Introduction

CS3230: Design and Analysis of Algorithms

Roger Zimmermann

National University of Singapore

Spring 2017

Chapter 1: Introduction

Topics: What we will cover today

- What is an algorithm?
- Fundamentals of algorithmic problem solving
- Important problem types
- Fundamental data structures

Why Study Algorithms?

- To know important existing standard algorithms, e.g., quicksort.
- To learn the design and analysis of algorithms for new problems.
- To sharpen analytical skills.
- To realize that some problems are unlikely to have fast solutions and to recognize them.

$\Rightarrow$ What is the definition of an algorithm?

Is this an Algorithm: Angel Food Cake Recipe?

Angel Food Cake with Toasted Almonds

SUBMITTED BY: Clayton McKee PHOTO BY: Chef Remi

"Homemade Angel Food Cake with almonds added."

RECIPE RATING:

★★★★½

Read Reviews (4)

Review/Rate This Recipe

Original recipe yield 1 - 10 inch tube pan

SERVINGS (Help)

14 Servings Calculate

US METRIC

INGREDIENTS (Nutrition)

- 1 1/4 cups confectioners' sugar
- 1 cup cake flour
- 1 1/2 cups egg whites
- 1/4 teaspoon salt
- 1 teaspoon cream of tartar
- 1 1/2 teaspoons vanilla extract
- 1/4 teaspoon almond extract
- 1 cup white sugar
- 1 cup chopped almonds

DIRECTIONS

1. Preheat oven to 375 degrees F (190 degrees C). Sift the confectioners sugar and cake flour together and set aside.
2. In a large bowl, whip the egg whites on high speed until foamy. Continue to whip on high speed while adding salt, cream of tartar, vanilla and almond extract. Gradually add the white sugar and continue to whip until whites have stiff peaks. Quickly fold in the flour mixture 1/4 cup at a time. Mix only until flour is incorporated. Do not deflate egg whites.
3. Pour batter into a 10 inch tube pan. Sprinkle top with chopped almonds. Bake at 375 degrees F (175 degrees C) for 35 to 40 minutes, or until top of cake bounces back when lightly tapped. Remove from oven, invert pan and allow to cool in the pan. When cake Is cool, loosen sides with a long knife and remove to a serving dish.

What are Algorithms?

- "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."

- Other descriptions also exist.

Characteristics of Algorithms

- Nonambiguity: each step can be carried out precisely.

  Note: nonambiguous does not mean deterministic.
- Input legitimacy: range of inputs needs to be carefully specified.
- Different implementations: data structures, language, compiler, machine architecture, etc.
- Different algorithms (based on different ideas) with dramatically different speeds for the same problem.

Different GCD Algorithms

- GCD: Greatest Common Divisor

- Three methods to find the GCD of two non-negative integers are discussed to illustrate the previous characteristics.
  1. Euclidean algorithm
  2. Consecutive divisor checking GCD algorithm
  3. Middle school procedure

Definitions

- The GCD of two non-negative, not-both-zero integers.

- Let $\mathbb{N}, \mathbb{N}'$ be respectively the sets of non-negative and positive integers:

$$\mathbb{N} = \{0, 1, 2, \ldots\}, \qquad \mathbb{N}' = \{1, 2, \ldots\}, \qquad \mathbb{N}' = \mathbb{N} \backslash \{0\}.$$

- Let $m, n \in \mathbb{N}$ and $m^2 + n^2 > 0$.
- Then $g = \text{GCD}(m, n)$ iff $g|m, g|n$, and if $d|m, d|n$ then $d|g$.
- Note that for any $0 \neq n \in \mathbb{N}$, $\text{GCD}(0, n) = n$.

Euclid

- Euclid of Alexandria was a Greek mathematician.
- Lived around 300 B.C.
- “Father of Geometry”. His Elements is one of the most influential works in the history of mathematics. Was used until 19th or early 20th century.

The Quotient-Remainder Theorem

- Let m be an integer and n be a positive integer.
- That is, $m \in \mathbb{N}, n \in \mathbb{N}'$.
- There are unique integers q (quotient), r (remainder) such that

$$m = qn + r \qquad \text{and} \qquad 0 \leq r < n.$$

- We denote the remainder $r = m\%n$.

