

Bitmask Dynamic Programming – Quy Hoạch Động Bitmask

Nguyễn Quân Bá Hồng*

Ngày 27 tháng 7 năm 2025

Tóm tắt nội dung

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:

URL: https://nqbh.github.io/advanced_STEM/.

Latest version:

- *Bitmask Dynamic Programming – Quy Hoạch Động Bitmask*.

PDF: URL: [.pdf](#).

T_EX: URL: [.tex](#).

- .

PDF: URL: [.pdf](#).

T_EX: URL: [.tex](#).

Mục tiêu của bài viết này là tổng hợp 1 số bài viết về quy hoạch động bitmask & dịch 1 số trang về bitmask DP viết bằng tiếng Anh, & bổ sung các giải thích mang tính cá nhân của tác giả để chuẩn bị cho kỳ thi Olympic Tin học Sinh viên & ICPC.

Mục lục

1 Preliminaries	1
1.1 Notation – Ký hiệu	1
2 Bitwise Operators – Toán Tử Bitwise	2
3 Introduction to Bitmask Dynamic Programming – Giới Thiệu Quy Hoạch Động Bitmask	4
3.1 Merging subsets – Gộp tập con	6
4 Some Applications of Bitmask Dynamic Programming – Vài Ứng Dụng của Quy Hoạch Động Bitmask	9
5 Miscellaneous	9

1 Preliminaries

1.1 Notation – Ký hiệu

- $\overline{m, n} := \{m, m+1, \dots, n-1, n\}$, $\forall m, n \in \mathbb{Z}$, $m \leq n$. Hence the notation “for $i \in \overline{m, n}$ ” means “for $i = m, m+1, \dots, n$ ”, i.e., chỉ số/biến chạy i chạy từ $m \in \mathbb{Z}$ đến $n \in \mathbb{Z}$. Trong trường hợp $a, b \in \mathbb{R}$, ký hiệu $\overline{a, b} := [\overline{a}], [b]$ có nghĩa như định nghĩa trước đó với $m := \overline{a}$, $n := [b] \in \mathbb{Z}$; khi đó ký hiệu “for $i \in \overline{a, b}$ ” với $a, b \in \mathbb{R}$, $a \leq b$ có nghĩa là “for $i = \overline{a}, \overline{a} + 1, \dots, [b] - 1, [b]$ ”, i.e., chỉ số/biến chạy i chạy từ \overline{a} đến $[b] \in \mathbb{Z}$.
- $\lfloor x \rfloor, \{x\}$ lần lượt được gọi là *phần nguyên & phần lẻ* (integer- & fractional parts) của $x \in \mathbb{R}$, see, e.g., [Wikipedia/floor & ceiling functions](#), [Wikipedia/fractional part](#).
- $x_+ := \max\{x, 0\}$, $x_- := \max\{-x, 0\} = -\min\{x, 0\}$ lần lượt được gọi là *phần dương & phần âm* (positive- & negative parts) của $x \in \mathbb{R}$.
- s.t.: abbreviation of ‘such that’.
- w.l.o.g.: abbreviation of ‘without loss of generality’.
- $|A|$ or $\#A$: the number of elements of a set A – số phần tử của 1 tập hợp A hữu hạn.
- $\text{card}(A)$: cardinality of a set A (finite or infinite) – lực lượng của 1 tập hợp A (hữu hạn hoặc vô hạn).

*A scientist- & creative artist wannabe, a mathematics & computer science lecturer of Department of Artificial Intelligence & Data Science (AIDS), School of Technology (SOT), UMT Trường Đại học Quản lý & Công nghệ TP.HCM, Hồ Chí Minh City, Việt Nam.
E-mail: nguyenquanbahong@gmail.com & hong.nguyenquanba@umt.edu.vn. Website: <https://nqbh.github.io/>. GitHub: <https://github.com/NQBH>.

- $[n] := \{1, 2, \dots, n\}$: the set of 1st $n \in \mathbb{N}^*$ positive integers, which serves as 1 of prototypical example of a finite set with n elements – tập hợp $n \in \mathbb{N}^*$ số nguyên dương đầu tiên, đóng vai trò là 1 tập hợp hữu hạn với n phần tử. Quy ước $[0] = \emptyset$ ký hiệu tập rỗng, i.e., tập hợp không chứa bất cứ phần tử nào.

Note: $[n]$ là ký hiệu ưa thích của dân Tổ hợp vì tập $[n]$ xuất hiện xuyên suốt trong các bài toán Tổ hợp với vai trò tập mẫu để biểu đạt số phần tử cần thiết.

