Bitmask Dynamic Programming – Quy Hoạch Động Bitmask

Nguyễn Quản 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: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/bitmask_DP/NQBH_bitmask_dynamic_programming.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/bitmask_DP/NQBH_bitmask_dynamic_programming.tex.

Mục tiêu của bài viết này là tổng hợp 1 số bài viết về quy hoạch động bitmask bằng tiếng Việt & 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 OLP & ICPC.

Mục lục

1 Preliminaries

1.1 Notation – Ký hiệu

- $\overline{m,n} := \{m, m+1, \dots, n-1, n\}$, $\forall m, n \in \mathbb{Z}$, $m \le n$. 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 $\overline{a,b} := \overline{\lceil a \rceil, \lfloor b \rfloor}$ có nghĩa như định nghĩa trước đó với $m := \lceil a \rceil, n := \lfloor b \rfloor \in \mathbb{Z}$; khi đó ký hiệu "for $i \in \overline{a,b}$" với $a, b \in \mathbb{R}$, $a \le b$ có nghĩa là "for $i = \lceil a \rceil, \lceil a \rceil + 1, \dots, \lfloor b \rfloor - 1, \lfloor b \rfloor$, i.e., chỉ số/biến chạy i chạy từ $\lceil a \rceil$ đến $\lfloor b \rfloor \in \mathbb{Z}$.
- $\lfloor x \rfloor, \{x\}$ 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.

*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.

- card(A): cardinality of a set A (finite or infinite) – lực lượng của 1 tập hợp A (hữu hạn hoặc vô hạn).
- $[n] := \{1, 2, \ldots, n\}$: the set of 1st $n \in \mathbb{N}^\star$ positive integers, which serves as 1 of prototypical example of a finite set with n elements – tập hợp $n \in \mathbb{N}^\star$ số nguyên dương đầu tiên, đóng vai trò là 1 trong ví dụ nguyên mẫu của 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.
- $A_n^k = \dfrac{n!}{(n-k)!}$: Chỉnh hợp chập k phần tử từ 1 tập hợp có n phần tử.
- $C_n^k = \binom{n}{k} = \dfrac{n!}{k!(n-k)!}$: 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.

2. Wikipedia/bitwise operation.

Problem 1 (CodeForces/take a guess)**.** *This is an interactive task:* William *has a certain sequence of integers* $a_1, \ldots, a_n$ *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 *&* k*. The 2nd line must contain* n *integers* $a_1, a_2, \ldots, a_n$ *with* $a_i \in \overline{0, 10^9}$*,* $\forall i \in [n]$*, the sequence* a*.*

Sample.

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

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

Bài toán 1 (Doán)**.** *Đây là 1 nhiệm vụ tương tác:* William *có 1 dãy số nguyên* $a_1, \ldots, a_n$ *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 1 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 1 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 *&* 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 1 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 1 số nguyên x*, kết quả của phép toán cho các số bạn đã chọn. Sau khi xuất 1 dòng, đừng quên xuất 1 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 1 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 *&* k*. Dòng thứ 2 phải chứa* n *số nguyên* $a_1, a_2, \ldots, a_n$ *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 1 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 1 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:

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

Chứng minh như sau. Nhớ lại 1 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. □

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.

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int ask(string s, int a, int b) {
    cout << s << ' ' << a << ' ' << b << '\n';
    int res;
    cin >> res;
    return res;
}

/** @return the sum of the elements at a and b (0-indexed) */
int sum(int a, int b) {
    int and_ = ask("and", ++a, ++b);
    int or_ = ask("or", a, b);
    int xor_ = ~and_ & or_; // a ^ b = ~(a & b) & (a | b)
    return 2 * and_ + xor_; // a + b = 2(a & b) + a ^ b
}

int main() {
    int n, k;
    cin >> n >> k;
    // acquire 1st 3 elements
    int a_plus_b = sum(0, 1);
    int a_plus_c = sum(0, 2);
    int b_plus_c = sum(1, 2);

