



Programming, Data Structures and Algorithms using Python

Prof. Madhavan Mukund

Director

Chennai Mathematical Institute

Mr. Omkar Joshi

Course Instructor

IITM Online Degree Programme



Programming, Data Structures and Algorithms using Python

Summary of weeks 7 to 9

Content

- Greedy algorithms
 - Interval scheduling
 - Minimizing lateness
 - Huffman coding
- Divide and conquer
 - Counting inversions
 - Closest pair of points
 - Integer multiplication
 - Quick select
 - Recursion trees
- Dynamic programming
 - Memoization
 - Grid paths
 - Common subwords and subsequences
 - Edit distance
 - Matrix multiplication

Greedy algorithms

- Need to make a sequence of choices to achieve a global optimum
- At each stage, make the next choice based on some local criterion
- Never go back and revise an earlier decision
- Drastically reduces space to search for solutions
- Greedy strategy needs a proof of optimality
- Examples: Dijkstra's algorithm, Prim's algorithm, Kruskal's algorithm
- Applications: Interval Scheduling, Minimizing Lateness, Huffman Coding

Interval scheduling

- IIT Madras has a special video classroom for delivering online lectures
 - Different teachers want to book the classroom
 - Slots may overlap, so not all bookings can be honored
 - Choose a subset of bookings to maximize the number of teachers who get to use the room
1. Choose the booking whose starting time is earliest
 2. Choose the booking spanning the shortest interval
 3. Choose the booking that overlaps with minimum number of other bookings
 4. Choose the booking whose finish time is the earliest

Minimizing lateness

- IIT Madras has a single 3D printer
 - A number of users need to use this printer
 - Each user will get access to the printer, but may not finish before deadline
 - Goal is to minimize the lateness
1. Schedule requests in increasing order of length
 2. Give priority to requests with smaller slack time
 3. Schedule requests in increasing order of deadlines

Huffman coding

- At each recursive step, extract letters with minimum frequency and replace by composite letter with combined frequency
- Store frequencies in an array A
 - Linear scan to find minimum values
 - $|A| = k$, number of recursive calls is $k - 1$
 - Complexity is $O(k^2)$
- Instead, maintain frequencies in an heap
 - Extracting two minimum frequency letters and adding back compound letter are both $O(\log k)$
 - Complexity drops to $O(k \log k)$

Divide and conquer

- Break the problem into disjoint subproblems
- Combine these subproblem solutions efficiently
- Examples: Merge sort, Quicksort
- Applications: Counting inversions, Closest pair of points, Integer multiplication, Quick select

Counting inversions

- Compare your profile with other customers
- Identify people who share your likes and dislikes
- No inversions – rankings identical
- Every pair inverted – maximally dissimilar
- Number of inversions ranges from 0 to $n(n - 1) / 2$
- An inversion is a pair (i, j) , $i < j$, where j appears before i
- Recurrence: $T(n) = 2T(n / 2) + n = O(n \log n)$

Closest pair of points

- Split the points into two halves by vertical line
- Recursively compute closest pair in each half
- Compare shortest distance in each half to shortest distance across the dividing line
- Recurrence: $T(n) = 2T(n / 2) + O(n)$
- Overall: $O(n \log n)$

Integer multiplication

- Traditional method: $O(n^2)$
- Naïve divide and conquer strategy: $T(n) = 4T(n / 2) + n$
 $= O(n^2)$
- Karatsuba's algorithm: $T(n) = 3T(n / 2) + n$
 $= O(n^{\log 3})$
 $\approx O(n^{1.59})$

Quick select

- Find the k^{th} largest value in a sequence of length k
- Sort in descending order and look at position k – $O(n \log n)$
- For any fixed k , find maximum for k times – $O(kn)$
 - $k = n / 2$ (median) – $O(n^2)$
- Median of medians – $O(n)$
- Selection becomes $O(n)$ – Fast select algorithm
- Quicksort becomes $O(n \log n)$

Recursion trees

- Uniform way to compute the asymptotic expression for $T(n)$
- Rooted tree with one node for each recursive subproblem
- Value of each node is time spent on that subproblem excluding recursive calls
- Concretely, on an input of size n
 - $f(n)$ is the time spent on non-recursive work
 - r is the number of recursive calls
 - Each recursive call works on a subproblem of size n / c
- Recurrence: $T(n) = rT(n / c) + f(n)$
- Root of recursion tree for $T(n)$ has value $f(n)$
- Root has r children, each (recursively) the root of a tree for $T(n / c)$
- Each node at level d has value $f(n / c^d)$

