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

Nguyễn Quản Bá Hồng[1]

Ngày 22 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).
- *Đo 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). *1* 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.*
– 1 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) $\sqrt{2} + \sqrt{3} \in A \overset{\text{(iii)}}{\Longrightarrow} (\sqrt{2} + \sqrt{3})^2 = 5 + 2\sqrt{6} \in A \underset{-5 \in \mathbb{Z} \subset A}{\overset{\text{(iii)}}{\Longrightarrow}} 2\sqrt{6} \in A \overset{\text{(iii)}}{\Longrightarrow} 2\sqrt{6}(\sqrt{2} + \sqrt{3}) = 4\sqrt{3} + 6\sqrt{2} \in A \Rightarrow \dfrac{1}{\sqrt{2} + \sqrt{3}} = \sqrt{3} - \sqrt{2} = 5(\sqrt{2} + \sqrt{3}) - (4\sqrt{3} + 6\sqrt{2}) \in A$.

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

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

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 Permutations & Combinations – Hoán vị & tổ hợp

Bài toán 8 (Consecutive coin toss – Gieo các đồng xu liên tiếp)**.** *Cho $n, k \in \mathbb{N}^\star$, $k \leq n$. 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, $\forall i = 1, \ldots, n$. Không gian mẫu: $|\Omega| = \prod_{i=1}^{n} 2 = 2^n$. (a) Vì chỉ có 1 trường hợp thuận lợi là $(S, S, \ldots, S)$ nên $\mathbb{P}(X_i = S,\ \forall i = 1, \ldots, n) = \mathbb{P}(|\{i; X_i = S\}| = n) = \dfrac{1}{2^n}$. Tương tự, vì chỉ có 1 trường hợp thuận lợi là $(N, N, \ldots, N)$ nên $\mathbb{P}(X_i = N,\ \forall i = 1, \ldots, n) = \mathbb{P}(|\{i; X_i = N\}| = n) = \dfrac{1}{2^n}$. (b) $\mathbb{P}(|\{i; X_i = S\}| = k) = \mathbb{P}(|\{i; X_i = N\}| = k) = \dfrac{C_n^k}{2^n}$, $\forall k = 0, \ldots, n$. (c) $\mathbb{P}(|\{i; X_i = S\}| \geq k) = \mathbb{P}(|\{i; X_i = N\}| \geq k) = \dfrac{C_n^k + C_n^{k+1} + \cdots + C_n^n}{2^n} = \dfrac{\sum_{i=k}^{n} C_n^i}{2^n}$, $\forall k = 0, \ldots, n$. (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 $\dfrac{b}{c}$: (i) tối giản. (ii) không tối giản.*

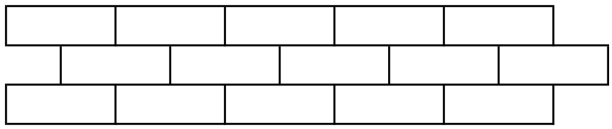
Definition 3 (Prime-counting function)**.** *The* prime-counting function *is the function counting the number of prime numbers less than or equal to some real number x, denoted by $\pi(x) \coloneqq |\{p \in \mathbb{N}^\star | p \text{ is a prime},\ p \leq x\}|$.*

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 $\pi(x) \coloneqq |\{p \in \mathbb{N}^\star | p \text{ là số nguyên tố},\ p \leq x\}|$.*

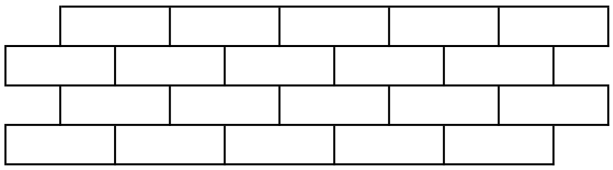
Bài toán 16 (Prime, composite – số nguyên tố, hợp số). *Cho $m, n, k \in \mathbb{N}^\star$. Đặt $A_n = \{1, 2, \dots, n\}$ là tập hợp n số nguyên dương đầu tiên, $\forall n \in \mathbb{N}^\star$. (a) Lấy m số từ A_n. Tính xác suất để m số này cùng chẵn, cùng lẻ, có ít nhất 1 số chẵn, có ít nhất 1 số 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 $A = [a, b] \cap \mathbb{Z} = \{a, a+1, a+2, \dots, b-1, b\}$. (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.

1.3.1 Permutation group – Nhóm hoán vị

Resources – Tài nguyên.

1. Wikipedia/permutation group.

Phần này liên quan đến các khái niệm của Đại Số Trừu Tượng (Abstract Algebra) & ứng dụng của chúng trong Tổ Hợp & Lý Thuyết Đồ Thị.

In mathematics, a *permutation group* is a group G whose elements are permutations of a given set M & whose group operation is the composition of permutations in G (which are thought of as bijective functions from the set M to itself). The group of *all* permutations of a set M is the symmetric group of M, often written as $\mathrm{Sym}(M)$. The term *permutation group* thus means a subgroup of the symmetric group. If $M = [n]$ then $\mathrm{Sym}(M)$ is usually denoted by S_n:

$$S_n \coloneqq \mathrm{Sym}([n]) = \mathrm{Sym}(\{1, 2, \dots, n\}), \ \forall n \in \mathbb{N}^\star,$$

& may be called the *symmetric group on n letters*.

– Trong toán học, 1 *permutation group* là 1 nhóm G có các phần tử là các hoán vị của 1 tập hợp M cho trước & có phép toán nhóm là hợp các hoán vị trong G (được coi là các hàm song ánh từ tập hợp M vào chính nó). Nhóm *all* permutation của 1 tập hợp M là nhóm đối xứng của M, thường được viết là $\text{Sym}(M)$. Do đó, thuật ngữ *permutation group* có nghĩa là 1 nhóm con của nhóm đối xứng. Nếu $M = [n]$ thì $\text{Sym}(M)$ thường được ký hiệu là S_n:

$$S_n \coloneqq \text{Sym}([n]) = \text{Sym}(\{1, 2, \dots, n\}),\ \forall n \in \mathbb{N}^\star,$$

& có thể được gọi là *nhóm đối xứng trên n chữ cái*.

By Cayley's theorem, every group is isomorphic to some permutation group.

– Theo định lý Cayley, mọi nhóm đều đồng cấu với 1 nhóm hoán vị nào đó.

The way in which the elements of a permutation group permute the elements of the set is called its group action. Group actions have applications in the study of symmetries, combinatorics, & many other branches of mathematics, physics, & chemistry.

– Cách mà các phần tử của 1 nhóm hoán vị hoán vị các phần tử của tập hợp được gọi là hành động nhóm của nó. Hành động nhóm có ứng dụng trong nghiên cứu về tính đối xứng, tổ hợp & nhiều nhánh khác của toán học, vật lý & hóa học.

1.3.1.1 Basic properties & terminology of permutation group – Tính chất cơ bản & thuật ngữ của nhóm hoán vị

***

1.3.1.2 Composition of permutations – the group product

***

1.4 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 1 ([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 2 ([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 3 ([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 4 ([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:*

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

1.5 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*

$$\sum_{i=1}^{n} x_i = m. \tag{1.1}$$

(a) Đếm số nghiệm nguyên dương của phương trình (1.1). (b) Đếm số nghiệm nguyên không âm của phương trình (1.1). (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) $C_{m+n-1-\sum_{i=1}^{n} m_i}$. □

1.6 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) $(a+1)^n - \frac{n(n-1)}{2}a^2 - an - 1 \vdots n^3$, $\forall a \in \mathbb{Z}$.*

Problem 5 ([Sha22], P1.1.2, p. 10). *Prove: (a)* (Formula of triangle numbers) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, $\forall n \in \mathbb{N}^\star$. *(b)* (Partial sums of geometric sequences)

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

(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* $\left\{\frac{n(n+1)}{2}\right\}_{n=1}^{\infty}$ *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 6 ([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?*

1.7 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}.

1.8 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*

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

1.9 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 7 ([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 8. *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 9 ([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 1 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 Piṅgala–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 1 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].*

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

Problem 10 ([Sha22], P1.3.1, p. 22)**.** *We are given the following recurrence relation $a_1 = 2$, $a_n = 3a_{n-1} + 2$ for $n \in \mathbb{N}, n \geq 2$. Find the values of a_n for small n. Do you see a pattern? Make a conjecture. Prove your conjecture using induction.*

– Chúng ta được cho hệ thức đệ quy sau $a_1 = 2$, $a_n = 3a_{n-1} + 2$ đối với $n \in \mathbb{N}, n \geq 2$. Tìm các giá trị của a_n đối với n nhỏ. Bạn có thấy 1 mô hình không? Đưa ra 1 phỏng đoán. Chứng minh phỏng đoán của bạn bằng cách sử dụng quy nạp.

Problem 11 ([Sha22], P1.3.2, p. 22)**.** *Let $n \in \mathbb{N}$. We know that $f(0) = -3$, & $f(n) = 3f(n-1) + 10$, $\forall n \in \mathbb{N}^\star$. Generate some data, & use the data to conjecture a closed formula for $f(n)$. Prove your conjecture using induction.*

– Cho $n \in \mathbb{N}$. Ta biết rằng $f(0) = -3$, & $f(n) = 3f(n-1) + 10$, $\forall n \in \mathbb{N}^\star$. Tạo 1 số dữ liệu, & sử dụng dữ liệu để suy ra công thức đóng cho $f(n)$. Chứng minh suy đoán của bạn bằng quy nạp.

Problem 12 ([Sha22], P1.3.3, p. 22)**.** *Let h_n denote the number of ways of covering a $2 \times n$ array with $1 \times n$ dominoes. Find h_1, h_2, h_3, & a recurrence relation for h_n. Use these to find h_8.*

– Giả sử h_n biểu thị số cách phủ mảng $2 \times n$ bằng $1 \times n$ domino. Tìm h_1, h_2, h_3, & 1 quan hệ đệ quy cho h_n. Sử dụng chúng để tìm h_8.

Problem 13 ([Sha22], P1.3.4, pp. 22–23)**.** *Let $n \in \mathbb{N}^\star$. Consider a $1 \times n$ strip of cardboard. We have (a large number of) 3 types of pieces: 1×2 red pieces, 1×4 yellow pieces, & 1×4 blue pieces. Let $f(n)$ be the number of ways that we can title the strip of cardboard with our pieces. E.g., $f(3) = 0$ since you cannot tile a 1×3 board with the available pieces, while $f(6) = 5$ since the possibilities are* `YR, RY, RB, BR, RRR`*. Find $f(14)$. (Give an actual number.)*

– Cho $n \in \mathbb{N}^\star$. Xét 1 dải bìa cứng $1 \times n$. Chúng ta có (1 số lượng lớn) 3 loại mảnh: 1×2 mảnh đỏ, 1×4 mảnh vàng, & 1×4 mảnh xanh. Cho $f(n)$ là số cách chúng ta có thể đặt tên cho dải bìa cứng bằng các mảnh của mình. Ví dụ, $f(3) = 0$ vì bạn không thể xếp 1 bảng 1×3 bằng các mảnh có sẵn, trong khi $f(6) = 5$ vì các khả năng là `YR, RY, RB, BR, RRR`*. Tìm $f(14)$. (Nêu 1 số thực tế.)*

Problem 14 ([Sha22], P1.3.5, p. 23). *Generations of Indian commentators, apparently beginning with Pingala in the 3rd to 2nd century BCE, were interested in the number of rhythmic patterns having a total of $n \in \mathbb{N}^\star$ beats. Assuming that we use only 1-beat & 2-beat rhythms, how many different patterns can we make that have a total of n beats? E.g., if $n = 4$, we can have $2-2, 2-1-1, 1-2-1, 1-1-2, 1-1-1-1$ for a total of 5 possibilities. If we denote the answer by $r(n)$, then find a recurrence relation for $r(n)$. Is $r(n)$ related to the Pingala–Fibonacci numbers?*

– *Nhiều thế hệ nhà bình luận Ấn Độ, rõ ràng bắt đầu từ Pingala vào thế kỷ thứ 3 đến thế kỷ thứ 2 trước Công nguyên, đã quan tâm đến số lượng các mẫu nhịp điệu có tổng cộng $n \in \mathbb{N}^\star$ nhịp. Giả sử rằng chúng ta chỉ sử dụng nhịp điệu 1-nhịp & 2-nhịp, thì chúng ta có thể tạo ra bao nhiêu mẫu nhịp điệu khác nhau có tổng cộng n nhịp? Ví dụ, nếu $n = 4$, chúng ta có thể có $2-2, 2-1-1, 1-2-1, 1-1-2, 1-1-1-1$ với tổng cộng 5 khả năng. Nếu chúng ta biểu thị câu trả lời bằng $r(n)$, thì hãy tìm 1 hệ thức đệ quy cho $r(n)$. $r(n)$ có liên quan đến dãy số Pingala–Fibonacci không?*

Problem 15 ([Sha22], P1.3.6, p. 23). *In his 1202 book* Liber Abaci, Leonardo *of Pisa (known as* Fibonacci *since the 19th century) poses the following problem:*

> *A certain man put a pair of rabbits in a place surrounded on all sides by a wall. How many pairs of rabbits can be produced from that pair in a year if it is supposed that every month each pair begets a new pair which from the 2nd month on becomes productive?*

Give a solution. Is this related to the Pingala–Fibonacci numbers? How?

– *Trong cuốn sách năm 1202 của mình* Liber Abaci, Leonardo *xứ Pisa (được biết đến với tên* Fibonacci *từ thế kỷ 19) đã đặt ra vấn đề sau:*

> *Một người đàn ông nào đó đặt 1 cặp thỏ vào 1 nơi được bao quanh bởi 1 bức tường ở mọi phía. Có bao nhiêu cặp thỏ có thể được tạo ra từ cặp đó trong 1 năm nếu giả sử rằng mỗi tháng, mỗi cặp lại sinh ra 1 cặp mới & từ tháng thứ 2 trở đi, cặp này sẽ sinh sản?*

Đưa ra giải pháp. Điều này có liên quan đến dãy số Pingala–Fibonacci không? Bằng cách nào?

Problem 16 ([Sha22], P1.3.7, p. 23). *Assume that, for $n \geq 3$, the sequence $\{a_n\}_{n=1}^{\infty}$ satisfies the recurrence relation $a_n = a_{n-1} + a_{n-2}$. Assume further that we know that $a_2 = 3, a_{50} = 300$. Find $\sum_{i=1}^{48}$ & justify your answer.*

– *Giả sử rằng, với $n \geq 3$, dãy $\{a_n\}_{n=1}^{\infty}$ thỏa mãn hệ thức đệ quy $a_n = a_{n-1} + a_{n-2}$. Giả sử thêm rằng chúng ta biết rằng $a_2 = 3, a_{50} = 300$. Tìm $\sum_{i=1}^{48}$ & chứng minh câu trả lời của bạn.*

Problem 17 ([Sha22], P1.3.8, p. 23). *You work at a car dealership that sells 3 models: A pickup truck, an SUV, & a compact hybrid. Your job is to park the vehicles in a row. The pickup trucks & the SUV take up 2 spaces while the hybrid takes up 1 space. Let $n \in \mathbb{N}^\star$ & let $f(n)$ be the number of ways of arranging vehicles in exactly n spaces. Find a recurrence relation for $f(n)$ & use it to find $f(10)$, the number of ways of arranging vehicles if you have 10 parking spaces.*

– *Bạn làm việc tại 1 đại lý ô tô bán 3 mẫu xe: Một xe bán tải, 1 xe SUV, & 1 xe hybrid nhỏ gọn. Công việc của bạn là đỗ những chiếc xe này thành 1 hàng. Xe bán tải & xe SUV chiếm 2 chỗ trong khi xe hybrid chiếm 1 chỗ. Giả sử $n \in \mathbb{N}^\star$ & giả sử $f(n)$ là số cách sắp xếp xe trong đúng n chỗ. Tìm 1 hệ thức đệ quy cho $f(n)$ & sử dụng nó để tìm $f(10)$, số cách sắp xếp xe nếu bạn có 10 chỗ đỗ xe.*

Problem 18 ([Sha22], P1.3.9, p. 23). *Continuing with the assumptions & the notation of Problem P1.3.8. (a) Find $f(1), \ldots, f(5)$. (b) Find & prove a 1-step recurrence relation for $f(n)$ (i.e., one that only depends on $f(n-1)$). (c) Find & prove a formula for $f(n)$ of the form $f(n) = \frac{1}{?}(2^{n+1}+?)$.*

– *Tiếp tục với các giả định & ký hiệu của Bài toán P1.3.8. (a) Tìm $f(1), \ldots, f(5)$. (b) Tìm & chứng minh 1 quan hệ đệ quy 1-bước cho $f(n)$ (tức là 1 quan hệ chỉ phụ thuộc vào $f(n-1)$). (c) Tìm & chứng minh 1 công thức cho $f(n)$ có dạng $f(n) = \frac{1}{?}(2^{n+1}+?)$.*

Problem 19 ([Sha22], P1.3.10, pp. 23–24). *At a dinner party on the spaceship Enterprise, 3 life forms are present: Human, Klingons, & Romulans. The dinner table is a long $1 \times n$ board, & the life forms sit on 1 side of it, next to one another. From each life form there are $> n$ individuals present, & so only a total of n sit at the table. The only problem is that no 2 humans want to sit next to each other*[1]*. Let h_n denote the number of different ways that n individuals can be seated at the dinner table. Assume that all humans look alike, as do all Klingons & all Romulans. (a) What is h_1, h_2? (b) Which one(s) (if any) of the following are true, & which are false? (i) $h_n = 3h_{n-1} - h_{n-2}$. (ii) $h_n = 2h_{n-1} + 2h_{n-2}$. (iii) $h_n = 3h_{n-1} - (n-1)!$. (iv) $h_n = h_{n-1} + 3h_{n-2} + 2h_{n-3}$. Give adequate & complete reasoning for your answers.*

– *Tại 1 bữa tiệc tối trên tàu vũ trụ Enterprise, có 3 dạng sống hiện diện: Con người, người Klingon, & Romulan. Bàn ăn là 1 tấm ván dài $1 \times n$, & các dạng sống ngồi ở 1 phía của bàn, cạnh nhau. Từ mỗi dạng sống có $> n$ cá thể hiện diện, & do đó chỉ có tổng cộng n ngồi vào bàn. Vấn đề duy nhất là không có 2 con người nào muốn ngồi cạnh nhau*[2]*. Giả sử h_n biểu thị số cách khác*

[1] NQBH: Damn these human beings!
[2] NQBH: Chết tiệt bọn người này!

nhau để n cá nhân có thể ngồi vào bàn ăn. Giả sử rằng tất cả con người đều giống nhau, giống như tất cả người Klingon & tất cả người Romulan. (a) h_1, h_2 là gì? (b) Trong các câu sau, câu nào (nếu có) là đúng, & câu nào là sai? (i) $h_n = 3h_{n-1} - h_{n-2}$. (ii) $h_n = 2h_{n-1} + 2h_{n-2}$. (iii) $h_n = 3h_{n-1} - (n-1)!$. (iv) $h_n = h_{n-1} + 3h_{n-2} + 2h_{n-3}$. Đưa ra lý lẽ & đầy đủ cho câu trả lời của bạn.

Problem 20 ([Sha22], P1.3.11, p. 24)**.** *Using only the digits $1, 2, 3$, how many integers can you construct in a way that the sum of the digits is 9? Examples would be $333, 1323, 2133, 22221$.*

– Chỉ sử dụng các chữ số $1, 2, 3$, bạn có thể tạo ra bao nhiêu số nguyên sao cho tổng các chữ số bằng 9? Ví dụ: $333, 1323, 2133, 22221$.

Problem 21 ([Sha22], P1.3.12, p. 24)**.** *I have 12 identical irises & 4 distinct flowerpots. All the irises are to be planted, & I want to plant $2, 3$, or 4 irises in each pot. I am interested in finding out the number of ways that this can be done. We 1st generalize the question. Given $n \in \mathbb{N}^\star$ identical irises & $k \in \mathbb{N}^\star$ distinct pots, let $F(n, k)$ be the number of ways that we can distribute the irises among the pots if each pot gets $2, 3$, or 4 irises. (a) Find $F(5, 2)$. (b) Find a recurrence relation for $F(n, k)$. Explain your reasoning. (c) Find $F(n, k)$ for the given values of n, k in a table. (d) What is $F(12, 4)$? Why?*

– Tôi có 12 cây diên vĩ giống hệt nhau & 4 chậu hoa riêng biệt. Tất cả các cây diên vĩ đều được trồng, & Tôi muốn trồng $2, 3$ hoặc 4 cây diên vĩ vào mỗi chậu. Tôi muốn tìm ra số cách thực hiện điều này. Đầu tiên, chúng ta khái quát hóa câu hỏi. Với $n \in \mathbb{N}^\star$ cây diên vĩ giống hệt nhau & $k \in \mathbb{N}^\star$ chậu riêng biệt, hãy để $F(n, k)$ là số cách chúng ta có thể phân phối cây diên vĩ vào các chậu nếu mỗi chậu có $2, 3$ hoặc 4 cây diên vĩ. (a) Tìm $F(5, 2)$. (b) Tìm 1 hệ thức đệ quy cho $F(n, k)$. Giải thích lý luận của bạn. (c) Tìm $F(n, k)$ cho các giá trị n, k đã cho trong 1 bảng. (d) $F(12, 4)$ là gì? Tại sao?

Problem 22 ([Sha22], P1.3.13, pp. 24–25)**.** *2 thick panes of glass are adjacent to each other. Light that enters from 1 side can be reflected by the internal faces. I.e., after entering the medium, the light may go straight through or be reflected back & forth by the 3 internal faces, as shown in* Fig. 1.4: The possibilities, resp., with $0, 1$, or 2 reflections *shows that $a_0 = 1, a_1 = 2, a_2 = 3$. Find a recurrence relation for a_n. Use the recurrence relation to find a_7.*

– 2 tấm kính dày nằm cạnh nhau. Ánh sáng đi vào từ 1 mặt có thể bị phản xạ bởi các mặt bên trong. Tức là, sau khi đi vào môi trường, ánh sáng có thể đi thẳng qua hoặc bị phản xạ qua lại & lui bởi 3 mặt bên trong, như thể hiện trong Hình 1.4: Các khả năng, tương ứng, với $0, 1$ hoặc 2 phản xạ *cho thấy $a_0 = 1, a_1 = 2, a_2 = 3$. Tìm 1 hệ thức đệ quy cho a_n. Sử dụng hệ thức đệ quy để tìm a_7.*

Problem 23 ([Sha22], P1.3.14, p. 25)**.** *Let $n \in \ltimes^\star$, & let $h(n)$ be the number of sequences $a_1, a_2, \ldots, a_n$ with the conditions that, for $i \in [n]$, each $a_i \in \{0, 4, 7\}$, & each 4 in the sequence is* immediately *followed by a 7. (a) Find $h(1), h(2)$. (b) Find a recurrence relation for $h(n)$. Use it to find $h(5)$. (c) I entered the 1st few terms of the sequence into the* Online Encyclopedia of Integer Sequences*, & it told me that a formula for $h(n)$ could be*

$$h(n) = \frac{(1+\sqrt{2})^{n+1} - (1-\sqrt{2})^{n+1}}{2\sqrt{2}}.$$

Is this true? Prove your assertion.

– Cho $n \in \ltimes^\star$, & cho $h(n)$ là số dãy $a_1, a_2, \ldots, a_n$ với các điều kiện cho $i \in [n]$, mỗi $a_i \in \{0, 4, 7\}$, & mỗi 4 trong dãy là ngay lập tức *theo sau là a 7. (a) Tìm $h(1), h(2)$. (b) Tìm 1 hệ thức đệ quy cho $h(n)$. Sử dụng nó để tìm $h(5)$. (c) Tôi đã nhập 1 vài số hạng đầu tiên của dãy số vào* Online Encyclopedia of Integer Sequences*, & nó cho tôi biết rằng công thức cho $h(n)$ có thể là*

$$h(n) = \frac{(1+\sqrt{2})^{n+1} - (1-\sqrt{2})^{n+1}}{2\sqrt{2}}.$$

Điều này có đúng không? Hãy chứng minh khẳng định của bạn.

Problem 24 ([Sha22], P1.3.15, pp. 25–26)**.** *Let $h(0) = 1$ & let $h(n)$ be the number of sequences $a_1, a_2, \ldots, a_n$, with the conditions that, for $i \in [n]$, each $a_i \in \{0, 4, 7\}$, & if both 4 & 7 occur in the sequence then the 1st 4 occurs before the 1st 7. (a) Find $h(1), h(2)$. (b) Find a recurrence relation for $h(n)$. (c) What is $h(5)$?*

– Cho $h(0) = 1$ & cho $h(n)$ là số dãy số $a_1, a_2, \ldots, a_n$, với các điều kiện là, đối với $i \in [n]$, mỗi $a_i \in \{0, 4, 7\}$, & nếu cả 4 & 7 xảy ra trong dãy số thì 4 đầu tiên xảy ra trước 7 đầu tiên. (a) Tìm $h(1), h(2)$. (b) Tìm 1 hệ thức đệ quy cho $h(n)$. (c) $h(5)$ là gì?

