Chapter 1

Analysis of algorithms

THE analysis of algorithms is to determine the amount of resource (time/space) necessary to execute algorithms. By analyzing the resources used in algorithms, we can compare different algorithms theoretically.

The amount of resource used in an algorithm is usually represented by a function $T(n)$, where n is the length of the input. The goal of the analysis of algorithms is to find out the asymptotic growth rate of $T(n)$ in terms of n. In computer science, since n denotes the length of the input, we only care about positive values of n. Similarly, since $T(n)$ denotes the amount of resource, we usually assume that $T(n)$ is positive. This is not a real limitation, but can simplify the discussion.

Each textbook usually discusses the analysis of algorithms in the first several chapters. For those who want to know further about the analysis of algorithms, you can watch the videos of Analysis of Algorithms on Coursera and read books [3, 5, 6].

1.1 Recurrence relations

During the exam, we are often given a recurrence relation $T(n) = a(n)T(b(n)) + f(n)$, and we need to give to a tight bound of $T(n)$. If the problem statements do not explicitly specify the base care, we usually assume that $T(n) = \Theta(1)$ when n is small (less than some constant). In general, we can just focus on some particular values of n as long as these values approach infinity[1]. For example, we can only consider $n = 2^i$ for all positive integer i. Making assumptions can make the analysis easier.

1.1.1 Master theorem

The most powerful technique in solving divide-and-conquer recurrence relation is the master theorem. Several forms of the master theorem have been proven. Verma gives the following master theorem [7]:

Theorem 1. *Let $T(n) = aT(n/b) + f(n)$ for all $n > 1$ and $T(1) = c$ for some constants $a \geq 1$, $b > 1$, $c > 0$, and non-negative function $f(n)$.*

1. *if $f(n) = O(n^{\lg_b a}/\lg n)(1+\epsilon)$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\lg_b a})$.*
2. *if $f(n) = \Theta(n^{\lg_b a}\lg^k n)$ for some $k \geq 0$, then $T(n) = \Theta(n^{\lg_b a}\lg^{k+1} n)$.*
3. *if $f(n) = \Omega(n^{\epsilon+\lg_b a})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq kf(n)$ for some constant $k < 1$ and all sufficiently large n, then $T(n) = \Omega(f(n))$.*
4. *if $f(n) = \Theta(n^{\lg_b a}/\lg n)$, then $T(n) = \Theta(f(n)\lg n \lg\lg n)$.*

Remark. When you apply the master theorem, please pay attention to the following:

[1] There should be a condition specifying what assumptions are applicable.

1. When the recursion involves floor or ceiling function, the master theorem does not apply. For example, $T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n$.
2. In order to apply the case 2 (the extended master theorem), k must be non-negative. For example, case 2 does not apply in the case of $T(n) = 2T(\frac{n}{2}) + n/\lg n$.
3. In order to apply the case 3, the regularity condition must be satisfied. For example, case 3 does not apply in the case of $T(n) = T(\frac{n}{2}) + n(2 - \cos n)$.

Exercise 1 (*NCTU CSIE 104*)

Let $T(n) = \Theta(f(n))$. Assume that $T(n)$ is a constant for sufficiently small n. Derive $f(n)$ in the simplest formula for each of the following $T(n)$.

1. $T(n) = 2T(\frac{n}{2}) + \frac{n}{\lg^2 n}$.

Answer of exercise 1

Problem 1 By case 1, we know $T(n) = \Theta(n)$.

Problem 1 alternative solution Suppose that $n = 2^k$. We have

$$\begin{aligned}
& T(n) = 2T(\frac{n}{2}) + \frac{n}{\lg^2 n} \\
\equiv\quad & T(n) = 4T(\frac{n}{4}) + \frac{n}{\lg^2 n} + \frac{n}{(\lg(n-1))^2} \\
\equiv\quad & T(n) = 2^k T(1) + n \sum_{i=}^{k} i^{-2} \\
\equiv\quad & T(n) = n + n \sum_{i=}^{k} i^{-2}
\end{aligned}$$

Since $\sum_{i=}^{k} i^{-2}$ is lower bounded by one and is upper bounded by $\sum_{i=}^{\infty} i^{-2} = \zeta(2) = \frac{\pi^2}{6}$[2], we have $T(n) = \Theta(n)$.

1.1.2 Akra-Bazzi method

The Akra and Bazzi method provides a more general way to solve divide-and-conquer recurrence relations [1]. The following version is from [4].

Theorem 2. *Let*

$$T(n) = \begin{cases} \Theta(1) & 1 \le n \le n_0 \\ \sum_{i=1}^{k} a_i T(b_i n + h_i(n)) + f(n) & \forall n > n_0 \end{cases}$$

where

1. *$k \ge 1$ is a constant and for all i, $a_i > 0$ and $b_i \in (0,1)$ are constants.*
2. *$|f(n)|$ is polynomially-bounded.*
3. *for all i, $|h_i(n)| = O(x/\lg^2 x)$.*
4. *n_0 is large enough.*

Let p be the unique solution for $\sum_{i=1}^{k} a_i b_i^p = 1$. Then

[2] This is the solution for the Basel problem.

1. *if* $f(n) = O(n^{p-\epsilon})$ *for some constant* $\epsilon > 0$*, then* $T(n) = \Theta(n^p)$.
2. *if* $f(n) = \Theta(n^p)$ *for some constant* $\epsilon > 0$*, then* $T(n) = \Theta(n^p \lg n)$.
3. *if* $f(n) = \Omega(n^{p+\epsilon})$ *and* $f(n)/x^{p+\epsilon}$ *is non-decreasing for some constant* $\epsilon > 0$*, then* $T(n) = \Theta(f(n))$.
4. $T(n) = \Theta\left(n^p\left(1 + \int_1^n \frac{f(u)}{u^{p+1}}du\right)\right)$.

Remark. This version of the Akra-Bazzi method can deal with floor and ceil function by picking h_i.

General version Some of the requirements can be relaxed [4].

1. The second condition is called *polynomial-growth condition* and can be replaced by the following weaker requirement: $g(n)$ is nonnegative and exist constants c_1 and c_2 such that for all i and for all $u \in [b_i n + h_i(n), n]$, we have $c_1 f(n) \le f(u) \le c_2 f(n)$.

2. The third condition can be replaced by the following weaker requirement: there exists a constant $\epsilon > 0$, $|h_i(n)| \le n/(\lg^{1+\epsilon} n)$ for all i and $n \ge n_0$.

3. The fourth condition is pretty technical and you can find the complete version in the original paper.

Drmota and Szpankowski prove a more general theorem that can deal with floor and ceil functions directly [2].

Exercise 2

Let $T(n) = \Theta(f(n))$. Assume that $T(n)$ is a constant for sufficiently small n. Derive $f(n)$ in the simplest formula for each of the following $T(n)$.

1. $T(n) = 5T(\frac{n}{5}) + n/\lg n$. [NCTU CSIE 93]
2. $T(n) = T(\frac{n}{2} + \sqrt{n}) + n$. [NTU CSIE 103]
3. $T(n) = 4T(\frac{n}{5}) + T(\frac{n}{4}) + n$. [NCTU BIOINFO 93]

Answer of exercise 2

Problem 1 Suppose that $n = 5^k$. We have

$$\begin{aligned} & T(n) = 5(5T(\frac{n}{25}) + n/(5(\lg_5 n - \lg_5 5))) + n/\lg n \\ \equiv\ & T(n) = 5^k T(1) + n/(\lg n + \lg(n-1) + \cdots + 1) \\ \equiv\ & T(n) = \Theta(n \lg\lg n) \end{aligned}$$

Problem 1 another solution apply the Akra-Bazzi method. Set $k = 1$, $a_1 = 5$, $b_1 = 1/5$, and solve $p = 1$. We get

$$T(n) = \Theta(n(1 + \int_1^n (x/\lg x)x^{-2}dx)) = \Theta(n \lg\lg n).$$

Remark. Similarly, for any constant c, we have $T(n) = cT(\frac{n}{c}) + n/\lg n = \Theta(n \lg\lg n)$.