- $A_n^k = \frac{n!}{(n-k)!}$: Chỉnh hợp chập k phần tử từ 1 tập hợp có n phần tử.
- $C_n^k = \binom{n}{k} = \frac{n!}{k!(n-k)!}$: Tổ hợp chập k phần tử từ 1 tập hợp có n phần tử.
- $P_n = n!$, $\forall n \in \mathbb{N}$: the number of permutations.
- O: Olympiad problem – Bài tập định hướng ôn luyện Olympic Toán Hoặc Olympic Tin.
- R: Research-oriented problems – Bài tập định hướng nghiên cứu. tBài tập hay các câu hỏi nhãn R, (R-labeled problems & R-labeled questions) thường sẽ có các bài báo nghiên cứu khoa học liên quan đính kèm.

Chiến thuật về cấu trúc nội dung: bitwise operators + dynamic programming \Rightarrow bitwise dynamic programming (kiểu như hợp thể tiến hóa của Pokemon hoặc Dragon Balls – 7 Viên Ngọc Rồng).

2 Bitwise Operators – Toán Tử Bitwise

Resources – Tài nguyên.

1. SIYONG HUANG, CHONGTIAN MA, MIHNEA BREBENEL. [USACO Guide/intro to bitwise operators](#).

Abstract. 6 bitwise operators & some of their common usages.

Problem 1 ([CodeForces/take a guess](#)). *This is an interactive task: WILLIAM has a certain sequence of integers a_1, \dots, a_n in his mind, but due to security concerns, he does not want to reveal it to you completely. WILLIAM is ready to respond to no more than $2n$ of the following questions:*

- What is the result of a bitwise AND of 2 items with indices $i \neq j$.
- What is the result of a bitwise OR of 2 items with indices $i \neq j$.

You can ask WILLIAM these questions & you need to find the k th smallest number of the sequence. Formally the k th smallest number is equal to the number at the k th place in a 1-indexed array sorted in non-decreasing order, e.g., in array $[5, 3, 3, 10, 1]$ 4th smallest number is equal to 5, & 2nd & 3rd are 3.

Input. *It is guaranteed that for each element in a sequence the condition $0 \leq a_i \leq 10^9$ is satisfied.*

Interaction. *In the 1st line you will be given 2 integers $n \in \overline{3, 10^4}, k \in [n]$, which are the number of items in the sequence a & the number k . After that, you can ask no more than $2n$ questions (not including **finish** operation). Each line of your output may be of 1 of the following types:*

- **or** $i \ j$ with $i, j \in [n], i \neq j$, where i, j are indices of items for which you want to calculate the bitwise OR.
- **and** $i \ j$ with $i, j \in [n], i \neq j$, where i, j are indices of items for which you want to calculate the bitwise AND.
- **finish** res , where res is the k th smallest number in the sequence. After outputting this line the program execution must conclude.

*In response to the 1st 2 types of queries, you will get an integer x , the result of the operation for the numbers you have selected. After outputting a line do not forget to output a new line character & flush the output buffer. Otherwise you will get the **Idleness limit exceeded**. To flush the buffer use:*

- `fflush(stdout)` in C++
- `System.out.flush()` in Java
- `stdout.flush()` in Python
- `flush(output)` in Pascal
- for other languages refer to documentation

*If you perform an incorrect query the response will be -1 . After receiving response -1 you must immediately halt your program in order to receive an **Incorrect answer** verdict.*

Hacking. To perform a hack you will need to use the following format: The 1st line must contain 2 integers $n \in \overline{3, 10^4}, k \in [n]$, which are the number of items in the sequence a & the number k . The 2nd line must contain n integers a_1, a_2, \dots, a_n with $a_i \in \overline{0, 10^9}, \forall i \in [n]$, the sequence a .

Sample.

take_guess.inp	take_guess.out
7 6	and 2 5
2	or 5 6
7	finish 5

Explanation. The hidden sequence is $[1, 6, 4, 2, 3, 5, 4]$.

Bài toán 1 (Đoán). Đây là một nhiệm vụ tương tác: WILLIAM có một dãy số nguyên a_1, \dots, a_n trong đầu, nhưng vì lý do bảo mật, anh ấy không muốn tiết lộ toàn bộ dãy số đó cho bạn. WILLIAM chỉ sẵn sàng trả lời tối đa $2n$ câu hỏi sau:

- Kết quả của phép toán AND trên bitwise của 2 phần tử có chỉ số $i \neq j$ là bao nhiêu?
- Kết quả của phép toán OR trên bitwise của 2 phần tử có chỉ số $i \neq j$ là bao nhiêu?

