# CMPT231

## Lecture 12: ch 25

### All-Pairs Shortest Paths

# James 3:13,17-18 (NASB)

Who among you is **wise** and **understanding**?
Let him show by his **good behavior**
his deeds in the **gentleness** of wisdom.

But the wisdom from above is first **pure**,
then **peaceable**, **gentle**, **reasonable**,
full of **mercy** and good **fruits**,
**unwavering**, without **hypocrisy**.

And the seed whose fruit is **righteousness**
is sown in peace by those who **make peace**.

# Outline for today

- **Single-source** shortest paths
  - **Dijkstra**: $O(|V|\log|V| + |E|)$
- **All-pairs** shortest paths
  - **Dynamic** programming by path length: $O\left(|V|^4\right)$
  - **Exponential** speedup: $O\left(|V|^3 \log|V|\right)$
  - **Floyd-Warshall** (dyn prog by vertex subset): $O\left(|V|^3\right)$
  - **Johnson** (iterative Dijkstra): $O\left(|V|^2 \log|V| + |V||E|\right)$

# Single-source shortest paths

- **Output**: for each vertex $v \in V$, store:
  - v.parent: links form a **tree** rooted at source
  - v.d: shortest-path **weight** from source
    - All initially $\infty$, except src.d = 0
- Method: **edge relaxation**
  - Will **using** the edge (u,v) give us a **shorter** path to v?
- Algorithms differ in **sequence** of relaxing edges

```
def relaxEdge( u, v, w ):
  if v.d > u.d + w( u, v ):
    v.d = u.d + w( u, v )
    v.parent = u
```

# Bellman-Ford algo for SSSP

- Allows **negative-weight** edges
  - If any **net-negative** cycle is reachable, returns FALSE
- **Relax** every edge, |V|-1 times (**complexity**?)
- Guaranteed to **converge**: shortest paths ≤ |V|-1 edges
  - Each **iteration** relaxes one edge along shortest path

```
def initSingleSource( V, E, src )
  for v in V:
    v.d = ∞
  src.d = 0
```

```
def ssspBellmanFord( V, E, w, src
  initSingleSource( V, E, src )
  for i in 1 .. |V| - 1:
```

Bellman-Ford: Fig 24-4(b)

# Dijkstra's algo for SSSP

- **Doesn't** handle negative-weight edges
  - Assumes adding an edge always increases cost
- Weighted version of **breadth-first** search
- Use **priority queue** instead of FIFO
  - Keys are the **shortest-path** estimates v.d
  - Similar to **Prim** but calculates v.d
- **Greedy** choice: select u with **lowest** u.d

```
def relaxEdge( u, v, w ):
  if v.d > u.d + w( u, v ):
    v.d = u.d + w( u, v )
    v.parent = u
```

```
def ssspDijkstra( V, E, w, src )
  initSingleSource( V, E, src )
  Q = new PriorityQueue( V )
```

Dijkstra, Fig 24-6(a)

# Dijkstra: correctness

- **Invariant**: at top of loop, u.d = y(src,u) ∀ u ∉ Q
- **Proof**: suppose **not**: let u ∉ Q with u.d ≠ y(src,u)
  - Assume u is the **first** such vertex popped from Q
  - ∃ **path** src ⇝ u (or else u.d = ∞ = y(src,u))
- Let p be a **shortest** path from src ⇝ u
  - Let (x,y) be the **first** edge in p crossing from !Q to Q
  - Since u is the **first** vertex to have u.d ≠ y(src,u),

`![Dijkstra proof, Fig 24-7](static/img/Fig-24-7.svg)`

# Dijkstra: correctness

- So $x.d = y(src,d)$ (i.e., $x$ has **converged**)
- By **convergence** property, after we **relaxed** $(x,y)$,
  - because $(x,y)$ is on the **shortest path** to $u$,
  - so now $y.d = y(src,y)$ (i.e., $y$ has **converged**)
  - And $y$ is on the **shortest path**, so $y(src,y) \leq y(src,u) \leq u.d$
- But **both** $y$ and $u \in Q$ when we `popMin()`, so $u.d \leq y.d$
- Hence $u.d = y.d = y(src,u)$, a **contradiction**

Dijkstra proof, Fig 24-7

# Dijkstra: complexity

- **Initialisation**: V( |V| )
- Q.popMin() is run exactly |V| times
- Q.setPriority() (called by relaxEdge) is run O( |E| ) times
- Using **binary min-heaps**, all ops are O( lg |V| )
  - **Total** time: O( |E| lg |V| )
- Using **Fibonacci heaps**:
  - Q.setPriority() takes only O(1) **amortised** time
  - **Total** time: O( |V| lg |V| + |E| )