Problem 2 apply the Akra-Bazzi method. Set $k = 1$, $a_1 = 1$, $b_1 = 1/2$, and solve $p = 0$. Then, we get

$$T(n) = \Theta(1(1 + \int_1^n (x/x)dx)) = \Theta(n).$$

Problem 3 apply the Akra-Bazzi method. Set $k = 2$, $a_1 = 4$, $b_1 = 1/5$, $a_2 = 1$, $b_2 = 1/4$, and solve $p \approx 1.03$. Then, we get

$$T(n) = \Theta(n^p), \text{where} p \approx 1.03.$$

Exercise 3 (*NCTU CSIE 92*)

Given positive constants c', $c_1, c_2, \ldots, c_k$, assume that $T(n) \le c'n + \sum_{i=1}^{k} T(c_i n)$ and $\sum_{i=1}^{k} c_i < 1$. Prove $T(n) = O(n)$.

Answer of exercise 3

Apply the Akra-Bazzi method.

1.1.3 Full-history recurrence

Exercise 4 (*NTU CSIE 90*)

Let $T(n) = \Theta(f(n))$. Assume that $T(n)$ is a constant for sufficiently small n. Derive $f(n)$ in the simplest formula for each of the following $T(n)$.

1. $T(n) = n + \frac{4}{n} \sum_{i=1}^{n-1} T(i)$.

Answer of exercise 4

Problem 1

$$\begin{aligned}
& T(n) = n + \frac{4}{n} \sum_{i=1}^{n-1} T(i) \\
\equiv \quad & nT(n) = n^2 + 4 \sum_{i=1}^{n-1} T(i) \\
\equiv \quad & (n+1)T(n+1) = n^2 + 4 \sum_{i=1}^{n} T(i) \\
\equiv \quad & (n+1)T(n+1) - nT(n) = 2n + 1 + 4T(n) \\
\equiv \quad & (n+1)T(n+1) = (n+4)T(n) + 2n + 1 \\
\equiv \quad & \frac{T(n+1)}{(n+2)(n+3)(n+4)} = \frac{T(n)}{(n+1)(n+2)(n+3)} + \frac{2n+1}{(n+1)(n+2)(n+3)(n+4)}
\end{aligned}$$

Let $S(n) = \frac{T(n)}{((n+1)(n+2)(n+3))}$.

$$\begin{aligned}
& S(n+1) = S(n) + \frac{2n+1}{(n+1)(n+2)(n+3)(n+4)} \\
\equiv \quad & S(n) = \sum_{i=0}^{n-1} \frac{2i+1}{(i+1)(i+2)(i+3)(i+4)} \\
\equiv \quad & S(n) = \Theta(1) \\
\equiv \quad & T(n) = (n+1)(n+2)(n+3)S(n) \\
\equiv \quad & T(n) = \Theta(n^3)
\end{aligned}$$

1.1.4 Range transformation

Exercise 5 (*NCTU CSIE 93*)

Let $T(n) = \Theta(f(n))$. Assume that $T(n)$ is a constant for sufficiently small n. Derive $f(n)$ in the simplest formula for each of the following $T(n)$.

1. $T(n) = \sqrt{n}T(\sqrt{n}) + \sqrt{n}$.

Answer of exercise 5

Problem 1 The trick is to divide n by both side and then expand. $\frac{T(n)}{n} = \frac{T(\sqrt{n})}{\sqrt{n}} + n^{-0.5}$. Suppose that $n = 2^{2^k}$. Let $S(k) = T(2^{2^k})/2^{2^k}$.

$$
\begin{aligned}
& T(n) = S(k) = S(k-1) + 2^{-2^{k-1}} \\
\equiv \quad & S(k) = \Theta(1) + \sum_{i=1}^{k-1} 2^{-2^{k-1}} = \Theta(1) \\
\equiv \quad & T(n) = n \cdot S(k) = n \cdot \Theta(1) = \Theta(n)
\end{aligned}
$$

1.1.5 Recursion trees

Exercise 6

Let $T(n) = \Theta(f(n))$. Assume that $T(n)$ is a constant for sufficiently small n. Derive $f(n)$ in the simplest formula for each of the following $T(n)$.

1. $T(n) = 2T(\sqrt{n}) + \lg n$. [NCKU CSIE 95]
2. $T(n) = nT(\sqrt{n}) + n^2 \lg n$. [NTU CSIE 98]

Answer of exercise 6

Problem 1 Suppose that $n = 2^{2^k}$. We have

$$
\begin{aligned}
& T(n) = 2^2 T(\sqrt[4]{n}) + 2(\lg n - 1) + \lg n \\
\equiv \quad & T(n) = 2^k T(1) + \lg n + 2(\lg n)/2 + 4(\lg n)/4 + ... + 2^k(\lg n)/2^k \\
\equiv \quad & T(n) = 2^k T(1) + \lg n \sum_{i=1}^{k} 1 \\
\equiv \quad & T(n) = \Theta(\lg n \lg \lg n)
\end{aligned}
$$

Problem 2 Suppose that $n = 2^{2^k}$. We have

$$
\begin{aligned}
& T(n) = n^{1.5} T(\sqrt[4]{n}) + \frac{n^2 \lg n}{2} + n^2 \lg n \\
\equiv \quad & T(n) = n^2 T(1) + n^2 \lg n \sum_{i=0}^{k} 2^{-i} \\
\equiv \quad & T(n) = \Theta(n^2 \lg n)
\end{aligned}
$$

1.1.6 Comparison

*** Exercise 7** (*NCTU BIOINFO 93*)

Let $T(n) = \Theta(f(n))$. Assume that $T(n)$ is a constant for sufficiently small n. Derive $f(n)$ in the simplest formula for each of the following $T(n)$.

1. $T(n) = T(\frac{n}{2}) + T(\sqrt{n}) + n$. [NCTU BIOINFO 93]

Answer of exercise 7

Problem 1 By the recurrence relation, we have $T(n) = \Omega(n)$. Let $F(n) = F(n/2) + F(n/3) + n$. Since $n/3 > \sqrt{n}$ for all $n > 9$, we have $F(n) \geq T(n)$. By using the Akra-Bazzi method, we get $F(n) = \Theta(n)$. Thus, we have $T(n) = \Theta(n)$.

References

[1] Mohamad Akra and Louay Bazzi. "On the Solution of Linear Recurrence Equations". In: *Computational Optimization and Applications* 10.2 (May 1998), pages 195–210. ISSN: 0926-6003. DOI: 10.1023/A:1018373005182. URL: http://dx.doi.org/10.1023/A:1018373005182 (cited on page 2).

[2] Michael Drmota and Wojciech Szpankowski. "A Master Theorem for Discrete Divide and Conquer Recurrences". In: *Journal of the ACM* 60.3 (June 2013), page 16. ISSN: 0004-5411. DOI: 10.1145/2487241.2487242. URL: http://doi.acm.org/10.1145/2487241.2487242 (cited on page 3).

[3] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete mathematics - a foundation for computer science*. 2nd edition. Addison-Wesley, 1994. ISBN: 978-0-201-55802-9 (cited on page 1).

[4] Tom Leighton. *Notes on Better Master Theorems for Divide-and-Conquer Recurrences*. Technical report. Department of Mathematics, Massachusetts Institute of Technology, October 1996 (cited on pages 2, 3).

[5] Paul Walton Purdom Jr. and Cynthia Brown. *The Analysis of Algorithms*. Oxford University Press, 2004. ISBN: 978-0-195-17479-3 (cited on page 1).

[6] Robert Sedgewick and Philippe Flajolet. *An introduction to the analysis of algorithms*. 2nd edition. Addison-Wesley, 2013. ISBN: 978-0-321-90575-8 (cited on page 1).

[7] Rakesh M. Verma. "A General Method and a Master Theorem for Divide-and-Conquer Recurrences with Applications". In: *Journal of Algorithms* 16.1 (January 1994), pages 67–79. ISSN: 0196-6774. DOI: 10.1006/jagm.1994.1004. URL: http://dx.doi.org/10.1006/jagm.1994.1004 (cited on page 1).