    // get actual values by solving equations
    vector<int> arr{(a_plus_b + a_plus_c + b_plus_c) / 2};
    arr.push_back(a_plus_b - arr[0]);
    arr.push_back(a_plus_c - arr[0]);

    // get rest of array
    for (int i = 3; i < n; ++i) arr.push_back(sum(i - 1, i) - arr.back());
    sort(arr.begin(), arr.end());
    cout << "finish " << arr[k - 1] << '\n';
}
```

□

Problem 2. *Implement addition & multiplication using only bitwise operators.*

– *Thực hiện phép cộng & phép nhân chỉ sử dụng các toán tử bit.*

Solution. If we perform addition without carrying, then we are simply performing the `XOR ^` operator. Then, the bits that we carry over are those equivalent to 1 in both numbers: $a\&b$.

– Nếu chúng ta thực hiện phép cộng mà không nhớ, thì chúng ta chỉ đơn giản thực hiện phép toán `XOR ^`. Khi đó, các bit được nhớ là các bit tương đương với 1 trong cả hai số: $a\&b$.

```
int add(int a, int b) {
    while (b > 0) {
        int carry = a & b;
        a ^= b;
        b = carry << 1;
    }
    return a;
}
```

For simplicity, we will use the `sum` function defined above. If we divide up b into $\sum_{i=1}^{n} 2^{b_i} = 2^{b_1} + 2^{b_2} + \cdots + 2^{b_n}$, we get the following

$$a \times b = a \times \sum_{i=1}^{n} 2^{b_i} = a(2^{b_1} + 2^{b_2} + \cdots + 2^{b_n})a2^{b_1} + a2^{b_2} + \cdots + a2^{b_n} = \sum_{\text{bits in } b} a << b_i.$$

Implementation:

```
int prod(int a, int b) {
    int c = 0;
    while ( b > 0) {
        if ((b & 1) == 1) c = add(c, a);
        a <<= 1;
        b >>= 1;
    }
    return c;
}
```

This same idea is used in binary exponentiation. □

2.1 XOR operation – Thao tác XOR

Perhaps 1 of the most common binary operations in practice is bitwise `xor`. The special property that differentiates it from the other bitwise operations is that `xor` is its own inverse, i.e., $x \oplus x = 0$.

– Có lẽ 1 trong những phép toán nhị phân phổ biến nhất trong thực tế là phép toán bitwise `xor`. Tính chất đặc biệt phân biệt nó với các phép toán bitwise khác là `xor` là phép toán nghịch đảo của chính nó, tức là $x \oplus x = 0$.

Problem 3 (AtCoder/Toyota Programming Contest 2024#8/E: Xor Sigma Problem)**.** *You are given an integer sequence* $a = \{a_i\}_{i=1}^{n}$ *of length* $n \in \mathbb{N}^\star$*. Find the value of the following expression:*

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} a_i \oplus a_{i+1} \oplus \cdots \oplus a_j.$$

Constraints. $n \in \overline{2, 2 \cdot 10^5}, a_i \in [10^8], \forall i \in [n]$. *All input values are integers.*

Input. *The input is given from Standard Input in the following format:*

```
n
a_1 a_2 ... a_n
```

Output. *Print the answer.*

Sample.

xor_sigma.inp	xor_sigma.out
3 1 3 2	3
7 2 5 6 5 2 1 7	83

Explanation. *For testcase 1,* $a_1 \oplus a_2 = 2, a_1 \oplus a_2 \oplus a_3 = 0, a_2 \oplus a_3 = 1$*, so the answer is* $2 + 0 + 1 = 3$.

Bài toán 2. *Bạn được cho 1 dãy số nguyên $a = \{a_i\}_{i=1}^n$ có độ dài $n \in \mathbb{N}^\star$. Tìm giá trị của biểu thức sau:*

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} a_i \oplus a_{i+1} \oplus \cdots \oplus a_j.$$

Constraints. *$n \in \overline{2, 2 \cdot 10^5}, a_i \in [10^8], \forall i \in [n]$. Tất cả các giá trị đầu vào đều là số nguyên.*

Input. *Dữ liệu đầu vào được cung cấp từ Đầu vào Chuẩn theo định dạng sau:*