Bạn có thể hỏi WILLIAM những câu hỏi này & bạn cần tìm số nhỏ nhất thứ k của dãy số. Về mặt hình thức, số nhỏ nhất thứ k bằng số ở vị trí k trong một mảng 1 được sắp xếp theo thứ tự không giảm, ví dụ: trong mảng $[5, 3, 3, 10, 1]$, số nhỏ nhất thứ 4 bằng 5, & số thứ 2 & số thứ 3 là 3.

Input. Đảm bảo rằng với mỗi phần tử trong một dãy, điều kiện $0 \leq a_i \leq 10^9$ được thỏa mãn.

Interaction. Dòng đầu tiên, bạn sẽ được cung cấp 2 số nguyên $n \in \overline{3, 10^4}, k \in [n]$, là số phần tử trong dãy a & số k . Sau đó, bạn có thể hỏi không quá $2n$ câu hỏi (không bao gồm thao tác finish). Mỗi dòng đầu ra của bạn có thể thuộc một trong các kiểu sau:

- or $i \ j$ với $i, j \in [n], i \neq j$, trong đó i, j là chỉ số của các mục mà bạn muốn tính toán OR theo bit.
- and $i \ j$ với $i, j \in [n], i \neq j$, trong đó i, j là chỉ số của các mục mà bạn muốn tính toán AND theo bit.
- finish res, trong đó res là số nhỏ thứ k trong chuỗi. Sau khi xuất dòng này, chương trình phải kết thúc thực thi.

Đáp lại 2 kiểu truy vấn đầu tiên, bạn sẽ nhận được một số nguyên x , kết quả của phép toán cho các số bạn đã chọn. Sau khi xuất một dòng, đừng quên xuất một ký tự xuống dòng & xóa bộ đệm đầu ra. Nếu không, bạn sẽ nhận được thông báo Idleness limit exceeded. Để xóa bộ đệm, hãy sử dụng:

- fflush(stdout) trong C++
- System.out.flush() trong Java
- stdout.flush() trong Python
- flush(output) trong Pascal
- để biết các ngôn ngữ khác, vui lòng tham khảo tài liệu

Nếu bạn thực hiện một truy vấn không chính xác, phản hồi sẽ là -1 . Sau khi nhận được phản hồi -1 , bạn phải dừng chương trình ngay lập tức để nhận được kết quả Câu trả lời không chính xác.

Hacking. Để thực hiện hack, bạn cần sử dụng định dạng sau: Dòng đầu tiên phải chứa 2 số nguyên $n \in \overline{3, 10^4}, k \in [n]$, là số phần tử trong chuỗi a & số k . Dòng thứ 2 phải chứa n số nguyên a_1, a_2, \dots, a_n với $a_i \in \overline{0, 10^9}, \forall i \in [n]$, dãy số a .

Solution. We can figure out the sum of 2 numbers using just their AND, OR, & XOR values by the following property:

Lemma 1. $a + b = 2 \cdot (a \& b) + a \oplus b$.

Proof. $a \oplus b$ is essentially just $a + b$ in base 2 but we never carry over to the next bit. Recall a bit in $a \oplus b$ is 1 iff the bit in a is different from the bit in b , thus 1 of them must be a 1. However, when we add 2 1 bits, we yield a 0, but we do not carry that 1 to the next bit. This is where $a \& b$ comes in. $a \& b$ is just the carry bits themselves, since a bit is 1 only if it's a 1 in both a & b , which is exactly what we need. We multiply this by 2 to shift all the bits to the left by 1, so every value carries over to the next bit. To acquire the XOR values of the 2 numbers, we can use the following:

$$a \oplus b = \neg(a \& b) \& (a | b).$$

The proof is as follows. Recall a bit in $a \oplus b$ is 1 iff the bit in a is different from the bit in b . By negating $a \& b$, the bits that are left on are in the following format: (i) if it's 1 in a & 0 in b , or (ii) if it's 0 in a & 1 in b , or (iii) if it's 0 in a & 0 in b . But we need to get rid of the 3rd case. By taking the bitwise AND with $a | b$, the ones that are left on is only if there is a 1 in either a or b . Obviously, the 3rd case isn't included in $a | b$ since both bits are off, & we successfully eliminate that case.