Chapter 2

Problems on sequences

Arrays are the simplest data structures in computers. Even the structure of an array is very simple, there are still some hard problems. In this chapter, we will go through several hard problems of arrays.

2.1 Searching

2.1.1 k-sum problem

Exercise 8

1. Given three sets of integers X, Y, and Z, and another integer k, we want to know if there exists three numbers x, y, and z, such that $x \in X$, $y \in Y$, and $z \in Z$, and $x + y + z = k$. Design an algorithm to solve this problem. A naive method by checking all possible sums of triples $(x + y + z)$ will take $\Theta(|X||Y||Z|)$ time. Your algorithm must be more efficient than it. Analyze the time complexity of your algorithm. [NCU CSIE 99]
2. Given an array of n numbers, and a number s, determine whether the array contains 4 elements whose sum is s. Analyze the time efficiency of your algorithm. Your algorithm should be more efficient than $O(n^4)$. [NCU CSIE 93]

Answer of exercise 8

Problem 1 This problem is called the 3SUM problem. Construct a set $Z' = \{k - z \mid z \in Z\}$. The problem is equal to finding $x \in X$, $y \in Y$, and $z \in Z'$, such that $x + y = z$. Sort the sets Y and Z'. For each element $x \in X$, construct a set $Y_x = \{y + x \mid y \in Y\}$. If one element a in both Y_x and Z' exists, then we find an answer. When Y_x is fixed, finding common elements in sorted Y_x and Z' can be done in $O(|Y| + |Z|)$ time. Since we apply this operation for each element $x \in X$, the running time is $\Theta(|X||Y||Z|)$.

Remark. This problem can be solved in $o(n^2)$ time [5].

Problem 2 Let a denote the input array. Let S_{ij} be the sum of $a[i]$ and $[j]$. Sort all S_{ij} in increasing order and store them in an array A of length L. The problem is equal to finding two elements in A sum to s, which can be solved in time linear in L. Since the number of pairs is $O(n^2)$, the sort takes $O(n^2 \lg n)$ time. Moreover, the search takes $O(n^2)$ time. Thus, the running time is $O(n^2 \lg n)$.

2.1.2 Binary search

Exercise 9

1. Let $A[n]$ be an array with n elements sorted in ascending order. It is simple to construct an $O(\lg n)$ algorithm to find the position k in $A[n]$ for a given value v. Assume that k is much less than n (i.e.,

$k \ll n$). Write an $O(\lg k)$ time algorithm to search for k. (Note: you do not know the value of k in advance, only v is known) [YZU CSIE 93]

2. Suppose you are given an array of n sorted numbers that has been circularly shifted m positions to the right. For example $\{36, 45, 5, 18, 26, 29\}$ is a sorted array that has been circularly shifted $m = 2$ positions to the right. Give an $O(\lg n)$ algorithm to find the largest number in this array. Note that you don't know what m is. [CYCU CSIE 89]

   Give an efficient algorithm to determine if there exists an integer i such that $A_i = i$ in an array of integers $A_1 < A_2 < \cdots < A_n$. What is the running time of your algorithm? [NDHU CSIE 96, CYCU CSIE90]

Answer of exercise 9

Problem 1 We use linear search to find an index i such that $2^i \leq k < 2^{i+1}$, which takes $O(\lg k)$ time. Then, we apply binary search for find $x = A[k]$ for $k \in [1 \dots 2^{i+1}]$, which takes $O(\lg k)$ time as well.

Remark. This technique is called exponential search [1].

Problem 2 Let $m = \frac{n}{2}$. We have three cases:

1. $A[m] > A[m + 1]$: $A[m]$ is the maximum.
2. $A[m] > A[1]$: the maximum locates in $A[m + 1] \sim A[n]$. Recurse.
3. $A[m] < A[1]$: the maximum locates in $A[1] \sim A[m - 1]$. Recurse.

Since the problem size is halved in each recursion, the running time is $O(\lg n)$.

Problem 3 Let $m = \frac{n}{2}$. We have three cases:

1. $A[m] = m$: $A[m]$ is the fixed point.
2. $A[m] > m$: the fixed point locates in $A[1] \sim A[m - 1]$. Recurse.
3. $A[m] < m$: the fixed point locates in $A[m + 1] \sim A[n]$. Recurse.

Since the problem size is halved in each recursion, the running time is $O(\lg n)$.

2.1.3 Majority problem

Exercise 10

1. An array $a[1, \dots, n]$ is said to contain a majority element if there is some element that appears more than $\frac{n}{2}$ times in the array. The task is to determine if a has a majority element and, if so, to find the element. We do not assume that the elements of the array come from some ordered domain such as the integers, so we cannot sort the array or perform comparisons such as $a[i] < a[j]$. However, we assume we are able to test if $a[i] = a[j]$ in constant time. Can you come up with a simple $O(n)$-time algorithm for this problem? [CCU CSIE 95]
2. Given an n-element array A of real numbers, design an $O(n)$ time algorithm which determines whether any value occurs more than $\frac{n}{7}$ times in A. [NTUT CSIE 102]

Answer of exercise 10

Problem 1 The algorithm is based on the following observation: Let x and y be two distinct elements in a. Discarding x and y from a will keep the majority unchanged.

We choose the first element as a candidate and set M as one. If the next element differs from the candidate, then decrease M by one. If M becomes zero, then choose the next element as a candidate. Repeat this process until all elements are processed. Finally, we need one more pass to test whether the candidate is indeed a majority. Since this is a two pass algorithm, the algorithm runs in linear time.

Remark. More information can be found in [7].

Problem 2 If one element occurs more than $\frac{n}{7}$ times in A, then the element must be one of $\frac{kn}{7}$-th largest element in A, where $1 \leq k \leq 7$. Thus, we can use the selection algorithm to find all $\frac{kn}{7}$-th largest elements in A for all k in linear time. For each of them, determine whether this element occurs more than $\frac{n}{7}$ times in A can also be done in linear time. Thus, the running time is linear.

Remark. For any threshold value t, finding element with frequency larger than $\frac{n}{t}$ can be done in linear time (independent from t) [6].

2.2 Sorting

* Exercise 11 Pancake sorting problem (*NCKU CSIE 102*)

Jeremy is a waiter working in a restaurant. The chef there is sloppy; when he prepares a stack of pancakes, they come out all different sizes. When Jeremy delivers the pancakes to the customer, he wants to rearrange them by grabbing several from the top and flipping them over on the way. After repeating this for several times, the smallest pancake is on top, and so on, down to the largest at the bottom. If there are n pancakes, how many flips are required? Design an algorithm to help Jeremy, and analyze its time complexity.

Answer of exercise 11

This problem is called the pancake sorting problem. For a stack of pancakes, first locate the largest pancake. Then flip the largest pancake to the top by using one flip and use another flip to move the largest pancake to the bottom. Sort the top $n-1$ pancakes recursively. Since every pancake except the smallest one needs at most two flips to move to the correct position, the number of flips is $2n-2$.

Remark. One algorithm with at most $\frac{18}{11}n$ flips exists [3].

Exercise 12 (*NTUT CSIE 95*)

Let A be an array of n arbitrary and distinct numbers. A has the following property: If we imagine B as being sorted version of A, then any element that is at position i in array A would, in B, be at a position j such that $|i-j| \leq k$. In other words, each element in A is not farther than k positions away from where it belongs in the sorted version of A. Suppose you are given such an array A, and you are told that A has this property for a particular value k (that value of k is also given to you). Design an $O(n \lg k)$ time algorithm for sorting A.

Answer of exercise 12

Initially, insert the first $2k$ elements into a min heap; then, repeat the following process $n-2k$ times. In iteration i, delete the minimum value in the heap and output it; then, insert the $(2k+i)$-th element into the heap. After loops terminates, sort the elements in the heap by the heap sort and output it.

For any position i in the sorted array, the possible positions in the unsorted array are from $i-k$ to $i+k$, so using a min heap guarantees the correctness. Since the size of heap is $O(k)$, the insertion and deletion takes $O(\lg k)$ time. Since we insert and delete $O(n)$ times, the running time is $O(n \lg k)$.

