
Introduction to Combinatorics

In discrete optimization, some or all of the variables in a model are required to belong to a discrete set; this is in contrast to continuous optimization in which the variables are allowed to take on any value within a range of values. There are two branches of discrete optimization: integer programming and combinatorial optimization where the discrete set is a set of objects, or combinatorial structures, such as assignments, combinations, routes, schedules, or sequences. Combinatorial optimization is the process of searching for maxima (or minima) of an objective function F whose domain is a discrete but large configuration space (as opposed to an N -dimensional continuous space). Typical combinatorial optimization problems are the travelling salesman problem (“TSP”), the minimum spanning tree problem (“MST”), and the knapsack problem. We start with basic combinatorics which is able to enumerate the all solutions exhaustively. Later on, other chapters we will dive into different combinatorial/discrete optimization problems.

Combinatorics, as a branch in mathematics that mainly concerns with counting and enumerating, is a means in obtaining results, and certain properties of finite structures. Combinatorics is used frequently in computer science to obtain formulas and estimates in both the design and analysis of algorithms. It is a broad and thus seemingly hard to define topic that can solve the following types of questions:

- The counting or enumerating of specified structures, sometimes referred to as arrangements or configurations in a very general sense, associated with finite systems,
- the existence of such structures that satisfy certain given criteria, this is usually called Constraint Restricted Problems (CSPs).

- optimization, finding the “best” structure or solution among several possibilities, be it the “largest”, “smallest” or satisfying some other optimality criterion.

In this section, we introduce common combinatorics that can help us come up with the simplest which potentially be quite large state space. At least, this is the first step, and solving a small problem in this way might offer us more insights on continuing finding a better solution.

When the situation is easy, we can mostly figure out the counting with some logic and get a closed-form solution; when the situation is more complex such as in the *partition* section, we detour by using recurrence relation and math induction.

0.1 Permutation

Given a list of integer $[1, 2, 3]$, how many way can we order these three numbers? Imagine that we have three positions for these three integers. For the first position, it can choose 3 integers, leaving the second position with 2 options. Further, when it reaches to the last position, it can only choose whatever that is left, we have 1. The total count will be $3 \times 2 \times 1$.

Similarly, for n distinct numbers, we will get the number of permutation easily as $n \times (n - 1) \times \dots \times 1$. A factorial, denoted as $n!$, is used to abbreviate it. Worth to notice, the factorial sequence grows even quicker than the exponential sequence, such as 2^n .

0.1.1 n Things in m positions

Permutation of n things on n positions is denoted as $p(n, n)$. Think about what if we have $m \in [1, n - 1]$ positions instead? How to get a closed-form function for $p(n, m)$. The process is the same: we fix each position and consider the number of choice of things each one has.

$$p(n, m) = n \times (n - 1) \times \dots \times (n - (m - 1)) \quad (1)$$

$$= \frac{n \times (n - 1) \times \dots \times (n - m + 1) \times (n - m) \times \dots \times 1}{(n - m) \times \dots \times 1} \quad (2)$$

$$= \frac{n!}{(n - m)!} \quad (3)$$

If we want $p(n, n)$ to follow the same form, it would require us to define $0! = 1$.



What if there are repeated things, that things are not distinct?

0.1.2 Recurrence Relation and Math Induction

The number or the full set of permutations can be generated incrementally. We demonstrate how with recurrence relation and math induction. We start from $P(0, 0) = 1$. Easily, we get $P(i, 0) = 1, i \in [1, n]$. With math induction, now assume we know $P(n-1, m-1)$, for the m -th position, what choice does it have? First, we need to pick this thing from the $n - (m-1)$ things. Then, we have $m-1$ things lined up linearly, there are m positions to insert the m -th item, resulting $P(n, m) = (n - m + 1) * mP(n-1, m-1)$.

Now, we can use iterative method to obtain the closed-form solution:

$$P(n, m) = (n - m + 1) * m * P(n-1, m-1) \quad (4)$$

$$= (n - m + 1) * m * (P(n-2, m-2)) \quad (5)$$

$$\dots \quad (6)$$

$$= m!P(n - m + 1, 1) \quad (7)$$

0.1.3 See Permutation in Problems

Suppose we want to sort an array of integers incrementally, say the array is $A = [4, 8, 2]$. The right order is $[2, 4, 8]$, which is trivial to obtain in this case. If we are about to form it as a search problem, we need to define a *search space*. Using our knowledge in combinatorics, we know all possible ordering of these numbers are $[4, 8, 2], [4, 2, 8], [2, 4, 8], [2, 8, 4], [8, 2, 4], [8, 4, 2]$. Generating all possible ordering and save it in an array maybe. Then this sorting problem is converted into checking which array is incrementally sorted. However, it comes with large price on space usage, since for n numbers there, the number of possible orderings are $n!$. A smarter way to do it is to check the ordering as we are generating the ordering set.

0.2 Combination


Same as before, we have to choose m things out of n but with one difference—the order does not matter, how many ways we have? This problem is called **combination**, and it is denoted as $C(n, m)$. For example, for $[1, 2, 3]$, $C(3, 2) = [1, 2], [2, 3], [1, 3]$. Comparatively, $P(3, 2) = [1, 2], [2, 1], [2, 3], [3, 2], [1, 3], [3, 1]$.

