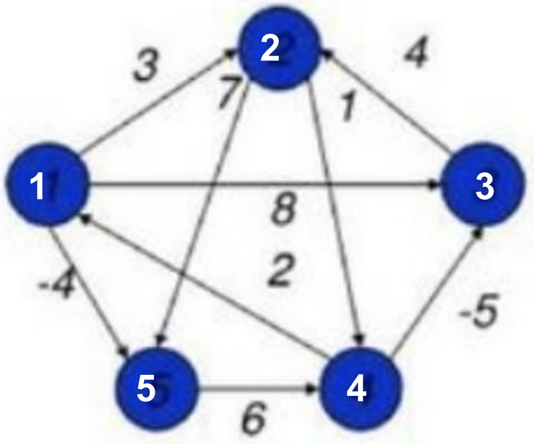
DP: All-Pairs Shortest Paths

L11. Soyoung Chung

All-Pairs Shortest Path (APSP) Problems

The all pair shortest path algorithm is also known as Floyd-Warshall algorithm is used to find all pair shortest path problem from a given weighted graph. As a result of this algorithm, it will generate a matrix, which will represent the minimum distance from any node to all other nodes in the graph.



To node:

$$
\begin{array}{c c}
\textbf{From node} & \begin{array}{ccccc} \textbf{1} & \textbf{2} & \textbf{3} & \textbf{4} & \textbf{5} \end{array} \\
\begin{array}{c} \textbf{1} \\ \textbf{2} \\ \textbf{3} \\ \textbf{4} \\ \textbf{5} \end{array} & \begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}
\end{array}
$$

Many ways to solve All-Pairs Shortest Path (APSP) Prob.

Introduction

Different types of algorithms can be used to solve the all-pairs shortest paths problem:

- Dynamic programming
- Matrix multiplication
- Floyd-Warshall algorithm
- Johnson's algorithm
- Difference constraints

Single Source Shortest Path

lgV for extract key, Constant amortized for each decreased key operation in Fibonacci heap (Dijkstra)

Single-source shortest paths

- given directed graph $G = (V, E)$, vertex $s \in V$ and edge weights $w : E \to \mathbb{R}$
- find $\delta(s, v)$, equal to the shortest-path weight $s- > v, \forall v \in V$ (or $-\infty$ if negative weight cycle along the way, or ∞ if no path)

Situtation	Algorithm	Time
unweighted ($w = 1$)	BFS	$O(V + E)$
non-negative edge weights	Dijkstra	$O(E + V \lg V)$
general	Bellman-Ford	$O(VE)$
acyclic graph (DAG)	Topological sort + one pass of B-F	$O(V + E)$

Linear Time

All of the above results are the best known. We achieve a $O(E + V \lg V)$ bound on Dijkstra's algorithm using Fibonacci heaps.

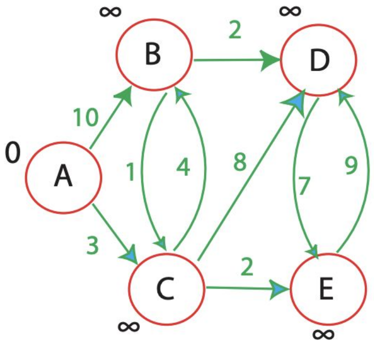
Dijkstra's Algorithm

Dijkstra's Algorithm

For each edge $(u, v) \epsilon E$, assume $w(u, v) \geq 0$, maintain a set S of vertices whose final shortest path weights have been determined. Repeatedly select $u \epsilon V - S$ with <u>minimum</u> shortest path estimate, add u to S, relax all edges out of u.

