Review Session

130B Midterm

Master’s theorem

T (n) = a T(n/b) + f(n)

- Case 1: f(n) = O($n^{log\_b\ (a-epsilon)}$) for some epsilon > 0
  - T (n) = ϴ($n^{log\_b\ (a)}$) ... f(n) grows polynomially slower than $n^{log\_b\ (a)}$
- Case 2: f(n) = O($n^{log\_b\ (a)}$)
  - T(n) = ϴ($n^{log\_b\ (a)}$ log n) … f(n) grows at the same rate as $n^{log\_b\ (a)}$
- Case 3: f(n) = O($n^{log\_b\ (a+epsilon)}$) for some epsilon > 0
  - T(n) = ϴ(f(n)) … f(n) satisfies regularity constraint -- a f(n/b) <= c f(n) for c < 1 and large n

Recurrences

1. $T(n) = 2\,T(n/3) + O(n)$

2. $T(n) = O(n) + 4\,T(n/2) + O(n^{1.99})$

3. $T(n) = T(n-1) + O(\log n)$

Solutions

1. $T(n) = 2\,T\,(n/3) + O(n)$

   $a = 2 \quad b = 3 \quad c = 1$

   Case 3 => $O(n)$

2. $T(n) = O(n) + 4\,T(n/2) + O(n^{1.99})$
   $a = 4 \quad b = 2 \quad c = 1.99$

   Case 1 => $O(n^2)$

3. $T(n) = T(n\text{-}1) + O(\log n)$

   n - levels with log n work at each level

   => $O(n \log n)$

How about $T(n) = T(n/2) + 2T(n/3)$?

DYNAMIC PROGRAMMING

Coin Exchange with coins {1,5,12}

Is it solvable using Greedy?

DP relations: D[N] = min (D[N-1], D[N-5], D[N-12])

DP is like brute-force with memorization:

- Brute-force: considers all possible “paths” (combinations of subproblems) to the solution.
- Memorization: records all the overlappings of the paths so they are only calculated once.
- In Divide and Conquer: We choose a set of subproblems that do not overlap.

DYNAMIC PROGRAMMING

A harder problem:

Longest Common Subsequence: Given an integer array A and B, design an algorithm to return the length of the longest common subsequence (LCS) of A, B. For an array A, a subsequence of A is an ordered subset of elements of A with a monotonically increasing index.

A = {5,2,1,-2,4,6}

B = {0,2,5,1,-2,4,6}

Output: 5

LCS: {5,1,-2,4,6}

What is the recurrence relation in this case? How to define subproblems?

DYNAMIC PROGRAMMING

Let D[i][j] be the solution to the subproblem with arrays A[0:i] and B[0:j],

```
If A[i] = B[j]:
    D[i][j] = D[i-1][j-1]+1
Else:
    D[i][j] = max (D[i][j-1], D[i-1][j])
```

GREEDY

Given 2 unordered lists of positive integers:

A = [a1, a2, ... , an]

B = [b1, b2, ... , bn]

Reorder B such that minimizes Σ ai*bi

GREEDY

Solution: Sort one list in ascending order and the other one in descending order and pair elements with the same position. Basically, pair the smallest element from one list with the largest element from the other list.

Any other solution is just another pairing of elements.

In this case, should we prove using Exchange Argument or Greedy Stay Ahead?

What is the intuition?

GREEDY

Greedy Stay Ahead:

- Define the form of the solution following the steps taken by greedy (g1, g2, …). Make sure other solutions can be represented similarly (o1,o2,...).
- Find a “measure” that estimates the optimality of a solution.
- Prove that the first part of greedy (g1, g2, …, gk) always equal or better than that of another solution (o1,o2,...,om) by induction. Usually, k=m.
- At the end, as proven using induction, greedy will be equal or better than any other solution by the measure.

GREEDY

Exchange Argument:

- Define the form of the solution by greedy G = (g1, g2, …,gk). Make sure other solutions can be represented similarly O = (o1,o2,...,om).
- Consider the differences between 2 solutions:
    - There is an element of G not in O and an element of O not in G.
    - There is a pair of elements of O that are in a different order than they are in G.
- In any case, swap the elements in an attempt to make O one step closer to G and proves that doing so does not worsen O (it only improves).

Make sure you review these concepts because the problems may be variants of them:

- Linear time search.
- Tangent finding for merging
- Closest pair
- Interval (Job) scheduling
- Huffman encoding.
- Matrix Multiplication
- Least Square Problem.