Lecture Note: Combinatorics & Graph Theory
Bài Giảng: Tổ Hợp & Lý Thuyết Đồ Thị

Nguyễn Quản Bá Hồng[1]

Ngày 2 tháng 6 năm 2025

[1]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.

Mục lục

Preface

Abstract

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:
URL: https://nqbh.github.io/advanced_STEM/.
Latest version:

- *Lecture Note: Combinatorics & Graph Theory – Bài Giảng: Tổ Hợp & Lý Thuyết Đồ Thị.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/lecture/NQBH_combinatorics_graph_theory_lecture.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/lecture/NQBH_combinatorics_graph_theory_lecture.tex.
- *Slide: Combinatorics & Graph Theory – Slide Bài Giảng: Tổ Hợp & Lý Thuyết Đồ Thị.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/slide/NQBH_combinatorics_graph_theory_slide.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/slide/NQBH_combinatorics_graph_theory_slide.tex.
- *Survey: Combinatorics & Graph Theory – Khảo Sát: Tổ Hợp & Lý Thuyết Đồ Thị.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/NQBH_combinatorics.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/NQBH_combinatorics.tex.
- Codes:
  - C/C++: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/C++.
  - Pascal: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/Pascal.
  - Python: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/Python.

Tài liệu này là bài giảng tôi dạy cho sinh viên Khoa Công Nghệ (undegraduate Computer Science students) chuyên ngành Kỹ Thuật Phần Mềm (Software Engineering, abbr., SE) & Trí Tuệ Nhân Tạo–Khoa Học Dữ Liệu (Artificial Intelligence–Data Science, abbr., AIDS) nên sẽ tập trung vào phương diện lập trình cho các khái niệm Tổ hợp & Lý thuyết đồ thị được nghiên cứu. Bài giảng này gồm 2 phần chính:

- **Part I: Combinatorics – Tổ Hợp**.
- **Part II: Graph Theory – Lý Thuyết Đồ Thị**. Tập trung vào các thuật toán trên cây (algorithms on trees) & thuật toán trên đồ thị (algorithms on graphs)

Preliminaries

Notation – Ký hiệu

- $\overline{m,n} := \{m, m+1, \ldots, n-1, n\}$, $\forall m, n \in \mathbb{Z}$, $m \leq n$. Hence the notation "for $i \in \overline{m,n}$" means "for $i = m, m+1, \ldots, n$", 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 $a, b \in \mathbb{R}$, $a \leq b$ có nghĩa là "for $i = \lceil a \rceil, \lceil a \rceil + 1, \ldots, \lfloor b \rfloor - 1, \lfloor b \rfloor$, 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.

- $x_+ \coloneqq \max\{x, 0\}$, $x_- \coloneqq \max\{-x, 0\} = -\min\{x, 0\}$ 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).
- $[n] \coloneqq \{1, 2, \ldots, n\}$: 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.

Phần I

Combinatorics – Tổ Hợp

Chương 1

Advanced Counting Techniques & Algorithms – Các Phép Đếm Nâng Cao & Thuật Toán

Contents

1.1 Basic Combinatorics – Tổ Hợp Cơ Bản

Combinatorics is a collection of techniques & a language for the study of (finite or countably infinite) discrete structures. Given a set of elements (& possibly some structure on that set), typical questions in combinatorics are:

- Does a specific arrangement of the elements exist?
- How many such arrangements are there?
- What properties do these arrangements have?
- Which 1 of the arrangements is maximal, minimal, or optimal according to some criterion?

– Tổ hợp là 1 tập hợp các kỹ thuật & 1 ngôn ngữ để nghiên cứu các cấu trúc rời rạc (hữu hạn hoặc vô hạn đếm được). Cho 1 tập hợp các phần tử (& có thể có 1 số cấu trúc trên tập hợp đó), các câu hỏi điển hình trong tổ hợp là:

- Có tồn tại sự sắp xếp cụ thể của các yếu tố không?
- Có bao nhiêu sự sắp xếp như vậy?
- Những sự sắp xếp này có những tính chất gì?
- Sự sắp xếp nào là tối đa, tối thiểu hoặc tối ưu theo 1 số tiêu chí?

1.2 Set theory

Resources – Tài nguyên.

1. [Hal60; Hal74]. Paul Richard Halmos, *Naive Set Theory*.

   Comments. 1 trong những quyển sách có nội dung đẹp nhất về Lý Thuyết Tập Hợp Ngây Thơ (distinguish Naive Set Theory vs. Axiomatic Set Theory – Lý Thuyết Tập Hợp Tiên Đề).

2. [Kap72; Kap77]. Irving Kaplansky. *Set Theory & Metric Spaces*.

Tập hợp là 1 khái niệm cơ bản của Toán học, thường không định nghĩa trong naive set theory (nhưng có thể trong axiomatic set theory). Thông thường, người ta dùng khái niệm tập hợp để chỉ 1 nhóm các đối tượng đã được chọn ra, hay đã được quy định từ trước, & thường dùng ký hiệu tập hợp bằng chữ in hay chữ viết hoa A, B, C, X, Y, Z, tuy nhiên, nên viết ký hiệu bao hàm ý nghĩa của tập hợp để tiện theo dõi trong nghiên cứu, e.g., $\mathbb{N}_{\text{odd}} = \{n \in \mathbb{N}; n \not\vdots 2\}$: tập hợp các số tự nhiên lẻ, $\mathbb{N}_{\text{even}} = \{n \in \mathbb{N}; n \vdots 2\}$: tập hợp các số tự nhiên chẵn. Cho tập A, 1 đối tượng x được nói đến trong A được gọi là 1 *phần tử* của A, ký hiệu $x \in A$, nếu x không nằm trong A, nói x không thuộc A, ký hiệu $x \notin A$.

- *Quy ước.* Tập rỗng $\emptyset$ là tập hợp không gồm phần tử nào cả. Tập $\emptyset$ là duy nhất (uniqueness of emptyset: *the* emptyset).
- *Do độ lớn của tập hợp.* Khi tập A có hữu hạn phần tử thì số phần tử của A ký hiệu là $|A|$ hoặc $\#A$ hoặc $\operatorname{card} A$ (cardinal – lực lượng).
- *2 cách đặt/xác định tập hợp.*
  - Liệt kê các phần tử của tập hợp.
  - Chỉ ra tính chất đặc trưng của các phần tử của tập hợp.

Remark 1 (Tính phân biệt của các phần tử trong cùng 1 tập hợp)**.** *Tập hợp gồm các phần tử phân biệt nhau, i.e., nếu* $A = \{a_1, \dots, a_n\} \Rightarrow a_i \neq a_j,\ \forall i, j = 1, \dots, n,\ i \neq j$.

Định nghĩa 1 (Subset – Tập con)**.** *Cho 2 tập* A, B. $A \subset B \Leftrightarrow (\forall x \in A \Rightarrow x \in B)$*: A là* tập con *của B, hay B là* tập mẹ *của A, ký hiệu* $B \supset A$.

Định nghĩa 2 (2 tập hợp bằng nhau)**.** *(i) 2 tập hợp bằng nhau:* $A = B \Leftrightarrow (A \subset B \wedge B \subset A)$. *(ii) Cho* $n \in \mathbb{N}$, $n \geq 2$, *n tập hợp bằng nhau:* $A_1 = A_2 = \cdots = A_n \Leftrightarrow A_i = A_j,\ \forall i, j = 1, \dots, n \Leftrightarrow (A_i \subset A_j \wedge A_j \subset A_i),\ \forall i, j = 1, \dots, n$.

Theorem 1 (Some basic properties of sets & operations on sets – Vài tính chất cơ bản của tập hợp & các phép toán trên tập hợp)**.** *Let* $n \in \mathbb{N}^\star$, $A, B, C, \dots, A_i$ *be sets,* $\forall i = 1, \dots, n$.

(i) (Tính chất của tập con) $A \subset B \wedge B \subset C \Rightarrow A \subset C$. $\emptyset \subset A \subset A,\ \forall$ *set* A.

(ii) (Reflective) $A = A,\ \forall$ *set* A. *(Symmetric)* $A = B \Rightarrow B = A$. *(Transitive)* $A = B \wedge B = C \Rightarrow A = C$. $(|A| = |B| \wedge A \subset B) \Rightarrow A = B$.

Định nghĩa 3 (Giao của 2 tập hợp)**.** Giao *của 2 tập hợp* A, B *được cho bởi* $A \cap B = \{x; x \in A \wedge x \in B\}$.

Định nghĩa 4 (Hợp của 2 tập hợp)**.** Hợp *của 2 tập hợp* A, B *được cho bởi* $A \cup B = \{x; x \in A \vee x \in B\}$.

Định nghĩa 5 (Hiệu của 2 tập hợp)**.** Hiệu *của 2 tập hợp* A, B *được cho bởi* $A \backslash B = \{x; x \in A \wedge x \notin B\}$.

Định nghĩa 6 (Phần bù của 2 tập hợp)**.** *Cho* $A \subset B$. Phần bù *của A trong B là tập* $\overline{A} = B \backslash A$ *hay* $C_B^A = B \backslash A$.

Định nghĩa 7 (Tích Descartes, [Phư10], p. 12)**.** *Cho* $n \in \mathbb{N}^\star$ *tập hợp* $A_1, \dots, A_n$. *Xét tập hợp A gồm tất cả các bộ n phần tử sắp thứ tự* $(a_1, \dots, a_n)$ *trong đó* $a_i \in A_i,\ \forall i = 1, \dots, n$. *Tập A được gọi là* tích Descartes *của n tập hợp* $A_1, \dots, A_n$ *& ký hiệu:*

$$A = \prod_{i=1}^{n} A_i = A_1 \times A_2 \times \cdots \times A_n = \{(a_1, \dots, a_n); a_i \in A_i,\ \forall i = 1, \dots, n\}.$$

Định lý 1 (Tính chất của tích Descartes). *(i)* $A \times B \neq B \times A$, $\forall A \neq B$. *(ii)* $|\prod_{i=1}^{n} A_i| = \prod_{i=1}^{n} |A_i|$.

A *multiset* is like a set except that its members need not be distinct.
– 1 *đa tập hợp* giống như 1 tập hợp ngoại trừ các thành viên của nó không cần phải khác biệt.

Definition 1 (Multisets, [Sha22], Def. 2.13, p. 53). *A* multiset *is a set together with a function that assigns a positive integer or* ∞ *– called a* repetition number *(or a* multiplicity*) – to each member of the set.*

Định nghĩa 8 (Đa tập). *Một* đa tập *là 1 tập hợp cùng với 1 hàm gán 1 số nguyên dương hoặc* ∞ *– được gọi là* số lặp lại *(hoặc* bội số*) – cho mỗi phần tử của tập hợp.*

Example 1 ([Sha22], Ex. 2.14, p. 53). $A = \{a, a, a, b, c, c, d, d\} = \{3 \cdot a, b, 2 \cdot c, 2 \cdot d\}$ *is a multiset with members* a, b, c, d *& with repetitions numbers* $3, 1, 2, 2$*, resp.*

Remark 2 ([Sha22], Rmk. 2.15, p. 53). *A set can be thought of as a multiset with all repetitionn numbers equal to 1. In an infinite multiset, some members could have infinite repetition numbers. I.e., we allow for the possibility that a multiset contains an infinite number of copies of a certain element.*
– Một tập hợp có thể được coi là 1 đa tập hợp với tất cả các số lặp lại bằng 1. Trong 1 đa tập hợp vô hạn, 1 số phần tử có thể có số lặp lại vô hạn. Tức là, chúng ta cho phép khả năng 1 đa tập hợp chứa 1 số lượng vô hạn các bản sao của 1 phần tử nhất định.

Definition 2 ($|\cdot|$ notation). *If* X *is a finite set, then* $|X|$ *will denote the number of elements in* X*. If* X *is a finite multiset, then* $|X|$ *will denote the sum of its repetition numbers.*

Định nghĩa 9 (Ký hiệu $|\cdot|$). *Nếu* X *là 1 tập hợp hữu hạn, thì* $|X|$ *sẽ biểu thị số phần tử trong* X*. Nếu* X *là 1 đa tập hữu hạn, thì* $|X|$ *sẽ biểu thị tổng số lần lặp lại của nó.*

Bài toán 1 ([Phư10], VD3, p. 16). *Cho* $a, b \in \mathbb{N}^\star$ *thỏa* $a + b$ *lẻ. Chia* $\mathbb{N}^\star$ *thành 2 tập rời nhau* A, B *(được gọi là* phân hoạch*). Chứng minh luôn tồn tại 2 phần tử* x, y *thuộc cùng 1 tập thỏa* $|x - y| \in \{a, b\}$.

Bài toán 2 ([Phư10], VD4, p. 17, [MOSP1997). *Phân hoạch* $\mathbb{N}^\star$ *thành* A, B*. Chứng minh* $\forall n \in \mathbb{N}^\star$*, tồn tại* $a, b \in \mathbb{N}^\star$*,* $a \neq b$*,* $a > n$*,* $b > n$ *thỏa* $\{a, b, a + b\} \subset A$ *hoặc* $\{a, b, a + b\} \subset B$.

Bài toán 3 ([Phư10], VD5, p. 19). *Cho* $A \subset [15]$ *thỏa tích của 3 phần tử khác nhau bất kỳ của* A *đều không phải là số chính phương. (a) Chỉ ra 1 tập* A *gồm* 10 *phần tử. (b) Xác định số phần tử lớn nhất của* M.

Bài toán 4 ([Phư10], 3., p. 20). *Cho các tập* $A_1, \ldots, A_n$ *thỏa* $A_i \neq A_j$*,* $\forall i, j = 1, \ldots, n$*,* $i \neq j$*. Chứng minh có ít nhất 1 tập hợp* A_i *không chứa tập nào trong các tập còn lại.*

Bài toán 5 ([Phư10], 4., p. 20). *(a) Cho tập* $A \subset \mathbb{R}$ *thỏa đồng thời: (i)* $\mathbb{Z} \subset A$*. (ii)* $\sqrt{2} + \sqrt{3} \in A$*. (iii)* $x + y \in S, xy \in S$*,* $\forall x, y \in S$*. Chứng minh* $\dfrac{1}{\sqrt{2} + \sqrt{3}} \in A$*. (b) Mở rộng giả thiết (ii) thành* $\sqrt{n} + \sqrt{n+1}$*,* $\forall n \in \mathbb{N}^\star$*. Chứng minh* $\dfrac{1}{\sqrt{n} + \sqrt{n+1}} \in A$*. (c) Mở rộng giả thiết (ii) thành* $\sqrt{a} + \sqrt{b}$*,* $\forall a, b \in \mathbb{N}^\star$*,* $a \neq b$*. Chứng minh* $\dfrac{(a-b)^2}{\sqrt{a} + \sqrt{b}} \in A$*. (d) Mở rộng giả thiết (ii) thành* $\sqrt[3]{a} + \sqrt[3]{b}$*,* $\forall a, b \in \mathbb{Z}$*,* $a \neq b$*. (e) Mở rộng giả thiết (ii) thành* $\sqrt[n]{a} + \sqrt[n]{b}$*,* $\forall a, b \in \mathbb{Z}$*,* $a \neq b$*, với* $n \in \mathbb{N}$*,* $n \geq 2$ *cho trước.*

Chứng minh. (a) $\sqrt{2} + \sqrt{3} \in A \xrightarrow{\text{(iii)}} (\sqrt{2} + \sqrt{3})^2 = 5 + 2\sqrt{6} \in A \xrightarrow[-5 \in \mathbb{Z} \subset A]{\text{(iii)}} 2\sqrt{6} \in A \xrightarrow{\text{(iii)}} 2\sqrt{6}(\sqrt{2} + \sqrt{3}) = 4\sqrt{3} + 6\sqrt{2} \in A \Rightarrow \dfrac{1}{\sqrt{2} + \sqrt{3}} = \sqrt{3} - \sqrt{2} = 5(\sqrt{2} + \sqrt{3}) - (4\sqrt{3} + 6\sqrt{2}) \in A$.

(b) $\sqrt{n} + \sqrt{n+1} \in A \xrightarrow{\text{(iii)}} (\sqrt{n} + \sqrt{n+1})^2 = 2n + 1 + 2\sqrt{n(n+1)} \in A \xrightarrow[-(2n+1) \in \mathbb{Z} \subset A]{\text{(iii)}} 2\sqrt{n(n+1)} \in A \xrightarrow{\text{(iii)}} 2\sqrt{n(n+1)}(\sqrt{n} + \sqrt{n+1}) = 2n\sqrt{n+1} + 2(n+1)\sqrt{n} \in A \Rightarrow \dfrac{1}{\sqrt{n} + \sqrt{n+1}} = \sqrt{n+1} - \sqrt{n} = (2n+1)(\sqrt{n} + \sqrt{n+1}) - (2n\sqrt{n+1} + 2(n+1)\sqrt{n}) \in A$.

(c) $\sqrt{a} + \sqrt{b} \in A \xrightarrow{\text{(iii)}} (\sqrt{a} + \sqrt{b})^2 = a + b + 2\sqrt{ab} \in A \xrightarrow[-(a+b) \in \mathbb{Z} \subset A]{\text{(iii)}} 2\sqrt{ab} \in A \xrightarrow{\text{(iii)}} 2\sqrt{ab}(\sqrt{a} + \sqrt{b}) = 2a\sqrt{b} + 2b\sqrt{a} \in A \Rightarrow \dfrac{(a-b)^2}{\sqrt{a} + \sqrt{b}} = (a-b)(\sqrt{a} - \sqrt{b}) = a\sqrt{a} + b\sqrt{b} - a\sqrt{b} - b\sqrt{a} = (a+b)(\sqrt{a} + \sqrt{b}) - (2a\sqrt{b} + 2b\sqrt{a}) \in A$. □

Bài toán 6 ([Phư10], 6., p. 20). *Phân hoạch* [9] *thành* A, B*. Chứng minh với mọi cách phân hoạch, luôn tồn tại 1 tập chứa 3 số lập thành 1 cấp số cộng.*

Bài toán 7 ([Phư10], 7., p. 20). *Chứng minh có thể chia* $\mathbb{N}$ *thành* $A, B \subset \mathbb{N}$ *có cùng lực lượng sao cho* $\forall n \in \mathbb{N}$ *tồn tại duy nhất cặp số* $(a, b) \in A \times B$ *thỏa* $n = a + b$.

1.3 Inclusion–exclusion principle – Nguyên lý bao hàm–loại trừ

In combinatorics, the *inclusion–exclusion principle* is a counting technique which generalizes the familiar method of obtaining the number of elements in the union of 2 finite sets; symbolically expressed as $|A \cup B| = |A| + |B| - |A \cap B|$ where A, B: 2 finite sets & $|S|$ indicates the cardinality of a set S (which may be considered as the number of elements of the set, if the set is finite). The formula expresses the fact that the sum of the sizes of the 2 sets may be too large since some elements may be counted twice. The double-counted elements are those in the intersection of the 2 sets & the count is corrected by subtracting the size of the intersection.

The inclusion–exclusion principle, being a generalization of the 2-set case, is perhaps more clearly seen in the case of 3 sets, which for the sets A, B, C is given by $|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |C \cap A| + |A \cap B \cap C|$. This formula can be verified by counting how many times each region in the Venn diagram figure is included in RHS of the formula. In this case, when removing the contributions of over-counted elements, the number of elements in the mutual intersection of 3 sets has been subtracted too often, so must be added back in to get the corrected total.

Generalizing the results of these examples gives the principle of inclusion–exclusion. To find the cardinality of the union of n sets:

1. Include the cardinalities of the sets.
2. Exclude the cardinalities of pairwise intersections.
3. Include the cardinalities of the triple-wise intersections.
4. Exclude the cardinalities of the quadruple-wise intersections.
5. Include the cardinalities of quintuple-wise intersections.
6. Continue, until the cardinality of the n-tuple-wise intersection is included (if n is odd) or excluded (if n is even).

The name comes from the idea that the principle is based on over-generous *inclusion*, followed by compensating *exclusion*. The principle can be viewed as an example of the sieve method extensively used in number theory & is sometimes referred to as the *sieve formula*.

As finite probabilities are computed as counts relative to the cardinality of the probability space, the formulas for the principle of inclusion–exclusion remain valid when the cardinalities of the sets are replaced by finite probabilities. More generally, both versions of the principle can be put under the common umbrella of measure theory.

In a very abstract setting, the principle of inclusion–exclusion can be expressed as the calculation of the inverse of a certain matrix. This inverse has a special structure, making the principle an extremely valuable technique in combinatorics & related areas of mathematics.

> "1 of the most useful principles of enumeration is discrete probability & combinatorial theory is the celebrated principle of inclusion–exclusion. When skillfully applied, this principle has yielded the solution to many a combinatorial problem." – Gian-Carlo Rota

For more details, see, e.g., Wikipedia/inclusion–exclusion principle.

Định lý 2 (Nguyên lý bao hàm–loại trừ)**.**

(i) Với 2 tập hợp hữu hạn A, B bất kỳ, $|A \cup B| = |A| + |B| - |A \cap B|$, $|A \backslash B| = |A| - |A \cap B|$.

(ii) Với 3 tập hợp hữu hạn A, B, C bất kỳ, $|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |C \cap A| + |A \cap B \cap C|$.

(iii) Với 4 tập hợp hữu hạn A, B, C, D bất kỳ, $|A \cap B \cap C \cap D| = |A| + |B| + |C| + |D| - |A \cap B| - |A \cap C| - |A \cap D| - |B \cap C| - |B \cap D| - |C \cap D| + |A \cap B \cap C| + |A \cap B \cap D| + |A \cap C \cap D| + |B \cap C \cap D| - |A \cap B \cap C \cap D|$.

(iv) Với $n \in \mathbb{N}^\star$, A_i, $i = 1, \ldots, n$, là n tập hợp hữu hạn bất kỳ:

$$\left| \bigcup_{i=1}^{n} A_i \right| = \sum_{T \subseteq \{1,\ldots,n\},\, T \neq \emptyset} (-1)^{|T|+1} \left| \bigcap_{i \in T} A_i \right|.$$

(v)

$$\left| \bigcup_{i=1}^{n} A_i \right| \geq \sum_{i=1}^{n} |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j|.$$

Prove that if you take into account only the 1st $m < n$ sums on the right (in the general form of the principle), then you will get an overestimate if m is odd & an underestimate if m is even.

– Chứng minh rằng nếu bạn chỉ tính tổng $m < n$ đầu tiên ở bên phải (theo dạng tổng quát của nguyên lý), thì bạn sẽ nhận được 1 ước tính cao hơn nếu m là số lẻ & 1 ước tính thấp hơn nếu m là số chẵn.

Chứng minh. Vẽ biểu đồ Venn (Venn diagram) để tiện lập luận.

(i) Các phần tử x của tập hợp A gồm 2 loại:

- $x \in A$ nhưng $x \notin B$, i.e., $x \in A\backslash B$: có đúng $|A\backslash B|$ phần tử x như vậy.
- $x \in A$ & $x \in B$, i.e., $x \in A \cap B$: có đúng $|A \cap B|$ phần tử như vậy.

Suy ra tổng số phần tử của tập A bằng $|A\backslash B| + |A \cap B|$, i.e., $|A| = |A\backslash B| + |A \cap B|$, hay $|A\backslash B| = |A| - |A \cap B|$. Chứng minh tương tự cho tập hợp B được: $|B| = |B\backslash A| + |B \cap A|$, hay $|B\backslash A| = |B| - |A \cap B|$. Áp dụng 2 kết quả này được: $|A \cup B| = |A\backslash B| + |A \cap B| + |B\backslash A| = |A| - |A \cap B| + |A \cap B| + |B| - |A \cap B| = |A| + |B| - |A \cap B|$.

(ii) Áp dụng kết quả ý (i) cho $(A, B) = (A, B \cup B)$ được:

$$\begin{aligned}|A \cup B \cup C| &= |A \cup (B \cup C)| = |A| + |B \cup C| - |A \cap (B \cup C)| = |A| + |B| + |C| - |B \cap C| - |(A \cap B) \cup (A \cap C)| \\ &= |A| + |B| + |C| - |B \cap C| - (|A \cap B| + |A \cap C| - |(A \cap B) \cap (A \cap C)|) \\ &= |A| + |B| + |C| - |B \cap C| - |C \cap A| - |A \cap B| + |A \cap B \cap C|.\end{aligned}$$

(iv) [PVT's] Chứng minh bằng phương pháp quy nạp Toán học. Trường hợp $n = 1$ hiển nhiên, $n = 2, 3, 4$ đã chứng minh lần lượt ở (i)–(iii). Giả sử đẳng thức đúng đến $n = N$, i.e.:

$$\left|\bigcup_{i=1}^{N} A_i\right| = \sum_{\emptyset \neq T \subset [N]} (-1)^{|T|+1} \left|\bigcap_{i \in T} A_i\right|.$$

Cần chứng minh đẳng thức cũng đúng với $n = N + 1$, i.e., cần chứng minh:

$$\left|\bigcup_{i=1}^{N+1} A_i\right| = \sum_{\emptyset \neq T \subset [N+1]} (-1)^{|T|+1} \left|\bigcap_{i \in T} A_i\right|.$$

Áp dụng (i) với $A = \bigcup_{i=1}^{N} A_i, B = A_{N+1}$ & giả thiết quy nạp, được:

$$\left|\bigcup_{i=1}^{N+1} A_i\right| = \left|\bigcup_{i=1}^{N} A_i\right| + |A_{N+1}| - \left|A_{N+1} \cap \left(\bigcup_{i=1}^{N} A_i\right)\right| = \sum_{\emptyset \neq T \subset [N]} (-1)^{|T|+1} \left|\bigcap_{i \in T} A_i\right| + |A_{N+1}| - \left|\bigcup_{i=1}^{N} A_{N+1} \cap A_i\right|.$$

Nhận xét: Tập con khác rỗng của $[N + 1]$ gồm 2 loại:

- các tập con của $[N]$, ký hiệu $S_1, \ldots, S_m$ (không có phần tử thứ $N + 1$)
- tập $\{N + 1\}$ & các tập $\{N + 1\} \cup S_i$ (có phần tử thứ $N + 1$).

Suy ra

$$\sum_{\emptyset \neq T \subset [N+1]} (-1)^{|T|+1} \left|\bigcap_{i \in T} A_i\right| = \sum_{\emptyset \neq T \subset [N]} (-1)^{|T|+1} \left|\bigcap_{i \in T} A_i\right| + |A_{N+1}| + \sum_{\emptyset \neq T \subset \{N+1\} \cap S_i} (-1)^{|T|+1} \left|\bigcap_{i \in T} A_i\right|.$$

Như vậy ta chỉ cần chứng minh

$$\left|\bigcup_{i=1}^{N} A_{N+1} \cap A_i\right| = -\sum_{\emptyset \neq T \subset \{N+1\} \cup S_i} (-1)^{|T|+1} \left|\bigcap_{i \in T} A_i\right|.$$

Thật vậy, áp dụng giả thiết quy nạp cho VT được

$$\begin{aligned}\left|\bigcup_{i=1}^{N} A_{N+1} \cap A_i\right| &= \sum_{\emptyset \neq T \subset [N]} (-1)^{|T|+1} \left|\bigcap_{i \in T} A_{N+1} \cap A_i\right| \\ &= -\sum_{\emptyset \neq T \subset [N]} (-1)^{|T|+2} \left|\left(\bigcap_{i \in T} A_i\right) \cap A_{N+1}\right| = -\sum_{\emptyset \neq T' \subset \{N+1\} \cup S_i} (-1)^{|T'|+1} \left|\bigcap_{i \in T'} A_i\right|.\end{aligned}$$

Như vậy đẳng thức cũng đúng với $n = N + 1$. Theo nguyên lý quy nạp toán học, ta có điều phải chứng minh.

(v) Đặt $P(n,k)$: $(-1)^{k+1}\sum_{1\le i_1<\cdots<i_k\le n}\left|\bigcap_{j=1}^{k}A_{i_j}\right|$, đẳng thức vừa chứng minh ở (iv) có thể được viết lại thành

$$\left|\bigcup_{i=1}^{n}A_i\right|=\sum_{k=1}^{n}(-1)^{k+1}\sum_{1\le i_1<\cdots<i_k\le n}\left|\bigcap_{j=1}^{k}A_{i_j}\right|=\sum_{k=1}^{n}P(n,k).$$

Cần chứng minh:

$$\left|\bigcup_{i=1}^{n}A_i\right|\begin{cases}\le\displaystyle\sum_{k=1}^{m}P(n,k),\ \forall m\in[n-1],m\not\vdots 2,\\ \ge\displaystyle\sum_{k=1}^{m}P(n,k),\ \forall m\in[n-1],m\vdots 2,\end{cases}$$

bằng phương pháp quy nạp Toán học. □

2nd chứng minh. (i) (Sử dụng phương pháp liệt kê, [Phư10, VD2, p. 16]) Xuất phát từ phần giao của 2 tập hợp A,B: Giả sử $A\cap B=\{c_1,\ldots,c_p\}$, $A=\{a_1,\ldots,a_m,c_1,\ldots,c_p\}$, $B=\{b_1,\ldots,b_n,c_1,\ldots,c_p\}$, thì $A\cup B=\{a_1,\ldots,a_m,b_1,\ldots,b_n,c_1,\ldots,c_p\}$, nên $|A|=m+p,|B|=n+p,|A\cup B|=m+n+p$, suy ra $|A\cup B|=|A|+|B|-|A\cap B$. (ii) Có thể sử dụng (i) 2 lần liên tiếp như 1st chứng minh hoặc sử dụng phương pháp liệt kê tương tự như (i) của 2nd chứng minh. □

Theorem 2 (Inclusion–exclusion principle)**.** *For any $n\in\mathbb{N}^\star$, & any finite sets $A_1,\ldots,A_n$, one has the identity*

$$\left|\bigcup_{i=1}^{n}A_i\right|=\sum_{i=1}^{n}|A_i|-\sum_{1\le i<j\le n}|A_i\cap A_j|+\sum_{1\le i<j<k\le n}|A_i\cap A_j\cap A_k|-\cdots+(-1)^{n+1}|A_1\cap\cdots\cap A_n|,$$

which can be compactly written as

$$\left|\bigcup_{i=1}^{n}A_i\right|=\sum_{k=1}^{n}(-1)^{k+1}\sum_{1\le i_1<\cdots<i_k\le n}|A_{i_1}\cap\cdots\cap A_{i_k}|,$$

or

$$\left|\bigcup_{i=1}^{n}A_i\right|=\sum_{\emptyset\ne J\subset\{1,\ldots,n\}}(-1)^{|J|+1}\left|\bigcap_{j\in J}A_j\right|.$$

In words, to count the number of elements in a finite union of finite sets, 1st sum the cardinalities of the individual sets, then subtract the number of elements that appear in at least 2 sets, then add back the number of elements that appear in at least 3 sets, then subtract the number of elements that appear in at least 4 sets, & so on. This process always ends since there can be no elements that appear in more than the number of sets in the union.

In applications, it is common to see the principle expressed in its complementary form:

Theorem 3 (Complementary form of inclusion–exclusion principle)**.** *Let S be a finite universal set containing all of the A_i & letting $\overline{A_i}$ denote the complement of A_i in S, by De Morgan's laws, one has*

$$\left|\bigcap_{i=1}^{n}\overline{A_i}\right|=\left|S\setminus\bigcup_{i=1}^{n}A_i\right|=|S|-\sum_{i=1}^{n}|A_i|+\sum_{1\le i<j\le n}|A_i\cap A_j|-\cdots+(-1)^n|A_1\cap\cdots\cap A_n|.$$

As another variant due to J. J. Sylvester of the statement, let $P_1,\ldots,P_n$ be a list of properties that elements of a set S may or may not have, then the principle of inclusion–exclusion provides a way to calculate the number of elements of S that have none of the properties. Just let A_i be the subset of elements of S which have the property P_i & use the principle in its complementary form.

Theorem 4 (Inclusion–exclusion principle in probability)**.** *For any $n\in\mathbb{N}^\star$, & for any events $A_1,\ldots,A_n$ in a probability space $(\Omega,\mathcal{F},\mathbb{P})$:*

(i) For $n=2$, $\mathbb{P}(A_1\cup A_2)=\mathbb{P}(A_1)+\mathbb{P}(A_2)-\mathbb{P}(A_1\cap A_2)$.

