ch25: All-Pairs Shortest Path

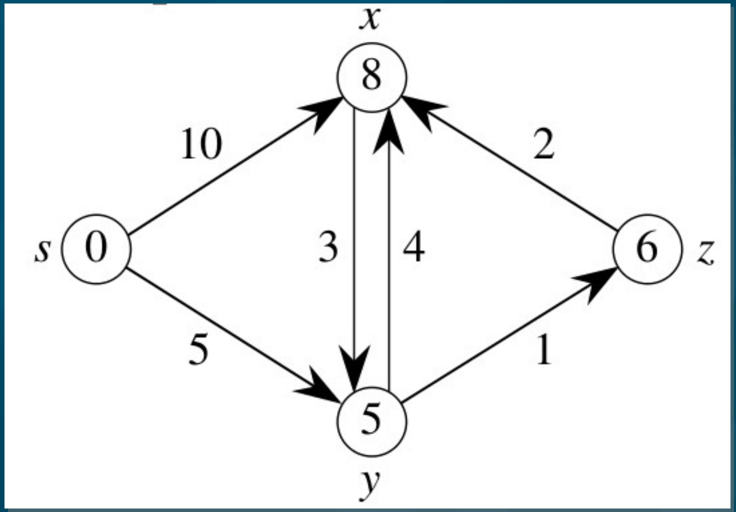
3 Dec 2013
CMPT231
Dr. Sean Ho
Trinity Western University

Outline for today

- Single-source shortest path:
  - Bellman-Ford greedy algorithm: O(VE)
  - Dijkstra's greedy algorithm: O(V lg V + E)
- All-pairs shortest paths:
  - Dyn prog by path length:
    - Naive bottom-up solution: O(V^4)
    - Exponential bottom-up: O(V^3 lg V)
  - Floyd-Warshall dyn prog: O(V^3)
    - Transitive closure
  - Johnson's algorithm for sparse: O(V^2 lg V + VE)
- Semester review

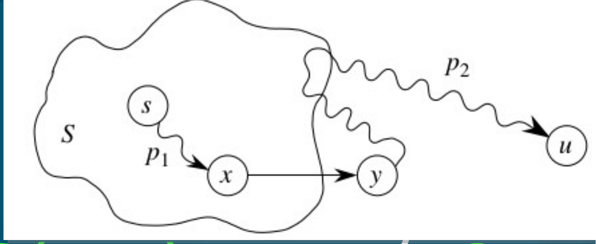
Dijkstra's algorithm

- No negative-weight edges allowed
- Weighted version of breadth-first search
- Use priority queue instead of FIFO
  - Keys are the shortest-path estimates v.d
  - Similar to Prim's algo but calculating v.d
    - Dijkstra(V, E, w, s):
      - InitSingleSource(V, E, s)
      - Q = new PriorityQueue(V)
      - while Q not empty:
        - u = ExtractMin(Q)
        - for each v ∊ Adj(u):
          - Relax(u,v,w)



- Greedy choice: select u with lowest u.d

Dijkstra: correctness



- Loop invariant: at top of loop, u.d = δ(s,u) ∀ u ∉ Q
- Proof: suppose not: let u be the first vertex removed from Q that has u.d ≠ δ(s,u)
  - ∃ path s ⇝ u (otherwise, u.d = ∞ = δ(s,u))
  - Let p be a shortest path s ⇝ u, and let (x,y) be the first edge in p crossing from !Q to Q
    - Then x.d = δ(s,d) (as u is first to have u.d ≠ δ(s,u))
  - (x,y) was then relaxed, so y.d = δ(s,y) (convgc)
    - y on shortest path, so δ(s,y) ≤ δ(s,u) ≤ u.d
  - But both y,u ∈ Q when ExtractMin, so u.d ≤ y.d
  - Hence y.d = u.d, so u.d = δ(s,u), contradiction

Dijkstra: running time

- Init for weights and Q takes $\Theta(V)$
- ExtractMin is run exactly $|V|$ times
- DecreaseKey (called by Relax) is run $O(E)$ times
- Using binary max-heaps:
  - All operations are $O(\lg V)$
  - $\Rightarrow$ Total time $O(E \lg V)$
- Using Fibonacci heaps:
  - ExtractMin takes $O(1)$ amortised time
  - Other operations total $O(V)$ ops with amortised time $O(\lg V)$ each
  - $\Rightarrow$ Total time $O(V \lg V + E)$