Euclid's Algorithm

- It is perhaps the first known algorithm.
- It exploits the fact that GCD$(m, n) =$ GCD$(n, m\%n)$.
- The algorithm:

```
1: function GCD(m, n)
2:     while n > 0 do
3:         r ← m%n
4:         m ← n
5:         n ← r
```

- On termination, m is the GCD.

Termination of Euclid's Algorithm

- Note that if $m < n$ then the first iteration swaps m and n.
- So without loss of generality we may assume $m \geq n$.
- Thus Euclid's algorithm always stops because

$$0 \leq \mathsf{min}(n, m\%n) < \mathsf{min}(m, n).$$

Consecutive Divisor Checking GCD Algorithm

- The algorithm:

```
1: function CDC-GCD(m, n)
2:     t ← min(m, n)
3:     loop
4:         if t|m and t|n then
5:             return t
6:         else
7:             t ← t − 1
```

- Legitimate inputs are $m, n \in \mathbb{N}'$. (That is, none is zero.)
- The algorithm fails if either m or n is zero.

Middle-School GCD Procedure

- Let $2 = p_1 < p_2 < \ldots$ be the primes.
- The procedure

  Find

$$m = \prod_{i=1}^{\infty} p_i^{m_i}, \qquad n = \prod_{i=1}^{\infty} p_i^{n_i}$$

  Return

$$\prod_{i=1}^{\infty} p_i^{\min\{m_i, n_i\}}$$

  Example: GCD(84,36); $84 = 2 \times 2 \times 3 \times 7$; $36 = 2 \times 2 \times 3 \times 3$; GCD(84,36) $= 2 \times 2 \times 3 = 12$.

More of a Procedure than an Algorithm

- The procedure is not quite an algorithm because it is not clear how the “Find” step should be carried out.
- This problem can be overcome by having some factorization algorithm such as the sieve of Eratosthenes.
- Eratosthenes lived around 200 B.C.

Sieve of Eratosthenes

- Given an integer $n \geq 2$, find all the primes $\leq n$.
- The algorithm:

```
1: function SieveOfEratosthenes(n)
2:     for p ← 2 to n do
3:         A[p] ← p
4:     for p ← 2 to ⌊√n⌋ do
5:         if A[p] ≠ 0 then
6:             q ← p × p                ▷ 2p, ..., (p − 1)p already sieved.
7:             while q ≤ n do
8:                 A[q] ← 0
9:                 q ← q + p
```

- The number $p \leq n$ is a prime if and only if $A[p] = p$.

Sieve of Eratosthenes Example

- Find the list of primes not exceeding $n = 25$.

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
2	3		5		7		9		11		13		15		17		19		21		23		25
2	**3**		5		7				11		13				17		19				23		25
2	3		**5**		7				11		13				17		19				23		

Take-Away from GCD Discussion

- Importance of input specifications.
- Importance of precise algorithm specifications.
- Importance of algorithm efficiency.
- Multiple algorithms may solve the same problem.

Algorithmic Problem Solving

- Typical Activities:
- Understanding the problem
- Ascertaining the capabilities of a computational device

  Recent multi-core architectures make parallel algorithms more important
- Choosing between exact and approximate problem solving
- Deciding on appropriate data structures

  Algorithms + Data Structures = Programs
- Exploring algorithm design techniques

Algorithmic Problem Solving (2)

- Selecting methods of specifying an algorithm
- Proving the correctness of an algorithm

  "Beware of bugs in the above code;
  I have only proved it correct, not tried it."
  – Donald E. Knuth
- Analyzing an algorithm
  - Time efficiency
  - Space efficiency
  - Simplicity, generality
- Coding an algorithm

Important Problem Types in Computing

- Sorting
  - Sorting a list of records with keys is to arrange these records in ascending or descending key order.
  - A sort is stable if records of equal key values retain their relative positions before and after the sort.
  - A sort is in place if no extra memory is needed.
- Searching
  - Searching is about looking for a record with a given key value among a set of records.

