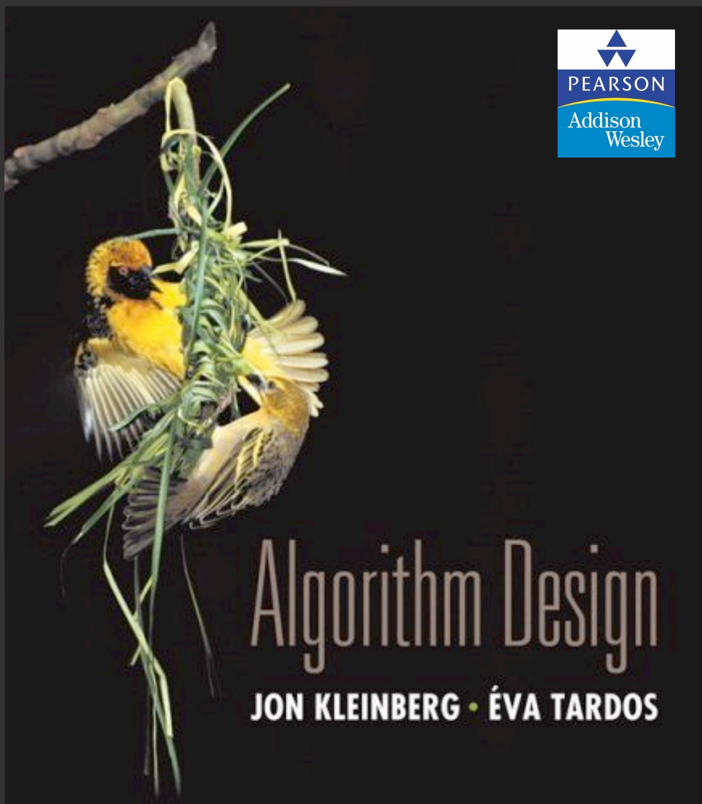


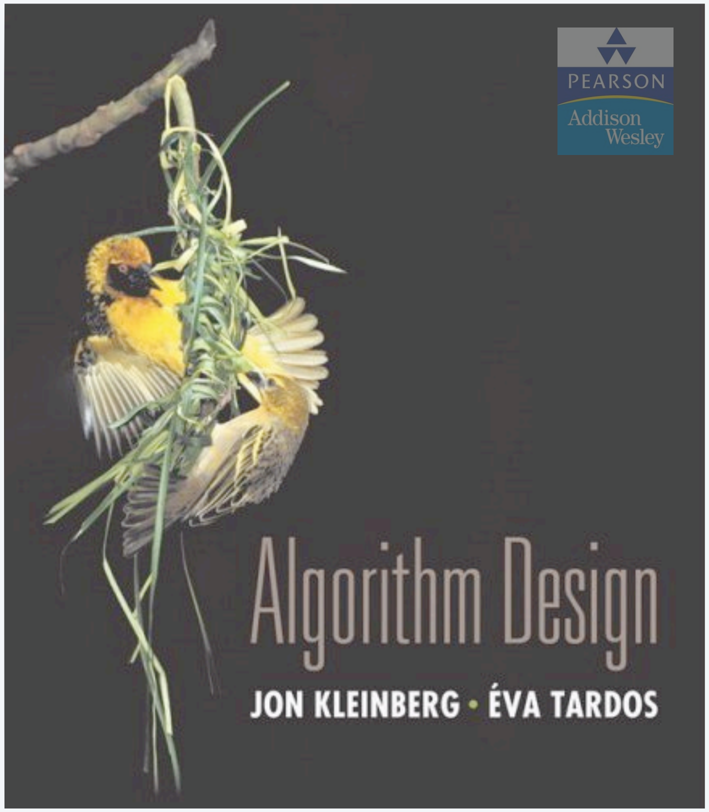
4. Greedy Algorithms II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal, Boruvka*
- *single-link clustering*
- *min-cost arborescences*

Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on Sep 8, 2013 6:30 AM



SECTION 4.4

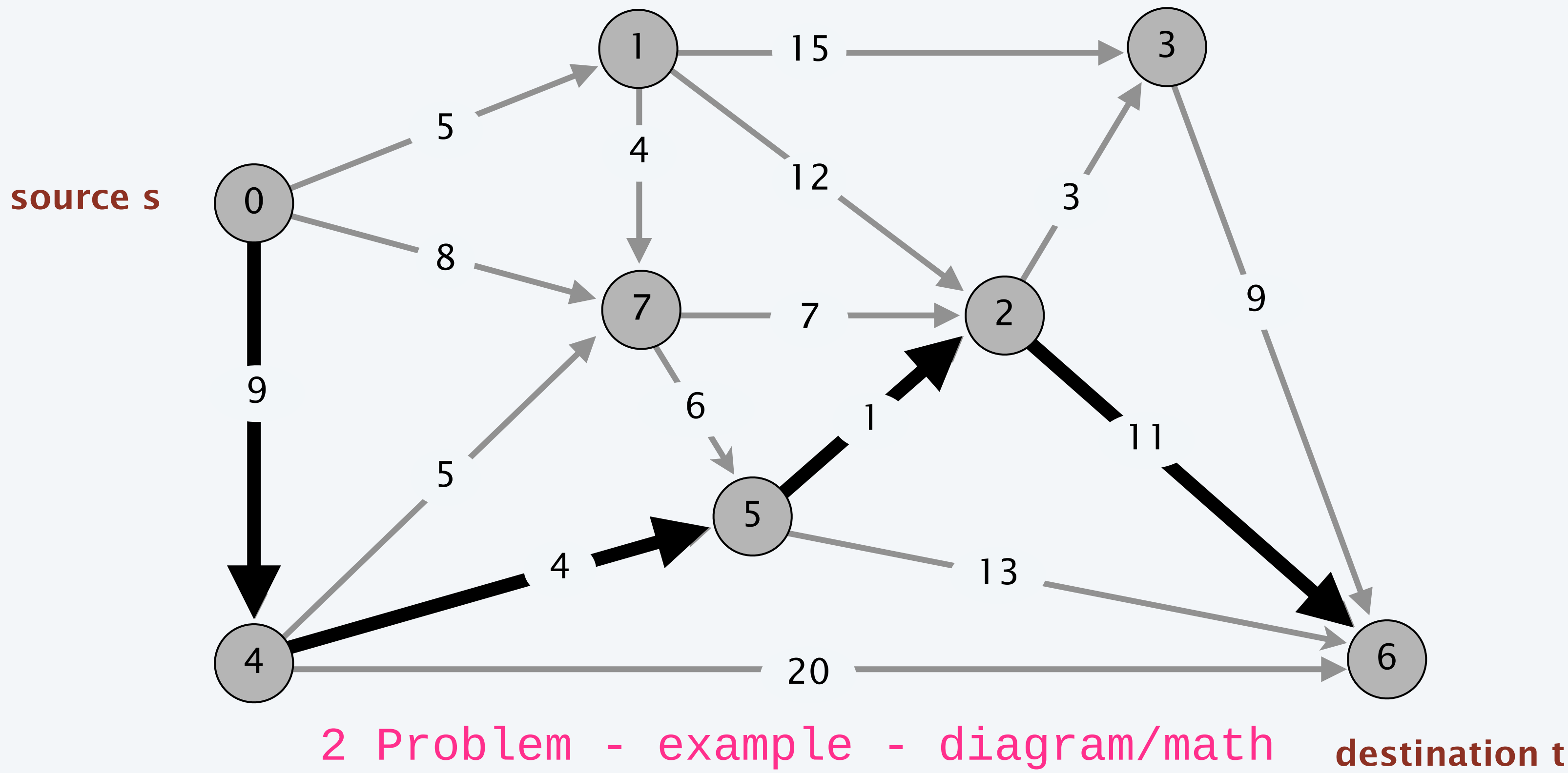
4. GREEDY ALGORITHMS II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal, Boruvka*
- *single-link clustering*
- *min-cost arborescences*

Shortest-paths problem

Problem. Given a digraph $G = (V, E)$, edge lengths $\ell_e \geq 0$, source $s \in V$, and destination $t \in V$, find the shortest directed path from s to t.

1 Problem - definition - English/Math



2 Problem - example - diagram/math

length of path = 9 + 4 + 1 + 11 = 25

Car navigation

3 Problem - application - picture

Shortest path applications

- PERT/CPM.
- Map routing.
- Seam carving.
- Robot navigation.
- Texture mapping.
- Typesetting in LaTeX.
- Urban traffic planning.
- Telemarketer operator scheduling.
- Routing of telecommunications messages.
- Network routing protocols (OSPF, BGP, RIP).
- Optimal truck routing through given traffic congestion pattern.

4 Problem - applications - list

Reference: Network Flows: Theory, Algorithms, and Applications, R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Prentice Hall, 1993.