Outline for today

- Single-source shortest path:
  - Bellman-Ford greedy algorithm: O(VE)
  - Dijkstra's greedy algorithm: O(V lg V + E)
- All-pairs shortest paths:
  - Dyn prog by path length:
    - Naive bottom-up solution: $O(V^4)$
    - Exponential bottom-up: $O(V^3 \lg V)$
  - Floyd-Warshall dyn prog: $O(V^3)$
    - Transitive closure
  - Johnson's algorithm for sparse: $O(V^2 \lg V + VE)$
- Semester review

All-pairs shortest paths

- Input: directed graph $G=(V,E)$, weights w
  - Vertices numbered $1..n$
- Output: nxn matrix $D = (d_{ij} = \delta_{ij})$ of shortest-path distances from vertex i to j
- Naive solution: run Bellman-Ford for every vertex:
  - $O(V^2E)$: if graph is dense, $E = \Theta(V^2)$, so get $O(V^4)$
- If no negative-weight edges, try Dijkstra:
  - binary heap: $O(VE \lg V)$, $= O(V^3 \lg V)$ if dense
  - Fib heap: $O(V^2 \lg V + VE)$, $= O(V^3)$ if dense
- Floyd-Warshall: $O(V^3)$, even if dense

Dynamic programming solution

- Work toward a dynamic programming solution:
  - Recall we have optimal substructure: subpaths of shortest paths are shortest paths
- Represent graph as weighted adjacency matrix:
  - $w_{ii} = 0$ ∀i, and $w_{ij} = \infty$ if no edge $i \to j$
- Recurrence for naive recursive solution:
  - Let $l_{ij}^{(m)}$ = wt of shortest-path $i \to j$ w/ $\leq m$ edges
  - Compute as $l_{ij}^{(m)} = \min_{k=1..n}\{ l_{ik}^{(m-1)} + w_{kj} \}$
  - Base case: $l_{ij}^{(0)} = 0$ if $i=j$, and ∞ otherwise
- Note $l_{ij}^{(1)} = w_{ij}$, and $l_{ij}^{(m \geq n-1)} = \delta_{ij}$ (i.e., the solution)

Bottom-up solution

- Start with $L^{(1)} = W$, the weighted adjacency matrix
- Extend paths $L^{(m-1)}$ of length $m-1$ to length m: $L^{(m)}$
    - Extend(L, W):
        - let $L' = (l'_{ij})$ be a new nxn matrix
        - for i in 1..n, for j in 1..n:
            - $l'_{ij} = \infty$
            - for k = 1..n:
                - $l'_{ij} = \min(l'_{ij}, l_{ik} + w_{kj})$
    - Call this $n-2$ times to get up to solution $L^{(n-1)}$
    - Time: $\Theta(n^3)$ for Extend, so $\Theta(n^4)$ total: not hot...
- Key observation:
  Extend looks a lot like matrix multiply!

Exponential bottom-up

- To find a matrix power A^n, can either do
  - $A*A*\ldots*A$ ($n-1$ times), or
  - $A*A \rightarrow A^2$, then $A^2*A^2 \rightarrow A^4$, etc:
    - Only $\lg n$ multiplications
- Apply this to extending all-pairs shortest paths:
    - Faster-APSP(W):
      - $L^{(1)} = W$
      - for $m = 1$ .. $\mathrm{ceil}(\lg n)$:
        - $L^{(2\hat{}m)}$ = Extend($L^{(m)}$, $L^{(m)}$, W)
      - return $L^{(2\hat{}m)}$
  - If n not a power of 2, this overshoots, but that's okay since products don't change after $L^{(n)}$
  - Time: $\Theta(n^3 \lg n)$: better!

Outline for today

- Single-source shortest path:
  - Bellman-Ford greedy algorithm: O(VE)
  - Dijkstra's greedy algorithm: O(V lg V + E)
- All-pairs shortest paths:
  - Dyn prog by path length:
    - Naive bottom-up solution: $O(V^4)$
    - Exponential bottom-up: $O(V^3 \lg V)$
  - Floyd-Warshall dyn prog: $O(V^3)$
    - Transitive closure
  - Johnson's algorithm for sparse: $O(V^2 \lg V + VE)$
- Semester review

Floyd-Warshall substructure

- Define subtasks not by path length, but by vertices in a subset of V:
  - Let $d_{ij}^{(k)}$ = weight of shortest-path $i \rightsquigarrow j$ where all intermediate vertices along path are in $\{1..k\}$
- Optimal substructure:
  - Let p_{ij} be a shortest path $i \rightsquigarrow j$ with intermediate vertices in $\{1..k\}$
  - Either vertex k is not on the path, or if it is, then split path into $i \rightsquigarrow k \rightsquigarrow j$, where each subpath has intermediate vertices only in $\{1..k\text{-}1\}$
  - Hence every optimal solution on $\{1..k\}$ has subpaths that are optimal on $\{1..k\text{-}1\}$

