Common problem types

Algorithms are often used to solve the following types of problems.

1. Sorting
2. Searching
3. String processing
4. Graph problems
5. Combinatorial problems
6. Geometric problems
7. Numerical problems

1. Sorting

A sorting problem is arranging items in a non-decreasing order. In practice, we usually sort items in numerical or alphabetical order. Items that often need to be sorted, such as employee lists, orders, and registration lists.

To solve each problem, we need to select pieces of data to guide the sorting. For example, we can choose to sort student records alphabetically, by student ID, or by GPA. Such a selected piece of data is called "**key**" or "**index**."

Why do we need a sorted list?

First, a sorted list can be the final result we are looking for, such as the ranking of search results and ranking students by GPA. Sorting also makes it easier to answer many questions, such as "*which products are the most profitable*," "*which student has the highest score*," and "*how many students scored below 50%*."

Sorting is also used as an auxiliary step in other algorithms, such as geometric algorithms and data compression. Algorithms like "closest pair of points" and "convex hull" require data to be sorted prior to the calculation.

There are many different sorting algorithms with different strengths and weaknesses. Some algorithms work best in specific situations, like when data are mostly sorted. Basic operations used in sorting algorithms are comparisons, copy or swap, and recursion.

Bubble sort is one of the simplest sorting algorithms. It repeatedly swaps the adjacent elements if they are in the wrong order until the list is sorted. In every iteration, we traverse the whole list and swap adjacent elements in the wrong order. Computer scientists call it a bubble sort because the numbers with the highest values 'bubble up' to the end of the list.

Insertion sort works like sorting a deck of cards. First, we split the list into two sub-lists: sorted and unsorted. Initially, the sorted list is empty, and the unsorted list is the whole input list. At each step, we pick the leftmost element in the unsorted list and place it in the correct position in the sorted list.

Quicksort uses a divide-and-conquer approach. It picks an element as a pivot and partitions the given list around the picked pivot. There are various versions of the quicksort algorithm depending on what we choose as the pivot element. All the sorting work is done in the divide (partitioning) step. One example is arranging students by height from shortest to tallest using a randomly picked individual in the group as a pivot and moving them around so that all individuals who are shorter than the chosen one move to the left of them, and all the rest move to their right, then repeating this step recursively.

Selection sort selects the current smallest element and swaps it into place. The algorithm divides the list into two sub-lists: sorted and unsorted. Initially, the sorted list is empty, and the unsorted list is the whole input list. In every iteration of the selection sort, the minimum element (considering ascending order) from the unsorted sub-list is picked and swapped into the sorted sub-list.

Merge sort is a recursive sorting algorithm that uses the concept of divide and conquer (or split and merge). It continually splits a list in half until there are one or more lists containing one item and then puts them back together in the correct order. All the sorting work is done in the merging step.

Sorting algorithm efficiency and robustness

There are many different sorting algorithms but no algorithm fits in every situation. Some algorithms are simple but quite slow, while some algorithms are faster but more complex. Some demand a lot of memory space, while some are more adaptable to sorting large files stored on a disk, and so on.

An important characteristic of a sorting algorithm is that it must maintain the **relative order** of two elements in the input. For example, the algorithm placed "I" before "J." Even if we add more items to the list, "I" must always be ranked before "J" if the condition is maintained.

In-place algorithms use comparisons and iterations to update elements in the original data array without using additional memory. **Out-of-place** algorithms require additional memory to store immediate results. Anyway, the time required for processing will increase as the dataset is too large.

2. Searching

A searching problem involves finding a given value called a "**search key**" in the given data set. There are many search algorithms available, from straightforward sequential search to powerful but limited binary search.

Linear search

Linear search (aka "sequential search") algorithms examine data from the first to the last elements in the data array to find the item(s) that matches the search condition(s). This technique will later be recognized as "**brute force**" analysis. The search may stop when the first match is found or progress to the end of the list to find all matches (or to later find out that there is no matched item).

Binary search

Binary search algorithms sort the data array by repeatedly dividing the search interval in half. This technique will be later recognized as "**Divide and conquer**." Since the list was pre-sorted, the time complexity is reduced. The binary search algorithm is especially important for real-world applications because it is indispensable for storing and retrieving data from large databases. Other searching algorithms are listed below.

- Linear search
- Sentinel linear search
- Binary search
- Meta binary search | One-sided binary search
- Ternary search
- Jump search
- Interpolation search
- Exponential search
- Fibonacci search
- Ubiquitous binary search

When it comes to search, there is no single algorithm that works best for every situation. Some algorithms are faster than others but require more memory. Some are very fast but only work with sorted arrays. Additionally, there are additional challenges such as adding or deleting data from the database.

3. String processing

A string is a sequence of characters arranged to form a text, name, phrase, or something else (such as a password). Strings may include letters, numbers, and special characters (such as ≥ and ω). There may also be control strings for new lines or paragraphs and validation strings for validating other strings in the set. String processing may include the followings:

Sorting strings

Strings are sorted according to their lexicographic order—that is, the order of their characters. In the alphabet, "a" comes before "b" and "d" comes after "c." The string "hello" comes before "jello" because "h" comes before "j" in the alphabet.