# Single-source shortest paths

- Generic **outline**: relax edges
  - To iteratively find shortest distance y to each vertex
- **Bellman-Ford**: total time $O(|V|\,|E|)$
  - Relax all edges, $|V|-1$ times
  - Only one pass needed if acyclic, using topological sort
- **Dijkstra**: $O(|V| \lg |V| + |E|)$
  - BFS from source, updating shortest distance as we go
  - Use Fibonacci heap for priority queue
  - Cannot handle negative-weight edges

# Outline for today

- Single-source shortest paths
  - Dijkstra: $O(|V|\log|V| + |E|)$
- **All-pairs shortest paths**
  - **Dynamic programming by path length:** $O\left(|V|^4\right)$
  - **Exponential speedup:** $O\left(|V|^3\log|V|\right)$
  - Floyd-Warshall (dyn prog by vertex subset): $O\left(|V|^3\right)$
  - Johnson (iterative Dijkstra): $O\left(|V|^2\log|V| + |V||E|\right)$

# All-pairs shortest paths

- **Input**: directed graph (V, E), weights w
- **Output**: |V| x |V| matrix D = $(\delta_{ij})$
  - **Shortest-path** distances from each vertex i to j
- **Naive** solution: do Bellman-Ford for each vertex:
  - $O\left(|V|^2|E|\right)$: if **dense**, then |E| $\in \Theta\left(|V|^2\right)$, so $O\left(|V|^4\right)$
- If no negative edges, try **Dijkstra**, |V| times:
  - **Binary heap**: $O(|V||E|\log|V|), \ = O\left(|V|^3 \log|V|\right)$ if dense
  - **Fibonacci heap**: $O\left(|V|^2 \log|V| + |V||E|\right), \ = O\left(|V|^3\right)$ if dense

# Dynamic prog: recurrence

- Recall we have **optimal substructure**:
  - Subpaths of shortest paths are themselves shortest paths
- Use **weighted adjacency matrix** W:
  - $w_{ij}$ = w(i,j) for all **edges**
  - $w_{ii} = 0$, and $w_{ij} = \infty$ if **no** edge (i,j)
- Let $l_{ij}^{(m)}$ be the **length** (weight) of a shortest path from i to j which only uses ≤ m **edges**
  - **Recurrence**: $l_{ij}^{(m)} = \min_{k \in V} \left( l_{ik}^{(m-1)} + w_{kj} \right)$
  - **Base**: $l_{ii}^{(0)} = 0$, and $l_{ij}^{(0)} = \infty$, for i ≠ j
- After **one** iteration: $l_{ij}^{(1)} = w_{ij}$

Example of dynamic programming solution, Fig 25-1

# Dynamic prog: bottom-up

```
def APSPExtend( L, W ):
  let M = new matrix, |V| x |V|
  for i in 2 .. |V|:
    for j in 1 .. |V|:
      M[ i, j ] = infinity
      for k in 1 .. |V|:
M[ i, j ] = min( M[ i, j ], L[ i, k ] + W[ k, j ] )
  return M
```

- **Initialise** (m=1): $L^{(1)} = W$, the weighted adjacency matrix
- **Extend** paths $L^{(m-1)}$ of length m-1 to length m: $L^{(m)}$
  - Do this |V|-2 times to get **solution**: $L^{(|V|-1)}$
- **Complexity**: each Extend is $\Theta\left(|V|^3\right)$
  - So **total** is $\Theta\left(|V|^4\right)$: not too hot…

# Exponential bottom-up

- **Matrix power** $A^n$ can be calculated by
  - $A \cdot A \cdot \ldots \cdot A$ (n-1 times), or by
  - $A^2 = A \cdot A$, then $A^4 = A^2 \cdot A^2$, $A^8 = A^4 \cdot A^4$, etc.
    - Only need lg n multiplications
- Now apply this to APSPExtend's triply-nested for loop
  - If n is not a **power of 2**, we'll overshoot
  - That's ok, it **converges** after $L^{(|V|)}$
- **Complexity**: $\Theta\left(|V|^3 \log|V|\right)$: much better!

```
def Fast_APSP( W ):
  L = W
  for m in 2 .. ceiling( lg |V| ):
    L = APSPExtend( L, L )
```