```
n
a_1 a_2 ... a_n
```

Output. *In ra đáp án.*

Solution. We analyze the bits of each xor-sum independently. In this case, we will check for every bit position, i.e., for every power of 2, whether or not it's set in the xor-sum of every contiguous subsequence. This way, we transformed the problem into counting the number of subsequences with nonzero xor-sum; this will be applied on the array form by the bits on the same position in all the values.

– Chúng ta phân tích các bit của mỗi tổng xor 1 cách độc lập. Trong trường hợp này, chúng ta sẽ kiểm tra xem vị trí bit, tức là với mọi lũy thừa của 2, có nằm trong tổng xor của mọi chuỗi con liền kề hay không. Bằng cách này, chúng ta đã chuyển đổi bài toán thành việc đếm số lượng chuỗi con có tổng xor khác 0; điều này sẽ được áp dụng trên dạng mảng bởi các bit ở cùng vị trí trong tất cả các giá trị.

Implementation: Time complexity: $O(n)$

```cpp
#include <iostream>
#include <numeric>
#include <vector>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> v(n);
    for (int &a : v) cin >> a;
    long long ans = -accumulate(v.begin(), v.end(), 0LL);
    // for every bit position, check if it is set on in the xor-sum of every subsequence
    for (int i = 0; i < 30; ++i) {
        int s = 0;
        // count the prefix sums
        // the # of 0 xor-sum prefixes starts from 1 to count the prefixes with xor-sum 1
        vector<int> pref = {1, 0};
        for (int a : v) {
            s ^= (a >> i) & 1;
            /*
            * Count the # of sequences ending at this position with xor-sum
            * non-zero by counting the prefixes of the inversed bit, i.e.
            * pref[i] ^ pref[j] = 1. Update the answer by adding the # of such
            * sequences multiplied by the respective power of two.
            */
            ans += pref[s ^ 1] * 1LL << i;
            ++pref[s]; // update prefixes
        }
    }
    cout << ans;
}
```

□

Some useful operations:

1. `x &= (1 << k)` will set the number to $1<<k$ if the kth bit was true, & 0 if the $k$bit was false.
   – `x &= (1 << k)` sẽ đặt số thành $1<<k$ nếu bit thứ k là đúng, & 0 nếu bit k là sai.
2. `x |= (1 << k)` will set the kth bit to true regardless of what it currently is.
   – `x |= (1 << k)` sẽ đặt bit thứ k thành true bất kể bit đó hiện tại là gì.

3. If the kth bit is already true, `x += (1 << k)` will make the kth bit false & mess up the rest of x.
   – Nếu bit thứ k đã đúng, `x += (1 << k)` sẽ làm cho bit thứ k sai & làm hỏng phần còn lại của x.
4. `1 << n` will give 2^n, which is the number of subsets for an array of size $n \in \mathbb{N}$ including $\emptyset$.
   – `1 << n` sẽ cho kết quả 2^n, là số tập hợp con của 1 mảng có kích thước $n \in \mathbb{N}$ bao gồm $\emptyset$.
5. `x & ( 1 << k)` will check if the kth bit in `int` x is true.
   – `x & ( 1 << k)` sẽ kiểm tra xem bit thứ k trong `int` x có đúng không.

3 Introduction to Dynamic Programming – Giới Thiệu Quy Hoạch Động

Resources – Tài nguyên.

1. Michael Cao, Benjamin Qi, Neo Wang, Daniel Zhu. USACO Guide/introduction to DP.
   Abstract. Speeding up naive recursive solutions with memoization.
2. Wikipedia/dynamic programming.

Dynamic Programming (DP) is an important algorithmic technique in Competitive Programming from the gold division to competitions like the International Olympiad of Informatics (abbr., IOI). By breaking down the full task into subproblem, DP avoids the redundant computations of brute force solutions. Although it is not too difficult to grasp the general ideas behind DP, the technique can be used in a diverse range of problems & is a must-know idea for competitors in the USACO Gold division.
– Lập trình Động (DP) là một kỹ thuật thuật toán quan trọng trong Lập trình Cạnh tranh, từ hạng Vàng đến các cuộc thi như Olympic Tin học Quốc tế (viết tắt là IOI). Bằng cách chia nhỏ toàn bộ bài toán thành các bài toán con, DP tránh được việc tính toán dư thừa của các giải pháp dùng vũ lực. Mặc dù không quá khó để nắm bắt các ý tưởng chung đằng sau DP, nhưng kỹ thuật này có thể được sử dụng trong nhiều bài toán khác nhau & là một kiến thức bắt buộc phải biết đối với các thí sinh ở hạng Vàng USACO.

Note 1. *"It is usually a good idea to write a slower solution 1st. E.g., if the complexity required for full points is $O(n)$ & you come up with a simple $O(n^2)$ solution, then you should definitely type that up 1st & earn some partial credit. Afterwards, you can rewrite parts of your slow solution until it is of the desired complexity. The slow solution might also serve as something to stress test against."*
– Thông thường, viết một giải pháp chậm hơn trước là một ý tưởng hay. Ví dụ, nếu độ phức tạp cần thiết cho điểm đầy đủ là $O(n)$ & bạn đưa ra một giải pháp đơn giản $O(n^2)$, thì bạn chắc chắn nên viết giải pháp đó trước & được cộng một phần điểm. Sau đó, bạn có thể viết lại các phần của giải pháp chậm cho đến khi đạt được độ phức tạp mong muốn. Giải pháp chậm này cũng có thể được dùng để kiểm tra ứng suất.

Problem 4 (AtCoder/educational DP contest/A: frog 1)**.** *There are n stones, numbered $1, 2, \ldots, n$. For each $i \in [n]$, the height of stone i is h_i. There is a frog who is initially on stone 1. He will repeat the following actions some number of times to reach stone n: If the frog is currently on stone i, jump to stone $i+1$ or stone $i+2$. Here, a cost of $|h_i - h_j|$ is incurred, where j is the stone to land on. Find the minimum possible total cost incurred before the frog reaches stone n.*

Constraints. *All values in input are integers. $n \in \overline{2, 10^5}, h_i \in [10^4], \forall i \in [n]$.*

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

