis:**
  We know the weights of the shortest paths between all pairs of vertices such that only $k$-paths, for some $k \leq m$ are considered.

- **Base case:** $m = 0$
  only direct edges can be considered
Inductive step

(extend $m - 1$ to $m$)

We consider all $k$-paths such that $k \leq m$.
The only new paths are $m$-paths.
Let the vertex with label $m$ be $v_m$.
Consider a shortest $m$-path between $u$ and $v$.

We can assume that this $m$-path includes $v_m$ only once!

Therefore this $m$-path is a shortest $k$-path (for some $k \leq m - 1$) between $u$ and $v_m$
appended by a shortest $j$-path (for some $j \leq m - 1$) from $v_m$ to $v$.
By induction we already know the weights of the $k$-path and the $j$-path!
We update shortest-path $(u, v)$ by:

$$\min\{\text{shortest-path}(u, v_m) + \text{shortest-path}(v_m, v), \text{shortest-path}(u, v)\}$$
This leads to a very simple program! (Floyd-Warshall algorithm)

for $x := 1$ to $|V|$ do  \{ base case \}
    for $y := 1$ to $|V|$ do
        if $(x, y) \in E$, then
            $d[x, y] := w(x, y)$;
        else
            $d[x, y] := \infty$;

for $x := 1$ to $|V|$ do
    $d[x, x] := 0$;

for $m := 1$ to $|V|$ do  \{ the induction sequence \}
    for $x := 1$ to $|V|$ do
        for $y := 1$ to $|V|$ do
            if $d[x, m] + d[m, y] < d[x, y]$ then
                $d[x, y] := d[x, m] + d[m, y]$

**Time:** $O(|V|^3)$. Again, if the graph is sparse, then $O(|V|^2 \log(|V|) + |V||E|)$ is a better solution when there is no negative weight.
If we need to find the shortest paths not just the weights. Let $\phi[i, j]$ be highest numbered vertex on the shortest path from $i$ to $j$.

\begin{verbatim}
for $x := 1$ to $|V|$ do  \{ base case \}
  for $y := 1$ to $|V|$ do
    if $(x, y) \in E$, then
      $d[x, y] := w(x, y); \quad \phi[x, y] := x$;
    else
      $d[x, y] := \infty; \quad \phi[x, y] := Nil$;

for $x := 1$ to $|V|$ do
  $d[x, x] := 0; \quad \phi[x, x] := Nil$;

for $m := 1$ to $|V|$ do  \{ the induction sequence \}
  for $x := 1$ to $|V|$ do
    for $y := 1$ to $|V|$ do
      if $d[x, m] + d[m, y] < d[x, y]$ then
        $d[x, y] := d[x, m] + d[m, y]$;
        $\phi[x, y] := m$;

Time: $O(|V|^3)$
\end{verbatim}
If we need to find the shortest paths not just the weights. Let $\pi[i,j]$ be the predecessor of $j$ on the shortest path from $i$ to $j$.

\[
\text{for } x := 1 \text{ to } |V| \text{ do } \{ \text{ base case } \}
\text{ for } y := 1 \text{ to } |V| \text{ do }
\text{ if } (x, y) \in E, \text{ then }
\quad d[x, y] := w(x, y); \quad \pi[x, y] := x;
\text{ else }
\quad d[x, y] := \infty; \quad \pi[x, y] := \text{Nil};
\]

\[
\text{for } x := 1 \text{ to } |V| \text{ do }
\quad d[x, x] := 0; \quad \pi[x, x] := \text{Nil};
\]

\[
\text{for } m := 1 \text{ to } |V| \text{ do } \{ \text{ the induction sequence } \}
\text{ for } x := 1 \text{ to } |V| \text{ do }
\text{ for } y := 1 \text{ to } |V| \text{ do }
\text{ if } d[x, m] + d[m, y] < d[x, y] \text{ then }
\quad d[x, y] := d[x, m] + d[m, y];
\quad \pi[x, y] := \pi[m, y];
\]

**Time:** $O(|V|^3)$
Example: Figure 25.1.
\[
D^{(0)} = \begin{pmatrix}
0 & 3 & 8 & \infty & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & \infty & \infty \\
2 & \infty & -5 & 0 & \infty \\
\infty & \infty & \infty & 6 & 0
\end{pmatrix}
\]
\[
\Phi^{(0)} = \begin{pmatrix}
\text{NIL} & 1 & 1 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\
4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL}
\end{pmatrix}
\]
\[
D^{(1)} = \begin{pmatrix}
0 & 3 & 8 & \infty & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & \infty & \infty \\
2 & 5 & -5 & 0 & -2 \\
\infty & \infty & \infty & 6 & 0
\end{pmatrix}
\]
\[
\Phi^{(1)} = \begin{pmatrix}
\text{NIL} & 1 & 1 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\
4 & 1 & 4 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL}
\end{pmatrix}
\]
\[
D^{(2)} = \begin{pmatrix}
0 & 3 & 8 & 4 & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & 5 & 11 \\
2 & 5 & -5 & 0 & -2 \\
\infty & \infty & \infty & 6 & 0
\end{pmatrix}
\]
\[
\Phi^{(2)} = \begin{pmatrix}
\text{NIL} & 1 & 1 & 2 & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & 2 & 2 \\
4 & 1 & 4 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL}
\end{pmatrix}
\]
\[
D^{(3)} = \begin{pmatrix}
0 & 3 & 8 & 4 & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & 5 & 11 \\
2 & -1 & -5 & 0 & -2 \\
\infty & \infty & \infty & 6 & 0
\end{pmatrix}
\]