FW: recurrence, solution

- Recurrence: $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
  - Either k not on path, or two subpaths through k
  - Base case: $d_{ij}^{(0)} = w_{ij}$
  - Final solution: $D^{(n)} = (d_{ij}^{(n)})$
- Bottom-up dynamic programming solution:
  - FloydWarshall(W):
    - $D^{(0)} = W$
    - for k = 1..n: for i = 1..n: for j = 1..n:
      - $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
    - return $D^{(n)}$
  - (can even drop superscripts to save memory)
  - Time: $\Theta(n^3)$

Use FW for transitive closure

- Transitive closure $G^* = (V, E^*)$ of a graph $G = (V, E)$:
  - $(i,j) \in E^*$ iff $\exists$ path $i \rightsquigarrow j$ in G
    (j is "reachable" from i)
- One way: run FW with $w=1$ on all edges:
  - $\exists$ path $i \rightsquigarrow j$ iff $d_{ij} < \infty$
- Even simpler: $d_{ij}^{(k)}$ is just a bit (0/1) tracking if $\exists$ path $i \rightsquigarrow j$ with all intermediate nodes in $1..k$
  - Recurrence: $d_{ij}^{(k)} = d_{ij}^{(k-1)}$ OR ($d_{ik}^{(k-1)}$ AND $d_{kj}^{(k-1)}$)
  - Same outline as FW, still $\Theta(n^3)$
  - But bitwise logical operations are even faster!

Outline for today

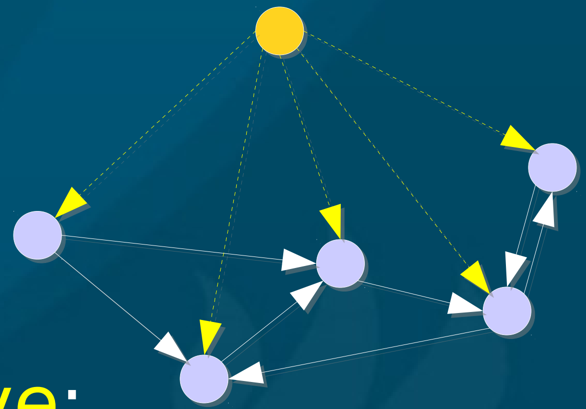
- Single-source shortest path:
  - Bellman-Ford greedy algorithm: O(VE)
  - Dijkstra's greedy algorithm: O(V lg V + E)
- All-pairs shortest paths:
  - Dyn prog by path length:
    - Naive bottom-up solution: O(V^4)
    - Exponential bottom-up: O(V^3 lg V)
  - Floyd-Warshall dyn prog: O(V^3)
    - Transitive closure
  - Johnson's algorithm for sparse: O(V^2 lg V + VE)
- Semester review

Using Dijkstra for APSP

- For sparse graphs, Dijkstra (w/Fib heaps) on every vertex can be better than FW:
  - $O(V^2 \lg V + VE)$ is better than $O(V^3)$ if $|E| = o(V^2)$
- But Dijkstra can't handle negative weights!
  - Need to reweight in a way that doesn't change shortest paths, but now has all $w \geq 0$
- Given $h: V \to \mathbb{R}$, let $w'(u,v) = w(u,v) + h(u) - h(v)$.
  - Lemma: p is a shortest path $u \rightsquigarrow v$ under w iff p is a shortest path $u \rightsquigarrow v$ under w'.
    - $w'(p) = w(p) + h(u) - h(v)$: indep of intermed verts
  - Neg-wt cycles also are preserved under this kind of reweighting

Johnson's reweighting

- Johnson's trick: add new vertex s: $V' = V \cup \{s\}$
  - Add zero-weight edges from s to all vertices: $E' = E \cup \{(s,v): v \in V\}$, and $w(s,v) = 0\ \forall\ v$
  - Compute $\delta(s,v)$ for all $v \in V$ (e.g., Bellman-Ford)
  - Reweight using $h(v) = \delta(s,v)$



- Proof that this makes weights positive:
  - By triangle inequality, $\delta(s,v) \leq \delta(s,u) + w(u,v)$
  - Hence $h(v) \leq h(u) + w(u,v)$
  - Hence $w'(u,v) = w(u,v) + h(u) - h(v) \geq 0$

Johnson's alg for APSP

- $\Theta(V+E)$
- $\Theta(VE)$
- $\Theta(E)$
- |V| times:

  O(V lg V + E)

  O(V)