Exercise 13 (*NCU CSIE 98*)

The input is a sequence of n integers with many duplications, such that the number of distinct integers in the sequence is $O(\lg n)$. Design a sorting algorithm to sort such sequences using at most $O(n \lg \lg n)$ comparisons in the worst case.

Answer of exercise 13

Augment an AVL-tree by creating a count field in each tree node. First process all elements in the input sequentially. For each integer x in input, search for x in the tree; if x has been inserted, then increment the x's count; otherwise, insert x into the tree and initialize the count to 1. Then, in-order traverse the data structure and output the elements with its multiplicity. Since the distinct integers is $O(\lg n)$, both of search and insertion take $O(\lg \lg n)$ time. Since the number of insertion and search is $O(n)$, the running time is $O(n \lg \lg n)$.

2.2.1 Bucket sort

Exercise 14 (*NDHU CSIE 97*)

We are given n points in an unit circle, $p_i = (x_i, y_i)$, such that $0 < \sqrt{x_i^2 + y_i^2} < 1$, for $i = 1, \ldots, n$. Suppose that the points are uniformly distributed; that is, the probability of finding a point in any region of the circle is proportional to the area of that region. Design and prove a $\Theta(n)$ expected-time algorithm to sort the n points by their distance $d_i = \sqrt{x_i^2 + y_i^2}$ from the origin. (Hints: Design the bucket sizes in Bucket-Sort to reflect the uniform distribution of the points in the unit circle.)

Answer of exercise 14

Divide the circle into n concentric circles with radii $\sqrt{\frac{1}{n}}, \sqrt{\frac{2}{n}}, \ldots, 1$. The area of the ith level is $\pi\sqrt{\frac{i}{n}}^2 - \pi\sqrt{\frac{i-1}{n}}^2 = \pi/n$. Since the areas are equal-size, the probability of point locates in any level is $1/n$. So we can partition the points by their levels and use the bucket sort to sort these points.

2.3 String

*** Exercise 15 Longest common substring problem** (*NTU CSIE 97*)

Given two length-n binary strings A and B, consider the problem of computing a longest string C that is a substring of both A and B. You are asked to prove or disprove that this so-called *longest common substring problem* can be solved in $O(n \lg n)$ time.

Answer of exercise 15

This problem is called the longest common substring problem. Build a generalized suffix tree from A and B. Find the deepest internal node that has leaves from two strings. The string corresponding to the path from the root to the deepest internal node is the longest common substring. Since building a suffix tree takes linear time, the running time is linear.

2.4 Array of arrays

*** Exercise 16** (*NTUST CSIE 98*)

It is trivial to find the median of the integers in the sorted array a with median $= a[\lfloor \frac{n}{2} \rfloor]$. Suppose we have $3n$ distinct integers that are randomly stored in arrays $a[0 \ldots n-1]$, $b[0 \ldots n-1]$, and $c[0 \ldots n-1]$, and each array is sorted independently. Write an algorithm to find the median of these $3n$ distinct integers. Please note that you are not allowed to merge arrays a, b, and c into a $3n$ integer array and then perform sorting.

Answer of exercise 16

The idea is as follows: in each iteration, we eliminate some elements that cannot be the answer. Repeat this process until only one element remains.

Let N be the number of all remaining elements and our goal is to find the k-th smallest element. Initially, we have $N = 3n$ and $k = \frac{N}{2}$. In each iteration, we collect the middle elements of all arrays as a set P. If $k < \frac{N}{2}$, then we pick the array with the maximum middle element. In this array, the elements that are larger than the middle element must be larger than $\frac{N}{2}$ elements in all elements. Thus, we can remove these elements safely. The case in which $k > \frac{N}{2}$ is symmetric. We adjust the k value and recurse on the remaining elements. Since one array is halved in each iteration, the number of iterations is $O(\lg n)$. Since each iteration takes constant time, the running time is $O(\lg n)$.

Exercise 17 (*NTU CSIE 93*)

M is an $n \times n$ integer matrix in which the entries of each row are in increasing order (reading left to right) and the entries in each column are in increasing order (reading top to bottom). Give an efficient algorithm to find the position of an integer x in M, or determine that x is not there. Tell how many comparisons of x with matrix entries your algorithm does in the worst case.

Answer of exercise 17

For any pair of indices , i and j, if $M[i][j] > x$, then we know $M[i][k] \neq x$ for all $j \leq k \leq n$ and $M[k][j] \neq x$ for all $i \leq k \leq n$. Similarly, if $M[i][j] < x$, then we know $M[i][k] \neq x$ for all $1 \leq k \leq j$ and $M[k][j] \neq x$ for all $1 \leq k \leq i$.

The idea is as follows: in each iteration, we compare an entry with x and the eliminate either a row or a column.

We always compare x of the top-right element of the remaining submatrix. If this entry equals x, then we found the answer. If this entry is smaller x, then the top row can be discarded. If this entry is larger than x, the the rightmost column can be discarded. Since the matrix has $O(n)$ rows and columns, the running time is $O(n)$.

Remark. This technique is called *saddleback search* [2].

Exercise 18 (*NCU CSIE 96*)

Suppose that k workers are given the task of scanning through a shelf of books in search of a given piece of information. To get the job done efficiently, the books are to be partitioned among k workers. To avoid the need to rearrange the books, it would be simplest to divide the shelf into k regions and assign each region to one worker. Each book can only be scanned by one worker. you are asked to find the fairest way to divide the shelf up. For example, if a shelf has 9 books of sizes 100, 200, 300, 400, 500, 600, 700, 800 and 900 pages, and $k = 3$, the fairest possible partition for the shelf would be

100 200 300 400 500 | 600 700 | 800 900

where the largest job is 1700 pages and the smallest job 1300. In general, we have the following problem: Given an arrangement S of n nonnegative numbers, and an integer k, partition S into k regions so as to minimize the difference between the largest and the smallest sum over all regions. Design algorithm to find an optimal solution for any given $k \leq n$.

Answer of exercise 18

For a number d, we can verify whether these n books can be partitions into k parts with the maximum load at most d. Create a matrix M, such that $M[i][j] = \sum_{k=i}^{j} S_k$. The minimum d value must be one of the entry of M. Thus, it suffice to do saddleback search on M. The running time is $O(n^2)$.

Remark. This problem can be solved in linear time [4].

2.5 Others

Exercise 19 Cycle detection problem (*MCU CSIE 95*)

Design an algorithm which detects whether there exists a cycle within a singly linked list. (Suppose that each node in this list contains data and next fields for storing data and the pointer to the next node respectively.) Please analyze the time and space complexities of your algorithm.

Answer of exercise 19

This problem is identical to the cycle detection problem. Maintain two pointers, fast and slow, and traverse the linked list from head in parallel. The fast pointer traverse the list two nodes per iteration, while the slow pointer traverses one node. If the list contains a cycle, then these two pointers will point to the same node in some iteration. Otherwise, the fast pointer will achieve the end of the list. The running time is linear and the space usage is constant.

Remark. More information can be found in [8].

Exercise 20 (*NCTU CSIE 91*)

Given a finite set A and a mapping function f from A to itself, describe an algorithm to find a subset S of A with maximum size such that f is one-to-one when restricted to S.

Answer of exercise 20

The idea is as follows: we repeatedly remove one element $x \in S$ such that $f(y) \neq x$ for all $y \in S$ until no such an element exists.

In implementation, we maintain an hashtable h, where $h[x]$ stores how many elements y satisfy $f(y) = x$. In each iteration, eliminate an element x with $h[x] = 0$, and decrement $h[f(x)]$, until $h[x] \neq 0$ for all x. The running time is $O(n^2)$.

2.5.1 Query support

Exercise 21 (*NTHU CSIE 100*)

Suppose we have n ranges $[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]$, where all a_i's are negative and all b_i's are positive. We are asked to preprocess these ranges so that for any input value x, we can efficiently count the number of the ranges containing x. Design an efficient representation of the ranges so that the desired counting can be done in $O(\lg n)$ time. Your representation must take $O(n)$ space. Describe you representation and how to answer a query in $O(\lg n)$ time in details.

