23CY603
Algorithm Design and Analysis

Dr. Manne Venu,
Assistant Professor,
Computer Science and Engineering
(CYBER SECURITY)

UNIT-I

Introduction to Algorithm

- An *Algorithm* is a sequence of unambiguous instructions for solving a problem,
- i.e., for obtaining a required output for any legitimate input in a finite amount of time.

PSEUDOCODE

- Pseudocode (pronounced SOO-doh-kohd) is a detailed yet readable description of what a computer program or algorithm must do, expressed in a formally-styled natural language rather than in a programming language.

- It is sometimes used as a detailed step in the process of developing a program.

- It allows programmers to express the design in great detail and provides programmers a detailed template for the next step of writing code in a specific programming language.

Formatting and Conventions in Pseudocoding

- INDENTATION in pseudocode should be identical to its implementation in a programming language. Try to indent at least four spaces.
- The pseudocode entries are to be cryptic, AND SHOULD NOT BE PROSE. NO SENTENCES.
- No flower boxes in pseudocode.
- Do not include data declarations in pseudocode.

Some Keywords That Should be Used

- As verbs, use the words
  - generate, Compute, Process,
  - Set, reset,
  - increment,
  - calculate,
  - add, sum, multiply, …
  - print, display,
  - input, output, edit, test , etc.

Methods of finding GCD

Competition

Computing Greatest Common Divisor: gcd(m,n)

Primary School

1. $t := \min(m, n)$
2. $m \bmod t = 0$?
3. Yes? $n \bmod t = 0$? Return t
4. No? $t = t\text{-}1$; goto 2

Secondary School

1. Find prime factors of m
2. Find prime factors of n
3. Identify common factors
4. Return product of these

University

1. $n = 0$?
2. Yes? Return m
3. $r = m \bmod n$, $m := n$, $n := r$
4. Go to 1

Euclid

Euclid's Algorithm

Problem: Find gcd(m,n), the greatest common divisor of two
nonnegative, not both zero integers m and n

Examples: gcd(60,24) = 12, gcd(60,0) = 60,
gcd(0,0) = ?

Euclid's algorithm is based on repeated application of
equality gcd(m,n) = gcd(n, m mod n)
until the second number becomes 0, which makes the problem
trivial.

Example: gcd(60,24) = gcd(24,12) = gcd(12,0) = 12

Two descriptions of Euclid's algorithm

Step 1 If $n = 0$, return m and stop; otherwise go to Step 2

Step 2 Divide m by n and assign the value fo the remainder to r

Step 3 Assign the value of n to m and the value of r to n. Go to Step 1.

while $n \neq 0$ do $r \leftarrow m \bmod n$ $m \leftarrow n$

$n \leftarrow r$

return m

Divide and Conquer

Sorting: mergesort and quicksort

- Tree traversals
- Binary search
- Matrix multiplication-Strassen's algorithm

UNIT-II

Disjoint Sets

- Some applications require maintaining a collection of disjoint sets.
- A Disjoint set S is a collection of sets
- S_1,.....SW_n $\quad \forall_{i \neq j} S_i \cap S_j = \phi$
- Each set has a representative which is a member of the set (Usually the minimum if the elements are comparable)

Disjoint Set Operations

- Make-Set(x) – Creates a new set where x is only element (and therefore it is the representative of the set). O(1) time
- Union(x,y) – Replaces one of the elements of containing x
- Find(x) – Returns the representative of the set O(log n) time

Analyzing Operations

- We usually analyze a sequence of *m* operations, of which *n* of them are Make Set operations, and *m* is the total of Make Set, Find, and Union operations

- Each union operations decreases the number of sets in the data structure, so there can not be more than *n-1* Union operations

Applications

- Equivalence Relations (e.g Connected Components)
- Minimal Spanning Trees

Connected Components

- Given a graph G we first preprocess G to maintain a set of connected components.

CONNECTED_COMPONENTS(G)

- Later a series of queries can be executed to check if two vertexes are part of the same connected component

SAME_COMPONENT(U,V)

Connected Components

- During the execution of CONNECTED- COMPONENTS on a undirected graph $G = (V, E)$ with k connected components, how many time is FIND-SET called? How many times is UNION called? Express you answers in terms of $|V|$, $|E|$, and k.

Solution

- FIND-SET is called 2|E| times. FIND-SET is called twice on line 4, which is executed once for each edge in E[G].

- UNION is called |V| - k times. Lines 1 and 2 create

|V| disjoint sets. Each UNION operation decreases the number of disjoint sets by one. At the end there are k disjoint sets, so UNION is called |V| - k times.

Linked List implementation

- We maintain a set of linked list, each list corresponds to a single set.
- All elements of the set point to the first element which is the representative
- A pointer to the tail is maintained so elements are inserted at the end of the list

UNIT-III

DYNAMIC PROGRAMMING

Dynamic programming is a technique for solving problems with overlapping sub problems.

Typically, these sub-problems arise from a recurrence relating a solution to a given problem with solutions to its smaller sub- problems of the same type.

Dynamic programming usually takes one of two approaches:

Bottom-up approach: All subproblems that might be needed are solved in advance and then used to build up solutions to larger problems. This approach is slightly better in stack space and number of function calls, but it is sometimes not intuitive to figure out all the subproblems needed for solving the given problem.

Top-down approach: The problem is broken into subproblems, and these subproblems are solved and the solutions remembered, in case they need to be solved again. This is recursion and Memory Function combined together

Examples of DP algorithms.

