Advanced Algorithms

Lecture 12: Shortest paths (contd.)

Announcements

- **HW 3 is out** (Wednesday after fall break) .

- Mid-term exam (read HW 3!)

  (10 : 45 - noon) .

Problem

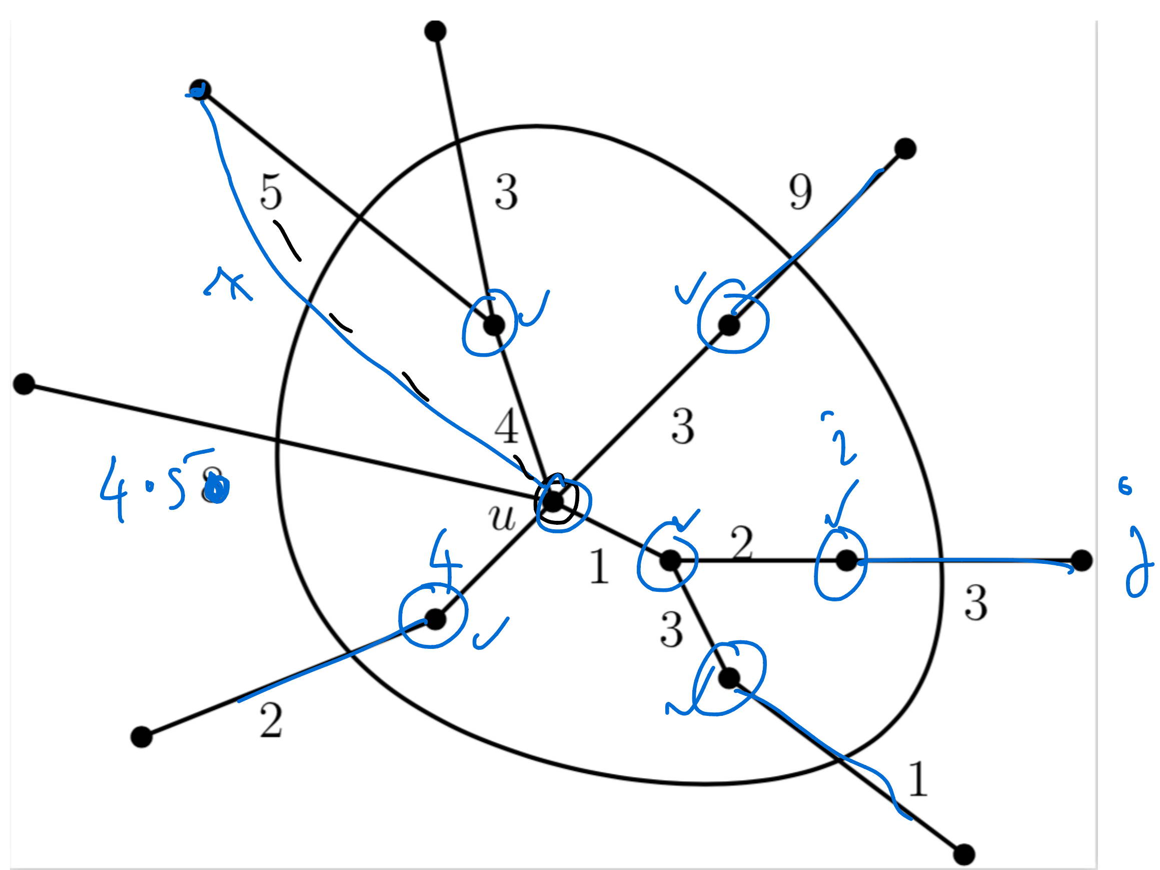
Problem: given a (directed) graph G = (V, E) with edge lengths (> 0), find shortest length path from u to v.

Dijkstra's algorithm

- Observation: can do O(m+n) time if graph is **unweighted** (BFS)
- Can we create regions of "growing radius" around u?

Construct ball of radius 'r'?

What is the closest vertex (to *u*) outside the ball?



r = 4

$\{i, j\}$.

- Consider all edges going out from S
- dist(u → i) + length(ij)
- Pick j that has the smallest value for this qty.

Dijkstra's algorithm

- Maintain "distance" array (will slowly get populated)
- Start with $S = \{u\}$, radius = 0, set dist(u) = 0

do the foll until ~~…~~ v is reached:

- For all {ij} where *i* is in S and *j* is outside:
    - candidate_dist(j) = dist(i) + length(ij)
- Pick the 'j' with the smallest candidate_dist(), add it to S, set dist(j)

break ties arbitrarily.

= dist(i) + length(i, j).

Correctness

- Claim 1: every time we set the "dist" value of a vertex w, it is the *minimum* distance from u to w.

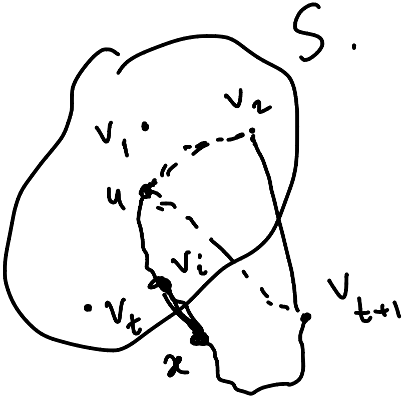
- Claim 2: all vertices reachable from u will eventually have dist value set

(by induction on 't').

