The Design and Analysis of Algorithms

Lecture 7 Greedy Algorithms II

Zhenbo Wang

Department of Mathematical Sciences, Tsinghua University

Content

Optimal Cashing

Minimum Spanning Tree

Clustering

Optimal Offline Caching

Caching. Cache with capacity to store k items.

Sequence of m item requests $d_1, d_2, \cdots, d_m$.

Cache hit: item already in cache when requested.

Cache miss: item not already in cache when requested: must bring requested item into cache, and evict some one, if full.

Goal. Schedule that minimizes number of evictions.

Ex. $k = 2$, initial cache = ab, requests: a, b, c, b, c, a, a, b.

Optimal eviction schedule: 2 evictions.

requests	cache	
a	a	b
b	a	b
c	c	b
b	c	b
c	c	b
a	a	b
a	a	b
b	a	b

Optimal Offline Caching: Greedy Algorithms

- *LIFO / FIFO*. Evict element brought in most/least recently.
- *LRU*. Evict element whose most recent access was earliest.
- *LFU*. Evict element that was least frequently requested.
- *Farthest-in-future*. Evict item in the cache that is not requested until farthest in the future.

current cache: a b c d e f

future queries: g a b c e d a b b a c d e a f a d e f g h ...

cache miss

eject this one

Claim. *FF* is optimal eviction schedule!

Algorithm is intuitive; proof is subtle.

Reduced Eviction Schedules

Def. A *reduced* schedule is a schedule that only inserts an item into the cache in a step in which that item is requested.

- *Intuition*. Can transform an unreduced schedule into a reduced one with no more cache misses.

a	a	b	c
a	a	x	c
c	a	d	c
d	a	d	b
a	a	c	b
b	a	x	b
c	a	c	b
a	a	b	c
a	a	b	c

an unreduced schedule

a	a	b	c
a	a	b	c
c	a	b	c
d	a	d	c
a	a	d	c
b	a	d	b
c	a	c	b
a	a	c	b
a	a	c	b

a reduced schedule

Reduced Eviction Schedules

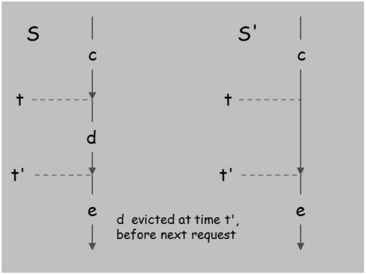
Claim. Given any unreduced schedule S, can transform it into a reduced schedule S' with no more evictions.

Pf. [by induction on number of unreduced items] Suppose S brings d into the cache at time t, without a request.

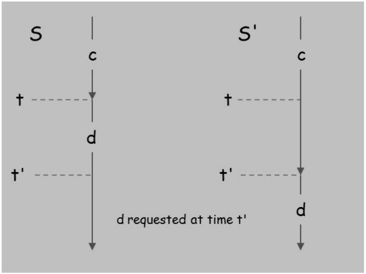
Let c be the item S evicts when it brings d into the cache.

Case 1. d evicted at time t', before next request for d.

Case 2. d requested at time t' before d is evicted. $\square$



Case 1



Case 2

Farthest-In-Future: Analysis

Theorem 1 (Bélády 1966)

FF is optimal eviction schedule.

Pf. Follows directly from the following claim.

Claim. There exists an optimal reduced schedule S that makes the same eviction schedule as S_{FF} through the first j requests.

Pf. [by induction on j] Let S be reduced schedule that satisfies the claim through j requests.

We produce S' that satisfies the claim after $j+1$ requests.

Proof–Con't

Consider $(j+1)^{st}$ request $d = d_{j+1}$.

Since S and S_{FF} have agreed up until now, they have the same cache contents before request $j+1$.

Case 1: [d is already in the cache]. $S' = S$ satisfies the claim.

Case 2: [d is not in the cache and S and S_{FF} evict the same element]. $S' = S$ satisfies the claim.

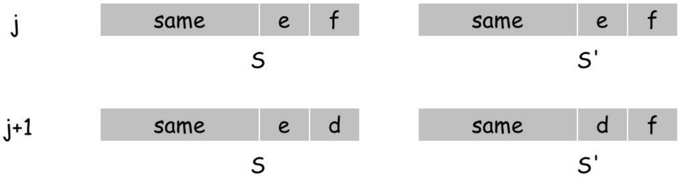
Proof–Con't

Case 3: [d is not in the cache; S_{FF} evicts e; S evicts $f \neq e$].

Construction of S' from S by evicting e instead of f.

S' agrees with S_{FF} on first $j+1$ requests; we show that having element f in cache is no worse than having element e.

Let S' behave the same as S until S' is forced to take a different action (because either S evicts e; or because either e or f is requested)



Proof–Con't

Let j' be the first time after $j+1$ that S' must take a different action from S, and let g be item requested at time j'.

- *Case 3a*: $g = e$.

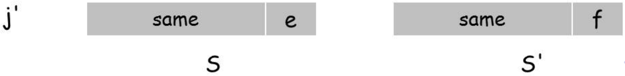
  Can't happen with FF since there must be a request for f before e.

- *Case 3b*: $g = f$.

  Element f can't be in cache of S, so let e' be the element that S evicts.

  if $e' = e$, S' accesses f; now S and S' have same cache;

  if $e' \neq e$, we make S' evict e' and brings e into the cache; now S and S' have the same cache.



Proof–Con't

We let S' behave exactly like S for remaining requests.

- *Case 3c*: $g \neq e, f$. S evicts e.

  Make S' evict f.

  Now S and S' have the same cache. □

Caching Perspective

- Online vs. offline algorithms.

  Offline: full sequence of requests is a priori.