Answer of exercise 21

Suppose that x is a positive number. Since a_i is negative and b_i is positive, we only need to count the number of intervals whose b_i is larger than x. If we preprocess the ranges in order by b_i, counting the can be done in $O(\lg n)$ time by using binary search. The case in which x is a negative number is symmetric.

* Exercise 22 Range minimum query problem (*NCTU BIOINFO 100*)

Suppose that we are given a sequence of n unsorted values, say $x_1, x_2, \ldots, x_n$, and at the same time, we are asked to quickly answer repeated queries defined as follows: Given i and j, where $1 \leq i \leq j \leq n$, find the smallest value in $x_i, x_{i+1}, \ldots, x_j$. Please design a data structure that uses $O(n)$ space and answers the queries in $O(\lg n)$ time.

Answer of exercise 22

This problem is called the range minimum query problem. We first divide n values into blocks of size $\lg n$. For a given query (i, j), since each block has $O(\lg n)$ elements, the minimum among all elements in i's block that are after i can be found in $O(\lg n)$ time. Similarly, the minimum among all elements in j's block that are before j can be found in $O(\lg n)$ time as well.

The problem becomes how to find the minimum of all full blocks between i and j in $O(\lg n)$ time. We can precompute a matrix M, such that $M[s][k]$ is the minimum in blocks $s, s+1, \ldots, s+2^k$. The minimum of all full blocks between i and j can be found in $O(\lg n)$ time by reading M. Since the number of blocks is $O(\frac{n}{\lg n})$, the size of M is $O(\frac{n}{\lg n} \lg \frac{n}{\lg n}) = O(n)$.

References

[1] Jon Louis Bentley and Andrew Chi-Chih Yao. “An Almost Optimal Algorithm for Unbounded Searching”. In: *Information Processing Letters* 5.3 (August 1976), pages 82–87. ISSN: 0020-0190. DOI: 10.1016/0020-0190(76)90071-5. URL: http://dx.doi.org/10.1016/0020-0190(76)90071-5 (cited on page 8).

[2] Richard S. Bird. “Improving Saddleback Search: A Lesson in Algorithm Design”. In: *Mathematics of Program Construction, MPC*. Volume 4014. Lecture Notes in Computer Science. Springer, 2006, pages 82–89. ISBN: 978-3-540-35631-8. DOI: 10.1007/11783596_8. URL: http://dx.doi.org/10.1007/11783596_8 (cited on page 11).

[3] Bhadrachalam Chitturi et al. "An (18/11)n upper bound for sorting by prefix reversals". In: *Theoretical Computer Science* 410.36 (August 2009), pages 3372–3390. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2008.04.045. URL: http://dx.doi.org/10.1016/j.tcs.2008.04.045 (cited on page 9).

[4] Greg N. Frederickson. "Optimal Algorithms for Tree Partitioning". In: *ACM/SIGACT-SIAM Symposium on Discrete Algorithms, SODA*. January 1991, pages 168–177. URL: http://dl.acm.org/citation.cfm?id=127787.127822 (cited on page 11).

[5] Allan Grønlund Jørgensen and Seth Pettie. "Threesomes, Degenerates, and Love Triangles". In: *IEEE Annual Symposium on Foundations of Computer Science, FOCS*. October 2014, pages 621–630. DOI: 10.1109/FOCS.2014.72. URL: http://dx.doi.org/10.1109/FOCS.2014.72 (cited on page 7).

[6] Richard M. Karp, Scott Shenker, and Christos H. Papadimitriou. "A simple algorithm for finding frequent elements in streams and bags". In: *ACM Transactions on Database Systems* 28 (March 2003), pages 51–55. ISSN: 0362-5915. DOI: 10.1145/762471.762473. URL: http://doi.acm.org/10.1145/762471.762473 (cited on page 9).

[7] Jayadev Misra and David Gries. "Finding Repeated Elements". In: *Science of Computer Programming* 2.2 (November 1982), pages 143–152. ISSN: 0167-6423. DOI: 10.1016/0167-6423(82)90012-0. URL: http://dx.doi.org/10.1016/0167-6423(82)90012-0 (cited on page 8).

[8] A. Yu. Nesterenko. "Cycle detection algorithms and their applications". In: *Journal of Mathematical Sciences* 182.4 (April 2012), pages 518–526. ISSN: 1573-8795. DOI: 10.1007/s10958-012-0755-x. URL: http://dx.doi.org/10.1007/s10958-012-0755-x (cited on page 11).

Chapter 3

Problems on graphs

3.1 Tree

Exercise 23 Vertex cover on trees (*NTUT CSIE 101*)

Let T be an n-node tree rooted at some node r. We want to place as few guards as possible on nodes in T, such that every edge of T is guarded: an edge between a parent node v and its child w is guarded if one places a guard on at least one of these two nodes v, w. Give an $O(n)$ time algorithm for finding an optimal solution to the problem. Please show the analysis on the time and correctness of your algorithm

Answer of exercise 23

For any edge incident with a leaf, since the edge should be guarded, either the leaf node or its parent node should be chosen. Choosing the parent node not only can guard this edge but only guard the edge incident with the parent node. Thus, choosing the parent node is no worse than choosing the leaf node. Hence, we can choose all nodes that connect to leaf nodes. After removing all edges that are guarded, we get a smaller tree. We can recursively apply this procedure to get a minimum vertex cover of the tree. The running time is $O(n)$.

3.1.1 Lowest common ancestor

Exercise 24

1. Let T be a binary tree rooted at r with vertex set V and edge set E. Suppose it is represented using adjacency list format. If node u is an ancestor of v, there is a path from r to v passing through u. Consider the function ancestor(u, v) which returns TRUE if u is a ancestor of v and FALSE otherwise. In order to have this function run in $O(1)$ time, we are asked to design an algorithm to preprocess the tree. Please provide a linear-time, i.e., $O(|V| + |E|)$ time algorithm for this preprocess. [NTUT CSIE 100]
2. a) Let T be a binary search tree, where each vertex contains a pointer to its parent and pointers to its children, and also a field named temp, which is of type integer. You are given two pointers q_1, q_2, pointing to two vertices v_1, v_2 in T. Find in time $O(k)$ what is the length of the shortest path in T connecting v_1 to v_2, where k is the length of this path. You may assume that before the execution of your program, the value of all the "temp" fields is zero, and you can use these fields for your algorithm.
   b) Same as above, but this time the "temp" fields do not exist, you cannot write on the tree (so its information is "read-only" and the expected running time of your algorithm should be $O(k)$. Hint: Assume that T is stored in the memory of your computer and you can find, in time $O(1)$, what is the address in which each node is stored. [CCU CSIE 93]

Answer of exercise 24

Problem 1 We can use the pre-order traversal and post-order traversal on the tree to get each node's pre-order and post-order numbers. A node u is an ancestor of a node v if and only if u's pre-order number is smaller than v's pre-order number and u's post-order number is larger than v's post-order number. Since pre-order traversal and post-order traversal take linear time, the preprocess takes linear time.

Problem 2 Question a: Initially, let two pointers a and b point to v_1 and v_2 respectively and set their temp fields to 1. In each iteration, move a and b to their parent nodes respectively. If their temp fields are zero, then set them to one and repeat; otherwise, we find the solution. Suppose that a's temp field is one without loss of generality, then a is the lowest common ancestor of v_1 and v_2. If a path between v_1 and v_2 of length k exists, then the process will terminate in k iterations. Thus, the running time is $O(k)$.

Question b: If the temp field does not exist, then we can use a hashtable instead. Thus, the running time is $O(k)$ in expectation assuming universal hash function is used.