\[
\Phi^{(3)} = \begin{pmatrix}
\text{NIL} & 1 & 1 & 2 & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & 2 & 2 \\
4 & 3 & 4 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL}
\end{pmatrix}
\]

\[
D^{(4)} = \begin{pmatrix}
0 & 3 & -1 & 4 & -4 \\
3 & 0 & -4 & 1 & -1 \\
7 & 4 & 0 & 5 & 3 \\
2 & -1 & -5 & 0 & -2 \\
8 & 5 & 1 & 6 & 0
\end{pmatrix}
\]

\[
\Phi^{(4)} = \begin{pmatrix}
\text{NIL} & 1 & 4 & 2 & 1 \\
4 & \text{NIL} & 4 & 2 & 4 \\
4 & 3 & \text{NIL} & 2 & 4 \\
4 & 3 & 4 & \text{NIL} & 1 \\
4 & 4 & 4 & 5 & \text{NIL}
\end{pmatrix}
\]

\[
D^{(5)} = \begin{pmatrix}
0 & 1 & -3 & 2 & -4 \\
3 & 0 & -4 & 1 & -1 \\
7 & 4 & 0 & 5 & 3 \\
2 & -1 & -5 & 0 & -2 \\
8 & 5 & 1 & 6 & 0
\end{pmatrix}
\]

\[
\Phi^{(5)} = \begin{pmatrix}
\text{NIL} & 5 & 5 & 5 & 1 \\
4 & \text{NIL} & 4 & 2 & 4 \\
4 & 3 & \text{NIL} & 2 & 4 \\
4 & 3 & 4 & \text{NIL} & 1 \\
4 & 4 & 4 & 5 & \text{NIL}
\end{pmatrix}
\]
\[
D^{(0)} = \left(\begin{array}{ccccccc}
0 & 3 & 8 & \infty & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & \infty & \infty \\
2 & \infty & -5 & 0 & \infty \\
\infty & \infty & \infty & 6 & 0 \\
\end{array}\right) \\
\Pi^{(0)} = \left(\begin{array}{cccc}
\text{NIL} & 1 & 1 & \text{NIL} \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\
4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \\
\end{array}\right)
\]

\[
D^{(1)} = \left(\begin{array}{ccccccc}
0 & 3 & 8 & \infty & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & \infty & \infty \\
2 & 5 & -5 & 0 & -2 \\
\infty & \infty & \infty & 6 & 0 \\
\end{array}\right) \\
\Pi^{(1)} = \left(\begin{array}{cccc}
\text{NIL} & 1 & 1 & \text{NIL} \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\
4 & 1 & 4 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \\
\end{array}\right)
\]

\[
D^{(2)} = \left(\begin{array}{ccccccc}
0 & 3 & 8 & 4 & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & 5 & 11 \\
2 & 5 & -5 & 0 & -2 \\
\infty & \infty & \infty & 6 & 0 \\
\end{array}\right) \\
\Pi^{(2)} = \left(\begin{array}{cccc}
\text{NIL} & 1 & 1 & 2 & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & 2 & 2 \\
4 & 1 & 4 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \\
\end{array}\right)
\]
\[
D^{(3)} = \begin{pmatrix}
0 & 3 & 8 & 4 & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & 5 & 11 \\
2 & -1 & -5 & 0 & -2 \\
\infty & \infty & \infty & 6 & 0
\end{pmatrix}
\]
\[
\Pi^{(3)} = \begin{pmatrix}
\text{NIL} & 1 & 1 & 2 & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & 2 & 2 \\
4 & 3 & 4 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL}
\end{pmatrix}
\]

\[
D^{(4)} = \begin{pmatrix}
0 & 3 & -1 & 4 & -4 \\
3 & 0 & -4 & 1 & -1 \\
7 & 4 & 0 & 5 & 3 \\
2 & -1 & -5 & 0 & -2 \\
8 & 5 & 1 & 6 & 0
\end{pmatrix}
\]
\[
\Pi^{(4)} = \begin{pmatrix}
\text{NIL} & 1 & 4 & 2 & 1 \\
4 & \text{NIL} & 4 & 2 & 1 \\
4 & 3 & \text{NIL} & 2 & 1 \\
4 & 3 & 4 & \text{NIL} & 1 \\
4 & 3 & 4 & 5 & \text{NIL}
\end{pmatrix}
\]