Important Problem Types in Computing

- String Processing
  - A string is a finite sequence of symbols from an alphabet. Examples of strings include text strings, bit strings, and gene sequences.
  - String processing can be string matching, sub-string searching, etc.
- Graph Problems
  - These include graph traversal in different orders, shortest path problems, topologial sorting, etc.
- Combinatorial Problems
  - These are problems involving some combinatorial objects that satisfy some constraints.

Important Problem Types in Computing

- Geometric Problems
  - These are problems dealing with geometric objects that have shapes and positions in a space. Examples include the closest-pair problem and the convex-hull problem.
- Numerical Problems
  - These are the problems involving mathematical objects of continuous nature. One of the challenges is the use of rational numbers to compute the real numbers. These problems usually arise in science and engineering and are traditionally known as numerical analysis.

Well-Known Algorithm Design Strategies

- Brute force
- Divide-, decrease-, transform-and-conquer
- Space-time tradeoff
- Greedy technique
- Dynamic programming
- Iterative improvement
- Backtracking
- Branch-and-bound

Fundamental Data Structures

- Data structures are ways of organizing data items such that a data item can be inserted, deleted, and updated in a certain prescribed manner.
- Fundamental data structures include
  1. Linear data structures: arrays, linked lists, stacks, queues, priority queues and heaps.
  2. Graphs: undirected graphs, directed graphs.
  3. Trees: rooted trees, ordered trees, binary trees.
  4. Sets and dictionaries.

Choosing the correct data structure often can simplify the problem by alot

Arrays

- An array is a sequence of n items of the same type that are stored in contiguous memory locations. An array item can be accessed directly with its index. Usually the index ranges from 0 to $n-1$, or from 1 to n.
- The memory required for all the n items of an $n-$item array is allocated once at the same time.

Linked Lists

- A linked list is a sequence of zero or more data items that are linked with pointers.
- Unlike arrays, a data item of a linked list can only be accessed sequentially by starting from the first data item.
- A singly linked list allows the traversal of the list forward only. But a doubly linked list allows both forward and backward traversal; this requires a data item to keep two pointers, one pointing at the succeeding item, the other pointing at the preceding item.
- Memory for a list item is usually dynamically allocated at the time when the item is inserted into the list.

Abstract Linear Lists

- A linear list, or simply list, is an abstract concept.
- It is a sequence of items such that insertion and deletion of data items are governed by certain disciplines.
- A linear list can be implemented using either arrays or linked lists.

Special Linear Lists: Stacks and Queues

- A stack allows the insertion and deletion only at the top end of the sequence. The operation *pop()* deletes an item from the top and the operation *push()* inserts an item at the top.
- A queue allows the insertion and deletion at different ends of the sequence. The operation *enqueue()* inserts an item to the rear end of the sequence and the operation *dequeue()* deletes and item from the front end of the sequence.

Priority Queues

- Some applications require a sequence of data items to be accessed according to their priorities.
- A data structure that supports such a requirement is known as a priority queue.
- It is very inefficient to implement a priority queue using an array sorted by priority.
- Usually heaps are used to implement priority queues.

Graphs

- Intuitively a graph is a set of points, called nodes or vertices, such that some of the points are connected by one or more edges.
- Formally a graph G is an ordered pair (V, E) where V is the set of vertices and E is the set of edges.
- For a simple undirected graph $G = (V, E)$, we have $E \subseteq 2^V$; for a simple directed graph $G = (V, E)$, we have $E \subseteq V \times V$.
- If a graph is not simple, an edge function is needed to specify the end-points of an edge:

$$F : E \to 2^V \quad \text{or} \quad F : E \to V \times V.$$

Some Basic Graph Definitions

- Special edges: loops.
- Vertex/edge relationships: incidence (between a vertex and an edge) and adjacency (between two vertices or between two edges).
- Special graphs: complete graphs K_n, bipartite graphs, complete bipartite graphs $K_{m,n}$, cycle graphs.
- Weighted graphs.

Graph Representations

- Adjacency Matrices and Adjacency Lists

- A graph $G = (V, E)$ can be represented as a $|V| \times |V|$ matrix A such that $A[i, j] = 1$ if and only if vertex i and vertex j are adjacent; otherwise $A[i, j] = 0$.
- A graph can also be represented as an array L of linked lists such that array item $L[i]$ is a list of vertices adjacent to vertex i.