# Outline for today

- Single-source shortest paths
  - Dijkstra: $O(|V|\log|V| + |E|)$
- All-pairs shortest paths
  - Dynamic programming by path length: $O\left(|V|^4\right)$
  - Exponential speedup: $O\left(|V|^3 \log|V|\right)$
  - **Floyd-Warshall (dyn prog by vertex subset):** $O\left(|V|^3\right)$
  - Johnson (iterative Dijkstra): $O\left(|V|^2 \log|V| + |V||E|\right)$

# Floyd-Warshall: substructure

- **Subtasks**: not by path length, but by **subset** of V:
  - $d_{ij}^{(k)}$ = weight of shortest-path from i to j where all **intermediate** vertices are in the subset {1 .. k} of vertices
- Let $p_{ij}$ be **shortest** path i ⇝ j w/nodes in {1 .. k}
  - If k is **not** on the path, then p has vertices only in {1 .. k-1}
  - If k is **on** the path, then **split** p into $p_{ik}$ and $p_{kj}$
    - These two subpaths have intermediate vertices only in {1 .. k-1}
  - By **induction**, the subpaths are **optimal**
  - Thus, every **shortest** path in {1 .. k} has subpaths

# Floyd-Warshall: solution

- **Recurrence**: $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$

```
def FloydWarshall( W ):
  D = W
  for k in 1 .. |V|:
    for i in 1 .. |V|:
      for j in 1 .. |V|:
D'[ i, j ] = min( D[ i , j ], D[ i, k ] + D[ k, j ] )
    D = D'
  return D
```

- Either k **not** on path, or two **subpaths** through k
  - **Base** case: \`d\_(ij)^((0))=w\_(ij)\`
  - Final **solution**: \`D^((n)) =

![Example Floyd-
Warshall: Fig 25-2]
(static/img/Fig-25-2.svg)

# F-W for transitive closure

- The **transitive closure** $(V, E*)$ of (V, E) is:
  - (i,j) $\in E* \Leftrightarrow$ ∃ **path** i ⇝ j
    - i.e., if j is **reachable** from i
- One method: run **Floyd-Warshall** with w=1 on all edges
  - ∃ path i ⇝ j iff $d_{ij} < \infty$
- Even **simpler**: $d_{ij}^{(k)}$ is just a **bit** (boolean)
  - Tracks if path i ⇝ j **exists** through {1 .. k}
- **Recurrence**: $d_{ij}^{(k)} = d_{ij}^{(k-1)} \text{ or } \left(d_{ik}^{(k-1)} \text{ and } d_{kj}^{(k-1)}\right)$
  - Same pseudocode structure as Floyd-Warshall
  - Still $\Theta\left(|V|^3\right)$, but **bitwise** logical ops are even

# Outline for today

- Single-source shortest paths
  - Dijkstra: $O(|V|\log|V| + |E|)$
- All-pairs shortest paths
  - Dynamic programming by path length: $O\left(|V|^4\right)$
  - Exponential speedup: $O\left(|V|^3 \log|V|\right)$
  - Floyd-Warshall (dyn prog by vertex subset): $O\left(|V|^3\right)$
  - **Johnson (iterative Dijkstra):** $O\left(|V|^2 \log|V| + |V||E|\right)$

# Dijkstra for all-pairs

- For **sparse** graphs, running **Dijkstra** on each vertex is **faster** than Floyd-Warshall:
  - If $|E| = o\left(|V|^2\right)$, then (using Fibonacci heaps) $O\left(|V|^2 \log|V| + |V||E|\right) = o\left(|V|^3\right)$
- But Dijkstra can't handle **negative weights**:
  - Need to **reweight** without changing shortest paths
- Given $h : V \to \mathbb{R}$, let w'(u,v) = w(u,v) + h(u) - h(v)
  - p is a **shortest path** u ⇝ v under w
    ⇔ p is a **shortest path** under w'
  - w'(p) = w(p) + h(u) - h(v) is **indep** of intermediate

# Outline for today

- **Single-source** shortest paths
  - **Dijkstra**: $O(|V|\log|V| + |E|)$
- **All-pairs** shortest paths
  - **Dynamic** programming by path length: $O\left(|V|^4\right)$
  - **Exponential** speedup: $O\left(|V|^3\log|V|\right)$
  - **Floyd-Warshall** (dyn prog by vertex subset): $O\left(|V|^3\right)$
  - **Johnson** (iterative Dijkstra): $O\left(|V|^2\log|V| + |V||E|\right)$