In computer programming, we use Unicode character sets to define characters. In most cases, uppercase letters come before the lowercase letters. So, the letter "H" comes before the letter "h" and the letter "Z" comes before the letter "a." As an example, the following strings are arranged in lexicographic order:

"" "!" "0" "A" "Andy" "Z" "Zero" "a" "an" "and" "andy" "candy" "zero"

Comparing strings

Comparing strings is another important task. For example, when a word processor performs a search and replace operation, it needs to identify strings in the text that match the target string. Comparing strings is done by checking the equality/identity of two or more strings. Other operations can be applied. For example, "Hello" can be equal to "hello" if "case-insensitive" condition is applied.

Finding things within a string

Some computing tasks are about finding the location of a particular character or substring in a string. For example, first and last names are sometimes stored in a single string separated by a special character, such as a single space. In order to process the first and last name separately (e.g., to create a citation), we need to locate that special character within the string.

Keyword search

A keyword search algorithm takes two parameters, the string being searched, and the keyword. The algorithm returns the number of keyword occurrences and the index of each occurrence. For example, if we asked this method to find all occurrences of "is" in "This is a test," it should return "2: 2 5" because there are two occurrences of "is," one starting at index 2 and the other at index 5 in the string.

Retrieving part of the string

Extracting a part of a character from a string is used when copying or deleting part of a string. The algorithm retrieves indexes indicating start and end points of the interval being extracted.

Encoding and decoding

Strings may be encrypted for security purposes. Many platforms encrypt passwords or even simple text before sending it over the network. Encryption and decryption algorithms are installed on both ends. Such algorithms may rely on checking for key matches. If they do not match, the decryption will be incomplete, and the message will be unreadable.

4. Graph problems

Graphs can be used to model a variety of applications such as transportation, communications, social networks, economics, project scheduling, and games. Graph problems are one of the current research areas of concern to computer scientists, economists, and social scientists.

Basic graph algorithms include graph traversal algorithms, known as traveling salesman problem (TSP). The aim is to find the shortest path to reach all the points in the network or the best route between two cities (with minimal traffic and road fee).

Some applications include finding the shortest path through all points only once for transportation. Another example is the production of circuit boards and chips with minimal mechanical arm movements.

Graph theory

The Seven Bridges of Königsberg is a historically notable problem in mathematics. Its negative resolution by Leonhard Euler in 1736 laid the foundations of graph theory. The city of Königsberg in Prussia (now Kaliningrad, Russia) was set on both sides of the Pregel River and included two large islands—Kneiphof and Lomse—which were connected to each other, and to the two mainland portions of the city, by seven bridges. Many people attempted to walk through the city that would cross each of those bridges only once and end at the point where they started.

Euler pointed out that the choice of route is irrelevant. The only important point is that the number of crossings "to" must be equal to the number of crossings "from" for each land mass, except for the start and end points. In other words, if the start and end points are the same, the number of crossings attached to each land mass must be an equal number.

5. Combinatorial problems

A combinatorial problem consists of a finite collection of data and a set of conditions/ constraints. The aim is to find a piece(s) of data that satisfies all conditions/constraints. Some examples include:

Routing -- Given a partially connected network on n nodes, find the shortest path between x and y.

Traveling Salesperson Problem (TSP) -- Given a partially connected network on n nodes, find a path that visits each node once and only once.

Scheduling -- Considering a number of tasks, each of which takes a different amount of time to complete and a number of operators/machines. Try to allocate tasks to operators/machines so that all tasks are completed within the deadline.

6. Geometric problems

Geometric algorithms deal with geometric objects such as points, lines, and polygons. The ancient Greeks were very interested in developing processes for solving various geometric problems, including constructing geometric figures with rulers and compasses.

In the age of computers, there are no longer rulers and compasses, only bits, bytes, and human ingenuity. People today are interested in geometric algorithms with different applications, such as computer graphics, robotics, and x-rays. Here are some examples of algorithms for classic geometry problems.

Closest pair of points: Given n points in metric space, find a pair of points with the smallest distance between them.

Convex hull problems: A convex hull is the smallest convex polygon that completely encloses a set of points in a two-dimensional or three-dimensional space. It can be thought of as the "envelope" or "wrapper" of the points.

7. Numerical problems

Numerical problems involve mathematical continuous data, such as solving equations and systems of equations, calculating definite integrals, and so on. This does not include binary data or class differences. Numerical problems in nature often give only approximate results. Examples include weight, speed, temperature, and current, where it is very difficult to find only integers at every step. This can lead to the accumulation of rounding errors to the point where they can significantly distort the output.

Numerical algorithms once played an important role in scientific and engineering applications. But the digital industry has shifted its focus to business applications. These new applications primarily require algorithms for data storage, retrieval, transmission over networks, and presentation to users.