(ii) For $n=3$, $\mathbb{P}(A_1\cup A_2\cup A_3)=\mathbb{P}(A_1)+\mathbb{P}(A_2)+\mathbb{P}(A_3)-\mathbb{P}(A_1\cap A_2)-\mathbb{P}(A_2\cap A_3)-\mathbb{P}(A_3\cap A_1)+\mathbb{P}(A_1\cap A_2\cap A_3)$.

(iii) In general

$$\mathbb{P}\left(\bigcup_{i=1}^{n}A_i\right)=\sum_{i=1}^{n}\mathbb{P}(A_i)-\sum_{i<j}\mathbb{P}(A_i\cap A_j)+\sum_{i<j<k}\mathbb{P}(A_i\cap A_j\cap A_k)+\cdots+(-1)^{n-1}\mathbb{P}\left(\bigcap_{i=1}^{n}A_i\right),\tag{1.1}$$

which can be written in closed form as

$$\mathbb{P}\left(\bigcup_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} (-1)^{k-1} \sum_{I\subset[n],\,|I|=k} \mathbb{P}(A_I), \tag{1.2}$$

where the last sum runs over all subsets I of the indices $1,\dots,n$ which contain exactly k elements, & $A_I \coloneqq \bigcap_{i\in I} A_i$ denotes the intersection of all those A_i with index in I.

In particular, if the probability of the intersection A_I only depends on the cardinality of I, i.e., for every $k \in [n]$, there is an a_k s.t. $a_k = \mathbb{P}(A_I)$ for every $I \in [n]$ with $|I| = k$, then (1.2) simplifies to

$$\mathbb{P}\left(\bigcup_{i=1}^{n} A_i\right) = \sum_{k=1}^{n} (-1)^{k-1} \binom{n}{k} a_k.$$

In addition, if the events A_i are independent & identically distributed (i.i.d.), then $\mathbb{P}(A_i) = p$, $\forall i$, & $a_k = p^k$, hence

$$\mathbb{P}\left(\bigcup_{i=1}^{n} A_i\right) = 1 - (1-p)^n.$$

Problem 1. *Prove this theorem (the last result can also be derived more simply by considering the intersection of the complements of the events A_i.)*

According to the Bonferroni inequalities, the sum of the 1st terms in the formula is alternately an upper bound & a lower bound for the LHS. This can be used in cases where the full formula is to cumbersome.

For a general measure space (S, Σ, μ) & measurable subsets $A_1, \dots, A_n$ of finite measure, the above identities also hold when the probability measure $\mathbb{P}$ is replaced by the measure μ.

Theorem 5 (General form)**.**

$$g(A) = \sum_{S\subset A} f(S) \Rightarrow f(A) = \sum_{S\subset A} (-1)^{|A|-|S|} g(S).$$

The combinatorial version Thm. 2 & the probabilistic version Thm. 4 of the inclusion-exclusion principle are instances of 5.

1.4 Problems on inclusion–exclusion principle

Problem 2 (Counting derangements – đếm số quân bài đánh tráo)**.** *Suppose there is a deck of n cards numbered from 1 to n. Suppose a card numbered m is in the correct position if it is the mth card in the deck. How many ways, W, can the cards be shuffled with at least 1 card being in the correct position?*

– Giả sử có 1 bộ bài n lá bài được đánh số từ 1 đến n. Giả sử 1 lá bài được đánh số m ở đúng vị trí nếu nó là lá bài thứ m trong bộ bài. Có bao nhiêu cách, W, có thể xáo trộn các lá bài sao cho ít nhất 1 lá bài ở đúng vị trí?

1.5 Permutations & Combinations – Hoán vị & tổ hợp

Bài toán 8 (Consecutive coin toss – Gieo các đồng xu liên tiếp)**.** *Cho $n, k \in \mathbb{N}^\star$, $k \le n$. Tung 1 đồng xu đồng chất ngẫu nhiên n lần. Tính xác suất lý thuyết của sự kiện: (a) Toàn bộ đều là mặt sấp (ngửa). (b) Có đúng k lần xuất hiện mặt sấp (ngửa). (c) Có ít nhất k lần xuất hiện mặt sấp (ngửa). (d) Có đúng k lần xuất hiện mặt sấp (ngửa) liên tiếp nhau. (e) Có ít nhất k lần xuất hiện mặt sấp (ngửa) liên tiếp nhau.*

Giải. Gọi $X_i \in \{S, N\}$ là biến cố ngẫu nhiên biểu diễn mặt đồng xu trong lần tung thứ i, $\forall i = 1, \dots, n$. Không gian mẫu: $|\Omega| = \prod_{i=1}^{n} 2 = 2^n$. (a) Vì chỉ có 1 trường hợp thuận lợi là $(S, S, \dots, S)$ nên $\mathbb{P}(X_i = S,\ \forall i = 1, \dots, n) = \mathbb{P}(|\{i; X_i = S\}| = n) = \dfrac{1}{2^n}$. Tương tự, vì chỉ có 1 trường hợp thuận lợi là $(N, N, \dots, N)$ nên $\mathbb{P}(X_i = N,\ \forall i = 1, \dots, n) = \mathbb{P}(|\{i; X_i = N\}| = n) = \dfrac{1}{2^n}$. (b) $\mathbb{P}(|\{i; X_i = S\}| = k) = \mathbb{P}(|\{i; X_i = N\}| = k) = \dfrac{C_n^k}{2^n}$, $\forall k = 0, \dots, n$. (c) $\mathbb{P}(|\{i; X_i = S\}| \ge k) = \mathbb{P}(|\{i; X_i = N\}| \ge k) = \dfrac{C_n^k + C_n^{k+1} + \cdots + C_n^n}{2^n} = \dfrac{\sum_{i=k}^{n} C_n^i}{2^n}$, $\forall k = 0, \dots, n$. (d) $\mathbb{P} = \dfrac{n-k+1}{2^n}$. (e) $\mathbb{P} = \dfrac{\sum_{i=k}^{n}(n-i+1)}{2^n} = \dfrac{(n+1)(n-k+1) - \dfrac{(n+k)(n-k+1)}{2}}{2^n}$.

□

Bài toán 9 (Simultaneous coin toss – Gieo các đồng xu đồng thời)**.** *Cho $n, k \in \mathbb{N}^\star$, $k \leq n$. Tung đồng thời n đồng xu đồng chất ngẫu nhiên. Tính xác suất lý thuyết của sự kiện: (a) Toàn bộ đều là mặt sấp (ngửa). (b) Có đúng k lần xuất hiện mặt sấp (ngửa). (c) Có ít nhất k lần xuất hiện mặt sấp (ngửa).*

Giải. Gọi X là biến cố ngẫu nhiên chỉ số mặt S xuất hiện khi tung đồng thời n đồng xu. (a) $\mathbb{P}(X = n) = \mathbb{P}(X = 0) = \frac{1}{n+1}$. (b) $\mathbb{P}(X = k) = \frac{1}{n+1}$ □

Bài toán 10 (Consecutive 2 dice rolls – Gieo 2 xúc xắc lần lượt)**.** *Gieo lần lượt 2 con xúc xắc. Tính xác suất lý thuyết của sự kiện: (a) 2 mặt có cùng số chấm, khác số chấm. (b) Số chấm 2 mặt có cùng tính chẵn lẻ, khác tính chẵn lẻ. (c) Số chấm 2 mặt đều là số nguyên tố, đều là hợp số, có ít nhất 1 số nguyên tố, có ít nhất 1 hợp số. (d) Số chấm 1 mặt là ước (bội) của số chấm trên mặt còn lại. (e) Tổng số chấm 2 mặt bằng $n \in \mathbb{N}$.*

Ans. (e) $f(n) = (\min\{n-1, 6\} - \max\{n-6, 1\} + 1)\mathbf{1}_{n\in\{2,3,\ldots,12\}}$.

Bài toán 11 (Simultaneous 2 dice rolls – Gieo 2 xúc xắc đồng thời)**.** *Gieo đồng thời 2 con xúc xắc. Tính xác suất lý thuyết của sự kiện: (a) 2 mặt có cùng số chấm, khác số chấm. (b) Số chấm 2 mặt có cùng tính chẵn lẻ, khác tính chẵn lẻ. (c) Số chấm 2 mặt đều là số nguyên tố, đều là hợp số, có ít nhất 1 số nguyên tố, có ít nhất 1 hợp số. (d) Số chấm 1 mặt là ước (bội) của số chấm trên mặt còn lại. (e) Tổng số chấm 2 mặt bằng $n \in \mathbb{N}$.*

Bài toán 12 (Consecutive n dice rolls – Gieo n xúc xắc lần lượt)**.** *Gieo lần lượt $n \in \mathbb{N}^\star$ con xúc xắc. Tính xác suất lý thuyết của sự kiện: (a) n mặt có cùng số chấm. (b) n mặt có khác số chấm. (c) Số chấm n mặt có cùng tính chẵn lẻ. (d) Số chấm 1 mặt là ước (bội) của số chấm trên các mặt còn lại. (e) Tổng số chấm n mặt bằng $a \in \mathbb{N}$.*

Bài toán 13 (Simultaneous n dice rolls – Gieo n xúc xắc đồng thời)**.** *Gieo đồng thời $n \in \mathbb{N}^\star$ con xúc xắc. Tính xác suất lý thuyết của sự kiện: (a) n mặt có cùng số chấm. (b) n mặt có khác số chấm. (c) Số chấm n mặt có cùng tính chẵn lẻ. (d) Số chấm 1 mặt là ước (bội) của số chấm trên các mặt còn lại. (e) Tổng số chấm n mặt bằng $a \in \mathbb{N}$.*

Bài toán 14 (Squares & rectangles with same perimeter – Hình vuông & hình chữ nhật cùng chu vi)**.** *Cho $n \in \mathbb{N}^\star$. Viết n thành tổng 2 số: $n = a + b$. Tính xác suất để a, b cùng là độ dài cạnh của 1 hình vuông, xác suất để a, b là độ dài 2 cạnh của 1 hình chữ nhật nếu: (a) $a, b \in \mathbb{N}^\star$. (b) $a, b \in \mathbb{N}$.*

Bài toán 15 (Squares & rectangles with same area – hình vuông & hình chữ nhật cùng diện tích)**.** *Cho $a \in \mathbb{N}^\star, a \geq 2$ có phân tích thừa số nguyên tố $a = \prod_{i=1}^{n} p_i^{a_i} = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}$ với p_i là số nguyên tố, $a_i \in \mathbb{N}^\star$, $\forall i = 1, 2, \ldots, n$. (a) Viết ngẫu nhiên a thành tích của 2 số: $a = bc$. Tính xác suất để b, c là độ dài 2 cạnh của 1 hình chữ nhật, xác suất để b, c cùng là độ dài cạnh của 1 hình vuông nếu: (i) $b, c \in \mathbb{N}$. (ii) $b, c \in \mathbb{Z}$. (b) Lấy ngẫu nhiên 2 số $b, c \in \mathrm{U}(a)$. Tính xác suất để phân số $\frac{b}{c}$: (i) tối giản. (ii) không tối giản.*

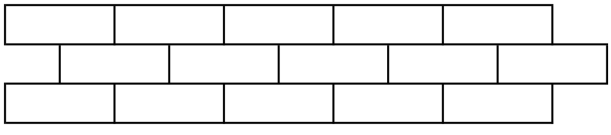
Definition 3 (Prime-counting function)**.** *The* prime-counting function *is the function counting the number of prime numbers less than or equal to some real number x, denoted by $\pi(x) \coloneqq |\{p \in \mathbb{N}^\star | p \text{ is a prime}, \ p \leq x\}|$.*

Định nghĩa 10 (Hàm đếm số số nguyên tố)**.** Hàm đếm số số nguyên tố *là hàm đếm số số nguyên tố nhỏ hơn hoặc bằng $x \in \mathbb{R}$, ký hiệu là $\pi(x) \coloneqq |\{p \in \mathbb{N}^\star | p \text{ là số nguyên tố}, \ p \leq x\}|$.*

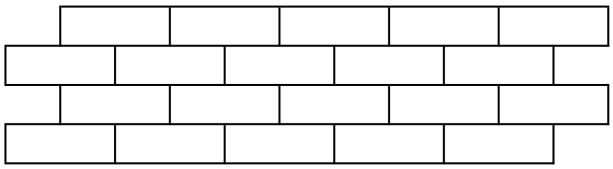
Bài toán 16 (Prime, composite – số nguyên tố, hợp số)**.** *Cho $m, n, k \in \mathbb{N}^\star$. Đặt $A_n = \{1, 2, \ldots, n\}$ là tập hợp n số nguyên dương đầu tiên, $\forall n \in \mathbb{N}^\star$. (a) Lấy m số từ A_n. Tính xác suất để m số này cùng chẵn, cùng lẻ, có ít nhất 1 số chẵn, có ít nhất 1 số lẻ, có đúng k số chẵn, có đúng k số lẻ, có ít nhất k số chẵn, có ít nhất k số lẻ. (b) Lấy m số phân biệt từ A_n. Tính xác suất để m số này đều là số nguyên tố, đều là hợp số, có đúng k số nguyên tố, có đúng k hợp số, có ít nhất 1 số nguyên tố, có ít nhất 1 hợp số, có ít nhất k số nguyên tố, có ít nhất k hợp số. (c) Viết chương trình* Pascal, Python C/C++ *để mô phỏng việc tính các xác suất đó.*

Bài toán 17 (Odd, even – chẵn, lẻ)**.** *Cho $a, b \in \mathbb{Z}, a < b$, $n, k \in \mathbb{N}^\star, n \geq 2, k \leq n$. Đặt $A = [a, b] \cap \mathbb{Z} = \{a, a+1, a+2, \ldots, b-1, b\}$. (a) Lấy 2 số từ tập A. Xét 2 trường hợp phân biệt, không nhất thiết phân biệt. Tính xác suất để 2 số này cùng tính chẵn lẻ, khác tính chẵn lẻ. (b) Lấy n số từ tập A. Tính xác suất để n số này đều chẵn, đều lẻ, cùng tính chẵn lẻ, có đúng k số chẵn, k số lẻ, có ít nhất k số chẵn, k số lẻ. (c) Viết chương trình* Pascal, Python C/C++ *để mô phỏng việc tính các xác suất đó.*

Bài toán 18 (VMC2024B4)**.** *(a) Đếm số cách chọn ra 3 viên gạch, mỗi viên từ 1 hàng trong 3×5 viên gạch xếp xen kẽ, sao cho không có 2 viên gạch nào được lấy ra nằm kề nhau (2 viên gạch được gọi là kề nhau nếu có chung 1 phần của 1 cạnh).*



(b) Đếm số cách chọn ra 4 viên gạch, mỗi viên từ 1 hàng trong 4×5 viên gạch xếp xen kẽ, sao cho không có 2 viên gạch nào được lấy ra nằm kề nhau.



(c) Cho $m, n \in \mathbb{N}^\star$. Đếm số cách chọn ra m viên gạch, mỗi viên từ 1 hàng trong $m \times n$ viên gạch xếp xen kẽ, sao cho không có 2 viên gạch nào được lấy ra nằm kề nhau. (d) Cho $m, n, k \in \mathbb{N}^\star$. Đếm số cách chọn ra k viên gạch, không nhất thiết mỗi viên từ 1 hàng trong $m \times n$ viên gạch xếp xen kẽ, sao cho không có 2 viên gạch nào được lấy ra nằm kề nhau. (e*) Mở rộng cho trường hợp $m \times n$ với số gạch mỗi hàng có thể khác nhau, cụ thể là hàng i chứa $a_i \in \mathbb{N}^\star$ viên gạch, $\forall i = 1, \dots, m$ với 2 trường hợp: (i) Mỗi viên từ 1 hàng. (ii) Lấy $k \in \mathbb{N}^\star$ viên gạch, mỗi hàng có thể lấy nhiều viên.

Nhận xét 1 (Left-right symmetry – Đối xứng trái phải)**.** *Nếu số viên gạch của mỗi hàng bằng nhau & được sắp xen kẽ như (a) & (b), thì thứ tự viên gạch đầu tiên từ bên trái của mỗi hàng lồi ra hay thụt vào không quan trọng, vì có thể lấy đối xứng gương trái–phải để chuyển đổi 2 trường hợp đó. Cũng chú ý đến tính đối xứng trên–dưới (top-bottom symmetry).*

Chứng minh. Số cách chọn gạch từ 3 hàng, mỗi hàng n viên gạch: $(n-1)(n-2)^2 + (n-1)^2 = (n-1)(n^2 - 3n + 3)$, $\forall n \in \mathbb{N}^\star$, $n \geq 2$. Số cách chọn gạch từ 4 hàng, mỗi hàng $n \in \mathbb{N}^\star$ viên gạch: $(n^2 - 3n + 3)^2$, $\forall n \in \mathbb{N}^\star$, $n \geq 2$. □

- C++ codes:
  - (DKAK): https://github.com/NQBH/advanced_STEM_beyond/blob/main/VMC/C++/brick_DPAK.cpp.
  - (NLDK): https://github.com/NQBH/advanced_STEM_beyond/blob/main/VMC/C++/brick_NLDK.cpp.

1.6 Problems: Counting – Bài tập: Đếm

Bài toán 19. *1 lớp có $n \in \mathbb{N}^\star$ sinh viên. Lớp muốn trong bức ảnh có $a \in \mathbb{N}^\star$ ngồi hàng đầu & $n - a$ sinh viên ngồi hàng sau. Đếm số cách.*

Chứng minh. Chọn ra a bạn ngồi hàng đầu & sắp vị trí: có A_n^a cách. Hoán vị $n - a$ bạn còn lại vào hàng sau: có $(n-a)!$ cách. Theo quy tắc nhân, có $A_n^a (n-a)! = \dfrac{n!}{(n-a)!}(n-a)! = n!$ cách. □

Problem 3 ([Sha22], p. 2)**.** *Given $m, n \in \mathbb{N}^\star$, $m \leq n$. Count the number of sequences $a_1, a_2, \dots, a_n$ consisting of m 0's & $n - m$ 1's, if no 2 consecutive terms are both 0's? Write* C/C++, Python *programs to illustrate.*

– Cho $m, n \in \mathbb{N}^\star$, $m \leq n$. Đếm số dãy số $a_1, a_2, \dots, a_n$ gồm m 0's & $n - m$ 1's, nếu không có 2 số hạng liên tiếp nào đều là 0's? Viết chương trình C/C++, Python *để mô phỏng.*

Chứng minh. C_{n-m+1}^m.

C++:

```
#include<bits/stdc++.h>
using namespace std;

long long total = 0;

long long nCk(int n, int k) {
 if (k>n||k<0) return 0;
 long long res=1;
 for(int i=1; i<=k; i++) {
  res*=(n-i+1);
  res/=i;
 }
 return res;
}

void generateSeq(int n, int m, int pos, int last, vector<int>& seq, int cnt0, int cnt1) {
 if (cnt0>m || cnt1>n-m) return;
```

```
 if (pos==n) {
  if(cnt0==m && cnt1==n-m) {
   for(int x:seq) cout<<x<<" ";
   cout<<"\n";
   total++;
  }
 }

 seq.push_back(1);
 generateSeq(n, m, pos+1, 1, seq, cnt0, cnt1+1);
 seq.pop_back();

 if(last!=0) {
  seq.push_back(0);
  generateSeq(n, m, pos+1, 0, seq, cnt0+1, cnt1);
  seq.pop_back();
 }
}


int main() {
 ios_base::sync_with_stdio(0);
 cin.tie(0); cout.tie(0);
 int n, m;
 cin>>n>>m;
 vector<int> seq;
 generateSeq(n, m, 0, -1, seq, 0, 0);
 cout<<"Tong so day hop le: "<<total<<"\n";
 cout<<"(n-m+1)Cm = "<<nCk(n-m+1, m);
}
```

□

Problem 4 ([Sha22], p. 3)**.** *Let $f(n)$ be the number of subsets of $[n]$. Prove that $f(n) = 2^n$, $\forall n \in \mathbb{N}^\star$.*

Problem 5 ([Sha22], p. 3)**.** *Assume $n \in \mathbb{N}$, $n \geq 2$, people given their n hats to a hat-check person. Let $f(n)$ be the number of ways that the hats can be returned, so that everyone has 1 hat, but no one has their own hat. Prove: (a) $f(n) = n! \sum_{i=0}^{n} \frac{(-1)^i}{i!}$, $\forall n \in \mathbb{N}^\star$. (b) $f(n)$ is the nearest integer to $\frac{n!}{e}$.*

Problem 6 ([Sha22], p. 3)**.** *Given $n \in \mathbb{N}^\star$. Let $f(n)$ be the number of subsets of $[n]$ that do not contain 2 consecutive integers. (a) Compute $f(1), f(2), f(3), f(4)$. (b) Prove: $f(n) = f(n-1) + f(n-2)$, $\forall n \in \mathbb{N}$, $n \geq 3$. (c) Prove:*

$$f(n) = \frac{1}{\sqrt{5}} \left(\tau^{n+2} - \overline{\tau}^{n+2} \right), \text{ where } \tau = \frac{1+\sqrt{5}}{2}, \ \overline{\tau} = \frac{1-\sqrt{5}}{2}.$$

1.7 Euler candy problem – Bài toán chia kẹo Euler

Bài toán 20 (Euler candy problem – Bài toán chia kẹo Euler)**.** *Cho $m, n \in \mathbb{N}^\star$. Xét phương trình nghiệm nguyên*

$$\sum_{i=1}^{n} x_i = m. \tag{1.3}$$

(a) Đếm số nghiệm nguyên dương của phương trình (1.3). (b) Đếm số nghiệm nguyên không âm của phương trình (1.3). (c) Đếm số nghiệm nguyên của phương trình

$$\sum_{i=1}^{m} x_i + \sum_{i=1}^{n} y_i = p \ \ s.t. \ \begin{cases} x_i \geq 1, \ \forall i \in [m], \\ y_i \geq 0, \ \forall i \in [n], \end{cases}$$

(d) Đếm số nghiệm nguyên của phương trình

$$\sum_{i=1}^{n} x_i = m \ s.t. \ x_i \geq m_i, \ \forall i \in [n].$$

(e) Đếm số nghiệm nguyên của phương trình

$$\sum_{i=1}^{n} x_i = m \ s.t. \ m_i \leq x_i \leq M_i, \ \forall i \in [n].$$

Chứng minh. (a) C_{m-1}^{n-1}. (b) C_{p+n-1}^{m+n-1}. (d) $C_{m+n-1-\sum_{i=1}^{n} m_i}$. □

1.8 Method of mathematical induction & recurrence – Phương pháp quy nạp toán học & truy hồi/đệ quy

Ideas of method of mathematical induction. In philosophy & in the sciences, the inductive method refers to the process of starting with observations & then looking for general laws & theories. Mathematical induction – which we just called *induction* – is quite different. For us, induction is a method of proof. When we have an infinite sequence $\{P_n\}_{n=1}^{\infty} = P_1, P_2, \ldots$ of (related) mathematical statements that need a proof, then instead of proving each 1 of these statements, we may be able to just prove that whenever 1 of the statements is true, then so is the next statement (i.e., if P_k is true, then so is P_{k+1}). If we manage such a feat, then proving P_1, the 1st statement, starts a domino effect resulting in all of the statements being true.

– Ý tưởng về phương pháp quy nạp toán học. Trong triết học & trong khoa học, phương pháp quy nạp đề cập đến quá trình bắt đầu bằng các quan sát & sau đó tìm kiếm các định luật & lý thuyết chung. Quy nạp toán học – mà chúng ta chỉ gọi là *quy nạp* – thì khá khác. Đối với chúng ta, quy nạp là 1 phương pháp chứng minh. Khi chúng ta có 1 chuỗi vô hạn $\{P_n\}_{n=1}^{\infty} = P_1, P_2, \ldots$ các mệnh đề toán học (có liên quan) cần được chứng minh, thì thay vì chứng minh từng 1 trong số các mệnh đề này, chúng ta có thể chỉ cần chứng minh rằng bất cứ khi nào 1 trong các mệnh đề là đúng, thì mệnh đề tiếp theo cũng vậy (tức là nếu P_k đúng, thì P_{k+1} cũng đúng). Nếu chúng ta thực hiện được kỳ tích như vậy, thì việc chứng minh P_1, mệnh đề đầu tiên, sẽ bắt đầu 1 hiệu ứng domino khiến tất cả các mệnh đề đều đúng.

Intuition. A sequence of *consecutive* pieces of domino falling.

– Trực giác. Phương pháp quy nạp toán học hoạt động như cách 1 chuỗi các quân cờ domino *liên tiếp* rơi xuống sau khi đẩy ngã quân domino đầu tiên (1 quân domino bất kỳ bị ngã sẽ khiến quân domino tiếp theo nó bị ngã).

Principle of Mathematical Induction. Given an infinite sequence of propositions $\{P_n\}_{n=1}^{\infty} = P_1, P_2, \ldots, P_n, \ldots$, in order to prove that all of them are true, it is enough to show 2 things:

- The base case: P_1 is true.
- The inductive step: For all $k \in \mathbb{N}^\star$, if P_k is true, then so is P_{k+1}.

Remark 3. *1 drawback of mathematical induction: to use it, you already have to know the pattern & the answer. Mathematical induction – unlike the inductive method in science – does not help in finding the pattern. 1 strength of induction: it allows you to use P_k to prove P_{k+1}. This is a big advantage – it almost seems like cheating – since often P_k looks very much like P_{k+1}.*

– 1 nhược điểm của quy nạp toán học: để sử dụng nó, bạn phải biết mô hình & câu trả lời. Quy nạp toán học – không giống như phương pháp quy nạp trong khoa học – không giúp tìm ra mô hình. 1 điểm mạnh của quy nạp: nó cho phép bạn sử dụng P_k để chứng minh P_{k+1}. Đây là 1 lợi thế lớn – gần giống như gian lận – vì thường thì P_k trông rất giống P_{k+1}.

Bài toán 21. *Chứng minh: (a)* (Tổng của n số nguyên dương đầu tiên, n số nguyên lẻ đầu tiên, n số nguyên dương chẵn đầu tiên) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, $\sum_{i=1}^{n}(2i-1) = n^2$, $\sum_{i=1}^{n} 2i = n(n+1)$, $\forall n \in \mathbb{N}^\star$. *(b)* (Tổng bình phương của n số nguyên dương đầu tiên, n số nguyên lẻ đầu tiên, n số nguyên dương chẵn đầu tiên) $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$, $\sum_{i=1}(2i-1)^2, \sum_{i=1}^{n}(2i)^2$, $\forall n \in \mathbb{N}^\star$. *(c)* $\sum_{i=1}^{n} i^3 = \left(\sum_{i=1}^{n} i\right)^2 = \frac{n^2(n+1)^2}{4}$, $\forall n \in \mathbb{N}^\star$. *(d) Tìm cách tính* $S(n,k) := \sum_{i=1}^{n} i^k$, $\forall n, k \in \mathbb{N}^\star$. *(e) Tính* , $\forall n \in \mathbb{N}^\star$. *(f) Tính* $\sum_{i=1}(2i-1)^3, \sum_{i=1}^{n}(2i)^3$, $\forall n \in \mathbb{N}^\star$. *()* $\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$, $\forall n \in \mathbb{N}^\star$. *()* $\sum_{i=1}^{n} \frac{1}{\sqrt{i}+\sqrt{i+1}} = \sqrt{n+1} - 1$, $\forall n \in \mathbb{N}^\star$. *()* $\prod_{i=1}^{n} \frac{i^3-1}{i^3+1} = \frac{2(n^2+n+1)}{3n(n+1)}$, $\forall n \in \mathbb{N}^\star$.

See, e.g., Wikipedia/ring of integers.

Bài toán 22. *Cho $m \in \mathbb{N}^\star$ là 1 số không chính phương. Chứng minh: (a) $\exists A, B \in \mathbb{N}^\star$ s.t. $(a+b\sqrt{m})^n = A + B\sqrt{m}$, $\forall n \in \mathbb{N}^\star$. (b) $\exists A, B \in \mathbb{N}^\star$ s.t. $(a-b\sqrt{m})^n = A - B\sqrt{m}$, $\forall n \in \mathbb{N}^\star$.*

Bài toán 23. *Chứng minh: (a)* $(a+1)^n - an - 1 \vdots n^2$, $\forall n \in \mathbb{N}^\star$, $\forall a \in \mathbb{Z}$. *(b)* $(a+1)^n - \dfrac{n(n-1)}{2}a^2 - an - 1 \vdots n^3$, $\forall a \in \mathbb{Z}$.

Problem 7 ([Sha22], P1.1.2, p. 10)**.** *Prove: (a)* (Formula of triangle numbers) $\sum_{i=1}^{n} i = \dfrac{n(n+1)}{2}$, $\forall n \in \mathbb{N}^\star$. *(b)* (Partial sums of geometric sequences)

$$S_n(a) \coloneqq \sum_{i=0}^{n} a^i = \begin{cases} n+1 & \text{if } a = 1, \\ \dfrac{a^{n+1}-1}{a-1} & \text{if } a \neq 1, \end{cases} \quad \forall n \in \mathbb{N}^\star.$$

(c) Compute $1 + \sum_{i=1}^{n} ii!$, $\forall n \in \mathbb{N}^\star$. *(d) Compute* $3\sum_{i=1}^{n} i(i+1)$, $\forall n \in \mathbb{N}^\star$. *(e) Compute* $\sum_{i=1}^{n} i^2$, $\forall n \in \mathbb{N}^\star$.

Hint. (ii) Compute $aS_n(a) - S_n(a)$.

Remark 4. *The sequence of integers* $\left\{\dfrac{n(n+1)}{2}\right\}_{n=1}^{\infty}$ *are called* triangle numbers *since they count the number of dots in progressively larger triangles.*

Bài toán 24 ([Sha22], p. 6)**.** *Trên 1 tờ giấy vuông lớn, vẽ* $n \in \mathbb{N}^\star$ *các đường thẳng bắt đầu từ 1 cạnh của hình vuông & kết thúc ở cạnh còn lại. Mỗi 2 đường thẳng cắt nhau nhưng không có 3 (hoặc nhiều hơn) đường thẳng nào đi qua cùng 1 điểm. Giả sử* $f(n)$ *là số vùng mà các đường thẳng chia tờ giấy. Tính 1 số giá trị của* $f(n)$ *& dự đoán công thức chung của nó.*

Problem 8 ([Sha22], p. 6)**.** *You have* 100 *briefcases numbered* 1 *through* 100*. For any* $n \in \mathbb{N}^\star$*, if a briefcase numbered* n *holds cash, then so does the briefcase numbered* $n+3$*. You open up briefcase numbered* 55 *& it has a stuffed animal in it. Can you conclude anything about any of the other briefcases?*

Bài toán 25 ([Sha22], p. 6)**.** *Bạn có* 100 *cặp được đánh số từ* 1 *đến* 100*. Đối với bất kỳ* $n \in \mathbb{N}^\star$ *nào, nếu 1 cặp được đánh số* n *đựng tiền mặt, thì cặp được đánh số* $n+3$ *cũng vậy. Bạn mở cặp được đánh số* 55 *& bên trong có 1 con thú nhồi bông. Bạn có thể kết luận điều gì về bất kỳ cặp nào khác không?*

1.9 Principle of strong induction – Nguyên lý quy nạp mạnh

Given an infinite sequence of propositions $\{P_n\}_{n=1}^{\infty} = P_1, P_2, \ldots, P_n, \ldots$ in order to demonstrate that all of them are true, it is enough to know 2 things:

- The base case: P_1 is true.
- The inductive step: $\forall k \in \mathbb{N}^\star$, if $P_1, P_2, \ldots, P_k$ are true, then so is P_{k+1}.