– $a \oplus b$ về cơ bản chỉ là $a + b$ trong cơ số 2 nhưng chúng ta không bao giờ chuyển sang bit tiếp theo. Hãy nhớ lại một bit trong $a \oplus b$ là 1 nếu và chỉ khi bit trong a khác với bit trong b , do đó 1 trong số chúng phải là 1. Tuy nhiên, khi chúng ta cộng

2 bit 1, chúng ta sẽ thu được 0, nhưng chúng ta không chuyển 1 đó sang bit tiếp theo. Đây là lúc $a \& b$ xuất hiện. $a \& b$ chỉ là bản thân các bit nhớ, vì một bit chỉ là 1 nếu nó là 1 trong cả a & b , đó chính xác là những gì chúng ta cần. Chúng ta nhân số này với 2 để dịch chuyển tất cả các bit sang trái 1, vì vậy mọi giá trị đều được chuyển sang bit tiếp theo. Để có được các giá trị XOR của 2 số, chúng ta có thể sử dụng lệnh sau:

$$a \oplus b = \neg(a \& b) \& (a | b).$$

Chúng minh như sau. Nhớ lại một bit trong $a \oplus b$ là 1 nếu và chỉ khi bit trong a khác với bit trong b . Bằng cách phủ định $a \& b$, các bit còn lại sẽ có định dạng sau: (i) nếu 1 trong a & 0 trong b , hoặc (ii) nếu 0 trong a & 1 trong b , hoặc (iii) nếu 0 trong a & 0 trong b . Nhưng ta cần loại bỏ trường hợp thứ 3. Bằng cách thực hiện phép toán AND bitwise với $a | b$, các bit còn lại chỉ được xem xét nếu có 1 trong a hoặc b . Rõ ràng, trường hợp thứ 3 không được bao gồm trong $a | b$ vì cả hai bit đều bị tắt, & ta đã loại bỏ thành công trường hợp đó. \square

Now that we can acquire the sum of any 2 numbers in 2 queries, we can easily solve the problem now. We can find the values of the 1st 3 numbers of the array using a system of equations involving their sum (note $n \geq 3$). Once we have acquired their independent values, we can loop through the rest of the array.

– Bây giờ, khi đã có thể tính tổng của 2 số bất kỳ trong 2 truy vấn, chúng ta có thể dễ dàng giải quyết vấn đề. Chúng ta có thể tìm giá trị của 3 số đầu tiên trong mảng bằng cách sử dụng hệ phương trình liên quan đến tổng của chúng (lưu ý $n \geq 3$). Sau khi đã tính được các giá trị độc lập của chúng, chúng ta có thể lặp qua phần còn lại của mảng.

C++: <https://usaco.guide/silver/intro-bitwise?lang=cpp>.

```

1  #include <iostream>
2  #include <algorithm>
3  #include <vector>
4  using namespace std;
5
6  int ask(string s, int a, int b) {
7      cout << s << ' ' << a << ' ' << b << '\n';
8      int res;
9      cin >> res;
10     return res;
11 }
12
13 /** @return the sum of the elements at a and b (0-indexed) */
14 int sum(int a, int b) {
15     int and_ = ask("and", ++a, ++b);
16     int or_ = ask("or", a, b);
17     int xor_ = ~and_ & or_; // a ^ b = ~(a & b) & (a | b)
18     return 2 * and_ + xor_; // a + b = 2(a & b) + a ^ b
19 }
20
21 int main() {
22     int n, k;
23     cin >> n >> k;
24     // acquire 1st 3 elements
25     int a_plus_b = sum(0, 1);
26     int a_plus_c = sum(0, 2);
27     int b_plus_c = sum(1, 2);
28
29     // get actual values by solving equations
30     vector<int> arr{(a_plus_b + a_plus_c + b_plus_c) / 2};
31     arr.push_back(a_plus_b - arr[0]);
32     arr.push_back(a_plus_c - arr[0]);
33
34     // get rest of array
35     for (int i = 3; i < n; ++i) arr.push_back(sum(i - 1, i) - arr.back());
36     sort(arr.begin(), arr.end());
37     cout << "finish " << arr[k - 1] << '\n';
38 }

```

\square

3 Introduction to Bitmask Dynamic Programming – Giới Thiệu Quy Hoạch Động Bitmask

1. MICHAEL CAO, SIYONG HUANG, PENG BAI. [USACO Guide/bitmask DP](#).

2. [CodeDream/quy hoạch động xử lý bit \(DP bitmask\) phần 1](#).

Bitmask dynamic programming (abbr., bitmask DP) attacks DP problems that require iterating over subsets.