3.2 Traversal

3.2.1 DFS

Exercise 25

1. An undirected graph $G = (V, E)$ is stored in a text file with the following format: The first line contains two integer numbers n and m that denote the numbers of vertices and edges of G respectively. Then, the first line is followed by m lines. Each line contains two distinct integers, say i and j, indicating that there is an edge between vertices i and j. Given such a file, design an $O(n)$ time algorithm to test if the undirected graph represented by the file is a tree. You should specify the data structure used to store the graph in the memory and how you construct such a data structure. [NCU CSIE 96, NTU CSIE 99]
2. For an undirected graph $G = (V, E)$, a vertex $v \in V$, and an edge $(x, y) \in E$, let $G \setminus v$ denote the subgraph of G obtained by removing v and all the edges incident to v from G; and let $G \setminus (x, y)$ denote the subgraph of G obtained by removing the edge (x, y) from G. If G is connected, then $G \setminus v$ can be disconnected or connected.
   a) Given a connected graph G, design an $O(|V|)$ time algorithm to find a vertex $v \in G$ such that $G \setminus v$ is connected.
   b) Given a connected graph G, design an $O(|V|)$ time algorithm to either find an edge $(x, y) \in G$ such that $G \setminus (x, y)$ is connected or report that no such an edge exists. [NCU CSIE 102]

Answer of exercise 25

Problem 1 Without loss of generality, suppose that $m = n - 1$, otherwise it is not a tree obviously. Since a tree is a connected graph without a cycle, we can use DFS to check whether the graph contains a cycle. Because DFS takes $O(|V| + |E|)$ time, the running time is $O(n)$.

Problem 2 I do not have any solution for Question a. For Question b, use DFS to find a cycle and any edge on this cycle can be removed without disconnecting the graph. If no cycle exists, this graph is a tree and every edge cannot be removed. The running time is $O(|V|)$.

3.2.2 BFS

Exercise 26

1. Given is a directed graph $G = (V, E)$ represented via adjacency lists and a vertex $v_a \in V$. Design an algorithm that outputs the length of the shortest cycle containing v_a in G. your algorithm should solve the problem in $O(|V| + |E|)$ time. [NTHU CSIE 95]

2. We have a directed graph $G = (V, E)$ represented using adjacency lists. The edge costs are integers in the range $\{1, 2, 3, 4, 5\}$. Assume that G has no self-loops or multiple edges. Design an algorithm that solves the single-source shortest path problem on G in $O(|V| + |E|)$. [NTHU CSIE 95]

Answer of exercise 26

Problem 1 Use BFS to traverse from v_a. Find the first back edge from vertex u back to v_a. This cycle is the shortest cycle containing v_a. The running time is $O(|V| + |E|)$.

Problem 2 The idea is to transform the problem to an unweighted graph and use BFS to find the shortest path. Construct a graph $G' = (V \cup V', E')$ as follows:. For every edge $(u, v) \in E$ of weight k, if $k = 1$, then add an edge (u, v) to E'. Otherwise, add $k-1$ vertices $uv_1 \sim uv_{k-1}$ to V' and add edges (u, uv_1), (uv_{k-1}, v), and (uv_i, uv_{i+1}) for all $1 \leq i \leq k-2$ to E'. This transformation does not create path between any vertex in V. Moreover, if one path from u to v of cost C exists in G, then one path from u to v of C edges also exists in G'. Since k is at most 5, the size of V' and E' is $O(|E|)$. Thus, the running time is $O(|E|)$.

3.2.3 Topological sort

Exercise 27 (*CYCU CSIE 92*)

Professor Lee wants to construct the tallest tower possible out of building blocks. She has n types of blocks, and an unlimited supply of blocks of each type. Each type-i block is rectangular solid with linear dimension (x_i, y_i, z_i). A block can be oriented so that any two of its three dimensions determine the dimensions of a base and the other dimension is the height. In building a tower, one block may be placed on top of another block as long as the two dimensions of the lower block. (Thus, for example, blocks oriented to have equal-sized bases cannot be stacked.) Use graph model to design an efficient algorithm to determine the tallest tower that the professor can build. Analyze the run time complexity.

Answer of exercise 27

For each type i block, construct three nodes, v_{i1}, v_{i2}, and v_{i3}, corresponding to three faces, $x_i \times y_i$, $y_i \times z_i$, and $x_i \times z_i$. If one type j block can be placed on top of a type i block, then create the edges between the corresponding faces and the edge's weight is the height of type j block. The longest path in this graph is the tallest tower can be built. Since the graph is a directed acyclic graph, we can find the longest path in $O(|V| + |E|)$ time by using topological sort. Bacause the graph has $3n$ vertices and $O(n^2)$ edges, the running time is $O(n^2)$.

3.3 Path

Exercise 28 Johnson's algorithm (*NTPU CSIE 100*)

Given a graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}$, describe a method to decide whether there is a function $h : V \to \mathbb{R}$ such that the new weight function w_h defined by $w_h = w(u, v) + h(u) - h(v)$ is non-negative.

Answer of exercise 28

Pick a vertex s and solve shortest path problem from s. If no negative cycle exists in the graph, the shortest distance between s and vertex v is a candidate of $h(v)$, since we know that $h(v) \leq h(u) + w(u, v)$ by the property of the shortest paths. On the other hand, if a negative cycle exists in G, the function h can not exist.

Exercise 29 Arbitrage (*NCTU CSIE 96*)

Given an N by N positive matrix R (i.e., each entry $R[I, J]$ is positive) design an efficient algorithm to determine whether or not there exists a sequence of distinct indices: $I_1, I_2, \ldots, I_k$, where $1 \leq k \leq N$, such that $R[I_1, I_2] \times R[I_2, I_3] \times \cdots \times R[I_{k-1}, I_k] \times R[I_k, I_1] > 1$. State your algorithm precisely and analyze the running time of your algorithm. [NCTU CSIE 96]

Answer of exercise 29

The idea is to reduce this problem to finding a negative weight cycle in graph and use Bellman-Ford algorithm to solve it. Let $A[i, j] = -\lg R[i, j]$. We know that one sequence satisfies the problem's criterion, if and only if, one negative weight cycle exists in the graph represented by A. Since the transformation take $O(|V|^2)$ time and the running time of Bellman-Ford algorithm is $O(|V||E|) = O(|V|^3)$, the running time is $O(|V|^3)$.

3.4 Spanning tree

Exercise 30

1. Consider the following variation of the Minimum Spanning Tree problem: Given a graph G of n vertices and m edges AND a minimum spanning tree T of graph G, we wish to add new edge e with weight w_e to G forming a new graph G' and construct the new minimum spanning tree of the new graph G'. Give an algorithm which constructs the minimum spanning tree of G' in $O(n)$ time. [NCU CSIE 102]
2. Suppose that a graph G has a minimum spanning tree already computed. How quickly can the minimum spanning be updated if a new vertex and incident edges are added to G? Please justify your answer. [NTUT CSIE 98]

Answer of exercise 30

Problem 1 Add e to T to form a cycle. Remove the largest weight edge in this cycle then we get the new minimum spanning tree.

Problem 2 Let T be the original MST. When we add a new vertex v to T, we need to check all cycles and then remove the edge with the largest weight from each cycle. This can be done by using DFS on T as follows: during the recursive call of DFS, we maintain the edge with the largest weight, m, from the current node, r, to v through all explored descents. For a new explored descent d, we find the edge with the largest weight, t, from the descent to v through its subtree. Since m, t, and (r, d) will form a cycle, we only keep the smaller two among these three edges. The total complexity is $O(|T|)$.

Remark. After deleting edges/vertices, new MST can also be found efficiently [1, 2].

3.5 Matching

Exercise 31 Edge cover problem (*NTHU CSIE 101*)

Let $X = \{1, \ldots, n\}$. For a subset of X, we say that it covers its elements. Given a set $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of m subsets of X such that $\cup_{i=1}^{m} S_i = X$, the set cover problem is to find the smallest subset T of S whose union is equal to X, that is, $\cup_{S_i \in T} S_i = X$. Suppose that each subset $S_i \in \mathcal{S}$ contains only two elements. Can the set cover problem then be solved in polynomial time? If yes, please also design a polynomial-time algorithm to solve this set cover problem and analyze its time complexity.

Answer of exercise 31

Since each subset S_i has only two elements, we can create a graph $G = (X, S)$. Finding the minimum set cover problem is equal to finding the minimum edge cover on G, which can be solved in $O(n^4)$ time.