Pseudo-code

Dijkstra (G, W, s) //uses priority queue Q
  Initialize (G, s)
  $S \leftarrow \phi$
  $Q \leftarrow V[G]$ //Insert into Q
  while $Q \neq \phi$
    do $u \leftarrow$ EXTRACT-MIN(Q) //deletes u from Q
    $S = S \cup \{u\}$
    for each vertex $v \epsilon$ Adj$[u]$
      do RELAX (u, v, w) $\leftarrow$ this is an implicit DECREASE_KEY operation



	A	B	C	D	E	
S = { }	{ A	B	C	D	E } = Q	
S = { A }	0	∞	∞	∞	∞	
S = { A, C }	0	10	3	∞	∞	← after relaxing edges from A
S = { A, C }	0	7	3	11	5	← after relaxing edges from C
S = { A, C, E }	0	7	3	11	5	
S = { A, C , E, B}	0	7	3	9	5	← after relaxing edges from B

Figure 4: Dijkstra Execution

All-Pairs Shortest Path

All-pairs shortest paths

Parent pointer to get shortest path.

- given edge-weighted graph, $G = (V, E, w)$
- find $\delta(u, v)$ for all $u, v \in V$

A simple way of solving All-Pairs Shortest Paths (APSP) problems is by running a single-source shortest path algorithm from each of the V vertices in the graph.

Situtation	Algorithm	Time	$E = \Theta(V^2)$
unweighted $(w = 1)$	$\lvert V \rvert \times$ BFS	$O(VE)$	$O(V^3)$
non-negative edge weights	$\lvert V \rvert \times$ Dijkstra	$O(VE + V^2 \lg V)$	$O(V^3)$
general	$\lvert V \rvert \times$ Bellman-Ford	$O(V^2 E)$	$O(V^4)$
general	Johnson's	$O(VE + V^2 \lg V)$	$O(V^3)$

Dense graph

These results (apart from the third) are also best known — don't know how to beat $\lvert V \rvert \times$ Dijkstra

Algorithms to solve APSP

Note that for all the algorithms described below, we assume that $\boxed{w(u,v) = \infty \text{ if}}$ $\boxed{(u,v) \notin E.}$

Introduction

Different types of algorithms can be used to solve the all-pairs shortest paths problem:

- Dynamic programming
- Matrix multiplication
- Floyd-Warshall algorithm
- Johnson's algorithm
- Difference constraints

Dynamic Programming, attempt 1

If no negative weight cycles (how to know?
No negative shortest distance, dvv(n-1))

DP 1

1. **Sub-problems:** $d_{uv}^{(m)}$ = weight of shortest path $u \to v$ using $\leq m$ edges
2. **Guessing:** What's the last edge (x, v)?
3. **Recurrence:**

$$d_{uv}^{(m)} = \min(d_{ux}^{(m-1)} + w(x, v) \text{ for } x \in V)$$

X: last vertex in incoming edges
(indegree)

$$d_{uv}^{(0)} = \begin{cases} 0 & \text{if } u = v \\ \infty & \text{otherwise} \end{cases}$$

No path (not connected)

4. **Topological ordering:** for $m = 0, 1, 2, \ldots, n-1$: for u and v in V:
5. **Original problem:**

   If graph contains no negative-weight cycles (by Bellman-Ford analysis), then shortest path is simple $\Rightarrow \delta(u, v) = d_{uv}^{(n-1)} = d_{uv}^{(n)} = \cdots$

Time complexity

In this Dynamic Program, we have $O(V^3)$ total sub-problems. u, x, v (for 3 vertices)

Each sub-problem takes $O(V)$ time to solve, since we need to consider V possible choices. This gives a total runtime complexity of $O(V^4)$.

Note that this is no better than $|V| \times$ Bellman-Ford

DP 2

Bottom-up via relaxation steps

for $m = 1$ **to** n **by** 1
  for u in V
    for v in V
      for x in V Incoming edges (indegree)
        if $d_{uv} > d_{ux} + d_{xv}$ duv(m) > dux(m-1) + dxv (w(x,v))
          $d_{uv} = d_{ux} + d_{xv}$ duv(m) = dux(m-1) + dxv (w(x,v))

In the above pseudocode, we omit superscripts because more relaxation can never hurt.