Problem 2 (CSES Problem Set/Hamiltonian flights). *There are $n \in \mathbb{N}^*$ cities & $m \in \mathbb{N}^*$ flight connections between them. You want to travel from Syrjälä to Lehmälä so that you visit each city exactly once. How many possible routes are there?*

Input. *The 1st input line has 2 integers $n, m \in \mathbb{N}^*$: the number of cities & flights. The cities are numbered $1, 2, \dots, n$. City 1 is Syrjälä, & city n is Lehmälä. Then, there are m lines describing the flights. Each line has 2 integers $a, b \in \mathbb{N}^*$: there is a flight from city a to city b . All flights are 1-way flights.*

Output. *Print 1 integer: the number of routes modulo $10^9 + 7$.*

Constraints. $n \in \overline{2, 20}, m \in [n^2], a, b \in [n]$.

Sample.

Hamiltonian_flight.inp	Hamiltonian_flight.out
4 6 1 2 1 3 2 3 3 2 2 4 3 4	2

Bài toán 2 (Chuyến bay Hamiltonian). *Có $n \in \mathbb{N}^*$ thành phố & $m \in \mathbb{N}^*$ chuyến bay nối liền chúng. Bạn muốn đi từ Syrjälä đến Lehmälä sao cho bạn ghé thăm mỗi thành phố đúng 1 lần. Có bao nhiêu tuyến đường khả thi?*

Input. *Dòng đầu vào thứ nhất chứa 2 số nguyên $n, m \in \mathbb{N}^*$: số lượng thành phố & chuyến bay. Các thành phố được đánh số $1, 2, \dots, n$. Thành phố 1 là Syrjälä, & thành phố n là Lehmälä. Sau đó, có m dòng mô tả các chuyến bay. Mỗi dòng chứa 2 số nguyên $a, b \in \mathbb{N}^*$: có 1 chuyến bay từ thành phố a đến thành phố b . Tất cả các chuyến bay đều là chuyến bay 1 chiều.*

Output. *In ra 1 số nguyên: số tuyến đường theo modulo $10^9 + 7$.*

Ràng buộc. $n \in \overline{2, 20}, m \in [n^2], a, b \in [n]$.

Solution. Let $dp[S][i]$ be the number of routes that visit all the cities in the subset $S \subset [n]$ & end at city i . The transitions will be

$$dp[S][i] = \sum_{x \in \text{adj}[i]} dp[S \setminus \{i\}][x] \text{ if } x \in S.$$

Time complexity: $O(2^n n^2)$.

– Giả sử $dp[S][i]$ là số tuyến đường đi qua tất cả các thành phố trong tập con $S \subset [n]$ & kết thúc tại thành phố i . Các phép chuyển tiếp sẽ là

$$dp[S][i] = \sum_{x \in \text{adj}[i]} dp[S \setminus \{i\}][x] \text{ nếu } x \in S.$$

Độ phức tạp thời gian: $O(2^n n^2)$.

C++: <https://usaco.guide/gold/dp-bitmasks?lang=cpp>.

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4 using ll = long long;
5 const int MAX_N = 20;
6 const ll MOD = (1ll)1e9 + 7;
7
8 ll dp[1 << MAX_N][MAX_N];
9 vector<int> come_from[MAX_N]; // come_from[i] contains the cities that can fly to i
10
11 int main() {
12     int city_num, flight_num;
13     cin >> city_num >> flight_num;
14     for (int f = 0; f < flight_num; ++f) {
15         int start, end;
16         cin >> start >> end;
17         come_from[--end].push_back(--start);
18     }
```