Dynamic programming

- Solution to original problem can be derived by combining solutions to subproblems
- Examples: Factorial, Insertion sort, Fibonacci series
- Anticipate the structure of subproblems
 - Derive from inductive definition
 - Dependencies form a DAG
- Solve subproblems in topological order
 - Never need to make a recursive call

Memoization

- Inductive solution generates same subproblem at different stages
- Naïve recursive implementation evaluates each instance of subproblem from scratch
- Build a table of values already computed – Memory table
- Store each newly computed value in a table
- Look up the table before making a recursive call

Grid paths

- Rectangular grid of one-way roads
- Can only go up and right
- How many paths from $(0, 0)$ to (m, n) ?
- Every path has $(m + n)$ segments
- What if an intersection is blocked?
- Need to discard paths passing through
- Identify DAG structure
 - Fill the grid by row, column or diagonal
- Complexity is $O(mn)$ using dynamic programming, $O(m + n)$ using memoization

Common subwords and subsequences

- Given two strings, find the (length of the) longest common subword
- Subproblems are $LCW(i, j)$, for $0 \leq i \leq m, 0 \leq j \leq n$
- Table of $(m + 1)(n + 1)$ values
- $LCW(i, j)$, depends on $LCW(i + 1, j + 1)$
- Start at bottom right and fill row by row or column by column
- Complexity: $O(mn)$

		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b	0	0	0	0	0	0	0
1	i	0	0	0	0	0	0	0
2	s	3	0	0	0	0	0	0
3	e	0	2	0	0	1	0	0
4	c	0	0	1	0	0	0	0
5	t	0	0	0	0	0	1	0
6	•	0	0	0	0	0	0	0

Common subwords and subsequences

- Subsequence – can drop some letters in between
- Subproblems are $LCS(i, j)$, for $0 \leq i \leq m, 0 \leq j \leq n$
- Table of $(m + 1)(n + 1)$ values
- $LCS(i, j)$, depends on $LCS(i + 1, j + 1)$, $LCS(i, j + 1)$, $LCS(i + 1, j)$
- Start at bottom right and fill row by row, column or diagonal
- Complexity: $O(mn)$

		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b	4	3	2	2	2	1	0
1	i	4	3	2	2	2	1	0
2	s	4	3	2	2	2	1	0
3	e	3	3	2	2	2	1	0
4	c	2	2	2	1	1	1	0
5	t	1	1	1	1	1	1	0
6	•	0	0	0	0	0	0	0









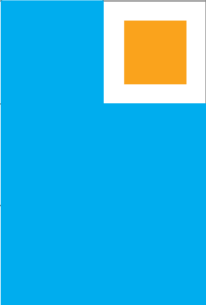






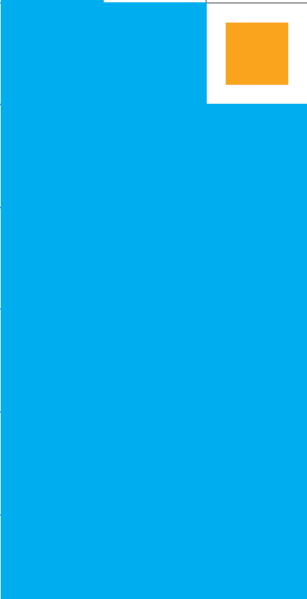
















Edit distance

- Minimum number of editing operations needed to transform one document to the other
- Subproblems are $ED(i, j)$, for $0 \leq i \leq m, 0 \leq j \leq n$
- Table of $(m + 1)(n + 1)$ values
- $ED(i, j)$, depends on $ED(i + 1, j + 1)$, $ED(i, j + 1)$, $ED(i + 1, j)$
- Start at bottom right and fill row, column or diagonal
- Complexity: $O(mn)$

		0	1	2	3	4	5	6
		s	e	c	r	e	t	•
0	b	4	4	4	4	4	5	6
1	i	3	4	3	3	3	4	5
2	s	2	3	3	2	2	3	4
3	e	3	2	3	2	1	2	3
4	c	4	3	2	2	1	1	2
5	t	5	4	3	2	1	0	1
6	•	6	5	4	3	2	1	0

Matrix multiplication

- Matrix multiplication is associative
- Bracketing does not change answer but can affect the complexity
- Find an optimal order to compute the product
- Compute $C(i, j)$, $0 \leq i, j < n$, only for $i \leq j$
- $C(i, j)$, depends on $C(i, k - 1)$, $C(i, k)$ for every $i < k \leq j$
- Diagonal entries are base case, fill matrix from main diagonal
- Complexity: $O(n^3)$

	0	...	i	j	...	$n-1$	
0									
...									
i									
...									
...									
j									
...									
$n-1$						