1.10 Fibonacci & Lucas numbers

Definition 4 (Fibonacci sequences)**.** Fibonacci sequences *are defined by*

$$\begin{cases} F_0 = 0, \ F_1 = 1, \\ F_n = F_{n-1} + F_{n-2}, \ \forall n \in \mathbb{N}, \ n \geq 2. \end{cases}$$

Bài toán 26 (Fibonacci numbers – Số Fibonacci)**.** *Tính dãy số Fibonacci bằng: (a) Truy hồi* $O(a^n)$ *với* $a \approx 1.61803$*. (b) Quy hoạch động* $O(n)$*. (c) Quy hoạch động cải tiến. Trong mỗi thuật toán, tính cụ thể số lần gọi hàm tính* $F(i)$*, với* $i = 0, 1, \ldots, n$*, số phép cộng đã thực hiện. Tính time- & space complexities.*

C++: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/C++/Fibonacci.cpp.

```
#include <iostream>
using namespace std;
const long nMAX = 10000;

long fib(long i) {
 if (i == 1 || i == 2)
 return 1;
 else
 return fib(i - 1) + fib(i - 2);
}
```

```
// \cite{Thu_Phuong_Tien_Triet_Phuong_KTLT}, p. 443
long fib_recurrence(long n) {
 long ans, Fn_1, Fn_2;
 if (n <= 2)
 ans = 1;
 else {
  Fn_1 = fib_recurrence(n - 1);
  Fn_2 = fib_recurrence(n - 2);
  ans = Fn_1 + Fn_2;
 }
 return ans;
}

// \cite{Thu_Phuong_Tien_Triet_Phuong_KTLT}, p. 443
long fib_dynamic(long n) {
 long F[nMAX + 1];
 F[0] = 0;
 F[1] = F[2] = 1;
 for (int i = 2; i <=n; ++i)
 F[i] = F[i - 1] + F[i - 2];
 return F[n];
}

// \cite{Thu_Phuong_Tien_Triet_Phuong_KTLT}, p. 443
long fib_dynamic_improved(long n) {
 long lastF = 1, F = 1;
 int i = 1;
 while (i < n) {
  F += lastF;
  lastF = F - lastF;
  ++i;
 }
 return F;
}

int main() {
 long n, i;
 cin >> n;
 cout << "Fibonacci sequence of length " << n << ":\n";

 for (i = 0; i <= n; ++i)
 cout << fib(i) << " ";
 cout << "\n";

 for (i = 0; i <= n; ++i)
 cout << fib_recurrence(i) << " ";
 cout << "\n";

 for (i = 0; i <= n; ++i)
 cout << fib_dynamic(i) << " ";
 cout << "\n";

 for (i = 0; i <= n; ++i)
 cout << fib_dynamic_improved(i) << " ";
 cout << "\n";
}
```

Definition 5 (Lucas sequences). *The sequence of* Lucas numbers *are defined by*

$$\begin{cases} L_0 = 2,\ L_1 = 1, \\ L_n = L_{n-1} + L_{n-2},\ \forall n \in \mathbb{N},\ n \geq 2. \end{cases}$$

1.11 Recurrence Relations – Quan hệ truy hồi/hồi quy/đệ quy

Resources – Tài nguyên.

1. Wikipedia/recurrence relation.

2. [Sha22]. Shahriar Shahriari. *An Invitation To Combinatorics*. Sect. 1.3: Recurrence Relations.

Problem 9 ([Sha22], Warm-Up 1.8, p. 19). *We have n dollars. Every day we buy exactly 1 of the following products: Mustard \$1, Mint \$2, Marjoram \$2. Let $f(n)$ be the number of possible ways of spending all the money. E.g., $f(3) = 5$, since the possible ways of spending \$3 are: Mustard-Mustard-Mustard, Mustard-Mint, Mustard-Marjoram, Mint-Mustard, & Marjoram-Mustard. (a) Compute $f(1), f(2)$. (b) Which one(s) (if any) of the following are true? (i) $f(n) = 2f(n-1) + f(n-3)$. (ii) $f(n) = f(n-1) + \frac{n-1}{2}[3 + (-1)^n]$. (iii) $f(n) = f(n-1) + 2f(n-2)$. (d) $f(n) = 2f(n-1) - f(n-2)$. Give adequate & complete reasoning for your answer.*

Solution. (a) $f(1) = 1$ since we can only 1 Mustard. $f(2) = 3$ since we can buy Mustard-Mustard, Mint, or Marjoram.

(b) Suppose we have n\$. On the last day, if we use \$1 to buy Mustard, then the number of possible ways of spending all $\$(n-1)$ is $f(n-1)$. Otherwise, on the last day, if we use \$2 to buy either Mint or Marjoram, then the number of possible ways of spending all $\$(n-2)$ is $f(n-2)$. Combining both cases yields $f(n) = f(n-1) + 2f(n-2)$, $\forall n \in \mathbb{N},\ n \geq 3$. □

You are asked to find a formula for $f(n)$ but not the usual kind of formula of $f(n)$ in terms of n. What you are asked to do is to find a formula that gives $f(n)$ in terms of $f(n-1), f(n-2)$, & possibly other values of the f function. Such a relation is called a *recurrence relation* &, while it does not give a direct closed formula for $f(n)$, it provides an efficient way for computing specific values of the function f.

Problem 10. *Solve the above problem if the prices of these items are changed as follows: (a) Mustard \$1, Mint \$2, & Marjoram \$3. (b) Mustard \$a, Mint \$b, & Marjoram \$c with $a, b, c \in \mathbb{N}^\star$.*

Problem 11 ([Sha22], p. 6). *On a large square piece of paper, draw $n \in \mathbb{N}^\star$ straight lines that start from 1 side of the square & end on another side. Each 2 of the lines intersect but no 3 (or more) lines go through the same point. Let $f(n)$ be the number of regions which the lines split the piece of paper. Compute some values of $f(n)$ & predict its general formula.*

– Trên 1 tờ giấy vuông lớn, vẽ $n \in \mathbb{N}^\star$ các đường thẳng bắt đầu từ 1 cạnh của hình vuông & kết thúc ở cạnh còn lại. Mỗi 2 đường thẳng cắt nhau nhưng không có 3 (hoặc nhiều hơn) đường thẳng nào đi qua cùng 1 điểm. Giả sử $f(n)$ là số vùng/miền mà các đường thẳng chia tờ giấy. Tính 1 số giá trị của $f(n)$ & dự đoán công thức chung của nó.

Question 1. *How do we know that the number of regions does not depend on the configuration of the lines? Maybe if we draw the lines in a different relation to each other, the number of regions will change.*

– Làm sao chúng ta biết được số lượng vùng không phụ thuộc vào cấu hình của các đường? Có thể nếu chúng ta vẽ các đường theo một mối quan hệ khác với nhau, số lượng vùng sẽ thay đổi.

Giải. Let $f(n)$ denote the number of regions created by $n \in \mathbb{N}^\star$ straight lines assuming that each pair of lines intersects & no 3 lines go through the same point (đồng quy). $f(1) = 2, f(2) = 4, f(3) = 7$. Instead of trying to find a formula for $f(n)$ or even trying to prove that $f(n)$ is independent of the position of the lines, we try to determine $f(n)$ in terms of $f(n-1)$. Not only will this allow us to inductively start with $f(1)$ & $f(2)$ & find other values of $f(n)$, but it could also show that $f(n)$ is well defined. We do a thought experiment. Assume that $n-1$ straight lines split the plane into $f(n-1)$ regions. How many additional regions are created when we add 1 more line? If we just zoom onto this new line & follow its path, it will start from 1 side of the square, 1 by 1 cross the $n-1$ other lines, & end at another side of the square. Hence, it will go through n regions created by the original $n-1$ lines. (The 1st region is the one our line is traversing before it hits the 1st line, the 2nd region is between the 1st & the 2nd, & so on until the $(n-1)$th region – i.e., between the $(n-2)$th & $(n-1)$th lines. Finally, the nth region is after the $(n-1)$th line.) Our line line will split each of these n regions into 2, & as a result will add n regions to what was there before, hence $f(n) = f(n-1) + n$, $\forall n \in \mathbb{N}^\star$. This relationship also proves that $f(n)$ does not depend on the configuration of the lines, & t hat it is well defined. (After all, no matter how the lines are configured, if $f(n-1)$ is well-defined, then $f(n)$ must be $f(n-1) + n$. Since $f(1)$ is well defined, by induction all values of $f(n)$ are well defined.) Note that we could have even started with the case $n = 0$. Use mathematical induction to obtain $f(n) = 1 + \frac{n(n+1)}{2}$, $\forall n \in \mathbb{N}$ (just 1 more than the triangle numbers). Thus $f(100) = 1 + \frac{100 \cdot 101}{2} = 5051$. □

Remark 5 (A common approach to some combinatorial problems, [Sha22], Rmk. 1.11, p. 22)**.** *Often a given combinatorial problem is an instance of a sequence of problems indexed by $n \in \mathbb{N}^\star$, where n is a natural parameter for the problem. If we let $f(n)$ denote the answer to the nth instance of our problem, ideally we may want a closed-form formula for $f(n)$. Sometimes, we can find a "recurrence relation" for f, i.e., finding a formula for $f(n)$ in terms of $f(n-1), f(n-2), \ldots$, e.g., $f(n) = f(n-1)+f(n-2)$, the defining recurrence relation for the sequence of Piñgala–Fibonacci numbers. Usually, the recurrence relation itself is not proved by induction. Rather, we often use a "thought experiment". What are the possibilities for the "1st" or "last" step of the problem?*

After we have a recurrence relation & a few base cases, we can easily generate data & record many values for $f(n)$. At this point, there are a number of possible approaches:

- *Use a simple computer program to find any particular value of $f(n)$ that you need.*
- *Look at the values of $f(n)$ for small n & try to guess the pattern. If your guess is correct, sometimes you can translate it to a closed formula, & often you can prove your conjecture using induction. You can use the recurrence relation in your proof by induction. Recurrence relations are especially suited to help with proofs by induction.*
- *Some classes of recurrence relations (e.g., so-called linear recurrence relations) can be solved systematically.*
- *Sometimes you can "unwind" the recurrence relation.*
- *Sometimes you can use "generating functions" to get information about the sequence $\{f(n)\}_{n=0}^{\infty}$.*

Guessing the pattern from the starting values of an infinite sequence is not easy. For 1 thing, there are many distinct infinite sequences that agree in the 1st few terms – after all, there are not that many very small integers, &, in contrast, there are many infinite sequences of integers. If you have the beginning of a sequence of numbers, & are wondering what the pattern could be, 1 very fun tool is the Online Encyclopedia of Integer Sequences *at* https://oeis.org/

Question 2 (1st step vs. last step)**.** *When should we focus on the 1st step of a problem to establish a recurrence relation? When on the last step?*

– Khi nào chúng ta nên tập trung vào bước đầu tiên của một bài toán để thiết lập mối quan hệ lặp lại? Khi nào là bước cuối cùng?

A recurrence, for the integer valued function f, of the form $f(n) = \sum_{i=1}^{k} \alpha_i f(n-i) + g(n) = \alpha_1 f(n-1) + \alpha_2 f(n-2) + \cdots + \alpha_k f(n-k) + g(n)$, where α_i are scalars, $\forall i \in [k]$, & g is a function of n, is called a *linear recurrence relation*. If you are lucky & the recurrence relation is linear, then the most fruitful method is to 1st ignore the initial conditions & find, basically by an educated guess, a set of functions that satisfy the recurrence relation, & then use the initial conditions to choose a function that satisfies both the initial conditions & the recurrence relation. The key observation – which will be used often – is that if 2 functions have the same initial values & satisfy the same recurrence conditions, then they will have to be the same for all later inputs. See [Sha22, Sect. 1.4] where both linear recurrence relations & the idea of unwinding a recurrence relation were explored.

Bài toán 27. *Cho $m, n \in \mathbb{N}^\star$ cố định. Đếm số nghiệm nguyên dương của phương trình $\sum_{i=1}^{d} x_i = n$ trong đó x_i chỉ có thể nhận 1 trong m giá trị cho trước $a_1, a_2, \ldots, a_m$, i.e., $x_i \in \{a_1, \ldots, a_m\}$, $\forall i \in [d]$, $d \in \mathbb{N}^\star$ có thể thay đổi.*

Remark 6 ([Sha22], Rmk. 1.11, p. 22)**.** *The problems in this section are ment to give you experience in setting up a recurrence relation for a counting problem. This skill will be used throughout the book. In some problems, you are also asked to use the recurrence relation, generate some data, guess the pattern, & prove it by induction. The 2 techniques: solving linear recurrences & unwinding when applicable give a better alternative to "guess-the-pattern" of the current section. Yet a 4th method – using generating functions – will be discussed in [Sha22, Chap. 9].*

1.11.1 Problems: Recurrence relation – Bài tập: Quan hệ hồi quy

Problem 12 ([Sha22], P1.3., p. 22)**.**

Problem 13 ([Sha22], P1.3., p. 22)**.**

Problem 14 ([Sha22], P1.3., p. 22)**.**

Problem 15 ([Sha22], P1.3., p. 22)**.**

Problem 16 ([Sha22], P1.3., p. 23)**.**

Problem 17 ([Sha22], P1.3., p. 23)**.**

Problem 18 ([Sha22], P1.3., p. 23)**.**

Problem 19 ([Sha22], P1.3., p. 23)**.**

Problem 20 ([Sha22], P1.3., p. 23).

Problem 21 ([Sha22], P1.3., p. 23).

Problem 22 ([Sha22], P1.3., p. 23).

Problem 23 ([Sha22], P1.3., p. 23).

1.12 Linear Recurrence Relations & Unwinding a Recurrence Relation –

In this section, we consider 2 methods for attacking recurrence relations. When they work – & there are many common recurrence relations that can be solved using these methods – they work well & give a closed formula for the underlying function. However, there are also plenty of recurrence relations that defy both of these methods.

– Trong phần này, chúng ta sẽ xem xét 2 phương pháp để tấn công các mối quan hệ tái diễn. Khi chúng hoạt động – & có nhiều mối quan hệ tái diễn phổ biến có thể được giải quyết bằng các phương pháp này – chúng hoạt động tốt & đưa ra một công thức đóng cho hàm cơ bản. Tuy nhiên, cũng có rất nhiều mối quan hệ tái diễn thách thức cả hai phương pháp này.

Problem 24 ([Sha22], Warm-Up 1.12, p. 27). *Let $f(n) = \alpha n + \beta$, where $\alpha, \beta \in \mathbb{R}$. Find α, β s.t. the resulting function f satisfies the recurrence relation $f(n) = 5f(n-1) - 6f(n-2) + n$, $\forall n \in \mathbb{N}$, $n \geq 2$.*

Definition 6 (Homogeneous- & nonhomogeneous linear recurrence relations, [Sha22], Def. 1.15, pp. 28–29). *Let $\{f(n)\}_{n=0}^{\infty} = f(0), f(1), \ldots, f(n), \ldots$ be a sequence of real (or complex) numbers. Assume that*

$$f(n) = \sum_{i=1}^{k} \alpha_i f(n-i) = \alpha_1 f(n-1) + \alpha_2 f(n-2) + \cdots + \alpha_k f(n-k) \text{ for some } k \in \mathbb{N}^\star \text{ \& some } \alpha_i \in \mathbb{R},\ \forall i \in [k]. \tag{1.4}$$

We then say that f satisfies a homogeneous linear recurrence relation *(with constant coefficients). If*

$$f(n) = \sum_{i=1}^{k} \alpha_i f(n-i) + g(n) = \alpha_1 f(n-1) + \alpha_2 f(n-2) + \cdots + \alpha_k f(n-k) + g(n) \text{ for some } k \in \mathbb{N}^\star,\ \text{some } \alpha_i \in \mathbb{R},\ \forall i \in [k], \tag{1.5}$$

& some nonzero function g, then we say that f satisfies a nonhomogeneous linear recurrence relation *(with constant coefficients).*

Example 2 ([Sha22], Ex. 1.16, p. 29). *A sequence $\{a_n\}_{n=0}^{\infty}$ with the recurrence $a_n = a_{n-1} + a_{n-2}$ satisfies a homogeneous linear recurrence, while a sequence $\{b_n\}_{n=0}^{\infty}$ with the recurrence relation $b_n = b_{n-1} + b_{n-2} + n^2$ satisfies a nonhomogeneous linear recurrence.*

We 1st formalize the "linearity" property of *linear* recurrence relations.

Lemma 1 ([Sha22], Lem. 1.17, p. 29). *(a) Assume that the functions $f_1, f_2, \ldots, f_m$ satisfy the homogeneous linear recurrence of (1.4). Then, for scalars $\beta_1, \ldots, \beta_m$, every linear combination $\sum_{i=1}^{m} \beta_i f_i = \beta_1 f_1 + \beta_2 f_2 + \cdots + \beta_m f_m$ also satisfies the homogeneous linear recurrence of (1.4).*

(b) If functions h_1, h_2 satisfy the nonhomogeneous linear recurrence (1.5), then their difference $h_1 - h_2$ satisfies the homogeneous linear recurrence of (1.4).

1.13 Pigeonhole Principle & Ramsey Theory – Nguyên Lý Chuồng Bồ Câu & Lý Thuyết Ramsey

Problem 25 ([Sha22], p. 42). *In an ancient cave, you see a seemingly arbitrary list of 100 positive integers carved in a row. You add up all the integers & the sum is 152. Show that, regardless of what the integers are, among them you can locate an unbroken block of adjacent integers that add up to exactly 47.*

1.13.1 Dirichlet/Pigeonhole principle – Nguyên lý Dirichlet/chuồng bồ câu

Problem 26 ([Sha22], Warm-Up 2.1, p. 42). *A bag contains 100 apples, 100 bananas, 100 oranges, & 100 pears. If, every minute, I randomly pick 1 piece of fruit out of the bag, how long will it be before I am assured of having picked at least a dozen pieces of fruit of the same kind?*

– 1 túi chứa 100 quả táo, 100 quả chuối, 100 quả cam, & 100 quả lê. Nếu mỗi phút, tôi ngẫu nhiên hái 1 quả từ túi, thì sau bao lâu tôi chắc chắn đã hái được ít nhất một tá quả cùng loại?

An almost trivial fact:

Theorem 6 (Dirichlet/Pigeonhole principle, [Sha22], Thm. 2.2, p. 42)**.** *Given $n \in \mathbb{N}^\star$. If we put $n+1$ pigeons into n pigeonholes, then at least 1 pigeonhole contains ≥ 2 pigeons.*

Theorem 7 (Pigeonhole principle, reformulated, [Sha22], Thm. 2.3, p. 42)**.** *Let P, H be 2 finite sets. Let $f : P \to H$ be a function. If $|P| > |H|$, then f is not 1-1.*

1.14 Stirling Numbers – Các Số Stirling

Resources – Tài nguyên.

1. Wikipedia/Stirling number.

2. TEX Stack Exchange/how to write Stirling numbers of the 2nd kind.

In mathematics, *Stirling numbers* arise in a variety of analytic & combinatorial problems. They are named after James Stirling, who introduced them in a purely algebraic setting in his book *Methodus differentialis* (1730). They were rediscovered & given a combinatorial meaning by Masanobu Saka in his 1782 *Sanpō-Gakkai (The Sea of Learning on Mathematics)*.

– Trong toán học, *số Stirling* phát sinh trong nhiều bài toán phân tích & tổ hợp. Chúng được đặt theo tên của James Stirling, người đã giới thiệu chúng trong bối cảnh hoàn toàn là đại số trong cuốn sách Methodus differentialis (1730) của ông. Chúng được Masanobu Saka khám phá lại & đưa ra ý nghĩa tổ hợp trong cuốn Sanpō-Gakkai (Biển học về toán học) năm 1782 của ông.

2 different sets of numbers bear this name: the Stirling numbers of the 1st kind & the Stirling numbers of the 2nd kind. Additionally, Lah numbers are sometimes referred to as Stirling numbers of the 3rd kind. Each kind is detailed on its respective article, this one serving as a description of relations between them.

– 2 bộ số khác nhau mang tên này: số Stirling loại 1 & số Stirling loại 2. Ngoài ra, số Lah đôi khi được gọi là số Stirling loại 3. Mỗi loại được trình bày chi tiết trong bài viết tương ứng, bài viết này đóng vai trò mô tả mối quan hệ giữa chúng.

A common property of all 3 kinds is that they describe coefficients relating 3 different sequences of polynomials that frequently arise in combinatorics. Moreover, all 3 can be defined as the number of partitions of n elements into k nonempty subsets, where each subset is endowed with a certain kind of order (no order, cyclical, or linear).

– 1 đặc tính chung của cả 3 loại là chúng mô tả các hệ số liên quan đến 3 chuỗi đa thức khác nhau thường xuất hiện trong tổ hợp. Hơn nữa, cả 3 đều có thể được định nghĩa là số phân hoạch của n phần tử thành k tập con không rỗng, trong đó mỗi tập con được ban cho 1 loại thứ tự nhất định (không có thứ tự, tuần hoàn hoặc tuyến tính).

Notation for Stirling numbers. Several different notations for Stirling numbers are in use. Ordinary (signed) *Stirling numbers of the 1st kind* are commonly denoted $s(n,k)$. *Unsigned Stirling numbers of the 1st kind*, which count the number of permutations of n elements with k disjoint cycles, are denoted $\left[{n \atop k}\right] = c(n,k) = |s(n,k)| = (-1)^{n-k}s(n,k)$. *Stirling numbers of the 2nd kind*, which count the number of ways to partition a set of n elements into k nonempty subsets: $\left\{{n \atop k}\right\} = S(n,k) = S_n^{(k)}$.

–*Ký hiệu cho số Stirling.* Có 1 số ký hiệu khác nhau cho số Stirling đang được sử dụng. Số Stirling thông thường (có dấu) *Số Stirling loại 1* thường được ký hiệu là $s(n,k)$. *Số Stirling không dấu loại 1*, đếm số hoán vị của n phần tử với k chu trình rời rạc, được ký hiệu là $\left[{n \atop k}\right] = c(n,k) = |s(n,k)| = (-1)^{n-k}s(n,k)$. *Số Stirling loại 2*, đếm số cách phân hoạch 1 tập hợp n phần tử thành k tập con không rỗng: $\left\{{n \atop k}\right\} = S(n,k) = S_n^{(k)}$.

The notation of brackets & braces, in analogy to binomial coefficients, was introduced in 1935 by Jovan Karamata & promoted later by Donald Knuth, though the bracket notation conflicts with a common notation for Gaussian coefficients. Another infrequent notation is $s_1(n,k), s_2(n,k)$.

– Ký hiệu của ngoặc & dấu ngoặc nhọn, tương tự như hệ số nhị thức, được giới thiệu vào năm 1935 bởi Jovan Karamata & sau đó được Donald Knuth thúc đẩy, mặc dù ký hiệu ngoặc nhọn xung đột với ký hiệu chung cho hệ số Gaussian. 1 ký hiệu không thường xuyên khác là $s_1(n,k), s_2(n,k)$.

1.14.1 Stirling Numbers of Type 1 – Số Stirling Loại 1

Resources – Tài nguyên.

1. Wikipedia/Stirling numbers of the 1st kind.

In mathematics, especially in combinatorics, *Stirling numbers of the 1st kind* arise in the study of permutations. In particular, the unsigned Stirling numbers of the 1st kind count permutations according to their number of cycles (counting fixed points as cycles of length 1).

– Trong toán học, đặc biệt là trong tổ hợp, *số Stirling loại 1* phát sinh trong nghiên cứu về hoán vị. Đặc biệt, số Stirling không dấu loại 1 đếm các hoán vị theo số chu kỳ của chúng (đếm các điểm cố định là chu kỳ có độ dài 1).

The Stirling numbers of the 1st & 2nd kind can be understood as inverses of one another when viewed as triangular matrices. Identities linking the 2 kinds appear in the article on Wikipedia/Stirling numbers.

– Số Stirling loại 1 & loại 2 có thể được hiểu là nghịch đảo của nhau khi xem như ma trận tam giác. Các bản sắc liên kết 2 loại này xuất hiện trong bài viết về Wikipedia/Stirling numbers.

Definition 7 (Definition of Stirling numbers of type 1 by algebra)**.** *The Stirling numbers of the 1st kind are the coefficients* $s(n,k)$ *in the expansion of the falling factorial* $(x)_n = x(x-1)(x-2)\cdots(x-n+1)$ *into powers of the variable* x*:*

$$(x)_n = \sum_{k=0}^{n} s(n,k)x^k.$$

Example 3 (Some 1st Stirling numbers of the 1st kind – Vài số Stirling loại 1 đầu tiên)**.** *(a) When* $n=1$*,* $(x)_1 = x \Rightarrow s(1,1) = 1$*. (b) When* $n=2$*,* $(x)_2 = x(x-1) = x^2 - x \Rightarrow s(2,2) = 1, s(2,1) = -1$*. (c) When* $n=3$*,* $(x)_3 = x(x-1)(x-2) = x^3 - 3x^2 + 2x \Rightarrow s(3,3) = 1, s(3,2) = -3, s(3,1) = 2$*. (d) When* $n=4$*,* $(x)_4 = x(x-1)(x-2)(x-3) = x^4 - 6x^3 + 11x^2 - 6x \Rightarrow s(4,4) = 1, s(4,3) = -6, s(4,2) = 11, s(4,1) = -6$*. (d) When* $n=5$*,* $(x)_5 = x(x-1)(x-2)(x-3)(x-4) = x^5 - 10x^4 + 35x^3 - 50x^2 + 24x \Rightarrow s(5,5) = 1, s(5,4) = -10, s(5,3) = 35, s(5,2) = -50, s(5,1) = 24$*.*

The unsigned Stirling numbers may also be defined algebraically as the coefficients of the rising factorial:

$$x^{\bar{n}} = x(x+1)\cdots(x+n-1) = \prod_{i=1}^{n}(x+i-1) = \sum_{k=0}^{n} \begin{bmatrix} n \\ k \end{bmatrix} x^k.$$

1.14.2 Stirling Numbers of Type 2 – Số Stirling Loại 2

Resources – Tài nguyên.

1. Wikipedia/Stirling numbers of the 2nd kind.

2. [HT24].

Problem 27 ([Sha22], Warm-Up 6.1, p. 193)**.** Kaitlyn & Megan *together take a course in Latin American literature in translation. They buy 1 copy each of* The President, The Labyrinth of Solitude, The Death of Artemio Cruz, The Time of the Hero, 100 Years of Solitude, The House of the Spirit, *&* The Motorcycle Diaries*. At the end of the course, they want to split the books between themselves. In how many ways can this be done if the only condition is that each gets at least* 1 *book?*

Motivation of new tools beyond permutations & combinations. While the elementary counting methods, including the use of permutations & combinations, can be used to solve many counting problems, other straightforward sounding problems will benefit from new tools, & cannot directly be reduced to permutations & combinations.

Problem 28. *Count the number of ways to put* s *distinct balls into* t *distinct boxes for which there is no restriction on the box capacities.*

If we have s distinct balls & t distinct boxes, & no restriction on the box capacities, then there are t distinct choices for where to put the 1st ball, the same t choices for where the 2nd ball should go, & so on, hence the total number of choices is t^s.

Example 4 ([Sha22], Example 6.2, pp. 193–194)**.** *We have* 4 *distinct balls* $1, 2, 3, 4$ *&* 3 *identical boxes. In how many ways can we place the balls into boxes s.t. no box is empty?s This is the same as asking: In how many ways can we group the balls into* 3 *nonempty parts?*

The answer is 6*:* $\{1,2,3,4\}, \{1,\{2,4\},3\}, \{1,\{2,3,\},4\}, \{\{1,2\},3,4\}, \{\{1,3\},2,4\}, \{\{1,4\},2,3\}$*, which is found by brute force & by enumerating all the possibilities. This method is clearly not going to work with much larger examples, & even if it did work, it does not give us any insight into the problem.*

In enumerative combinatorics, we often encounter new counting problems, i.e., even if we don't know the answer, we give a name – often a function of 1 or more variables – to the answer to the problem.

> "Sometimes naming a thing – giving it a name or discovering its name – helps one to begin to understand it. Knowing the name of a thing & knowing what that thing is for gives me even more of a handle on it." – Octavio Butler, *Parable of the Sower*

We then proceed to find properties of the function. If we can find a recurrence relation for the function, then we can calculate many of the values. The recurrence relation will also allow us to prove further properties of the function – possibly even a formula – using induction.

Definition 8 (Stirling numbers of the 2nd kind). *Let $s, t \in \mathbb{N}$. We define $\left\{ {s \atop t} \right\}$ to be the number of ways of putting s distinct balls into t identical boxes with at least 1 ball per box. Equivalently, $\left\{ {s \atop t} \right\}$ is the number of ways of partitioning a set of s elements, i.e.,*

$$[s] = \begin{cases} \{1, 2, \ldots, s\} & \text{if } s > 0, \\ [0] = \emptyset & \text{if } s = 0. \end{cases}$$

into t nonempty parts. $\left\{ {s \atop t} \right\}$ is called a Stirling number of the 2nd kind *(Some denote the Stirling numbers of the 2nd kind by $S(s,t)$ or $s_2(s,t)$.)*

Remark 7. *We read $\left\{ {s \atop t} \right\}$ as "s subset t" or "s squig t". Also, $\left\{ {0 \atop 0} \right\} = 1$. This could be mandated separately as a part of the definition, but that is really not necessary. If we have no balls & no boxes, can I place each of the balls in some box in a way that none of the boxes is empty? Yes. We are done before we start, since we don't have any balls*[1] *to worry about, & there are no boxes to remain empty. Hence, the conditions – distributing all the balls & making sure that all the boxes are nonempty – are satisfied trivially: we do nothing & hence there is exactly 1 way of placing 0 distinct balls in 0 identical boxes with no empty boxes.*

Question 3. *How to calculate $\left\{ {s \atop t} \right\}$.*

A few cases are easy to calculate.

Lemma 2 ([Sha22], Lem. 6.6, p. 195). *Let $s \in \mathbb{N}^\star, t \in \mathbb{N}$. Then: (a) $\left\{ {s \atop t} \right\} = 0$ if $t > s$. (b) $\left\{ {s \atop s} \right\} = 1$. (c) $\left\{ {s \atop 1} \right\} = 1$. (d) $\left\{ {s \atop s-1} \right\} = \binom{s}{2}$. (e) $\left\{ {s \atop 2} \right\} = 2^{s-1} - 1$.*

Chứng minh. See [Sha22, p. 195]. Since $\left\{ {s \atop t} \right\}$ is the number of ways of putting s distinct balls into t identical boxes while making sure that no box goes empty. (a) There are too many boxes, & hence there are 0 ways of making sure no box goes empty. (b) There are many boxes as balls, & so we must put each ball in a separate box. (c) There is only 1 box, & so all the balls go into that 1 box. (d) We have 1 fewer box than the number of balls, & so we only have to decide which 2 balls go into the same box. There are exactly $\binom{s}{2}$ ways of choosing 2 balls to share a box. □

Problem 29. *Given $s \in \mathbb{N}^\star$. Compute: (a) $\left\{ {s \atop 2} \right\}$. (b) $\left\{ {s \atop 3} \right\}$. (c) $\left\{ {s \atop 4} \right\}$. (d) $\left\{ {s \atop s-2} \right\}$. (e) $\left\{ {s \atop s-3} \right\}$. (f) $\left\{ {s \atop s-4} \right\}$.*

Bài toán 28. *Viết chương trình* C/C++, Pascal, Python *để: (a) Tính số Stirling loại 2 $\left\{ {s \atop t} \right\}$ với $s, t \in \mathbb{N}$ được nhập vào, bằng: (i) Đệ quy. (ii) Quy hoạch động. (b) Tạo bảng số Stirling loại 2 $\left(\left\{ {s \atop t} \right\} \right)_{s,t=0}^{n}$ với $n \in \mathbb{N}^\star$ được nhập vào.*

Theorem 8 ([Sha22], Thm. 6.7, p. 196).

$$\left\{ {s \atop t} \right\} = t \left\{ {s-1 \atop t} \right\} + \left\{ {s-1 \atop t-1} \right\}, \ \forall s, t \in \mathbb{N}^\star.$$

Example 5 ([Sha22], Ex. 6.8, pp. 197–198). *Given $s, t \in \mathbb{N}^\star$. Assume that we want to partition $[s] = \{1, 2, \ldots, s\}$ into t nonempty distinct parts. I.e., we have s distinct balls & t distinct boxes, & we want to distribute the balls into the boxes while making sure that none of the boxes are empty. If we had allowed empty boxes, the count would be easy. There are t choices for each ball, & the choices are independent of each other. So the total number of choices would be t^s. However, some of these choices require putting all the balls into just $1, 2, \ldots, t-2$ or t nonempty parts. We could try to adjust the count by subtracting the choiecs that lead to empty parts by the inclusion–exclusion principle Thm. 2 but here we use a different strategy. The boxes are distinct, & so assume each has an identifying label on it. 1st take the labels off so that the boxes become identical, distribute the balls, & then put the labels back on. The total number of ways of partitioning $[s]$ into t nonempty parts is $\left\{ {s \atop t} \right\}$. Putting the labels back on, we have t choices for where the 1st label goes, $t-1$ choices for the 2nd label, …, & 1 choice for the last one. Hence, the total number of ways of playing s distinct balls into t distinct nonempty boxes is $t! \left\{ {s \atop t} \right\}$.*

Theorem 9 ([Sha22], Thm. 6.9, p. 198). *Let $s, t \in \mathbb{N}$, & let $R = \{\infty \cdot U_1, \infty \cdot U_2, \ldots, \infty \cdot U_t\}$ be a multiset with t types of elements, each with an infinite repetition number. Then the following numbers are equal: (a) The number of ways of placing s distinct balls into t nonempty distinct boxes. (b) The number of s-permutations of R that contain each of $U_1, \ldots, U_t$ at least once. (c) $t! \left\{ {s \atop t} \right\}$.*

Chứng minh. See [Sha22], p. 198. (a) = (c) 1st partition the s balls into t nonempty parts & then label the parts with the name of each box. There are $\left\{ {s \atop t} \right\}$ ways of doing the former, & $t!$ ways of doing the latter. (a) = (b) Think of the U's (i.e., $\{U_i\}_{i=1}^{t}$) as t distinct boxes, & have s balls numbered 1 through s. Putting a ball in a box chooses the box & determine the place of the box in a list. Hence, every placing of s distinct balls into t nonempty distinct boxes corresponds to an ordered list of s of the elements of R in such a way that each U appears at least once. □

[1] Không được dịch thành: không có bi/dái, i.e., tiếng lóng của nhát gan.

