GREEDY ALGORITHMS, PART 2: THEORY OF GREEDY ALGORITHMS

Marcus Ritt

CMP 601 – Algorithms and Theory of Computation —

<2020-04-14 ter>

Outline

1. Subset systems
2. Optimal caching
3. Shortest path

- A *subset system* $S = (U, \mathcal{V})$ has
  - universe of elements U
  - a set of feasible sets $\mathcal{V}$, closed under inclusion (=independent)
- For weights $w_u \geq 0, u \in U$ we have the *optimization problem*

$$S^* = \operatorname{argmax}_{S \in \mathcal{V}} W(S) \tag{1}$$

to find the feasible set S^* of largest total weight $W(S^*) = \sum_{e \in S} w_e$.

Natural greedy algorithm

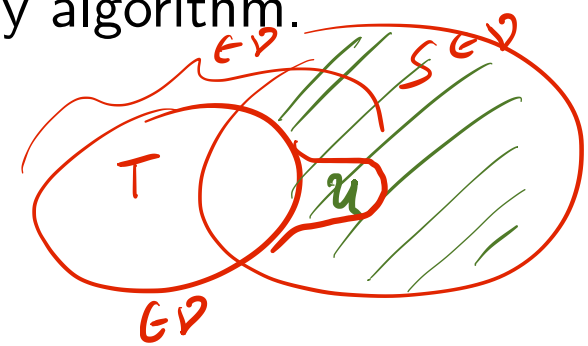
$S := \emptyset$
while $U \neq \emptyset$ *do*
  select $u \in U$ *s.t.* w_u *maximal*
  $U := U \setminus \{u\}$
  if $S \cup \{u\} \in \mathcal{V}$ *then*
    $S := S \cup \{u\}$
  end if
end while
return S

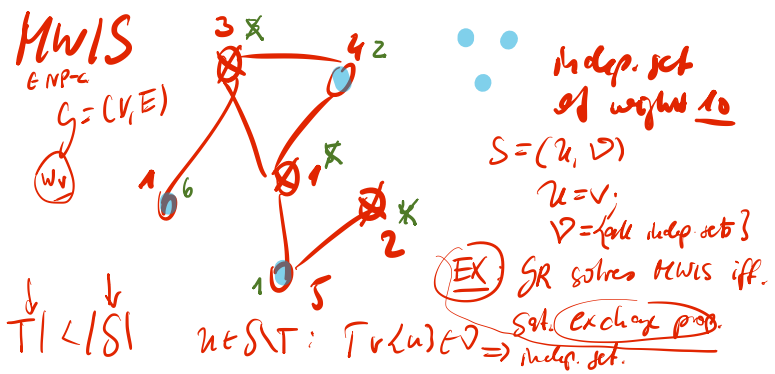
$O(n)$ $O(n)$ $O(n)$ $\Rightarrow O(n^2)$

- A subset system has the *exchange property* (EP) if for all $S, T \in \mathcal{V}$ s.t. $|S| > |T|$ there is a $u \in S \setminus T$ s.t. $T \cup \{u\} \in \mathcal{V}$.
- An independent subset system that satisfies EP is a *matroid*.
- Main result:

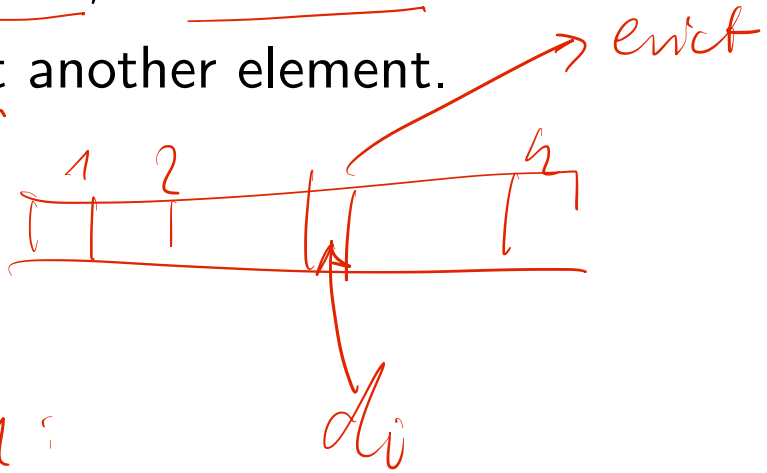
  The greedy algorithm solves the corresponding optimization problem of an independent subset system $S = (U, \mathcal{V})$ iff S is a matroid. (for any weights)

Proof: $\rightarrow$: exchange argument; $\leftarrow$: weights that fool the greedy algorithm.