Problem 25 ([Sha22], P1.3.16, p. 25)**.** *You are interested in finding the number of sequences $a_1, a_2, \ldots, a_n$ s.t. each $a_i \in \{0, 1, 2, 3, 4$ & the number of 3's in the sequence is even. Denote the number of such sequences by $E(n)$. (a) Find $E(1), E(2)$. (b) Let $O(n)$ be the number of sequences $a_1, a_2, \ldots, a_n$ s.t. each $a_i \in \{0, 1, 2, 3, 4\}$ & the number of 3's in the sequence is odd. How is $O(n)$ related to $E(n)$? (c) Find a recurrence relation for $E(n)$ & state clearly your reasoning for why it works. (d) Find $E(4)$.*

– Bạn muốn tìm số dãy số $a_1, a_2, \ldots, a_n$ s.t. mỗi $a_i \in \{0, 1, 2, 3, 4$ & số 3 trong dãy số là số chẵn. Ký hiệu số các dãy số như vậy là $E(n)$. (a) Tìm $E(1), E(2)$. (b) Giả sử $O(n)$ là số dãy số $a_1, a_2, \ldots, a_n$ s.t. mỗi $a_i \in \{0, 1, 2, 3, 4\}$ & số 3 trong dãy số là số lẻ. $O(n)$ liên quan như thế nào đến $E(n)$? (c) Tìm 1 hệ thức đệ quy cho $E(n)$ & nêu rõ lý do tại sao nó hiệu quả. (d) Tìm $E(4)$.

Problem 26 ([Sha22], P1.3.17, p. 26). *We want to divide \$275,000 among 4 people. We have to divide all the money, & we can only divide the money in \$25,000 increments. In how many ways can this be done? Make a conjecture.*

To make the question clear, consider a simpler example. If we had \$100,000 to divide among 4 people, then we could give all of it to 1 person, give \$50,000 to 2 people, give \$75,000 to 1 person & \$25,000 to another, give \$50,000 to 41 & \$25,000 to 2 others, or finally give \$25,000 to each of the 4 people. Thus we can divide \$100,000 among 4 people in 5 different ways. Note that we consider the 4 people indistinguishable. If it mattered to us who got how much, then the number of ways of dividing the money would have been considerably higher.

– Chúng ta muốn chia \$275.000 cho 4 người. Chúng ta phải chia hết số tiền, & chúng ta chỉ có thể chia số tiền theo \$25.000 gia số. Có bao nhiêu cách để thực hiện điều này? Hãy đưa ra 1 phỏng đoán.

Để làm rõ câu hỏi, hãy xem xét 1 ví dụ đơn giản hơn. Nếu chúng ta có \$100.000 để chia cho 4 người, thì chúng ta có thể đưa toàn bộ cho 1 người, đưa \$50.000 cho 2 người, đưa \$75.000 cho 1 người & \$25.000 cho 41 người & \$25.000 cho 2 người khác, hoặc cuối cùng đưa \$25.000 cho mỗi 4 người. Do đó, chúng ta có thể chia \$100.000 cho 4 người theo 5 cách khác nhau. Lưu ý rằng chúng ta coi 4 người là không thể phân biệt được. Nếu chúng ta quan tâm đến việc ai nhận được bao nhiêu thì số cách chia tiền hẳn phải cao hơn đáng kể.

Problem 27 ([Sha22], P1.3.18, p. 26, Gamma function). *Let $x \in (0, \infty)$. Define the function $\Gamma(x)$ by $\Gamma(x) \coloneqq \int_0^\infty t^{x-1} e^{-t} \, \mathrm{d}t$. The integral defining $\Gamma(x)$ is an improper integral – since 1 of the end points is ∞ – & yet it converges $\forall x \in (0, \infty)$. (This is not hard to prove using standard calculus techniques, but for the purposes of this problem, you can assume that the integral converges $\forall x \in (0, \infty)$. In fact, interesting things happen when one extends the definition to complete numbers.) (a) What is $\Gamma(1)$? (b) Use integration by parts, & the fact that, for a fixed x, $\lim_{t \to \infty} \frac{t^x}{e^t} = 0$, to prove the following recurrence relation for $\Gamma(x)$: $\Gamma(x+1) = x\Gamma(x)$. (c) Use the recurrence relation to find $\Gamma(2), \ldots, \Gamma(5)$. If $n \in \mathbb{N}^\star$, what is $\Gamma(n)$? Prove your assertion using induction. (d) The Encyclopedia Britannica entry for the Gamma function [Encyclopedia Britannica 2019] begins: "Gamma function, generalization of the factorial function to nonintegral values introduced by the Swiss mathematician* Leonhard Euler *in the 18th century." Explain the relation of the Gamma function to the factorial function.*

– Cho $x \in (0, \infty)$. Định nghĩa hàm $\Gamma(x)$ theo $\Gamma(x) \coloneqq \int_0^\infty t^{x-1} e^{-t} \, \mathrm{d}t$. Tích phân xác định $\Gamma(x)$ là 1 tích phân không chuẩn – vì 1 trong các điểm cuối là ∞ – & nhưng nó hội tụ $\forall x \in (0, \infty)$. (Điều này không khó để chứng minh bằng các kỹ thuật tính toán chuẩn, nhưng đối với mục đích của bài toán này, bạn có thể giả sử rằng tích phân hội tụ $\forall x \in (0, \infty)$. Trên thực tế, những điều thú vị xảy ra khi người ta mở rộng định nghĩa cho các số hoàn chỉnh.) (a) $\Gamma(1)$ là gì? (b) Sử dụng tích phân từng phần, & thực tế là, đối với x cố định, $\lim_{t \to \infty} \frac{t^x}{e^t} = 0$, để chứng minh mối quan hệ đệ quy sau cho $\Gamma(x)$: $\Gamma(x+1) = x\Gamma(x)$. (c) Sử dụng mối quan hệ đệ quy để tìm $\Gamma(2), \ldots, \Gamma(5)$. Nếu $n \in \mathbb{N}^\star$, thì $\Gamma(n)$ là gì? Chứng minh khẳng định của bạn bằng cách sử dụng quy nạp. (d) Mục từ của Bách khoa toàn thư Britannica về hàm Gamma [Bách khoa toàn thư Britannica 2019] bắt đầu: "Hàm Gamma, tổng quát hóa hàm giai thừa thành các giá trị không nguyên được nhà toán học người Thụy Sĩ Leonhard Euler *giới thiệu vào thế kỷ 18." Giải thích mối quan hệ của hàm Gamma với hàm giai thừa.*

Problem 28 ([Sha22], P1.3.19, pp. 26–27). *Define a* selfish set *to be a set which has its own cardinality (number of elements) as an elements. [Adapted from Problem B-1 of the 1996 Putnam Mathematical Competition.] A set is a* minimal selfish set *if it is selfish, & no proper subset of it is selfish. So, e.g., $\{2, 47\}$ is a minimal selfish set, $\{2, 3, 5\}$ is selfish but not minimal, while $\{4, 5, 6\}$ is not selfish at all. Let $n \in \mathbb{N}^\star$, & let f_n be the number of subsets of $[n]$ which are minimal selfish sets. Find a recurrence relation for f_n. The following steps may be helpful:*

1. *Find f_1, f_2, f_3.*
2. *Which minimal selfish sets contain 1?*
3. *Is the size of a minimal selfish subset of $[n]$ ever n?*
4. *What can you say about the number of minimal selfish subsets of $[n]$ that do not include n?*
5. *Assume a minimal selfish subset of $[n]$ includes n. If you subtract 1 from each of the elements of the subset, & then throw out $n - 1$, do you get a minimal selfish set?*
6. *Write down, with proof, a recurrence relation for f_n.*

– Định nghĩa 1 tập ích kỷ *là 1 tập có số lượng phần tử riêng (số phần tử) như 1 phần tử. [Chuyển thể từ Bài toán B-1 của Cuộc thi Toán học Putnam năm 1996.] Một tập hợp là* tập ích kỷ tối thiểu *nếu nó ích kỷ, & không có tập con thực sự nào của nó là ích kỷ. Vì vậy, ví dụ, $\{2, 47\}$ là 1 tập ích kỷ tối thiểu, $\{2, 3, 5\}$ là ích kỷ nhưng không tối thiểu, trong khi $\{4, 5, 6\}$ không ích kỷ chút nào. Giả sử $n \in \mathbb{N}^\star$, & giả sử f_n là số tập con của $[n]$ là các tập ích kỷ tối thiểu. Tìm 1 hệ thức đệ quy cho f_n. Các bước sau đây có thể hữu ích:*

1. *Tìm f_1, f_2, f_3.*
2. *Những tập ích kỷ tối thiểu nào chứa 1?*

3. *Kích thước của 1 tập con ích kỷ tối thiểu của* $[n]$ *có bao giờ là* n *không?*

4. *Bạn có thể nói gì về số lượng các tập con ích kỷ tối thiểu của* $[n]$ *không bao gồm* n*?*

5. *Giả sử 1 tập con ích kỷ tối thiểu của* $[n]$ *bao gồm* n*. Nếu bạn trừ* 1 *khỏi mỗi phần tử của tập con, & sau đó loại bỏ* $n-1$*, bạn có nhận được 1 tập ích kỷ tối thiểu không?*

6. *Viết ra, kèm theo bằng chứng, 1 quan hệ đệ quy cho* f_n*.*

1.10 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 1 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 29 ([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*

$$f(n) = \sum_{i=1}^{k} \alpha_i f(n-i) = \alpha_1 f(n-1) + \alpha_2 f(n-2) + \cdots + \alpha_k f(n-k) \text{ for some } k \in \mathbb{N}^\star \text{ \& some } \alpha_i \in \mathbb{R},\ \forall i \in [k]. \tag{1.2}$$

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

$$f(n) = \sum_{i=1}^{k} \alpha_i f(n-i) + g(n) = \alpha_1 f(n-1) + \alpha_2 f(n-2) + \cdots + \alpha_k f(n-k) + g(n) \text{ for some } k \in \mathbb{N}^\star,\ \text{some } \alpha_i \in \mathbb{R},\ \forall i \in [k], \tag{1.3}$$

& 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* (1.2)*. 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* (1.2)*.*

(b) If functions h_1, h_2 *satisfy the nonhomogeneous linear recurrence* (1.3)*, then their difference* $h_1 - h_2$ *satisfies the homogeneous linear recurrence of* (1.2)*.*

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

Problem 30 ([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*.*

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

Problem 31 ([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 1 tá quả cùng loại?*

An almost trivial fact:

Theorem 2 (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 3 (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.*

1.12 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 $s_1(n,k), s_2(n,k)$.

1.12.1 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:

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

1.12.2 Stirling Numbers of Type 2 – Số Stirling Loại 2

Resources – Tài nguyên.

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

2. [HT24].

Problem 32 ([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 33. *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*[3] *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 34. *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 4 ([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. 13 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 5 ([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. □

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

Theorem 6 ([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 35 ([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 $\frac{k!}{k^n}\left\{ {n-1 \atop k-1} \right\}$. □

1.12.3 Problems: Stirling numbers

1.13 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*

$$B(s) \coloneqq \sum_{i=0}^{s} \left\{ {s \atop i} \right\}, \ \forall s \in \mathbb{N}. \tag{1.4}$$

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 36. *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 7 ([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 8 ([Sha22], Prop. 6.15, p. 201).

$$B(n) = \frac{1}{e} \sum_{i=0}^{\infty} \frac{k^n}{k!}, \ \forall n \in \mathbb{N}. \tag{1.5}$$

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 (1.5), 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).*

1.14 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 nth Catalan number can be expressed directly in terms of the central binomial coefficients by

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{n!(n+1)!}, \ \forall n \in \mathbb{N}^\star. \tag{1.6}$$

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).

1.14.1 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 (1.6) 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 (1.6) 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 37. *Prove: (a) The Catalan numbers satisfy the recurrence relations*

$$C_0 = 1, \ C_n = \sum_{i=1}^{n} C_{i-1}C_{n-i}, \ \forall n \in \mathbb{N}^\star, \tag{1.7}$$
$$C_0 = 1, \ C_n = \frac{2(2n-1)}{n+1}C_{n-1}, \ \forall n \in \mathbb{N}^\star.$$

(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 1 cách giải thích xác suất đơn giản. Hãy xem xét 1 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 $\sum_{n=0}^{\infty}\frac{C_n}{2^{2n+1}} = 1$.

1.14.2 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: ((())), (()()), (())(), ()(()), ()()().

1.14.3 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 (1.7). 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;
}

```

```
unsigned long long permutation(int n, int k) {
    unsigned long long a = factorial(n);
    unsigned long long b = factorial(n - k);
    return (b == 0) ? 0 : a / b;
}

unsigned long long combination(int n, int k) {
    unsigned long long a = factorial(n);
    unsigned long long b = factorial(k);
    unsigned long long c = factorial(n - k);
    return (b == 0 || c == 0) ? 0 : a / (b * c);
}

unsigned long long catalan(int n) {
    unsigned long long a = factorial(2 * n);
    unsigned long long b = factorial(n + 1);
    unsigned long long c = factorial(n);
    return (b == 0 || c == 0) ? 0 : a / (b * c);
}

int main() {
    int n, k;
    cout << "Nhap n (n > 0): ";
    cin >> n;
    cout << "Nhap k (0 <= k <= n): ";
    cin >> k;

    cout << "\n=== Ket qua ===\n";

    unsigned long long fn = factorial(n);
    if (fn != 0) cout << "Pn = " << fn << endl;

    unsigned long long Ank = permutation(n, k);
    if (Ank != 0) cout << "A^k_n = " << Ank << endl;

    unsigned long long Cnk = combination(n, k);
    if (Cnk != 0) cout << "C^k_n = " << Cnk << endl;