```
n
h_1 h_2 ... h_n
```

Output. *Print the minimum possible total cost incurred.*

Sample.

frog_1.inp	frog_1.out
4 10 30 40 20	30
2 10 10	0
6 30 10 60 10 60 50	40

Explanation. *For testcase 1, if we follow the path $1 \to 2 \to 4$, the total cost incurred would be $|10 - 30| + |30 - 20| = 30$. For testcase 2, if we follow the path $1 \to 2$, the total cost incurred would be $|10 - 10| = 0$. For testcase 3, if we follow the path $1 \to 3 \to 5 \to 6$, the total cost incurred would be $|30 - 60| + |60 - 60| + |60 - 50| = 40$*

Bài toán 3 (Ếch 1)**.** *Có n viên đá, được đánh số $1, 2, \ldots, n$. Với mỗi $i \in [n]$, độ cao của Viên đá i là h_i. Có một con ếch ban đầu ở trên Viên đá 1. Nó sẽ lặp lại các hành động sau một số lần để đến được Viên đá n: Nếu con ếch hiện đang ở trên Viên đá i, hãy nhảy đến Viên đá $i+1$ hoặc Viên đá $i+2$. Ở đây, chi phí là $|h_i - h_j|$, trong đó j là viên đá để đáp xuống. Tìm tổng chi phí nhỏ nhất có thể phát sinh trước khi con ếch đến được Viên đá n.*

Ràng buộc. *Tất cả các giá trị trong đầu vào là số nguyên. $n \in \overline{2, 10^5}, h_i \in [10^4], \forall i \in [n]$.*

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