To get combination, we can leverage and apply permutation first. However, this results over-counting. As shown in our example, when there are two things in the combination, a permutation would double count it. If there are m things, we over count by $m!$ times. Therefore, if we divide the permutation by all permutation of m things, we get out formula for

combination:

$$C(n, m) = \frac{P(n, m)}{P(m, m)} \quad (8)$$

$$= \frac{n!}{(n - m)!m!} \quad (9)$$

 **Back to the last question, when there are repeats in the permutation. We can use the same idea. Assume we have n, m that n things in total but only m types and $a_i, i \in [1, m]$ to denote the number of each type, this means $a_1 + a_2 + \dots + a_m = n$. The number of ways to linearly order these objects is $\frac{n!}{a_1!a_2!\dots a_m!}$.**

The combination of k things out of n , will be the same as choosing $(n - k)$ things.

$$C(n, k) = C(n, n - k) \quad (10)$$

0.2.1 Recurrence Relation and Math Induction

We also show how the combination can be generated incrementally. We start from $C(0, 0) = 1$, and easily we get $C(n, 0) = 1$. Assume we know $C(n - 1, k - 1)$, now we need to add the k -th item into the combination? :

- Use k -th item, then we just need to put the k -th item into any sets in $C(n - 1, k - 1)$, resulting $C(n - 1, k - 1)$.
- Not use k -th item, this means we need to pick k items from the other $n - 1$ items, resulting $C(n - 1, k)$.

Thus, we have $C(n, k) = C(n - 1, k - 1) + C(n - 1, k)$, this is called **Pascal's Identity**.

What if things are not distinct?

0.3 Partition

We discuss three types of partitions: (1) integer partition, (2) set partition, and (3) array partition. In this section, counting becomes less obvious compared with combination and permutation, this is where we rely more on **recurrence relation** and **math induction**.

0.3.1 Integer Partition

Integer Partition Definition Integer partitions is to partition a given integer n into distinct subsets that add up to n .

For example, given $n=5$, the resulting partitioned subsets are these 7 subsets:

```
{5}
{4, 1},
{3, 2}
{3, 1, 1},
{2, 2, 1},
{2, 1, 1, 1},
{1, 1, 1, 1, 1}
```

Analysis Let us assume the resulting sequence is (a_1, a_2, \dots, a_k) , and $a_1 \geq a_2 \geq \dots \geq a_k \geq 1$, and $a_1 + a_2 + \dots + a_k = n$. The ordering is simply to help us to track the sequence. We use

The easiest way to generate integer partition is to construct them incrementally. We first start from the partition of n . For $n=5$, we get 5 first. Then we subtract one from the largest item that is larger than 1, and add it to the smallest item if it exists and that the resulting $s+1 < l$, $s < l-1$, and other option is to put it aside. For 5, there is no other item, so that it becomes 4, 1. For 4,1, following the same rule, we get 3, 2, for 3, 2, we get 3,1,1.

```
1 {5}, no other smaller item, put it aside
2 {4, 1}, satisfy  $s < l-1$ , become {3,2}
3 {3, 2}, not satisfy  $s < l-1$ , put it aside
4 {3, 1, 1}, satisfy  $s < l-1$ , add it to
5   {2, 2, 1}, not satisfy, put it aside
6   {2, 1, 1, 1}, not satisfy, put it aside
7 {1, 1, 1, 1, 1}
```



Try to generate the partition when $n=6$.

If we draw out the transfer graph, we can see a lot of overlapping of some state. Therefore, we add one more limitation on the condition, $s > 1$.

0.3.2 Set Partition

Set Partition Problem Definition How many ways exist to partition a set of n distinct items a_1, a_2, \dots, a_n into k nonempty subsets, $k \leq n$.

Here are 7 ways that we can partition the set $\{a_1, a_2, a_3, a_4\}$ into 2 nonempty subsets. They are

```
{a1}, {a2, a3, a4};
{a2}, {a1, a3, a4};
```

```
{a3}, {a1, a2, a4};
{a4}, {a1, a2, a3}
{a1, a2}, {a3, a4};
{a1, a3}, {a2, a4};
{a1, a4}, {a2, a3};
```

Let us denote the total ways as $s(n, k)$. As seen in the example, given 2 groups and 4 items, there are two combination of each group's size: 1+3 and 2+2. For combination 1,3, this is equivalent to choose one item from the set to put at the first subset $C(n, 1)$, and then choose 3 items for the other subset $C(3, 3)$. For combination 2, 2, we have $C(4, 2)$ for one subset and $C(2, 2)$ for the other subset. However, because the ordering of the subsets does not matter, we need to divide it by $2!$. The set partition problem thus consists of two steps:

- Partition n into k integers: This subroutine can be solved with integer partition we just learned. We have $b_1 + b_2 + \dots + b_k = n$.
- For each combination of integer partition, we compute the number of ways choosing b_i items for that set, we get $C(n, b_1) \times C(n - b_1, b_2) \times C(n - b_1 - b_2, b_3) \times \dots \times C(b_k, b_k)$. Now, we find the distinct b_i and its number of appearance in the sequence. If we have m distinct number denoted as b_i , and its count c_i , then we divide the above ways by $c_1!c_2!\dots c_m!$.