    unsigned long long Catn = catalan(n);
    if (Catn != 0) cout << "Catalan(n) = " << Catn << endl;
    return 0;
}
```

Chương 2

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

Contents

2.1 Binomial Theorem/Binomial Expansion – Định Lý Khai Triển Nhị thức Newton

Resources – Tài nguyên.

1. Wikipedia/binomial theorem.

In elementary algebra, the *elementary algebra*, the *binomial theorem* (or *binomial expansion*) describes the algebraic expansion of powers of a binomial. According to the theorem, the power $(x+y)^n$ expands into a polynomial with terms of the form $ax^k y^m$, where the exponents $k, m \in \mathbb{N}$ satisfying $k+m=n$ & the coefficient a of each term is a specific positive integer depending on n, k. The coefficient a in each term $ax^k y^m$ is known as the binomial coefficient $\binom{n}{k}$ or $\binom{n}{m}$ (the two have the same value). These coefficients for varying n, k can be arranged to form Pascal's triangle. These numbers also occur in combinatorics, where $\binom{n}{k}$ gives the number of different combinations (i.e., subsets) of k elements that can be chosen from an n-element set. Therefore $\binom{n}{k}$ is usually produced as "n choose k".

– Trong đại số sơ cấp, *đại số sơ cấp*, *định lý nhị thức* (hoặc *khai triển nhị thức*) mô tả khai triển đại số các lũy thừa của 1 nhị thức. Theo định lý, lũy thừa $(x+y)^n$ khai triển thành 1 đa thức với các số hạng có dạng $ax^k y^m$, trong đó các số mũ $k, m \in \mathbb{N}$ thỏa mãn $k+m=n$ & hệ số a của mỗi số hạng là 1 số nguyên dương cụ thể phụ thuộc vào n, k. Hệ số a trong mỗi số hạng $ax^k y^m$ được gọi là hệ số nhị thức $\binom{n}{k}$ hoặc $\binom{n}{m}$ (cả hai có cùng giá trị). Các hệ số này khi n, k thay đổi có thể được sắp xếp để tạo thành tam giác Pascal. Những con số này cũng xuất hiện trong tổ hợp, trong đó $\binom{n}{k}$ đưa ra số lượng các tổ hợp khác nhau (tức là các tập hợp con) của k phần tử có thể được chọn từ 1 tập hợp n phần tử. Do đó, $\binom{n}{k}$ thường được tạo ra dưới dạng "n chọn k".

Theorem 9 (Binomial theorem/formula/identity)**.** *The expansion of any nonnegative integer power $n \in \mathbb{N}$ of the binomial $x+y$ is a sum of the form*

$$(x+y)^n = \binom{n}{0}x^n y^0 + \binom{n}{1}x^{n-1}y^1 + \binom{n}{2}x^{n-2}y^2 + \cdots + \binom{n}{n}x^0 y^n = \sum_{k=0}^{n}\binom{n}{k}x^{n-k}y^k = \binom{n}{k}x^k y^{n-k},$$

where each $\binom{n}{k}$ is a positive integer known as a binomial coefficient, defined as

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)(n-2)\cdots(n-k+1)}{1\cdot 2\cdots(k-1)k}.$$

Combinatorial proof – chứng minh bằng tổ hợp. Expanding $(x+y)^n$ yields the sum of the 2^n products of the form $\prod_{i=1}^{n} e_i = e_1 e_2 \cdots e_n$ where $e_i \in \{x, y\}$, $\forall i \in [n]$. Rearranging factors show that each product equals $x^{n-i}y^i$ for some $i \in \overline{0,n}$. For a given $i \in \overline{0,n}$, the following are proved equal in succession:

1. the number of terms equal to $x^{n-i}y^i$ in the expansion
2. the number of n-character x, y strings having y in exactly i positions
3. the number of i-element subsets of $[n]$
4. $\binom{n}{i}$, either by definition, or by a short combinatorial argument if one is defining $\binom{n}{i}$ as $\dfrac{n!}{i!(n-i)!}$.

This proves the binomial theorem.

– Khai triển $(x+y)^n$ tạo ra tổng của 2^n tích có dạng $\prod_{i=1}^{n} e_i = e_1 e_2 \cdots e_n$ trong đó $e_i \in \{x, y\}$, $\forall i \in [n]$. Sắp xếp lại các thừa số cho thấy rằng mỗi tích bằng $x^{n-i}y^i$ đối với 1 số $i \in \overline{0,n}$. Đối với $i \in \overline{0,n}$ cho trước, các điều sau đây được chứng minh là bằng nhau theo trình tự:

1. số lượng các số hạng bằng $x^{n-i}y^i$ trong khai triển
2. số lượng các chuỗi n ký tự x, y có y ở đúng i vị trí
3. số lượng các tập hợp con i phần tử của $[n]$
4. $\binom{n}{i}$, theo định nghĩa hoặc theo 1 đối số tổ hợp ngắn nếu ta định nghĩa $\binom{n}{i}$ là $\dfrac{n!}{i!(n-i)!}$.

Điều này chứng minh định lý nhị thức. □

Inductive proof – chứng minh bằng quy nạp. Induction yields another proof of the binomial theorem. When $n = 0$, both sides equal 1, since $x^0 = 1, \binom{0}{0} = 1$. Now suppose that the equality hold for a given n, we will prove it for $n+1$. For $i, j \in \mathbb{N}$, let $[P(x,y)]_{i,j}$ denote the coefficient of $x^i y^j$ in the polynomial $P(x,y)$. By the inductive hypothesis, $(x+y)^n$ is a polynomial in x, y s.t. $[(x+y)^n]_{i,j}$ is $\binom{n}{j}$ if $i+j=n$, & 0 otherwise. The identity $(x+y)^{n+1} = x(x+y)^n + y(x+y)^n$ shows that $(x+y)^{n+1}$ is also a polynomial in x, y, & $[(x+y)^{n+1}]_{i,j} = [(x+y)^{n+1}]_{i-1,j} + [(x+y)^{n+1}]_{i,j-1}$, since if $i+j = n+1$, then $(i-1)+j = i+(j-1) = n$. Now, the RHS is $\binom{n}{j} + \binom{n}{j-1} = \binom{n+1}{j}$, by Pascal's identity. On the other hand, if $i+j \neq n+1$, then $(i-1)+j = i+(j-1) \neq n$, so we get $0+0=0$. Thus

$$(x+y)^{n+1} = \sum_{j=0}^{n+1} \binom{n+1}{j} x^{n+1-j} y^j,$$

which is the inductive hypothesis with $n+1$ substituted for n & so completes the inductive step.

– Quy nạp đưa ra 1 cách chứng minh khác của định lý nhị thức. Khi $n = 0$, cả hai vế bằng 1, vì $x^0 = 1, \binom{0}{0} = 1$. Bây giờ giả sử rằng đẳng thức giữ nguyên cho 1 số nguyên n cho trước, ta sẽ chứng minh nó cho $n+1$. Với $i, j \in \mathbb{N}$, hãy để $[P(x,y)]_{i,j}$ biểu thị hệ số của $x^i y^j$ trong đa thức $P(x,y)$. Theo giả thiết quy nạp, $(x+y)^n$ là 1 đa thức trong x, y s.t. $[(x+y)^n]_{i,j}$ là $\binom{n}{j}$ nếu $i+j=n$, & 0 nếu ngược lại. Đẳng thức $(x+y)^{n+1} = x(x+y)^n + y(x+y)^n$ cho thấy $(x+y)^{n+1}$ cũng là 1 đa thức trong x, y, & $[(x+y)^{n+1}]_{i,j} = [(x+y)^{n+1}]_{i-1,j} + [(x+y)^{n+1}]_{i,j-1}$, vì nếu $i+j = n+1$, thì $(i-1)+j = i+(j-1) = n$. VP là $\binom{n}{j} + \binom{n}{j-1} = \binom{n+1}{j}$, theo đẳng thức Pascal. Mặt khác, nếu $i+j \neq n+1$, thì $(i-1)+j = i+(j-1) \neq n$, do đó ta có $0+0=0$. Do đó

$$(x+y)^{n+1} = \sum_{j=0}^{n+1} \binom{n+1}{j} x^{n+1-j} y^j,$$

là giả thuyết quy nạp với $n+1$ thay thế cho n & do đó hoàn thành bước quy nạp. □

Định lý 2 (Định lý/công thức/hằng đẳng thức nhị thức)**.** *Khai triển của bất kỳ lũy thừa số nguyên không âm $n \in \mathbb{N}$ nào của nhị thức $x+y$ là tổng có dạng*

$$(x+y)^n = \binom{n}{0} x^n y^0 + \binom{n}{1} x^{n-1} y^1 + \binom{n}{2} x^{n-2} y^2 + \cdots + \binom{n}{n} x^0 y^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k = \binom{n}{k} x^k y^{n-k},$$

trong đó mỗi $\binom{n}{k}$ là 1 số nguyên dương được gọi là hệ số nhị thức, được định nghĩa là

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)(n-2)\cdots(n-k+1)}{1 \cdot 2 \cdots (k-1)k}.$$

The final expression follows from the previous one by the symmetry of x & y in the 1st expression, & by comparison it follows that the sequence of binomial coefficients in the formula is symmetrical, $\binom{n}{k} = \binom{n}{n-k}$.

– Biểu thức cuối cùng theo sau biểu thức trước đó bởi tính đối xứng của x & y trong biểu thức thứ nhất, & khi so sánh ta thấy chuỗi hệ số nhị thức trong công thức là đối xứng, $\binom{n}{k} = \binom{n}{n-k}$.

A simple variant of the binomial formula is obtained by substituting 1 for y, so that it involves only a single variable. In this form, the formula reads

$$(x+1)^n = \sum_{i=0}^{n} \binom{n}{i} x^i = \binom{n}{0} x^0 + \binom{n}{1} x^1 + \binom{n}{2} x^2 + \cdots + \binom{n}{n} x^n.$$

Example 7. *The 1st few cases of the binomial theorem are:*

$$\begin{aligned}
(x+y)^0 &= 1,\\
(x+y)^1 &= x+y,\\
(x+y)^2 &= x^2+2xy+y^2,\\
(x+y)^3 &= x^3+3x^2y+3xy^2+y^3,\\
(x+y)^4 &= x^4+4x^3y+6x^2y^2+4xy^3+y^4.
\end{aligned}$$

In general, for the expansion of $(x+y)^n$ on the RHS in the nth row (numbered so that the top row is the 0th row):

1. *the exponents of x in the terms are $n, n-1, \ldots, 2, 1, 0$ (the last term implicitly contains $x^0 = 1$)*
2. *the exponents of y in the terms are $0, 1, 2, \ldots, n-1, n$ (the 1st term implicitly contains $y^0 = 1$)*
3. *the coefficients form the nth row of Pascal's triangle*
4. *before combining like terms, there are 2^n terms $x^i y^j$ in the expansion*
5. *after combining like terms, there are $n+1$ terms, & their coefficients sum to 2^n.*

– Vài trường hợp đầu tiên của định lý nhị thức là:

$$\begin{aligned}
(x+y)^0 &= 1,\\
(x+y)^1 &= x+y,\\
(x+y)^2 &= x^2+2xy+y^2,\\
(x+y)^3 &= x^3+3x^2y+3xy^2+y^3,\\
(x+y)^4 &= x^4+4x^3y+6x^2y^2+4xy^3+y^4.
\end{aligned}$$

Nhìn chung, đối với phép khai triển $(x+y)^n$ trên RHS ở hàng thứ n (được đánh số sao cho hàng trên cùng là hàng thứ 0):

1. *các số mũ của x trong các điều khoản là $n, n-1, \ldots, 2, 1, 0$ (điều khoản cuối cùng ngầm chứa $x^0 = 1$)*
2. *các số mũ của y trong các điều khoản là $0, 1, 2, \ldots, n-1, n$ (điều khoản thứ nhất ngầm chứa $y^0 = 1$)*
3. *các hệ số tạo thành hàng thứ n của tam giác Pascal*
4. *trước khi kết hợp các điều khoản giống nhau, có 2^n điều khoản $x^i y^j$ trong phép khai triển*
5. *sau khi kết hợp các điều khoản giống nhau, có $n+1$ điều khoản, & các hệ số của chúng tổng bằng 2^n.*

2.1.1 Geometric explanation – Giải thích về mặt hình học

For positive values of a & b, the binomial theorem with $n = 2$ is the geometrically evident fact that a square of side $a+b$ can be cut into a square of side a, a square of side b, & 2 rectangles with sides a, b. With $n = 3$, the theorem states that a cube of size $a+b$ can be cut into a cube of size a, a cube of side b, 3 $a \times a \times b$ rectangular boxes, & 3 $a \times b \times b$ rectangular boxes.

– Đối với các giá trị dương của a & b, định lý nhị thức với $n = 2$ là sự thật hiển nhiên về mặt hình học rằng 1 hình vuông có cạnh $a+b$ có thể được cắt thành 1 hình vuông có cạnh a, 1 hình vuông có cạnh b, & 2 hình chữ nhật có cạnh a, b. Với $n = 3$, định lý phát biểu rằng 1 hình lập phương có kích thước $a+b$ có thể được cắt thành 1 hình lập phương có kích thước a, 1 hình lập phương có cạnh b, 3 $a \times a \times b$ hộp chữ nhật, & 3 $a \times b \times b$ hộp chữ nhật.

In calculus, this picture also gives a geometric proof of the derivative $(x^n)' = nx^{n-1}$: if one sets $a = x$ & $b = \Delta x$, interpreting b as an infinitesimal change in a, then this picture shows the infinitesimal change in the volume of an n-dimensional hypercube, $(x + \Delta x)^n$, where the coefficient of the linear term (in Δx) is nx^{n-1}, the area of the n faces, each of dimension $n - 1$:

$$(x + \Delta x)^n = x^n + nx^{n-1}\Delta x + \binom{n}{2} x^{n-2}(\Delta x)^2 + \cdots .$$

Substituting this into the definition of the derivative via a difference quotient & taking limits means that the higher order terms, $(\Delta x)^k$ with $k \geq 2$, become negligible, & yields the formula $(x^n)' = nx^{n-1}$, interpreted as "the infinitesimal rate of change in volume of an n-cube as side length varies is the area of n of its $(n-1)$-dimensional faces". If one integrates this picture, which corresponds to applying the fundamental theorem of calculus, one obtains Cavalieri's quadrature formula, the integral $\int x^{n-1}\,\mathrm{d}x = \frac{1}{n}x^n$, see, e.g., Wikipedia/proof of Cavalieri's quadrature formula.

– Trong phép tính, hình ảnh này cũng đưa ra 1 bằng chứng hình học về đạo hàm $(x^n)' = nx^{n-1}$: nếu ta đặt $a = x$ & $b = \Delta x$, giải thích b là 1 thay đổi vô cùng nhỏ trong a, thì hình ảnh này cho thấy sự thay đổi vô cùng nhỏ trong thể tích của 1 siêu khối n chiều, $(x + \Delta x)^n$, trong đó hệ số của số hạng tuyến tính (trong Δ) là nx^{n-1}, diện tích của n mặt, mỗi mặt có chiều $n - 1$:

$$(x + \Delta x)^n = x^n + nx^{n-1}\Delta x + \binom{n}{2} x^{n-2}(\Delta x)^2 + \cdots .$$

Thay thế điều này vào định nghĩa của đạo hàm thông qua thương số hiệu & lấy giới hạn có nghĩa là các số hạng bậc cao hơn, $(\Delta x)^k$ với $k \geq 2$, trở nên không đáng kể, & tạo ra công thức $(x^n)' = nx^{n-1}$, được hiểu là "tốc độ thay đổi vô cùng nhỏ về thể tích của 1 khối lập phương n khi độ dài cạnh thay đổi là diện tích của n của các mặt $(n-1)$ chiều". Nếu ta tích hợp hình ảnh này, tương ứng với việc áp dụng định lý cơ bản của phép tính, ta sẽ thu được công thức tích phân Cavalieri, tích phân $\int x^{n-1}\,\mathrm{d}x = \frac{1}{n}x^n$, xem, e.g., Wikipedia/bằng chứng về công thức tích phân Cavalieri.

Bài toán 32. *Chứng minh: (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)*

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

1st chứng minh. (i) Viết $(a+b)^n = (a+b)(a+b)\cdots(a+b)$, trong mỗi nhân tử, chỉ có thể chọn a hoặc b. Nên số cách chọn được $i \in \overline{0,n}$ biến b từ i nhân tử $(a+b)$ & $n-i$ biến a bằng $C_n^i = C_n^{n-i}$, nên hệ số của $a^{n-i}b^i$ bằng $C_n^i = C_n^{n-i}$. (ii) Cho $a = b = 1$ trong nhị thức Newton, được $2^n = (1+1)^n = \sum_{i=0}^n C_n^i$. (iii) Cho $a = 1, b = -1$ trong nhị thức Newton, được $0 = 0^n = (1-1)^n = \sum_{i=0}^n C_n^i(-1)^i$. □

2nd chứng minh. Chứng minh bằng quy nạp toán học.

(a) Hiển nhiên đúng với $n = 0$: RHS $= C_0^0 a^{0-0}b^0 = 1 = (a+b)^0 =$ LHS, & đúng với $n = 1$: RHS $= C_1^0 a^{1-0}b^0 + C_1^1 a^{1-1}b^1 = a + b = (a+b)^1 =$ LHS. Giả sử $(a+b)^n = \sum_{i=0}^n C_n^i a^{n-i}b^i$, $\forall a, b \in \mathbb{R}$ với $n \in \mathbb{N}$ nào đó. Có:

$$\begin{aligned}(a+b)^{n+1} = (a+b)(a+b)^n = (a+b)\sum_{i=0}^n C_n^i a^{n-i}b^i &= \sum_{i=0}^n C_n^i a^{n-i+1}b^i + \sum_{i=0}^n C_n^i a^{n-i}b^{i+1} \\ &= \sum_{i=0}^n C_n^i a^{n-i+1}b^i + \sum_{i=1}^{n+1} C_n^{i-1} a^{n-i+1}b^i \\ &= C_n^0 a^{n+1}b^0 + \sum_{i=1}^n (C_n^i + C_n^{i-1})a^{n-i+1}b^i + C_n^{n+1-1}a^0 b^{n+1} = \sum_{i=0}^{n+1} C_{n+1}^i a^{n+1-i}b^i.\end{aligned}$$

Theo nguyên lý quy nạp toán học, công thức nhị thức Newton đúng $\forall n \in \mathbb{N}$. □

2.2 Multinomial Theorem – Định Lý Khai Triển Multinomial

Resources – Tài nguyên.

1. Wikipedia/multinomial theorem.

In mathematics, the *multinomial theorem* describes how to expand a power of a sum in terms of powers of the terms in that sum. It is the generalization of the binomial theorem from binomials to multinomials.

– Trong toán học, *định lý đa thức* mô tả cách khai triển lũy thừa của 1 tổng theo lũy thừa của các số hạng trong tổng đó. Đây là sự tổng quát hóa của định lý nhị thức từ nhị thức sang đa thức.

Bài toán 33. *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)* $(\sum_{i=1}^m a_i)^n$, $\forall m, n \in \mathbb{N}^\star$, $\forall a_i \in \mathbb{R}$, $\forall i = 1, \ldots, m$. *(d)* z^n, $\forall z \in \mathbb{C}$, $\forall n \in \mathbb{N}^\star$ *với* $z = a + bi$, $a := \Re z$, $b := \Im z$.

Solution. (a) Khai triển nhị thức 2 lần liên tiếp:

$$(a+b+c)^n = ((a+b)+c)^n = \sum_{i=0}^{n} C_n^i (a+b)^{n-i} c^i = \sum_{i=0}^{n} C_n^i \left(\sum_{j=0}^{n-i} C_{n-i}^j a^{n-i-j} b^j \right) c^i = \sum_{i=0}^{n} \sum_{j=0}^{n-i} C_n^i C_{n-i}^j a^{n-i-j} b^j c^i,$$

với hệ số của $a^{n-i-j}b^j c^i$ bằng $C_n^i C_{n-i}^j = \dfrac{n!}{i!(n-i)!} \cdot \dfrac{(n-i)!}{j!(n-i-j)!} = \dfrac{n!}{i!j!(n-i-j)!}$, nên nếu đặt $k := n-i-j$ thì $i+j+k=n$, suy ra

$$(a+b+c)^n = \sum_{i=0}^{n} \sum_{j=0}^{n-i} \sum_{k=0}^{n-i-j} \frac{n!}{i!j!k!} a^k b^j c^i = \sum_{i,j,k \in [n],\, i+j+k=n} \frac{n!}{i!j!k!} a^i b^j c^k = \sum_{i,j,k \in [n],\, i+j+k=n} \binom{n}{i,j,k} a^i b^j c^k,$$

với hệ số multinomial $\binom{n}{i,j,k} := \dfrac{n!}{i!j!k!}$ (có thể khai triển bằng cách $(a+b+c)^n = (a+(b+c))^n = ((a+c)+b)^n$ vẫn sẽ thu được cùng kết quả).

(b) Có 2 chiến thuật: tách $4 = 2+2$ hoặc $4 = 1+3$. Chiến thuật $4 = 2+2$ nghĩa là:

$$\begin{aligned}((a+b)+(c+d))^n &= \sum_{i=0}^{n} C_n^i (a+b)^{n-i} (c+d)^i = \sum_{i=0}^{n} C_n^i \left(\sum_{j=0}^{n-i} C_{n-i}^j a^{n-i-j} b^j \right) \left(\sum_{k=0}^{i} C_i^k c^{i-k} d^k \right) \\ &= \sum_{i=0}^{n} \sum_{j=0}^{n-i} \sum_{k=0}^{i} C_n^i C_{n-i}^j C_i^k a^{n-i-j} b^j c^{i-k} d^k,\end{aligned}$$

với hệ số của $a^{n-i-j}b^j c^{i-k}d^k$ bằng $C_n^i C_{n-i}^j C_i^k = \dfrac{n!}{i!(n-i)!} \cdot \dfrac{(n-i)!}{j!(n-i-j)!} \cdot \dfrac{i!}{k!(i-k)!} = \dfrac{n!}{j!k!(i-k)!(n-i-j)!}$, nên nếu đặt $\alpha := n-i-j, \beta := j, \gamma := i-k, \delta = k$ thì $\alpha+\beta+\gamma+\delta = n$, suy ra

$$((a+b)+(c+d))^n = \sum_{\alpha,\beta,\gamma,\delta \in [n],\ \alpha+\beta+\gamma+\delta=n} \frac{n!}{\alpha!\beta!\gamma!\delta!} a^\alpha b^\beta c^\gamma d^\delta = \sum_{\alpha,\beta,\gamma,\delta \in [n],\ \alpha+\beta+\gamma+\delta=n} \binom{n}{\alpha,\beta,\gamma,\delta} a^\alpha b^\beta c^\gamma d^\delta,$$

với hệ số multinomial $\binom{n}{\alpha,\beta,\gamma,\delta} := \dfrac{n!}{\alpha!\beta!\gamma!\delta!}$ (có thể khai triển bằng cách $(a+b+c+d)^n = ((a+c)+(d+d))^n = ((a+d)+(b+c))^n$ vẫn sẽ thu được cùng kết quả).

Chiến thuật $4 = 1+3$ nghĩa là:

$$\begin{aligned}((a+b+c)+d)^n &= \sum_{l=0}^{n} C_n^l (a+b+c)^{n-l} d^l = \sum_{l=0}^{n} C_n^l \left(\sum_{i,j,k \in [n-l],\, i+j+k=n-l} \binom{n-l}{i,j,k} a^i b^j c^k \right) d^l \\ &= \sum_{l=0}^{n} \sum_{i,j,k \in [n-l],\, i+j+k=n-l} C_n^l \binom{n-l}{i,j,k} a^i b^j c^k d^l,\end{aligned}$$

với hệ số của $a^i b^j c^k d^l$ bằng $C_n^l \binom{n-l}{i,j,k} = \dfrac{n!}{l!(n-l)!} \cdot \dfrac{(n-l)!}{i!j!k!} = \dfrac{n!}{i!j!k!l!} = \binom{n}{i,j,k,l}$, nên

$$\begin{aligned}((a+b+c)+d)^n &= \sum_{l=0}^{n} \sum_{i,j,k \in [n-l],\, i+j+k=n-l} \binom{n}{i,j,k,l} a^i b^j c^k d^l = \sum_{l=0}^{n} \sum_{i,j,k \in [n-l],\, i+j+k=n-l} \binom{n}{i,j,k,l} a^i b^j c^k d^l \\ &= \sum_{i,j,k,l \in [n],\, i+j+k+l=n} \binom{n}{i,j,k,l} a^i b^j c^k d^l.\end{aligned}$$

(có thể khai triển bằng cách $(a+b+c+d)^n = ((a+b+d)+c)^n = ((a+c+d)+b)^n = (a+(b+c+d))^n$ vẫn sẽ thu được cùng kết quả).

(c) □

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++/VNTA_Pascal_triangle_multinomial.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;
}

void pascalTriangle(int n) {
    cout << "1st " << n + 1 << " lines of the Pascal triangle: \n";
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= i; j++) {
            cout << nCk(i, j) << " ";
        }
        cout << '\n';
    }
    cout << "\n=========================================\n";
}

void generatePartitions(int n, int m, int idx, vector<int>& cur, vector<vector<int>>& res) {
    if (idx == m - 1) {
        cur[idx] = n;
        res.push_back(cur);
        return;
    }
    for (int i = 0; i <= n; i++) {
        cur[idx] = i;
        generatePartitions(n - i, m, idx + 1, cur, res);
    }
}

double multinomialExpansion(int n, const vector<double>& a) {
    int m = a.size();
    vector<vector<int>> partitions;
    vector<int> cur(m, 0);
    generatePartitions(n, m, 0, cur, partitions);

    double res = 0;
    for (const auto& k : partitions) {
        unsigned long long coeff = Pn(n);
        double term = 1;
        for (int i = 0; i < m; i++) {
            coeff /= Pn(k[i]);
            term *= pow(a[i], k[i]);
        }
        res += coeff * term;
```

```
    }
    return res;
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    int n, m;
    cin >> n >> m;

    pascalTriangle(n);

    vector<double> a(m);
    for (int i = 0; i < m; i++) cin >> a[i];
    double result = multinomialExpansion(n, a);
    if (m > 2) cout << "(a1 +... + a" << m << ")^" << n << " = " << result << "\n";
    else cout <<  "(a1 + a2)^" << n << " = " << result << "\n";
}
```

2.3 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*).

2.3.1 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 10 (Pascal's rule)**.** *One has*

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

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

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

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

Problem 38. *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 $n \in \mathbb{N}^\star$.

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

□

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

2.4 Multinomial distribution – Phân Phối Multinomial

2.5 Problem: Newton Binomial Theorem

Bài toán 34 ([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 35 ([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 36 ([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 37 ([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 & 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 & 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, & "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 38 (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 8. *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 39 (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

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

are related by the divisor sum

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

Theorem 12. *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*

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

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 9. *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 40 ([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 40. *(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 41. *Dùng hàm sinh, giải phương trình nghiệm nguyên*

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

Chương 5

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

Contents

Resources – Tài nguyên.

1. Wikipedia/inclusion–exclusion principle. [Sha22]. Shahriar Shahriari. *An Invitation To Combinatorics*. Chap. 8: The Inclusion–Exclusion Principle.

5.1 Some Variants of Inclusion–Exclusion Principle – Vài Biến Thể của 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.

The inclusion–exclusion principle is a counting method that generalizes familiar ideas. The addition principle told us that, if a set was partitioned into subsets, we could count the number of elements in the set by adding the number of elements in each part. In a partition, the subsets cover the original set – i.e., the union of the subsets is the original set – & the pairwise intersection of the subsets is the empty set. The inclusion–exclusion principle extends the addition principle to the case when we have a collection of subsets that cover the original set but – unlike a partition – may have nontrivial intersections.

– Nguyên lý bao gồm-loại trừ là 1 phương pháp đếm khái quát hóa các ý tưởng quen thuộc. Nguyên lý cộng cho chúng ta biết rằng, nếu 1 tập hợp được phân hoạch thành các tập con, chúng ta có thể đếm số phần tử trong tập hợp bằng cách cộng số phần tử trong mỗi phần. Trong 1 phân hoạch, các tập con bao phủ tập hợp ban đầu – tức là, hợp của các tập con là tập hợp ban đầu – & giao từng cặp của các tập con là tập rỗng. Nguyên lý bao gồm-loại trừ mở rộng nguyên lý cộng sang trường hợp khi chúng ta có 1 tập hợp các tập con bao phủ tập hợp ban đầu nhưng – không giống như 1 phân hoạch – có thể có các giao điểm không tầm thường.

Remark 10 ([Sha22], Rmk. 8.2, p. 261)**.** *The inclusion–exclusion principle is also an example of a sieve method. In a sieve method you start with a set larger than what you are interested in & systematically throw out extraneous elements. Historically, an early use of a sieve method was by* Eratosthenes. *For the sieve of Eratostheses, you start with the integers from* 2 *to* n *&, 1 by 1, cross out the multiples of the 1st integer that has not been crossed out (you don't cross out that integer itself & you stop when you get to* $\sqrt{n}$*). At the end of the process, you are left with all the primes up to* n*.*

– Nguyên lý bao gồm-loại trừ cũng là 1 ví dụ về phương pháp sieve. Trong phương pháp sieve, bạn bắt đầu với 1 tập hợp lớn hơn tập hợp bạn quan tâm & loại bỏ 1 cách có hệ thống các phần tử không liên quan. Theo truyền thống, 1 ứng dụng sớm của phương pháp sieve là của Eratosthenes. *Đối với sieve của Eratostheses, bạn bắt đầu với các số nguyên từ* 2 *đến* n *&, 1 x 1, gạch bỏ các bội số của số nguyên đầu tiên chưa bị gạch bỏ (bạn không gạch bỏ chính số nguyên đó & bạn dừng lại khi bạn đến* $\sqrt{n}$*). Khi kết thúc quá trình, bạn còn lại tất cả các số nguyên tố lên đến* n*.*

Definition 17 (Complement, [Sha22], Def. 8.3, p. 261)**.** *Let* S *be a set &* $A \subset S$*. Then the* complement *of* A *in* S*, written* A^{c}_S*, is defined by* $A^{\mathrm{c}}_S = S - A = \{x \in S; x \notin A\}$*.*

Định lý 4 (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 13 (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 14 (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 15 (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

$$\mathbb{P}\left(\bigcup_{i=1}^{n}A_i\right)=\sum_{i=1}^{n}\mathbb{P}(A_i)-\sum_{1\le i<j\le n}\mathbb{P}(A_i\cap A_j)+\sum_{1\le i<j<k\le n}\mathbb{P}(A_i\cap A_j\cap A_k)+\cdots+(-1)^{n-1}\mathbb{P}\left(\bigcap_{i=1}^{n}A_i\right),\tag{5.1}$$

which can be written in closed form as

$$\mathbb{P}\left(\bigcup_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} (-1)^{i-1} \sum_{I\subset[n],\,|I|=i} \mathbb{P}(A_I), \tag{5.2}$$

where the last sum runs over all subsets I of the indices $1,\dots,n$ which contain exactly i elements, & $A_I := \bigcap_{i\in I} A_i$ 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 (5.2) 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 41. *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 16 (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. 13 & the probabilistic version Thm. 15 of the inclusion-exclusion principle are instances of 16.

5.2 Problems on inclusion–exclusion principle

Problem 42 ([Sha22], p. 261)**.** *You compose 10 different emails, each to a different friend. The prank-ware on your email server permutes the recipients' email addresses & randomly sends each of the messages to 1 of the individuals on your list. What is the probability that none of your fiends receives the email intended for them? Would this probability go up or down if instead of 10 emails, you had written* 10000 *emails?*

Problem 43. *A coffee company spent 10 cents per person to question patrons about their coffee preferences. Of the persons interviewed, $a = 250$ liked latte, $b = 200$ liked cappuccino, $c = 80$ liked both, & 100 did not like either choice. What was the total amount of money the company had to pay?*

5.3 Derangement – Sự rối loạn/Số hoán vị sai vị trí

Resources – Tài nguyên.

1. Wikipedia/derangement.

In combinatorial mathematics, a *derangement* is a permutation of the elements of a set in which no element appears in its original position. I.e., a derangement is a permutation that has no fixed points.

– Trong toán học tổ hợp, 1 *derangement* là 1 hoán vị của các phần tử của 1 tập hợp trong đó không có phần tử nào xuất hiện ở vị trí ban đầu của nó. Nghĩa là, 1 derangement là 1 hoán vị không có điểm cố định.

The number of derangements of a set of size $n \in \mathbb{N}^\star$ is known as the *subfactorial* of n or the nth *derangement number* or nth *de Montmort number* (after Pierre Remond de Montmort). Notations for subfactorials in common use include $!n, D_n, d_n$, or $n_¡$.

– Số lượng các biến đổi của 1 tập hợp có kích thước $n \in \mathbb{N}^\star$ được gọi là *subfactorial* của n hoặc *derangement number* thứ n hoặc *de Montmort number* thứ n (theo Pierre Remond de Montmort). Các ký hiệu cho các biến đổi thường dùng bao gồm $!n, D_n, d_n$ hoặc $n_¡$.

For $n > 0$, the subfactorial $!n$ equals too the nearest integer to $\frac{n!}{e}$, where $n!$ denotes the factorial of n & e is Euler's number.

The problem of counting derangements was 1st considered by Pierre Raymond de Montmort in his *Essay d'analyse sur les jeux de hazard* in 1708; he solved it in 1713, as did Nicholas Bernoulli at about the same time.

Problem 44 (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?

Example 10. *Suppose that a professor gave a test to 4 students – A, B, C, & D – & wants to let them grade each other's tests. Of course, no student should grade their own test. How many ways could the professor hand the tests back to the students for grading, s.t. no student receives their own test back? Out of 24 possible permutations 4! for handing back the tests, ABCD, ABDC, ACBD, ACDB, ADBC, ADCB, BACD, BADC, BCAD, BCDA, BDAC, BDCA, CABD, CADB, CBAD, CBDA, CDAB, CDBA, DABC, DACB, DBAC, DBCA, DCAB, DCBA. there are only 9 derangements BADC, BCDA, BDAC, CADB, CDAB, CDBA, DABC, DCAB, DCBA. In every other permutation of this 4-member set, at least 1 student gets their own test back.*

– Giả sử 1 giáo sư giao bài kiểm tra cho 4 sinh viên – A, B, C, & D – & muốn để họ chấm bài kiểm tra của nhau. Tất nhiên, không có sinh viên nào được chấm bài kiểm tra của chính mình. Có bao nhiêu cách giáo sư có thể trả bài kiểm tra lại cho sinh viên để chấm, tức là không có sinh viên nào nhận lại bài kiểm tra của chính mình? Trong số 24 cách hoán vị có thể có 4! để trả bài kiểm tra, ABCD, ABDC, ACBD, ACDB, ADBC, ADCB, BACD, BADC, BCAD, BCDA, BDAC, BDCA, CABD, CADB, CBAD, CBDA, CDAB, CDBA, DABC, DACB, DBAC, DBCA, DCAB, DCBA. chỉ có 9 cách sắp xếp BADC, BCDA, BDAC, CADB, CDAB, CDBA, DABC, DCAB, DCBA. Trong mỗi hoán vị khác của tập 4 phần tử này, có ít nhất 1 sinh viên nhận lại bài kiểm tra của chính mình.

Another version of the problem arises when we ask for the number of ways n letters, each addressed to a different person, can be placed in n pre-addressed envelopes so that no letter appears in the correctly addressed envelope.

– 1 phiên bản khác của bài toán này nảy sinh khi chúng ta yêu cầu tìm số cách để bỏ n lá thư, mỗi lá thư được gửi đến 1 người khác nhau, vào n phong bì có ghi sẵn địa chỉ sao cho không có lá thư nào xuất hiện trong phong bì có ghi đúng địa chỉ.

5.3.1 Counting derangements – Đếm số hoán vị sai vị trí

Counting derangements of a set amounts to the *hat-check problem*, in which one considers the number of ways in which n hats (call them h_1 through h_n) can be returned to n people (P_1 through P_n) s.t. no hat makes it back to its owner.

– Đếm các sự xáo trộn của 1 tập hợp tương đương với *bài toán kiểm tra mũ*, trong đó người ta xem xét số cách mà n chiếc mũ (gọi là h_1 đến h_n) có thể được trả lại cho n người (P_1 đến P_n) nghĩa là không có chiếc mũ nào được trả lại cho chủ nhân của nó.

Each person may receive any of the $n-1$ hats that is not their own. Call the hat which the person P_1 receives h_i & consider h_i's owner: P_i receives either P_1's hat, h_1, or some other. Accordingly, the problem splits into 2 possible cases:

1. P_i receives a hat other than h_1. This case is equivalent to solving the problem with $n-1$ people & $n-1$ hats because for each of the $n-1$ people besides P_1 there is exactly 1 hat from among the remaining $n-1$ hats that they may not receive (for any P_j besides P_i, the unreceivable hat is h_j, while for P_i it is h_1). Another way to see this is to rename h_1 to h_i, where the derangement is more explicit: for any j from 2 to n, P_j cannot receive h_j.

2. P_i receives h_1. In this case the problem reduces to $n-2$ people & $n-2$ hats, because P_1 received h_i's hat & P_i received h_1's hat, effectively putting both out of further consideration.