```
n
h_1 h_2 ... h_n
```

Đầu ra. *In ra tổng chi phí phát sinh nhỏ nhất có thể.*

The problem asks us to compute the minimum total cost it takes for a frog to travel from stone 1 to stone $n \leq 10^5$ given that the frog can only jump a distance of 1 or 2. The cost to travel between any 2 stones i, j is given by $c(i, j) \coloneqq |h_i - h_j|$, where h_i represents the height of stone i.

– Bài toán yêu cầu chúng ta tính tổng chi phí tối thiểu để một con ếch di chuyển từ hòn đá 1 đến hòn đá $n \leq 10^5$ biết rằng con ếch chỉ có thể nhảy được quãng đường là 1 hoặc 2. Chi phí để di chuyển giữa 2 hòn đá i, j bất kỳ được xác định bởi $c(i, j) \coloneqq |h_i - h_j|$, trong đó h_i biểu thị chiều cao của hòn đá i.

1st solution: Without DP. Time complexity: $O(2^n)$. Since there are only 2 options, we can use recursion to compute what would happen if we jumped either 1 stone or 2 stones. There are 2 possibilities, so recursively computing would require computing both a left & right subtree. Therefore, for every additional jump, each branch splits into 2, which results in an exponential time complexity.

– Vì chỉ có 2 lựa chọn, chúng ta có thể sử dụng đệ quy để tính toán điều gì sẽ xảy ra nếu chúng ta nhảy 1 hoặc 2 viên đá. Có 2 khả năng, vì vậy tính toán đệ quy sẽ yêu cầu tính toán cả cây con trái & cây con phải. Do đó, với mỗi lần nhảy bổ sung, mỗi nhánh sẽ tách thành 2, dẫn đến độ phức tạp thời gian theo cấp số nhân.

However, this can be sped up with dynamic programming by keeping track of "optimal states" in order to avoid calculating states multiple times. E.g., recursively calculating jumps of length $1, 2, 1$ & $2, 1, 2$ reuses the state of stone 3. Dynamic programming provides the mechanism to cache such states.

– Tuy nhiên, điều này có thể được tăng tốc bằng lập trình động bằng cách theo dõi "trạng thái tối ưu" để tránh phải tính toán trạng thái nhiều lần. Ví dụ: tính toán đệ quy các bước nhảy có độ dài $1, 2, 1$ & $2, 1, 2$ sẽ tái sử dụng trạng thái của viên đá 3. Lập trình động cung cấp cơ chế lưu trữ đệm các trạng thái như vậy. □

2nd solution: With DP. Time complexity: $O(n)$. There are 2 main DP approaches:

1. *Push DP*, where we update future states based on the current state;

2. *Pull DP*, where we calculate the current state based on past states.

– Có 2 phương pháp DP chính:

1. *Push DP*, trong đó chúng ta cập nhật các trạng thái tương lai dựa trên trạng thái hiện tại;

2. *Pull DP*, trong đó chúng ta tính toán trạng thái hiện tại dựa trên các trạng thái trước đó.

(a) **Push DP.** There are only 2 options: jumping once, or jumping twice. Define $\text{dp}[i]$ as the minimum cost to reach stone i. Then, our transitions are as follows:

- Jump 1 stone, incurring a cost of $|h_i - h_{i+1}|$:

$$\text{dp}[i+1] = \min\{\text{dp}[i+1], \text{dp}[i] + |h_i - h_{i+1}|\}.$$

- Jump 2 stones, incurring a cost of $|h_i - h_{i+2}|$:

$$\text{dp}[i+2] = \min\{\text{dp}[i+2], \text{dp}[i] + |h_i - h_{i+2}|\}.$$

We can start with the base case that $\text{dp}[0] = 0$, since the frog is already on that square, & proceed to calculate $\text{dp}[1], \text{dp}[2], \ldots, \text{dp}[n-1]$.

– **DP đẩy.** Chỉ có 2 lựa chọn: nhảy một lần hoặc nhảy hai lần. Xác định $\text{dp}[i]$ là chi phí tối thiểu để đến được hòn đá i. Sau đó, các bước chuyển tiếp của chúng ta như sau:

- Nhảy 1 hòn đá, với chi phí là $|h_i - h_{i+1}|$:

$$\text{dp}[i+1] = \min\{\text{dp}[i+1], \text{dp}[i] + |h_i - h_{i+1}|\}.$$

- Nhảy 2 viên đá, tốn $|h_i - h_{i+2}|$:

$$\text{dp}[i+2] = \min\{\text{dp}[i+2], \text{dp}[i] + |h_i - h_{i+2}|\}.$$

Ta có thể bắt đầu với trường hợp cơ sở là $\text{dp}[0] = 0$, vì con ếch đã ở trên ô đó, & tiến hành tính $\text{dp}[1], \text{dp}[2], \ldots, \text{dp}[n-1]$.

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> height(n);
    for (int i = 0; i < n; ++i) cin >> height[i];
    // dp[i] is the minimum cost to get to the i-th stone
    vector<int> dp(n, INT_MAX);
    dp[0] = 0; // base case since we are already at the 1st stone
    for (int i = 0; i < n - 1; ++i) { // for each state, calculate the states it leads to
        dp[i + 1] = min(dp[i + 1], dp[i] + abs(height[i] - height[i + 1])); // jump 1 stone
        if (i + 2 < n) dp[i + 2] = min(dp[i + 2], dp[i] + abs(height[i] - height[i + 2])); // jump 2 stones
    }
    cout << dp[n - 1];
}
```