Prove inductively that vertices added in first 't' iterations satisfy this property.

t=0 is the base case → trivially correct.

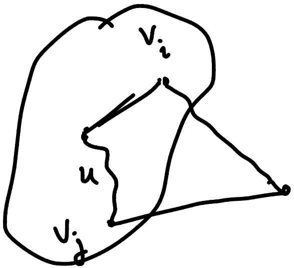
Inductive step: Assume that dist values are correct for the first 't' iterations ($t \geq 0$). Prove for $t+1$.



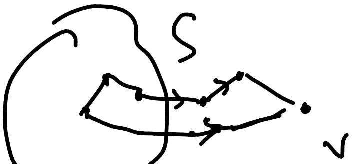
I.e., dist value assigned by alg to v_{t+1} is the true shortest path length from $u \to v_{t+1}$

- Proof by contradiction; suppose there was some other (shorter) path to v_{t+1}. That path must have exited S at some pt. If $u \rightsquigarrow v_i \rightsquigarrow x \rightsquigarrow v_{t+1}$ was shorter than path we found, then $u \rightsquigarrow v_i \rightsquigarrow x$ is also shorter than path we found to v_{t+1}

– This contradicts the choice of V_{t+1}

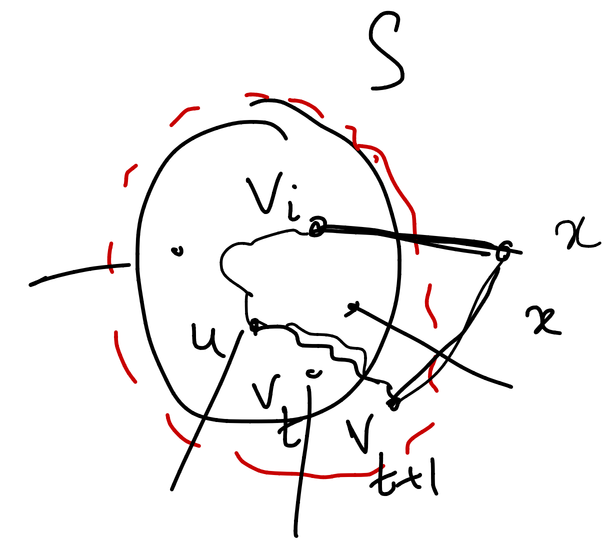


Claim 2 $\iff$ if v is reachable from u, then v is eventually included in S.



[If S stopped growing, S must have reached v.].

Running time



Implementing as stated:

- each iteration takes time = const. # edges out of S.

$\leq m$ (# edges in the graph).

Overall run-time $\leq n \cdot m$.

Improved algorithm:

- for every $x \in V$, maintain the best (i.e. shortest candidate path so far).

- when we add a new vertex v_{t+1}, then:

  - for x outside S, check if $u \to v_{t+1} \rightsquigarrow x$ is better than current candidate path.
  - if yes, update the candidate path.

[The algorithm can actually be implemented in $O((m+n)\log n)$ time

Deja vu – Prim's algorithm

(MST algorithm)

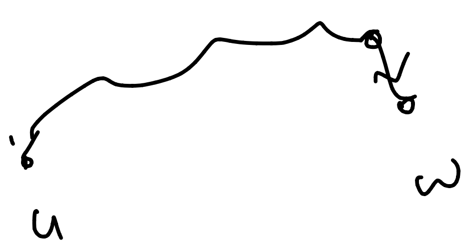
Comparisons

$O((m+n)\log n)$.

- Dynamic programming (Bellman-Ford / Shimbel's algorithm) — O(n(m+n)) time

- Much nicer in some settings …

always had "start" u .



distance from $u \to w$ using L hops.

$\text{dist}(w, L)$.

$$\text{dist}(w, L+1) = \min_{x:\, x \to w \text{ is a path}} \underbrace{\text{dist}(x, L) + \text{len}}(x, w)$$

Alternate view: Bellman-Ford

- Maintain dist(v) — array initialized to INF, dist(u) = 0
- For t = 1, 2, …, n:
  - for every vertex "w":
    set new_dist(w) = $\min_{\text{neighbors } x}$ {dist(x) + length(xw)}
  - dist(w) = min(dist(w), new_dist(w))

after t iterations,
we get the best path
with $\leq t$ hops.

Simple, "parallel" algorithm…

Also faster in "small world" graphs

All pairs shortest paths

– Given a graph $G = (V, E)$, goal is to output an $n \times n$ matrix whose $(i,j)^{th}$ entry is the length of the shortest path from $i \to j$.

($|V| = n$, $|E| = m$)

Easy:

→ Do Dijkstra for every pair i, j. $\left(\underline{\underline{n^2}} \times (m+n)\log n \right)$

Slightly better:

→ Observe that Dijkstra actually gives shortest paths from u to all other vertices $\left[\underbrace{n \times (m+n) \log n}_{n^3} \right]$.

Bellman-Ford idea:

$$\underline{\underline{dist[u,v]}} = \min \begin{cases} \min\limits_{x \in nbrhood(v)} dist(u,x) + leng(x,v) \\ dist(u,v). \end{cases}$$

→ Can be viewed as Matrix multiplication with "appropriate operators".

Matrix multiplication

Approximate shortest paths

- Reducing number of edges — spanners