From this solution, it is hard to get a closed form for $s(n, k)$.

Find Recurrence Relation There is just one way to handle this problem, let us try the incremental method—find a recurrence relation. We first start with $s(0, 0) = 0$, and we can also easily get $s(n, 0) = 0$. Now, with the mathematical induction, we assume we solved a subproblem, say $s(n - 1, k - 1)$, can we induce $s(n, k)$? What do we need?

Now we have $n-1$ items in $k-1$ groups, now there is one addition group and one additional item. There are two ways:

- put the additional item into the additional group. In this way, $s(n, k)$ is simply the same as of $s(n - 1, k - 1)$.
- spread the $n-1$ items from the original $k-1$ groups into k groups, that is $s(n - 1, k)$ and our additional item has k options now, making $k \times s(n - 1, k)$ in total

Combing together the count of these two ways, we get a recurrence relation that

$$s(n, k) = s(n - 1, k - 1) + ks(n - 1, k) \quad (11)$$

0.4 Array Partition

Problem Definition How many ways exist to partition an array of n items a_1, a_2, \dots, a_n into subarrays. There are different subtypes depending on the number of subarrays, say m :

1. When the number m is as flexible as $m \in [1, n - 1]$.
2. When the number m is fixed as a number in range $[2, n - 1]$.

When the number of subarray is fixed For example, it is common to partition an array into 2 or 3 subarrays. First, we find an item a_i as a partition point, getting the last subarray $a[i : n]$ and left an array to further consider $a[0 : i]$. If $m = 2$, $a[0 : i]$ results the first subarray and the partition process is done. This gives out n ways of partition. When $m = 3$, we need to further partition $a[0 : i]$ into two parts. This can be represented with recurrence relation:

$$d(n, m) = (d(i, m - 1), a[i : n]), i \in [0, n - 1] \quad (12)$$

Further, for $d(i, m - 1)$:

$$d(i, m - 1) = (d(j, m - 2), a[j : n]), j \in [0, i - 1] \quad (13)$$

This can be done recursively: we will have a recursive function with depth m .

When the number of subarray is flexible The process is the same other than m can be as large as $n - 1$. If we are about to use dynamic programming, for all these states, we need to come up with an ordering of the state (i, j) , where i is the subproblem $a[0 : i]$, and j is the number of partitions. We imagine it as a matrix with i, j as row and column respectively:

	0	1	2	n-1:	partition
0	X	—	—	—	
1	X	X	—	—	
2	X	X	X		
n-1					
n	X	X	X	X	X

Does the ordering of the `for` loop matter? Actually it does not.

Applications There are many applications that involve splitting an array/string or cutting a rod. This relates to splitting type of dynamic programming.

0.5 Merge

0.6 More Combinatorics

Combinatorics is about enumerating specified structures, there are some structures are of our main interests through this book and often appears in the interviews, they are: *subarray*, *subsequence*, and *subsets*.

Subarray We have solved one example with subarray. Subarray is defined as a contiguous sequence in the array, which can be represented as $a[i, \dots, j]$. The number of subarray exist in an array of size n will be:

$$sa = \sum_{i=1}^{i=n} i = n * (n + 1) / 2 \quad (14)$$

A substring is a contiguous sequence of characters within a string. For instance, "the best of" is a substring of "It was the best of times". This is not to be confused with subsequence, which is a generalization of substring. For example, "Itwastimes" is a subsequence of "It was the best of times", but not a substring.

Prefix and suffix are special cases of substring. A prefix of a string S is a substring of S that occurs at the beginning of S . A suffix of a string S is a substring that occurs at the end of S .

Subsequence For a subsequence means any sequence we can find the array, which is not required to be contiguous, but the ordering still matters. For example, in the array of [ABCD], the subsequence will be

```

1      [ ] ,
2 [A] , [B] , [C] , [D] ,
3 [AB] , [AC] , [AD] , [BC] , [BD] , [CD] ,
4 [ABC] , [ABD] , [ACD] , [BCD] ,
5 [ABCD]
```

You would actually see for $n = 4$, there are 16 possible subsequence, which is 2^4 . This is not coincidence. Imagine for each item in the array, they have two options, either be chosen into the possible sequence or not chosen, which make it to 2^n .

$$ss = 2^n \quad (15)$$

Subset The Subset B of a set A is defined as a set within all elements of this subset are from set A . In other words, the subset B is contained inside the set A , $B \in A$. There are two kinds of subsets: if the order of the subset doesn't matter, it is a combination problem, otherwise, it is a permutation problem.

If it is the case that ordering does not matter, for n distinct things, the number of possible subsets, also called *the power set* will be:

$$power_{set} = C(n, 0) + C(n, 1) + \dots + C(n, n) \quad (16)$$