Theorem 10 ([Sha22], Thm. 6.10, p. 198)**.** *Let $s,t \in \mathbb{N}$. The number of ways of placing s distinct balls into t identical boxes (with empty boxes allowed), or equivalently the number of partitions of $[s]$ into t or fewer parts, is $\sum_{i=0}^{t} \left\{ {s \atop i} \right\}$.*

Chứng minh. You could put s balls into $0, 1, \ldots, t$ nonempty boxes, corresponding to $\left\{ {s \atop 0} \right\}, \left\{ {s \atop 1} \right\}, \ldots, \left\{ {s \atop t} \right\}$, resp. Summing up all of these gives the desired result. □

Problem 30 ([Sha22], p. 193)**.** *Let $n, k \in \mathbb{N}^\star$, $n \geq k$. Each time you open your favorite language app, an advertisement randomly chosen from among k possible choices pops up. Far from being annoyed, in fact, you would like to see each of the k ads. Assuming that the algorithm is equally likely to choose any of the ads each time, what is the probability that it takes you exactly n tries to see all the k ads?*

Remark 8. *Cụm từ "takes you exactly n tries to see all the k ads" nghĩa là phải tới đúng lần thử thứ n thì mới gom đủ bộ k quảng cáo, tinh hoa mới hội tụ, còn $n-1$ lần thử trước đó, dù cho nhiều cỡ nào, thì chỉ mới xem được $k-1$ quảng cáo, còn thiếu đúng 1 loại quảng cáo còn lại để đủ bộ quảng cáo.*

Solution. See [Sha22, p. 199]. For each try, there are k choices, & so the total number of sequences of ads that you could possibly see is k^n. From among these, how many include all the k ads in such a way that all n tries were necessary to see all k? It must have been that 1 of the ads is shown exactly once on the nth try. Choose which one. There are k choices. Now we are left with $k-1$ ads & $n-1$ tries. Think of the $n-1$ tries are distinct balls & $k-1$ ads as distinct boxes. We have to put the balls into the boxes in such a way that no box remains empty. The number of ways of doing this is $(k-1)!\left\{ {n-1 \atop k-1} \right\}$. Hence, the total number of ways of getting all the k ads in n tries (& not earlier) is $k(k-1!\left\{ {n-1 \atop k-1} \right\} = k!\left\{ {n-1 \atop k-1} \right\}$. The sought-after probability is then $\frac{k!}{k^n}\left\{ {n-1 \atop k-1} \right\}$. □

1.14.3 Problems: Stirling numbers

1.15 Bell Numbers – Số Bell

Resources – Tài nguyên.

1. Wikipedia/Bell number.

2. [Sha22]. Shahriar Shahriari. *An Invitation To Combinatorics.*

Bell numbers are named after E. T. Bell, who wrote a paper about them in 1938. Bell himself wrote that these numbers had been rediscovered many times, & had been studied by many mathematicians. 1 of the earliest appearances of Bell numbers is in Japan around 1500. In an incense-comparing game called *genji-kou*, a host would burn 5 different packets of incense. The guests would have to decide which, if any, of the incenses were the same. So a solution may be $\{\{1,3\},\{2,4,5\}\}$, indicating that the 1st & the 3rd were the same incense, as were the other 3. So, the set of all possible solutions is all the partitions of $[5]$, & the number of such solutions is $B(5) = 52$. For ease of remembering, each of the 52 partitions was named after 1 chapter of the classic 11th-century *Tale of Genji*. Following this tradition, Japanese mathematicians e.g. Seki Takakazu (circa 1642–1708) & his student Yoshisuke Matsunaga (1690–1744) studied partitions of finite sets systematically. Matsunaga, e.g., gave a recurrence relation for the Bells numbers & posed & gave a solution to the problem of finding n if we know that $B(n) = 678570$.

– Số Bell được đặt theo tên của E. T. Bell, người đã viết 1 bài báo về chúng vào năm 1938. Bản thân Bell đã viết rằng những con số này đã được khám phá lại nhiều lần, & đã được nhiều nhà toán học nghiên cứu. 1 trong những lần xuất hiện sớm nhất của số Bell là ở Nhật Bản vào khoảng năm 1500. Trong 1 trò chơi so sánh hương gọi là *genji-kou*, 1 người chủ nhà sẽ đốt 5 gói hương khác nhau. Các vị khách sẽ phải quyết định xem hương nào, nếu có, là giống nhau. Vì vậy, 1 giải pháp có thể là $\{\{1,3\},\{2,4,5\}\}$, chỉ ra rằng hương thứ nhất & hương thứ 3 là cùng 1 hương, giống như 3 hương còn lại. Vì vậy, tập hợp tất cả các giải pháp có thể là tất cả các phân hoạch của $[5]$, & số các giải pháp như vậy là $B(5) = 52$. Để dễ nhớ, mỗi 1 trong 52 phân vùng được đặt tên theo 1 chương của tác phẩm kinh điển thế kỷ 11 *Tale of Genji*. Theo truyền thống này, các nhà toán học Nhật Bản ví dụ như Seki Takakazu (khoảng 1642–1708) & học trò của ông Yoshisuke Matsunaga (1690–1744) đã nghiên cứu các phân vùng của các tập hợp hữu hạn 1 cách có hệ thống. Matsunaga, ví dụ, đã đưa ra 1 mối quan hệ đệ quy cho các số Bells & đặt ra & đưa ra 1 giải pháp cho bài toán tìm n nếu chúng ta biết rằng $B(n) = 678570$.

Consider the table of Stirling numbers of 2nd kind $\left(\left\{ {s \atop t} \right\}\right)_{s,t=0}^{n}$ where $n \in \mathbb{N}$ is the size of the table of Stirling numbers of 2nd kind that we want to consider. The diagonal entries are always 1, & the numbers right below the main diagonal are the triangular numbers $\binom{s}{2}$. What can we say about the *sum* of the entries in each row? The (s,t) entry of the t able is $\left\{ {s \atop t} \right\}$, the number of partitions of $[s]$ into t nonempty parts. So the sum of every row is the *total* number of partitions of $[s]$. This is an interesting sequence of numbers, which is named as follows.

Question 4. *What can we say about the sum of the entries in each column of the table of Stirling numbers of 2nd kind $\left(\left\{ {s \atop t} \right\}\right)_{s,t=0}^{n}$ where $n \in \mathbb{N}$ is the size of the table of Stirling numbers of 2nd kind that we want to consider.*

Definition 9 (Bell number, [Sha22], Def. 6.12, p. 199). *Let $s \in \mathbb{N}$. The* Bell number $B(s)$ *is defined by*

$$B(s) \coloneqq \sum_{i=0}^{s} \left\{ {s \atop i} \right\}, \ \forall s \in \mathbb{N}. \tag{1.6}$$

I.e., $B(s)$ is the total number of ways of partitioning the set $[s]$.

Example 6 ([Sha22], Ex. 6.13, p. 200). *Multiply the matrix of the Stirling numbers by a column vector consisting of all ones, to find the 1st few Bell numbers:* $1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, \ldots$

Problem 31. *Write* C/C++, Pascal, Python *programs to multiple the matrix of Stirling numbers of the 2nd kind by a column vector consisting of all ones.*

Theorem 11 ([Sha22], Prop. 6.14, p. 200). *Let $f_0(x) = e^x$, define $f_n(x) \coloneqq x \frac{d}{dx} f_{n-1}(x) = x f'_{n-1}(x)$, $\forall n \in \mathbb{N}^\star$. Then*

$$f_n(x) = \left[\left\{ {n \atop 0} \right\} + \left\{ {n \atop 1} \right\} x + \cdots + \left\{ {n \atop n} \right\} x^n \right] e^x = e^x \sum_{i=0}^{n} \left\{ {n \atop i} \right\} x^i = \sum_{i=1}^{\infty} \frac{i^n x^i}{i!}, \ \forall n \in \mathbb{N}^\star.$$

In particular, $f_n(1) = e \sum_{i=0}^{n} \left\{ {n \atop k} \right\} = B(n)e$.

Theorem 12 ([Sha22], Prop. 6.15, p. 201).

$$B(n) = \frac{1}{e} \sum_{i=0}^{\infty} \frac{k^n}{k!}, \ \forall n \in \mathbb{N}. \tag{1.7}$$

In the case of $n = i = 0$, we make the stipulation that $0^0 = 1$ in this formula.

While we have an infinite series on RHS, we can approximate it by finding a partial sum, & use the partial sum & some estimate of the remaining terms to find $B(n)$, $\forall n \in \mathbb{N}^\star$, e.g., $B(6) \approx \frac{1}{e} \sum_{i=0}^{10} \frac{i^6}{i!} \approx 202.98$.

Bài toán 29. *Viết chương trình* C/C++, Pascal, Python *để xấp xỉ số Bell $B(n)$ tới d chữ số thập phân.*

Remark 9 (Approximate Bell numbers & compute their approximations in Maple, [Sha22], Rmk. 6.17, p. 202). *To approximate $B(n)$, you can use any mathematical software to find the partial sums of the series (1.7), i.e.:*

$$B(n) \approx B_{\text{appr}}(m, n) \coloneqq \frac{1}{e} \sum_{i=0}^{m} \frac{k^n}{k!}, \ \forall m, n \in \mathbb{N}.$$

Có $\lim_{m \to \infty} B_{\text{appr}}(m, n) = B(n)$, $\forall n \in \mathbb{N}^\star$. For small values of n, a symbolic algebra software e.g. Maple can actually give you exact answers. In Maple, you could try

```
> Bn:=n -> sum(k^{n}/k!, k=0..infinity)/exp(1);
> Bn(7);
```

to get $B(7)$ (in Maple, `exp(1)` *stands for e, & we 1st defined* `Bn` *as a function of n).*

1.16 Catalan Numbers – Số Catalan

Resources – Tài nguyên.

1. Wikipedia/Catalan number

The *Catalan numbers* are a sequence of natural numbers that occur in various counting problems, often involving recursionly defined objects. They are named after Eugène Catalan, though they were previously discovered in the 1730s by Minggatu.

– Số Catalan là 1 chuỗi số tự nhiên xuất hiện trong nhiều bài toán đếm khác nhau, thường liên quan đến các đối tượng được xác định đệ quy. Chúng được đặt theo tên của Eugène Catalan, mặc dù trước đó chúng đã được Minggatu phát hiện vào những năm 1730.

The n-th Catalan number can be expressed directly in terms of the central binomial coefficients by

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{n!(n+1)!}, \ \forall n \in \mathbb{N}^\star. \tag{1.8}$$

The 1st Catalan numbers for $n = 0, 1, 2, 3, \ldots$ are $1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$ (sequence A000108 in the OEIS).

1.16.1 Some properties of Catalan numbers – Vài tính chất của số Catalan

An alternative expression for C_n is

$$C_n = \binom{2n}{n} - \binom{2n}{n+1},\ \forall n \in \mathbb{N}^\star,$$

which is equivalent to the expression (1.8) because $\binom{2n}{n+1}\frac{n}{n+1}\binom{2n}{n}$. This expression shows that $C_n \in \mathbb{Z}$, which is not immediately obvious from the 1st formula (1.8) given. Another alternative expression is

$$C_n = \frac{1}{2n+1}\binom{2n+1}{n},\ \forall n \in \mathbb{N}^\star,$$

which can be directly interpreted in terms of the cycle lemma.

Problem 32. *Prove: (a) The Catalan numbers satisfy the recurrence relations*

$$C_0 = 1,\ C_n = \sum_{i=1}^{n} C_{i-1}C_{n-i},\ \forall n \in \mathbb{N}^\star, \tag{1.9}$$
$$C_0 = 1,\ C_n = \frac{2(2n-1)}{n+1}C_{n-1},\ \forall n \in \mathbb{N}^\star.$$

(b) Asymptotically, the Catalan numbers grow as

$$C_n \sim \frac{4^n}{n^{\frac{3}{2}}\sqrt{\pi}},\ \forall n \in \mathbb{N}^\star,$$

in the sense $\lim_{n\to\infty} \frac{C_n n^{\frac{3}{2}}\sqrt{\pi}}{4^n} = 1$.

Hint. (b) can be proved by using the asymptotic growth of the central binomial coefficients, by Stirlings' approximation for $n!$, or via generating functions.

The only Catalan numbers C_n that are odd are those for which $n = 2^k - 1$; all others are even. The only prime Catalan numbers are $C_2 = 2, C_3 = 5$. More generally, the multiplicity with which a prime p divides C_n can be determined by 1st expressing $n+1$ in base p. For $p = 2$, the multiplicity is the number of 1 bits, minus 1. For p an odd prime, count all digits $> \frac{p+1}{2}$; also count digits $= \frac{p+1}{2}$ unless final; & count digits $= \frac{p-1}{2}$ if not final & the next digit is counted. The only known odd Catalan numbers that do not have last digit 5 are $C_0 = 1, C_1 = 1, C_7 = 429, C_{31}, C_{127}, C_{255}$. The odd Catalan numbers, C_n for $n = 2^k - 1$, do not have last digit 5 if $n+1$ has a base 5 representation containing $0, 1, 2$ only, except in the least significant place, which could also be a 3.

The Catalan numbers have the integral representations

$$C_n = \frac{1}{2\pi}\int_0^4 x^n\sqrt{\frac{4-x}{x}}\,\mathrm{d}x = \frac{2}{\pi}4^n\int_{-1}^{1} t^{2n}\sqrt{1-t^2}\,\mathrm{d}t,$$

which immediately yields $\sum_{n=0}^{\infty}\frac{C_n}{4^n} = 2$.

This has a simple probabilistic interpretation. Consider a random walk on the integer line, starting at 0. Let -1 be a "trap" state, s.t. if the walker arrives at -1, it will remain there. The walker can arrive at the trap state at times $1, 3, 5, \ldots$, & the number of ways the walker can arrive at the trap state at time $2k+1$ is C_k. Since the 1D random walk is recurrent, the probability that the walker eventually arrives at -1 is $\sum_{n=0}^{\infty}\frac{C_n}{2^{2n+1}} = 1$.

– Điều này có một cách giải thích xác suất đơn giản. Hãy xem xét một bước đi ngẫu nhiên trên đường số nguyên, bắt đầu từ 0. Giả sử -1 là trạng thái "bẫy", tức là nếu người đi bộ đến -1, thì nó sẽ vẫn ở đó. Người đi bộ có thể đến trạng thái bẫy tại các thời điểm $1, 3, 5, \ldots$, & số cách người đi bộ có thể đến trạng thái bẫy tại thời điểm $2k+1$ là C_k. Vì bước đi ngẫu nhiên 1D là tuần hoàn, nên xác suất người đi bộ cuối cùng đến -1 là $\sum_{n=0}^{\infty}\frac{C_n}{2^{2n+1}} = 1$.

1.16.2 Some applications of Catalan numbers in Combinatorics – Vài ứng dụng của số Catalan trong Tổ hợp

There are many counting problems in combinatorics whos solution is given by the Catalan numbers. The book *Enumerative Combinatorics: Vol 2* by combinatorialist Richard P. Stanley contains a set of exercises which describe 66 different interpretations of the Catalan numbers, e.g.:

1. C_n is the number of Dyck words of length $2n$. A Dyck word is a string consisting of n X's & n Y's s.t. no initial segment of the string has more Y's than X's. E.g., the following are Dyck words up to length 6: XY, XXYY, XYXY, XXXYYY, XYXXYY, XYXYXY, XXYYXY, XXYXYY.

2. Re-interpreting the symbol X as an open parenthesis & Y as close parenthesis, C_n counts the number of expressions containing n pairs of parentheses which are correctly matched: ((())), (()()), (())(), ()(()), ()()().

1.16.3 Problems: Catalan numbers – Bài tập: Số Catalan

Bài toán 30. *Cho $n \in \mathbb{N}^\star$. Chứng minh số cách khác nhau để đặt n dấu ngoặc mở & n dấu ngoặc đóng đúng đắn bằng số Catalan thứ n:*

$$C_n := \frac{1}{n+1}C_{2n}^n = \frac{(2n)!}{n!(n+1)!},\ \forall n \in \mathbb{N}^\star.$$

VNTA's proof by using strong induction. Số Catalan thỏa hệ thức truy hồi (1.9). Gọi D_n là số chuỗi đúng đắn gồm n dấu ngoặc mở & n dấu ngoặc đóng. Cần chứng minh $D_n = C_n = \frac{1}{n+1}\binom{2n}{n}$, $\forall n \in \mathbb{N}^\star$. Khi $n = 0$, chỉ có 1 cách (chuỗi rỗng) nên $D_0 = C_0 = 1$. Giả sử $D_i = C_i = \frac{1}{i+1}\binom{2i}{i}$ đúng $\forall i \in [n]$. Cần chứng minh $D_{n+1} = C_{n+1}$. Thật vậy, mỗi chuỗi ngoặc đúng có thể viết thành dạng $(S_1)S_2$, trong đó: S_1 là chuỗi ngoặc đúng với i cặp ngoặc, S_2 là chuỗi ngoặc dúng với $n-i$ cặp ngoặc. Khi đó, tổng số cặp ngoặc trong chuỗi $(S_1)S_2$ là $i + (n-i) + 1 = n+1$ cặp ngoặc nên số chuỗi $n+1$ cặp ngoặc đúng có thể được biểu diễn thành $D_{n+1} = \sum_{i=0}^n D_i D_{n-i} = \sum_{i=0}^n C_i C_{n-i} = C_{n+1}$. Theo nguyên lý quy nạp mạnh, suy ra $D_n = C_n$, $\forall n \in \mathbb{N}^\star$. □

C++:

1. VNTA's C++: valid parentheses: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/C++/VNTA_valid_parentheses.cpp.

```cpp
#include<bits/stdc++.h>
using namespace std;

void validParentheses(int open, int close, string cur, vector<string>& res) {
    if (open == 0 && close == 0) {
        res.push_back(cur);
        return;
    }
    if (open > 0) validParentheses(open - 1, close, cur + '(', res);
    if (close > open) validParentheses(open, close - 1, cur + ')', res);
}

long long catalan(int n) {
    long long res = 1;
    for (int i = 0; i < n; i++) {
        res *= 2 * (2 * i + 1);
        res /= (i + 2);
    }
    return res;
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    int n;
    cin >> n;
    vector<string> valid;
    validParentheses(n, n, "", valid);
    cout << "So chuoi dau ngoac dung dan: " << valid.size() << '\n';
    for (const string& s : valid) cout << s << '\n';
    cout << "\nSo catalan thu n: " << catalan(n);
}
```

Bài toán 31. *Cho $n \in \mathbb{N}^\star$, $k \in \mathbb{N}$, $0 \le k \le n$. Viết chương trình* C/C++, Pascal, Python *để tính & cho biết giá trị nhỏ nhất của n gặp lỗi overflow: (a) $P_n = n!$. (b) A_n^k. (c) C_n^k. (d) Số Catalan thứ n.*

C++:

1. VNTA's: https://github.com/vntanh1406/Graph_SUM2025/blob/main/Combinatorics/CalculateP_A_C_Catalan.cpp.
   https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/C++/VNTA_calculate_P_A_C_Catalan.cpp.

```cpp
#include<bits/stdc++.h>
using namespace std;

// n_max = 20
long long Pn(int n) {
    long long res = 1;
    for (int i = 2; i <= n; i++) res *= i;
    return res;
}

// n_max = 20
long long nAk(int n, int k) {
    long long res = Pn(n) / Pn(n - k);
    return res;
}

// n_max = 20
long long nCk(int n, int k) {
    long long res = Pn(n) / (Pn(n - k) * Pn(k));
    return res;
}

// n_max = 10
long long catalan_n(int n) {
    long long res = Pn(2 * n) / (Pn(n) * Pn(n + 1));
    return res;
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    int n, k;
    cin >> n >> k;
    cout << Pn(n) << endl << nAk(n, k) << endl << nCk(n, k) << endl << catalan_n(n);
}
```

2. ST's C++: PAC Catalan: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/C++/ST_calculate_P_A_C_Catalan.cpp.

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

unsigned long long factorial(int n) {
    unsigned long long res = 1;
    for (int i = 2; i <= n; ++i) {
        if (res > ULLONG_MAX / i) {
            cout << "Overflow tại n = " << i << " khi tính giai thừa.\n";
            return 0;
        }
        res *= i;
    }
    return res;
}

```

```
17 unsigned long long permutation(int n, int k) {
18     unsigned long long a = factorial(n);
19     unsigned long long b = factorial(n - k);
20     return (b == 0) ? 0 : a / b;
21 }
22
23 unsigned long long combination(int n, int k) {
24     unsigned long long a = factorial(n);
25     unsigned long long b = factorial(k);
26     unsigned long long c = factorial(n - k);
27     return (b == 0 || c == 0) ? 0 : a / (b * c);
28 }
29
30 unsigned long long catalan(int n) {
31     unsigned long long a = factorial(2 * n);
32     unsigned long long b = factorial(n + 1);
33     unsigned long long c = factorial(n);
34     return (b == 0 || c == 0) ? 0 : a / (b * c);
35 }
36
37 int main() {
38     int n, k;
39     cout << "Nhap n (n > 0): ";
40     cin >> n;
41     cout << "Nhap k (0 <= k <= n): ";
42     cin >> k;
43
44     cout << "\n=== Ket qua ===\n";
45
46     unsigned long long fn = factorial(n);
47     if (fn != 0) cout << "Pn = " << fn << endl;
48
49     unsigned long long Ank = permutation(n, k);
50     if (Ank != 0) cout << "A^k_n = " << Ank << endl;
51
52     unsigned long long Cnk = combination(n, k);
53     if (Cnk != 0) cout << "C^k_n = " << Cnk << endl;
54
55     unsigned long long Catn = catalan(n);
56     if (Catn != 0) cout << "Catalan(n) = " << Catn << endl;
57     return 0;
58 }
```

Chương 2

Newton Binomial Theorem & Polynomials – Nhị Thức Newton & Đa Thức

Contents

2.1 Newton Binomial Theorem – Nhị thức Newton

Theorem 13 (Binomial theorem). *(i)* $(a+b)^n = \sum_{i=0}^{n} C_n^i a^{n-i}b^i$, $\forall a,b \in \mathbb{R}$, $\forall n \in \mathbb{N}$. *(ii)* $\sum_{i=0}^{n} C_n^i = 2^n$. *(iii)*

$$\sum_{i=0}^{n}(-1)^i C_n^i = 0 \Leftrightarrow \sum_{i=0,i:\text{even}}^{n} C_n^i = \sum_{i=0,i:\text{odd}}^{n} C_n^i \Leftrightarrow \begin{cases} C_{2n}^0 + C_{2n}^2 + \cdots + C_{2n}^{2n} = C_{2n}^1 + C_{2n}^3 + \cdots + C_{2n}^{2n-1}, \\ C_{2n+1}^0 + C_{2n+1}^2 + \cdots + C_{2n+1}^{2n} = C_{2n+1}^1 + C_{2n+1}^3 + \cdots + C_{2n+1}^{2n+1}. \end{cases}$$

Bài toán 32. *Khai triển: (a)* $(a+b+c)^n$, $\forall a,b,c \in \mathbb{R}$, $\forall n \in \mathbb{N}^\star$. *(b)* $(a+b+c+d)^n$, $\forall a,b,c,d \in \mathbb{R}$, $\forall n \in \mathbb{N}^\star$. *(c)* $\left(\sum_{i=1}^{m} a_i\right)^n$, $\forall m,n \in \mathbb{N}^\star$, $\forall a_i \in \mathbb{R}$, $\forall i = 1,\ldots,m$. *(d)* z^n, $\forall z \in \mathbb{C}$, $\forall n \in \mathbb{N}^\star$ *với* $z = a+bi$, $a := \Re z$, $b := \Im z$.

Chứng minh. Add PVT's proofs+++ □

C++:

1. VNTA's C++: Pascal triangle & multinomial:

- https://github.com/vntanh1406/Graph_SUM2025/blob/main/Combinatorics/PascalTriaAndMultinomial.cpp.
- https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/C%2B%2B/VNTA_Pascal_triangle_multinomia cpp.

```
#include<bits/stdc++.h>
using namespace std;

unsigned long long Pn(int n) {
    if (n == 0) return 1;
    unsigned long long res = 1;
    for (int i = 2; i <= n; i++)res *= i;
    return res;
}

unsigned long long nCk(int n, int k) {
    long long res = Pn(n) / (Pn(n - k) * Pn(k));
    return res;
}

```