Dijkstra's algorithm

5。Algorithm - analogy - English

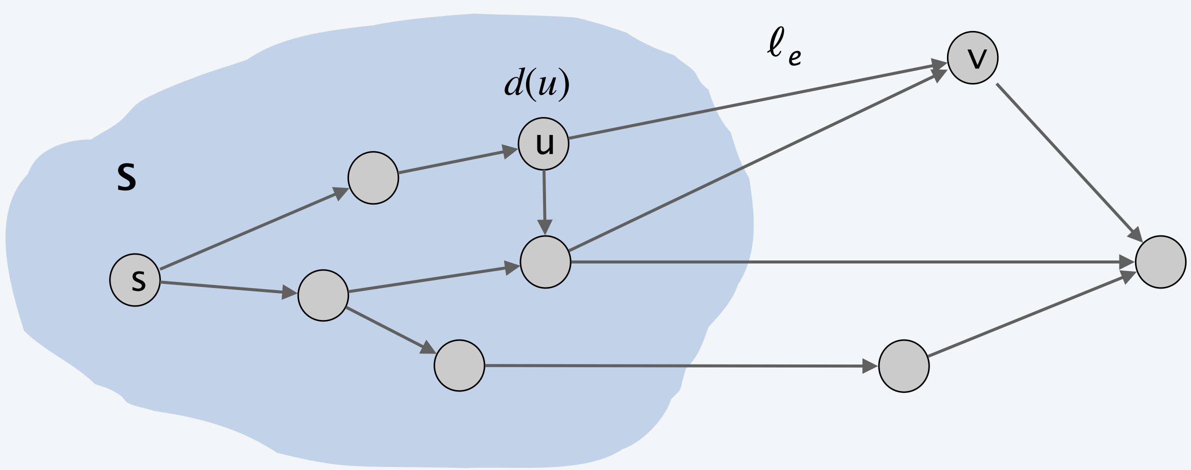
Greedy approach. Maintain a set of explored nodes S for which algorithm has determined the shortest path distance $d(u)$ from s to u.

- Initialize $S = \{ s \},\ d(s) = 0$.
- Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e ,$$

shortest path to some node u in explored part, followed by a single edge (u, v)

6 Algorithm - definition - English/math/diagram



Dijkstra's algorithm

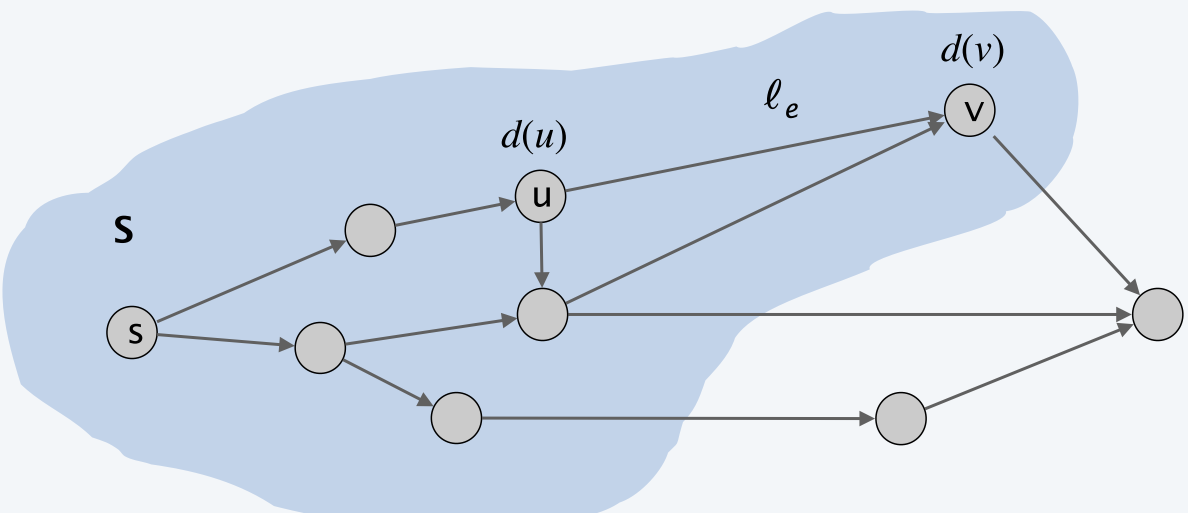
Greedy approach. Maintain a set of explored nodes S for which algorithm has determined the shortest path distance $d(u)$ from s to u.

- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e = (u,v) \,:\, u \in S} d(u) + \ell_e ,$$

add v to S, and set $d(v) = \pi(v)$.

shortest path to some node u in explored part, followed by a single edge (u, v)



Dijkstra's algorithm: proof of correctness

7 Algorithm - proof - English/diagram/math

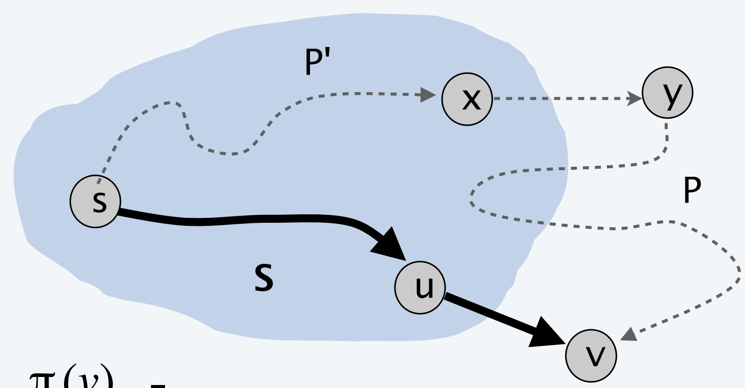
Invariant. For each node $u \in S$, $d(u)$ is the length of the shortest $s \rightsquigarrow u$ path.