Pull DP. There are 2 ways to get to stone i: from stone $i-1$ & stone $i-2$:

- Jump from stone $i-1$, incurring a cost of $|h_i - h_{i-1}|$:

$$\text{dp}[i] = \min\{\text{dp}[i], \text{dp}[i-1] + |h_i - h_{i-1}|\}.$$

- Jump from stone $i-2$, incurring a cost of $|h_i - h_{i-2}|$:

$$\text{dp}[i] = \min\{\text{dp}[i], \text{dp}[i-2] + |h_i - h_{i-2}|\}.$$

We can start with the base case that $\text{dp}[0] = 0$, since the frog is already on that square, & proceed to calculate $\text{dp}[1], \text{dp}[2], \ldots, \text{dp}[n-1]$.

– **DP kéo.** Có 2 cách để đến hòn đá i: từ hòn đá $i-1$ & hòn đá $i-2$:

- Nhảy từ hòn đá $i-1$, tốn $|h_i - h_{i-1}|$:

$$\text{dp}[i] = \min\{\text{dp}[i], \text{dp}[i-1] + |h_i - h_{i-1}|\}.$$

- Nhảy từ hòn đá $i-2$, chịu chi phí $|h_i - h_{i-2}|$:

$$\text{dp}[i] = \min\{\text{dp}[i], \text{dp}[i-2] + |h_i - h_{i-2}|\}.$$

Ta có thể bắt đầu với trường hợp cơ sở là $\text{dp}[0] = 0$, vì con ếch đã ở trên ô vuông đó, & tiến hành tính $\text{dp}[1], \text{dp}[2], \ldots, \text{dp}[n-1]$.

