CompSci 260P: Week 7

Greedy Algorithms

Ryuto Kitagawa

University of California, Irvine

Before We Begin

- Greedy algorithms make *greedy* choices
  - Making optimal decisions based on local information
- Recall in Dynamic Programming, we break the problem down into making a single choice
  - Dynamic programming looks at the choice and uses recursion to find the optimal
  - Greedy algorithms instead make the optimal choice without recursion
- The burden of greedy algorithms is in the **proof**!

Before We Begin

Theorem 1

Testing

Algorithmic Pizza

- You are the manager and have n pizza orders
- Making a pizza is divided in two stages:
  i. Preparation
  ii. Baking
- Order i takes p_i time in prep and b_i time in baking
- Your oven is infinitely large, but you can only prep one pizza at a time
- What order should you prep the pizza such that the last pizza comes out of the oven as soon as possible?

Algorithmic Pizza: Solution

- It is tempting to start with an algorithm, then prove it
  - However, proofs can guide our algorithms instead
- Notice this is a *permutation* problem
  - Therefore, what happens if we swap adjacent pizza orders?
- Let's look at two consecutive orders i and $i + 1$

Algorithmic Pizza: Solution

- Let s_i be how long it takes us to get to order i
  - How long does pizza i and $i + 1$ take to get out of the oven?
  - What if we were to flip it?
- How do we then use this information to find the optimal ordering?

Algorithmic Pizza: Solution

- Consider if pizza i is prepared then pizza $i + 1$
  - Pizza i comes out after: $s_i + p_i + b_i$
  - Pizza $i + 1$ comes out after: $s_i + p_i + p_{i+1} + b_{i+1}$
- Consider if pizza $i + 1$ is prepared then pizza i
  - Pizza i comes out after: $s_i + p_i + p_{i+1} + b_i$
  - Pizza $i + 1$ comes out after: $s_i + p_{i+1} + b_{i+1}$

Algorithmic Pizza: Solution

- Let's see which pizza comes out earlier in the first configuration
  - Which is larger: $s_i + p_i + b_i$ or $s_i + p_i + p_{i+1} + b_{i+1}$
  - We can simplify the above by canceling like terms: b_i or $p_{i+1} + b_{i+1}$
  - The above depends on the value of the variables, so let's consider when $b_i < b_{i+1}$
  - Then we know that $b_i < p_{i+1} + b_{i+1}$, meaning pizza $i + 1$ takes longer
  - Therefore, we take the time it takes for the longer pizza and compare it to the others

Algorithmic Pizza: Solution

- Therefore, we compare $s_i + p_i + p_{i+1} + b_{i+1}$ to the times for when pizza $i + 1$ is prepared first
- Let's first compare with the time of pizza i (when pizza $i + 1$ is prepared first)
  - $s_i + p_i + p_{i+1} + b_{i+1}$ and $s_i + p_i + p_{i+1} + b_i$
  - Simplified for comparison we get, b_{i+1} and b_i
  - So we know that preparing pizza i first takes longer in this comparison

Algorithmic Pizza: Solution

- Now let's compare with the time of pizza $i + 1$ (when pizza $i + 1$ is prepared first)
  - $s_i + p_i + p_{i+1} + b_{i+1}$ and $s_i + p_{i+1} + b_{i+1}$
  - Simplified for comparison we get, $p_i + b_{i+1}$ and b_{i+1}
  - So again, preparing pizza i first takes longer here as well
- **Conclusion:** When $b_i < b_{i+1}$, preparing pizza i first *always* takes longer
- The same logic follows for $b_i > b_{i+1}$

Interval Scheduling Approximation

- Given a set of intervals, find the largest subset of non-overlapping intervals
- Algorithm:
  - Pick the shortest interval
  - Discard any overlapping intervals
  - Repeat until we have no more intervals
- Prove this algorithm is a 2-approximation

Interval Scheduling Approximation

- Consider the optimal solution and compare it to our own
  - What properties does the optimal solution have compared to our own?
  - The magic number to look for is 2
- What happens when the optimal solution has more intervals than the greedy
  - Consider an interval that invalidates more than 1 interval
  - Can it invalidate more than 2?
  - No, because of the middle interval!

Interval Scheduling Approximation

- For each interval, it can overlap at most 2 optimal intervals
- Therefore, our solution is at most half of the optimal solution
- Thus making it a 2 approximation