COMPSCI 130B Discussion

01/21/2021

Kha-Dinh (Jacob) Luong

Office hours: F 9-11 AM

Objectives

- Go over the homework problems.

Q1. Coin Exchange

- Answer: Yes, greedy using coins that are powers of *c* is optimal.
- Proof: follow the same idea discussed in class because in this case, any coin value is divisible by the coin value preceding it.
- In any optimal solution:
  - Number of c^i coins is at most $c - 1$ for $i \in [0, k-1]$.
  - The maximum value that the optimal solution can be built with only coins c^0, c^1, ..., c^{i-1} is $c^i - 1$.
  - So for a value in $[c^i, c^{i+1} - 1]$, we must keep using coins c^i until the value drops below c^i, which is what Greedy does.

Q2. Maximum-reward Schedule

- Sort jobs by decreasing rewards.
- For job i with finish time t[i]: Schedule it at the latest time slot between 0 and t[i] or drop it if no such slot exists.

Q3. K-machine Interval Scheduling

- For each machine, assign a variable that keeps track of the latest finish time of the jobs that have been assigned to that machine. Process the jobs by increasing finishing time,

- For each job:
  - Pick the machine with the latest finishing time that is also compatible with the job (finishes before the job start time)
  - Update the finish time of that machine as the finish time of the job.
  - If there is no available machine, discard the job.

Q3. K-machine Interval Scheduling

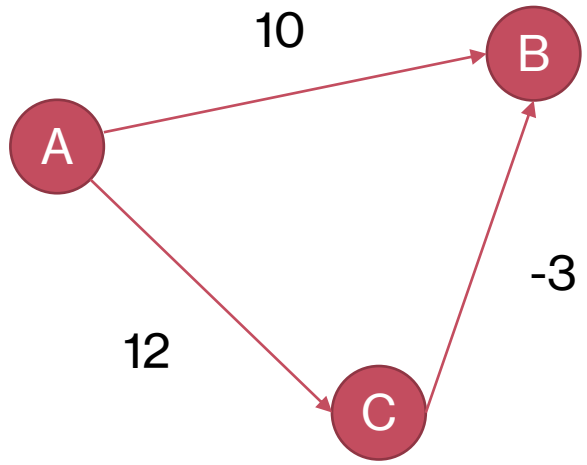
- We use an array S to represent the jobs in the order that they are considered by Greedy (increasing finish time). S[i] = k means job i is assigned to machine k and S[i] = 0 means the job is discarded. Note that f(i) < f(j) if i < j because of the definition of S.
- In fact, we can represent any schedule produced by any algorithm with S. Let S_G be the array by Greedy and S_O the schedule by another optimal algorithm. S_G[i] = k means job i is assigned to machine k by Greedy and S_O[i] = k' means job i is assigned to machine k' by the optimal algorithm.
- Induction hypothesis: $\forall i, \exists S\_O$ s.t S_G[0:i] = S_O[0:i]
- The intuition is that, at any step of Greedy, we are on a path to an optimal solution.

Q3. K-machine Interval Scheduling

- Base case: i = 0, Greedy has not assigned a job yet so any S_O can fill the remaining path.
- Induction step: assume ∃S_O s.t S_G[0:i] = S_O[0:i], consider S_G[i+1].
- If S_G[i+1] = 0, then S_O[i+1] is 0 as well because S_G[0:i] = S_O[0:i] so if job i+1 conflicts with the previous jobs 1 to i in S_G then it also conflicts with them in S_O. Therefore, S_G[0:i+1] = S_O[0:i+1],
- If S_G[i+1] = k,
  - If S_O[i+1] = k, this case is done.
  - If S_O[i+1] = k', notice that before scheduling i+1, f(k') on S_G ≤ f(k) on S_O because k' is the latest finishing machine according to Greedy. Therefore, we can swap all the schedules of k and k' on S_O for jobs that start after i without creating any conflict. Jobs 1 to i-1 on k and k' are not affected because according to Greedy, they all finish earlier than when job i starts. We have made S_O[i+1] = k by this swapping.
  - If S_O[i+1] = 0, then there is a job j that conflicts with i+1 on k in S_O. This j cannot belong to [1:i] because Greedy ensures that there is no conflict before i. Notice that j is unique, meaning i+1 only conflicts with this j and no other job after i+1 because according to the way the array is built, f(j) > f(i+1), and i+1 cannot conflict with more than one job that finishes after it. We can then make S_O[i+1] = k and S_O[j] = 0 while keeping S_O optimal.

Q4. Dijkstra on Negative Edges

- Counter example: A as the starting node.
  - Dijkstra's output: A->B
  - Optimal: A->C->B



Q5. Minimum Spanning Tree

- Show by using Kruskal's Algorithm.
- Observe that the k minimum edges cannot form a cycle because if they do, they cannot span k+1 vertices.
- With that, Kruskal naturally finds the MST that include all the k minimum edges.

Q6. Clustering

- Find the smallest and largest values (O(n)) and divide the range into n+1 equal-size intervals.
- Pigeonhole Principle -> There are empty intervals.
- The largest separation must span over empty intervals. (Because of this, we never need to sort the values).
- For each interval, calculate the min distances of its elements to the left and right (no sorting needed, and it's O(2n)).
- Use this information to determine the largest separation.

Q7. Spider and Rafters

- Greedy: the spider traveling to the furthest rafter that it can reach within its limit for the day.
- Proof:

Induction. Greedy stays ahead: after n-th choice, the path traveled by the spider according to Greedy is no shorter than that of any other algorithm.

Base case: Consider the first choice by the greedy algorithm and the first choice by another optimal solution. The rafter chosen by the optimal solution must be the same or before the one chosen by the greedy algorithm. If that choice is after the one chosen by the greedy algorithm, then it's either that one is beyond the reach of the spider or greedy would have chosen it.

Induction step: Assume greedy stays ahead at the k-th choice. It's easy to see that whichever k+1-th raft that the optimal solution chooses is also within the reach of the k-th raft chosen by the greedy algorithm, but not vice versa. So greedy will stay ahead at the k+1-th choice.

Q8. Caching

- A: 9 cache misses.
  - **2, 3, 4, 1, 5,** 6, 8, 2, 4, 5, 1, 6, 8, 2, 4, 1, 3, 6
  - 2, 3, 4, 1, 5 each gives one miss.
  - 6 gives another miss + one of the cached elements must be removed.
  - Another miss when the removed element appears again.
  - Another way to prove is using Farthest in the Future algorithm, which has been proven to be optimal. The number of cache misses is 9 and thus is the lower bound.

- B: 16