```

19
20     dp[1][0] = 1;
21     for (int s = 2; s < 1 << city_num; ++s) {
22         // only consider subsets that have the 1st city
23         if ((s & (1 << 0)) == 0) continue;
24         // also only consider subsets with the last city if it's the full subset
25         if ((s & (1 << (city_num - 1))) && s != ((1 << city_num) - 1)) continue;
26         for (int end = 0; end < city_num; ++end) {
27             if ((s & (1 << end)) == 0) continue;
28             // the subset that doesn't include the current end
29             int prev = s - (1 << end);
30             for (int j : come_from[end]) {
31                 if ((s & (1 << j))) {
32                     dp[s][end] += dp[prev][j];
33                     dp[s][end] %= MOD;
34                 }
35             }
36         }
37     }
38     cout << dp[(1 << city_num) - 1][city_num - 1] << '\n';
39 }

```

□

3.1 Merging subsets – Gộp tập con

Definition 1 (Proper/strict subset). Let A be a set. A subset B is called a proper subset (also strict subset) of A iff $B \subset A$ & $B \neq A$, denoted by $B \subsetneq A$ (or $B \subsetneqq A$).

Định nghĩa 1 (Tập con ngặt). Cho A là 1 tập hợp. Tập con B được gọi là tập con thực sự (cũng là tập con nghiêm ngặt) của A nếu và chỉ $B \subset A$ & $B \neq A$, ký hiệu là $B \subsetneq A$ (hoặc $B \subsetneqq A$).

Bài toán 3. Cho $n \in \mathbb{N}$ & 1 tập hợp $A = \{a_1, a_2, \dots, a_n\}$ có n phần tử. Đếm: (a) Số tập con của A . (b) Số tập con ngặt của A .

1st solution. (a) Vì mỗi phần tử a_i của tập A có thể thuộc hoặc không thuộc 1 tập con $B \subset A$, theo quy tắc nhân, số tập con của A bằng $\prod_{i=1}^n 2 = 2^n$. (b) Trong 2^n tập con của A , chỉ có duy nhất tập con A không phải là tập con ngặt, nên số tập con ngặt của A bằng $2^n - 1$. □

2nd solution. (a) Với mỗi $i \in \overline{0, n}$, có $C_n^i = \binom{n}{i}$ tập con của A có đúng i phần tử, nên tổng số tập con của A bằng $\sum_{i=0}^n \binom{n}{i} = (1+1)^n = 2^n$ (theo định lý nhị thức Newton). (b) Ta loại ra bản thân tập hợp A , với mỗi $i \in \overline{0, n-1}$, có $C_n^i = \binom{n}{i}$ tập con của A có đúng i phần tử, nên tổng số tập con của A bằng $\sum_{i=0}^{n-1} \binom{n}{i} = (1+1)^n - C_n^n = 2^n - 1$ (theo định lý nhị thức Newton). □

In some problems, for a set S , it is insufficient to transition from $S \setminus \{i\}$. Instead, it is necessary to transition from all strict subsets (also proper subset) of S . Though it may seem like we have to do $O(2^n 2^n) = O(4^n)$ transitions, there's really only $O(3^n)$ transitions. Indeed, we count the number of ordered pairs (T, S) where $T \subset S \subset [n]$. Instead of counting directly, notice that each element x is either: (i) in T & S , i.e., $x \in S \cap T$; (ii) in either, i.e., $x \notin S \cup T$; (iii) in S but not in T , i.e., $x \in S \setminus T$. If x is in T but not in S , i.e., $x \in T \setminus S$, T is not a valid subset. Given that each element can be in 3 possible states, our overall complexity is actually $O(3^n)$. To implement this, we can do some bitwise tricks:

– Trong 1 số bài toán, đối với tập S , việc chuyển đổi từ $S \setminus \{i\}$ là không đủ. Thay vào đó, cần phải chuyển đổi từ tất cả các tập con nghiêm ngặt của S . Mặc dù có vẻ như chúng ta phải thực hiện $O(2^n 2^n) = O(4^n)$ chuyển đổi, nhưng thực ra chỉ có $O(3^n)$ chuyển đổi. Thật vậy, chúng ta đếm số cặp có thứ tự (T, S) trong đó $T \subset S \subset [n]$. Thay vì đếm trực tiếp, hãy lưu ý rằng mỗi phần tử x là: (i) trong T & S , tức là $x \in S \cap T$; (ii) trong 1 trong hai trường hợp, tức là $x \notin S \cup T$; (iii) trong S nhưng không trong T , tức là $x \in S \setminus T$. Nếu x nằm trong T nhưng không nằm trong S , tức là $x \in T \setminus S$, thì T không phải là 1 tập con hợp lệ. Vì mỗi phần tử có thể ở 3 trạng thái, nên độ phức tạp tổng thể của chúng ta thực tế là $O(3^n)$. Để thực hiện điều này, chúng ta có thể thực hiện 1 số thủ thuật bitwise:

```

1  for (int mask = 0; mas < (1 << n); ++mask)
2      for (int submask = mask; submask != 0; submask = (submask - 1) & mask) {
3          int subset = mask ^ submask;
4          // do whatever you need to do here
5      }

```

When we subtract 1 from `submask`, the rightmost bit flips to a 0 & all bits to the right of it will become 1. Applying the bitwise AND with `mask` removes all extra bits not in `mask`. From this process, we can get all strict subsets in increasing order by calculating `mask ⊕ submask`, which does set subtraction $S \setminus T$.

– Khi chúng ta trừ 1 khỏi `submask`, bit ngoài cùng bên phải sẽ chuyển thành 0 & tất cả các bit bên phải nó sẽ trở thành 1. Áp dụng phép toán AND bitwise với `mask` sẽ loại bỏ tất cả các bit thừa không nằm trong `mask`. Từ quy trình này, chúng ta có

thể lấy tất cả các tập con nghiêm ngặt theo thứ tự tăng dần bằng cách tính toán $\text{mask} \oplus \text{submask}$, điều này sẽ thực hiện phép trừ tập hợp $S \setminus T$.

Bài toán 4. Cho $n \in \mathbb{N}$. Đếm số cặp (S, T) thỏa: (a) $T \subset S \subset [n]$. (b) $T \subsetneq S \subsetneq [n]$. (c) $T \subset S \subsetneq [n]$. (d) $T \subsetneq S \subset [n]$.

Solution. (a) Có $\binom{n}{i}$ tập con $S \subset [n]$ có đúng i phần tử, $\forall i \in \overline{0, n}$, tiếp theo với mỗi bộ i phần tử được chọn, có $\binom{i}{j}$ tập con $T \subset S$ có đúng j phần tử, $\forall j \in \overline{0, i}$, nên theo quy tắc cộng & quy tắc nhân, số cặp (S, T) thỏa $T \subset S \subset [n]$ bằng

$$\sum_{i=0}^n \sum_{j=0}^i \binom{n}{i} \binom{i}{j} = \sum_{i=0}^n \sum_{j=0}^i \frac{n!}{j!(i-j)!(n-i)!} = 3^n.$$

(b)

$$\sum_{i=1}^{n-1} \sum_{j=0}^{i-1} \binom{n}{i} \binom{i}{j} =$$

(c)

$$\sum_{i=0}^{n-1} \sum_{j=0}^i \binom{n}{i} \binom{i}{j} =$$

(d)

$$\sum_{i=1}^n \sum_{j=0}^{i-1} \binom{n}{i} \binom{i}{j} =$$

□

Problem 3 (AtCoder/close group). Given a simple undirected graph with $n \in \mathbb{N}^*$ vertices & $m \in \mathbb{N}$ edges. The vertices are numbered $1, 2, \dots, n$, & the i th edge connects vertices a_i & b_i . Find the minimum possible number of connected components in the graph after removing zero or more edges so that the following condition will be satisfied: For every pair of vertices (a, b) such that $1 \leq a < b \leq n$, if vertices a, b belong to the same connected component, there is an edge that directly connects vertices a & b .

Constraints. All values in input are integers. $n \in [18], 0 \leq m \leq \binom{n}{2} = \frac{n(n-1)}{2}, 1 \leq a_i < b_i \leq n, (a_i, b_i) \neq (a_j, b_j)$ for $i \neq j$.

Input. Input is given from Standard Input in the following format:

```
n m
a_1 b_1
...
a_m b_m
```

Output. Print the answer.

Sample.

close_group.inp	close_group.out
3 2 1 2 1 3	2
4 6 1 2 1 3 1 4 2 3 2 4 3 4	1
10 11 9 10 2 10 8 9 3 4 5 8 1 8 5 6 2 5 3 6 6 9 1 9	5
18 0	18

Explanation. For testcase 1, without removing edges, the pair (2,3) violates the condition. Removing 1 of the edges disconnects vertices 2 & 3 (i.e., after removing an edge, the graph has 2 connected components $\{1,2\}, \{3\}$ if removing the edge (1,3), or $\{1,3\}, \{2\}$ if removing the edge (1,2)), satisfying the condition.

Bài toán 5 (Nhóm đóng). Cho 1 đồ thị vô hướng đơn giản với $n \in \mathbb{N}^*$ đỉnh & $m \in \mathbb{N}$ cạnh. Các đỉnh được đánh số $1, 2, \dots, n$, & cạnh i nối các đỉnh a_i & b_i . Tìm số lượng thành phần liên thông nhỏ nhất có thể có trong đồ thị sau khi loại bỏ không hoặc nhiều cạnh sao cho điều kiện sau được thỏa mãn: Với mọi cặp đỉnh (a, b) such that $1 \leq a < b \leq n$, nếu các đỉnh a, b thuộc cùng 1 thành phần liên thông, thì tồn tại 1 cạnh nối trực tiếp các đỉnh a & b .

Ràng buộc. Tất cả các giá trị trong đầu vào đều là số nguyên. $n \in [18], 0 \leq m \leq \binom{n}{2} = \frac{n(n-1)}{2}, 1 \leq a_i < b_i \leq n, (a_i, b_i) \neq (a_j, b_j)$ với $i \neq j$.

Input. Đầu vào được cung cấp từ Đầu vào Chuẩn theo định dạng sau:

```
n m
a_1 b_1
...
a_m b_m
```

Output. In ra đáp án.

Solution. The goal of this problem is to partition the nodes into sets such that the nodes in each set form a complete graph. Let $dp[S]$ be the minimum number of partitions such that in each partition, the graph formed is a complete graph. We can 1st find which sets T form a complete graph, setting $dp[T] = 1$ & ∞ otherwise. This can be done naively in $O(n^2 2^n)$ or $O(n 2^n)$ by setting the adjacency list as a bitmask & using bit manipulations if a set of nodes is a complete graph. Then we can transition as follows:

$$dp[S] = \min_{T \subset S} dp[T] + dp[S \setminus T].$$

– Mục tiêu của bài toán này là phân hoạch các nút thành các tập hợp sao cho các nút trong mỗi tập hợp là 1 đồ thị hoàn chỉnh. Giả sử $dp[S]$ là số phân hoạch nhỏ nhất sao cho trong mỗi phân hoạch, đồ thị được tạo thành là 1 đồ thị hoàn chỉnh. Trước tiên, ta tìm các tập hợp T tạo thành 1 đồ thị hoàn chỉnh, đặt $dp[T] = 1$ & ∞ trong các trường hợp còn lại. Điều này có thể được thực hiện 1 cách đơn giản trong $O(n^2 2^n)$ hoặc $O(n 2^n)$ bằng cách đặt danh sách kề là 1 mặt nạ bit & sử dụng các thao tác bit nếu 1 tập hợp các nút là 1 đồ thị hoàn chỉnh. Sau đó, ta có thể chuyển đổi như sau:

$$dp[S] = \min_{T \subset S} dp[T] + dp[S \setminus T].$$

C++: <https://usaco.guide/gold/dp-bitmasks?lang=cpp>.

```
1 #include <iostream>
2 #include <stdint>
3 #include <vector>
4 using namespace std;
5
6 int main() {
7     int n, m;
8     cin >> n >> m;
9     vector<int> adj(n);
10    for (int i = 0; i < m; ++i) {
11        int u, v;
12        cin >> u >> v;
13        --u; --v; // 0-based indexing
14        // adjacency list represented as a bitmask
15        adj[u] |= (1 << v);
16        adj[v] |= (1 << u);
17    }
18    vector<int> dp(1 << n, INT32_MAX);
19    for (int mask = 0; mask < (1 << n); ++mask) {
20        bool connected = true;
21        for (int u = 0; u < n; ++u)
22            if (((mask >> u) & 1) != 0)
23                // check if u is connected to all other nodes in mask
24                if (((adj[u] | (1 << u)) & mask) != mask) {
25                    connected = false;
26                    break;
27                }
28    }
```



```

28     if (connected) dp[mask] = 1;
29 }
30 for (int mask = 0; mask < (1 << n); ++mask)
31     for (int submask = mask; submask; submask = (submask - 1) & mask) {
32         int subset = mask ^ submask;
33         // submask has everything in mask but not in subset
34         if (dp[subset] != INT32_MAX && dp[submask] != INT32_MAX)
35             dp[mask] = min(dp[mask], dp[subset] + dp[submask]);
36     }
37 cout << dp[(1 << n) - 1] << '\n';
38 }

```

□

4 Some Applications of Bitmask Dynamic Programming – Vài Ứng Dụng của Quy Hoạch Động Bitmask

In some number theory problems, it helps to represent each number with a bitmask of its prime divisors, e.g., the set $\{6, 10, 15\}$ can be represented by $\{0b011, 0b101, 0b110\}$ (in binary, where the $0b$ prefix means that the number is binary), where the bits correspond to divisibility by $[2, 3, 5]$. Then, here are some equivalent operations between masks & these integers:

1. Bitwise AND is gcd.
2. Bitwise OR is lcm.
3. Iterating over bits is iterating over prime divisors.
4. Iterating over submasks is iterating over divisors.

Choosing a set with $\text{gcd} = 1$ is equivalent to choosing a set of bitmasks that AND to 0. E.g., we can see that $\{6, 10\}$ has $\text{gcd}\{6, 10\} \neq 1$ since $0b011 \& 0b101 = 0b001 \neq 0$. On the other hand, $\{6, 10, 15\}$ has $\text{gcd}\{6, 10, 15\} = 1$ since $0b011 \& 0b101 \& 0b110 = 0b000 = 0$.

5 Miscellaneous