Graphs: Paths and Cycles

- A path in a graph is a sequence of adjacent edges.
- A simple path is a path in which all the vertices in the sequence of edges are distinct.
- The length of a path is the number of edges in the path.
- A cycle is a positive length path with distinct edges that begins and ends at the same vertex.

Graphs: Connectivity and Acyclicity

- A graph is connected if there is a path between two distinct vertices.
- If a graph is not connected, it consists of two or more connected components.
- A graph is acyclic if it has no cycles.

Trees

- A tree is a connected acyclic graph.
- A forest is a set of trees. Alternatively, a forest is an acyclic graph which may or may not be connected.
- Let $G = (V, E)$ be a graph. If G is a tree then $|E| = |V| - 1$. But if $|E| = |V| - 1$ and G is connected then G is a tree.
- Note that a graph $G = (V, E)$ with $|E| = |V| - 1$ may not be a tree. For example, if $G_1 = (V_1, E_1)$ is a tree and $G_2 = (V_2, E_2)$ is a cycle graph then their union $G = (V, E) = (V_1 \cup V_2, E_1 \cup E_2)$ is not a tree, but $|E| = |V| - 1$.

Rooted Trees

- If a vertex of a tree is designated to be the root, the (free) tree becomes a rooted tree.
- A rooted tree represents some hierarchy. Thus we may define the depth, also known as level, of a vertex (the root is the only vertex at depth 0) and the height of a vertex (vertices at the bottom are at height 0). The height of a tree is the height of the root.
- Other concepts due to the hierarchy are: ancestors, descendants, parent, children, siblings, and subtrees.

Ordered Trees

- An ordered tree is a rooted tree such that the children of a parent are ordered (usually from left to right).
- A binary tree is an ordered tree such that a parent has either a left child or a right child (or both).
- A binary search tree is a binary tree in which every vertex has a key and such that the key of a parent is greater than the keys of all vertices of its left subtree but less than the keys of all the vertices of its right subtree.

Height of Binary Trees

- The maximum height of a binary tree with n vertices is $n - 1$.

  This happens when each parent has only one child.
- The minimum height h' of a binary tree with n vertices is $\lfloor \log_2 n \rfloor$.

  This happens when there are 2^l vertices at levels $l = 0, \ldots, h' - 1$, and n' vertices at level h', $1 \leq n' \leq 2^{h'}$.
- Let h be the height of a binary tree with n vertices. We have

$$\lfloor \log_2 n \rfloor \leq h \leq n - 1.$$

Height of Binary Trees (Explanation)

- Let h' be the minimum height of a binary tree with n vertices.
- We have

$$\sum_{i=0}^{h'-1} 2^i + 1 \leq n \leq \sum_{i=0}^{h'} 2^i = 2^{h'+1} - 1.$$

- This gives

$$2^{h'} \leq n < 2^{h'+1}.$$

- Consequently,

$$h' = \lfloor \log_2 n \rfloor.$$

Height of Binary Trees (Explanation)

- Let h' be the minimum height of a binary tree with n vertices.
- We have
$$n = 2^0 + \ldots + 2^{h'-1} + 1 + x2^{h'}$$
where $0 \leq x < 1$.
- Thus
$$n = (1 + x)2^{h'} = 2^{h'+y}$$
where $0 \leq y < 1$ since $1 \leq x + 1 < 2$.
- That is,
$$h' = \lfloor \log_2 n \rfloor.$$

Sets and Dictionaries

- A set is a collection of unordered, distinct items called elements or members of the set.
- There are several ways to represent sets in computing: bit vectors (if the universal set is known), lists, parent arrarys (for the union and find applications).
- A data structure representing a set that supports the searching, inserting, and deleting of a member is called a dictionary. A dictionary may be implemented as an array, a balanced search tree, or by hashing.

Abstract Data Types

- By now we realize that it is what we can do with a set of data items that is important in implementing an algorithm; how the data items are represented is the least of the concerns of a programmer (if efficiency is ignored).
- This leads to the concept of abstract data types: a set of abstract objects representing data items with a collection of operations that can be performed on them.