For each of the $n-1$ hats that P_1 may receive, the number of ways that $P_2, \ldots, P_n$ may all receive hats is the sum of the counts for the 2 cases. This gives us the solution to the hat-check problem: Stated algebraically, the number $!n$ of derangements of an n-element set is

$$!0 = 1,\ !1 = 0,\ !n = (n-1)(!(n-1)+!(n-2)),\ \forall n \in \mathbb{N},\ n \geq 2.$$

– Mỗi người có thể nhận được bất kỳ chiếc mũ nào trong số $n-1$ chiếc mũ không phải của mình. Gọi chiếc mũ mà người P_1 nhận được là h_i & xét đến chủ sở hữu của h_i: P_i nhận được mũ của P_1, h_1 hoặc 1 chiếc mũ khác. Theo đó, bài toán chia thành 2 trường hợp có thể xảy ra:

1. P_i nhận được 1 chiếc mũ khác với h_1. Trường hợp này tương đương với việc giải bài toán với $n-1$ người & $n-1$ chiếc mũ vì đối với mỗi $n-1$ người ngoài P_1 thì có đúng 1 chiếc mũ trong số $n-1$ chiếc mũ còn lại mà họ không được nhận (đối với bất kỳ P_j nào ngoài P_i, chiếc mũ không được nhận là h_j, trong khi đối với P_i thì là h_1). 1 cách khác để thấy điều này là đổi tên h_1 thành h_i, trong đó sự sắp xếp rõ ràng hơn: đối với bất kỳ j nào từ 2 đến n, P_j không thể nhận được h_j.

2. P_i nhận được h_1. Trong trường hợp này, bài toán được rút gọn thành $n-2$ người & $n-2$ mũ, vì P_1 nhận được mũ của h_i & P_i nhận được mũ của h_1, về cơ bản là loại cả hai ra khỏi việc xem xét thêm.

Đối với mỗi $n-1$ mũ mà P_1 có thể nhận được, số cách mà $P_2, \ldots, P_n$ có thể nhận được mũ là tổng số đếm của 2 trường hợp. Điều này cung cấp cho chúng ta giải pháp cho bài toán kiểm tra mũ: Nói theo đại số, số $!n$ các phép sắp xếp của 1 tập hợp n phần tử là

$$!0 = 1,\ !1 = 0,\ !n = (n-1)(!(n-1)+!(n-2)),\ \forall n \in \mathbb{N},\ n \geq 2.$$

The number of derangements of small lengths, (sequence A000166 in the OEIS), is given as follows. $!0 = 1, !1 = 1, !2 = 1, !3 = 2, !4 = 9, !5 = 44, !6 = 265, !7 = 1,854, !8 = 14,833, !9 = 133,496, !10 = 1,334,961, !11 = 14,684,570, !12 = 176,214,841, !13 = 2,290,792,932, !14 = 32,071,101,049, !15 = 481,066,515,734, !16 = 7,697,064,251,745, !17 = 130,850,092,279,664, !18 = 2,355,301,661,033,953, !19 = 44,750,731,559,645,106, !20 = 895,014,631,192,902,121, !21 = 18,795,307,255,050,944,540, !22 = 413,496,759,611,120,779,881l!23 = 9,510,425,471,055,777,937,262, !24 = 228,250,211,305,338,670,494,289, !25 = 5,706,255,282,6 148,362,637,348,470,135,821,287,825, !27 = 4,005,791,208,408,693,667,174,771,274, !28 = 112,162,153,835,443,422,680,893,595, 3,252,702,461,227,859,257,745,914,274,516, !30 = 97,581,073,836,835,777,732,377,428,235,481, \ldots$

There are various other expressions for $!n$, equivalent to the formula given above.

Theorem 17. *The number of derangements of an n-element set can be given by the following formulas:*

$$!n = n! \sum_{i=0}^{n} \frac{(-1)^n}{i!}, \ \forall n \in \mathbb{N},$$
$$!n = \left[\frac{n!}{e}\right] = \left\lfloor \frac{n!}{e} + \frac{1}{2} \right\rfloor, \ \forall n \in \mathbb{N}^\star,$$

where $[x]$ is the nearest integer function & $\lfloor x \rfloor$ is the floor function, or

$$!n = \lfloor \frac{n! + 1}{e} \rfloor, \ \forall n \in \mathbb{N}^\star,$$
$$!n = \lfloor \left(e + \frac{1}{e}\right) n! \rfloor - \lfloor en! \rfloor, \ \forall n \in \mathbb{N}, \ n \geq 2,$$
$$!n = n! - \sum_{i=1}^{n} \binom{n}{i} \cdot !(n-i), \ \forall n \in \mathbb{N}^\star,$$
$$!n = n! \sum_{i=0}^{n} \frac{(-1)^i}{i!}.$$

The following recurrence also holds:

$$!n = \begin{cases} 1 & \text{if } n = 0, \\ n(!(n-1)) + (-1)^n & \text{if } n \in \mathbb{N}^\star. \end{cases}$$

5.3.2 Derivation of derangements by inclusion–exclusion principle – Sự suy ra derangement bằng nguyên lý bao gồm-loại trừ

Theorem 18. *The following formulas hold*

$$!n = n! \sum_{i=0}^{n} \frac{(-1)^i}{i!}, \ \forall n \in \mathbb{N},$$
$$n! = \sum_{i=0}^{n} \binom{n}{i} !i, \ \forall n \in \mathbb{N}.$$

One may derive a non-recursive formula for the number of derangements of an n-set as well. For $1 \leq k \leq n$ we define S_k to be the set of permutations of n objects that fix the kth object. Any intersection of a collection of i of these sets fixes a particular set of i objects & therefore contains $(n-i)!$ permutations. There are $\binom{n}{i}$ such collections, so the inclusion-exclusion principle yields

$$|S_1 \cup \cdots \cup S_n| = \sum_i |S_i| - \sum_{i<j} |S_i \cap S_j| + \sum_{i<j<k} |S_i \cap S_j \cap S_k| + \cdots + (-1)^{n+1} |S_1 \cap \cdots \cap S_n|$$
$$= \binom{n}{1}(n-1)! - \binom{n}{2}(n-2)! + \binom{n}{3}(n-3)! - \cdots + (-1)^{n+1}\binom{n}{n}0! = \sum_{i=1}^{n} (-1)^{i+1} \binom{n}{i} (n-i)! = n! \sum_{i=1}^{n} \frac{(-1)^{i+1}}{i!},$$

& since a derangement is a permutation that leaves none of the n objects fixed, this implies

$$!n = n! - |S_1 \cup \cdots \cup S_n| = n! \sum_{i=0}^{n} \frac{(-1)^i}{i!}.$$

On the other hand, $n! = \sum_{i=0}^{n} \binom{n}{i} !i$ since we can choose $n-i$ elements to be in their own place & derange the other i elements in just $!i$ ways, by definition.

– Người ta cũng có thể suy ra 1 công thức không đệ quy cho số lần sắp xếp của 1 tập n. Với $1 \leq k \leq n$, chúng ta định nghĩa S_k là tập hợp các hoán vị của n đối tượng cố định đối tượng thứ k. Bất kỳ giao điểm nào của 1 tập hợp i trong số các tập hợp này đều cố định 1 tập hợp cụ thể gồm i đối tượng & do đó chứa $(n-i)!$ hoán vị. Có $\binom{n}{i}$ các tập hợp như vậy, vì vậy nguyên lý bao gồm-loại trừ tạo ra

$$\begin{aligned}|S_1 \cup \cdots \cup S_n| &= \sum_i |S_i| - \sum_{i<j} |S_i \cap S_j| + \sum_{i<j<k} |S_i \cap S_j \cap S_k| + \cdots + (-1)^{n+1} |S_1 \cap \cdots \cap S_n| \\ &= \binom{n}{1}(n-1)! - \binom{n}{2}(n-2)! + \binom{n}{3}(n-3)! - \cdots + (-1)^{n+1}\binom{n}{n}0! = \sum_{i=1}^{n}(-1)^{i+1}\binom{n}{i}(n-i)! = n!\sum_{i=1}^{n}\frac{(-1)^{i+1}}{i!},\end{aligned}$$

& vì 1 sự sắp xếp là 1 hoán vị không để lại bất kỳ đối tượng nào trong số n đối tượng cố định, điều này ngụ ý

$$!n = n! - |S_1 \cup \cdots \cup S_n| = n!\sum_{i=0}^{n}\frac{(-1)^i}{i!}.$$

Mặt khác, $n! = \sum_{i=0}^{n}\binom{n}{i}!i$ vì chúng ta có thể chọn $n-i$ phần tử ở đúng vị trí của chúng & sắp xếp lại các phần tử i khác theo đúng $!i$ cách, theo định nghĩa.

5.3.3 Growth of number of derangements as n approaches ∞

From $!n = n!\sum_{i=0}^{n}\frac{(-1)^i}{i!}$ & $e^x = \sum_{i=0}^{\infty}\frac{x^i}{i!}$ by substituting $x = -1$ one immediately obtains that

$$\lim_{n\to\infty}\frac{!n}{n!} = \lim_{n\to\infty}\sum_{i=0}^{n}\frac{(-1)^i}{i!} = \frac{1}{e} \approx 0.36787944117144233\ldots$$

This is the limit of the probability that a randomly selected permutation of a large number of objects is a derangement. The probability converges to this limit extremely quickly as n increases, which is why $!n$ is the nearest integer to $\frac{n!}{e}$. The above semi-log graph shows that the derangement graph lags the permutation graph by an almost constant value. More information about this calculation & the above limit may be found in the article on the statistics of random permutations.

– Đây là giới hạn của xác suất mà 1 hoán vị được chọn ngẫu nhiên của 1 số lượng lớn các đối tượng là 1 sự sắp xếp. Xác suất hội tụ đến giới hạn này cực kỳ nhanh khi n tăng, đó là lý do tại sao $!n$ là số nguyên gần nhất với $\frac{n!}{e}$. Đồ thị bán logarit ở trên cho thấy đồ thị sắp xếp chậm hơn đồ thị hoán vị 1 giá trị gần như hằng số. Có thể tìm hiểu thêm thông tin về phép tính này & giới hạn trên trong bài viết về thống kê hoán vị ngẫu nhiên.

5.3.4 Asymptotic expansion in terms of Bell numbers – Mở rộng tiệm cận theo số Bell

Theorem 19 (Asymptotic expansion for the number of derangements in terms of Bell numbers)**.** *An asymptotic expansion for the number of derangements in terms of Bell numbers is as follows:*

$$!n = \frac{n!}{e} + \sum_{i=1}^{m}(-1)^{n+i-1}\frac{B_i}{n^i} + O\left(\frac{1}{n^{m+1}}\right),$$

where $m \in \mathbb{N}^\star$ is any fixed positive integer, & B_i denotes the ith Bell number. Moreover, the constant implied by the big O-term does not exceed B_{m+1}.

5.3.5 Generalizations of derangements – Các tổng quát hóa của hoán vị sai vị trí

The problème des rencontres asks how many permutations of a size-n set have exactly k fixed points. Derangements are an example of the wider field of constrained permutations. E.g., the ménage problem asks if $n \in \mathbb{N}^\star$ opposite-sex couples are seated man-woman-man-woman-... around a table, how many ways can they be seated so that nobody is seated next to his or her partner?

– Bài toán về sự gặp gỡ hỏi có bao nhiêu hoán vị của 1 tập hợp có kích thước n có đúng k điểm cố định. Sự sắp xếp là 1 ví dụ về lĩnh vực rộng hơn của các hoán vị bị ràng buộc. E.g., bài toán ménage hỏi nếu $n \in \mathbb{N}^\star$ cặp đôi khác giới ngồi theo kiểu nam-nữ-nam-nữ-... quanh 1 chiếc bàn, có bao nhiêu cách sắp xếp chỗ ngồi cho họ sao cho không ai ngồi cạnh bạn đời của mình?

More formally, given sets A, S, & some sets U, V of surjections $A \to S$, we often wish to know the number of pairs of functions (f, g) s.t. $f \in U, g \in V$, & $\forall a \in, f(a) \neq g(a)$; i.e., where for each f, g, there exists a derangement φ of S s.t. $f(a) = \varphi(g(a))$.

– Chính thức hơn, cho các tập A, S, & 1 số tập U, V các phép toàn ánh $A \to S$, chúng ta thường muốn biết số cặp hàm (f, g) s.t. $f \in U, g \in V$, & $\forall a \in, f(a) \neq g(a)$; tức là, với mỗi f, g, tồn tại 1 phép biến đổi φ của S s.t. $f(a) = \varphi(g(a))$.

Another generalization:

Problem 45. *How many anagrams with no fixed letters of a given word are there?*
– Có bao nhiêu từ đảo chữ không có chữ cái cố định của 1 từ nhất định?

E.g., for a word made of only 2 different letters, say $n \in \mathbb{N}^\star$ letters A & $m \in \mathbb{N}^\star$ letters B, the answer is, of course, 1 or 0 according to whether $n = m$ or not, for the only way to form an anagram without fixed letters is to exchange all the A with B, which is possible iff $n = m$. In general case, for a word with n_i letters X_i, $\forall i \in [r]$, it turns out (after a proper use of the inclusion–exclusion formula) that the answer has the form

$$\int_0^\infty \prod_{i=1}^{r} P_{n_i}(x)e^{-x}\,\mathrm{d}x = \int_0^\infty P_{n_1}(x)P_{n_2}(x)\cdots P_{n_r}(x)e^{-x}\,\mathrm{d}x$$

for a certain sequence of polynomials $\{P_n\}_{n=1}^\infty$, where $\deg P_n = n$. But the above answer for the case $r = 2$ gives an orthogonality relation, whence the P_n's are the Laguerre polynomials up to a sign that is easily decided.

– E.g., đối với 1 từ chỉ được tạo thành từ 2 chữ cái khác nhau, chẳng hạn $n \in \mathbb{N}^\star$ chữ cái A & $m \in \mathbb{N}^\star$ chữ cái B, thì câu trả lời tất nhiên là 1 hoặc 0 tùy thuộc vào việc $n = m$ hay không, vì cách duy nhất để tạo thành 1 từ đảo chữ mà không có chữ cái cố định là hoán đổi tất cả A bằng B, điều này có thể thực hiện được khi & chỉ khi $n = m$. Trong trường hợp chung, đối với 1 từ có n_i chữ cái X_i, $\forall i \in [r]$, thì (sau khi sử dụng đúng công thức bao gồm-loại trừ) câu trả lời có dạng

$$\int_0^\infty \prod_{i=1}^{r} P_{n_i}(x)e^{-x}\,\mathrm{d}x = \int_0^\infty P_{n_1}(x)P_{n_2}(x)\cdots P_{n_r}(x)e^{-x}\,\mathrm{d}x$$

đối với 1 dãy đa thức nhất định $\{P_n\}_{n=1}^\infty$, trong đó $\deg P_n = n$. Nhưng câu trả lời trên cho trường hợp $r = 2$ đưa ra 1 quan hệ trực giao, do đó P_n là Da thức Laguerre cho đến 1 dấu có thể dễ dàng quyết định.

In particular, for the classical derangements:

Theorem 20. *One has*

$$!n = \frac{\Gamma(n+1,-1)}{e} = \int_0^\infty (x-1)^n e^{-x}\,\mathrm{d}x,$$

where $\Gamma(s,x)$ *is the upper incomplete gamma function.*

5.3.6 Computational complexity of derangement computations

It is NP-complete to determine whether a given permutation group (described by a given set of permutations that generate it) contains any derangements.

– Việc xác định xem 1 nhóm hoán vị nhất định (được mô tả bởi 1 tập hợp các hoán vị tạo ra nó) có chứa bất kỳ sự xáo trộn nào hay không là NP-đầy đủ.

5.3.7 Problems: Derangement – Bài tập: Hoán vị sai vị trí

Problem 46 (CSES Problem Set/Christmas party)**.** *There are $n \in \mathbb{N}^\star$ children at a Christmas party, & each of them has brought a gift. The idea is that everybody will get a gift brought by someone else. In how many ways can be gifts be distributed?*

Input. *The only input line has an integer $n \in \mathbb{N}^\star$: the number of children.*

Output. *Print the number of ways module $10^9 + 7$.*

Constraints. $n \in [10^6]$.

Sample.

Christmas_party.inp	Christmas_party.out
4	9

C++:

1. VNTA's C++: Christmas party:

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

const int MOD = 1e9 + 7;

int main() {
```

```
    ios_base::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    int n;
    cin >> n;
    vector<int>dp(n + 1);
    dp[0] = 1; dp[1] = 0; dp[2] = 1;
    for (int i = 3; i <= n; i++) dp[i] = ((i - 1LL) * (dp[i - 1] + dp[i - 2])) % MOD;
    cout << dp[n];
}
```

2. NHH's C++: Christmas party:

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

static const long long MOD = 1000000007;

int main() {
    int n; cin >> n;
    // base cases: !0 = 1,  !1 = 0
    if (n == 0) {
        cout << 1 << "\n";
        return 0;
    }
    if (n == 1) {
        cout << 0 << "\n";
        return 0;
    }

    long long prev2 = 1; // !(0) = 1
    long long prev1 = 0; // !(1) = 0

    for (int i = 2; i <= n; ++i) {
        long long current = ( (long long)(i - 1) * ((prev1 + prev2) % MOD) ) % MOD;
        prev2 = prev1;
        prev1 = current;
    }

    cout << prev1 << "\n";
    return 0;
}
```

3. NLDK's C++: Christmas party:

```
#include<bits/stdc++.h>
#pragma GCC optimize ("O3")
#pragma GCC optimize ("unroll-loops")
#define Sanic_speed ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);
#define Ret return 0;
#define ret return;
#define all(x) x.begin(), x.end()
#define el "\n";
#define elif else if
#define ll long long
#define fi first
#define se second
#define pb push_back
#define pops pop_back
#define cYES cout << "YES" << "\n";
#define cNO cout << "NO" << "\n";
```

```
#define cYes cout << "Yes" << "\n";
#define cNo cout << "No" << "\n";
#define frs(i, a, b) for(int i = a; i < b; ++i)
#define fre(i, a, b) for(int i = a; i <= b; ++i)
#define wh(t) while (t--)
#define SORAI int main()
using namespace std;
typedef unsigned long long ull;

const int M = 1e9 + 7;

void solve() {
    int n;
    cin >> n;
    int dp[1000001];
    dp[0] = dp[1] = 0;
    dp[2] = 1;
    dp[3] = 2;
    fre(i, 4, n) {
        dp[i] = (1LL * (i - 1) * (dp[i - 1] + dp[i - 2]) % M);
    }
    cout << dp[n];
}

SORAI {
    Sanic_speed
    int t = 1; //cin >> t;
    wh(t) {solve();}
}
```

4. PGL's C++: Christmas party:

```
#include <iostream>
#include <algorithm>
#define ll long long
#define mod 1000000007
using namespace std;
ll fac[1000001];
ll pow(ll a, ll b) {
    ll res = 1;
    while (b > 0) {
        if (b & 1) res = (res * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return res;
}
ll inv(ll x) {
    return pow(x, mod - 2);
}
ll ncr(ll a, ll b) {
    return fac[a] * inv(fac[b] * fac[a - b] % mod) % mod;
}
int main() {
    fac[0] = 1;
    for (int i = 1; i <= 1000000; i++) {
        fac[i] = fac[i - 1] * i % mod;
    }
    ll n; cin >> n;
    ll add;
```

```
    ll count = 0;
    for (int i = 1; i <= n; i++) {
        add = ncr(n, i) * fac[n - i] % mod;
        if (i % 2 == 0) add = -add;
        count += add;
    }
    ll ans = ((fac[n] - count) % mod + mod) % mod;
    cout << ans << "\n";
}
```

5. NHT's C++: Christmas party:

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

const int mod = 1e9 + 7;
int n;
ll dp[1000005];

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> n;
    dp[2] = 1;
    for (int i = 3; i <= n; ++i)
        dp[i] = (dp[i - 1] + dp[i - 2]) % mod * (i - 1) % mod;
    cout << dp[n] << '\n';
    return 0;
}
```

6. PDHT's C++: Christmas party:

```
#include <bits/stdc++.h>
#define ll long long
#define MOD 1000000007
#define MAX 1000005
using namespace std;
ll der[MAX];
int main() {
    int n;
    cin >> n;
    der[1] = 0;
    if (n >= 2)
        der[2] = 1;
    for (int i = 3; i <= n; ++i)
            der[i] = (1ll * (i - 1) * (der[i - 1] + der[i - 2])) % MOD;
    cout << der[n] << '\n';
    return 0;
}
```

Phần II

Graph Theory – Lý Thuyết Đồ Thị

Contents

Chương 6

Basic Graph Theory – Lý Thuyết Đồ Thị Cơ Bản

Contents

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. [Sha22]. Shahriar Shahriari. *An Invitation To Combinatorics*. Chap. 10: Graph Theory.

3. [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]

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

Problem 47 ([Sha22], Warm-Up 2.12, p. 52). *Some people* $\{P_i\}_{i=1}^n$ *are on the same social media platform. Some of them are friends with some of the others. For each we have a list of their friends. If 2 people are friends, we say that their distance is* 1*. If 2 people are not friends, but have a friend in common, we say that their distance is* 2*, & so on. We are interested in questions e.g. "What is the distance between* P_i *&* P_j *for some* $i, j \in [n], i \neq j$*?" & "What is the largest distance between any 2 individuals?"*

$$\max_{i,j\in[n],\, i\neq j} \operatorname{dist}(P_i, P_j).$$

Suggest a visual way to present the data that makes answering such questions easier.

– 1 số người $\{P_i\}_{i=1}^n$ *có cùng nền tảng mạng xã hội. 1 số người là bạn của 1 số người khác. Đối với mỗi người, chúng ta có 1 danh sách bạn bè của họ. Nếu 2 người là bạn, chúng ta nói rằng khoảng cách của họ là* 1*. Nếu 2 người không phải là bạn, nhưng có 1 người bạn chung, chúng ta nói rằng khoảng cách của họ là* 2*, & v.v. Chúng ta quan tâm đến các câu hỏi ví dụ như "Khoảng cách giữa* P_i *&* P_j *là bao nhiêu đối với 1 số* $i, j \in [n], i \neq j$*?" & "Khoảng cách lớn nhất giữa bất kỳ 2 cá nhân nào là bao nhiêu?"*

$$\max_{i,j\in[n],\, i\neq j} \operatorname{dist}(P_i, P_j).$$

Đề xuất 1 cách trực quan để trình bày dữ liệu giúp trả lời các câu hỏi như vậy dễ dàng hơn.

Answer. Use a graph $G = (V, E)$ with $V = \{P_i\}_{i=1}^n$, $E = \{e_{ij}; P_i, P_j \text{ are friends}\}$. The distance $\operatorname{dist}(P_i, P_j)$ is then defined as the length of the shortest path between 2 vertices P_i, P_j. The problem of finding the longest distance between 2 vertices can be reformulated as: Find the maximal shortest path(s) between 2 vertices P_i, P_j. □

Graph are such a good way of organizing certain kinds of information & formulating problems. 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 & a set of edges. Intuitively (& visually), an edge is a direct connection between 2 vertices. More formally, an edge is a set consisting of a pair of vertices.

– Đồ thị là 1 cách rất hay để sắp xếp 1 số loại thông tin & xây dựng bài toán. Định nghĩa trong lý thuyết đồ thị rất đa dạng. Để biết danh sách đầy đủ các khái niệm trong lý thuyết đồ thị, hãy xem, ví dụ, Wikipedia/glossary of graph theory. Đồ thị bao gồm 1 tập hợp các đỉnh & 1 tập hợp các cạnh. Theo trực giác (& trực quan), 1 cạnh là 1 kết nối trực tiếp giữa 2 đỉnh. Theo nghĩa chính thức hơn, 1 cạnh là 1 tập hợp bao gồm 1 cặp đỉnh.

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* $V' \subseteq V$ *&* $E' \subseteq E$*.*

Định nghĩa 12 (Đồ thị & đồ thị con). *1* đồ thị đơn *(hoặc chỉ là* đồ thị*)* G *là 1 cặp tập hợp* (V, E) *trong đó* V *là 1 tập hợp không rỗng được gọi là tập hợp* đỉnh *của* G*, &* E *là 1 tập hợp (có thể rỗng) gồm các cặp phần tử phân biệt không có thứ tự của* V*. Tập hợp* E *được gọi là tập hợp* cạnh *của* G*. Nếu số đỉnh của* G *là hữu hạn, thì* G *là 1* đồ thị hữu hạn *(hoặc 1* đồ thị đơn hữu hạn*).*

Nếu $G = (V, E)$ *&* $H = (V', E')$ *đều là đồ thị, thì* H *là 1* đồ thị con *của* G *nếu* $V' \subseteq V$ *&* $E' \subseteq E$*.*

Problem 48. *Find different visualizations of the Peterson graph* $G_{\text{Peterson}} = (V, E)$ *& describe* V, E *in details. Find a subgraph of the Peterson graph.*

– Tìm cách cách vẽ khác nhau của đồ thị Peterson $G_{\text{Peterson}} = (V, E)$ *& mô tả cụ thể* V, E*. Tìm 1 đồ thị con của đồ thị Peterson.*

Solution. See, e.g., [Sha22, Example 2.19, p. 54]. For different visualizations, see, e.g., Wikipedia/Peterson graph. Mathematically, Peterson graph $G = (V, E)$ with 10 vertices & 15 edges:

$$V = [n],\ E = \{\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{1,5\},\{1,6\},\{2,7\},\{3,8\},\{4,9\},\{5,10\},\{6,8\},\{7,9\},\{8,10\},\{6,9\},\{7,10\}\}.$$

To construct a subgraph of this graph, 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. So, e.g., if $V' = \{1,3,6,7,8\}, E' = \{\{1,6\},\{3,8\}\}$, then $H = (V', E')$ is a subgraph of G. The graph H consists of 2 disjoint edges, their vertices, & 1 other isolated vertex. □

Remark 11. *We can draw the picture of a graph $G = (V, E)$ with V, E given in many ways. All that matters is what the vertices are & which vertices are connected to which vertices.*

– Chúng ta có thể vẽ hình ảnh của đồ thị $G = (V, E)$ với V, E được đưa ra theo nhiều cách. Tất cả những gì quan trọng là các đỉnh là gì & đỉnh nào được kết nối với đỉnh nào.

Remark 12. *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.*

– Để xây dựng 1 đồ thị con của 1 đồ thị cho trước, chúng ta lấy 1 số đỉnh & 1 số cạnh, nhưng chúng ta phải đảm bảo rằng chúng ta đã bao gồm tất cả các đỉnh của các cạnh mà chúng ta đã chọn.

Definition 19 (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.

– Nếu đồ thị G không có bất kỳ cạnh kép hoặc vòng lặp nào thì G là đồ thị đơn. Nếu đồ thị G có các cạnh lặp lại nhưng không có vòng lặp, thì G là đồ thị đa. Nếu G có cả cạnh lặp lại & vòng lặp lặp lại, thì G sẽ là đồ thị tổng quát.

Remark 13 ([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 20 (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*.*

Định nghĩa 14. *Cho $G = (V, E)$ là 1 đồ thị tổng quát. Khi đó $|V|$, số đỉnh, được gọi là* cấp *của G. Nếu $\alpha = \{x, y\} \in E$ – tức là α là 1 cạnh nối các đỉnh x, y – thì ta nói rằng x & y là* đỉnh *hoặc* đầu *của α, rằng α* khớp *x & y, rằng x & y là* kề*, & rằng x & α là* sự cố*.*

Definition 21 (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.*

Định nghĩa 15. *Nếu x là 1 đỉnh của đồ thị G, thì số cạnh liên quan đến x được gọi là* bậc *(hoặc* hóa trị*) của x. Trong trường hợp đồ thị tổng quát, 1 vòng lặp tại 1 đỉnh x đóng góp* 2 *vào bậc của x.*

Example 11. *Every vertex of the Peterson graph has degree* 3*.*

$$\deg v = 3,\ \forall v \in V(G_{\text{Peterson}}).$$

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.

Definition 22 (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*, &* 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 16 (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,* $m = |E|$*. 1 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 finite simple graphs)**.** *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 23 ([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 21 ([Val21], Thm. 1.1, p. 4)**.** *Let* $G = (V, E)$ *be a graph with* n *vertices &* m *edges, & let* $V = \{v_1, \ldots, v_n\}$*. Then*

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

Proof. Let $G = (V, E$ be a graph. Every edge $(v_i, v_j) \in E$ contributes 1 to $\text{indeg}(v_i)$ & 1 to $\text{outdeg}(v_i)$. □

A few oft-used terms to graph vocabularies.

Definition 24 (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 25 (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 26 (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*).*

6.1.1 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 bộ, 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 27 (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 17 (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 $k = i+1, \ldots, j$. 1* 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 28 (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. 1 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 29 (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 18 (Đồ thị con). *Cho $G = (V, E)$ là 1 đồ thị, & cho $W \subseteq V$. 1 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 $(W, E \cap W \times W)$.*

6.1.2 Undirected graphs – Đồ thị vô hướng

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 30 (Undirected graph, [Val21], Def. 1.6, 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 19 (Đồ thị vô hướng). *Đồ thị $G = (V, E)$ là* vô hướng *nếu $(v, w) \in E \Rightarrow (w, v) \in E$, $\forall v, w \in V$.*

Undirected graphs are often drawn as a set of points in the plane & a set of line segments, each of which joins 2 (not necessarily different) points. In a drawing of an undirected graph $G = (V, E)$, each vertex $v \in V$ is drawn as a point or a small circle & each pair of counter-parallel edges $(v, w), (w, v) \in E$ is drawn as a line segment between the points or circles corresponding to vertices v & w.

– Đồ thị vô hướng thường được vẽ như 1 tập hợp các điểm trên mặt phẳng & 1 tập hợp các đoạn thẳng, mỗi đoạn thẳng nối 2 điểm (không nhất thiết phải khác nhau). Trong bản vẽ đồ thị vô hướng $G = (V, E)$, mỗi đỉnh $v \in V$ được vẽ như 1 điểm hoặc 1 đường tròn nhỏ & mỗi cặp cạnh đối song song $(v, w), (w, v) \in E$ được vẽ như 1 đoạn thẳng giữa các điểm hoặc đường tròn tương ứng với các đỉnh v & w.

The terminology of directed graphs carries over to undirected graphs, although a few differences deserve mention. 1st of all, since an edge in an undirected graph is understood as a pair of counter-parallel edges in the corresponding bidirected graph, the number of edges coming into a given vertex & the number of edges going out of the vertex coincide & therefore, no distinction is made between the indegree & the outdegree of a vertex. I.e., the *degree* of a vertex in an undirected graph is equal to the indegree & the outdegree of the vertex in the corresponding bidirected graph. For the same reason, the size of an undirected graph is half the size of the corresponding bidirected graph.

– Thuật ngữ đồ thị có hướng được áp dụng cho đồ thị vô hướng, mặc dù có 1 số điểm khác biệt đáng được đề cập. Trước hết, vì 1 cạnh trong đồ thị vô hướng được hiểu là 1 cặp cạnh đối song song trong đồ thị hai hướng tương ứng, nên số cạnh đi vào 1 đỉnh nhất định & số cạnh đi ra khỏi đỉnh trùng nhau & do đó, không có sự phân biệt nào giữa bậc vào & bậc ra của 1 đỉnh. Tức là, *bậc* của 1 đỉnh trong đồ thị vô hướng bằng với bậc vào & bậc ra của đỉnh trong đồ thị hai hướng tương ứng. Vì lý do tương tự, kích thước của đồ thị vô hướng bằng 1 nửa kích thước của đồ thị hai hướng tương ứng.

Definition 31 (Degree, degree sequence, [Val21], Def. 1.7, p. 7)**.** *The* degree *of a vertex v in an undirected graph $G = (V, E)$ is the number of edges in G that are incident with v. The* degree sequence *of G is the sequence of n nonnegative integers obtained by arranging the vertex degrees in nondecreasing order.*

Định nghĩa 20 (Bậc, dãy bậc của đồ thị)**.** Bậc *của đỉnh v trong đồ thị vô hướng $G = (V, E)$ là số cạnh trong G liên quan đến v.* Dãy bậc *của G là dãy n số nguyên không âm thu được bằng cách sắp xếp các bậc đỉnh theo thứ tự không giảm.*

A walk, trail, or path in an undirected graph is a walk, trail, or path, respectively, in the corresponding bidirected graph. Another important graph-theoretical notion is that of connected & disconnected graphs. An undirected graph is *connected* if every pair of its vertices is joined by some walk, & an undirected graph that is not connected is said to be *disconnected*. On the other hand, a directed graph is connected if for all vertices v, w in the graph, there is a walk from v to w, & it is *strongly connected* if there are walks from v to w & from w back to v, for all vertices v, w in the graph.

– 1 walk, trail hoặc path trong 1 đồ thị vô hướng lần lượt là 1 walk, trail hoặc path trong đồ thị hai hướng tương ứng. Một khái niệm quan trọng khác về lý thuyết đồ thị là đồ thị liên thông & không liên thông. Một đồ thị vô hướng là *liên thông* nếu mọi cặp đỉnh của nó được nối bằng 1 walk nào đó, & 1 đồ thị vô hướng không liên thông được gọi là *không liên thông*. Mặt khác, 1 đồ thị có hướng là liên thông nếu với mọi đỉnh v, w trong đồ thị, có 1 walk từ v đến w, & nó là *liên thông mạnh* nếu có các walk từ v đến w & từ w trở lại v, với mọi đỉnh v, w trong đồ thị.

Definition 32 (Connected undirected graph, connected directed graph, strongly connected directed graph, [Val21], Def. 1.8, p. 8)**.** *An undirected graph $G = (V, E)$ is* connected *if for every pair of vertices $v, w \in E$, there is a walk between v, w. A directed graph is* connected *if the underlying undirected graph is connected. Graph G is* strongly connected *if for every pair of vertices $v, w \in E$, there are walks from v to w & from w to v.*

Định nghĩa 21 (Đồ thị vô hướng liên thông, đồ thị có hướng liên thông, đồ thị có hướng liên thông mạnh)**.** *1 đồ thị vô hướng $G = (V, E)$ là* liên thông *nếu với mọi cặp đỉnh $v, w \in E$, có 1 đường đi giữa v, w. Một đồ thị có hướng là* liên thông *nếu đồ thị vô hướng cơ sở là liên thông. Đồ thị G là* liên thông mạnh *nếu với mọi cặp đỉnh $v, w \in E$, có các đường đi từ v đến w & từ w đến v.*

Connectivity is an equivalence relation on the vertex set of a graph, which induces a partition of the graph into subgraphs induced by connected vertices, called *connected components* or just *components*.

– Kết nối/liên thông là 1 quan hệ tương đương trên tập đỉnh của 1 đồ thị, tạo ra sự phân hoạch của đồ thị thành các đồ thị con được tạo ra bởi các đỉnh được kết nối, được gọi là *thành phần được kết nối* hoặc chỉ là *thành phần*.

Definition 33 (Component of undirected graphs, strong component of directed graphs, [Val21], Def. 1.9, p. 8)**.** *A* component *of an undirected graph G is a connected subgraph of G which is not properly contained in any other connected subgraph of G. A* strong component *of a directed graph G is a strongly connected subgraph of G which is not properly contained in any other strongly connected subgraph of G.*

Định nghĩa 22 (Thành phần của đồ thị vô hướng, thành phần mạnh của đồ thị có hướng)**.** *1* thành phần *của 1 đồ thị vô hướng G là 1 đồ thị con liên thông của G không được chứa đúng trong bất kỳ đồ thị con liên thông nào khác của G. Một* thành phần mạnh *của 1 đồ thị có hướng G là 1 đồ thị con liên thông mạnh của G không được chứa đúng trong bất kỳ đồ thị con liên thông mạnh nào khác của G.*

Some common families of undirected graphs are the trees, the complete graphs, the path graphs, the cycle graphs, the wheel graphs, the biparite graphs, & the regular graphs. A *tree* is a connected graph having no cycles.

– 1 số họ phổ biến của đồ thị vô hướng là cây, đồ thị đầy đủ, đồ thị đường đi, đồ thị chu trình, đồ thị bánh xe, đồ thị hai đỉnh, & đồ thị đều. Một *tree* là đồ thị liên thông không có chu trình.

Definition 34 (Undirected tree of undirected graph, [Val21], Def. 1.10, p. 8)**.** *An undirected graph $G = (V, E)$ is said to be an* undirected tree *if it is connected & has no cycles.*

Định nghĩa 23 (Cây vô hướng của đồ thị vô hướng)**.** *1 đồ thị vô hướng $G = (V, E)$ được gọi là* cây vô hướng *nếu nó liên thông & không có chu trình.*

See [Val21, Fig. 1.8: The 1st 14 undirected trees – 14 cây vô hướng đầu tiên].

In a *complete* graph, every pair of distinct vertices is joined by an edge.

– Trong 1 đồ thị *hoàn chỉnh*, mỗi cặp đỉnh riêng biệt được nối với nhau bằng 1 cạnh.

Definition 35 (Complete graph, [Val21], Def. 1.11, p. 9)**.** *An undirected graph $G = (V, E)$ is said to be a* complete graph *if $(v, w) \in E$, $\forall v, w \in V$ with $v \neq w$. The complete graph on $n \in \mathbb{N}^\star$ vertices is denoted by K_n.*

See [Val21, & Fig. 1.9: The 1st 6 complete graphs – 6 đồ thị đầy đủ đầu tiên].

Định nghĩa 24 (Đồ thị đầy đủ)**.** *1 đồ thị vô hướng $G = (V, E)$ được gọi là* đồ thị đầy đủ *nếu $(v, w) \in E$, $\forall v, w \in V$ với $v \neq w$. Đồ thị đầy đủ trên $n \in \mathbb{N}^\star$ đỉnh được ký hiệu là K_n.*

A *path* graph can be drawn s.t. all vertices lie on a straight line.

– 1 đồ thị *path* có thể được vẽ sao cho tất cả các đỉnh đều nằm trên 1 đường thẳng.

Definition 36 (Path graph, [Val21], Def. 1.12, p. 9)**.** *A connected undirected graph $G = (V, E)$ with $|V| = |E| + 1$ is said to be a* path graph *if it can be drawn s.t. all vertices lie on a straight line. The path graph on n vertices is denoted by P_n.*

Định nghĩa 25 (Đồ thị đường đi)**.** *1 đồ thị vô hướng liên thông $G = (V, E)$ với $|V| = |E| + 1$ được gọi là* path graph *nếu nó có thể được vẽ sao cho tất cả các đỉnh nằm trên 1 đường thẳng. Đồ thị đường đi trên n đỉnh được ký hiệu là P_n.*

See [Val21, Fig. 1.10: The 1st 6 path graphs – 6 đồ thị đường đi đầu tiên].

A *cycle* graph can be drawn s.t. all vertices lie on a circle.

– 1 đồ thị *cycle* có thể được vẽ bằng cách tất cả các đỉnh nằm trên 1 đường tròn.

Definition 37 (Cycle graph, [Val21], Def. 1.13, p. 10)**.** *A connected undirected graph $G = (V, E)$ with $|V| = |E|$ is said to be a* cycle graph *if it can be drawn s.t. all vertices lie on a circle. The cycle graph on $n \in \mathbb{N}^\star$ vertices is denoted by C_n.*

Định nghĩa 26 (Đồ thị chu kỳ)**.** *1 đồ thị vô hướng liên thông $G = (V, E)$ với $|V| = |E|$ được gọi là* cycle graph *nếu nó có thể được vẽ sao cho mọi đỉnh đều nằm trên 1 đường tròn. Đồ thị chu trình trên $n \in \mathbb{N}^\star$ đỉnh được ký hiệu là C_n.*

See [Val21, Fig. 1.11: The 1st 6 cycle graphs – 6 đồ thị chu kỳ đầu tiên].

A *wheel* graph has a distinguished (inner) vertex that is connected to all other (outer) vertices, & it can be drawn s.t. all outer vertices lie on a circle centered at the inner vertex.

– Đồ thị *wheel* có 1 đỉnh (bên trong) riêng biệt được kết nối với tất cả các đỉnh (bên ngoài) khác & có thể vẽ nó sao cho tất cả các đỉnh bên ngoài nằm trên 1 đường tròn có tâm tại đỉnh bên trong.

Definition 38 (Wheel graph, [Val21], Def. 1.14, p. 10)**.** *A connected undirected graph $G = (V, E)$ is said to be a* wheel graph *if it can be drawn s.t. all but a single vertex lie on a circle centered at the single vertex, which is connected to all other vertices. The wheel graph with $n \in \mathbb{N}^\star$ outer vertices is denoted by W_{n+1}.*

Định nghĩa 27 (Đồ thị bánh xe)**.** *1 đồ thị vô hướng liên thông $G = (V, E)$ được gọi là* wheel graph *nếu nó có thể được vẽ sao cho tất cả trừ 1 đỉnh nằm trên 1 đường tròn có tâm tại đỉnh duy nhất, được kết nối với tất cả các đỉnh khác. Đồ thị bánh xe có $n \in \mathbb{N}^\star$ đỉnh ngoài được ký hiệu là W_{n+1}.*

See [Val21, Fig. 1.12: The 1st 6 wheel graphs – 6 đồ thị bánh xe đầu tiên].

Another common family of undirected graphs is the *bipartite* graphs. The vertex set of a bipartite graph can be partitioned into 2 subsets in such a way that every edge of the graph joins a vertex of 1 subset with a vertex of the other subset. In a *complete bipartite* graph, every vertex of 1 subset is joined by some edge with every vertex of the other subset.

– 1 họ phổ biến khác của đồ thị vô hướng là đồ thị *bipartite*. Tập đỉnh của đồ thị bipartite có thể được phân chia thành 2 tập con theo cách mà mọi cạnh của đồ thị nối 1 đỉnh của 1 tập con với 1 đỉnh của tập con khác. Trong đồ thị *full bipartite*, mọi đỉnh của 1 tập con được nối bằng 1 cạnh với mọi đỉnh của tập con khác.

Definition 39 (Bipartite graph, complete bipartite graph, [Val21], Def. 1.15, p. 11). *An undirected graph $G = (V, E)$ is said to be a* bipartite graph *if V can be partitioned into 2 disjoint subsets X, Y s.t. $\forall (x, y) \in E$, either $x \in X$ & $y \in Y$, or $x \in Y$ & $y \in X$, & it is said to be a* complete bipartite graph *if, furthermore, $(x, y), (y, x) \in E$, $\forall x \in X, \forall y \in Y$. The complete bipartite graph on $p + q$ vertices, where subset X has p vertices & subset Y has q vertices, is denoted by $K_{p,q}$.*

Định nghĩa 28 (Đồ thị 2 phía, đồ thị 2 phía đầy đủ). *1 đồ thị vô hướng $G = (V, E)$ được gọi là* đồ thị hai phía *nếu V có thể phân hoạch thành 2 tập con rời rạc X, Y s.t. $\forall (x, y) \in E$, hoặc $x \in X$ & $y \in Y$, hoặc $x \in Y$ & $y \in X$, & nó được gọi là* đồ thị hai phía đầy đủ *nếu, hơn nữa, $(x, y), (y, x) \in E$, $\forall x \in X, \forall y \in Y$. Đồ thị hai phía đầy đủ trên $p + q$ đỉnh, trong đó tập con X có p đỉnh & tập con Y có q đỉnh, được ký hiệu là $K_{p,q}$.*

See [Val21, Fig. 1.13: The 1st 12 complete bipartite graphs – 12 đồ thị 2 phía đầy đủ đầu tiên].

Definition 40 (Matching, maximal matching, & maximum matching, perfect matching of bipartite graphs, [Val21], Def. 1.16, p. 12). *Let $G = (X \cup Y, E)$ be a bipartite graph with $E \subseteq X \times Y$ & $X \cap Y = \emptyset$. A* matching *in G is a set of edges $M \subseteq E$ s.t. $x_1 \neq x_2, y_1 \neq y_2$ for all distinct edges $(x_1, y_1), (x_2, y_2) \in M$. A matching M in a bipartite graph G is* maximal *if there is no matching M' in G s.t. $M \subsetneq M'$, & it is* maximum *if there is no matching in G with more edges than M. Further, a matching M in a bipartite graph G is* perfect *if $|M| = \min(|X|, |Y|)$.*

Định nghĩa 29 (Ghép cặp, ghép cặp cực đại, ghép cặp lớn nhất của độ thị 2 phía). *Cho $G = (X \cup Y, E)$ là 1 đồ thị hai phía với $E \subseteq X \times Y$ & $X \cap Y = \emptyset$. Một* ghép *trong G là 1 tập các cạnh $M \subseteq E$ s.t. $x_1 \neq x_2, y_1 \neq y_2$ với mọi cạnh phân biệt $(x_1, y_1), (x_2, y_2) \in M$. Một M ghép trong 1 đồ thị hai phía G là* cực đại *nếu không có M' ghép trong G s.t. $M \subsetneq M'$, & nó là* cực đại *nếu không có phép ghép nào trong G có nhiều cạnh hơn M. Hơn nữa, 1 M ghép trong 1 đồ thị hai phía G là* hoàn hảo *nếu $|M| = \min(|X|, |Y|)$.*

See [Val21, Fig. 1.14: Maximal & maximum matching in a bipartite graph – Ghép cặp cực đại & tối đa trong đồ thị hai phần & Fig. 1.15: Maximum cardinality & maximum weight matching in a bipartite graph with weighted edges – Số lượng tối đa & khớp trọng số tối đa trong đồ thị hai phần với các cạnh có trọng số.].

A maximum matching in a bipartite graph is also called a *maximum cardinality bipartite matching*, in order to distinguish it from a *maximum weight bipartite matching* in a bipartite graph with weighted edges, where the weight of a matching is the total weight of the edges in the matching.

– 1 phép ghép tối đa trong đồ thị hai phần cũng được gọi là *phép ghép hai phần có số lượng cực đại*, để phân biệt với *phép ghép hai phần có trọng số cực đại* trong đồ thị hai phần có các cạnh có trọng số, trong đó trọng số của phép ghép là tổng trọng số của các cạnh trong phép ghép đó.

A natural generalization of bipartite graphs is the *k-partite* graphs. The vertex set of a k-partite graph can be partitioned into k subsets in such a way that every edge of the graph joins a vertex of 1 subset with a vertex of another subset.

– 1 tổng quát hóa tự nhiên của đồ thị hai phần là đồ thị *k-partite*. Tập đỉnh của đồ thị k-partite có thể được phân vùng thành k tập con theo cách mà mỗi cạnh của đồ thị nối 1 đỉnh của 1 tập con với 1 đỉnh của 1 tập con khác.

Definition 41 (k-partite graph, [Val21], Def. 1.17, p. 13). *An undirected graph $G = (V, E$ is said to be a* k-partite graph *if V can be partitioned into $k \geq 2$ subsets $V_1, V_2, \ldots, V_k$ s.t. $\forall (v, w) \in E$ with $v \in V_i, w \in V_j$, it holds that $i \neq j$.*

Định nghĩa 30 (Đồ thị ghép k thành phần). *Đồ thị vô hướng $G = (V, E$ được gọi là đồ thị* k-partite *nếu V có thể phân hoạch thành $k \geq 2$ tập con $V_1, V_2, \ldots, V_k$ s.t. $\forall (v, w) \in E$ với $v \in V_i, w \in V_j$, ta có $i \neq j$.*

Complete graphs & cycle graphs are trivial examples of *regular* graphs. In a regular graph, all vertices have the same degree.

– Đồ thị hoàn chỉnh & đồ thị chu trình là những ví dụ đơn giản về đồ thị *chính quy*. Trong đồ thị chính quy, tất cả các đỉnh đều có cùng bậc.

Definition 42 (Regular graph, [Val21], Def. 1.18, p. 13). *An undirected graph $G = (V, E$ is said to be a* regular graph *if $\deg v = \deg w$, $\forall v, w \in V$.*

Định nghĩa 31 (Đồ thị chính quy). *Đồ thị vô hướng $G = (V, E$ được gọi là* đồ thị chính quy *nếu $\deg v = \deg w$, $\forall v, w \in V$.*

Any regular graph G will have its degree sequence as $[k, k, \ldots, k]$ (n times) with $\deg v = k$, $\forall v \in V(G)$.

– Bất kỳ đồ thị chính quy G nào cũng sẽ có chuỗi bậc của nó là $[k, k, \ldots, k]$ (n lần) với $\deg v = k$, $\forall v \in V(G)$.

Example 12 (Paths or Chains, [Sha22], Example 2.25, p. 55). *Let $n \in \mathbb{N}^\star$ & let $V = [n]$, & let $E = \{\{1, 2\}, \{2, 3\}, \ldots, \{n-1, n\}\}$ be the set of consecutive pairs of elements of $[n]$. A* path *(or a* chain*) of length $n - 1$, denoted often by P_{n-1}, is the graph that has V as its vertices & E as its edges. The graph P_{n-1} has order n (i.e., it has n vertices, $|V(P_{n-1})| = n$) & $n - 1$ edges, i.e., $|E(P_{n-1})| = n - 1$. Note that the subscript in P_k refers to the length of the path, which is the same as the number of edges in the path. This is 1 less than the number of vertices in the graph. (When appropriate, we make a distinction between the length & the size of an object. In graph theory, length refers to the number of edges. While this is very common, it is not universal. There are authors who use the length of a path to mean the number of vertices. In addition, many authors use the notation P_n to mean a path with n vertices (as opposed to n edges).)*

– *Cho $n \in \mathbb{N}^\star$ & cho $V = [n]$, & cho $E = \{\{1,2\},\{2,3\},\dots,\{n-1,n\}\}$ là tập hợp các cặp phần tử liên tiếp của $[n]$. Một* path *(hoặc 1* chain*) có độ dài $n-1$, thường được ký hiệu là P_{n-1}, là đồ thị có V là các đỉnh & E là các cạnh. Đồ thị P_{n-1} có cấp n (tức là nó có n đỉnh, $|V(P_{n-1})| = n$) & $n-1$ cạnh, tức là $|E(P_{n-1})| = n-1$. Lưu ý rằng chỉ số dưới trong P_k đề cập đến độ dài của đường đi, giống như số cạnh trong đường đi. Chỉ số này ít hơn 1 so với số đỉnh trong đồ thị. (Khi thích hợp, chúng ta phân biệt giữa độ dài & kích thước của 1 đối tượng. Trong lý thuyết đồ thị, độ dài đề cập đến số cạnh. Mặc dù điều này rất phổ biến, nhưng không phải là phổ biến. Có những tác giả sử dụng độ dài của đường đi để chỉ số đỉnh. Ngoài ra, nhiều tác giả sử dụng ký hiệu P_n để chỉ đường đi có n đỉnh (khác với n cạnh).)*

Example 13 (Cycles, [Sha22], Example 2.26, p. 56)**.** *Let $n \in \mathbb{N}$, $n \geq 3$. A* cycle of length n *is a connected simple graph with n vertices & n edges s.t. each vertex has degree 2. We denote a cycle of length n by C_n. The graph C_n has n vertices & n edges: $|V(C_n)| = |E(C_n) = n$. (Distinguish the notation C_n for a cycle of length n with the nth Catalan number.)*

– *Cho $n \in \mathbb{N}$, $n \geq 3$. Một* chu trình có độ dài n *là 1 đồ thị đơn liên thông với n đỉnh & n cạnh s.t. mỗi đỉnh có bậc 2. Ta ký hiệu 1 chu trình có độ dài n là C_n. Đồ thị C_n có n đỉnh & n cạnh: $|V(C_n)| = |E(C_n) = n$. (Phân biệt ký hiệu C_n cho 1 chu trình có độ dài n với số Catalan thứ n.)*

Example 14 (Complete graphs, [Sha22], Example 2.27, pp. 56–57)**.** *Let K_n denote the simple graph with $n \in \mathbb{N}^\star$ vertices & all possible edges (no loops or repeated edges). Then K_n is called the* complete graph of order n*. The number of vertices & edges of K_n are $|V(K_n)| = n, |E(K_n)| = C_n^2 = \binom{n}{2} = \frac{n(n-1)}{2}$. Indeed, the 1st vertex is adjacent to $n-1$ other vertices, & this accounts for $n-1$ edges. The 2nd vertex is also adjacent to $n-1$ vertices but we had already counted 1 of them (the edge between the 1st & the 2nd vertex). So we have $n-2$ new edges. Continuing in this way, we get that the number of edges of K_n is $(n-1)+(n-2)+\cdots+1 = \frac{n(n-1)}{2}$. Alternatively, we could have said that every vertex in K_n has degree $n-1$ & there are n vertices. This sounds like $n(n-1)$ edges, but each edge was counted twice – once for each of its 2 vertices – & so $|E(K_n)| = \frac{n(n-1)}{2}$. Alternatively, the number of edges of K_n is the number of pairs of distinct vertices, so there are $C_n^2 = \binom{n}{2} = \frac{n(n-1)}{2}$ edges.*

– *Giả sử K_n biểu thị đồ thị đơn giản với $n \in \mathbb{N}^\star$ đỉnh & tất cả các cạnh có thể (không có vòng lặp hoặc cạnh lặp lại). Khi đó K_n được gọi là* đồ thị đầy đủ cấp n*. Số đỉnh & cạnh của K_n là $|V(K_n)| = n, |E(K_n)| = C_n^2 = \binom{n}{2} = \frac{n(n-1)}{2}$. Thật vậy, đỉnh thứ nhất kề với $n-1$ đỉnh khác, & điều này chiếm $n-1$ cạnh. Đỉnh thứ 2 cũng kề với $n-1$ đỉnh nhưng chúng ta đã đếm 1 trong số chúng (cạnh nằm giữa đỉnh thứ 1 & đỉnh thứ 2). Vì vậy, chúng ta có $n-2$ cạnh mới. Tiếp tục theo cách này, ta có số cạnh của K_n là $(n-1)+(n-2)+\cdots+1 = \frac{n(n-1)}{2}$. Hoặc là, ta có thể nói rằng mọi đỉnh trong K_n có bậc $n-1$ & có n đỉnh. Nghe có vẻ giống như $n(n-1)$ cạnh, nhưng mỗi cạnh được đếm hai lần – 1 lần cho mỗi 2 đỉnh của nó – & do đó $|E(K_n)| = \frac{n(n-1)}{2}$. Hoặc là, số cạnh của K_n là số cặp đỉnh phân biệt, do đó có $C_n^2 = \binom{n}{2} = \frac{n(n-1)}{2}$ cạnh.*

Example 15 (Complete bipartite graphs, [Sha22], Example 2.28, p. 57)**.** *Let $m, n \in \mathbb{N}^\star$, & let L be a set of m vertices & R be a set of n vertices. Connect every vertex in L to every vertex in R. The resulting graph is called the* complete bipartite graph *on m & n vertices, & is denoted by $K_{m,n}$. Note that there are no edges between the vertices in R, & no edges between the vertices in L. The graph $K_{m,n}$ has $m+n$ vertices, &, since each of the m vertices in L is connected to each of the n vertices in R, $K_{m,n}$ has mn edges. More generally, a* bipartite graph *is a graph whose vertices can be partitioned into 2 sets X, Y in such a way that all the edges have 1 end in X & another end in Y. I.e., a bipartite graph is a $K_{m,n}$ after you possibly delete some of the edges.*

– *Cho $m, n \in \mathbb{N}^\star$, & cho L là tập hợp m đỉnh & R là tập hợp n đỉnh. Nối mọi đỉnh trong L với mọi đỉnh trong R. Đồ thị kết quả được gọi là* đồ thị hai phần hoàn chỉnh *trên m & n đỉnh, & được ký hiệu là $K_{m,n}$. Lưu ý rằng không có cạnh nào giữa các đỉnh trong R, & không có cạnh nào giữa các đỉnh trong L. Đồ thị $K_{m,n}$ có $m+n$ đỉnh, &, vì mỗi m đỉnh trong L được nối với mỗi n đỉnh trong R, $K_{m,n}$ có mn cạnh. Tổng quát hơn, 1* bipartite graph *là 1 đồ thị có các đỉnh có thể được phân chia thành 2 tập X, Y theo cách mà tất cả các cạnh đều có 1 đầu trong X & 1 đầu khác trong Y. Tức là, 1 đồ thị bipartite là $K_{m,n}$ sau khi bạn có thể xóa 1 số cạnh.*

In summary:

Theorem 22 (Number of vertices & edges of some special finite simple graphs)**.** *One has:*

(i) (Paths or chains) $|V(P_{n-1})| = n, |E(P_{n-1})| = n-1$, $\forall n \in \mathbb{N}^\star$, $\deg v_1 = \deg v_n = 1$, $\deg v_i = 2$, $\forall i \in \overline{2, n-1}$.

(ii) (Cycles) $|V(C_n)| = |E(C_n)| = n$, $\deg v = 2$, $\forall v \in V(C_n)$, $\forall n \in \mathbb{N}$, $n \geq 3$.

(iii) (Complete graphs) $|V(K_n)| = n, |E(K_n)| = C_n^2 = \binom{n}{2} = \frac{n(n-1)}{2}$, $\deg v = n$, $\forall v \in V(K_n)$.

(iv) (Complete bipartite graphs) $|V(K_{m,n})| = m+n, |E(K_{m,n}) = mn$, $\deg v_i = n$, $\forall i \in [m]$, $\deg v_i = m$, $\forall i \in \overline{m+1, m+n}$, $\forall m, n \in \mathbb{N}^\star$.

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.

– Về mặt kỹ thuật, trong lý thuyết đồ thị tổ hợp – trái ngược với lý thuyết đồ thị tôpô – chúng ta không quan tâm đến cách vẽ đồ thị. Bản vẽ chỉ là 1 biểu diễn giúp chúng ta "nhìn thấy" các đỉnh & các cạnh. Tất cả những gì quan trọng là số lượng các đỉnh & các cạnh kề của chúng. Do đó, chúng ta coi 2 đồ thị là giống nhau nếu sau khi chúng ta dán nhãn lại các đỉnh của 1 trong các đồ thị 1 cách thích hợp, tập hợp các cạnh của 2 đồ thị trở nên giống nhau. 2 đồ thị như vậy được gọi là đẳng cấu.