  Online (reality): requests are not known in advance.

  Caching is among most fundamental online problems in CS.

- *LIFO*. Evict page brought in most recently.

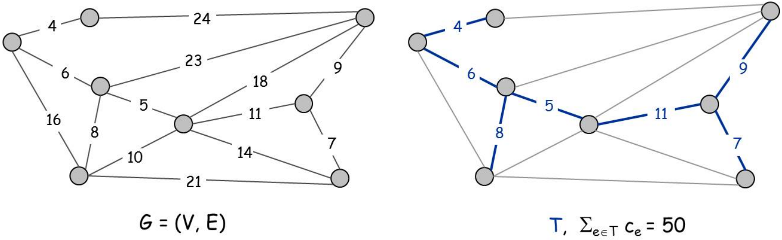
- *LRU*. Evict page whose most recent access was earliest.

  LRU is k-competitive. [Section 13]

  LIFO is arbitrarily bad.

Minimum Spanning Tree

- *Minimum spanning tree.* Given a connected graph $G = (V, E)$ with real-valued edge weights c_e, an MST is a subset of the edges $T \subseteq E$ such that T is a spanning tree whose sum of edge weights is minimized.



Theorem 2 (Cayley's Theorem)

There are n^{n-2} spanning trees of K_n.

Applications

- MST is fundamental problem with diverse applications.

  Network design.

  Approximation algorithms for NP-hard problems.

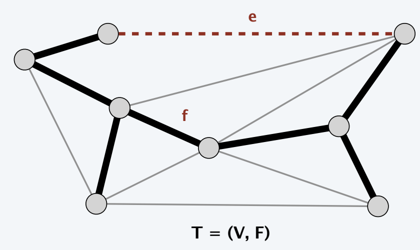
  Max bottleneck paths.

  Cluster analysis.

Fundamental Cycle

- Adding any non-tree edge e to a spanning tree T forms unique cycle C.
- Deleting any edge $f \in C$ from $T \cup \{e\}$ results in new spanning tree.

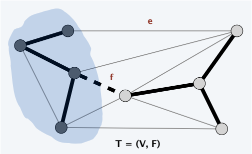
  Observation. If $c_e < c_f$, then T is not an MST.



Fundamental Cutset

- Deleting any tree edge f from a spanning tree T divide nodes into two connected components. Let D be cutset.
- Adding any edge $e \in D$ to $T - \{f\}$ results in new spanning tree.

  Observation. If $c_e < c_f$, then T is not an MST.



Greedy Algorithms

- Kruskal's algorithm. Start with $T = \emptyset$. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

- Prim's algorithm. Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T.

Theorem 3

Kruskal's (Prim's) algorithm can find MST in $O(m \log n)$ time.

Clustering

- *Clustering*. Given a set U of n objects labeled $p_1, \cdots, p_n$, classify into coherent groups.
- *Distance function*. Numeric value specifying "closeness" of two objects.
- *Fundamental problem*. Divide into clusters so that points in different clusters are far apart.

  Routing in mobile ad hoc networks.

  Identify patterns in gene expression.

  Document categorization for web search.

  Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.

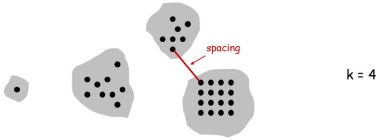
Clustering of Maximum Spacing

- *k-clustering*. Divide objects into k non-empty groups.
- *Distance function*. Assume it satisfies several natural properties.

  $d(p_i, p_j) = 0$ iff $p_i = p_j$ (indiscernible)

  $d(p_i, p_j) \geq 0$ (nonnegativity)

  $d(p_i, p_j) = d(p_j, p_i)$ (symmetry)
- *Spacing*. Min distance between any pair of points in different clusters.
- *Clustering of maximum spacing*. Given an integer k, find a k-clustering of maximum spacing.



Greedy Clustering Algorithm

- *Single-link k-clustering algorithm*.

  Form a graph on the vertex set U, corresponding to n clusters.

  Find the closest pair of objects such that each object is in a different cluster, and add an edge between them.

  Repeat $n - k$ times until there are exactly k clusters.

- *Key observation*. This procedure is precisely Kruskal's algorithm (except we stop when there are k connected components).

Remark. Equivalent to finding an MST and deleting the $k - 1$ most expensive edges.

Greedy Clustering Algorithm: Analysis

Theorem 4

Let C^* denote the clustering $C_1^*, \cdots, C_k^*$ formed by deleting the $k-1$ most expensive edges of a MST. C^* is a k-clustering of max spacing.

Pf. Let C denote some other clustering $C_1, \cdots, C_k$.

The spacing of C^* is the length d^* of the $(k-1)^{st}$ most expensive edge.

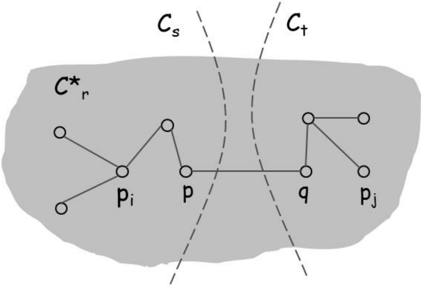
Let p_i, p_j be in the same cluster in C^*, say C_r^*, but different clusters in C, say C_s and C_t.

Proof-Con't

Some edge (p, q) on $p_i - p_j$ path in C_r^* spans two different clusters in C.

All edges on $p_i - p_j$ path have length $\leq d^*$ since Kruskal chose them.

Spacing of C is $\leq d^*$ since p and q are in different clusters. $\square$



Homework

- Read Chapter 4 of the textbook.
- Exercises 8 & 18 in Chapter 4.