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> height(n);
    for (int i = 0; i < n; ++i) cin >> height[i];
    // dp[i] is the minimum cost to get to the i-th stone
    vector<int> dp(n, INT_MAX);
    dp[0] = 0; // base case since we are already at the 1st stone
    for (int i = 1; i < n; ++i) { // for each state, calculate the states it leads to
        if (i >= 1) dp[i] = min(dp[i], dp[i - 1] + abs(height[i] - height[i - 1])); // jump 1 stone
        if (i >= 2) dp[i] = min(dp[i], dp[i - 2] + abs(height[i] - height[i - 2])); // jump 2 stones
    }
    cout << dp[n - 1];
}
```

□

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

Resources – Tài nguyên.

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 5 (CSES Problem Set/Hamiltonian flights)**.** *There are $n \in \mathbb{N}^\star$ cities & $m \in \mathbb{N}^\star$ 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 $n, m \in \mathbb{N}^\star$: the number of cities & flights. The cities are numbered $1, 2, \ldots, n$. City 1 is Syrjälä, & city n is Lehmälä. Then, there are m lines describing the flights. Each line has 2 integers $a, b \in \mathbb{N}^\star$: 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 4 (Chuyến bay Hamiltonian)**.** *Có $n \in \mathbb{N}^\star$ thành phố & $m \in \mathbb{N}^\star$ 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 $n, m \in \mathbb{N}^\star$: số lượng thành phố & chuyến bay. Các thành phố được đánh số $1, 2, \ldots, n$. 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 $a, b \in \mathbb{N}^\star$: 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 $\mathrm{dp}[S][i]$ be the number of routes that visit all the cities in the subset $S \subset [n]$ & end at city i. The transitions will be

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

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

– Giả sử $\mathrm{dp}[S][i]$ 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

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

Độ phức tạp thời gian: $O(2^n n^2)$.

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

```
#include <iostream>
#include <vector>
using namespace std;
using ll = long long;
const int MAX_N = 20;
const ll MOD = (ll)1e9 + 7;

ll dp[1 << MAX_N][MAX_N];
vector<int> come_from[MAX_N]; // come_from[i] contains the cities that can fly to i

int main() {
    int city_num, flight_num;
    cin >> city_num >> flight_num;
```

```
    for (int f = 0; f < flight_num; ++f) {
        int start, end;
        cin >> start >> end;
        come_from[--end].push_back(--start);
    }

    dp[1][0] = 1;
    for (int s = 2; s < 1 << city_num; ++s) {
        // only consider subsets that have the 1st city
        if ((s & (1 << 0)) == 0) continue;
        // also only consider subsets with the last city if it's the full subset
        if ((s & (1 << (city_num - 1))) && s != ((1 << city_num) - 1)) continue;
        for (int end = 0; end < city_num; ++end) {
            if ((s & (1 << end)) == 0) continue;
            // the subset that doesn't include the current end
            int prev = s - (1 << end);
            for (int j : come_from[end]) {
                if ((s & (1 << j))) {
                    dp[s][end] += dp[prev][j];
                    dp[s][end] %= MOD;
                }
            }
        }
    }
    cout << dp[(1 << city_num) - 1][city_num - 1] << '\n';
}
```

□

4.1 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 5. *Cho $n \in \mathbb{N}$ & 1 tập hợp $A = \{a_1, a_2, \ldots, a_n\}$ 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 $S\backslash\{i\}$. 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., $x \in S\backslash T$. If x is in T but not in S, i.e., $x \in T\backslash S$, 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 $S\backslash\{i\}$ 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 $x \in S\backslash T$. Nếu x nằm trong T nhưng không nằm trong S, tức là $x \in T\backslash S$, 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:

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

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` $\oplus$ `submask`, which does set subtraction $S\backslash T$.

– 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 `mask` $\oplus$ `submask`, điều này sẽ thực hiện phép trừ tập hợp $S\backslash T$.

Bài toán 6. *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

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

(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 6 (AtCoder/close group)**.** *Given a simple undirected graph with $n \in \mathbb{N}^\star$ vertices & $m \in \mathbb{N}$ edges. The vertices are numbered $1, 2, \ldots, n$, & the ith 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 $1 \le a < b \le n$, 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. $n \in [18], 0 \le m \le \binom{n}{2} = \frac{n(n-1)}{2}, 1 \le a_i < b_i \le n, (a_i, b_i) \ne (a_j, b_j)$ for $i \ne j$.*

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 7 (Nhóm đóng)**.** *Cho 1 đồ thị vô hướng đơn giản với* $n \in \mathbb{N}^\star$ *đỉnh &* $m \in \mathbb{N}$ *cạnh. Các đỉnh được đánh số* $1, 2, \ldots, n$*, & 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.* $n \in [18], 0 \leq m \leq \binom{n}{2} = \dfrac{n(n-1)}{2}, 1 \leq a_i < b_i \leq n, (a_i, b_i) \neq (a_j, b_j)$ *với* $i \neq j$*.*

Input. *Dầu vào được cung cấp từ Dầu 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 from a complete graph. Let $\mathrm{dp}[S]$ 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 $\mathrm{dp}[T] = 1$ & ∞ otherwise. This can be done naively in $O(n^2 2^n)$ or $O(n2^n)$ 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:

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

– 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 $\mathrm{dp}[S]$ 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 $\mathrm{dp}[T] = 1$ & ∞ trong các trường hợp còn lại. Điều này có thể được thực hiện 1 cách đơn giản trong $O(n^2 2^n)$ hoặc $O(n2^n)$ 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:

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

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