Definition 43 (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.*

Định nghĩa 32 (Đồng cấu đồ thị)**.** *Cho G, H là đồ thị. Đồ thị G là* đẳng cấu *với đồ thị H nếu tồn tại 1-1, trên ánh xạ $f : V(G) \to V(H)$ s.t. $\{x, y\}$ là 1 cạnh của G khi & chỉ khi $\{f(x), f(y)\}$ là 1 cạnh của H. Một ánh xạ f như vậy được gọi là* đẳng cấu *giữa G & H.*

6.1.3 Labeled graphs – Đồ thị có nhãn

Most applications of graphs in computer science involve *labeled* graphs, where vertices & edges have additional attributes e.g. color or weight. For a labeled graph $G = (V, E)$, the label of a vertex $v \in V$ is denoted by label$[v]$, & the label of an edge $(v, w) = e \in E$ is denoted by label$[e]$ or just by label$[v, w]$.

– Hầu hết các ứng dụng của đồ thị trong khoa học máy tính đều liên quan đến đồ thị *có nhãn*, trong đó các đỉnh & cạnh có các thuộc tính bổ sung, ví dụ như màu sắc hoặc trọng số. Đối với đồ thị có nhãn $G = (V, E)$, nhãn của đỉnh $v \in V$ được ký hiệu là label$[v]$, & nhãn của cạnh $(v, w) = e \in E$ được ký hiệu là label$[e]$ hoặc chỉ là label$[v, w]$.

Example 16 ([Val21], Example 1.18, p. 13)**.** *An undirected graph of the geographical adjacencies between some of the 1st European states to adopt the European currency is shown in* Fig. 1.17: A labeled undirected graph of geographical adjacencies between some European states. *Vertices represent European states & are labeled with a standard abbreviation for the name of the state. There is an edge between 2 vertices if the corresponding states share a border. Edges are labeled with the distance in kilometers between the capital cities of the states they join.*

– 1 đồ thị vô hướng về các vùng lân cận địa lý giữa 1 số quốc gia châu Âu đầu tiên áp dụng đồng tiền chung châu Âu được thể hiện trong Hình 1.17: Đồ thị vô hướng có nhãn về các vùng lân cận địa lý giữa 1 số quốc gia châu Âu. *Các đỉnh biểu diễn các quốc gia châu Âu & được gắn nhãn bằng chữ viết tắt chuẩn cho tên của quốc gia đó. Có 1 cạnh giữa 2 đỉnh nếu các quốc gia tương ứng có chung đường biên giới. Các cạnh được gắn nhãn bằng khoảng cách tính bằng kilômét giữa các thủ đô của các quốc gia mà chúng tham gia.*

6.1.4 Ordered graphs – Đồ thị có thứ tự

An ordered graph is a graph in which the relative order of the adjacent vertices is fixed for each vertex. Ordered graphs arise when a graph is drawn or *embedded* on a certain surface, e.g., in the Euclidean plane.

– Đồ thị có thứ tự là đồ thị trong đó thứ tự tương đối của các đỉnh liền kề được cố định cho mỗi đỉnh. Đồ thị có thứ tự phát sinh khi đồ thị được vẽ hoặc *nhúng* trên 1 bề mặt nhất định, ví dụ, trên mặt phẳng Euclid.

See [Val21, Fig. 1.18: An ordered undirected graph. The relative order of the vertices adjacent with a given vertex reflects the counter-clockwise ordering of the outgoing edges of the vertex in the drawing].

6.1.5 Trees – Cây

The families of undirected graphs introduced above have a directed graph counterpart. In particular, while the notion of tree most often found in discrete mathematics is that of an undirected tree, the notion of the tree which is most useful in computer science is that of the rooted directed tree or just tree. A tree is the particular case of a graph in which there is a distinguished vertex, called the *root* of the tree, s.t. there is a unique walk from the root to any vertex of the tree. The vertices of a tree are called *nodes*.

– Các họ đồ thị vô hướng được giới thiệu ở trên có 1 đồ thị có hướng tương ứng. Đặc biệt, trong khi khái niệm cây thường được tìm thấy nhất trong toán học rời rạc là khái niệm cây vô hướng, thì khái niệm cây hữu ích nhất trong khoa học máy tính là khái niệm cây có hướng có gốc hoặc chỉ là cây. Cây là trường hợp cụ thể của đồ thị trong đó có 1 đỉnh phân biệt, được gọi là *root* của cây, tức là có 1 đường đi duy nhất từ gốc đến bất kỳ đỉnh nào của cây. Các đỉnh của cây được gọi là các *nút*.

Definition 44 (Tree, [Val21], Def. 1.19, p. 15)**.** *A connected graph $G = (V, E)$ is said to be a* tree *if the underlying undirected graph has no cycles & there is a distinguished node $r \in V$, called the* root *of the tree & denoted by* root$[T]$ *s.t. for all nodes $v \in V$, there is a path in G from the root r to node v.*

Định nghĩa 33 (Cây). *1 đồ thị liên thông $G = (V, E)$ được gọi là 1* cây *nếu đồ thị vô hướng cơ sở không có chu trình & có 1 nút phân biệt $r \in V$, được gọi là* gốc *của cây & được ký hiệu là* $\text{gc}[T]$ *.t. đối với mọi nút $v \in V$, có 1 đường đi trong G từ gốc r đến nút v.*

The existence of a unique path in a tree from the root to any other node imposes a hierarchical structure in the tree. The root lies at the top, & nodes can be partitioned in hierarchical levels according to their distance from the root of the tree.

– Sự tồn tại của 1 đường dẫn duy nhất trong 1 cây từ gốc đến bất kỳ nút nào khác áp đặt 1 cấu trúc phân cấp trong cây. Gốc nằm ở trên cùng, & các nút có thể được phân vùng theo các cấp phân cấp theo khoảng cách của chúng từ gốc của cây.

Definition 45 (Depth of node in a tree, [Val21], Def. 1.20, p. 15). *Let $T = (V, E)$ be a tree. The* depth *of node $v \in V$, denoted by* $\text{depth}[v]$*, is the length of the unique path from the root node* $\text{root}[T]$ *to node v, for all nodes $v \in V$. The depth of T is the maximum among the depth of all node $v \in V$.*

Định nghĩa 34 (Độ sâu của nút trong cây). *Cho $T = (V, E)$ là 1 cây.* depth *của nút $v \in V$, được biểu thị bằng* $\text{depth}[v]$*, là độ dài của đường dẫn duy nhất từ nút gốc* $\text{root}[T]$ *đến nút v, đối với mọi nút $v \in V$. Độ sâu của T là độ sâu lớn nhất trong số các độ sâu của mọi nút $v \in V$.*

The hierarchical structure embodied in a tree leads to further distinguishing among the nodes of the tree: *parent* nodes are joined by edges to their *children* nodes, nodes sharing the same parent are called *sibling*, & nodes having no children are called *leaves*.

– Cấu trúc phân cấp thể hiện trong cây dẫn đến sự phân biệt sâu hơn giữa các nút của cây: các nút cha được nối với các nút con bằng các cạnh, các nút chia sẻ cùng 1 nút cha được gọi là anh em, & các nút không có con được gọi là lá.

Definition 46 (Parent, child, children, sibling, binary tree, complete binary tree, [Val21], Def. 1.21, p. 15). *Let $T = (V, E)$ be a tree. Node $v \in V$ is said to be the* parent *of node $w \in V$, denoted by* $\text{parent}[w]$*, if $(v, w) \in E$ &, in such a case, node w is said to be a* child *of node v. The* children *of node $v \in V$ are the set of nodes $W \subseteq V$ s.t. $(v, w) \in E$, $\forall w \in W$. A node having no children is said to be a* leaf *node. Non-root nodes $v, w \in V$ are said to be* sibling *nodes if* $\text{parent}[v] = \text{parent}[w]$*. The number of children of node $v \in V$ is denoted by* $\text{children}[v]$*. The tree is said to be a* binary tree *if* $|\text{children}[v]| \leq 2$ *for every node $v \in V$, & a* complete binary tree *if, furthermore, it has $\lfloor \frac{n}{2} \rfloor$ non-leaf nodes, where $n = |V|$, & at most 1 node $v \in V$ with* $|\text{children}[v]| = 1$*, whose only child is a leaf node.*

Định nghĩa 35 (Cha mẹ, con cái, con cái, anh chị em ruột, cây nhị phân, cây nhị phân hoàn chỉnh). *Cho $T = (V, E)$ là 1 cây. Nút $v \in V$ được gọi là* cha *của nút $w \in V$, ký hiệu là* $\text{cha}[w]$*, nếu $(v, w) \in E$ &, trong trường hợp như vậy, nút w được gọi là* con *của nút v.* con *của nút $v \in V$ là tập các nút $W \subseteq V$ sao cho $(v, w) \in E$, $\forall w \in W$. Một nút không có con nào được gọi là nút* lá*. Các nút không phải gốc $v, w \in V$ được gọi là nút* anh chị em *nếu* $\text{cha}[v] = \text{cha}[w]$*. Số lượng con của nút $v \in V$ được ký hiệu là* $\text{children}[v]$*. Cây được gọi là* cây nhị phân *nếu* $|\text{children}[v]| \leq 2$ *với mọi nút $v \in V$, & 1* cây nhị phân hoàn chỉnh *nếu, ngoài ra, nó có $\lfloor \frac{n}{2} \rfloor$ các nút không phải lá, trong đó $n = |V|$, & nhiều nhất là 1 nút $v \in V$ với* $|\text{children}[v]| = 1$*, mà con duy nhất của nó là 1 nút lá.*

One can prefer to consider the hierarchical structure of a tree the other way around. The leaves constitute the 1st level of height zero, their parents form the 2nd level of height 1, & so on. I.e., nodes can be partitioned in levels according to their height.

– Người ta có thể thích xem xét cấu trúc phân cấp của cây theo cách ngược lại. Các lá tạo thành cấp độ thứ nhất có chiều cao bằng không, các cha mẹ của chúng tạo thành cấp độ thứ hai có chiều cao bằng 1, & v.v. Tức là, các nút có thể được phân vùng thành các cấp độ theo chiều cao của chúng.

Definition 47 (Height of nodes in trees, [Val21], Def. 1.22, p. 16). *Let $T = (V, E)$ be a tree. The* height *of node $v \in V$, denoted by* $\text{height}[v]$*, is the length of a longest path from node v to any node in the subtree of T rooted at node v, for all nodes $v \in V$. The height of T is the maximum among the height of all nodes $v \in V$.*

Định nghĩa 36 (Chiều cao của nút trong cây). *Cho $T = (V, E)$ là 1 cây.* height *của nút $v \in V$, được ký hiệu là* $\text{height}[v]$*, là độ dài của đường đi dài nhất từ nút v đến bất kỳ nút nào trong cây con của T có gốc tại nút v, đối với mọi nút $v \in V$. Chiều cao của T là chiều cao lớn nhất trong số các nút $v \in V$.*

In a tree $T = (V, E)$, $\text{depth}[v] \geq 0$, $\forall v \in V$, & $\text{depth}[v] = 0$ iff $v = \text{root}[T]$. Further, $\text{height}[v] \geq 0$, $\forall v \in V$, & $\text{height}[v] = 0$ iff v is a leaf node.

– Trong 1 cây $T = (V, E)$, $\text{depth}[v] \geq 0$, $\forall v \in V$, & $\text{depth}[v] = 0$ nếu & chỉ nếu $v = \text{root}[T]$. Hơn nữa, $\text{height}[v] \geq 0$, $\forall v \in V$, & $\text{height}[v] = 0$ nếu & chỉ nếu v là 1 nút lá.

Trees are, indeed, the least connected graphs. The number of edges in a tree is 1 less than the number of nodes in the tree.

– Cây thực sự là đồ thị ít kết nối nhất. Số cạnh trong cây ít hơn 1 so với số nút trong cây.

Theorem 23 ([Val21], Thm. 1.2, p. 16). *Let $T = (V, E)$ be a tree. Then $|E| = |V| - 1$.*

Proof. By induction on the number of nodes n in the tree. For $n = 1$, a tree with 1 node has no edges, i.e., $|E| = 0$. Assume, then, that every tree with $n = |V| \geq 1$ nodes has $n - 1$ edges. Let $T = (V, E)$ be a tree with $n + 1$ nodes, & let $v \in V$ by any leaf node of T. The subgraph of T induced by $V \backslash \{v\}$ is a tree with n nodes & has, by induction hypothesis, $n - 1$ edges. Now, since there is only 1 edge $(v, w) \in E$ (namely, with w the parent in T of node v), it follows that the subgraph of T induced by $V \backslash \{v\}$ has 1 edge less than T. Therefore, T has $n + 1$ nodes & n edges.

– Bằng quy nạp theo số nút n trong cây. Với $n = 1$, 1 cây có 1 nút không có cạnh nào, tức là $|E| = 0$. Khi đó, giả sử rằng mọi cây có $n = |V| \geq 1$ nút đều có $n - 1$ cạnh. Cho $T = (V, E)$ là 1 cây có $n + 1$ nút, & cho $v \in V$ bởi bất kỳ nút lá nào của T. Đồ thị con của T sinh ra bởi $V \backslash \{v\}$ là 1 cây có n nút & có, theo giả thuyết quy nạp, $n - 1$ cạnh. Bây giờ, vì chỉ có 1 cạnh $(v, w) \in E$ (cụ thể là, với w là cạnh cha trong T của nút v), nên suy ra đồ thị con của T sinh ra bởi $V \backslash \{v\}$ có 1 cạnh nhỏ hơn T. Do đó, T có $n + 1$ nút & n cạnh. □

As with graphs, there is also a basic relationship between the number of nodes in a tree & the number of children of the nodes of the tree, which will prove to be very useful in analyzing the computational complexity of algorithms on trees.

– Tương tự như đồ thị, cũng có 1 mối quan hệ cơ bản giữa số lượng nút trong 1 cây & số lượng nút con của các nút đó trên cây, điều này sẽ rất hữu ích trong việc phân tích độ phức tạp tính toán của các thuật toán trên cây.

Lemma 3. *Let $T = (V, E)$ be a tree on $n \in \mathbb{N}^\star$ nodes, & let $V = \{v_1, \ldots, v_n\}$. Then,*

$$\sum_{i=1}^{n} \text{children}[v_i] = n - 1.$$

Proof. Let $T = (V, E)$ be a tree. Since every non-root node $w \in V$ is the target of a different edge $(v, w) \in E$, it holds $\sum_{i=1}^{n} \text{children}[v_i] = \sum_{i=1}^{n} \text{outdeg}(v_i) = m$ & then, by Thm. 23, $\sum_{i=1}^{n} \text{children}[v_i] = n - 1$.

– Cho $T = (V, E)$ là 1 cây. Vì mọi nút không phải gốc $w \in V$ là mục tiêu của 1 cạnh khác $(v, w) \in E$, nên nó giữ $\sum_{i=1}^{n} \text{children}[v_i] = \sum_{i=1}^{n} \text{outdeg}(v_i) = m$ & sau đó, theo Thm. **??**, $\sum_{i=1}^{n} \text{children}[v_i] = n - 1$. □

Given that trees are a particular case of graphs, it is natural to consider trees as subgraphs of a given graph &, those trees that span all the vertices of a graph arise in several graph algorithms. A *spanning tree* of a graph is a subgraph that contains all the vertices of the graph & is a tree.

– Vì cây là 1 trường hợp cụ thể của đồ thị, nên việc coi cây là đồ thị con của 1 đồ thị nhất định & là điều tự nhiên, những cây trải dài tất cả các đỉnh của đồ thị sẽ xuất hiện trong 1 số thuật toán đồ thị. Một *cây trải dài* của đồ thị là đồ thị con chứa tất cả các đỉnh của đồ thị & là 1 cây.

Definition 48 (Spanning tree of graph, [Val21], Def. 1.23, p. 17)**.** *Let $G = (V, E)$ be a graph. A subgraph (W, S) of G is a* spanning tree *of graph G if $W = V$ & (W, S) is a tree.*

Định nghĩa 37 (Cây khung của đồ thị)**.** *Cho $G = (V, E)$ là 1 đồ thị. Một đồ thị con (W, S) của G là 1* cây khung *của đồ thị G nếu $W = V$ & (W, S) là 1 cây.*

Example 17 (Generalization of [Val21], Example 1.23, p. 17)**.** *A graph $G = (V, E)$ has $n = |V|$ vertices & $m = |E|$ edges, & thus $\binom{m}{n-1}$ subgraphs $G' = (V, E')$ with n vertices & $n - 1$ edges, $|E'| = n - 1$. But only some of these $\binom{m}{n-1}$ subgraphs are trees, which are exactly the number of spanning trees of G.*

– 1 đồ thị $G = (V, E)$ có $n = |V|$ đỉnh & $m = |E|$ cạnh, & do đó $\binom{m}{n-1}$ đồ thị con $G' = (V, E')$ với n đỉnh & $n - 1$ cạnh, $|E'| = n - 1$. Nhưng chỉ 1 số trong những đồ thị con $\binom{m}{n-1}$ này là cây, chính xác là số cây khung của G.

Not every graph has a spanning tree, though, but every graph has a *spanning forest*, i.e., an ordered set of pairwise-disjoint subgraphs that are trees & which, together, span all the vertices of the graph.

– Tuy nhiên, không phải mọi đồ thị đều có cây khung, nhưng mọi đồ thị đều có 1 *spanning forest*, tức là 1 tập hợp có thứ tự các đồ thị con rời rạc từng cặp là các cây &, cùng nhau trải dài tất cả các đỉnh của đồ thị.

Definition 49 (Spanning forest, [Val21], Def. 1.24, p. 18)**.** *Let $G = (V, E)$ be a graph. A sequence $[(W_1, S_1), \ldots, (W_k, S_k)]$ of $k \in \mathbb{N}^\star$ subgraphs of G is a* spanning forest *of graph G if $W_1 \cup \cdots \cup W_k = V$, $W_i \cap W_j = \emptyset$, $\forall i, j \in [k]$, $i \neq j$, & (W_i, S_i) is a tree, $\forall i \in [k]$.*

6.1.6 Ordered trees – Cây có thứ tự

An ordered tree is a tree in which the relative order of the children is fixed for each node. As a particular case of ordered graphs, ordered trees arise when a tree is drawn or *embedded* in the Euclidean plane.

– Cây có thứ tự là cây trong đó thứ tự tương đối của các con được cố định cho mỗi nút. Là 1 trường hợp cụ thể của đồ thị có thứ tự, cây có thứ tự phát sinh khi 1 cây được vẽ hoặc *nhúng* trong mặt phẳng Euclid.

The relative order of children nodes leads to further distinguishing among the nodes of an ordered tree: children nodes are ordered from the *1st* up to the *last*, non-1st children nodes have a *previous sibling*, & non-last children nodes have a *next sibling*. For sibling nodes v, w, let $v < w$ denote that v precedes w in the ordered tree.

– Thứ tự tương đối của các nút con dẫn đến sự phân biệt sâu hơn giữa các nút của 1 cây có thứ tự: các nút con được sắp xếp từ *1st* đến *last*, các nút con không phải thứ nhất có *anh chị em trước*, & các nút con không phải cuối cùng có *anh chị em tiếp theo*. Đối với các nút anh chị em v, w, hãy để $v < w$ biểu thị rằng v đứng trước w trong cây có thứ tự.

Definition 50 (1st child, last child, next sibling, previous sibling, [Val21], Def. 1.25, p. 19)**.** *Let $T = (V, E)$ be an ordered tree, & let $(v, w) \in E$. Node $w \in V$ is said to be the* 1st child *of node $v \in V$, denoted by* first$[v]$*, if there is no node $z \in V$ s.t. $(v, z) \in E, z < w$. Node $w \in V$ is said to be the* last child *of node $v \in V$, denoted by* last$[v]$*, if there is no node $z \in V$ s.t. $(v, z) \in E$ & $w < z$. Node $z \in V$ is said to be the* next sibling *of node $w \in V$, denoted by* next$[w]$*, if $(v, z) \in E$, $w < z$, & there is no node $x \in V$ s.t. $(v, x) \in E$ & $w < x < z$. Node $w \in V$ is said to be the* previous sibling *of node $z \in V$, denoted by* previous$[z]$*, if* next$[w] = z$.

Định nghĩa 38 (Con thứ nhất, con út, anh chị em tiếp theo, anh chị em trước)**.** *Cho $T = (V, E)$ là 1 cây có thứ tự, & cho $(v, w) \in E$. Nút $w \in V$ được gọi là* 1st child *của nút $v \in V$, ký hiệu là* first$[v]$*, nếu không có nút $z \in V$ s.t. $(v, z) \in E, z < w$. Nút $w \in V$ được gọi là* last child *của nút $v \in V$, ký hiệu là* last$[v]$*, nếu không có nút $z \in V$ s.t. $(v, z) \in E$ & $w < z$. Nút $z \in V$ được gọi là* anh chị em tiếp theo *của nút $w \in V$, ký hiệu là* next$[w]$*, nếu $(v, z) \in E$, $w < z$, & không có nút $x \in V$ s.t. $(v, x) \in E$ & $w < x < z$. Nút $w \in V$ được gọi là* anh chị em trước *của nút $z \in V$, ký hiệu là* previous$[z]$*, nếu* next$[w] = z$.

6.1.7 Problem: Basic graphs – Bài tập: Đồ thị cơ bản

Example 18 ([Sha22], Example 2.30, p. 57)**.** *Consider C_4, a cycle of length* 4*, & label its vertices around the cycle has* $1, 2, 3, 4$*. Now, vertices* $1, 3$ *are not adjacent, but each is adjacent to both* $2, 4$*. As such C_4 is isomorphic to the bipartite graph $K_{2,2}$.*

Problem 49 ([Sha22], P2.2.1, p. 58)**.** *In the early 1970s, the National Football League consisted of* 2 *conferences of* 13 *teams each. The rules of the league specified that during the* 14*-week season, each team would play* 11 *games against teams in its own conference & 3 games against teams in the opposite conference. Prove that this is impossible.*

– Vào đầu những năm 1970, Giải bóng bầu dục quốc gia bao gồm 2 hội nghị, mỗi hội nghị có 13 đội. Các quy tắc của giải đấu quy định rằng trong mùa giải kéo dài 14 tuần, mỗi đội sẽ chơi 11 trận với các đội trong hội nghị của mình & 3 trận với các đội trong hội nghị đối diện. Chứng minh rằng điều này là không thể.

Problem 50 ([Sha22], P2.2.2, p. 58)**.** *Is it possible, in a finite simple graph, for all the degrees of the vertices to be distinct? Either give an example where all the degrees are distinct or prove that in a finite simple graph, we can always find at least* 2 *vertices with the same degree.*

– Trong 1 đồ thị đơn hữu hạn, liệu tất cả các bậc của các đỉnh có thể khác nhau không? Hãy đưa ra 1 ví dụ trong đó tất cả các bậc đều khác nhau hoặc chứng minh rằng trong 1 đồ thị đơn hữu hạn, chúng ta luôn có thể tìm thấy ít nhất 2 đỉnh có cùng bậc.

Problem 51 ([Sha22], P2.2.3, p. 58)**.** *You are given a simple graph with $n \in \mathbb{N}^\star$ vertices. You also know that the graph is bipartite. What is the maximum possible number of edges in this graph?*

– Bạn được cung cấp 1 đồ thị đơn giản với $n \in \mathbb{N}^\star$ đỉnh. Bạn cũng biết rằng đồ thị là đồ thị hai phần. Số cạnh tối đa có thể có trong đồ thị này là bao nhiêu?

Problem 52 ([Sha22], P2.2.4, p. 58)**.** *I have a simple graph with* 47 *vertices. What is the maximum number of edges? What is the maximum number of edges if, in addition, I know that the graph is not connected (a graph is connected if you can go from any vertex to any other vertex by traversing the edges)?*

– Tôi có 1 đồ thị đơn giản với 47 *đỉnh. Số cạnh tối đa là bao nhiêu? Số cạnh tối đa là bao nhiêu nếu, ngoài ra, tôi biết rằng đồ thị không được kết nối (một đồ thị được gọi là kết nối nếu bạn có thể đi từ bất kỳ đỉnh nào đến bất kỳ đỉnh nào khác bằng cách đi qua các cạnh)?*

Problem 53 ([Sha22], P2.2.5, p. 58)**.** *Repeat Problem P.2.2.4 but replace* 47 *with another integer > 1. Do you see a pattern? Let the number of vertices of a simple graph be n. Let M_1 be the maximum number of edges that the graph could have, & let M_2 be the maximum possible number of edges if the graph was not connected. What is $M_1 - M_2$?*

– Lặp lại Bài toán P.2.2.4 nhưng thay 47 *bằng 1 số nguyên khác > 1. Bạn có thấy 1 mô hình không? Giả sử số đỉnh của 1 đồ thị đơn giản là n. Giả sử M_1 là số cạnh tối đa mà đồ thị có thể có, & giả sử M_2 là số cạnh tối đa có thể có nếu đồ thị không được kết nối. $M_1 - M_2$ là gì?*

Problem 54 ([Sha22], P2.2.6, p. 58)**.** *Each of* 9 *users tends* 3 *friend requests on a social media platform. Is it possible that every user receives friend requests from precisely the* 3 *to whom she sent requests? What if the number of users was* 8*, or more generally $n \in \mathbb{N}^\star$?*

– Mỗi 9 *người dùng có xu hướng* 3 *yêu cầu kết bạn trên 1 nền tảng truyền thông xã hội. Có khả năng là mọi người dùng đều nhận được yêu cầu kết bạn từ chính* 3 *người mà họ đã gửi yêu cầu không? Nếu số lượng người dùng là* 8*, hoặc nói chung là $n \in \mathbb{N}^\star$ thì sao?*

Problem 55 ([Sha22], P2.2.7, pp. 58–59). *Let G be a graph whose 7 vertices are Mojdeh, Mehrdokht, Māmak, Marjān, Mehrnāz, Mahshid, & Marzieh. There is an edge between 2 of the vertices if the two are "friends" on the social media platform that they all belong to. The list of these connections is given on the table for Warm-Up 2.12. Make a drawing of a graph G & use it to answer the following questions. (a) Can you find a path of length 6 in G? I.e., can you eliminate some vertices &/or some edges so that the remaining graph is a path of length 6? (If you eliminate a vertex, you have to eliminate all of the edges incident with it, & remember that the length of a path is the number of* edges *on the path.) (b) Can you find a cycle of length 6 in G? Again, this means that you want a cycle of length 6 after possibly eliminating some vertices &/edges. (c) The* distance *between 2 vertices is the length of the shortest path between them. What is the distance between Māmak & Marzieh? (d) Which 2 vertices are the furthest apart? I.e., the distance between which 2 vertices is the largest possible? (e) What is the maximum degree of a vertex in G? What is the minimum degree?*

– Cho G là 1 đồ thị có 7 đỉnh là Mojdeh, Mehrdokht, Māmak, Marjān, Mehrnāz, Mahshid, & Marzieh. Có 1 cạnh giữa 2 đỉnh nếu cả hai là "bạn bè" trên nền tảng mạng xã hội mà tất cả họ đều tham gia. Danh sách các kết nối này được đưa ra trên bảng cho phần Khởi động 2.12. Vẽ 1 đồ thị G & sử dụng nó để trả lời các câu hỏi sau. (a) Bạn có thể tìm thấy 1 đường đi có độ dài 6 trong G không? Nghĩa là, bạn có thể loại bỏ 1 số đỉnh &hoặc 1 số cạnh để đồ thị còn lại là 1 đường đi có độ dài 6 không? (Nếu bạn loại bỏ 1 đỉnh, bạn phải loại bỏ tất cả các cạnh liên hợp với đỉnh đó, & hãy nhớ rằng độ dài của 1 đường đi là số cạnh *trên đường đi.) (b) Bạn có thể tìm thấy 1 chu trình có độ dài 6 trong G không? Một lần nữa, điều này có nghĩa là bạn muốn 1 chu trình có độ dài 6 sau khi có thể loại bỏ 1 số đỉnh &/cạnh. (c) Khoảng cách giữa 2 đỉnh là độ dài của đường đi ngắn nhất giữa chúng. Khoảng cách giữa Māmak & Marzieh là bao nhiêu? (d) 2 đỉnh nào cách xa nhau nhất? Tức là, khoảng cách giữa 2 đỉnh nào là lớn nhất có thể? (e) Bậc tối đa của 1 đỉnh trong G là bao nhiêu? Bậc tối thiểu là bao nhiêu?*

Problem 56 ([Sha22], P2.2.8, p. 59). *A queen wants to build 10 castles connected by ditches. She wants exactly 5 ditches in the form of straight lines, & she wants 4 castles on every ditch. Her advisors suggested a star-shaped configuration with castles at the points of intersections of the line (see* Fig. 2.9: 10 castles at the pairwise intersection of 5 straight ditches. *). She liked the idea of placing castles at the intersections of ditches, but decided to add a new condition. She wants 1 (or maybe even 2) of the castles to be surrounded by ditches, & thus not subject to direct assault from the outside. Can you submit a design? (Mathematical folklore abounds with different versions of this puzzle.)*

– 1 nữ hoàng muốn xây 10 lâu đài được nối với nhau bằng các con mương. Bà muốn chính xác 5 con mương theo hình dạng các đường thẳng, & bà muốn 4 lâu đài trên mỗi con mương. Các cố vấn của bà đề xuất 1 cấu hình hình ngôi sao với các lâu đài tại các giao điểm của đường thẳng (xem Hình 2.9: 10 lâu đài tại giao điểm từng cặp của 5 con mương thẳng. *). Bà thích ý tưởng đặt các lâu đài tại các giao điểm của các con mương, nhưng quyết định thêm 1 điều kiện mới. Bà muốn 1 (hoặc thậm chí có thể là 2) lâu đài được bao quanh bởi các con mương, & do đó không phải chịu sự tấn công trực tiếp từ bên ngoài. Bạn có thể gửi 1 thiết kế không? (Truyện dân gian toán học có rất nhiều phiên bản khác nhau của câu đố này.)*

Problem 57 ([Sha22], P2.2.9, p. 59). *By the time the queen of Problem P2.2.8 considered all the submitted design, an economic slowdown meant that building 10 castles was imprudent. Back to the drawing broad. Since building ditches was a lot cheaper than building castles, the queen requested designs for 7 castles & 7 ditches. 6 of the ditches were to be straight lines & one was to be circular. There would be 3 castles on each ditch & each castle would be at the intersection of 3 ditches. 1 of the castles would be surrounded by ditches & not in immediate danger of a direct attack. Submit a design.*

– Vào thời điểm nữ hoàng của Bài toán P2.2.8 xem xét tất cả các thiết kế đã nộp, 1 sự suy thoái kinh tế có nghĩa là việc xây dựng 10 lâu đài là không khôn ngoan. Quay lại bản vẽ rộng. Vì xây dựng mương rẻ hơn nhiều so với xây dựng lâu đài, nữ hoàng đã yêu cầu thiết kế cho 7 lâu đài & 7 mương. 6 trong số các mương phải là đường thẳng & 1 mương phải là đường tròn. Sẽ có 3 lâu đài trên mỗi mương & mỗi lâu đài sẽ nằm tại giao điểm của 3 mương. 1 trong số các lâu đài sẽ được bao quanh bởi mương & không ở trong nguy cơ bị tấn công trực tiếp ngay lập tức. Nộp 1 thiết kế.

Problem 58 ([Sha22], P2.2.10, p. 59). *A college swim conference has 7 teams. During the season, each team hosts 1 "meet", & the meets are scheduled every other weekend throughout the season. Each meet brings together some of the teams, & those teams participate in all of the events. Is it possible to design a schedule so that each pair of teams ends up in the same meet exactly once? I.e., team A will host 1 meet & will go to a few others & in the course of these should be in the same meet as each of the other teams exactly once. Submit a plan. Could you make sure that each meet has the same number of teams participating?*

– 1 hội nghị bơi lội của trường đại học có 7 đội. Trong suốt mùa giải, mỗi đội tổ chức 1 "cuộc gặp gỡ", & các cuộc gặp gỡ được lên lịch vào mỗi cuối tuần cách tuần trong suốt mùa giải. Mỗi cuộc gặp gỡ quy tụ 1 số đội, & các đội đó tham gia vào tất cả các sự kiện. Có thể thiết kế 1 lịch trình sao cho mỗi cặp đội kết thúc trong cùng 1 cuộc gặp gỡ đúng 1 lần không? Tức là, đội A sẽ tổ chức 1 cuộc gặp gỡ & sẽ tham gia 1 vài cuộc gặp gỡ khác & trong quá trình này phải tham gia cùng 1 cuộc gặp gỡ với mỗi đội khác đúng 1 lần. Nộp 1 kế hoạch. Bạn có thể đảm bảo rằng mỗi cuộc gặp gỡ có cùng số lượng đội tham gia không?

Problem 59 ([Sha22], P2.2.11, p. 60). *The conference of Problem P2.2.10 has expanded & now has 13 teams. What should they do now? Could each team still host a meet once (therefore 13 meets), & each team compete against every other team in exactly 1 meet?*

– Hội nghị của Bài toán P2.2.10 đã mở rộng & hiện có 13 đội. Họ nên làm gì bây giờ? Mỗi đội vẫn có thể tổ chức 1 cuộc gặp gỡ 1 lần (do đó 13 cuộc gặp gỡ), & mỗi đội thi đấu với mọi đội khác trong đúng 1 cuộc gặp gỡ?

Problem 60 ([Sha22], P2.2.12, p. 60). *Consider the complete graph K_4. This graph has 6 edges. Can you color 3 of the edges red & the other 3 blue in such a way that the graph consisting of the red edges is isomorphic to the graph consisting of the blue edges? Either show how or prove why it is not possible.*

– Xét đồ thị hoàn chỉnh K_4. Đồ thị này có 6 cạnh. Bạn có thể tô 3 cạnh màu đỏ & 3 cạnh còn lại màu xanh lam theo cách mà đồ thị bao gồm các cạnh màu đỏ đồng cấu với đồ thị bao gồm các cạnh màu xanh lam không? Hoặc là chỉ ra cách thực hiện hoặc chứng minh tại sao điều đó là không thể.

Problem 61 ([Sha22], P2.2.13, p. 60). *Are the 2 graphs in* Fig. 2.10 *isomorphic? If so, label the vertices for the graphs on the right, in order to show which vertex corresponds to which.*

– 2 đồ thị trong Hình 2.10 *có đẳng cấu không? Nếu có, hãy dán nhãn các đỉnh cho đồ thị bên phải để chỉ ra đỉnh nào tương ứng với đỉnh nào.*

Problem 62 ([Sha22], P2.2.14, p. 60). *Repeat Problem P2.2.12 with K_5. I.e., can you partition the edges of K_5 into 2 sets s.t. the 2 resulting (sub)graphs are isomorphic?*

– Lặp lại Bài toán P2.2.12 với K_5. Tức là, bạn có thể phân hoạch các cạnh của K_5 thành 2 tập hợp sao cho 2 đồ thị (con) kết quả là đẳng cấu không?

Problem 63 ([Sha22], P2.2.15, p. 60). *Which pairs of graphs in* Fig. 2.11 *are isomorphic?*

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?

6.1.8 Graphic sequences

Problem 64 ([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 51 (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 65 ([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 24 ([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 25 (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. 25 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. 25 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 66 ([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 67 ([Sha22], P10.1.2., p. 367). *Is* $7, 7, 6, 5, 4, 4, 4, 3, 2$ *a graphic sequence?*

Problem 68 ([Sha22], P10.1.3., p. 367). *Is there a simple regular graph of degree* 5 *with* 8 *vertices? Why?*

Problem 69 ([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 70 ([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. 25 to construct a graph with this degree sequence. Is the resulting graph isomorphic to the graph of [Sha22, Fig. 10.1]?*

Problem 71 ([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 72 ([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 73 ([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 74 ([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 75 ([Sha22], P10.1.10., p. 367). *Let* $d_1, \ldots, d_n$ *be a non-increasing sequence of* n *nonnegative integers. By Thm. 24, 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 76 ([Sha22], P10.1.11., p. 367). *Find 2 non-isomorphic regular simple graphs of degree* 3 *& order* 6*.*

Problem 77 ([Sha22], P10.1.12., p. 368). *Apply the Havel–Hakimi algorithm of Thm. 25 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 78 ([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 79 ([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 80 ([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 81 ([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 82 ([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 14. *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 83 ([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 84 ([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* $|S| = 1$*, & let* $S = \{a_1\}$*. 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* $|S| = 2$*, & let* $S = \{a_1, a_2\}$ *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 39 ([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 5 (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 24 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 24 & thuật toán Havel–Hakimi 25 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 25.*

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";
    }
}
```