Note that we can change our relaxation step to $d_{uv}^{(m)} = \min(d_{ux}^{\lceil m/2 \rceil} + d_{xv}^{\lceil m/2 \rceil}$ for $x \in V)$. This change would produce an overall running time of $O(n^3 \lg n)$ time. (student suggestion)

Matrix multiplication

MM1

Recall the task of standard matrix multiplication,

Given $n \times n$ matrices A and B, compute $C = A \cdot B$, such that $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$.

- $O(n^3)$ using standard algorithm
- $O(n^{2.807})$ using Strassen's algorithm
- $O(n^{2.376})$ using Coppersmith-Winograd algorithm
- $O(n^{2.3728})$ using Vassilevska Williams algorithm

Connection to shortest paths

- Define $\oplus = \min$ and $\odot = +$
- Then, C $= A \odot B$ produces $c_{ij} = \min_k(a_{ik} + b_{kj})$
- Define $D^{(m)} = (d_{ij}^{(m)})$, $W = (w(i,j))$, $V = \{1, 2, \ldots, n\}$

With the above definitions, we see that $D^{(m)}$ can be expressed as $D^{(m-1)} \odot W$. In other words, $D^{(m)}$ can be expressed as the circle-multiplication of W with itself m times.

MM2

Matrix multiplication algorithm

- $n - 2$ multiplications $\Rightarrow O(n^4)$ time (stil no better)
- Repeated squaring: $((W^2)^2)^{2\cdots} = W^{2^{\lg n}} = W^{n-1} = (\delta(i,j))$ if no negative-weight cycles. Time complexity of this algorithm is now $O(n^3 \lg n)$.

We can't use Strassen, etc. since our new multiplication and addition operations don't support negation.

Floyd-Warshall: Dynamic Programming, attempt 2

FW1

1. **Sub-problems:** $c_{uv}^{(k)}$ = weight of shortest path $u \to v$ whose intermediate vertices $\in \{1, 2, \ldots, k\}$

2. **Guessing:** Does shortest path use vertex k?

3. **Recurrence:**

Use k Not use k

$$c_{uv}^{(k)} = \min(c_{uv}^{(k-1)}, c_{uk}^{(k-1)} + c_{kv}^{(k-1)})$$

$$c_{uv}^{(0)} = w(u, v)$$ Not using any intermediate vertice k

4. **Topological ordering:** for k: for u and v in V: O(V+E)

5. **Original problem:** $\delta(u, v) = c_{uv}^{(n)}$. Negative weight cycle $\Leftrightarrow$ negative $c_{uu}^{(n)}$

Time complexity

This Dynamic Program contains $O(V^3)$ problems as well. However, in this case, it takes only $O(1)$ time to solve each sub-problem, which means that the total runtime of this algorithm is $O(V^3)$.

Best for APSP in dense graph

FW2

Bottom up via relaxation

$C = (w(u, v))$
for $k = 1$ **to** n **by** 1
    for u in V
        for v in V
            if $c_{uv} > c_{uk} + c_{kv}$
                $c_{uv} = c_{uk} + c_{kv}$

As before, we choose to ignore subscripts.

Johnson's algorithm

Better for sparse graph. Never be worse than Floyd Warshall

JS1

1. Find function $h : V \to \mathbb{R}$ such that $w_h(u, v) = w(u, v) + h(u) - h(v) \geq 0$ for all $u, v \in V$ or determine that a negative-weight cycle exists.