- Johnson(G,w):
  - Create G' = (V', E') with new vertex s
  - BellmanFord(G', w, s) to get $\delta(s,v)$ for all $v \in V$
    - If returns FALSE, we have a neg-wt cycle: quit
  - Reweight: $w'(u,v) = w(u,v) + \delta(s,u) - \delta(s,v)$
  - for each $u \in V$:
    - Dijkstra(G, w', u) to get $\delta'(u,v)$ for all $v \in V$
    - for each $v \in V$:
      - $d_{uv} = \delta'(u,v) - \delta(s,u) + \delta(s,v)$
  - return $D = (d_{uv})$
- Innermost for loop converts back to original weighting
- Total time: $O(V^2 \lg V + VE)$

Outline for today

- Single-source shortest path:
  - Bellman-Ford greedy algorithm: O(VE)
  - Dijkstra's greedy algorithm: O(V lg V + E)
- All-pairs shortest paths:
  - Dyn prog by path length:
    - Naive bottom-up solution: O(V^4)
    - Exponential bottom-up: O(V^3 lg V)
  - Floyd-Warshall dyn prog: O(V^3)
    - Transitive closure
  - Johnson's algorithm for sparse: O(V^2 lg V + VE)
- Semester review

Semester review

- 1-4: Intro: Θ/O/Ω, induction, solving recurrences
  - Divide-and-conquer: max-subarray, Strassen
- 6-8: Sorts: insert, merge, heap, quick, radix, buckt
- 10-12: Data structs: hash, list, stack/Q, tree/BST
- 15-16: Dynamic programming & greedy
  - Rod, Fib, mat-chain, opt BST, activity sel, Huff
- 22-25: Graphs:
  - Breadth-first, depth-first, topo sort, conn comps
  - Min span tree: Kruskal, Prim
  - Single-source: Bellman-Ford, Dijkstra, DAG
  - All-pairs: Floyd-Warshall, Johnson

ch1-4: Algorithmic analysis

- Definitions: O, Ω, Θ, o, ω
- Analysis: pseudocode ⇒ running time
- Divide-and-conquer:
  - Merge sort: $\Theta(n \lg n)$ but out-of-place
  - Max subarray in $\Theta(n \lg n)$
  - Matrix multiply: divide-and-conquer $\Theta(n^3)$
    - Strassen $\Theta(n^{\lg 7})$
- Math & logic: proofs, ⇒, ∀, ∃, log, n!, Stirling
- Solving recurrences:
  - Induction (“substitution”) w/recurrence tree
  - Master method

ch6-8: Sorting

- Comparison sorts: $\Omega(n \lg n)$ theoretical bound
  - Worst-case inputs? Best-case inputs?
  - Insertion sort: $\Theta(n^2)$
  - Merge sort: $\Theta(n \lg n)$ but out-of-place
  - Heap sort: $\Theta(n \lg n)$, and priority queue
  - Quick sort: $\Theta(n \lg n)$ average case
    - Randomised variant
- Linear-time sorts: assumptions?
  - Counting sort: $\Theta(n + k)$ *(k values)*
  - Radix sort: $\Theta(d(n+k))$ *(d digits)*
  - Bucket sort on [0,1): $\Theta(n)$

ch10-12: Data structures

- Hash tables:
  - Collision handling by chaining
    - Load factor and time for search / insert / delete
  - Hash functions: criteria
    - div, mul, universal hashing
  - Collision handling by open addressing
    - Probe sequences: linear, quad, double-hash
- Linked lists: single, double, circular
  - Stacks and queues (impl using linked lists)
- Trees: degree, height, depth, traversals
  - BSTs: min/max, search, insert, delete

ch15-16: Dynamic prog

- Optimal substructure
  - Naive top-down recursive solution
  - Top-down with memoisation
  - Bottom-up dynamic programming
- Examples: 1D: rod cutting, Fibonacci
  - 2D: matrix-chain mult, optimal BST
  - Shortest unweighted path vs longest
- Greedy choice property
  - Activity selection, Huffman coding
  - Fractional knapsack vs 0-1 knapsack

ch22-25: Graph algorithms

- $G = (V,E)$: adjacency list vs adjacency matrix
- Breadth-first w/FIFO; depth-first w/recursion stack
  - Appl: topological sort, connected components
- Minimum spanning tree
  - Kruskal using disjoint-sets: $O(E \lg E)$
  - Prim using priority queue: $O(V \lg V + E)$
- Single-source shortest path
  - Bellman-Ford: $\Theta(VE)$; on DAG: $\Theta(V+E)$
  - Dijkstra w/priority queue: $O(V \lg V + E)$
- All-pairs: Floyd-Warshall: $O(V^3)$
  - Johnson w/priority queue: $O(V^2 \lg V + E)$