□

6.1.9 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 4 ([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 26 ([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 40 ([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 41 ([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 42 ([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.*

6.2 Basic Data Structures – Cấu Trúc Dữ Liệu Cơ Bản

Resources – Tài nguyên.

1. [Val21]. Gabriel Valiente. *Algorithms on Trees & Graphs With Python Code.* 2e. Sect. 1.2: Basic Data Structures.

Some of the basic data structures needed for the description of algorithms on trees & graphs are illustrated below using a fragment of *pseudocode*. Pseudocode conventions follow modern programming guidelines, e.g. avoiding global side effects (with the only exception of object attributes, which are hidden behind dot-notation) & the unconditional transfer of control by way of goto or gosub statements.

– 1 số cấu trúc dữ liệu cơ bản cần thiết để mô tả các thuật toán trên cây & đồ thị được minh họa bên dưới bằng 1 đoạn mã giả . Các quy ước mã giả tuân theo các hướng dẫn lập trình hiện đại, ví dụ như tránh các tác động phụ toàn cục (trừ các thuộc tính đối tượng, được ẩn sau ký hiệu dấu chấm) & chuyển giao quyền điều khiển vô điều kiện thông qua các câu lệnh goto hoặc gosub.

- Assignment of value a to variable x is denoted by $x \leftarrow a$.

  – Việc gán giá trị a cho biến x được ký hiệu là $x \leftarrow a$.

- Comparison of equality between the values of 2 variables x, y is denoted by $x = y$; comparison of strict inequality is denoted by either $x \neq y, x < y$, or $x > y$, & comparison of non-strict inequality is denoted by either $x \leq y$ or $x \geq y$.

  – So sánh sự bằng nhau giữa các giá trị của 2 biến x, y được ký hiệu là $x = y$; so sánh bất đẳng thức chặt chẽ được ký hiệu là $x \neq y, x < y$ hoặc $x > y$, & so sánh bất đẳng thức không chặt chẽ được ký hiệu là $x \leq y$ hoặc $x \geq y$.

- Logical *true, false* are denoted by `true, false`, resp.

  – Logic *true, false* được biểu thị bằng `true, false`, tương ứng.

- Logical negation, conjunction, & disjunction are denoted by `not, and, or`, resp.

  – Phủ định logic, kết hợp & phân tách được biểu thị bằng `không, và, hoặc`, tương ứng.

- Non-existence is denoted by *nil*.

  – Sự không tồn tại được biểu thị bằng *nil*.

- Mathematical notation is preferred over programming notation. E.g., the cardinality of a set S is denoted by $|S|$, membership of an element x in a set S is denoted by $x \in S$, insertion of an element x into a set S is denoted by $S \leftarrow S \cup \{x\}$, & deletion of an element x from a set S is denoted by $S \leftarrow S \backslash \{x\}$.

  – Ký hiệu toán học được ưa chuộng hơn ký hiệu lập trình. Ví dụ, số lượng phần tử của 1 tập hợp S được ký hiệu là $|S|$, thành viên của 1 phần tử x trong 1 tập hợp S được ký hiệu là $x \in S$, việc chèn 1 phần tử x vào 1 tập hợp S được ký hiệu là $S \leftarrow S \cup \{x\}$, & việc xóa 1 phần tử x khỏi 1 tập hợp S được ký hiệu là $S \leftarrow S \backslash \{x\}$.

- Control structures use the following reserved words: `all, break, do, else, for, if, repeat, return, then, to, until, while`.

  – Cấu trúc điều khiển sử dụng các từ dành riêng sau: `all, break, do, else, for, if, repeat, return, then, to, until, while`.

- Blocks of statements are shown by means of indention.
  – Các khối câu lệnh được hiển thị bằng cách thụt lề.

```
if ... then
    ...
else
    ...

for all ... do
    ...
return ...

while ... do
    ...
    if ... then
        ...

repeat
    ...
until ...
```

The collection of abstract operations on arrays, matrices, lists, stacks, queues, priority queues, sets, & dictionaries is presented next by way of examples.

6.2.1 Arrays – Mảng

An *array* is a 1D array indexed by nonnegative integers. The $[i]$ operation returns the ith element of the array, assuming there is such an element.

– Mảng là mảng 1 chiều được lập chỉ mục bằng số nguyên không âm. Phép toán $[i]$ trả về phần tử thứ i của mảng, giả sử có 1 phần tử như vậy.

```
let A[1..n] be a new array
for i = 1 to n do
    A[i] <- false
```

6.2.2 Matrices – Ma trận

A *matrix* is a 2D array index by nonnegative integers. The $[i, j]$ operation returns the element in the ith row & the jth column of the matrix, assuming there is such an element.

– Ma trận là 1 chỉ số mảng 2 chiều theo số nguyên không âm. Hoạt động $[i, j]$ trả về phần tử trong hàng thứ i & cột thứ j của ma trận, giả sử có 1 phần tử như vậy.

```
let M[1..m][1..n] be a new matrix
for i = 1 to m do
    for j = 1 to n do
        A[i,j] <- 0
```

6.2.3 Lists – Danh sách

A *list* is just a sequence of elements. The `front` operation returns the 1st element & the `back` operation returns the last element in the list, assuming the list is not empty. The `prev` operation returns the element before a given element in the list, assuming the given element is not at the front of the list. The `next` operation returns the element after a given element in the list, assuming the given element is not at the back of the list. The `append` operation inserts an element at the rear of the list. The `concatenate` operation deletes the elements of another list & inserts them at the rear of the list.

– 1 *danh sách* chỉ là 1 chuỗi các phần tử. Hoạt động `front` trả về phần tử thứ nhất & hoạt động `back` trả về phần tử cuối cùng trong danh sách, giả sử danh sách không rỗng. Hoạt động `prev` trả về phần tử trước 1 phần tử nhất định trong danh sách, giả sử phần tử nhất định không nằm ở đầu danh sách. Hoạt động `next` trả về phần tử sau 1 phần tử nhất định trong danh sách, giả sử phần tử nhất định không nằm ở cuối danh sách. Hoạt động `append` chèn 1 phần tử vào phía sau danh sách. Hoạt động `concatenate` xóa các phần tử của 1 danh sách khác & chèn chúng vào phía sau danh sách.

```
let L be an empty list
append x to L // L.append(x)
let L' be an empty list
append x to L'
concatenate L' to L // L.concatenate(L')
let x be the element at the front of L // x <- L.front()
while x != nil do
    output x
    x <- L.next(x)
let x be the element at the back of L // x <- L.back()
while x != nil do
    output x
    x <- L.prev(x)
```

6.2.4 Stacks – Ngăn xếp

A *stack* is a sequence of elements that are inserted & deleted at the same end (the top) of the sequence. The `top` operation returns the top element in the back, assuming the stack is not empty. The `pop` operation deletes & returns the top element in the stack, also assuming the stack is not empty. The `push` operation inserts an element at the top of the stack.

– 1 ngăn xếp là 1 chuỗi các phần tử được chèn & xóa ở cùng 1 đầu (phía trên cùng) của chuỗi. Hoạt động `top` trả về phần tử trên cùng ở phía sau, giả sử ngăn xếp không rỗng. Hoạt động `pop` xóa & trả về phần tử trên cùng trong ngăn xếp, cũng giả sử ngăn xếp không rỗng. Hoạt động `push` chèn 1 phần tử ở phía trên cùng của ngăn xếp.

```
let S be an empty stack
push x onto S // S.push(x)
let x be the element at the top of S // x <- S.top()
output x
while S is not empty do // not S.empty()
    pop from S the top element x // x <- S.pop()
    output x
```

6.2.5 Queues – Hàng đợi

A *queue* is a sequence of elements that are inserted at 1 end (the rear) & deleted at the other end (the front) of the sequence. The `front` operation returns the front element in the queue, assuming the queue is not empty. The `dequeue` operation deletes & returns the front element in the queue, assuming the queue is not empty. The `enqueue` operation inserts an element at the rear end of the queue.