```
16 void pascalTriangle(int n) {
17     cout << "1st " << n + 1 << " lines of the Pascal triangle: \n";
18     for (int i = 0; i <= n; i++) {
19         for (int j = 0; j <= i; j++) {
20             cout << nCk(i, j) << " ";
21         }
22         cout << '\n';
23     }
24     cout << "\n=======================================\n";
25 }
26
27 void generatePartitions(int n, int m, int idx, vector<int>& cur, vector<vector<int>>& res) {
28     if (idx == m - 1) {
29         cur[idx] = n;
30         res.push_back(cur);
31         return;
32     }
33     for (int i = 0; i <= n; i++) {
34         cur[idx] = i;
35         generatePartitions(n - i, m, idx + 1, cur, res);
36     }
37 }
38
39 double multinomialExpansion(int n, const vector<double>& a) {
40     int m = a.size();
41     vector<vector<int>> partitions;
42     vector<int> cur(m, 0);
43     generatePartitions(n, m, 0, cur, partitions);
44
45     double res = 0;
46     for (const auto& k : partitions) {
47         unsigned long long coeff = Pn(n);
48         double term = 1;
49         for (int i = 0; i < m; i++) {
50             coeff /= Pn(k[i]);
51             term *= pow(a[i], k[i]);
52         }
53         res += coeff * term;
54     }
55     return res;
56 }
57
58 int main() {
59     ios_base::sync_with_stdio(0);
60     cin.tie(0); cout.tie(0);
61     int n, m;
62     cin >> n >> m;
63
64     pascalTriangle(n);
65
66     vector<double> a(m);
67     for (int i = 0; i < m; i++) cin >> a[i];
68     double result = multinomialExpansion(n, a);
69     if (m > 2) cout << "(a1 +... + a" << m << ")^" << n << " = " << result << "\n";
70     else cout <<  "(a1 + a2)^" << n << " = " << result << "\n";
71 }
```

2.2 Combinatorial identities – Đẳng thức tổ hợp

Các phương pháp chứng minh đẳng thức tổ hợp:

- Sử dụng phương pháp quy nạp toán học.
- Bắt đầu từ 1 đẳng thức tổ hợp, e.g., 1 khai triển của 1 biểu thức, sau đó lấy đạo hàm 2 vế (có thể lấy đạo hàm nhiều lần nếu cần thiết).
- Bắt đầu từ 1 đẳng thức tổ hợp, e.g., 1 khai triển của 1 biểu thức, sau đó lấy tích phân 2 vế (có thể lấy tích phân bội nếu cần thiết).
- Lý luận tổ hợp: Đếm bằng 2 cách (giống nguyên lý Fubini cho đổi cận của tích phân 2 hay nhiều lớp, see, e.g., Huỳnh Quang Vũ, *Bài Giảng Giải Tích 3: Tích Phân Đường, Tích Phân Mặt, Tích Phân Bội*).

2.2.1 Pascal's rule – Quy tắc Pascal

In mathematics, *Pascal's rule* (or *Pascal's formula*) is a combinatorial identity about binomial coefficients. The binomial coefficients are the numbers that appear in Pascal's triangle.

Theorem 14 (Pascal's rule)**.** *One has*

$$C_{n-1}^{k}+C_{n-1}^{k-1}=C_{n}^{k},\ \textit{i.e.,}\ \binom{n-1}{k}+\binom{n-1}{k-1}=\binom{n}{k},\ \forall n,k\in\mathbb{N}^{\star}, \tag{Pasr}$$

where $\binom{n}{k}$ *is the binomial coefficient, namely the coefficient of the* x^k *term in the expansion of* $(1+x)^n$*. There is no restriction on the relative sizes of* n,k*; in particular,* (Pasr) *remains valid when* $n<k$ *since* $\binom{n}{k}=0$ *whenever* $n<k$*.*

Together with the boundary conditions $\binom{n}{0}=\binom{n}{n}=1$, $\forall n\in\mathbb{N}$, Pascal's rule determines that

$$\binom{n}{k}=\frac{n!}{k!(n-k)!},\ \forall n,k\in\mathbb{N},\ 0\le k\le n.$$

In this sense, Pascal's rule is the recurrence relation that defines the binomial coefficients.

Problem 33. *Prove Pascal's rule* (Pasr)*.*

For a combinatorial- & an algebraic proofs of Pascal's rule, see, e.g., Wikipedia/Pascal's rule.

A combinatorial proof of Pascal's rule when $n\in\mathbb{N}^{\star}$*.* $\binom{n}{k}$ equals the number of subsets with k elements from a set with n elements. Suppose 1 particular element is uniquely labeled X in a set with n elements. To construct a subset of k elements containing X, include X & choose $k-1$ elements from the remaining $n-1$ elements in the set. There are $\binom{n-1}{k-1}$ such subsets. To construct a subset of k elements not containing X, choose k elements from the remaining $n-1$ elements in the set. There are $\binom{n-1}{k}$ such subsets. Every subset of k elements either contains X or not. The total number of subsets with k elements in a set of n elements is the sum of the number of subsets containing X & the number of subsets that do not contain X, $\binom{n-1}{k-1}+\binom{n-1}{k}$. This equals $\binom{n}{k}$, hence (Pasr) holds. □

1st algebraic proof of Pascal's rule when $n\in\mathbb{N}^{\star}$*.*

$$\begin{aligned}\binom{n-1}{k}+\binom{n-1}{k-1}&=\frac{(n-1)!}{k!(n-1-k)!}+\frac{(n-1)!}{(k-1)!(n-k)!}=(n-1)!\left(\frac{1}{k!(n-1-k)!}+\frac{1}{(k-1)!(n-k)!}\right)\\&=(n-1)!\left(\frac{n-k}{k!(n-k)!}+\frac{k}{n!(n-k)!}\right)=(n-1)!\frac{n}{k!(n-k)!}=\frac{n!}{k!(n-k)!}=\binom{n}{k}.\end{aligned}$$

□

2nd algebraic proof of Pascal's rule when $n\in\mathbb{C}$*.* An alternative algebraic proof using the alternative definition of binomial coefficients $\binom{n}{k}=\frac{n(n-1)\cdots(n-k+1)}{k!}$, which is used as the extended definition of the binomial coefficient when $n\in\mathbb{C}$, hence (Pasr) holds more generally for $n\in\mathbb{C}$. □

Pascal's rule can be generalized to multinomial coefficients.

Theorem 15 (Generalized Pascal's rule). *For any $p \in \mathbb{N}$, $p \geq 2$, $k_1, \ldots, k_p \in \mathbb{N}$, $n = \sum_{i=1}^{p} k_i \geq 1$,*

$$\binom{n-1}{k_1-1, k_2, k_3, \ldots, k_p} + \binom{n-1}{k_1, k_2-1, k_3, \ldots, k_p} + \cdots + \binom{n-1}{k_1, k_2, k_3, \ldots, k_p-1} = \binom{n-1}{k_1, k_2, k_3, \ldots, k_p}, \tag{gPasr}$$

where $\binom{n}{k_1,k_2,\ldots,k_p}$ is the coefficient of the $\prod_{i=1}^{p} x_i^{k_i}$ term in the expansion of $\left(\sum_{i=1}^{p} x_i\right)^n$.

Problem 34. *Prove generalized Pascal's rule (gPasr) in both combinatorial & algebraic ways.*

Bài toán 33 ([Phư10], VD2, p. 54). *Chứng minh $\sum_{i=1}^{n} iC_n^i = n2^{n-1}$, $\forall n \in \mathbb{N}^\star$ bằng 4 cách: (a) Sử dụng phương pháp quy nạp toán học. (b) Biến đổi số hạng tổng quát nhờ đẳng thức Pascal. (c) Xét khai triển $(1+x)^n$ rồi lấy đạo hàm 2 vế. (d) Lý luận tổ hợp.*

Bài toán 34 ([Phư10], VD3, p. 54). *Chứng minh $\sum_{i=0}^{n} \frac{C_n^i}{i+1} = \frac{2^{n+1}-1}{n+1}$, $\forall n \in \mathbb{N}^\star$ bằng 4 cách: (a) Sử dụng phương pháp quy nạp toán học. (b) Biến đổi số hạng tổng quát nhờ đẳng thức Pascal. (c) Xét khai triển $(1+x)^n$ rồi lấy tích phân $\int_0^1$ 2 vế. (d) Lý luận tổ hợp.*

Bài toán 35 ([Phư10], VD4, p. 55). *Chứng minh $\sum_{i=0}^{n} (C_n^i)^2 = C_{2n}^n$, $\forall n \in \mathbb{N}^\star$ bằng 2 cách: (a) Tính hệ số của x^n trong khai triển của nhị thức Newton của $(x+1)^{2n} = (x+1)^n(1+x)^n$ bằng 2 cách. (d) Lý luận tổ hợp.*

Bài toán 36 ([Phư10], VD5, p. 55, ĐH-A 2006). *(a) Tìm hệ số của số hạng chứa x^{26} trong khai triển nhị thức Newton của $\left(\frac{1}{x^4} + x^7\right)^n$ biết $\sum_{i=1}^{n} C_{2n+1}^i = 2^{20} - 1$. (b) Tìm hệ số của số hạng chứa x^k trong khai triển nhị thức Newton của $(x^a + x^n)^n$ biết $\sum_{i=1}^{n} C_{2n+1}^i = 2^{n_0} - 1$ với $n_0 \in \mathbb{N}^\star$, $a, b \in \mathbb{R}$.*

Chương 3

Phân Vùng Số Nguyên & Nguyên Tắc Loại Suy

Chương 4

Generating Functions – Hàm Sinh

Contents

4.1 Basic Generating functions – Các Hàm Sinh Cơ Bản

Resources – Tài nguyên.

1. Wikipedia/generating function.

2. MathScope. *Chuyên đề Tổ Hợp.*

3. [Vin+25]. Lê Anh Vinh, Lê Phúc Lữ, Nguyễn Huy Tùng, Trần Đăng Phúc, Vũ Văn Luân, Phạm Văn Thắng, Lê Quang Quân, Nguyễn Văn Thế, Nguyễn Tuấn Hải Đăng, Hà Hữu Cao Trình, Phạm Việt Hùng. *Định Hướng Bồi Dưỡng Học Sinh Năng Khiếu Toán. Tập 4: Tổ Hợp.* Chap. 13: Hàm sinh & ứng dụng.

In mathematics, a *generating function* is a representation of an infinite sequence of numbers as the coefficients of the formal power series. Generating functions are often expressed in closed form (rather than as a series), by some expression involving operations on the formal series.

– Trong toán học, hàm sinh là biểu diễn của 1 chuỗi số vô hạn dưới dạng hệ số của 1 chuỗi lũy thừa chính thức. Các hàm sinh thường được biểu diễn ở dạng đóng (thay vì dạng chuỗi), bằng 1 số biểu thức liên quan đến các phép toán trên chuỗi chính thức.

There are various types of generating functions, including *ordinary generating functions, exponential generating functions, Lambert series, Bell series, & Dirichlet series.* Every sequence in principle has a generating function of each type (except that Lambert & Dirichlet series require indices to start at 1 rather than 0), but the ease with which they can be handled may differ considerably. The particular generating function, if any, that is most useful in a given context will depend upon the nature of the sequence & the details of the problem being addressed.

– Có nhiều loại hàm sinh, bao gồm hàm sinh thông thường, hàm sinh mũ, chuỗi Lambert, chuỗi Bell & chuỗi Dirichlet. Về nguyên tắc, mỗi chuỗi đều có 1 hàm sinh của từng loại (trừ chuỗi Lambert và chuỗi Dirichlet yêu cầu chỉ số bắt đầu từ 1 thay vì 0), nhưng mức độ dễ dàng xử lý chúng có thể khác nhau đáng kể. Hàm sinh cụ thể, nếu có, hữu ích nhất trong 1 bối cảnh nhất định sẽ phụ thuộc vào bản chất của chuỗi và các chi tiết của vấn đề đang được giải quyết.

Generating functions are sometimes called *generating series*, in that a series of terms can be said to be the *generator* of its sequence of term coefficients.

– Các hàm sinh đôi khi được gọi là *tạo chuỗi*, theo đó 1 chuỗi các số hạng có thể được coi là *tạo chuỗi* của chuỗi các hệ số số hạng của nó.

History of generating function. Generating functions were 1st introduced by Abraham de Moivre in 1730, in order to solve the general linear recurrence problem.

> "The name "generating function" is due to Laplace. Yet, without giving it a name, Euler used the device of generating functions long before Laplace [...]. He applied this mathematical tool to several problems in Combinatory Analysis & the Theory of Numbers." – George Pólya, *Mathematics & Plausible Reasoning* (1954)
>
> "A generating function is a device somewhat similar to a bag. Instead of carrying many little objects detachedly, which could be embarrassing, we put them all in a bag, & then we have only 1 object to carry, the bag." – George Pólya, *Mathematics & Plausible Reasoning* (1954)
>
> "A generating function is a clothesline on which we hang up a sequence of numbers for display." – Herbert Wilf, Generatingfunctionology (1994)

Convergence. Unlike an ordinary series, the *formal* power series is not required to converge: in fact, the generating function is not actually regarded as a function, & the "variable" remains an indeterminate. One can generalize to formal power series in > 1 indeterminate, to encode information about infinite multi-dimensional arrays of numbers. Thus generating functions are not functions in the formal sense of a mapping from a domain to a codomain.

– Không giống như 1 chuỗi thông thường, chuỗi lũy thừa chính thức không bắt buộc phải hội tụ: trên thực tế, hàm sinh không thực sự được coi là 1 hàm, và "biến" vẫn là 1 hàm bất định. Người ta có thể khái quát hóa thành chuỗi lũy thừa chính thức trong nhiều hơn 1 hàm bất định, để mã hóa thông tin về các mảng số đa chiều vô hạn. Do đó, các hàm sinh không phải là các hàm theo nghĩa chính thức của phép ánh xạ từ 1 miền tới 1 miền đồng dạng.

These expressions in terms of the indeterminate x may involve arithmetic operations, differentiation w.r.t. x & composition with (i.e., substitution into) other generating functions; since these operations are also defined for functions, the result looks like a function of x. Indeed, the closed form expression can often be interpreted as a function that can be evaluated at (sufficiently small) concrete values of x, & which has the formal series as its series expansion; this explains the designation "generating functions". However such interpretation is not required to be possible, because formal series are not required to give a convergent series when a nonzero numeric value is substituted for x.

– Những biểu thức này theo x bất định có thể bao gồm các phép toán số học, phép tính vi phân đối với x & phép hợp với (tức là phép thế vào) các hàm sinh khác; vì các phép toán này cũng được định nghĩa cho các hàm, nên kết quả trông giống như 1 hàm của x. Thật vậy, biểu thức dạng đóng thường có thể được diễn giải như 1 hàm có thể được đánh giá tại các giá trị cụ thể (đủ nhỏ) của x, & có chuỗi chính thức là phép khai triển chuỗi của nó; điều này giải thích cho tên gọi "hàm sinh". Tuy nhiên, không nhất thiết phải có cách diễn giải như vậy, vì chuỗi chính thức không bắt buộc phải đưa ra 1 chuỗi hội tụ khi 1 giá trị số khác không được thay thế cho x.

Limitations of generating function. Not all expressions that are meaningful as functions of x are meaningful as expressions designating formal series, e.g., negative & fractional powers of x are examples of functions that do not have a corresponding formal power series.

– *Hạn chế của hàm sinh.* Không phải tất cả các biểu thức có ý nghĩa như hàm của x đều có ý nghĩa như các biểu thức chỉ định chuỗi chính thức, ví dụ, lũy thừa phân số & âm của x là ví dụ về các hàm không có chuỗi lũy thừa chính thức tương ứng.

Idea of generating functions. If we want to find a formula for some function $f(n)$ with $n \in \mathbb{N}$, then we can form the generating function of $f(n)$:

$$F(x) := \sum_{n \geq 0} f(n)x^n = f(0) + f(1)x + f(2)x^2 + \cdots + f(n)x^n + \cdots, \tag{genf}$$

so that a_n is the coefficient of x^n in the Taylor series expansion of $F(x)$ defined by (genf).

Here we are concerned with formal power series, & questions of convergence do not come up (at least for elementary applications). Sometimes this power series has a nice closed form & then we can manipulate this function & get information about $f(n)$, $\forall n \in \mathbb{N}$.

– Ở đây chúng ta quan tâm đến chuỗi lũy thừa chính thức, & các câu hỏi về sự hội tụ không xuất hiện (ít nhất là đối với các ứng dụng cơ bản). Đôi khi chuỗi lũy thừa này có dạng đóng đẹp & sau đó chúng ta có thể thao tác hàm này & lấy thông tin về $f(n)$, $\forall n \in \mathbb{N}$.

4.2 Types of generating functions – Các loại hàm sinh

4.2.1 Ordinary generating function (OGF) – Hàm sinh thường

When the term *generating function* is used without quantification, it is usually taken to mean an ordinary generating function.

Definition 10 (Ordinary generating function (OGF)). *The* ordinary generating function *of a sequence* $\{a_n\}_{n=0}^{\infty} \subset \mathbb{C}$ *of complex numbers is defined by*

$$G(\{a_n\}_{n=0}^{\infty}; x) \coloneqq \sum_{n=0}^{\infty} a_n x^n.$$

If a_n *is the probability mass function of a discrete random variable, then its ordinary generating function is called a probability-generating function.*

Bài toán 37 (Probability-generating function). *Make clear the concept of probability-generating function – Làm rõ khái niệm hàm sinh xác suất.*

Định lý 3 (Some basic properties of generating function – Vài tính chất cơ bản của hàm sinh). *Cho* $G(a,x), G(b,x)$ *là hàm sinh tương ứng của 2 dãy số* $a = \{a_n\}_{n=0}^{\infty}, b = \{b_n\}_{n=0}^{\infty}$.

(i) Nếu tồn tại $k \in \mathbb{N}$ *mà* $a_i = 0$, $\forall i \in \mathbb{N}$, $i < k$, *&* $b_n = a_{n+k}$, $\forall n \in \mathbb{N}$, *thì* $G(a,x) = x^k G(b,x)$.

(ii) Nếu $b_n = \sum_{i=0}^{n} a_i$, $n \in \mathbb{N}$, *thì* $G(b,x) = \dfrac{G(a,x)}{1-x}$.

Định nghĩa 11 (Generalized binomial coefficient – Hệ số nhị thức mở rộng). *Với* $x \in \mathbb{R}, k \in \mathbb{N}$, *hệ số nhị thức mở rộng* $\binom{x}{k}$ *được định nghĩa bởi*

$$\binom{x}{k} \coloneqq \begin{cases} \dfrac{x(x-1)\cdots(x-k+1)}{k!} & \text{if } k \in \mathbb{N}^{\star}, \\ 1 & \text{if } k = 0. \end{cases}$$

4.2.2 Exponential generating function (EGF)

Definition 11 (Exponential generating function (EGF)). *The* exponential generating function *of a sequence* $\{a_n\}_{n=0}^{\infty} \subset \mathbb{C}$ *of complex numbers is defined by*

$$\mathrm{EG}(\{a_n\}_{n=0}^{\infty}; x) = \sum_{n=0}^{\infty} a_n \frac{x^n}{n!}.$$

Advantages of EGFs vs. OGFs. Exponential generating functions are generally more convenient than ordinary generating functions for combinatorial enumeration problems that involve labeled objects.

– Các hàm sinh mũ thường thuận tiện hơn các hàm sinh thông thường đối với các bài toán liệt kê tổ hợp liên quan đến các đối tượng có nhãn.

Another benefit of exponential generating functions is that they are useful in transferring linear recurrence relations to the realm of differential equations.

a functionality of EFGs : linear recurrence relations $\mapsto$ differential equations.

Example 7. *Take the Fibonacci sequence* $\{F_n\}_{n=0}^{\infty}$ *satisfying the linear recurrence relation* $F_{n+2} = F_{n+1} + F_n$. *The corresponding exponential generating function has the form*

$$\mathrm{EG}_{\mathrm{Fibonacci}}(x) \coloneqq \mathrm{EG}(\{F_n\}_{n=0}^{\infty}; x) = \sum_{n=0}^{\infty} \frac{F_n}{n!} x^n,$$

& its derivatives can readily be shown to satisfy the differential equation

$$\mathrm{EG}''_{\mathrm{Fibonacci}}(x) = \mathrm{EG}'_{\mathrm{Fibonacci}}(x) + \mathrm{EG}_{\mathrm{Fibonacci}}(x)$$

as a direct analogue with the recurrence relation above. In this view, the factorial term $n!$ *is merely a counter-term to normalize the derivative operator acting on* x^n.

Bài toán 38 (Exponential generating functions of Fibonacci sequence). *Chứng minh* $\mathrm{EG}''_{\mathrm{Fibonacci}}(x) = \mathrm{EG}'_{\mathrm{Fibonacci}}(x) + \mathrm{EG}_{\mathrm{Fibonacci}}(x)$.

4.2.3 Poisson generating function – Hàm sinh Poisson

Definition 12 (Poisson generating function). *The* Poisson generating function *of a sequence* $\{a_n\}_{n=0}^{\infty} \subset \mathbb{C}$ *of complex numbers is defined by*

$$\mathrm{PG}(\{a_n\}_{n=0}^{\infty}; x) = \sum_{n=0}^{\infty} a_n e^{-x} \frac{x^n}{n!} = e^{-x} \mathrm{EG}(\{a_n\}_{n=0}^{\infty}; x). \tag{4.1}$$

4.2.4 Lambert series

Resources – Tài nguyên.

1. Wikipedia/Lambert series.

Definition 13 (Lambert series)**.** *The* Lambert series *of a sequence* $\{a_n\}_{n=0}^{\infty} \subset \mathbb{C}$ *of complex numbers is defined by*

$$\mathrm{LG}(\{a_n\}_{n=0}^{\infty}; x) = \sum_{n=1}^{\infty} a_n \frac{x^n}{1-x^n}. \tag{4.2}$$

Note that in a Lambert series the index n starts at 1, not at 0, as the 1st term would otherwise be undefined.
The Lambert series coefficients in the power series expansions

$$b_n \coloneqq [x^n]\mathrm{LG}(\{a_n\}_{n=0}^{\infty}; x),\ \forall n \in \mathbb{N}^{\star},$$

are related by the divisor sum

$$b_n = \sum_{d|n} a_d.$$

Theorem 16. *For* $x \in \mathbb{R}$, $|x| < 1, |xq| < 1$,

$$\sum_{n=1}^{\infty} \frac{q^n x^n}{1-x^n} = \sum_{n=1}^{\infty} \frac{q^n x^{n^2}}{1-qx^n} + \sum_{n=1}^{\infty} \frac{q^n x^{n(n+1)}}{1-x^n}.$$

where we have the special case identity for the generating function fo the divisor function,

$$\sum_{n=1}^{\infty} \frac{x^n}{1-x^n} = \sum_{n=1}^{\infty} \frac{x^{n^2}(1+x^n)}{1-x^n}.$$

4.2.5 Bell series

Definition 14. *The Bell series of a sequence* $\{a_n\}_{n=1}^{\infty} \subset \mathbb{C}$ *is an expression in terms of both an intermediate* x *& a prime* p *& is given by*

$$\mathrm{BG}_p(\{a_n\}_{n=1}^{\infty}; x) = \sum_{n=0}^{\infty} a_{p^n} x^n.$$

4.2.6 Dirichlet series generating functions (DGFs)

Formal Dirichlet series are often classified as generating functions, although they are not strictly formal power series.

Definition 15 (Dirichlet series generating functions (DGFs))**.** *The* Dirichlet series generating functions (DGFs) *of a sequence* $\{a_n\}_{n=0}^{\infty} \subset \mathbb{C}$ *is*

$$\mathrm{DG}(\{a_n\}_{n=0}^{\infty}; x) \coloneqq \sum_{n=1}^{\infty} \frac{a_n}{n^s}.$$

The Dirichlet series generating function is especially useful when a_n is a multiplicative function, in which case it has an Euler product expression in terms of the function's Bell series:

$$\mathrm{DG}(\{a_n\}_{n=0}^{\infty}; x) = \prod_p \mathrm{BG}_n(\{a_n\}_{n=0}^{\infty}; p^{-s}).$$

4.2.7 Polynomial sequence generating functions

The idea of generating functions can be extended to sequences of other objects, e.g., polynomials:

Definition 16 (Polynomial sequence generating functions)**.** Polynomial sequences *of binomial type are generated by*

$$e^{xf(t)} = \sum_{n=0}^{\infty} \frac{p_n(x)}{n!} t^n,$$

where $\{p_n(x)\}_{n=0}^{\infty}$ *is a sequence of polynomials & $f(t)$ is a function of a certain form.*

Sheffer sequences are generated in a similar way. For more information, see, e.g., Wikipedia/generalized Appell polynomials.

Example 8. *Examples of polynomial sequences generated by more complex generating function include: Appell polynomials, Chebyshev polynomials, difference polynomials, generalized Appel polynomials, q-difference polynomials.*

4.2.8 Other generating functions

Other sequences generated by more complex generating functions include:

- Double exponential generating functions, e.g., the Bell numbers
- Hadamard products of generating functions & diagonal generating functions, & their corresponding integral transformations.

4.3 Problem: Generating functions

Problem 35 ([Sha22], p. 4)**.** *Let* $\{a_n\}_{n=0}^{\infty}$ *be a sequence defined by*

$$a_0 = 1,\ \sum_{i=0}^{n} a_i a_{n-i} = 1,\ \forall n \in \mathbb{N}^\star.$$

(a) Let $F(x) \coloneqq \sum_{i=0}^{\infty} a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots$ *be the generating function of* $\{a_n\}_{n=0}^{\infty}$*. Prove:* $F(x)F(x) = \dfrac{1}{1-x}$*, thus* $F(x) = \dfrac{1}{\sqrt{1-x}}$*. (b) Prove:* $a_n = \dfrac{(2n-1)!!}{2^n n!} = \dfrac{1 \cdot 3 \cdot 5 \cdot (2n-1)}{2^n n!}$, $\forall n \in \mathbb{N}^\star$.

Bài toán 39. *(a) Đếm số cách chọn ra* 15 *$ từ* 20 *người nếu* 19 *người đầu, mỗi người có thể đưa ra nhiều nhất* 1 *$, người thứ* 20 *có thể đưa ra* 1 *$,* 5 *$, hoặc không $ nào. (b) Đếm số cách chọn ra* $m \in \mathbb{N}^\star$ *$ từ* $n \in \mathbb{N}^\star$ *người nếu* $k \in \mathbb{N}$ *người đầu, mỗi người có thể đưa ra nhiều nhất* a *$,* $n - k \in \mathbb{N}$ *người sau có thể đưa ra* $b_1, b_2, \ldots,$ *hoặc* b_l *$ với* $b_i \in \mathbb{N}$, $\forall i \in [l]$, $b_i \neq b_j$, $\forall i, j \in [l]$, $i \neq j$.

Bài toán 40. *Dùng hàm sinh, giải phương trình nghiệm nguyên*

$$\sum_{i=1}^{m} x_i = n \text{ s.t. } m_i \leq x_i \leq M_i \text{ where } m_i, M_i \in \mathbb{Z},\ m_i \leq M_i,\ \forall i \in [m].$$

Phần II

Graph Theory – Lý Thuyết Đồ Thị

Chương 5

Basic Graph Theory – Lý Thuyết Đồ Thị Cơ Bản

5.1 Preliminaries

5.1.1 Search problems – Bài toán tìm kiếm

Resources – Tài nguyên.

1. Wikipedia/search problem.

In computational complexity theory & computability theory, a *search problem* is a computational problem of finding an *admissible* answer for a given input value, provided that such an answer exists. In fact, a search problem is specified by a binary relation R where xRy iff "y is an admissible answer given x". Search problems frequently occur in graph theory & combinatorial optimization, e.g. searching for matchings, optional cliques, & stable sets in a given undirected graph.

– Trong lý thuyết độ phức tạp tính toán và lý thuyết khả năng tính toán, 1 bài toán tìm kiếm là 1 bài toán tính toán tìm 1 câu trả lời có thể chấp nhận được cho 1 giá trị đầu vào nhất định, với điều kiện là câu trả lời như vậy tồn tại. Trên thực tế, 1 bài toán tìm kiếm được chỉ định bởi 1 quan hệ nhị phân R trong đó xRy nếu và chỉ nếu "y là 1 câu trả lời có thể chấp nhận được cho x". Các bài toán tìm kiếm thường xảy ra trong lý thuyết đồ thị và tối ưu hóa tổ hợp, ví dụ như tìm kiếm các phép khớp, các nhóm tùy chọn và các tập ổn định trong 1 đồ thị vô hướng nhất định.

An algorithm is said to solve a search problem if, for every input value x, it returns an admissible answer y for x when such an answer exists; otherwise, it returns any appropriate output, e.g., "not found" for x with no such answer.

Definition 17 (Search problem)**.** *If R is a binary relation s.t.* field$(R) \subseteq \Gamma^+$ *& T is a Turing machine, then T calculates f if:*

- *if x is s.t. there is some y s.t. $R(x, y)$ then T accepts x with output z s.t. $R(x, z)$ (there may be multiple y, & T need only find 1 of them).*
- *If x is s.t. there is no y s.t. $R(x, y)$ then T rejects x.*

The graph of a partial function is a binary relation, & if T calculates a partial function then there is at most 1 possible output.

A R can be viewed as a *search problem*, & a Turing machine which calculates R is also said to solve it. Every search problem has a corresponding decision problem, namely $L(R) = \{x; \exists y,\ R(x, y)\}$. This definition can be generalized to n-ary relations by any suitable encoding which allows multiple strings to be compressed into 1 string (e.g., by listing them consecutively with a delimiter).

5.1.2 Search algorithms – Các thuật toán tìm kiếm

Resources – Tài nguyên.

1. Wikipedia/search algorithm.

In CS, a *search algorithm* is an algorithm designed to solve a search problem. Search algorithms work to retrieve information stored within particular data structure, or calculated in the search space/feasible region of a problem domain, with either discrete or continuous values.

– Trong khoa học máy tính, thuật toán tìm kiếm là thuật toán được thiết kế để giải quyết vấn đề tìm kiếm. Thuật toán tìm kiếm hoạt động để truy xuất thông tin được lưu trữ trong cấu trúc dữ liệu cụ thể hoặc được tính toán trong không gian tìm kiếm của miền vấn đề, với các giá trị rời rạc hoặc liên tục.

Although search engines use search algorithms, they belong to the study of information retrieval, not algorithmics.

– Mặc dù công cụ tìm kiếm sử dụng thuật toán tìm kiếm nhưng chúng thuộc về lĩnh vực nghiên cứu về truy xuất thông tin chứ không phải thuật toán.

The appropriate search algorithm to use often depends on the data structure being searched, & may also include prior knowledge about the data. Search algorithms can be made faster or more efficient by specially constructed database structures, e.g. search trees, hash maps, & database indexes.

– Thuật toán tìm kiếm phù hợp để sử dụng thường phụ thuộc vào cấu trúc dữ liệu đang được tìm kiếm và cũng có thể bao gồm kiến thức trước đó về dữ liệu. Thuật toán tìm kiếm có thể được thực hiện nhanh hơn hoặc hiệu quả hơn bằng các cấu trúc cơ sở dữ liệu được xây dựng đặc biệt, chẳng hạn như cây tìm kiếm, bản đồ băm và chỉ mục cơ sở dữ liệu.

Search algorithms can be classified based on their mechanism of searching into 3 types of algorithms: linear, binary, & hashing. Linear search algorithms check every record for the one associated with a target key in a linear fashion. Binary, or half-interval, searches repeatedly target the center of the search structure & divide the search space in half. Comparison search algorithms improve on linear searching by successively eliminating records based on comparisons of the keys until the target record is found, & can be applied on data structures with a defined order. Digital search algorithms work based on the properties of digits in data structures by using numerical keys. Finally, hashing directly maps keys to records based on a hash function.

– Thuật toán tìm kiếm có thể được phân loại dựa trên cơ chế tìm kiếm của chúng thành ba loại thuật toán: tuyến tính, nhị phân, & băm. Thuật toán tìm kiếm tuyến tính kiểm tra mọi bản ghi để tìm bản ghi được liên kết với khóa mục tiêu theo cách tuyến tính. Tìm kiếm nhị phân hoặc nửa khoảng, liên tục nhắm mục tiêu vào tâm của cấu trúc tìm kiếm và chia đôi không gian tìm kiếm. Thuật toán tìm kiếm so sánh cải thiện tìm kiếm tuyến tính bằng cách loại bỏ liên tiếp các bản ghi dựa trên các phép so sánh khóa cho đến khi tìm thấy bản ghi mục tiêu và có thể được áp dụng trên các cấu trúc dữ liệu có thứ tự được xác định. Thuật toán tìm kiếm kỹ thuật số hoạt động dựa trên các thuộc tính của chữ số trong cấu trúc dữ liệu bằng cách sử dụng các khóa số. Cuối cùng, băm trực tiếp ánh xạ khóa thành các bản ghi dựa trên hàm băm.

Algorithms are often evaluated by their computational complexity, or maximum theoretical run time. Binary search functions, e.g., have a maximum complexity of $O(\log n)$, or logarithmic time. In simple terms, the maximum number of operations needed to find the search target is a logarithmic function of the size of the search space.

– Thuật toán thường được đánh giá theo độ phức tạp tính toán hoặc thời gian chạy lý thuyết tối đa. E.g., hàm tìm kiếm nhị phân có độ phức tạp tối đa là $O(\log n)$ hoặc thời gian logarith. Nói 1 cách đơn giản, số lượng thao tác tối đa cần thiết để tìm mục tiêu tìm kiếm là hàm logarit của kích thước không gian tìm kiếm.

5.1.2.1 Applications of search algorithms – Ứng dụng của thuật toán tìm kiếm

Specific applications of search algorithms include:

- Problems in combinatorial optimization, e.g.:
  - The vehicle routing problem, a form of shortest path problem
  - The knapsack problem: Given a set of items, each with a weight & a value, determine the number of each item to include in a collection so that the total weight is $\leq$ a given limit & the total value is as large as possible.
  - The nurse scheduling problem
- Problems in constraint satisfaction, e.g.:
  - The map coloring problem
  - Filling in a sudoku or crossword puzzle
- In game theory & especially combinatorial game theory, choosing the best move to make next (e.g. with the minmax algorithm)
- Finding a combination or password from the whole set of possibilities
- Factoring an integer (an important problem in cryptography)
- Search engine optimization (SEO) & content optimization for web crawlers
- Optimizing an industrial process, e.g. a chemical reaction, by changing the parameters of the process (like temperature, pressure, & pH)
- Retrieving a record from a database
- Finding the maximum or minimum value in a list or array
- Checking to see if a given value is present in a set of values

5.1.2.2 Classes of search algorithms

1. **For virtual search spaces.** Algorithms for searching virtual spaces are used in the constraint satisfaction problem, where the goal is to find a set of value assignments to certain varieties that will satisfy specific mathematical equations & inequations/equalities. They are also used when the goal is to find a variable assignment that will maximize or minimize a certain function of those variables. Algorithms for these problems include the basic brute-force search (also called "naïve" or "uninformed" search), & a variety of heuristics that try to exploit partial knowledge about the structure of this space, e.g. linear relaxation, constraint generation, & constraint propagation.

   – Các thuật toán tìm kiếm không gian ảo được sử dụng trong bài toán thỏa mãn ràng buộc, trong đó mục tiêu là tìm 1 tập hợp các phép gán giá trị cho các biến nhất định sẽ thỏa mãn các phương trình và bất phương trình toán học/bằng nhau cụ thể. Chúng cũng được sử dụng khi mục tiêu là tìm 1 phép gán biến sẽ tối đa hóa hoặc tối thiểu hóa 1 hàm nhất định của các biến đó. Các thuật toán cho các bài toán này bao gồm tìm kiếm brute-force cơ bản (còn gọi là tìm kiếm "ngây thơ" hoặc "không có thông tin"), và nhiều phương pháp tìm kiếm khác nhau cố gắng khai thác kiến thức 1 phần về cấu trúc của không gian này, chẳng hạn như thư giãn tuyến tính, tạo ràng buộc và truyền ràng buộc.

   An important subclass are the local search methods, that view the elements of the search space as the vertices of a graph, with edges defined by a set of heuristics applicable to the case; & scan the space by moving from item to item along the edges, e.g. according to the steepest descent or best 1st criterion, or in a stochastic search. This category includes a great variety of general metaheuristic methods, e.g. simulated annealing, tabu search, A-teams, & general programming, that combine arbitrary heuristics in specific ways. The opposite of local search would be global search methods. This method is applicable when the search space is not limited & all aspects of the given network are available to the entity running the search algorithm.

   – 1 phân lớp quan trọng là các phương pháp tìm kiếm cục bộ, xem các phần tử của không gian tìm kiếm như các đỉnh của 1 đồ thị, với các cạnh được xác định bởi 1 tập hợp các phương pháp tìm kiếm áp dụng cho trường hợp này; và quét không gian bằng cách di chuyển từ mục này sang mục khác dọc theo các cạnh, ví dụ theo tiêu chí dốc nhất hoặc tiêu chí tốt nhất đầu tiên, hoặc trong tìm kiếm ngẫu nhiên. Thể loại này bao gồm nhiều phương pháp siêu tìm kiếm chung, chẳng hạn như ủ mô phỏng, tìm kiếm tabu, nhóm A và lập trình di truyền, kết hợp các phương pháp tìm kiếm tùy ý theo những cách cụ thể. Ngược lại với tìm kiếm cục bộ sẽ là các phương pháp tìm kiếm toàn cục. Phương pháp này có thể áp dụng khi không gian tìm kiếm không bị giới hạn và tất cả các khía cạnh của mạng đã cho đều khả dụng đối với thực thể chạy thuật toán tìm kiếm.

   This class also includes various tree search algorithms, that view the elements as vertices of a tree, & traverse that tree in some special order. Examples of the latter include the exhaustive methods e.g. depth-1st search & breadth-1st search, as well as various heuristic-based search tree pruning methods e.g. backtracking & branch & bound. Unlike general metaheuristics, which at best work only in a probabilistic scene, many of these tree-search methods are guaranteed to find the exact or optimal solution, if given enough time. This is called "completeness".

   – Lớp này cũng bao gồm nhiều thuật toán tìm kiếm cây khác nhau, xem các phần tử như các đỉnh của 1 cây và duyệt cây đó theo 1 thứ tự đặc biệt nào đó. Ví dụ về thứ tự sau bao gồm các phương pháp đầy đủ như tìm kiếm theo chiều sâu và tìm kiếm theo chiều rộng, cũng như nhiều phương pháp cắt tỉa cây tìm kiếm dựa trên phương pháp heuristic như quay lui và rẽ nhánh và giới hạn. Không giống như các siêu phương pháp heuristic chung, tốt nhất chỉ hoạt động theo nghĩa xác suất, nhiều phương pháp tìm kiếm cây này được đảm bảo tìm ra giải pháp chính xác hoặc tối ưu, nếu có đủ thời gian. Điều này được gọi là "hoàn chỉnh".

   Another important subclass consists of algorithms for exploring the game tree of multiple-player games, e.g. chess or backgammon, whose nodes consist of all possible game situations that could result from the current situation. The goal in these problems is to find the move that provides the best chance of a win, taking into account all possible moves of the opponent(s). Similar problems occur when humans or machines have to make successive decisions whose outcomes are not entirely under one's control, e.g. in robot guidance or in marketing, financial, or military strategy planning. This kind of problem – combinatorial search – has been extensively studied in the context of AI. Examples of algorithms for this class are the minimax algorithm, alpha–beta pruning, & the A* algorithm & its variants.