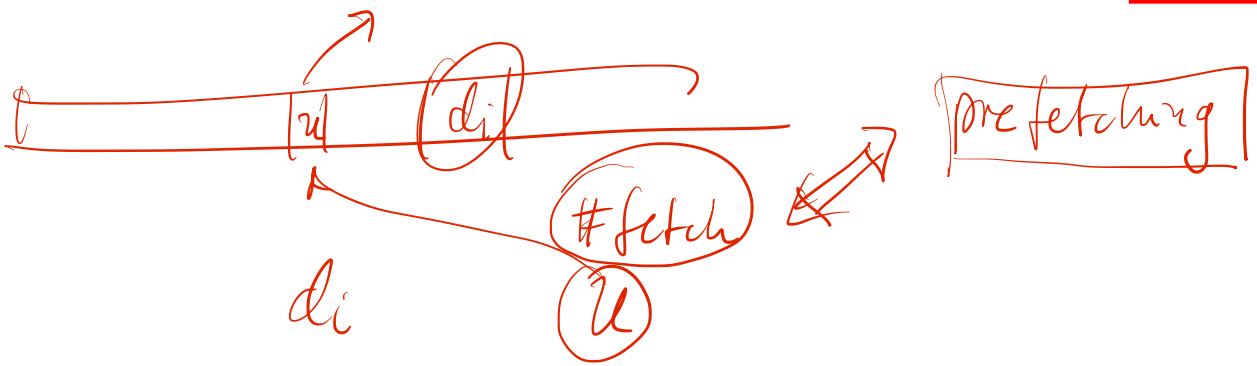




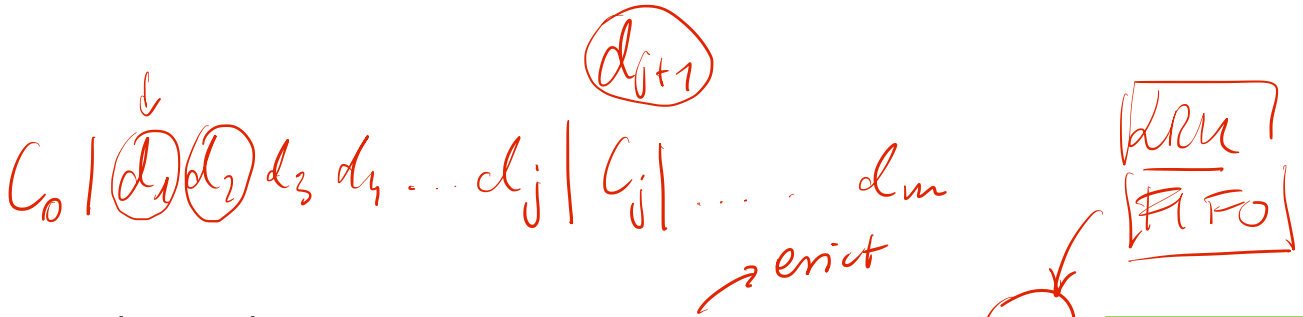
- Take a universe U of n elements, a cache of size k, initially some arbitrary items in cache.
- Consider a sequence of references $d_1, d_2, \ldots, d_m$, $d_i \in U$ to the cache.
- To process d_i: either cache hit, or cache miss.
- On miss: fetch d_i and evict another element.



Reduced schedules

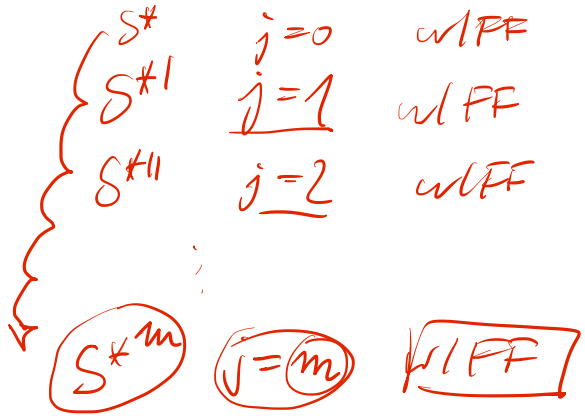


- In principle: we could take action at any time.
- A *reduced schedule*: acts only if d is required but not in cache.
- Every schedule can be made reduced, by lazy evaluation: postpone acts until needed.