Pf. [by induction on $|S|$]

Base case: $|S| = 1$ is easy since $S = \{ s \}$ and $d(s) = 0$.

Inductive hypothesis: Assume true for $|S| = k \geq 1$.

- Let v be next node added to S, and let (u, v) be the final edge.
- The shortest $s \rightsquigarrow u$ path plus (u, v) is an $s \rightsquigarrow v$ path of length $\pi(v)$.
- Consider any $s \rightsquigarrow v$ path P. We show that it is no shorter than $\pi(v)$.
- Let (x, y) be the first edge in P that leaves S, and let P' be the subpath to x.
- P is already too long as soon as it reaches y.



$$\underbrace{\ell(P) \geq \ell(P') + \ell(x, y)}_{\text{nonnegative lengths}} \;\; \underbrace{\geq d(x) + \ell(x, y)}_{\text{inductive hypothesis}} \;\; \underbrace{\geq \pi(y)}_{\text{definition of } \pi(y)} \;\; \underbrace{\geq \pi(v)}_{\text{Dijkstra chose v instead of y}} \;\; \blacksquare$$

Dijkstra's algorithm: efficient implementation

8 Implementation - idea - English/math

Critical optimization 1. For each unexplored node v, explicitly maintain $\pi(v)$ instead of computing directly from formula:

$$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e .$$

- For each $v \notin S$, $\pi(v)$ can only decrease (because S only increases).
- More specifically, suppose u is added to S and there is an edge (u, v) leaving u. Then, it suffices to update:

$$\pi(v) = \min \{ \pi(v), \; d(u) + \ell(u, v) \}$$

9 Implementation - idea - English

Critical optimization 2. Use a priority queue to choose the unexplored node that minimizes $\pi(v)$.

Dijkstra's algorithm: efficient implementation

Implementation.

- Algorithm stores $d(v)$ for each explored node v.
- Priority queue stores $\pi(v)$ for each unexplored node v.
- Recall: $d(u) = \pi(u)$ when u is deleted from priority queue.

10 Implementation - definition - list/pseudocode

DIJKSTRA (V, E, s)

Create an empty priority queue.

FOR EACH $v \neq s$: $d(v) \leftarrow \infty$; $d(s) \leftarrow 0$.

FOR EACH $v \in V$: *insert* v with key $d(v)$ into priority queue.

WHILE (the priority queue *is not empty*)

  $u \leftarrow$ *delete-min* from priority queue.

  FOR EACH edge $(u, v) \in E$ leaving u:

    IF $d(v) > d(u) + \ell(u, v)$

      decrease-key of v to $d(u) + \ell(u, v)$ in priority queue.

      $d(v) \leftarrow d(u) + \ell(u, v)$.

Dijkstra's algorithm: which priority queue?

11 Implementation - performances - list/table

Performance. Depends on PQ: n insert, n delete-min, m decrease-key.

- Array implementation optimal for dense graphs.
- Binary heap much faster for sparse graphs.
- 4-way heap worth the trouble in performance-critical situations.
- Fibonacci/Brodal best in theory, but not worth implementing.

PQ implementation	insert	delete-min	decrease-key	total
unordered array	$O(1)$	$O(n)$	$O(1)$	$O(n^2)$
binary heap	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(m \log n)$
d-way heap (Johnson 1975)	$O(d \log_d n)$	$O(d \log_d n)$	$O(\log_d n)$	$O(m \log_{m/n} n)$
Fibonacci heap (Fredman-Tarjan 1984)	$O(1)$	$O(\log n)$ †	$O(1)$ †	$O(m + n \log n)$
Brodal queue (Brodal 1996)	$O(1)$	$O(\log n)$	$O(1)$	$O(m + n \log n)$

† amortized

Extensions of Dijkstra's algorithm

Dijkstra's algorithm and proof extend to several related problems:

- Shortest paths in undirected graphs: $d(v) \leq d(u) + \ell(u, v)$.
- Maximum capacity paths: $d(v) \geq \min \{ \pi(u),\ c(u, v) \}$.
- Maximum reliability paths: $d(v) \geq d(u) \times \gamma(u, v)$.
- ...

12 Algorithm - variants - list

Key algebraic structure. Closed semiring (tropical, bottleneck, Viterbi).