   – 1 phân lớp quan trọng khác bao gồm các thuật toán để khám phá cây trò chơi của các trò chơi nhiều người chơi, chẳng hạn như cờ vua hoặc cờ cá ngựa, có các nút bao gồm tất cả các tình huống trò chơi có thể xảy ra do tình huống hiện tại. Mục tiêu của các bài toán này là tìm ra nước đi mang lại cơ hội chiến thắng cao nhất, có tính đến tất cả các nước đi có thể có của đối thủ. Các bài toán tương tự xảy ra khi con người hoặc máy móc phải đưa ra các quyết định liên tiếp mà kết quả không hoàn toàn nằm trong tầm kiểm soát của mình, chẳng hạn như trong hướng dẫn rô-bốt hoặc trong lập kế hoạch chiến lược tiếp thị, tài chính hoặc quân sự. Loại bài toán này – tìm kiếm kết hợp – đã được nghiên cứu rộng rãi trong bối cảnh trí tuệ nhân tạo. Các ví dụ về thuật toán cho lớp này là thuật toán minimax, cắt tỉa alpha–beta và thuật toán A* cùng các biến thể của nó.

2. **For sub-structures of a given structure.** An important & extensively studied subclass are the graph algorithms, in particular graph traversal algorithms, for finding specific sub-structures in a given graph – e.g. subgraphs, paths, circuits, etc. Examples include Dijkstra's algorithm, Kruskal's algorithm, the nearest neighbor algorithm, & Prim's algorithm.

   – *Đối với các cấu trúc con của 1 cấu trúc nhất định.* 1 phân lớp quan trọng và được nghiên cứu rộng rãi là các thuật toán đồ thị, đặc biệt là các thuật toán duyệt đồ thị, để tìm các cấu trúc con cụ thể trong 1 đồ thị nhất định — chẳng hạn như các đồ

thị con, đường dẫn, mạch, etc. Các ví dụ bao gồm thuật toán Dijkstra, thuật toán Kruskal, thuật toán hàng xóm gần nhất và thuật toán Prim.

Another important subclass of this category are the string searching algorithms, that search for patterns within strings. 2 famous examples are the Boyer–Moore & Knuth–Morris–Pratt algorithms, & several algorithms based on the suffix tree data structure.

– 1 phân lớp quan trọng khác của thể loại này là các thuật toán tìm kiếm chuỗi, tìm kiếm các mẫu trong chuỗi. Hai ví dụ nổi tiếng là các thuật toán Boyer–Moore & Knuth–Morris–Pratt, & 1 số thuật toán dựa trên cấu trúc dữ liệu cây hậu tố.

3. **Search for the maximum of a function.** In 1953, American statistician Jack Kiefer devised Fibonacci search which can be used to find the maximum of a unimodal function & has many other applications in CS.

   – *Tìm kiếm giá trị lớn nhất của 1 hàm số.* Vào năm 1953, nhà thống kê người Mỹ Jack Kiefer đã phát minh ra thuật toán tìm kiếm Fibonacci có thể được sử dụng để tìm giá trị lớn nhất của 1 hàm số đơn thức và có nhiều ứng dụng khác trong khoa học máy tính.

4. **For quantum computers.** There are also search methods designed for quantum computers, like Grover's algorithm, that are theoretically faster than linear or brute-force search even without the help of data structures or heuristics. While the ideas & applications behind quantum computers are still entirely theoretical, studies have been conducted with algorithms like Grover's that accurately replicate the hypothetical physical versions of quantum computing systems.

   – *Đối với máy tính lượng tử*: Cũng có những phương pháp tìm kiếm được thiết kế cho máy tính lượng tử, như thuật toán Grover, về mặt lý thuyết nhanh hơn tìm kiếm tuyến tính hoặc tìm kiếm bằng vũ lực ngay cả khi không có sự trợ giúp của cấu trúc dữ liệu hoặc phương pháp tìm kiếm. Trong khi các ý tưởng và ứng dụng đằng sau máy tính lượng tử vẫn hoàn toàn là lý thuyết, các nghiên cứu đã được tiến hành với các thuật toán như thuật toán Grover sao chép chính xác các phiên bản vật lý giả định của hệ thống máy tính lượng tử.

5.2 Basic Graph Theory – Lý Thuyết Đồ Thị Cơ Bản

Resources – Tài nguyên.

1. [AD10]. Titu Andreescu, Gabriel Dospinescu. *Problems From the Book*. Chap. 6: *Some Classical Problems in Extremal Graph Theory – Vài Bài Toán Cổ Điển trong Lý Thuyết Đồ Thị Cực Trị*, pp. 119–136.

2. [Val02; Val21]. Gabriel Valiente. *Algorithms on Trees & Graphs With Python Code*. 2e.

"In mathematics & computer science, *graph theory* is the study of *graphs*, which are mathematical structures used to model pairwise relations between objects. A graph in this context is made up of *vertices* (also called *nodes* or *points*) which are connected by *edges* (also called *arcs, links* or *lines*). A distinction is made between *undirected graphs*, where edges link 2 vertices symmetrically, & *directed graphs*, where edges link 2 vertices asymmetrically. Graphs are 1 of the principal objects of study in discrete mathematics." – Wikipedia/graph theory.

Graphs are such a good way of organizing certain kinds of information & formulating problems in discrete mathematics in general & combinatorics & related fields of mathematics in particular. Intuitively, a graph is a set of dots – called *vertices* – where some of the dots are connected to some of the other dots. The connections are called *edges*. Each edge of a graph connects just 2 vertices & so it can be represented as a pair of vertices.

– Đồ thị là 1 cách rất hay để sắp xếp 1 số loại thông tin & xây dựng các bài toán trong toán học rời rạc nói chung & tổ hợp & các lĩnh vực toán học liên quan nói riêng. Theo trực giác, đồ thị là 1 tập hợp các chấm – được gọi là các *đỉnh* – trong đó 1 số chấm được kết nối với 1 số chấm khác. Các kết nối được gọi là các *cạnh*. Mỗi cạnh của đồ thị chỉ kết nối 2 đỉnh & do đó nó có thể được biểu diễn dưới dạng 1 cặp đỉnh.

"Đồ thị là 1 mô hình toán học diễn tả 1 cách rất tường minh những mối quan hệ 2 ngôi (1 chiều cũng như 2 chiều) giữa các đối tượng cần được xem xét. Chính vì vậy, đồ thị được sử dụng rộng rãi trong rất nhiều lĩnh vực khác nhau; e.g., khoa học, kỹ thuật, kinh tế, xã hội, etc." – [Thà13, Sect. 0.1.5, p. 13]

5.3 Trees & graphs: Some basic concepts – Cây & đồ thị: Vài khái niệm cơ bản

Definitions in graph theory vary. For a comprehensive list of concepts in graph theory, see, e.g., Wikipedia/glossary of graph theory. A graph consists of a set of vertices and a set of edges. Intuitively (and visually), an edge is a direct connection between two vertices. More formally, an edge is a set consisting of a pair of vertices.

The notion of graph which is most useful in computer science is that of a directed graph or just a graph. A graph is a combinatorial structure consisting of a finite nonempty set of objects, called vertices, together with a finite (possibly empty) set of ordered pairs of vertices, called directed edges or arcs.

– Khái niệm đồ thị hữu ích nhất trong khoa học máy tính là đồ thị có hướng hoặc chỉ là đồ thị. Đồ thị là 1 cấu trúc tổ hợp bao gồm 1 tập hợp hữu hạn không rỗng các đối tượng, được gọi là các đỉnh, cùng với 1 tập hợp hữu hạn (có thể rỗng) các cặp đỉnh có thứ tự, được gọi là các cạnh có hướng hoặc cung.

Definition 18 (Graphs & subgraphs, [Sha22], Def. 2.17, p. 54)**.** *A* simple graph *(or just a* graph*) G is a pair of sets (V, E) where V is a nonempty set called the set of* vertices *of G, & E is a (possibly empty) set of unordered pairs of distinct elements of V. The set E is called the set of* edges *of G. If the number of vertices of G is finite, then G is a* finite graph *(or a* finite simple graph*).*

If $G = (V, E)$ & $H = (V', E')$ are both graphs, then H is a subgraph *of G if $V' \subseteq V$ & $E' \subseteq E$.*

Bài toán 41. *Vẽ đồ thị Peterson $G_{\text{Peterson}} = (V, E)$ & mô tả cụ thể V, E.*

To construct a subgraph of a graph given, we take some of the vertices & some of the edges, but we have to make sure that we have included all the vertices of the edges that we chose.

Definition 19 (Directed graph, [Val21], Def. 1.1, p. 3)**.** *A* graph *$G = (V, E)$ consists of a finite nonempty set V of vertices & a finite set $E \subseteq V \times V$ of edges. The* order *of a graph $G = (V, E)$, denoted by n, is the number of vertices, $n = |V|$ & the* size*, denoted by m, is the number of edges, $m = |E|$. An edge $e = (v, w)$ is said to be* incident *with vertices v & w, where v is the* source *& w the* target *of edge e, & vertices v, w are said to be* adjacent*. Edges $(u, v), (v, w)$ are said to be* adjacent*, as are edges $(u, v), (w, v)$, & also edges $(v, u), (v, w)$.*

Định nghĩa 12 (Đồ thị có hướng)**.** *1 đồ thị $G = (V, E)$ bao gồm 1 tập hữu hạn không rỗng V các đỉnh & 1 tập hữu hạn $E \subseteq V \times V$ các cạnh. Bậc của 1 đồ thị $G = (V, E)$, ký hiệu là n, là số đỉnh, $n = |V|$ &* size*, ký hiệu là m, là số cạnh, $m = |E|$. Một cạnh $e = (v, w)$ được gọi là* incident *với các đỉnh v & w, trong đó v là* source *& w* target *của cạnh e, & các đỉnh v, w được gọi là* kề*. Các cạnh $(u, v), (v, w)$ được gọi là* kề*, cũng như các cạnh $(u, v), (w, v)$, & cũng vậy các cạnh $(v, u), (v, w)$.*

Question 5 (Size/Complexity of simple graph)**.** *How to measure the size or the complexity of a simple graph?*
– Làm thế nào để đo kích thước hoặc độ phức tạp của 1 đồ thị đơn?

Có thể định nghĩa vài độ đo thích hợp để đo kích thước hoặc độ phức tạp của 1 đồ thị đơn G, e.g.,

$$m(G) = |V(G)| + |E(G)|.$$

Graphs are often drawn as a set of points in the plane & a set of arrows, each of which joins 2 (not necessarily different) points. In a drawing of a graph $G = (V, E)$, each vertex $v \in V$ is drawn as a point or a small circle & each edge $(v, w) \in E$ is drawn as an arrow from point or circle of vertex v to the point or circle corresponding to vertex w.

– Đồ thị thường được vẽ như 1 tập hợp các điểm trên mặt phẳng & 1 tập hợp các mũi tên, mỗi mũi tên nối 2 điểm (không nhất thiết phải khác nhau). Trong bản vẽ đồ thị $G = (V, E)$, mỗi đỉnh $v \in V$ được vẽ như 1 điểm hoặc 1 đường tròn nhỏ & mỗi cạnh $(v, w) \in E$ được vẽ như 1 mũi tên từ điểm hoặc đường tròn của đỉnh v đến điểm hoặc đường tròn tương ứng với đỉnh w.

A vertex has 2 degrees in a graph, one given by the number of edges coming into the vertex & the other given by the number of edges in the graph going out of the vertex.

– Mỗi đỉnh có 2 bậc trong đồ thị, 1 bậc được xác định bởi số cạnh đi vào đỉnh & bậc còn lại được xác định bởi số cạnh trong đồ thị đi ra khỏi đỉnh.

Definition 20 ([Val21], Def. 1.2, p. 4)**.** *The* indegree *of a vertex v in a graph $G = (V, E)$ is the number of edges in G whose target is v, i.e.,* $\text{indeg}(v) = |\{(u, v) | (u, v) \in E\}|$*. The* outdegree *of a vertex v in a graph $G = (V, E)$ is the number of edges in G whose source is v, i.e.,* $\text{outdeg}(v) = |\{(v, w) | (v, w) \in E\}|$*. The* degree *of a vertex v in a graph $G = (V, E)$ is the sum of the indegree & the outdegree of the vertext, i.e.,* $\deg(v) = \text{indeg}(v) + \text{outdeg}(v)$*.*

A basic relationship between the size of a graph & the degree of its vertices, which will prove to be very useful in analyzing the computational complexity of algorithms on graphs:

– Mối quan hệ cơ bản giữa kích thước của đồ thị & bậc của các đỉnh, sẽ rất hữu ích trong việc phân tích độ phức tạp tính toán của các thuật toán trên đồ thị:

Theorem 17. *Let $G = (V, E)$ be a graph with n vertices & m edges, & let $V = \{v_1, \ldots, v_n\}$. Then*

$$\sum_{i=1}^{n} \text{indeg}(v_i) = \sum_{i=1}^{n} \text{outdeg}(v_i) = m.$$

Definition 21 (Multigraphs, general graphs, [Sha22], Def. 2.20, p. 54)**.** *In the definition of a graph, if we allow E to be a multiset – i.e., allow* repeated edges *– then G is called a* multigraph *or a* graph with repeated edges*. If we also allow pairs of non-distinct elements in E – i.e., allow* loops *– then G is called a* general graph *or a* graph with repeated edges & loops*.*

Định nghĩa 13 (Đa đồ thị, đồ thị tổng quát). *Trong định nghĩa của đồ thị, nếu chúng ta cho phép E là 1 đa tập – tức là cho phép* các cạnh lặp lại *– thì G được gọi là* đa đồ thị *hoặc* đồ thị có các cạnh lặp lại. *Nếu chúng ta cũng cho phép các cặp phần tử không phân biệt trong E – tức là cho phép* các vòng lặp *– thì G được gọi là* đồ thị tổng quát *hoặc* đồ thị có các cạnh lặp lại & vòng lặp.

If a graph G does not have any double edges or loops then $G$is a simple graph. If a graph G has repeated edges but no loops, then G is a multigraph. If G has both repeated edges & repeated loops, then G would be a general graph.

Remark 10 ([Sha22], Rmk. 2.22, p. 55). *If $G = (V, E)$ is a general graph & $e \in E$ is an edge of G, then, unlike in the case of simple graphs, we cannot necessarily identify e by its vertices. I.e., it will be inadequate to say $e = \{v, w\}$, where $v, w \in V$. This is because there may be multiple edges connecting v & w. Hence, to be formally correct, we should say that a general graph consists of 2 sets V & E & an incidence map that gives 2 vertices for each $e \in E$. I.e., every edge will have its own name in addition to being associated with its 2 vertices. To avoid unnecessary clutter, in what follows we will continue to identify edges of general graphs as pairs of vertices unless there is a need to be more formal.*

– Nếu $G = (V, E)$ là 1 đồ thị tổng quát & $e \in E$ là 1 cạnh của G, thì, không giống như trong trường hợp của đồ thị đơn giản, chúng ta không nhất thiết phải xác định e theo các đỉnh của nó. Tức là, sẽ không đủ nếu nói $e = \{v, w\}$, trong đó $v, w \in V$. Điều này là do có thể có nhiều cạnh kết nối v & w. Do đó, để chính xác về mặt hình thức, chúng ta phải nói rằng 1 đồ thị tổng quát bao gồm 2 tập hợp V & E & 1 bản đồ tới cung cấp 2 đỉnh cho mỗi $e \in E$. Tức là, mỗi cạnh sẽ có tên riêng của nó ngoài việc được liên kết với 2 đỉnh của nó. Để tránh sự lộn xộn không cần thiết, trong phần sau, chúng ta sẽ tiếp tục xác định các cạnh của đồ thị tổng quát là các cặp đỉnh trừ khi cần phải chính thức hơn.

Definition 22 (Order, adjacency, incidence, [Sha22], Def. 2.23, p. 55). *Let $G = (V, E)$ be a general graph. Then $|V|$, the number of vertices, is called the* order *of G. If $\alpha = \{x, y\} \in E$ – i.e., α is an edge connecting the vertices x, y – then we say that x & y are* vertices *or* ends *of α, that α* joints *x & y, that x & y are* adjacent, *& that x & α are* incident.

Definition 23 (Degree of a vertex, [Sha22], Def. 2.24, p. 55). *If x is a vertex of a graph G, then the number of edges incident with x is called the* degree *(or the* valence*) of x. In the case of general graphs, a loop at a vertex x contributes* 2 *to the degree of x.*

The degree of a vertex is the number of edges incident with it & note that, in a general graph, a loop contributes 2 to the degree of its vertex. One usually records the degrees of each vertices of a graph in a table.

– Bậc của 1 đỉnh là số cạnh liên quan đến nó & lưu ý rằng, trong 1 đồ thị tổng quát, 1 vòng lặp đóng góp 2 vào bậc của đỉnh của nó. Người ta thường ghi lại bậc của mỗi đỉnh của đồ thị trong 1 bảng.

Roughly speaking, in graph theory, we are not concerned with the labels of the vertices. So, if we keep the vertices & edges intact, but rearrange the names of the vertices, we get a new graph that is basically the same graph as G. We say that these 2 graphs are *isomorphic*.

– Nói 1 cách đại khái, trong lý thuyết đồ thị, chúng ta không quan tâm đến nhãn của các đỉnh. Vì vậy, nếu chúng ta giữ nguyên các đỉnh & cạnh, nhưng sắp xếp lại tên của các đỉnh, chúng ta sẽ có 1 đồ thị mới về cơ bản là cùng 1 đồ thị với G. Chúng ta nói rằng 2 đồ thị này là *đẳng cấu*.

More technically, in combinatorial graph theory – as opposed to topological graph theory – we are not concerned with how a graph is drawn. The drawing is just a representation that helps us "see" the vertices & the edges. All that matters is the number of vertices & their adjacencies. Hence, we consider 2 graphs the same if, after we appropriately relabel the vertices of 1 of the graphs, the set of edges of the 2 graphs become the same. 2 such graphs are called isomorphic.

Definition 24 (Graph isomorphism). *Let G, H be graphs. The graph G is* isomorphic *to the graph H if there exists a 1-1, onto map $f : V(G) \to V(H)$ s.t. $\{x, y\}$ is an edge of G iff $\{f(x), f(y)\}$ is an edge of H. Such a map f is called an* isomorphism *between G & H.*

Example 9. *In the Peterson graph, every vertex has degree* 3.

A few oft-used terms to graph vocabularies.

Definition 25 (Isolated vertex, leaf, [Sha22], Def. 10.2, p. 362). *Let G be a graph. A vertex with degree* 0 *is called an* isolated vertex*, while a vertex with degree* 1 *is called a* leaf.

Definition 26 (Regular graph, [Sha22], Def. 10.3, p. 362). *A graph is called* regular of degree d *or* d-regular *if each vertex has degree equal to d.*

In mathematical notation:

$$G = (V, E) \text{ is a } d\text{-regular} \Leftrightarrow \deg v = d, \ \forall v \in V.$$

Definition 27 (Cubic graph, [Sha22], Def. 10.4, p. 362). *A graph is called* cubic *if it is regular of degree* 3 *(i.e., if each vertex has degree equal to* 3*).*

5.3.1 Walks, trails, paths, cycles, & complete graphs

Walks, trails, & paths in a graph are alternating sequences of vertices & edges in the graph s.t. each edge in the sequence is preceded by its source vertex & followed by its target vertex. Trails are walks having no repeated edges, & paths are trails having no repeated vertices.

– Đường đi, đường mòn, & đường đi trong đồ thị là chuỗi xen kẽ các đỉnh & cạnh trong đồ thị, tức là mỗi cạnh trong chuỗi được đi trước bởi đỉnh nguồn & theo sau bởi đỉnh đích. Đường mòn là đường đi không có cạnh lặp lại, & đường đi là đường mòn không có đỉnh lặp lại.

Definition 28 (Walk, trail, path, [Val21], Def. 1.3). *A* walk *from vertex v_i to vertex v_j in a graph is an alternating sequence $[v_i, e_{i+1}, v_{i+1}, e_{i+2}, \ldots, v_{j-1}, e_j, v_j]$ of vertices & edges in the graph, s.t. $e_k = (v_{k-1}, v_k)$ for $k = i+1, \ldots, j$. A* trail *is a walk with no repeated edges, & a* path *is a trail with no repeated vertices (except, possibly, the initial & final vertices). The length of a walk, trail, or path is the number of edges in the sequence.*

Định nghĩa 14 (Đường đi dạo, đường mòn, đường đi). *1* đường đi dạo *từ đỉnh v_i đến đỉnh v_j trong 1 đồ thị là 1 chuỗi xen kẽ $[v_i, e_{i+1}, v_{i+1}, e_{i+2}, \ldots, v_{j-1}, e_j, v_j]$ các đỉnh & cạnh trong đồ thị, s.t. $e_k = (v_{k-1}, v_k)$ với $k = i+1, \ldots, j$. Một* đường mòn *là 1 cuộc đi bộ không có cạnh nào lặp lại, & 1* đường đi *là 1 cuộc đi bộ không có đỉnh nào lặp lại (ngoại trừ, có thể là, các đỉnh đầu & cuối). Độ dài của 1 cuộc đi bộ, trail hoặc path là số cạnh trong chuỗi.*

Since an edge in a graph is uniquely determined by its source & target vertices, a walk, trail, or path can be abbreviated by just enumerating either the vertices $[v_i, v_{i+1}, \ldots, v_{j-1}, v_j]$ or the edges $[e_{i+1}, e_{i+2}, \ldots, e_j]$ in the alternating sequence $[v_i, e_{i+1}, v_{i+1}, e_{i+2}, \ldots, v_{j-1}, e_j, v_j]$ of vertices & edges.

– Vì 1 cạnh trong đồ thị được xác định duy nhất bởi các đỉnh nguồn & đích của nó, nên 1 đường đi, đường mòn hoặc đường dẫn có thể được rút gọn chỉ bằng cách liệt kê các đỉnh $[v_i, v_{i+1}, \ldots, v_{j-1}, v_j]$ hoặc các cạnh $[e_{i+1}, e_{i+2}, \ldots, e_j]$ trong chuỗi xen kẽ $[v_i, e_{i+1}, v_{i+1}, e_{i+2}, \ldots, v_{j-1}, e_j, v_j]$ các đỉnh & cạnh.

Walks are closed if their initial & final vertices coincide. – Đường đi sẽ khép lại nếu đỉnh đầu & đỉnh cuối trùng nhau.

Definition 29 (Cycle, [Val21], Def. 1.4). *A walk, trail, or path $[v_i, e_{i+1}, v_{i+1}, e_{i+2}, \ldots, v_{j-1}, e_i, v_j]$ is said to be* closed *if $v_i = v_j$. A* cycle *is a closed path of length at least* 1.

The combinatorial structure of a graph encompasses 2 notions of the substructure. A subgraph of a graph is just a graph whose vertex & edge sets are contained in the vertex & edge sets of the given graph, resp. The subgraph of a graph induced by a subset of its vertices has as edges the set of edges in the given graph whose source & target belong to the subset of vertices.

– Cấu trúc tổ hợp của 1 đồ thị bao gồm 2 khái niệm về cấu trúc con. Một đồ thị con của 1 đồ thị chỉ là 1 đồ thị có tập đỉnh & cạnh được chứa trong tập đỉnh & cạnh của đồ thị đã cho, tương ứng. Đồ thị con của 1 đồ thị được tạo ra bởi 1 tập con các đỉnh của nó có các cạnh là tập các cạnh trong đồ thị đã cho có nguồn & đích thuộc về tập con các đỉnh.

Definition 30 (Subgraph, [Val21], Def. 1.5, p. 6). *Let $G = (V, E)$ be a graph, & let $W \subseteq V$. A graph (W, S) is a* subgraph *of G if $S \subseteq E$. The subgraph of G* induced *by W is the graph $(W, E \cap W \times W)$.*

Định nghĩa 15 (Đồ thị con). *Cho $G = (V, E)$ là 1 đồ thị, & cho $W \subseteq V$. Một đồ thị (W, S) là 1* đồ thị con *của G nếu $S \subseteq E$. Đồ thị con của G* được tạo ra *bởi W là đồ thị $(W, E \cap W \times W)$.*

The notion of graph which is most often found in mathematics is that of an undirected graph. Unlike the directed edges or edges of a graph, edges of an undirected graph have no direction association with them & therefore, no distinction is made between the source & target vertices of an edge. In a mathematical sense, an undirected graph consists of a set of vertices & a finite set of undirected edges, where each edge has a set of 1 or 2 vertices associated with it. In the computer science view of undirected graphs, though, an undirected graph is the particular case of a directed graph in which for every edge (v, w) of the graph, the reversed edge (w, v) also belongs to the graph. Undirected graphs are also called *bidirected*.

– Khái niệm đồ thị thường thấy nhất trong toán học là đồ thị vô hướng. Không giống như các cạnh hoặc cạnh có hướng của đồ thị, các cạnh của đồ thị vô hướng không có liên kết hướng nào với chúng & do đó, không có sự phân biệt nào được tạo ra giữa các đỉnh nguồn & đích của 1 cạnh. Theo nghĩa toán học, đồ thị vô hướng bao gồm 1 tập hợp các đỉnh & 1 tập hợp hữu hạn các cạnh vô hướng, trong đó mỗi cạnh có 1 tập hợp gồm 1 hoặc 2 đỉnh được liên kết với nó. Tuy nhiên, theo quan điểm khoa học máy tính về đồ thị vô hướng, đồ thị vô hướng là trường hợp cụ thể của đồ thị có hướng trong đó đối với mọi cạnh (v, w) của đồ thị, cạnh đảo ngược (w, v) cũng thuộc về đồ thị. Đồ thị vô hướng cũng được gọi là *song hướng*.

Definition 31 (Undirected graph, [Val21], Def. 1.5, p. 6). *A graph $G = (V, E)$ is* undirected *if $(v, w) \in E \Rightarrow (w, v) \in E$, $\forall v, w \in V$.*

Định nghĩa 16 (Đồ thị vô hướng). *Đồ thị $G = (V, E)$ là* vô hướng *nếu $(v, w) \in E \Rightarrow (w, v) \in E$, $\forall v, w \in V$.*

Bài toán 42 ([Sha22], p. 361). *A soccer ball is often tiled with* 12 *pentagons &* 20 *hexagons. Is there anything special about those numbers? If you tile a soccer ball with pentagons & hexagons, what are the possibilities for the number of pentagons & the number of hexagons?*

– 1 quả bóng đá thường được lát bằng 12 *hình ngũ giác &* 20 *hình lục giác. Có điều gì đặc biệt về những con số đó không? Nếu bạn lát 1 quả bóng đá bằng hình ngũ giác & hình lục giác, thì khả năng có bao nhiêu hình ngũ giác & số hình lục giác?*

5.3.2 Graphic sequences

Problem 36 ([Sha22], Warm-Up 10.5, p. 363). *Is there a simple graph with 4 vertices s.t. the degrees of the vertices are* $3, 2, 1, 1$. *What if the degrees were* $2, 2, 1, 1$*?*

Chứng minh. □

Definition 32 (Degree sequence of a graph; graphic sequences, [Sha22], Def. 10.6, p. 363). *The* degree sequence *of a graph is the list of the degrees of its vertices in non-increasing order.*

A non-increasing sequence of nonnegative integers is called graphic *if there exists a simple graph whose degree sequence is precisely that sequence.*

Question 6. *Given a non-increasing sequence of nonnegative integers, when is the sequence graphic?*
– Với 1 dãy số nguyên không âm không tăng, khi nào thì dãy số này là dãy đồ họa?

Problem 37 ([Sha22], Ques. 10.8, p. 363). *Which of the following sequences are graphic? (a)* $7, 5, 5, 4, 3, 3, 3, 3, 3, 2$. *(b)* $1, 1, 1$. *(c)* $5, 5, 4, 3, 3, 2, 2, 1$. *(d)* $7, 5, 5, 4, 3, 2, 2, 0$. *(e)* $6, 6, 6, 6, 4, 3, 3, 0$. *(f)* $7, 6, 5, 5, 4, 4, 4, 2, 1$.

Theorem 18 ([Sha22], Thm. 10.9, p. 363). *Let* $G = (V, E)$ *be a general graph. Let* $\{d_i\}_{i=1}^{|V|}$ *be the degrees of the vertices. Then*

$$\sum_{i=1}^{|V|} d_i = 2|E|.$$

In particular, the number of vertices of G *with odd degree is even.*

Theorem 19 (Havel–Hakimi algorithm). *Consider the following 2 sequences of nonnegative integers. Assume the 1st sequence is in nonincreasing order &* $t_s \geq 1$*: (a)* $s, t_1, \ldots, t_s, d_1, \ldots, d_n$. *(b)* $t_1 - 1, t_2 - 1, \ldots, t_s - 1, d_1, \ldots, d_n$. *Sequence (a) is graphic iff sequence (b) is graphic (after possibly rearranging it to make it non-decreasing).*

Because of this theorem, if you are given a sequence of nonnegative integers & want to know if it is graphic or not, you repeatedly use Thm. 19 to reduce your sequence to simpler sequences. If, at any point, the simpler sequence is graphic (or not graphic), then so is the original sequence.

– Vì định lý này, nếu bạn được cung cấp 1 chuỗi các số nguyên không âm & muốn biết nó có đồ họa hay không, bạn sử dụng Thm. 19 nhiều lần để rút gọn chuỗi của bạn thành các chuỗi đơn giản hơn. Nếu, tại bất kỳ thời điểm nào, chuỗi đơn giản hơn là đồ họa (hoặc không đồ họa), thì chuỗi ban đầu cũng vậy.

Problem 38 ([Sha22], P10.1.1., p. 367). *A simple graph* G *has* 9 *edges & the degree of each vertex is at least* 3. *What are the possibilities for the number of vertices? Give an example, for each possibility.*

Problem 39 ([Sha22], P10.1.2., p. 367). *Is* $7, 7, 6, 5, 4, 4, 4, 3, 2$ *a graphic sequence?*

Problem 40 ([Sha22], P10.1.3., p. 367). *Is there a simple regular graph of degree* 5 *with* 8 *vertices? Why?*

Problem 41 ([Sha22], P10.1.4., p. 367). *Is there a simple graph on* 8 *vertices where half of the degrees are* 5 *& the other half are* 3*?*

Problem 42 ([Sha22], P10.1.5., p. 367). *The sequence* $7, 5, 5, 4, 3, 3, 3, 3, 3, 2$ *is graphic because it is the degree sequence of the simple graph of [Sha22, Fig. 10.1]. Apply the Havel–Hakimi algorithm Thm. 19 to construct a graph with this degree sequence. Is the resulting graph isomorphic to the graph of [Sha22, Fig. 10.1]?*

Problem 43 ([Sha22], P10.1.6., p. 367). *Assume that you applied the Havel–Hakimi algorithm to a given sequence, &, at the end of the process, you arrived at a graphic sequence. You draw a simple graph corresponding to this final sequence, & work your way back to construct a simple graph with the original sequence as its degree sequence. As you work your way back, is it the case that, at every step, after adding a new vertex, you add edges between this new vertex & those existing vertices that have the highest degrees? Either prove that you do or provide an example where you don't.*

Problem 44 ([Sha22], P10.1.7., p. 367). *Assume that you have a sequence* d_1, d_2, d_3, d_4 *(with* $d_1 \geq d_2 \geq d_3 \geq d_4 \geq 0$*) that you know is not graphic. You consider the sequence* $d_1, d_2, d_3, d_4 + 1, 1$. *Can this sequence be graphic (after possible rearranging so that it is non-increasing order)? Either prove that it is never graphic or given an example where it becomes graphic.*

Problem 45 ([Sha22], P10.1.8., p. 367). *A sequence is graphic if it is the degree sequence of a* simple *graph. Is there a sequence that is not graphic, & yet is the degree sequence of a multigraph? Either prove that there are no such sequences or give a specific example.*

Problem 46 ([Sha22], P10.1.9., p. 367). *Can you find a sequence that is* not *the degree sequence of a multigraph, but is the degree sequence of a general graph?*

Problem 47 ([Sha22], P10.1.10., p. 367). *Let $d_1, \ldots, d_n$ be a non-increasing sequence of n nonnegative integers. By Thm. 18, if this sequence is the degree sequence of a general graph, then the sum $\sum_{i=1}^{n} d_i$ is even. What about the converse?*

Problem 48 ([Sha22], P10.1.11., p. 367). *Find 2 non-isomorphic regular simple graphs of degree 3 & order 6.*

Problem 49 ([Sha22], P10.1.12., p. 368). *Apply the Havel–Hakimi algorithm of Thm. 19 to the sequence $5, 4, 4, 2, 2, 1$. Is the sequence graphic? Can you use the Havel–Hakimi algorithm to find a general graph with this degree sequence that has exactly 1 loop?*

Problem 50 ([Sha22], P10.1.13., p. 368). *(a) Prove that a sequence $d_1, \ldots, d_p$ is a graphic sequence iff the sequence $p - d_p - 1, \ldots, p - d_1 - 1$ is graphic. (b) Is $9, 9, 9, 9, 9, 9, 9, 9, 8, 8, 8$ a graphic sequence?*

Problem 51 ([Sha22], P10.1.14., p. 368). *I have a simple graph with 6 vertices. The degrees of 5 of the vertices are $5, 4, 4, 2, 2$. What are the possibilities for the degree of the 6th vertex?*

Problem 52 ([Sha22], P10.1.15., p. 368). *Can we have a simple graph where the degree sequence consists of all distinct integers? What about a multigraph?*

Problem 53 ([Sha22], P10.1.16., p. 368). *Let G be a simple graph with 94 vertices. Assume that all the degrees of the vertices are odd integers. Prove that there are at least 3 vertices with the same degree.*

Problem 54 ([Sha22], P10.1.17., p. 368). *Assume that $a_1 \geq a_2 \geq \cdots a_n$ is a graphic sequence. Prove that*

$$\sum_{i=1}^{k} a_i \leq k(k-1) + \sum_{i=k+1}^{p} \min\{k, a_i\}.$$

Remark 11. *The Erdős–Gallai theorem asserts that this necessary condition, together with the trivial condition that the sum of the degrees be even, gives a sufficient condition for identifying graphic sequences. This theorem can be proved (see [Choudrum1986]) by induction on the sum of the degrees. Given a degree sequence satisfying the condition, you subtract 1 from the smallest degree & 1 from 1 of the other degrees (the smallest index t with $a_t > a_{t-1}$ or, if all the degrees are equal, then from a_{n-1}). After (tediously) showing that this new sequence also satisfies the Erdős–Gallai condition, you use the inductive hypothesis, & finish the proof using the result in [Sha22, Prob. P10.1.18, p. 368].*

Problem 55 ([Sha22], P10.1.18., p. 368). *Let $p, t \in \mathbb{N}^\star$ with $2 \leq t < p$. Assume $a_1 \geq a_2 \geq \cdots \geq a_{t-1} > a_t \geq \cdots \geq a_{p-1} > a_p$ is the degree sequence of a simple graph. Prove that $a_1 \geq a_2 \geq \cdots \geq a_{t-1} \geq a_t + 1 \geq \cdots \geq a_{p-1} \geq a_p + 1$ is also the degree sequence of a simple graph.*

Problem 56 ([Sha22], P10.1.19., pp. 368–369, Kapoor, Polimeni, Wall1977). *Does there exist a simple graph with 48 vertices where the set of the degrees of the vertices is $\{4, 7, 47\}$? I.e., we want all the degrees of the vertices to be either 4, or 7, or 47, & for there to be at least 1 vertex with each of these degrees. More generally, let $S = \{a_1, \ldots, a_k\}$ be any nonempty set of k positive integers with $a_1 < a_2 < \cdots < a_k$. Prove that there exists some simple graph G with $a_k + 1$ vertices, where the set of the degrees of the vertices is precisely S. (Note that we are not specifying the degree sequence of graph, just the set of numbers that occur as degrees.) You may find the following steps helpful:*

- *Step 1: Assume $|S| = 1$, & $S = \{a_1\}$. Give an example of a simple graph where all the degrees are equal to a_1.*
- *Step 2: Construct a simple graph G with $p + q$ vertices as follows. Start with a complete graph K_p & q isolated vertices. Now add an edge between every isolated vertex & every vertex of K_p. What is the set of degrees of G?*
- *Step 3: Assume $|S| = 2$, & $S = \{a_1, a_2\}$ with $a_1 < a_2$. By a judicious choice of p, q in the previous step, given an example of a simple graph where the set of the degrees is precisely S.*
- *Step 4: Let $p, q, r, m \in \mathbb{N}^\star$. Assume that H is a simple graph with r vertices, & that $S_1 = \{b_1, \ldots, b_m\}$ is the set of the degrees of H. Construct a simple graph G with $p + q + r$ vertices as follows. Start with a K_p (complete graph of order p), a copy of H, & q isolated vertices. Add an edge between every vertex in K_p & each of the other vertices. What is the set of degrees of the vertices of G?*
- *Step 5: To prove the general statement, induct on $|S|$. Using the inductive hypothesis, start with a graph H with degree set equal to $\{a_2 - a_1, a_3 - a_1, \ldots, a_{k-1} - a_1\}$, & judiciously choose p, q in Step 4.*

Goal 1 (Tính khả dĩ của dãy bậc của đồ thị). *Tìm vài dấu hiệu hoặc vài điều kiện cần & đủ để có thể quyết định liệu 1 dãy số nguyên dương $(a_i)_{i=1}^n \subset \mathbb{N}$ cho trước có thể thể là dãy bậc của đồ thị mà không phải vẽ biểu đồ.*

Định nghĩa 17 ([HT24], Def. 7.6, p. 249, Dãy bậc của đồ thị, chuỗi đồ thị). Chuỗi bậc *của đồ thị là dãy bậc của các đỉnh của nó theo thứ tự không tăng. 1 dãy số nguyên không âm không tăng được gọi là* đồ thị *nếu tồn tại 1 đồ thị có chuỗi bậc chính xác là dãy số nguyên không âm đó.*

Ví dụ 1 (Sequence $1, 1, \ldots, 1$)**.** *$1, 1, 1$ không phải là 1 dãy đồ thị vì không thể xây dựng 1 đồ thị có 3 đỉnh sao cho tất cả 3 bậc là 1. Nhưng $1, 1$ & $1, 1, 1, 1$, hay nói chung các dãy chỉ toàn số 1 với độ dài là 1 số chẵn, i.e., $\{1\}_{i=1}^{2n}$, $\forall n \in \mathbb{N}^\star$, là các dãy đồ thị, nhưng bất kỳ dãy chỉ toàn số 1 với độ dài là 1 số lẻ, i.e., $\{1\}_{i=1}^{2n+1}$, $\forall n \in \mathbb{N}^\star$, thì không phải là 1 dãy đồ thị (why?)*

Định lý 4 (Euler's, [HT24], Thm. 7.9, p. 250)**.** *Cho $G = (V, E)$ là đồ thị tổng quát với $d_1, \ldots, d_{|V|} \in \mathbb{N}$ là bậc của các đỉnh. Khi đó $\sum_{i=1}^{|V|} d_i = d_1 + d_2 + \cdots + d_{|V|} = 2|E|$. Nói riêng, số đỉnh của G có bậc lẻ là số chẵn.*

Briefly:

$$d_1, \ldots, d_{|V|} \text{ are degrees of vertices of a graph } G = (V, E) \Rightarrow \sum_{i=1}^{|V|} d_i = 2|E| \Rightarrow |\{i; d_i \not\vdots 2\}| \vdots 2.$$

Chú ý chiều ngược lại chưa chắc đúng:

Ví dụ 2. *Dãy số $7, 5, 5, 4, 3, 2, 2, 0$ không mâu thuẫn với Định lý 18 nhưng nó không phải là đồ thị (why?).*

Question 7. *Có thể suy ra được những hệ quả nào từ đẳng thức $\sum_{i=1}^{|V|} d_i = 2|E|$?*

Bài toán 43. *Cho $G = (V, E)$ là đồ thị tổng quát với $d_1, \ldots, d_p \in \mathbb{N}$ là bậc của các đỉnh. Chứng minh: Bậc cao nhất $d_{\max} := \max_{1 \le i \le p} d_i$ thỏa $d_{\max} \ge \dfrac{2|E|}{|V|}$.*

Bài toán 44. *Viết chương trình* C/C++, Python *sử dụng định lý Euler 18 & thuật toán Havel–Hakimi 19 để kiểm tra 1 dãy số nguyên không âm được nhập có phải là 1 graphical sequence hay không.*

Input. *Dòng 1 chứa số bộ test $t \in \mathbb{N}^\star$. Tiếp theo, mỗi bộ test gồm 2 dòng: Dòng 1 chứa $n := |V| \in \mathbb{N}^\star$: số đỉnh của đồ thị $G = (V, E)$. Dòng 2 chứa dãy $d_1, \ldots, d_n \in \mathbb{N}$.*

Output. *Xuất ra* `1` *nếu dãy đó là dãy graphical sequence,* `0` *nếu dãy đó không phải là dãy graphical sequence, nếu chưa quyết định được thì xuất ra dãy không thể giảm được nữa thu được từ thuật toán Havel–Hakimi 19.*

Sample.

graphical_sequence.inp	graphical_sequence.out
3 4 3 2 1 1 4 2 2 1 1 10 7 7 6 6 6 5 5 4 3 1 9 7 7 6 5 4 4 4 3 2 10 7 5 5 4 3 3 3 3 3 2	

Chứng minh. • Input: https://github.com/NQBH/advanced_STEM_beyond/tree/main/combinatorics/input.

- Output: https://github.com/NQBH/advanced_STEM_beyond/tree/main/combinatorics/output.
- Python code:
  - LDL's Havel–Hakimi algorithm Python code: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/Python/LDL_Havel_Hakimi_alg.py.