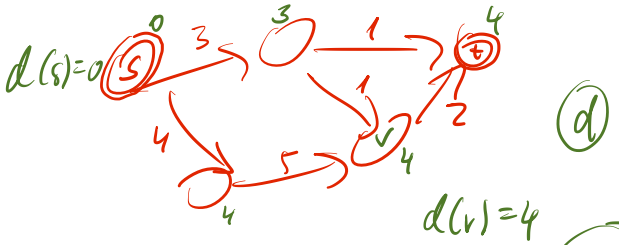


- Belady (1966): farthest in future schedule FF is optimal.
- Claim: If schedules S and FF agree on prefix j, then there is S' that agrees on prefix $j+1$ and does not cost more.
- This allows us to transform optimal schedule S^* to FF.

Proof: (more complicated) exchange argument.



SSSP ⇐ Shortest (st)-path.
APSP (all pairs shortest path) ⇒ $n \times$ SSSP



- Directed graph $G = (V, A)$, with distances $d_a = d_{(u,v)} \geq 0$ on arcs $a = (u, v) \in A$.
- Source vertex $v \in V$
- Find: shortest distance $dist(s, v)$ from s to all vertices v (single-source shortest path)
  (Store it in array d, initially $d(s) = 0$ and $d(v) = \infty, v \in V \setminus \{s\}$).

$d(v) = dist(s, v)$ ←

1) Shortest paths ✓
2) Maximum flow ✓
3) Max matchings ✓
} tP

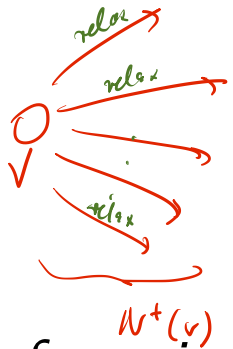
Bog 3

- To relax an edge (u, v):

  if $d(v) > d(u) + d_{uv}$*:* $d(v) := d(u) + d_{uv}$

- To relax a vertex v:

  for $w \in N^+(v)$*: relax*$((v, w))$



- Key insight (Dijkstra):

  (Greedily) Relaxing vertices in order of non-increasing current distances d *is optimal.*

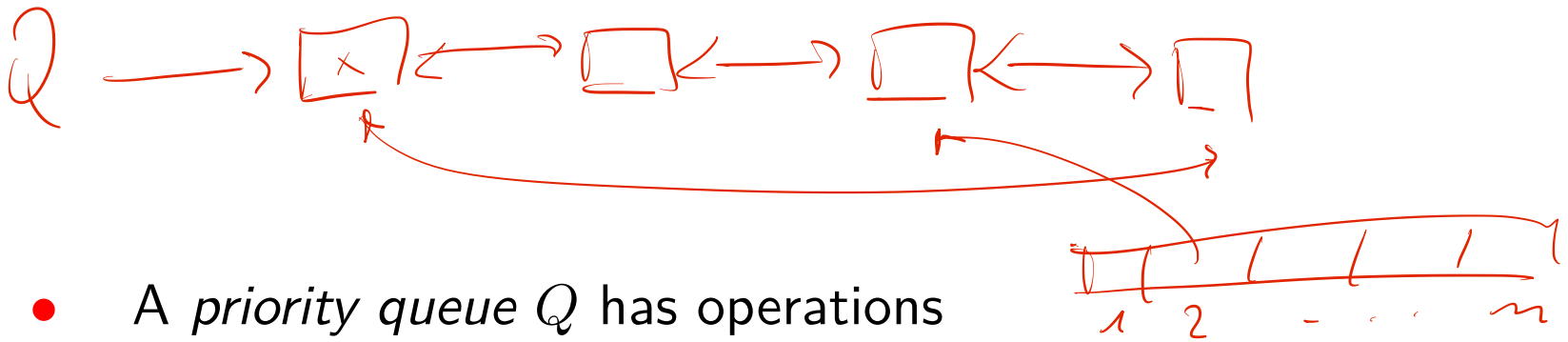
Once

n relaxations

Algorithm, high-level

all vertices unvisited; set initial distances
while there's a unvisited vertex
  select unvisited vertex v of minimal $d(v)$
  relax(v) and mark v as visited
end while