- Computing a binomial coefficient
- Longest common subsequence
- Warshall's algorithm for transitive closure
- Floyd's algorithm for all-pairs shortest paths
- Constructing an optimal binary search tree
- Some instances of difficult discrete optimization problems:
  - traveling salesman
  - knapsack

Memory Function.

In general, we cannot expect more than a constant-factor gain in using the memory function method for the knapsack problem because its time efficiency class is the same as that of the bottom-up algorithm

A memory function method may be less space-efficient than a space efficient version of a bottom-up algorithm

UNIT-IV

Greedy Algorithm

Similar to dynamic programming, but simpler approach

- Also used for optimization problems

Idea: When we have a choice to make, make the one that looks best right now

- Make a locally optimal choice in hope of getting a globally optimal solution

Greedy algorithms don't always yield an optimal solution

Makes the choice that looks best at the moment in order to get optimal solution.

.

Fractional Knapsack Problem

Knapsack capacity: W

There are n items: the i-th item has value v_i and weight w_i

Goal:

find x_i such that for all $0 \leq x_i \leq 1, \quad i = 1, 2, .., n$

$\sum w_i x_i \leq W$ and

$\sum x_i v_i$ is maximum

Fractional Knapsack Problem

Greedy strategy 1:

Pick the item with the maximum value

E.g.:

$W = 1$

$w_1 = 100$, $v_1 = 2$

$w_2 = 1$, $v_2 = 1$

Taking from the item with the maximum value:

Total value taken = $v_1/w_1 = 2/100$

Smaller than what the thief can take if choosing the other item

Total value (choose item 2) = $v_2/w_2 = 1$

Huffman Code Problem

Huffman's algorithm achieves data compression by finding the best variable length binary encoding scheme for the symbols that occur in the file to be compressed.

Huffman Code Problem

In the pseudocode that follows:

we assume that C is a set of n characters and that each character c €C is an object with a defined frequency f [c].

The algorithm builds the tree T corresponding to the optimal code

A min-priority queue Q, is used to identify the two least-frequent objects to merge together.

The result of the merger of two objects is a new object whose frequency is the sum of the frequencies of the two objects that were merged

UNIT-V

Branch-and-Bound Technique
Basic Concepts

The basic concept underlying the branch-and-bound technique is to *divide and conquer*.

Since the original "large" problem is hard to solve directly,
it is *divided* into smaller and smaller subproblems
until these subproblems can be *conquered*.

The *dividing (branching)* is done by partitioning the entire set of feasible solutions into smaller and smaller subsets.

The *conquering (fathoming)* is done partially by
(i) giving a *bound* for the best solution in the subset;
(ii) discarding the subset if the bound indicates that it can't contain an optimal solution.

These three basic steps
– branching, bounding, and fathoming –
are illustrated on the following example.

Utilizing the information about the optimal solution of the LP-relaxation

➢*Fact*: If LP-relaxation has integral optimal solution x*, then x* is optimal for IP too.

▪In our case, $(x_1, x_2) = (2.25, 3.75)$ is the optimal solution of the LP-relaxation. But, unfortunately, it is *not* integral.

The optimal value is 41.25 .

➢*Fact*: OPT(LP-relaxation) ≥ OPT(IP)
(for maximization problems)

That is, the optimal value of the LP-relaxation is an *upper bound* for the optimal value of the integer program.

In our case, 41.25 is an upper bound for OPT(IP).

Branching step

In an attempt to find out more about the location of the IP's optimal solution, *partition* the feasible region of the LP-relaxation.
Choose a variable that is fractional in the optimal solution to the LP-relaxation – say, x_2 . Observe that every feasible IP point must have either $x_2 \leq 3$ or $x_2 \geq 4$.
With this in mind, branch on the variable x_2 to create the following two subproblems:

Subproblem 1

$$\text{Max } Z = 5x_1 + 8x_2$$
$$\text{s.t. } x_1 + x_2 \leq 6$$
$$5x_1 + 9x_2 \leq 45$$
$$x_2 \leq 3$$
$$x_1 , x_2 \geq 0$$

Subproblem 2

$$\text{Max } Z = 5x_1 + 8x_2$$
$$\text{s.t. } x_1 + x_2 \leq 6$$
$$5x_1 + 9x_2 \leq 45$$
$$x_2 \geq 4$$
$$x_1 , x_2 \geq 0$$

Solve both subproblems
(note that the original optimal solution (2.25, 3.75) can't recur)

Solution tree

- If a subproblem is infeasible, then it is fathomed.
  In our case, Subproblem 4 is infeasible; fathom it.

- The upper bound for OPT(IP) is updated: $39 \leq OPT(IP) \leq 40.55$.

- Next branch Subproblem 3 on x_2 .
  (Note that the branching variable might recur).

Solution tree

- If the optimal value of a sub problem is $\leq Z^*$, then it is fathomed.
  - In our case, Sub problem 5 is fathomed because $37 \leq 39 = Z^*$.

- If a sub problem has integral optimal solution x*, and its value > Z*, then x* replaces the current incumbent.
  - In our case, Sub problem 5 has integral optimal solution, and its value 40>39=Z*. Thus, (0,5) is the new incumbent, and new Z*=40.
- If there are no unfathomed subproblems left, then the current incumbent is an optimal solution for (IP).
  - In our case, (0, 5) is an optimal solution with optimal value 40.

Branch and Bound Example

Solve the following BIP be branch and bound.

$$\text{Max } Z = 8x_1 + 11x_2$$
$$\text{s.t. } 5x_1 + 7x_2 \leq 14$$
$$0 \leq x_1 , x_2 \leq 1 \quad \text{integer}$$


THANK
YOU