   To make all edges non-negative

2. Run Dijkstra's algorithm on (V, E, w_h) from every source vertex $s \in V \Rightarrow$ get $\delta_h(u, v)$ for all $u, v \in V$

   |V| * Dijkstra

3. Given $\delta_h(u, v)$, it is easy to compute $\delta(u, v)$

Claim. $\delta(u, v) = \delta_h(u, v) - h(u) + h(v)$

Re-weighting preserves shortest path with parent pointers

Proof. Look at any $u \to v$ path p in the graph G

- Say p is $v_0 \to v_1 \to v_2 \to \cdots \to v_k$, where $v_0 = u$ and $v_k = v$.

$$
\begin{aligned}
w_h(p) &= \sum_{i=1}^{k} w_h(v_{i-1}, v_i) \\
&= \sum_{i=1}^{k} [w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)] \\
&= \sum_{i=1}^{k} w(v_{i-1}, v_i) + h(v_0) - h(v_k) \\
&= w(p) + h(u) - h(v)
\end{aligned}
$$

- Hence all $u \to v$ paths change in weight by the same offset $h(u) - h(v)$, which implies that the shortest path is preserved (but offset).

JS2

How to find h? Function that makes all edges non-negative

We know that

$$w_h(u,v) = w(u,v) + h(u) - h(v) \geq 0$$

This is equivalent to,

$$h(v) - h(u) \leq w(u,v)$$

for all $(u,v) \in V$. This is called a **system of difference constraints**.

Theorem. *If (V, E, w) has a negative-weight cycle, then there exists no solution to the above system of difference constraints.*

Proof. Say $v_0 \to v_1 \to \cdots \to v_k \to v_0$ is a negative weight cycle.

JS3

Let us assume to the contrary that the system of difference constraints has a solution; let's call it h.

This gives us the following system of equations,

$$\begin{aligned} h(v_1) - h(v_0) &\leq w(v_0, v_1) \\ h(v_2) - h(v_1) &\leq w(v_1, v_2) \\ &\vdots \\ h(v_k) - h(v_{k-1}) &\leq w(v_{k-1}, v_k) \\ h(v_0) - h(v_k) &\leq w(v_k, v_0) \end{aligned}$$

Summing all these equations gives us

$$0 \leq w(\text{cycle}) < 0$$

which is obviously not possible.

From this, we can conclude that no solution to the above system of difference constraints exists if the graph (V, E, w) has a negative weight cycle.

JS4

Theorem. *If* (V, E, w) *has no negative-weight cycle, then we can find a solution to the difference constraints.*

Proof. Add a new vertex s to G, and add edges (s, v) of weight 0 for all $v \in V$.

- Clearly, these new edges do not introduce any new negative weight cycles to the graph
- Adding these new edges ensures that there now exists at least one path from s to v. This implies that $\delta(s, v)$ is finite for all $v \in V$
- We now claim that $h(v) = \delta(s, v)$. This is obvious from the triangle inequality: $\delta(s, u) + w(u, v) \geq \delta(s, v) \Leftrightarrow \delta(s, v) - \delta(s, u) \leq w(u, v) \Leftrightarrow h(v) - h(u) \leq w(u, v)$

JS5

Time complexity

Check if G has negative weight cycle

1. The first step involves running Bellman-Ford from s, which takes $O(VE)$ time. We also pay a pre-processing cost to reweight all the edges ($O(E)$)
2. We then run Dijkstra's algorithm from each of the V vertices in the graph; the total time complexity of this step is $O(VE + V^2 \lg V)$
3. We then need to reweight the shortest paths for each pair; this takes $O(V^2)$ time.

h function

The total running time of this algorithm is $O(VE + V^2 \lg V)$.

Applications

Bellman-Ford consult any system of difference constraints (or report that it is unsolvable) in $O(VE)$ time where V = variables and E = constraints.

An exercise is to prove the Bellman-Ford minimizes $\max_i x_i - \min_i x_i$.

This has applications to

- Real-time programming
- Multimedia scheduling
- Temporal reasoning

For example, you can bound the duration of an event via difference constraint $LB \leq t_{end} - t_{start} \leq UB$, or bound a gap between events via $0 \leq t_{start2} - t_{end1} \leq \varepsilon$, or synchronize events via $|t_{start1} - t_{start2}| \leq \varepsilon$ or 0.