- Requires: data structure that finds the next vertex quickly.
- Otherwise: what's the complexity



- A *priority queue* Q has operations

  $Q := \emptyset$ create an empty priority queue

  push$(Q,(e,k))$ add element e with key k

  deletemin(Q) remove and return element of smallest key

  update$((e,k))$ set key of element e to k

Dijkstra

$O(n \cdot \text{deletemin} + m \cdot \text{insert} + m \cdot \text{update})$

push

$O(\log n)$ $O(1)$

$m > n$ connected

Sort n numbers:

$m \times$ insert
$m \times$ deletemin

$O(m \log n + m)$

$\Omega(n \log n)$

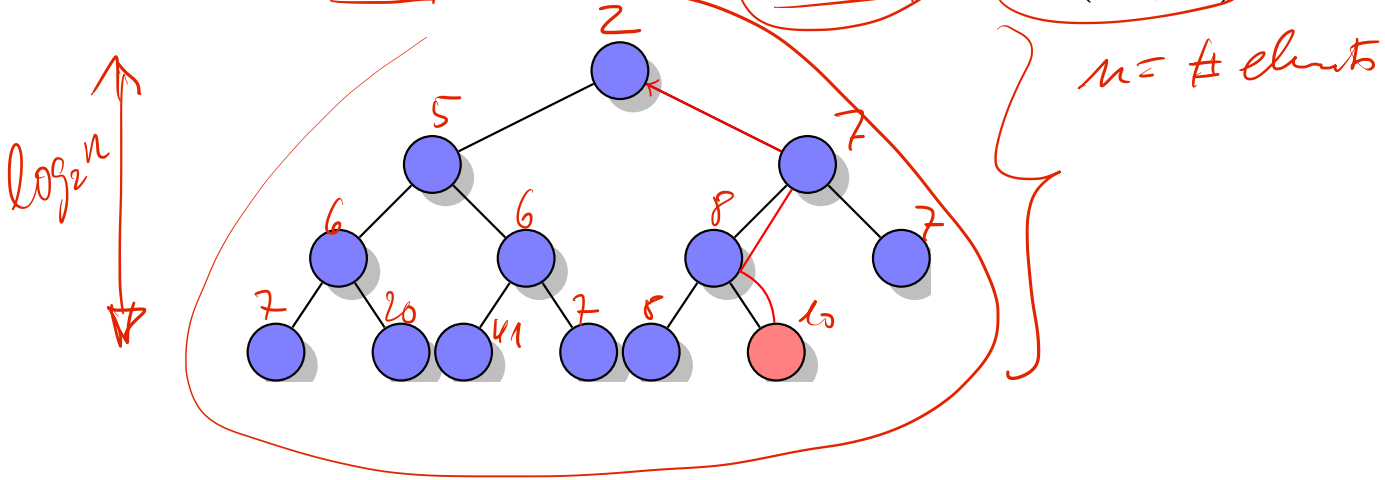
$O(n(\text{insert} + \text{deletemin}))$

$\log n$

$O(1)$ Fib-heap

$O(n \log n + m \log n) = O((n+m) \cdot \log n)$

- Binary heaps can do push, deletemin, update in $O(\log n)$.



Algorithm, details

$d_s := 0; d_v := \infty, \forall v \in V \setminus \{s\}$
$\text{visited}(v) := \text{false}, \forall v \in V$
$Q := \emptyset$; $\text{insert}(Q, (s, 0))$
while $Q \neq \emptyset$ *do*
  $v := \text{deletemin}(Q)$
  $\text{visited}(v) := \text{true}$
  for $u \in N^+(v) \mid \text{not}\text{visited}(u)$ *do*
    if $d_u = \infty$ *then*
      $d_u := d_v + d_{vu}$
      $\text{insert}(Q, (u, d_u))$
    else if $d_v + d_{vu} < d_u$
      $d_u := d_v + d_{vu}$
      $\text{update}(Q, (u, d_u))$
    end if
  end for
end while

n × deletemin

(*) $d(v) = dist(s, v)$

n × insert

m × update