```
def havel_hakimi(sequence):
    while True:
        sequence = [d for d in sequence if d != 0]

        if not sequence:
            return True

        # Sort in non-increasing order
        sequence.sort(reverse=True)
        d = sequence.pop(0)
```

```
            if d > len(sequence):
                return False

            # Subtract 1 from the next d elements
            for i in range(d):
                sequence[i] -= 1
                if sequence[i] < 0:
                    return False

            print(f"After processing: {sequence}, removed: {d}")

    def main():
        t = int(input("Enter number of test cases: "))
        for _ in range(t):
            n = int(input("Enter number of members: "))
            sequence = list(map(int, input(f"Enter {n} degree values: ").split()))
            if havel_hakimi(sequence):
                print("YES")
            else:
                print("NO")

    if __name__ == "__main__":
        main()
```

- C++ code:
  - VNTA's Havel–Hakimi algorithm Python code: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/C++/VNTA_Havel_Hakimi_alg.cpp.

```
#include <bits/stdc++.h>
using namespace std;

void print(vector<int>&d) {
    for (int x : d) cout<<x<<' ';
    cout<<'\n';
}

bool allZero(vector<int>&d) {
 for (int x : d)
 if (x!=0) return false;
 return true;
}

bool check(vector<int>&d) {
    long long sum=0;
    for(int x : d) sum+=x;
    if (sum%2!=0) return false;
    int cur = d[0];
    while (true) {
        sort(d.begin(), d.end(), greater<int>());
        print(d);
        cur = d[0];
        if (d[cur]==0) return false;
        d.erase(d.begin());
        if (cur > (int)d.size()) return false;
        for (int i=0; i<cur; i++) d[i]--;
        if (allZero(d)) {
            print(d);
            return true;
```

```
            }
        }
    }

    int main() {
        ios_base::sync_with_stdio(0);
        cin.tie(0); cout.tie(0);
        int t; cin>>t;
        while (t--) {
            int n; cin>>n;
            vector<int>d(n);
            for(int i=0; i<n; i++) cin>>d[i];
            cout<<"\n----------------------------\n";
            if(check(d)) cout<<1<<'\n';
            else cout<<0<<'\n';
            cout<<"\n----------------------------\n";
        }
    }
```

□

5.3.3 Miscellaneous: Graph theory

Denote by $d(V), C(V)$ the number, & the set of vertices adjacent to a vertex V, respectively. A graph is said to have a *complete k-subgraph* if there are k vertices any 2 of which are connected. A graph is said to be *k-free* if it does not contain a complete k-subgraph.

Lemma 3 ([AD10], Example 1, p. 121, Zarankiewicz's lemma)**.** *If G is a k-free graph, then there exists a vertex having degree at most $\left\lfloor \frac{k-2}{k-1} n \right\rfloor$.*

Zarankiewicz's lemma is the main step in the proof of Turan's theorem – a famous classical result about k-free graphs.

Theorem 20 ([AD10], Example 2, p. 123, Turan's theorem)**.** *The greatest number of edges of a k-free graph with n vertices is*

$$\frac{k-2}{k-1} \cdot \frac{n^2 - r^2}{2} + \binom{r}{2},$$

where r is the remainder left by n when divided to $k-1$.

Định nghĩa 18 ([HT24], Def. 7.2, p. 249, Đỉnh cô lập, lá)**.** *Cho G là 1 đồ thị. Đỉnh có bậc 0 được gọi là* đỉnh cô lập*, đỉnh có bậc 1 được gọi là* lá*.*

Định nghĩa 19 ([HT24], Def. 7.3, p. 249, Đồ thị chính quy)**.** *1 đồ thị được gọi là* chính quy bậc d *hoặc* d-chính quy *nếu mỗi đỉnh có bậc bằng $d \in \mathbb{N}$.*

Định nghĩa 20 ([HT24], Def. 7.3, p. 249, Đồ thị khối)**.** *1 đồ thị được gọi là* đồ thị bậc 3 *nếu nó chính quy bậc 3, i.e., mỗi đỉnh đồ thị có bậc bằng 3.*

Chương 6

Algorithms on Graphs – Các Thuật Toán Trên Đồ Thị

6.1 Breadth-first Search (BFS) – Tìm Kiếm Theo Chiều Rộng

Resources – Tài nguyên.

1. Wikipedia/breadth-1st search.

2. VNOI Wiki/ BFS (breadth-1st search).

Breadth-first search (BFS) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root & explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.

E.g., in a chess endgame, a chess engine may build the game tree from the current position by applying all possible moves & use BFS to find a win position for White. Implicit trees (e.g. game trees or other problem-solving trees) may be of infinite size; BFS is guaranteed to find a solution node (i.e., a node satisfying the specified property) if one exists.

– *Tìm kiếm theo chiều rộng (BFS)* là 1 thuật toán tìm kiếm 1 cấu trúc dữ liệu cây cho 1 nút thỏa mãn 1 thuộc tính nhất định. Thuật toán này bắt đầu từ gốc cây & khám phá tất cả các nút ở độ sâu hiện tại trước khi chuyển sang các nút ở mức độ sâu tiếp theo. Cần thêm bộ nhớ, thường là hàng đợi, để theo dõi các nút con đã gặp nhưng chưa được khám phá.

In contrast, (plain) DFS, which explores the node branch as far as possible before backtracking & expanding other nodes, may get lost in an infinite branch & never make it to the solution node. Iterative deepening DFS avoids the latter drawback at the price of exploring the tree's t op parts over & over again. On the other hand, both depth-1st algorithms typically require far less extra memory than BFS.

BFS can be generalized to both undirected graphs & directed graphs with a given start node (sometimes referred to as a 'search key'). In state space search in AI, repeated searches of vertices are often allowed, while in theoretical analysis of algorithms based on BFS, precautions are typically taken to prevent repetitions.

– BFS có thể được khái quát hóa thành cả đồ thị vô hướng & đồ thị có hướng với 1 nút bắt đầu nhất định (đôi khi được gọi là 'khóa tìm kiếm'). Trong tìm kiếm không gian trạng thái trong AI, việc tìm kiếm lặp lại các đỉnh thường được phép, trong khi trong phân tích lý thuyết các thuật toán dựa trên BFS, các biện pháp phòng ngừa thường được thực hiện để ngăn ngừa sự lặp lại.

BFS & its application in finding connected components of graphs were invented in 1945 by Konrad Zuse, in this (rejected) PhD thesis on the Plankalkül programming language, but this was not published until 1972. It was reinvented in 1959 by Edward F. Moore, who used it to find the shortest path out of a maze, & later developed by C. Y. Lee into a wire routing algorithm (published in 1961).

– BFS & ứng dụng của nó trong việc tìm các thành phần liên thông của đồ thị được Konrad Zuse phát minh vào năm 1945, trong luận án tiến sĩ (bị từ chối) này về ngôn ngữ lập trình Plankalkül, nhưng luận án này không được công bố cho đến năm 1972. Nó được Edward F. Moore phát minh lại vào năm 1959, người đã sử dụng nó để tìm đường đi ngắn nhất ra khỏi mê cung, & sau đó được C. Y. Lee phát triển thành thuật toán định tuyến có dây (được công bố vào năm 1961).

6.1.1 Pseudocode of BFS

- Input. A graph $G = (V, E)$ & a starting vertex *root* of G.
- Output. Goal state. The *parent* links trace the shortest path back to *root*

```
procedure BFS(G, root) is
    let Q be a queue
    label root as explored
    Q.enqueue(root)
    while Q is not empty do
        v := Q.dequeue()
        if v is the goal then
            return v
        for all edges from v to w in G.adjacentEdges(v) do
            if w is not labeled as explored then
                label w as explored
                w.parent := v
                Q.enqueue(w)
```

This non-recursive implementation is similar to the non-recursive implementation of DFS, but differs from it in 2 ways:

1. it uses a queue (First In First Out, abbr., FIFO) instead of a stack (Last In First Out, abbr., LIFO) &
2. it checks whether a vertex has been explored before enqueueing vertex rather than delaying this check until the vertex is dequeued from the queue.

If G is a tree, replacing the queue of this BFS algorithm with a stack will yield a DFS algorithm. For general graphs, replacing the stack of the iterative DFS implementation with a queue would also produce a DFS algorithm, although a somewhat nonstandard one.

– Triển khai không đệ quy này tương tự như triển khai không đệ quy của DFS, nhưng khác ở 2 điểm:

1. sử dụng hàng đợi (First In First Out, viết tắt là FIFO) thay vì ngăn xếp (Last In First Out, viết tắt là LIFO) &
2. kiểm tra xem đỉnh đã được khám phá trước khi đưa đỉnh vào hàng đợi hay chưa thay vì trì hoãn việc kiểm tra này cho đến khi đỉnh được đưa ra khỏi hàng đợi.

Nếu G là 1 cây, việc thay thế hàng đợi của thuật toán BFS này bằng 1 ngăn xếp sẽ tạo ra 1 thuật toán DFS. Đối với đồ thị tổng quát, việc thay thế ngăn xếp của triển khai DFS lặp lại bằng 1 hàng đợi cũng sẽ tạo ra 1 thuật toán DFS, mặc dù là 1 thuật toán không chuẩn.

The Q queue contains the frontier along which the algorithm is currently searching.

– Hàng đợi Q chứa biên giới (tiền tuyến?) mà thuật toán hiện đang tìm kiếm.

Nodes can be labeled as explored by storing them in a set, or by an attribute on each node, depending on the implementation.

– Các nút có thể được gắn nhãn là đã khám phá bằng cách lưu trữ chúng trong 1 tập hợp hoặc bằng thuộc tính trên mỗi nút, tùy thuộc vào cách triển khai.

Note: the word *node* is usually interchangeable with the word *vertex*.

– Lưu ý: từ nút thường có thể thay thế cho từ đỉnh.

The *parent* attribute of each node is useful for accessing the nodes in a shortest path, e.g., by backtracking from the destination node up to the starting node, once the BFS has been run, & the predecessors nodes have been set.

– Thuộc tính cha mẹ của mỗi nút rất hữu ích để truy cập các nút theo đường dẫn ngắn nhất, ví dụ, bằng cách quay lại từ nút đích đến nút bắt đầu sau khi BFS đã chạy và các nút tiền nhiệm đã được thiết lập.

BFS produces a so-called *breadth 1st tree*, e.g., an example of breath-1st tree obtained by running a BFS on German cities starting from Frankfurt.

6.1.2 Analysis of BFS – Phân tích BFS

6.1.2.1 Time & space complexity of BFS – Độ phức tạp không gian & thời gian của BFS

The time complexity can be expressed as $O(|V| + |E|)$, since every vertex & every edge will be explored in the worst case. Note: $O(|E|)$ may be vary between $O(1)$ & $O(|V|^2)$, depending on how sparse the input graph is.

– Độ phức tạp về thời gian có thể được biểu thị là $O(|V| + |E|)$, vì mọi đỉnh & mọi cạnh sẽ được khám phá trong trường hợp xấu nhất. Lưu ý: $O(|E|)$ có thể thay đổi giữa $O(1)$ & $O(|V|^2)$, tùy thuộc vào mức độ thưa thớt của đồ thị đầu vào.

When the number of vertices in the graph is known ahead of time, & additional data structures are used to determine which vertices have already been added to the queue, the space complexity can be expressed as $O(|V|)$, where $|V|$ is the number of vertices. This is in addition to the space required for the graph itself, which may vary depending on the graph representation used by an implementation of the algorithm.

– Khi số lượng đỉnh trong đồ thị được biết trước, & các cấu trúc dữ liệu bổ sung được sử dụng để xác định những đỉnh nào đã được thêm vào hàng đợi, độ phức tạp không gian có thể được biểu thị là $O(|V|)$, trong đó $|V|$ là số lượng đỉnh. Đây là phần bổ sung cho không gian cần thiết cho chính đồ thị, có thể thay đổi tùy thuộc vào biểu diễn đồ thị được sử dụng bởi 1 triển khai của thuật toán.

When working with graphs that are too large to store explicitly (or infinite), it is more practical to describe the complexity of BFS in different terms: to find the nodes that are at distance d from the start node (measured in number of edge traversals), BFS takes $O(b^{d+1})$ time & memory, where b is the "branching factor" of the graph (the average out-degree).

– Khi làm việc với các đồ thị quá lớn để lưu trữ rõ ràng (hoặc vô hạn), thì thực tế hơn khi mô tả độ phức tạp của BFS theo các thuật ngữ khác nhau: để tìm các nút cách nút bắt đầu ở khoảng cách d (được đo bằng số lần duyệt cạnh), BFS mất $O(b^{d+1})$ thời gian & bộ nhớ, trong đó b là "hệ số phân nhánh" của đồ thị (bậc ra trung bình).

6.1.2.2 Completeness of BFS – Tính đầy đủ của BFS

In the analysis of algorithms, the input to BFS is assumed to be a finite graph, represented as an adjacency list, adjacency matrix, or similar representation. However, in the application of graph traversal methods in AI the input may be an implicit representation of an infinite graph. In this context, a search method is described as being *complete* if it is guaranteed to find a goal state if one exists. BFS is complete, but DFS is not. When applied to infinite graphs represented implicitly, BFS will eventually find the goal state, but DFS may get lost in parts of the graph that have no goal state & never return.

– Trong phân tích thuật toán, đầu vào của BFS được coi là 1 đồ thị hữu hạn, được biểu diễn dưới dạng danh sách kề, ma trận kề hoặc biểu diễn tương tự. Tuy nhiên, trong ứng dụng của các phương pháp duyệt đồ thị trong AI, đầu vào có thể là biểu diễn ngầm của 1 đồ thị vô hạn. Trong bối cảnh này, 1 phương pháp tìm kiếm được mô tả là hoàn chỉnh nếu nó được đảm bảo tìm thấy trạng thái mục tiêu nếu có. BFS là hoàn chỉnh, nhưng DFS thì không. Khi áp dụng cho các đồ thị vô hạn được biểu diễn ngầm, BFS cuối cùng sẽ tìm thấy trạng thái mục tiêu, nhưng DFS có thể bị mất trong các phần của đồ thị không có trạng thái mục tiêu & không bao giờ trả về.

Intuition 1 ((BFS vs. DFS) vs. (Wide learning vs. Deep learning)). *Việc so sánh giữa BFS vs. DFS na ná với việc học rộng & học sâu. 1 cách nôm na, 1 người học rộng sẽ tìm được mục tiêu cá nhân, bù lại là người đó sẽ mất nhiều thời gian. Còn 1 người học sâu, nếu không nhìn big picture hay big map sẽ dễ bị mắc kẹt, trapped ở những chỗ sâu tối tăm mà không nhận ra mục tiêu của mình nằm ở 1 chỗ khác, thậm chí là 1 lĩnh vực khác.*

6.1.3 BFS ordering – Thứ tự BFS

Definition 33 (BFS ordering). *An enumeration of the vertices of a graph is said to be a* BFS ordering *if it is the possible output of the application of BFS to this graph.*

Let $G = (V, E)$ be a graph with $n \in \mathbb{N}^\star$ vertices. Recall that $N(v)$ is the set of neighbors of v. Let $\sigma = (v_1, \dots, v_m)$ be a list of distinct elements of V, for $v \in V \backslash \{v_1, \dots, v_m\}$, let $\nu_\sigma(v)$ be the least i s.t. v_i is a neighbor of v, if such a i exists, & be ∞ otherwise:

$$\nu_\sigma(v) = \begin{cases} \min\{i; v_i \in N(v)\} & \text{if } V \cap N(v) \neq \emptyset, \\ \infty & \text{if } V \cap N(v) = \emptyset. \end{cases}$$

Let $\sigma = (v_1, \dots, v_n)$ be an enumeration of the vertices of V. The enumeration σ is said to be a *BFS ordering* (with source v_1) if, $\forall i = 2, 3, \dots, n$, v_i is the vertex $w \in V \backslash \{v_1, \dots, v_{i-1}\}$ s.t. $\nu_{(v_1,\dots,v_{i-1})}(w)$ is minimal. Equivalently, σ is a BFS ordering if, for all $1 \leq i < j < k \leq n$ with $v_i \in N(v_k) \backslash N(v_j)$, there exists a neighbor v_m of v_j s.t. $m < i$.

6.1.4 Some applications of BFS – Vài ứng dụng của BFS

BFS can be used to solve many problems in graph theory, e.g.:

1. Copying garbage collection, Cheney's algorithm
2. Finding the shortest path between 2 nodes u, v, with path length measured by number of edges (an advantage over DFS)
3. (Reverse) Cuthill–McKee mesh numbering
4. Ford–Fulkerson method for computing the maximum flow in a flow network
5. Serialization/Deserialization of a binary tree vs. serialization in sorted order, allows the tree to be reconstructed in an efficient manner
6. Construction of the *failure function* of the Aho-Corasick pattern matcher
7. Testing bipartiteness of a graph
8. Implementing parallel algorithms for computing a graph's transitive closure.

6.2 Depth-first Search (DFS) – Tìm Kiếm Theo Chiều Sâu

Resources – Tài nguyên.

1. Wikipedia/depth-1st search.

2. VNOI Wiki/cây DFS (depth-1st search tree) & ứng dụng.

Depth-1st search (DFS) is an algorithm for traversing or search tree or graph data structures.The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) & explores as far as possible along each branch before backtracking. Extra memory, usually a stack, is needed to keep track of the nodes discovered so far along a specified branch which helps in backtracking of the graph.

– Tìm kiếm theo chiều sâu (DFS) là 1 thuật toán để duyệt hoặc tìm kiếm các cấu trúc dữ liệu cây hoặc đồ thị. Thuật toán bắt đầu tại nút gốc (chọn 1 số nút tùy ý làm nút gốc trong trường hợp đồ thị) và khám phá càng xa càng tốt dọc theo mỗi nhánh trước khi quay lại. Bộ nhớ bổ sung, thường là 1 ngăn xếp, là cần thiết để theo dõi các nút đã phát hiện cho đến nay dọc theo 1 nhánh được chỉ định, giúp quay lại đồ thị.

A version of depth-1st was investigated in the 19th century by French mathematician Charles Pierre Trémaux as a strategy for solving mazes.

6.2.1 Some properties of DFS – Vài tính chất của DFS

The time & space analysis of DFS differs according to its application area. In theoretical computer science, DFS is typically used to traverse an entire graph, & takes time $O(|V| + |E|)$, where $|V|$ is is the number of vertices & $|E|$ the number of edges. This is linear in the size of the graph. In these applications it also uses space $O(|V|)$ in the worst case to store the stack of vertices on the current search path as well as the set of already-visited vertices. Thus, in this setting, the time & space bounds are the same as for breadth-1st search & the choice of which of these 2 algorithms to use depends less on their complexity & more on the different properties of the vertex orderings the 2 algorithms produce.

– Phân tích thời gian & không gian của DFS khác nhau tùy theo lĩnh vực ứng dụng của nó. Trong khoa học máy tính lý thuyết, DFS thường được sử dụng để duyệt toàn bộ đồ thị, & mất thời gian $O(|V| + |E|)$, trong đó $|V|$ là số đỉnh & $|E|$ là số cạnh. Đây là tuyến tính trong kích thước của đồ thị. Trong các ứng dụng này, nó cũng sử dụng không gian $O(|V|)$ trong trường hợp xấu nhất để lưu trữ ngăn xếp các đỉnh trên đường dẫn tìm kiếm hiện tại cũng như tập hợp các đỉnh đã được truy cập. Do đó, trong cài đặt này, giới hạn thời gian & không gian giống như đối với breadth-1st search & việc lựa chọn sử dụng thuật toán nào trong 2 thuật toán này phụ thuộc ít hơn vào độ phức tạp của chúng & nhiều hơn vào các thuộc tính khác nhau của thứ tự đỉnh mà 2 thuật toán tạo ra.

For applications of DFS in relation to specific domains, e.g. searching for solutions in AI or web-crawling, the graph to be traversed is often either too large to visit in its entirety or infinite (DFS may suffer from non-termination). In such cases, search is only performed to a limited depth; due to limited resources, e.g. memory or disk space, one typically does not use data structures to keep track of the set of all previously visited vertices. When search is performed to a limited depth, the time is still linear in terms of the number of expanded vertices & edges (although this number is not the same as the size of the entire graph because some vertices may be searched more than once & others not at all) but the space complexity of this variant of DFS is only proportional to the depth limit, & as a result, is much smaller than the space needed for searching to the same depth using breadth-1st search. For such applications, DFS also lends itself much better to heuristic methods for choosing a likely-looking branch. When an appropriate depth limit is not known a priori, iterative deepening depth-1st search applies DFS repeatedly with a sequence of increasing limits. In the AI mode of analysis, with a branching factor > 1, iterative deepening increases the running time by only a constant factor over the case in which the correct depth limit is known due to the geometric growth of the number of nodes per level.

– Đối với các ứng dụng của DFS liên quan đến các miền cụ thể, ví dụ như tìm kiếm các giải pháp trong AI hoặc thu thập dữ liệu trên web, đồ thị cần duyệt thường quá lớn để duyệt toàn bộ hoặc vô hạn (DFS có thể bị lỗi không kết thúc). Trong những trường hợp như vậy, tìm kiếm chỉ được thực hiện ở độ sâu hạn chế; do tài nguyên hạn chế, ví dụ như bộ nhớ hoặc dung lượng đĩa, người ta thường không sử dụng các cấu trúc dữ liệu để theo dõi tập hợp tất cả các đỉnh đã truy cập trước đó. Khi tìm kiếm được thực hiện ở độ sâu hạn chế, thời gian vẫn tuyến tính theo số lượng đỉnh mở rộng & cạnh (mặc dù số này không giống với kích thước của toàn bộ đồ thị vì 1 số đỉnh có thể được tìm kiếm nhiều hơn 1 lần & 1 số đỉnh khác thì không) nhưng độ phức tạp về không gian của biến thể DFS này chỉ tỷ lệ thuận với giới hạn độ sâu, & do đó, nhỏ hơn nhiều so với không gian cần thiết để tìm kiếm ở cùng độ sâu khi sử dụng tìm kiếm theo chiều rộng thứ nhất. Đối với các ứng dụng như vậy, DFS cũng phù hợp hơn nhiều với các phương pháp tìm kiếm theo kinh nghiệm để chọn 1 nhánh có vẻ khả thi. Khi không biết trước giới hạn độ sâu thích hợp, tìm kiếm theo chiều sâu lặp lại lần thứ nhất áp dụng DFS nhiều lần với 1 chuỗi giới hạn tăng dần. Trong chế độ phân tích AI, với hệ số phân nhánh > 1, việc lặp lại lần thứ nhất làm tăng thời gian chạy chỉ bằng 1 hệ số hằng số trong trường hợp giới hạn độ sâu chính xác được biết do sự tăng trưởng theo hình học của số lượng nút trên mỗi cấp.

DFS may also be used to collect a sample of graph nodes. However, incomplete DFS, similarly to incomplete BFS, is biased towards nodes of high degree.

– DFS cũng có thể được sử dụng để thu thập mẫu các nút đồ thị. Tuy nhiên, DFS không đầy đủ, tương tự như BFS không đầy đủ, thiên về các nút có bậc cao.

Problem 57 (DFS for binary trees)**.** *Describe DFS for: (a) An arbitrary binary tree. (b) A fully binary tree of height* $n \in \mathbb{N}^\star$.

Problem 58. *Describe DFS for Trémaux tree – a structure with important applications in graph theory. Are there cases in which there are some vertices that cannot be reached by DFS?*

Problem 59 (Iterative deepening, R)**.** *Study iterative deepning – a technique to avoid possible infinite loops in applying DFS toTrémaux trees in order to be able to reach all nodes.*

6.2.2 Output of a DFS – Kết quả đầu ra của DFS

The result of a DFS of a graph can be conveniently described in terms of a spanning tree of the vertices reached during the search. Based on this spanning tree, the edges of the original graph can be divided into 3 classes:

- *Forward edges*, which point from a node of the tree to 1 of its descendants
- *Back edges*, which point from a node to 1 of its ancestors
- *Cross edges*, which do neither.
- Sometimes *tree edges*, edges which belong to the spanning tree itself, are classified separately from forward edges.

If the original graph is undirected then all of its edges are tree edges or back edges.

– Kết quả của DFS của 1 đồ thị có thể được mô tả 1 cách thuận tiện theo spanning tree của các đỉnh đạt được trong quá trình tìm kiếm. Dựa trên cây khung này, các cạnh của đồ thị gốc có thể được chia thành 3 lớp:

- *Các cạnh tiến*, trỏ từ 1 nút của cây đến 1 trong các con cháu của nó
- *Các cạnh lùi*, trỏ từ 1 nút đến 1 trong các tổ tiên của nó
- *Các cạnh chéo*, không làm cả hai.
- Đôi khi *các cạnh cây*, các cạnh thuộc về chính cây khung, được phân loại riêng biệt với các cạnh tiến.

Nếu đồ thị gốc không có hướng thì tất cả các cạnh của nó đều là các cạnh cây hoặc các cạnh lùi.

6.2.2.0.1 Vertex orderings. It is also possible to use DFS to linearly order the vertices of a graph or tree. There are 4 possible ways of doing this:

- A *preordering* is a list of the vertices in the order that they were 1st visited by the DFS algorithm. This is a compact & natural way of describing the progress of the search. A preordering of an expression tree is the expression in Polish notation.
- A *postordering* is a list of the vertices in the order that they were *last* visited by the algorithm. A postordering of an expression tree is the expression in reverse Polish notation.
- A *reverse preordering* is the reverse of a preordering, i.e., a list of the vertices in the opposite order of their 1st visit. Reverse preordering is not the same as postordering.
- A *reverse postordering* is the reverse of a postordering, i.e., a list of the vertices in the opposite order of their last visit. Reverse postordering is not the same as preordering.

For binary trees there is additionally in-ordering & reverse in-ordering.

– Cũng có thể sử dụng DFS để sắp xếp tuyến tính các đỉnh của đồ thị hoặc cây. Có 4 cách có thể thực hiện việc này:

- A *preordering* là danh sách các đỉnh theo thứ tự mà thuật toán DFS đã truy cập lần đầu tiên. Đây là cách & tự nhiên & cô đọng để mô tả tiến trình tìm kiếm. A preordering của expression tree là biểu thức trong ký hiệu Ba Lan.
- A *postordering* là danh sách các đỉnh theo thứ tự mà thuật toán đã truy cập *lần cuối*. A postordering của cây biểu thức là biểu thức trong ký hiệu Ba Lan ngược.
- A *sắp xếp ngược thứ tự trước* là đảo ngược của sắp xếp trước, tức là danh sách các đỉnh theo thứ tự ngược lại với lần truy cập đầu tiên của chúng. Sắp xếp ngược thứ tự trước không giống với sắp xếp sau.
- A *sắp xếp ngược thứ tự sau* là đảo ngược của sắp xếp sau, tức là danh sách các đỉnh theo thứ tự ngược lại với lần truy cập cuối cùng của chúng. Sắp xếp ngược thứ tự sau không giống với sắp xếp trước.

Đối với cây nhị phân còn có sắp xếp trong & sắp xếp ngược thứ tự trong.

Reverse postordering produces a topological sorting of any directed acyclic graph. This ordering is also useful in control-flow analysis as it often represents a natural linearization of the control flows.

– Sắp xếp ngược sau tạo ra 1 sắp xếp tôpô của bất kỳ đồ thị có hướng không có chu trình nào. Sắp xếp này cũng hữu ích trong phân tích luồng điều khiển vì nó thường biểu diễn 1 tuyến tính hóa tự nhiên của luồng điều khiển.

6.2.2.0.2 Pseudocode of DFS.

A recursive implementation of DFS:

```
procedure DFS(G, v) is
    label v as discovered
    for all directed edges from v to w that are in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G, w)
```

A non-recursive implementation of DFS with worst-case space complexity $O(|E|)$, with the possibility of duplicate vertices on the stack:

```
procedure DFS_iterative(G, v) is
    let S be a stack
    S.push(v)
    while S is not empty do
        v = S.pop()
        if v is not labeled as discovered then
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```

These 2 variations of DFS visit the neighbors of each vertex in the opposite order from each other: the 1st neighbor of v visited by the recursive variation is the 1st one in the list of adjacent edges, while in the iteration variation the 1st visited neighbor is the last one in the list of adjacent edges.

– 2 biến thể DFS này sẽ thăm các đỉnh lân cận của mỗi đỉnh theo thứ tự ngược nhau: hàng xóm thứ nhất của v được thăm bởi biến thể đệ quy là hàng xóm thứ nhất trong danh sách các cạnh liền kề, trong khi ở biến thể lặp, hàng xóm thứ nhất được thăm là hàng xóm cuối cùng trong danh sách các cạnh liền kề.

The non-recursive implementation is similar to BFS but differs from it in 2 ways:

1. it uses a stack instead of a queue, &
2. it delays checking whether a vertex has been discovered until the vertex is popped from the stack rather than making this check before adding the vertex.

– Triển khai không đệ quy tương tự như BFS nhưng khác ở 2 điểm:

1. sử dụng ngăn xếp thay vì hàng đợi, &
2. trì hoãn việc kiểm tra xem đỉnh đã được phát hiện hay chưa cho đến khi đỉnh đó được bật ra khỏi ngăn xếp thay vì thực hiện kiểm tra này trước khi thêm đỉnh.

If G is a tree, replacing the queue of the BFS algorithm with a stack will yield a DFS algorithm. For general graphs, replacing the stack of the iterative DFS implementation with a queue would also produce a BFS algorithm, although a somewhat nonstandard one.

– Nếu G là 1 cây, việc thay thế hàng đợi của thuật toán BFS bằng 1 ngăn xếp sẽ tạo ra 1 thuật toán DFS. Đối với đồ thị chung, việc thay thế ngăn xếp của triển khai DFS lặp lại bằng 1 hàng đợi cũng sẽ tạo ra 1 thuật toán BFS, mặc dù là 1 thuật toán không chuẩn.

Another possible implementation of iterative DFS uses a stack of iterators of the list of neighbors of a node, instead of a stack of nodes. This yields the same traversal as recursive DFS.

```
procedure DFS_iterative(G, v) is
    let S be a stack
    label v as discovered
    S.push(iterator of G.adjacentEdges(v))
    while S is not empty do
        if S.peek().hasNext() then
        w = S.peek().next()
```

```
if w is not labeled as discovered then
    label w as discovered
    S.push(iterator of G.adjacentEdges(w))
else
    S.pop()
```

6.2.3 Applications of DFS

Algorithms that use DFS as a building block include:

- Finding connected components.
- Topological sorting.
- Finding 2-(edge or vertex)-connected components.
- Finding 3-(edge or vertex)-connected components.
- Finding the bridges of a graph.
- Generating words in order to plot the limit set of a group.
- Finding strongly connected components.
- Determining whether a species is closer to 1 species or another in a phylogenetic tree.
- Planarity testing.
- Solving puzzles with only 1 solution, e.g. mazes. (DFS can be adapted to find all solutions to a maze by only including nodes on the current path in the visited set.)
- Maze generation may use a randomized DFS.
- Finding biconnectivity in graphs.
- Succession to the throne shared by the Commonwealth realms.

6.2.4 Complexity of DFS

The computational complexity of DFS was investigated by John Reif. More precisely, given a graph G, let $O = (v_1, \ldots, v_n)$ be the ordering computed by the standard recursive DFS algorithm. This ordering is called the *lexicographic DFS ordering*. John Reif considered the complexity of computing the lexicographic DFS ordering, given a graph & a source. A decision version of the problem (testing whether some vertex u occurs before some vertex v in this order) is P-complete, i.e., it is "a nightmare for parallel processing".

– Độ phức tạp tính toán của DFS đã được John Reif nghiên cứu. Chính xác hơn, với 1 đồ thị G, hãy để $O = (v_1, \ldots, v_n)$ là thứ tự được tính toán bởi thuật toán DFS đệ quy chuẩn. Thứ tự này được gọi là *thứ tự DFS theo từ điển*. John Reif đã xem xét độ phức tạp của việc tính toán thứ tự DFS theo từ điển, với 1 đồ thị & 1 nguồn. Một phiên bản quyết định của bài toán (kiểm tra xem 1 số đỉnh u có xuất hiện trước 1 số đỉnh v theo thứ tự này hay không) là P-hoàn chỉnh, tức là, nó là "cơn ác mộng đối với xử lý song song".

A DFS ordering (not necessarily the lexicographic one), can be computed by a randomized parallel algorithm in the complexity class RNC. As of 1997, it remained unknown whether a depth-1st traversal could be constructed by a deterministic parallel algorithm, in the complexity class NC.

6.3 Shortest path problem – Bài toán tìm đường đi ngắn nhất

In graph theory, the *shortest path problem* is the problem of finding a path between 2 vertices (or nodes) in a graph s.t. the sum of weights of its constituent edges is minimized.

The problem of finding the shortest path between 2 intersections on a road map may be modeled as a special case of the shortest path problem in graphs, where the vertices correspond to intersections & the edges correspond to road segments, each weighted by the length or distance of each segment.

The shortest path problem can be defined for graphs whether undirected, directed, or mixed. The definition for undirected graphs states that every edge can be traversed in either direction. Directed graphs require that consecutive vertices be connected by an appropriate directed edge.

Recall that 2 vertices are adjacent when they are both incident to a common edge. A *path* in an undirected graph is a sequence of vertices $P = (v_1, v_2, \ldots, v_n) \in V^n = V \times V \times \cdots \times V$ s.t. v_i is adjacent to v_{i+1}, $\forall i = 1, \ldots, n-1$. Such a path P is called a path of length $n-1$ from v_1 to v_n. (The v_i are variables; their numbering relates to their position in the sequence & need not relate to a canonical labeling.)

Let $E = \{e_{i,j}\}$ where $e_{i,j}$ is the edge incident to both v_i & v_j. Given a real-valued weight function $f : E \to \mathbb{R}$, & an undirected simple graph G, the shortest path from v to v' is the path $P = (v_1, \ldots, v_n)$ (where $v_1 = v$ & $v_n = v'$) that over all possible n minimizes the sum $\sum_{i=1}^{n-1} f(e_{i,i+1})$. When each edge in the graph has unit weight or $f : E \to \{1\}$, this is equivalent to finding the path with fewest edges.

$$\min_{P=(v_1,\ldots,v_n)\in V^n:\text{path},\, v_1=v,\, v_n=v'} \sum_{i=1}^{n} f(e_{i,i+1}).$$

The problem is also sometimes called the *single-pair shortest path problem*, to distinguish it from the following variations:

- The *single-source shortest path problem*, in which we have to find shortest paths from a source vertex v to all other vertices in the graph.

$$\min_{P=(v_1,\ldots,v_n)\in V^n:\text{path},\, v_1=v} \sum_{i=1}^{n} f(e_{i,i+1}).$$

- The *single-destination shortest path problem*, in which we have to find shortest paths from all vertices in the directed graph to a single destination vertex v. This can be reduced to the single-source shortest path problem by reversing the arcs in the directed graph.

$$\min_{P=(v_1,\ldots,v_n)\in V^n:\text{path},\, v_n=v'} \sum_{i=1}^{n} f(e_{i,i+1}).$$

- The *all-pairs shortest path problem*, in which we have to find shortest paths between every pair of vertices v, v' in the graph.

$$\min_{P=(v_1,\ldots,v_n)\in V^n:\text{path}} \sum_{i=1}^{n} f(e_{i,i+1}).$$

These generalizations have significantly more efficient algorithms than the simplistic approach of running a single-pair shortest path algorithm on all relevant pairs of vertices.

– Những tổng quát hóa này có thuật toán hiệu quả hơn đáng kể so với cách tiếp cận đơn giản là chạy thuật toán đường dẫn ngắn nhất 1 cặp trên tất cả các cặp đỉnh có liên quan.

6.3.1 Algorithms

Several well-known algorithms exist for solving the shortest path problem & its variants.

- Dijkstra's algorithm solves the single-source shortest path problem with only nonnegative edge weights.
- Bellman–Ford algorithm solves the single-source problem if edge weights may be negative.
- A* search algorithm solves for single-pair shortest path using heuristics to try to speed up the search.
- Floyd–Warshall algorithm solves all pairs shortest paths.
- Johnson's algorithm solves all pairs shortest paths, & may be faster than Floyd–Warshall on sparse graphs.
- Viterbi algorithm solves the shortest stochastic path problem with an additional probabilistic weight on each node.

6.4 Dijkstra's algorithm – Thuật toán Dijkstra

"*Dijkstra's algorithm* is an algorithm for finding the shorest paths between nodes in a weighted graph, which may represent, e.g., a road network. It was conceived by computer scientist Edsger W. Dijkstra in 1956 & published 3 years later.

– *Thuật toán Dijkstra* là 1 thuật toán để tìm đường đi gần bờ nhất giữa các nút trong 1 đồ thị có trọng số, có thể biểu diễn, ví dụ, 1 mạng lưới đường bộ. Thuật toán này được nhà khoa học máy tính Edsger W. Dijkstra nghĩ ra vào năm 1956 & xuất bản 3 năm sau đó.

Dijkstra's algorithm finds the shortest path from a given source code to every node. It can be used to find the shortest path to a specific destination node, by terminating the algorithm after determining the shortest path to the destination node. E.g., if nodes of the graph represents cities, & the costs of edges represent the distances between pairs of cities connected by a direct road, then Dijkstra's algorithm can be used to find the shortest route between 1 city & all other cities. A common application

of shortest path algorithms is network routing protocols, most notably IS-IS (Intermediate System to Intermediate System) & OSPF (Open Shortest Path 1st). It is also employed as a subroutine in algorithms e.g. Johnson's algorithm.

– Thuật toán Dijkstra tìm đường đi ngắn nhất từ 1 mã nguồn nhất định đến mọi nút. Thuật toán này có thể được sử dụng để tìm đường đi ngắn nhất đến 1 nút đích cụ thể, bằng cách kết thúc thuật toán sau khi xác định được đường đi ngắn nhất đến nút đích. Ví dụ, nếu các nút của đồ thị biểu diễn các thành phố, & chi phí của các cạnh biểu diễn khoảng cách giữa các cặp thành phố được kết nối bằng 1 con đường trực tiếp, thì thuật toán Dijkstra có thể được sử dụng để tìm tuyến đường ngắn nhất giữa 1 thành phố & tất cả các thành phố khác. 1 ứng dụng phổ biến của thuật toán đường đi ngắn nhất là các giao thức định tuyến mạng, đáng chú ý nhất là IS-IS (Hệ thống trung gian đến Hệ thống trung gian) & OSPF (Đường dẫn ngắn nhất mở thứ nhất). Thuật toán này cũng được sử dụng như 1 chương trình con trong các thuật toán, ví dụ như thuật toán Johnson.

Chương 7

CSES Problem Set/Graph Algorithms

Problem 60 (CSES Problem Set/counting rooms, https://cses.fi/problemset/task/1192)**.** *You are given a map of a building, & your task is to count the number of its rooms. The size of the map is $n \times m$ squares, & each square is either floor or wall. You can walk left, right, up, & down through the floor squares.*

Input. *The 1st input lines has 2 integers n, m: the height & width of the map. Then there are n lines of m characters describing the map. Each character is either* `.` *(floor) or* `#` *(wall).*

Output. Print 1 integer: the number of rooms.

Constraints. $1 \leq n, m \leq 10^3$.

Sample. *Input:*

```
5 8
########
#..#...#
####.#.#
#..#...#
########
```

Output: `3`.

Problem 61 (CSES Problem Set/labyrinth, https://cses.fi/problemset/task/1193)**.** *You are given a map of a labyrinth, task: find a path from start to end. You can walk left, right, up, & down.*

Input. *The 1st input line has 2 integers n, m: the height & width of the map. Then there are n lines m characters describing the labyrinth. Each character is* `.` *(floor),* `#` *(wall),* `A` *(start), or* `B` *(end). There is exactly 1* `A` *& 1* `B` *in the input.*

Output. *1st print* `YES`*, if there is a path, &* `No` *otherwise. If there is a path, print the length of the shortest such path & its description as a string consisting of characters* `L` *(left),* `R` *(right),* `U` *(up), &* `D` *(down). You can print any valid solution.*

Constraints. $1 \leq n, m \leq 1000$.

Sample. *Input:*

```
5 8
########
#.A#...#
#.##.#B#
#......#
########
```

Output:

```
YES
9
LDDRRRRRU
```

Problem 62 (CSES Problem Set/building roads, https://cses.fi/problemset/task/1666). *Byteland has n cities, & m roads between them. Goal: construct new roads so that there is a route between any 2 cities. Task: find out the minimum number of roads required, & also determine which roads should be built.*

Input. *The 1st input line has 2 integers n, m: the number of cities & roads. The cities are numbered $1, 2, \ldots, n$. After that, there are m lines describing the roads. Each lines has 2 integers a, b: there is a road between those cities. A road always connects 2 different cities, & there is at most 1 road between any 2 cities.*

Output. *1st print an integer k: the number of required roads. Then, print k lines that describe the new roads. You can print any valid solution.*

Constraints. $1 \le n \le 10^5, 1 \le m \le 2 \cdot 10^5, 1 \le a, b \le n$.

Sample.

build_road.inp	build_road.out
4 2 1 2 3 4	1 2 3

Problem 63 (CSES Problem Set/message route, https://cses.fi/problemset/task/1667). *Syrjälä's network has n computers & m connections. Task: find out if Uolevi can send a message to Maija, & if it is possible, what is the minimum number of computers on such a route.*

Input. *The 1st input line has 2 integers n, m: the number of computers & connections. The computers are numbered $1, 2, \ldots, n$. Uolevi's computer is 1 & Maija's computer is n. Then, there are m lines describing the connections. Each line has 2 integers a, b: there is a connection between those computers. Every connection is between 2 different computers, & there is at most 1 connection between any 2 computers.*

Output. *If it is possible to send a message, 1st print k: the minimum number of computers on a valid route. After this, print an example of such a route. You can print any valid solution. If there are no routes, print* `IMPOSSIBLE`.

Constraints. $2 \le n \le 10^5, 1 \le m \le 2 \cdot 10^5, 1 \le a, b \le n$.

Sample.

message_route.inp	message_route.out
5 5 1 2 1 3 1 4 2 3 5 4	3 1 4 5

Problem 64 (CSES Problem Set/building team, https://cses.fi/problemset/task/1668). *There are n pupils in Uolevi's class, & m friendships between them. Task: divide pupils into 2 teams in such a way that no 2 pupils in a team are friends. You can freely choose the sizes of the teams.*

Input. *The 1st input line has 2 integers n, m: the number of pupils & friendships. The pupils are numbered $1, 2, \ldots, n$. Then, there are m lines describing the friendships. Each line has 2 integers a, b: pupils a, b are friends. Every friendship is between 2 different pupils. You can assume that there is at most 1 friendship between any 2 pupils.*

Output. *Print an example of how to build the teams. For each pupil, print* `1` *or* `2` *depending on to which team the pupil will be assigned. You can print any valid team. If there are no solutions, print* `IMPOSSIBLE`.

Constraints. $1 \le n \le 10^5, 1 \le m \le 2 \cdot 10^5, 1 \le a, b \le n$.

Sample.

build_team.inp	build_team.out
5 3 1 2 1 3 4 5	1 2 2 1 2

Problem 65 (CSES Problem Set/round trip, https://cses.fi/problemset/task/1669). *Byteland has n cities & m roads between them. Task: design a round trip that begins in a city, goes through 2 or more other cities, & finally returns to starting city. Every intermediate city on the route has to be distinct.*

Input. *The 1st input line has 2 integers n, m: the number of cities & roads. The cities are numbered $1, 2, \ldots, n$. Then, there are m lines describing the roads. Each line has 2 integers a, b: there is a road between those cities. Every road is between 2 different cities, & there is at most 1 road between any 2 cities.*

Output. *1st print an integer k: the number of cities on the route. Then print k cities in order they will be visited. You can print any valid solution. If there are no solutions, print* `IMPOSSIBLE`.

Constraints. $1 \leq n \leq 10^5, 1 \leq m \leq 2 \cdot 10^5, 1 \leq a, b \leq n$.

Sample.

build_team.inp	build_team.out
5 6 1 3 1 2 5 3 1 5 2 4 4 5	4 3 5 1 3

Problem 66 (CSES Problem Set/monsters, https://cses.fi/problemset/task/1194). *You & some monsters are in a labyrinth. When taking a step to some direction in the labyrinth, each monster may simultaneously take 1 as well. Goal: reach 1 of the boundary squares without ever sharing a square with a monster. Task: find out if your goal is possible, & if it is, print a path that you can follow. Your plan has to work in any situation; even if the monsters know your path beforehand.*

Input. *The 1st input line has 2 integers n, m: the height & width of the map. After this there are n lines of m characters describing the map. Each character is* `.` *(floor),* `#` *(wall),* `A` *(start), or* `M` *(monster). There is exactly 1* `A` *in the input.*

Output. *1st print* `YES` *if your goal is possible, &* `NO` *otherwise. If your goal is possible, also print an example of a valid path (the length of the path & its description using characters* `D, U, L, R`*). You can print any path, as long as its length is at most mn steps.*

Constraints. $1 \leq m, n \leq 10^3$.

Sample. *Input:*

```
5 8
########
#M..A..#
#.#.M#.#
#M#..#..
#.######
```

Output:

```
YES
5
RRDDR
```

Problem 67 (CSES Problem Set/shortest routes I, https://cses.fi/problemset/task/1671). *There are n cities & m flight connections between them. Task: determine the length of the shortest route from Syrjälä to every city.*

Input. *The 1st input line has 2 integers n, m: the number of cities & flight connections. The cities are numbered $1, 2, \ldots, n$, & city 1 is Syrjälä. After that, there are m lines describing the flight connections. Each line has 3 integers a, b, c: a flight begins at city a, ends at city b, & its length is c. Each flight is a 1-way flight. You can assume that it is possible to travel from Syrjälä to all other cities.*

Output. *Print n integers: the shortest route lengths from Syrjälä to cities $1, 2, \ldots, n$.*

Constraints. $1 \leq n \leq 10^5, 1 \leq m \leq 2 \cdot 10^5, 1 \leq a, b \leq n, 1 \leq c \leq 10^9$.

Sample.

shortest_route_I.inp	shortest_route_I.out
3 4 1 2 6 1 3 2 3 2 3 1 3 4	0 5 2

Problem 68 (CSES Problem Set/shortest routes II, https://cses.fi/problemset/task/1672). *There are n cities & m roads between them. Task: process q queries where you have to determine the length of the shortest route between 2 given cities.*

Input. *The 1st input line has 3 integers n, m, q: the number of cities, roads, & queries. Then, there are m lines describing the roads. Each line has 3 integers a, b, c: there is a road between cities a & b whose length is c. All roads are 2-way roads. Finally, there are q lines describing the queries. Each line has 2 integers a, b: determine the length of the shortest route between cities a & b.*

Output. *Print the length of the shortest route for each query. If there is no route, print* `-1` *instead.*

Constraints. $1 \le n \le 500, 1 \le m \le n^2, 1 \le q \le 10^5, 1 \le a, b \le n, 1 \le c \le 10^9$.

Sample.

shortest_route_II.inp	shortest_route_II.out
4 3 5 1 2 5 1 3 9 2 3 3 1 2 2 1 1 3 1 4 3 2	5 5 8 -1 3

Problem 69 (CSES Problem Set/high score, https://cses.fi/problemset/task/1673). *You play a game consisting of n rooms & m tunnels. Your initial score is 0, & each tunnel increases your score by x where x may be both positive or negative. You may go through a tunnel several times. Task: walk from room 1 to room n. What is the maximum score you can get?*

Input. *The 1st input line has 2 integers n, m: the number of rooms & tunnels. The rooms are numbered $1, 2, \ldots, n$. Then, there are m lines describing the tunnels. Each line has 3 integers a, b, x: the tunnel starts at room a, ends at room b, & it increases your score by x. All tunnels are 1-way tunnels. You can assume that it is possible to get from room 1 to room n.*

Output. *Print 1 integer: the maximum score you can get. However, if you can get an arbitrarily large score, print* `-1`.

Constraints. $1 \le n \le 2500, 1 \le m \le 5000, 1 \le a, b \le n, -10^9 \le x \le 10^9$.

Sample.

high_score.inp	high_score.out
4 5 1 2 3 2 4 -1 1 3 -2 3 4 7 1 4 4	5

Problem 70 (CSES Problem Set/flight discount, https://cses.fi/problemset/task/1195). *Task: find a minimum-price flight route from Syrjälä to Metsälä. You have 1 discount coupon, using which you can halve the price of any single flight during the route. However, you can only use the coupon once. When you use the discount coupon for a flight whose price is x, its price becomes $\lfloor \frac{x}{2} \rfloor$.*

Input. *The 1st input line has 2 integers n, m: the number of cities & flight connections. The cities are numbered $1, 2, \ldots, n$. City 1 is Syrjälä, & city n is Metsälä. After this there are m lines describing the flights. Each line has 3 integers a, b, c: a flight begins at city a, ends at city b, & its price is c. Each flight is unidirectional. You can assume that it is always possible to get from Syrjälä to Metsälä.*

Output. *Print 1 integer: the price of the cheapest route from Syrjälä to Metsälä.*

Constraints. $1 \le n \le 10^5, 1 \le m \le 2 \cdot 10^5, 1 \le a, b \le n, 1 \le c \le 10^9$.

Sample.

flight_discount.inp	flight_discount.out
3 4 1 2 3 2 3 1 1 3 7 2 1 5	2

Problem 71 (CSES Problem Set/cycle finding, https://cses.fi/problemset/task/1197)**.** *You are given a directed graph, & task: find out if it contains a negative cycle, & also give an example of such a cycle.*

Input. *The 1st input line has 2 integers n, m: the number of nodes & edges. The nodes are numbered $1, 2, \ldots, n$. After this, the input has m lines describing the edges. Each line has 3 integers a, b, c: there is an edge from node a to node b whose length is c.*

Output. *If the graph contains a negative cycle, print 1st* `YES`*, & then the nodes in the cycle in their correct order. If there are several negative cycles, you can print any of them. If there are no negative cycles, print* `NO`.

Constraints. $1 \le n \le 2500, 1 \le m \le 5000, 1 \le a, b \le n, -10^9 \le c \le 10^9$.

Sample.

cycle_finding.inp	cycle_finding.out
4 5 1 2 1 2 4 1 3 1 1 4 1 -3 4 3 -2	YES 1 2 4 1

Chương 8

CSES Problem Set/Tree Algorithms

Chương 9

Posets, Kết Nối, Lưới Boolean

Chương 10

Miscellaneous

10.1 Contributors

1. Võ Ngọc Trâm Anh [VNTA].
   - VNTA's [UMT_SOT_SUM25] Combinatorics & Graph Theory https://github.com/vntanh1406/Graph_SUM2025.
2. Sơn Tân [ST].
3. Phan Vĩnh Tiến [PVT].

Tài liệu tham khảo

[AD10] Titu Andreescu and Gabriel Dospinescu. *Problems from the Book*. 2nd. XYZ Press, 2010, p. 571. ISBN: 978-0979926907.

[Hal60] Paul Richard Halmos. *Naive set theory*. The University Series in Undergraduate Mathematics. D. Van Nostrand Co., Princeton, N.J.-Toronto-London-New York, 1960, pp. vii+104.

[Hal74] Paul Richard Halmos. *Naive set theory*. Undergraduate Texts in Mathematics. Reprint of the 1960 edition. Springer-Verlag, New York-Heidelberg, 1974, pp. vii+104.

[HT24] Bùi Việt Hà and Vương Trọng Thanh. *Các Vấn Đề Trong Tổ Hợp*. Nhà Xuất Bản Đại Học Quốc Gia Hà Nội, 2024, p. 429.

[Kap72] Irving Kaplansky. *Set theory and metric spaces*. Allyn and Bacon Series in Advanced Mathematics. Allyn and Bacon, Inc., Boston, Mass., 1972, pp. xii+140.

[Kap77] Irving Kaplansky. *Set theory and metric spaces*. Second. Chelsea Publishing Co., New York, 1977, xii+140 pp. ISBN 0-8284-0298–1.

[Phư10] Phạm Minh Phương. *Một Số Chuyên Đề Toán Tổ Hợp Bồi Dưỡng Học Sinh Giỏi Trung Học Phổ Thông*. Nhà Xuất Bản Giáo Dục Việt Nam, 2010, p. 179.

[Sha22] Shahriar Shahriari. *An invitation to combinatorics*. Cambridge Mathematical Textbooks. Cambridge University Press, Cambridge, 2022, pp. xv+613. ISBN: 978-1-108-47654-6.

[Thà13] Lê Công Thành. *Lý Thuyết Độ Phức Tạp Tính Toán*. Nhà Xuất Bản Khoa Học Tự Nhiên & Công Nghệ, 2013, p. 370.

[Val02] Gabriel Valiente. *Algorithms on trees and graphs*. Springer-Verlag, Berlin, 2002, pp. xiv+490. ISBN: 3-540-43550-6. DOI: 10.1007/978-3-662-04921-1. URL: https://doi.org/10.1007/978-3-662-04921-1.

[Val21] Gabriel Valiente. *Algorithms on trees and graphs—with Python code*. Texts in Computer Science. Second edition [of 1926815]. Springer, Cham, [2021] ©2021, pp. xv+386. ISBN: 978-3-303-81884-5; 978-3-303-81885-2. DOI: 10.1007/978-3-030-81885-2. URL: https://doi.org/10.1007/978-3-030-81885-2.

[Vin+25] Lê Anh Vinh, Lê Phúc Lữ, Nguyễn Huy Tùng, Trần Đăng Phúc, Vũ Văn Luân, Phạm Văn Thắng, Lê Quang Quân, Nguyễn Văn Thế, Nguyễn Tuấn Hải Đăng, Hà Hữu Cao Trình, and Phạm Việt Hùng. *Định Hướng Bồi Dưỡng Học Sinh Năng Khiếu Toán. Tập 4: Tổ Hợp*. Nhà Xuất Bản Đại Học Quốc Gia Hà Nội, 2025, pp. x+373.