CSE 421
Algorithms

Richard Anderson
Lecture 8
Optimal Caching
Dijkstra's algorithm

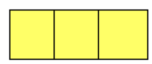
Today's Lecture

- Optimal Caching (Section 4.3)
- Dijkstra's Algorithm (Section 4.4)

Optimal Caching

- Caching problem:
  – Maintain collection of items in local memory
  – Minimize number of items fetched

Caching example



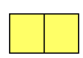
A, B, C, D, A, E, B, A, D, A, C, B, D, A

Optimal Caching

- If you know the sequence of requests, what is the optimal replacement pattern?
- Note – it is rare to know what the requests are in advance – but we still might want to do this:
  – Some specific applications, the sequence is known
  – Competitive analysis, compare performance on an online algorithm with an optimal offline algorithm

Farthest in the future algorithm

- Discard element used farthest in the future



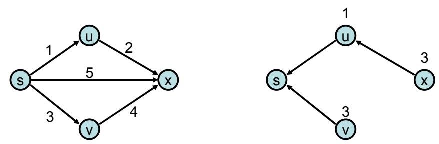
A, B, C, A, C, D, C, B, C, A, D

Correctness Proof

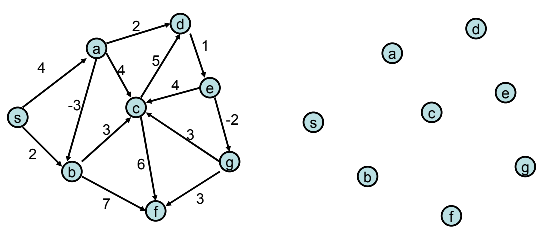
- Sketch
- Start with Optimal Solution O
- Convert to Farthest in the Future Solution F-F
- Look at the first place where they differ
- Convert O to evict F-F element
  - There are some technicalities here to ensure the caches have the same configuration . . .

Single Source Shortest Path Problem

- Given a graph and a start vertex s
  - Determine distance of every vertex from s
  - Identify shortest paths to each vertex
    - Express concisely as a “shortest paths tree”
    - Each vertex has a pointer to a predecessor on shortest path

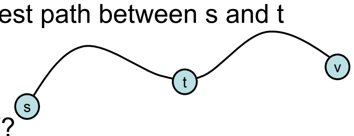


Construct Shortest Path Tree from s



Warmup

- If P is a shortest path from s to v, and if t is on the path P, the segment from s to t is a shortest path between s and t

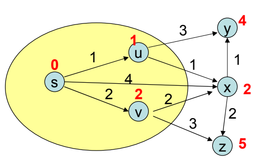


- WHY?

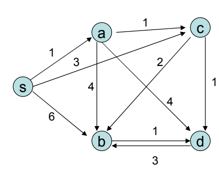
Assume all edges have non-negative cost

Dijkstra’s Algorithm

S = {}; d[s] = 0; d[v] = infinity for v != s
While S != V
    Choose v in V-S with minimum d[v]
    Add v to S
    For each w in the neighborhood of v
        d[w] = min(d[w], d[v] + c(v, w))



Simulate Dijkstra’s algorithm (strarting from s) on the graph



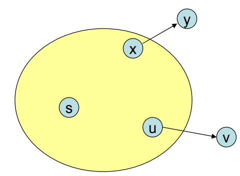
Round	Vertex Added	s	a	b	c	d
1						
2						
3						
4						
5						

Dijkstra’s Algorithm as a greedy algorithm

- Elements committed to the solution by order of minimum distance

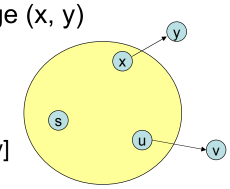
Correctness Proof

- Elements in S have the correct label
- Key to proof: when v is added to S, it has the correct distance label.



Proof

- Let P_v be the path of length d[v], with an edge (u,v)
- Let P be some other path to v. Suppose P first leaves S on the edge (x, y)
  - $P = P_{sx} + c(x,y) + P_{yv}$
  - $Len(P_{sx}) + c(x,y) >= d[y]$
  - $Len(P_{yv}) >= 0$
  - $Len(P) >= d[y] + 0 >= d[v]$

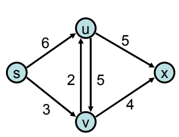


Negative Cost Edges

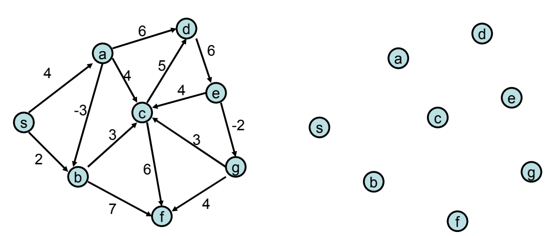
- Draw a small example a negative cost edge and show that Dijkstra’s algorithm fails on this example

Bottleneck Shortest Path

- Define the bottleneck distance for a path to be the maximum cost edge along the path



Compute the bottleneck shortest paths



How do you adapt Dijkstra's algorithm to handle bottleneck distances

- Does the correctness proof still apply?