3.6 Network flow

Exercise 32 (*NTU CSIE 100*)

The escape problem is defined as the following. An $n \times n$ grid is an undirected graph consisting of n rows and n columns of vertices. We denote the vertex in the i-th row and j-th column by (i, j). All vertices in a grid have exactly four neighbors, except for the boundary vertices, which are the vertices (i, j) for which $i = 1$, $i = n$, $j = 1$, or $j = n$. Given $m \leq n^2$ starting vertices in the grid, the escape problem is to determine whether or not there are m vertex-disjoint paths from the starting vertices to any m different vertices on the boundary. Vertex-disjoint paths mean that each vertex can be used at most once in the escape. Show how to convert the escape problem into the maximum flow problem. It is enough to give the conversion procedure. It is not require to show the correctness of your procedure.

Answer of exercise 32

We can create a graph $G = (V, E)$, where V consists of all vertices on the grid and two new vertices s and t. For every adjacent pair of vertices, we create an edge on G. We connect s to every starting vertices and connect all boundary vertices to t. Set the node capacity for each node be one. If the maximum flow from s to t equals m, then m vertex-disjoint path exist.

Although traditional network has only edge capacity, we can transform node-capacitated network into edge-capacitated graph. For each vertex v, we create two vertices v' and v'', such that all the incoming edges of v connect to v' and all outgoing edges of v connect to v''. Moreover, we connect v' to v'' with edge capacity equals to v's node capacity. It is easy to see that a feasible flow in the transformed network can be transformed to a feasible flow in the original network, and *vice versa*.

3.7 Others

Exercise 33 Strong orientation (*NCKU IM 99*)

Suppose you are asked to assign direction for each edge in the graph to make it a digraph such that each vertex can connect to each other vertex by some directed graph (i.e. strongly connected). How do you know whether such strongly connected orientation exists for an undirected graph G of n vertices and m edges. Explain your method and discuss its complexity.

Answer of exercise 33

We can prove that a graph $G = (V, E)$ has a strong orientation if and only if G has no bridge. If G has a bridge, then it is impossible to assign the direction to become strongly connected. On the other hand, suppose that G has no bridge but the orientation does not exist. One maximum orientable subgraph H of G must exist. For a vertex $v \in V - H$, since G is 2-edge-connected, two paths from v to some vertex $u \in H$ must exist as well. Let the two paths be P and Q and let the first vertex of P and Q that enter H be p and q. Then, we can construct orientation of paths from v to p and from q to v. Thus, we construct a larger subgraph with orientation, which is impossible. Finally, testing for 2-edge-connectivity can be done in $O(n + m)$ time.

Remark. This theorem is called Robbins theorem.

Exercise 34 (*NCKU CSIE 100*)

A tournament $T = (V, E)$ is a simple digraph of $|V| = n$ vertices and $|E| = \frac{n(n-1)}{2}$ edges, suppose you already know outdeg[i], the outdegree for each vertex i. A tournament is transitive, whenever edge $(u, v) \in E$ and $(v, w) \in E$ implies $(u, w) \in E$. In other words, if there exists any 3 vertices i, j, k in T with edges (i, j), (j, k), and (k, i), then T is NOT transitive. Now you want to check whether T is transitive or not.

Answer of exercise 34

A naive solution is to generate all 3-tuple (i, j, k) and check. This method needs $O(n^3)$ time. We know that a tournament is transitive if and only if T is acyclic. Thus, we can use DFS to test whether the graph is acyclic in $O(n^2)$ time. Moreover, we know that a tournament is transitive if and only if all values in outdegree are distinct. Since the range of outdegree is from 0 to $n-1$, we can use an array to test whether all elements are distinct in $O(n)$ time.

Exercise 35 (*NCU CSIE 95*)

Given an undirected graph $G=(V,E)$ with $n=|V|$ vertices, four vertices of G, say, u, v, x, and y, are said to form a 4-cycle if (u,v), (v,x), (x,y) and (y,u) are in E. Consider the problem of determining whether G contains a 4-cycle. A naive method by checking all possible 4-combinations of the vertex set will need $\Omega(n^4)$ time to complete the job. Design a more efficient algorithm (i.e., the time complexity of your algorithm should be $O(n^k)$ with $k<4$) to solve the problem. Analysis the execution time of your algorithm.

Answer of exercise 35

Construct another matrix A with all zeros. For each vertex u, for each pairs of u's neighbors, v and x, we look the value of $A[v][x]=y$. If y is zero, then set $A[v][x]$ to be u, which means that v and x have a common neighbor u. If y is non-zero, then we found a 4-cycle u, v, y, x. Since each time we either find a 4-cycle or change the value of one entry in A, the running time is $O(n^2)$.

Remark. For any even number k, finding a cycle of length k can be done in $O(n^2)$ time [4]. Finding a path of length at least $k>0$ can be done in $O^*(k^2)$ [3].

References

[1] Francis Y. L. Chin and David Houck. "Algorithms for Updating Minimal Spanning Trees". In: *Journal of Computer and System Sciences* 16.3 (June 1978), pages 333–344. ISSN: 0022-0000. DOI: 10.1016/0022-0000(78)90022-3. URL: http://dx.doi.org/10.1016/0022-0000(78)90022-3 (cited on page 18).

[2] B. Das and Michael C. Loui. "Reconstructing a Minimum Spanning Tree after Deletion of Any Node". In: *Algorithmica* 31.4 (2001), pages 530–547. ISSN: 0178-4617. DOI: 10.1007/s00453-001-0061-3. URL: http://dx.doi.org/10.1007/s00453-001-0061-3 (cited on page 18).

[3] Ryan Williams. "Finding paths of length k in $O^*(2^k)$ time". In: *Information Processing Letters* 109.6 (February 2009), pages 315–318. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2008.11.004. URL: http://dx.doi.org/10.1016/j.ipl.2008.11.004 (cited on page 20).

[4] Raphael Yuster and Uri Zwick. "Finding Even Cycles Even Faster". In: *SIAM Journal on Discrete Mathematics* 10.2 (1997), pages 209–222. ISSN: 1095-7146. DOI: 10.1137/S0895480194274133. URL: http://dx.doi.org/10.1137/S0895480194274133 (cited on page 20).

Chapter 4

Problems on computational geometry

4.1 Sweep line

** Exercise 36 Power diagrams (*NTU CSIE 103*)

Suppose that you have n circles on a 2D plane. The radius and the center coordinate of each circle can be retrieved in $O(1)$ time. A closed region is defined as a non-empty set of connected 2-D points, and each point is covered by at least one circle.

1. Your task is to find the number of closed regions. Describe your algorithm and data structure in detail. What is the time complexity of your algorithm.

2. Now we start to add more circles one-by-one to the plane. After each addition, we want to keep track of the number of closed regions. Describe an algorithm and data structure to do so. What is the time complexity of your algorithm for each addition.

Answer of exercise 36

This problems can be solved by using power diagrams, a generalization of Voronoi diagrams. For the first problem, we build a power diagram and then construct the dual graph of the diagram, where each vertex represents a circle. Then, remove each edge of two non-intersecting circles. The number of components of the resulting graph is the answer. Since power diagram can be built in $O(n \lg n)$ time and the size of the graph is $O(n)$, the running time is $O(n \lg n)$. For the second problem, we need to build the power diagram incrementally and this problem can also be done in $O(n \lg n)$ [1].

Exercise 37 (*NCTU CSIE 93*)

The input is a set of n rectangles all of whose edges are parallel to the axes. Design an $O(n \lg n)$ algorithm to mark all the rectangles that are contained in other rectangles.

Answer of exercise 37

Use the sweep line algorithm. Let the position of left-top point and the right-bottom point of i-th rectangle be (L_i^x, L_i^y) and (R_i^x, R_i^y) respectively. Sort all L's and R's by increasing x-coordinate. If some L's or R's have the same x-coordinate, then sort them by decreasing y-coordinate. According to this ordering, insert or delete all vertical edges of rectangles into an interval tree I one by one.