```
1 #include <iostream>
2 #include <cstdint>
```

```cpp
#include <vector>
using namespace std;

int main() {
    int n, m;
    cin >> n >> m;
    vector<int> adj(n);
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        --u; --v; // 0-based indexing
        // adjacency list represented as a bitmask
        adj[u] |= (1 << v);
        adj[v] |= (1 << u);
    }
    vector<int> dp(1 << n, INT32_MAX);
    for (int mask = 0; mask < (1 << n); ++mask) {
        bool connected = true;
        for (int u = 0; u < n; ++u)
            if (((mask >> u) & 1) != 0)
                // check if u is connected to all other nodes in mask
                if (((adj[u] | (1 << u)) & mask) != mask) {
                    connected = false;
                    break;
                }
        if (connected) dp[mask] = 1;
    }
    for (int mask = 0; mask < (1 << n); ++mask)
        for (int submask = mask; submask; submask = (submask - 1) & mask) {
            int subset = mask ^ submask;
            // submask has everything in mask but not in subset
            if (dp[subset] != INT32_MAX && dp[submask] != INT32_MAX)
                dp[mask] = min(dp[mask], dp[subset] + dp[submask]);
        }
    cout << dp[(1 << n) - 1] << '\n';
}
```

□

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

– Trong 1 số bài toán lý thuyết số, việc biểu diễn mỗi số bằng 1 mặt nạ bit của các ước số nguyên tố của nó sẽ hữu ích. Ví dụ, tập hợp $\{6, 10, 15\}$ có thể được biểu diễn bằng $\{0b011, 0b101, 0b110\}$ (ở dạng nhị phân, tiền tố $0b$ có nghĩa là số ở dạng nhị phân), trong đó các bit tương ứng với khả năng chia hết cho $[2, 3, 5]$. Sau đó, đây là 1 số phép toán tương đương giữa các mặt nạ & các số nguyên này:

1. Bitwise `AND` là gcd.
2. Bitwise `OR` là lcm.
3. Lặp qua các bit là lặp qua các ước số nguyên tố.

4. Lặp qua các mặt nạ con là lặp qua các ước số.

Việc chọn 1 tập hợp có UCLN = 1 tương đương với việc chọn 1 tập hợp các mặt nạ bit `AND` bằng 0. Ví dụ, ta thấy $\{6, 10\}$ có UCLN$\{6, 10\} \neq 1$ vì `0b011&0b101 = 0b001` $\neq 0$. Mặt khác, $\{6, 10, 15\}$ có UCLN$\{6, 10, 15\} = 1$ vì `0b011&0b101&0b110 = 0b000 = 0`.

6 Miscellaneous