CS6515 Exam 1 Notes

Dynamic Programming

Common Archetypes

- LIS
  - L(i) = length of LIS in a_i that included a_i
  - L(i) = I+max j over {L(j): aj < ai & j < i}

- LCS
  - L(I, j) = length of LCS in x_i and y_j
  - L(I,j) = if x_i=y_j, 1+L(I-1, j-1)
    if x_i =/= y_j, max {L(I-1, J), L(I, J-1)}

- Knapsack
  - No repetition
    - K(I, w) = maximum value achievable with up to x_i items and capacity w
    - K(I, w) = Max (K(i-1, w), k(I-1, w - w_i) + v_i)
  - Repetition
    - K(w) = maximum value achievable with capacity w
    - K(w) = max over all w_i <= w {K(w-w_i)+v_i}

- Substrings
  - Matrix Multiplication
    - C(I,j) = cost of multiplying A_i, A_i+1, A_i+2,..., A_j-1 A_j
    - Base case: I = j, c(I,j) = 0
    - C(I, j) = min over I <=k < j {C(I,k) + C(k+1, j)+ m_{i-1} * m_{k} * m_{j}}
    - Increase the width at each iteration
    - Look to find the best breakpoint between I and j
    - recurrence is always of the form min or max over {C(I, k) + C(k+1, j)}

Pseudo Code Conventions

```
For I = 1 to n
    T[I]=T[I]
If [condition]:
    [logic]
Else:
    [logic]
Max(T[.])
```

Graph Algorithms

BFS
- Queue
- O(|V|+|E|)

DFS
- Stack
- O(|V|+|E|)

Dijkstra's
- BFS with priority queue
- O(|E|*Tdk +|V|*Tem)

Bellman-Ford
- Dijkstra but updates all edges V-1 times
- O(|V| * |E|)
- Detects negative cycles

Floyd-Warshall
- Recursively finds the shortest path through k between I and j
- O(|V|^3)

Other Graph Formulas
- Max edges = n * (n-1) / 2 undirected, double for directed

Divide and Conquer

Common Archetypes

- Binary Search
  - O(log n)
- Merge Sort
  - O(n log n)
- Median of Median
  - O(n)
- Fast Select
  - O(n)

Fast Multiplication

x*y = 2^n*xl*yl + 2^(n/2)*(xl*yr +XR*yl) + xryr

Solving Recurrence

T(n) = aT([n/b]) + O(n^d)
T(n) = { O(n^d) for d > log_b(a),
          O((n^d)*log n) for d = log_b(a),
          O(n^log_b(a)) for d < log_b(a)}

a^(log base a N) = N

n+(n-1)...+1 = n(n+1)/2

Big O

F = O(g), f grows no faster than g, f(n) <= c * g(n)
F = shell(g), g grows no faster than f
F = theta (g) f = g

FFT

Roots of Unity

- Complex num X = r cos theta, y = r sin theta
- Forms = e ^ 2*pi*k/n, kth element of the nth roots of unity
- Sum is 1 if if n > 0, if n =0, 1

Core formula:
A(x) = A_e(x^2) + x * A_o(x^2)

<Values> = FFT (<Coeffients>, w)
<Coefficient> = 1/n FFT (<values>, w^-1)

C(x) = A(x)B(x) = {A(x1)B(x1), A(x2)B(x2), ..., A(xi)B(xi)}

Reduction to FFT

- Represent the problem as some kind of polynomial multiplication problem