When encountering a point (L_i^x, L_i^y), we can insert the left edge of rectangle i into I and test whether it is contained by some other left edge of rectangle j. If so, we can determine whether rectangle j contains rectangle i by their right edge. When encountering a point (R_i^x, R_i^y), we delete the left edge of rectangle i from I. The sorting only takes $O(n \lg n)$ time. Since each insertion and deletion takes $O(\lg n)$ time and the algorithm inserts and deletes $O(n)$ times, the running time is $O(n \lg n)$.

4.2 Convex hull

Exercise 38 Farthest pair (*NTU CSIE 97*)

Let the input set X consists of n points on the 2-dimensional plane with integral coordinates. The *farthest pair problem* is to identify two points in X whose Euclidean distance is maximum over all pairs X. You are also asked to prove or disprove that the farthest pair problem can be solved in $O(n \lg n)$ time. [NTU CSIE 97]

Answer of exercise 38

Problem 38 The first step is to find the convex hull, since the farthest pair must be on the convex hull. Then, apply the technique rotating calipers and find the farthest pair of points on the convex hull. Since finding the convex hull takes $O(n \lg n)$ time and rotating calipers takes linear time, the running time is $O(n \lg n)$.

4.3 Duality

Exercise 39

Show how to determine in $O(n^2 \lg n)$ time whether any three points in the set $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ are collinear. [NTU CSIE 92]

Answer of exercise 39

First, transform the points to dual lines. If there are three lines intersect at the same point on the dual plane, then the three points in the original plane are collinear. We can construct the arrangement of lines to find whether there are three lines intersect at the same point. The running time is $O(n^2)$.

Remark. Building the arrangement of lines can be done in $O(n^2)$ time in linear space [2].

4.4 Others

Exercise 40 Maximal points (*NCU CSIE 98*)

In a 2D plane, we say that a point (x_1, y_1) dominates (x_2, y_2) if $x_1 > x_2$ and $y_1 > y_2$. A point is called a *maximal point* if no other point dominates it. Given a set of n points, the maxima finding problem is to find all of the maximal points.

1. Write a divide and conquer algorithm to solve the maxima finding problem with the time complexity $O(n \lg n)$.
2. Show that your algorithm is indeed of the time complexity $O(n \lg n)$. [NCU CSIE 98]

Answer of exercise 40

The idea is based on divide and marriage before conquest method. First, sort the points by increasing x-coordinate. Then, we recursively find maximal points. If there are only two points, then finding the maximal points is easy.

When the number of points is more than two, divide the points into two parts, L and R, by the x-coordinate, such that $||L| - |R|| \leq 1$. We first find R's maximal points R_m. Let y_m be the maximum y-coorinate in R_m. Since each point in L has smaller x-coordinate than any points in R_m, we remove all points in L with y-coordinate smaller than y_m. Then, we recursively find the maximal points in the remaining points in L.

Sorting takes $O(n \lg n)$ time. Since merging two solutions takes linear time, the recurrence relation of the running time is $T(n) = 2T(\frac{n}{2}) + O(n)$. By using the master theorem, we prove that the running time is $O(n \lg n)$.

Remark. In each recursive call, we will find at least one maxima in R, and we either find one maxima in L or all points in L will be removed. The running time actually is $O(n \lg h)$, where h is the number of maximal points [3].

References

[1] Franz Aurenhammer. "Improved Algorithms for Discs and Balls Using Power Diagrams". In: *Journal of Algorithms* 9.2 (June 1988), pages 151–161. ISSN: 0196-6774. DOI: 10.1016/0196-6774(88)90035-1. URL: http://dx.doi.org/10.1016/0196-6774(88)90035-1 (cited on page 21).

[2] Herbert Edelsbrunner and Leonidas J. Guibas. "Topologically Sweeping an Arrangement". In: *Journal of Computer and System Sciences* 38.1 (February 1989), pages 165–194. ISSN: 0022-0000. DOI: 10.1016/0022-0000(89)90038-X. URL: http://dx.doi.org/10.1016/0022-0000(89)90038-X (cited on page 22).

[3] David G. Kirkpatrick and Raimund Seidel. "Output-size sensitive algorithms for finding maximal vectors". In: *Symposium on Computational Geometry, SoCG*. June 1985, pages 89–96. DOI: 10.1145/323233.323246. URL: http://doi.acm.org/10.1145/323233.323246 (cited on page 23).

Chapter 5

Algorithm design problems

5.1 Greedy

Exercise 41 (*NTU CSIE 97, NTNU CSIE 97*)

Consider the following scheduling problem. Suppose a man has several jobs waiting for his treatments. Each job takes one unit of time to finish and has a deadline and a profit. He can only do one job at any time. If a job starts before or at its deadline, its profit is obtained. The goal is to schedule the jobs so as to maximize the total profit. But not all jobs have to be scheduled. Please design an efficient algorithm to find a schedule that maximizes the total profit.

Answer of exercise 41

The idea is based on the greedy method. Sort the jobs by increasing deadline and process jobs in the sorted order. Let the schedule be empty initially. When considering the job j, if assigning job j to the last slot will not violate the constraint, then assign job j to the latest time slot; otherwise if there is a job k in schedule that has profit smaller than job j, then replace job k by job j. We prove the correctness of this algorithm by contradiction. Suppose that there exists an optimal solution that is different from the solution found by the greedy algorithm, then it has some jobs that we did not select. The most profitable job among these unselected jobs is assigned into an optimal schedule but unassigned in our schedule; however, it is impossible. The running time is $O(n \lg n)$.

5.2 Dynamic programming

Exercise 42 (*CYCU CSIE 90*)

A one way railway has n stops. Suppose that for all $i < j$, the price of the ticket fro the i-th stop to j-th stop is know, denoted $\text{cost}(i, j)$. (There is no traffic in the reverse direction since the railway is one-way.) Apply the dynamic programming technique to design your algorithm that outputs the minimum travel cost from stop 1 to stop n, and all the intermediate stops that the travel takes. What is the time complexity of your algorithm.

Answer of exercise 42

Let $F(i, j)$ be the minimum cost of traveling from stop i to stop j. The recurrence relation of F is

$$F(i,j) = \begin{cases} 0 & \text{if } i = j \\ \min(\min_k F(i,k) + F(k,j), C_{ij}) & \text{if } i < j. \end{cases}$$

The running time is $O(n^3)$.

Exercise 43 (*NCNU CSIE 93*)

Suppose that we cut a stick of length L (a positive integer) with the probability P at each position such that its distance from the left end is a positive integer. Design an efficient dynamic programming algorithm for calculating the probability that a stick of length at least n remains.

Answer of exercise 43

Let $F(k,n)$ be the probability of cutting stick of length k such that at least length n remains. The recurrence relation of F is

$$F(k,n) = \begin{cases} 0 & \text{for all } 1 \leq k \leq n-1 \\ (1-P)^{n-1} + \sum_{i=1}^{n-1} P(1-P)^{i-1} F(k-i-1,n)+ & \text{for all } k \geq n. \end{cases}$$

We can use this recurrence relation to calculate the probability.

Chapter 6

Complexity theory

6.1 NP-completeness

6.2 Approximation algorithms

Exercise 44 Christofides algorithm (*NCU CSIE 100*)

Let V be a set of n points in the plane. Let $G = (V, E)$ be the complete graph over V, and the weight of each edge $e \in E$ is the length of this edge. The Euclidean Traveling Salesman Problem (ETSP) of V is to find the cycle C^* such that C^* visits each node exactly once, and it has the minimum weight among all such cycles. Let T^* be a minimum spanning tree of G.

1. Show that $w(T^*) \leq w(C^*)$, where $w(X)$ is the weight of a subgraph X of G.
2. Given T^*, design an algorithm to compute a cycle C that is a 2-approximation of the optimal ETSP cycle C^*. Namely, $w(C^*) \leq w(C) \leq 2w(C^*)$. (Hint: first show that $w(T^8) \leq w(C^*) \leq 2w(T^*)$.)

[NCU CSIE 100]

Answer of exercise 44

This algorithm is called Christofides algorithm.