\[
D^{(5)} = \begin{pmatrix}
0 & 1 & -3 & 2 & -4 \\
3 & 0 & -4 & 1 & -1 \\
7 & 4 & 0 & 5 & 3 \\
2 & -1 & -5 & 0 & -2 \\
8 & 5 & 1 & 6 & 0
\end{pmatrix}
\]
\[
\Pi^{(5)} = \begin{pmatrix}
\text{NIL} & 3 & 4 & 5 & 1 \\
4 & \text{NIL} & 4 & 2 & 1 \\
4 & 3 & \text{NIL} & 2 & 1 \\
4 & 3 & 4 & \text{NIL} & 1 \\
4 & 3 & 4 & 5 & \text{NIL}
\end{pmatrix}
\]
If we need to find the shortest paths and shortest cycles, let $\pi[i, j]$ be the predecessor of $j$ on the shortest path from $i$ to $j$.

for $x := 1$ to $|V|$ do  \{ base case \}
    for $y := 1$ to $|V|$ do
        if $(x, y) \in E$, then
            $d[x, y] := w(x, y); \; \pi[x, y] := x$;
        else
            $d[x, y] := \infty; \; \pi[x, y] := \text{Nil};$

for $m := 1$ to $|V|$ do  \{ the induction sequence \}
    for $x := 1$ to $|V|$ do
        for $y := 1$ to $|V|$ do
            if $d[x, m] + d[m, y] < d[x, y]$ then
                $d[x, y] := d[x, m] + d[m, y];$
                $\pi[x, y] := \pi[m, y]$;

**Time:** $O(|V|^3)$
\[
D^{(0)} = \begin{pmatrix}
\infty & 3 & 8 & \infty & -4 \\
\infty & \infty & \infty & 1 & 7 \\
\infty & 4 & \infty & \infty & \infty \\
2 & \infty & -5 & \infty & \infty \\
\infty & \infty & \infty & 6 & \infty
\end{pmatrix}, \quad \Pi^{(0)} = \begin{pmatrix}
\text{NIL} & 1 & 1 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\
4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL}
\end{pmatrix}
\]

\[
D^{(1)} = \begin{pmatrix}
\infty & 3 & 8 & \infty & -4 \\
\infty & \infty & \infty & 1 & 7 \\
\infty & 4 & \infty & \infty & \infty \\
2 & 5 & -5 & \infty & -2 \\
\infty & \infty & \infty & 6 & \infty
\end{pmatrix}, \quad \Pi^{(1)} = \begin{pmatrix}
\text{NIL} & 1 & 1 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\
4 & 1 & 4 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL}
\end{pmatrix}
\]

\[
D^{(2)} = \begin{pmatrix}
\infty & 3 & 8 & 4 & -4 \\
\infty & \infty & \infty & 1 & 7 \\
\infty & 4 & \infty & 5 & 11 \\
2 & 5 & -5 & 6 & -2 \\
\infty & \infty & \infty & 6 & \infty
\end{pmatrix}, \quad \Pi^{(2)} = \begin{pmatrix}
\text{NIL} & 1 & 1 & 2 & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & 2 & 2 \\
4 & 1 & 4 & 2 & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL}
\end{pmatrix}
\]
\[
D^{(3)} = \begin{pmatrix}
\infty & 3 & 8 & 4 & -4 \\
\infty & \infty & \infty & 1 & 7 \\
\infty & 4 & \infty & 5 & 11 \\
2 & -1 & -5 & 0 & -2 \\
\infty & \infty & \infty & 6 & \infty \\
\end{pmatrix}
\]

\[
\Pi^{(3)} = \begin{pmatrix}
\text{NIL} & 1 & 1 & 2 & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & 2 & 2 \\
4 & 3 & 4 & 2 & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \\
\end{pmatrix}
\]

\[
D^{(4)} = \begin{pmatrix}
6 & 3 & -1 & 4 & -4 \\
3 & 0 & -4 & 1 & -1 \\
7 & 4 & 0 & 5 & 3 \\
2 & -1 & -5 & 0 & -2 \\
8 & 5 & 1 & 6 & 4 \\
\end{pmatrix}
\]

\[
\Pi^{(4)} = \begin{pmatrix}
4 & 1 & 4 & 2 & 1 \\
4 & 3 & 4 & 2 & 1 \\
4 & 3 & 4 & 2 & 1 \\
4 & 3 & 4 & 5 & 1 \\
\end{pmatrix}
\]

\[
D^{(5)} = \begin{pmatrix}
4 & 1 & -3 & 2 & -4 \\
3 & 0 & -4 & 1 & -1 \\
7 & 4 & 0 & 5 & 3 \\
2 & -1 & -5 & 0 & -2 \\
8 & 5 & 1 & 6 & 4 \\
\end{pmatrix}
\]

\[
\Pi^{(5)} = \begin{pmatrix}
4 & 3 & 4 & 5 & 1 \\
4 & 3 & 4 & 2 & 1 \\
4 & 3 & 4 & 2 & 1 \\
4 & 3 & 4 & 2 & 1 \\
4 & 3 & 4 & 5 & 1 \\
\end{pmatrix}
\]