– 1 hàng đợi là 1 chuỗi các phần tử được chèn vào đầu 1 (phía sau) & xóa ở đầu kia (phía trước) của chuỗi. Hoạt động `front` trả về phần tử phía trước trong hàng đợi, giả sử hàng đợi không rỗng. Hoạt động `dequeue` xóa & trả về phần tử phía trước trong hàng đợi, giả sử hàng đợi không rỗng. Hoạt động `enqueue` chèn 1 phần tử vào đầu cuối của hàng đợi.

```
let Q be an empty queue
enqueue x into Q // Q.enqueue(x)
let x be the element at the front of Q // x <- Q.front()
output x
repeat
    dequeue from Q the front element x // x <- Q.dequeue()
    output x
until Q is empty // Q.empty()
```

6.2.6 Priority queues – Hàng đợi ưu tiên

A *priority queue* is a queue of elements with both an information & a priority associated with each element, where there is a linear order defined on the priorities. The `front` operation returns an element with the minimum priority, assuming the priority queue is not empty. The `dequeue` operation deletes & returns an element with the minimum priority in the queue, assuming the priority queue is not empty. The `enqueue` operation inserts an element with a given priority in the priority queue.

– Hàng đợi ưu tiên là hàng đợi các phần tử có cả thông tin & mức ưu tiên liên kết với mỗi phần tử, trong đó có thứ tự tuyến tính được xác định trên các mức ưu tiên. Hoạt động `front` trả về 1 phần tử có mức ưu tiên tối thiểu, giả sử hàng đợi ưu tiên không trống. Hoạt động `dequeue` xóa & trả về 1 phần tử có mức ưu tiên tối thiểu trong hàng đợi, giả sử hàng đợi ưu tiên không trống. Hoạt động `enqueue` chèn 1 phần tử có mức ưu tiên nhất định vào hàng đợi ưu tiên.

```
let Q be an empty priority queue
enqueue (x, y) into Q // Q.enqueue(x, y)
let (x, y) be an element x
    with the minimum priority y in Q // (x, y) <- Q.front()
output (x, y)
repeat
    dequeue from Q an element x
        with the minimum priority y // (x, y) <- Q.dequeue()
    output (x, y)
until Q is empty // Q.empty()
```

6.2.7 Sets – Tập hợp

A *set* is just a set of elements. The `insert` operation inserts an element in the set. The `delete` operation deletes an element from the set. The `member` operation returns true if an element belongs to the set & false otherwise.

– 1 tập hợp chỉ là 1 tập hợp các phần tử. Thao tác `insert` chèn 1 phần tử vào tập hợp. Thao tác `delete` xóa 1 phần tử khỏi tập hợp. Thao tác `member` trả về true nếu 1 phần tử thuộc về tập hợp & false nếu không.

```
let S be an empty set
S = S cup {x} // S.insert(x)
for all x in S do
    output x
    delete x from S // S.delete(x)
```

6.2.8 Dictionaries – Từ điển

A *dictionary* is an associative container, consisting of a set of elements with both an information & a unique key associated with each element, where there is a linear order defined on the keys & the information associated with an element is retrieved on the basis of its key. The `member` operation returns true if there is an element with a given key in the dictionary, & false otherwise. The `lookup` operation returns the element with a given key in the dictionary, or `nil` if there is no such element. The `insert` operation inserts & returns an element with a given key & given information in the dictionary, replacing the element (if any) with the given key. The `delete` operation deletes the element with a given key from the dictionary if there is such an element.

– Từ điển là 1 bộ chứa liên kết, bao gồm 1 tập hợp các phần tử có cả thông tin & khóa duy nhất được liên kết với mỗi phần tử, trong đó có 1 thứ tự tuyến tính được xác định trên các khóa & thông tin được liên kết với 1 phần tử được truy xuất trên cơ sở khóa của phần tử đó. Hoạt động `member` trả về true nếu có 1 phần tử có khóa đã cho trong từ điển, & false nếu không. Hoạt động `lookup` trả về phần tử có khóa đã cho trong từ điển hoặc `nil` nếu không có phần tử nào như vậy. Hoạt động `insert` chèn & trả về 1 phần tử có khóa đã cho & thông tin đã cho trong từ điển, thay thế phần tử (nếu có) bằng khóa đã cho. Hoạt động `delete` xóa phần tử có khóa đã cho khỏi từ điển nếu có phần tử như vậy.

```
Let D an empty dictionary
D[x] <- x // D.insert(x, y)
for all x in D do // D.member(x)
    y <- D[x]
    output (x, y)
    delete x from D // D.delete(x)
```

6.3 Representation of Trees & Graphs – Biểu Diễn Cây & Đồ Thị

Resources – Tài nguyên.

1. [Val21]. Gabriel Valiente. *Algorithms on Trees & Graphs With Python Code*. 2e. Sect. 1.3: Representation of Trees & Graphs.

There are several different ways in which graphs &, in particular, trees can be represented in a computer, & the choice of a data structure often depends on the efficiency with which the operations that access & update the data need to be supported. As a matter of fact, there is often a tradeoff between the space complexity of a data structure & the time complexity of the operations.

– Có 1 số cách khác nhau để biểu diễn đồ thị &, đặc biệt là cây trong máy tính, & việc lựa chọn cấu trúc dữ liệu thường phụ thuộc vào hiệu quả mà các hoạt động truy cập & cập nhật dữ liệu cần được hỗ trợ. Trên thực tế, thường có sự đánh đổi giữa độ phức tạp về không gian của 1 cấu trúc dữ liệu & độ phức tạp về thời gian của các hoạt động.

6.3.1 Representation of graphs – Biểu diễn đồ thị

A graph representation consists of a collection of abstract operations, a concrete representation of graphs by appropriate data structures, & an implementation of the abstract operations using the concrete data structures.

– Biểu diễn đồ thị bao gồm 1 tập hợp các phép toán trừu tượng, 1 biểu diễn cụ thể của đồ thị bằng các cấu trúc dữ liệu thích hợp và 1 triển khai các phép toán trừu tượng bằng cách sử dụng các cấu trúc dữ liệu cụ thể.

The choice of abstract operations to be included in a graph representation depends on the particular application area. E.g., the representation of graphs in the LEDA library of efficient data structures & algorithms supports about 120 different operations, roughly half of which address the needs of computational geometry algorithms, & the representation of graphs in the BGL library of graph algorithms supports about 50 different operations. The following collection of 32 abstract operations, though, covers the needs of most of the algorithms on graphs presented in [Val21].

– Lựa chọn các phép toán trừu tượng để đưa vào biểu diễn đồ thị phụ thuộc vào lĩnh vực ứng dụng cụ thể. Ví dụ, biểu diễn đồ thị trong thư viện LEDA về các cấu trúc dữ liệu hiệu quả & thuật toán hỗ trợ khoảng 120 phép toán khác nhau, trong đó khoảng 1 nửa đáp ứng nhu cầu của các thuật toán hình học tính toán, & biểu diễn đồ thị trong thư viện BGL về các thuật toán đồ thị hỗ trợ khoảng 50 phép toán khác nhau. Tuy nhiên, bộ sưu tập 32 phép toán trừu tượng sau đây đáp ứng nhu cầu của hầu hết các thuật toán trên đồ thị được trình bày trong [Val21].

1. `G.vertices(), G.edges()` give, resp., a list of the vertices & a list of the edges of graph G in the order fixed by the representation of G.

   – `G.vertices(), G.edges()` cung cấp danh sách các đỉnh & danh sách các cạnh của đồ thị G theo thứ tự được xác định bởi biểu diễn của G.

2. `G.incoming(v), G.outgoing(v)` give a list of the edges of graph G coming into & going out of vertex v, resp., in the order fixed by the representation of G.

   – `G.incoming(v), G.outgoing(v)` cung cấp danh sách các cạnh của đồ thị G đi vào & đi ra khỏi đỉnh v, tương ứng, theo thứ tự được xác định bởi biểu diễn của G.

3. `G.number_of_vertices(), G.number_of_edges()` give, resp., the order n & the size m of graph G.

   – `G.number_of_vertices(), G.number_of_edges()` tương ứng đưa ra bậc n & kích thước m của đồ thị G.

4. `G.indeg(v), G.outdeg(v)` give, resp., the number of edges coming into & going out of vertex v in graph G.

   – `G.indeg(v), G.outdeg(v)` tương ứng cho số cạnh đi vào & đi ra khỏi đỉnh v trong đồ thị G.

5. `G.adjacent(v, w)` is true if there is an edge in graph G going out of vertex v & coming into vertex w, & false otherwise.

   – `G.adjacent(v, w)` là đúng nếu có 1 cạnh trong đồ thị G đi ra khỏi đỉnh v & đi vào đỉnh w, & false nếu không.

6. `G.source(e), G.target(e)` give, resp., the source & the target vertex of edge e in graph G.

   – `G.source(e), G.target(e)` tương ứng cung cấp đỉnh nguồn & đỉnh đích của cạnh e trong đồ thị G.

7. `G.opposite(v, e)` gives `G.target(e)` if vertex v is the source of edge e in graph G, & `G.source(e)` otherwise.

   – `G.opposite(v, e)` cung cấp `G.target(e)` nếu đỉnh v là nguồn của cạnh e trong đồ thị G, & `G.source(e)` nếu không.

8. `G.first_vertex(), G.last_vertex()` give, resp., the 1st & the last edge in the representation of graph G. Further, `G.pred_edge(e), G.succ_edge(e)` give, resp., the predecessor & the successor of edge e in the representation of graph G. These operations support iteration over the edges of the graph.

   – `G.first_vertex(), G.last_vertex()` cung cấp, tương ứng, cạnh thứ nhất & cạnh cuối cùng trong biểu diễn của đồ thị G. Hơn nữa, `G.pred_edge(e), G.succ_edge(e)` cung cấp, tương ứng, cạnh tiền nhiệm & cạnh kế nhiệm của cạnh e trong biểu diễn của đồ thị G. Các hoạt động này hỗ trợ lặp lại trên các cạnh của đồ thị.

9. `G.first_in_edge(v), G.last_in_edge(v)` give, resp., the 1st & the last edge in the representation of graph G coming into vertex v. Further, `G.in_pred(e), G.in_succ(e)` give, resp., the previous & the next edge after e in the representation of graph G coming into vertex `G.target(e)`. These operations support iteration over the vertices of the graph adjacent with a given vertex.

   – `G.first_in_edge(v), G.last_in_edge(v)` cung cấp, tương ứng, cạnh thứ nhất & cạnh cuối cùng trong biểu diễn của đồ thị G đi vào đỉnh v. Hơn nữa, `G.in_pred(e), G.in_succ(e)` cung cấp, tương ứng, cạnh trước & cạnh tiếp theo sau e trong biểu diễn của đồ thị G đi vào đỉnh `G.target(e)`. Các phép toán này hỗ trợ lặp lại trên các đỉnh của đồ thị liền kề với 1 đỉnh đã cho.

10. `G.first_adj_edge(v), G.last_adj_edge(v)` give, resp., the 1st & the last edge in the representation of graph G going out of vertex v. Further, `G.adj_pred(e), G.adj_succ(e)` give, resp., the previous & the next edge after e in the representation of graph G going out of vertex `G.source(e)`. These operations also support iteration over the vertices of the graph adjacent with a given vertex.

    – `G.first_adj_edge(v), G.last_adj_edge(v)` cung cấp, tương ứng, cạnh thứ nhất & cạnh cuối cùng trong biểu diễn của đồ thị G đi ra khỏi đỉnh v. Hơn nữa, `G.adj_pred(e), G.adj_succ(e)` cung cấp, tương ứng, cạnh trước & cạnh tiếp theo sau e trong biểu diễn của đồ thị G đi ra khỏi đỉnh `G.source(e)`. Các phép toán này cũng hỗ trợ lặp lại trên các đỉnh của đồ thị liền kề với 1 đỉnh đã cho.

11. `G.new_vertex()` inserts a new vertex in graph G, & `G.new_edge(v, w)` inserts a new edge in graph G going out of vertex v & coming into vertex w. Further, `G.del_vertex(v)` deletes vertex v from graph G, together with all those edges going out of or coming into vertex v, & `G.del_edge(e)` deletes edge e from graph G. These operations support dynamic graphs.

    – `G.new_vertex()` chèn 1 đỉnh mới vào đồ thị G, & `G.new_edge(v, w)` chèn 1 cạnh mới vào đồ thị G đi ra khỏi đỉnh v & đi vào đỉnh w. Hơn nữa, `G.del_vertex(v)` xóa đỉnh v khỏi đồ thị G, cùng với tất cả các cạnh đi ra khỏi hoặc đi vào đỉnh v, & `G.del_edge(e)` xóa cạnh e khỏi đồ thị G. Các phép toán này hỗ trợ đồ thị động.

Some of these operations are actually defined in terms of a smaller set of 10 abstract operations on graphs::

`G.vertices(), G.edges(), G.incoming(v), G.outgoing(v), G.source(e), G.target(e), G.new_vertex(), G.new_edge(v, w`

using the abstract operations on basic data structures presented as follows:

1. `G_number_of_vertices()` is given by `G.vertices().size()`.
2. `G_number_of_edges()` is given by `G.edges().size()`.
3. `G.indeg(v)` is given by `G.incoming(v).size()`.
4. `G.outdeg(v)` is given by `G.outcoming(v).size()`.
5. `G.adjacent(v, w)` is true if there is an edge $e \in$ `G.outing`(v) s.t. G.`target`$(e) = w$, & it is false otherwise.
6. `G.opposite(v, e)` is given by `G.target(e)` if `G.source(e) = v`, & it is given by `G.source(e)` otherwise, i.e., if `G.target(e) = v`.
7. `G.first_vertex()` is given by `G.vertices().front()`.
8. `G.last_vertex()` is given by `G.vertices().back()`.
9. `G.pred_vertex(v)` is given by `G.vertices().prev(v)`.
10. `G.succ_vertex(v)` is given by `G.vertices().next(v)`.
11. `G.first_edge()` is given by `G.edges().front()`.
12. `G.last_edge()` is given by `G.edges().back()`.
13. `G.pred_edge(e)` is given by `G.edges().prev(e)`.
14. `G.succ_edge(e)` is given by `G.edges().next(e)`.
15. `G.first_in_edge(v)` is given by `G.incoming(v).front()`.
16. `G.last_in_edge(v)` is given by `G.incoming(v).back()`.
17. `G.in_pred(e)` is given by `G.incoming(v).prev(e)`.
18. `G.in_succ(e)` is given by `G.incoming(v).next(e)`.
19. `G.first_adj_edge(v)` is given by `G.outgoing(v).front()`.
20. `G.last_adj_edge(v)` is given by `G.outgoing(v).back()`.
21. `G.adj_pred(e)` is given by `G.outgoing(v).prev(e)`.
22. `G.adj_succ(e)` is given by `G.outgoing(v).next(e)`.

Together with the abstract operations on the underlying basic data structures, these operations support iteration over the vertices & edges of a graph. E.g., in the following procedure, variable v is assigned each of the vertices of graph G in turn,

```
for all vertices v of G do // v in G.vertices()
    ...
```

& in the following procedure, all those vertices of graph G which are adjacent with vertex v are assigned in turn to variable w.

```
for all vertices w adjacent with vertex v in G do // e in G.outgoing(v)
    ... // w = G.target(e)
```

Together, they provide for a simple traversal of a graph, in which vertices & edges are visited in the order fixed by the representation of the graph.

```
for all vertices v of G do // v in G.vertices()
    for all vertices w adjacent with vertex v in G do // e in G.outgoing(v)
        ... // w = G.target(e)
```

– Cùng với các phép toán trừu tượng trên các cấu trúc dữ liệu cơ bản bên dưới, các phép toán này hỗ trợ phép lặp qua các đỉnh & cạnh của đồ thị. Ví dụ, trong quy trình sau, biến v được gán lần lượt cho từng đỉnh của đồ thị G,

```
for all vertices v of G do // v in G.vertices()
    ...
```

& trong quy trình sau, tất cả các đỉnh của đồ thị G kề với đỉnh v thì lần lượt được gán cho biến w.

```
for all vertices w adjacent with vertex v in G do // e in G.outgoing(v)
    ... // w = G.target(e)
```

Cùng nhau, chúng cung cấp 1 phép duyệt đơn giản trên đồ thị, trong đó các đỉnh & cạnh được truy cập theo thứ tự cố định theo biểu diễn của đồ thị.

```
for all vertices v of G do // v in G.vertices()
    for all vertices w adjacent with vertex v in G do // e in G.outgoing(v)
        ... // w = G.target(e)
```

6.3.1.1 Adjacency matrix – Ma trận kề

The data structures most often used for representing graphs are adjacency matrices & adjacency lists. The adjacency matrix representation of a graph is a Boolean matrix, with an entry for each ordered pair of vertices in the graph, where the entry corresponding to vertices v, w has the value `true` if there is an edge in the graph going out of vertex v & coming into vertex w, & it has the value `false` otherwise.

– Cấu trúc dữ liệu thường được sử dụng nhất để biểu diễn đồ thị là ma trận kề & danh sách kề. Biểu diễn ma trận kề của đồ thị là ma trận Boolean, với 1 mục nhập cho mỗi cặp đỉnh có thứ tự trong đồ thị, trong đó mục nhập tương ứng với các đỉnh v, w có giá trị `true` nếu có 1 cạnh trong đồ thị đi ra khỏi đỉnh v & đi vào đỉnh w, & có giá trị `false` nếu không.

Definition 52 (Adjacency matrix, [Val21], Def. 1.26, p. 26)**.** *Let $G = (V, E)$ be a graph with $n \in \mathbb{N}^\star$ vertices. The* adjacency matrix *representation of G is an $n \times n$ Boolean matrix with an entry for each ordered pair of vertices of G, where the entry corresponding to vertices v, w is equal to* `true` *if $(v, w) \in E$ & is equal to* `false` *otherwise, for all vertices $v, w \in V$.*

Định nghĩa 43 (Ma trận kề)**.** *Cho $G = (V, E)$ là 1 đồ thị có $n \in \mathbb{N}^\star$ đỉnh. Biểu diễn* adjacency matrix *của G là 1 ma trận Boolean $n \times n$ với 1 mục nhập cho mỗi cặp đỉnh có thứ tự của G, trong đó mục nhập tương ứng với các đỉnh v, w bằng* `true` *nếu $(v, w) \in E$ & bằng* `false` *nếu không, đối với mọi đỉnh $v, w \in V$.*

With most data structures used for representing graphs, there is an order on the vertices fixed by the representation. In the case of the adjacency matrix representation A of a graph $G = (V, E)$, vertex v precedes vertex w iff the row of A corresponding to v lies above the row of A corresponding to w or, in an equivalent formulation, the column of A corresponding to v lies to the left of the column of A corresponding to w, for all vertices $v, w \in V$.

– Với hầu hết các cấu trúc dữ liệu được sử dụng để biểu diễn đồ thị, có 1 thứ tự trên các đỉnh được cố định bởi biểu diễn. Trong trường hợp biểu diễn ma trận kề A của đồ thị $G = (V, E)$, đỉnh v đứng trước đỉnh w nếu/nếu hàng của A tương ứng với v nằm phía trên hàng của A tương ứng với w hoặc, theo 1 công thức tương đương, cột của A tương ứng với v nằm bên trái cột A tương ứng với w, đối với mọi đỉnh $v, w \in V$.

Since the edges are implicit in the adjacency matrix representation of a graph, those operations having an edge as argument or giving an edge as result cannot be implemented using an adjacency matrix representation. These operations are: `G.edges()`, `G.incoming(v)`, `G.outgoing(v)`, `G.source(e)`, `G.target(e)`, & `G.del_edge(e)`. Furthermore, `G.new_vertex()`, `G.del_vertex()`

cannot be implemented unless the adjacency matrix can be dynamically resized, & `G.new_edge(v, w)` can be implemented by setting to `true` the entry of A at the row corresponding to vertex v & the column corresponding to vertex w, & takes $O(1)$ time, but the operation cannot give the new edge as result.

– Vì các cạnh được ngầm định trong biểu diễn ma trận kề của đồ thị, nên các phép toán có cạnh làm đối số hoặc đưa ra cạnh làm kết quả không thể được triển khai bằng cách sử dụng biểu diễn ma trận kề. Các phép toán này là: `G.edges(), G.incoming(v), G.outgoing(v), G.source(e), G.target(e)`, & `G.del_edge(e)`. Hơn nữa, `G.new_vertex(), G.del_vertex()` không thể được triển khai trừ khi ma trận kề có thể được thay đổi kích thước động, & `G.new_edge(v, w)` có thể được triển khai bằng cách đặt thành `true` mục nhập của A tại hàng tương ứng với đỉnh v & cột tương ứng với đỉnh w, & mất $O(1)$ thời gian, nhưng phép toán không thể đưa ra cạnh mới làm kết quả.

Let $G = (V, E)$ be a graph with $n \in \mathbb{N}^\star$ vertices, assume the vertices are numbered $1, 2, \ldots, n$ in some arbitrary manner & their number is stored in the attribute *index*, & let A be the adjacency matrix representation of G. The only remaining operation, `G.vertices()`, is just the sequence of all vertices $v \in V$ ordered by the *index* attribute.

– Giả sử $G = (V, E)$ là 1 đồ thị có $n \in \mathbb{N}^\star$ đỉnh, giả sử các đỉnh được đánh số $1, 2, \ldots, n$ theo 1 cách tùy ý & số của chúng được lưu trữ trong thuộc tính *index*, & giả sử A là biểu diễn ma trận kề của G. Phép toán duy nhất còn lại, `G.vertices()`, chỉ là chuỗi của tất cả các đỉnh $v \in V$ được sắp xếp theo thuộc tính *index*.

The adjacency matrix representation of a graph $G = (V, E)$ with $n \in \mathbb{N}^\star$ vertices takes $\Theta(n^2)$ space. Therefore, no graph algorithm can be implemented using an adjacency matrix representation to run in $O(n)$ time. The main advantage of the adjacency matrix representation is the support of the adjacency test in $O(1)$ time.

– Biểu diễn ma trận kề của đồ thị $G = (V, E)$ với $n \in \mathbb{N}^\star$ đỉnh chiếm không gian $\Theta(n^2)$. Do đó, không có thuật toán đồ thị nào có thể được triển khai bằng cách sử dụng biểu diễn ma trận kề để chạy trong thời gian $O(n)$. Ưu điểm chính của biểu diễn ma trận kề là hỗ trợ kiểm tra kề trong thời gian $O(1)$.

6.3.1.2 Adjacency list – Danh sách kề

The adjacency list representation of a graph, on the other hand, is an array of lists, 1 for each vertex in the graph, where the list corresponding to vertex v contains the target vertices of the edges coming out of vertex v.

– Mặt khác, biểu diễn danh sách kề của đồ thị là 1 mảng các danh sách, 1 cho mỗi đỉnh trong đồ thị, trong đó danh sách tương ứng với đỉnh v chứa các đỉnh mục tiêu của các cạnh đi ra từ đỉnh v.

Definition 53 (Adjacency list, [Val21], Def. 1.27, p. 28)**.** *Let $G = (V, E)$ be a graph with $n \in \mathbb{N}^\star$ vertices & $m \in \mathbb{N}^\star$ edges. The* adjacency list *representation of G consists of a list of n elements (the vertices of the graph) & a list of n lists with a total of m elements (the target vertices for the edges of the graph). The list corresponding to vertex v contains all vertices $w \in V$ with $(v, w) \in E$, for all vertices $v \in V$.*

Định nghĩa 44 (Danh sách kề)**.** *Cho $G = (V, E)$ là 1 đồ thị có $n \in \mathbb{N}^\star$ đỉnh & $m \in \mathbb{N}^\star$ cạnh. Biểu diễn* adjacency list *của G bao gồm 1 danh sách n phần tử (các đỉnh của đồ thị) & 1 danh sách n danh sách với tổng cộng m phần tử (các đỉnh mục tiêu cho các cạnh của đồ thị). Danh sách tương ứng với đỉnh v chứa tất cả các đỉnh $w \in V$ với $(v, w) \in E$, đối với mọi đỉnh $v \in V$.*

Adjacency lists are not necessarily arranged in any particular order although there is, as a matter of fact, an order on the vertices & edges fixed by the adjacency list representation of the graph. Given the adjacency list representation of a graph $G = (V, E)$, vertex precedence is given by the order of the corresponding entries in the list of lists, & edge (v, w) precedes edge (v, z) iff vertex w precedes vertex z in the list corresponding to vertex v, for all edges $(v, w), (v, z) \in E$.

– Danh sách kề không nhất thiết được sắp xếp theo bất kỳ thứ tự cụ thể nào mặc dù trên thực tế, có 1 thứ tự trên các đỉnh & cạnh được cố định bởi biểu diễn danh sách kề của đồ thị. Với biểu diễn danh sách kề của đồ thị $G = (V, E)$, thứ tự ưu tiên của đỉnh được đưa ra theo thứ tự của các mục tương ứng trong danh sách các danh sách, & cạnh (v, w) đi trước cạnh (v, z) khi và chỉ khi đỉnh w đi trước đỉnh z trong danh sách tương ứng với đỉnh v, đối với mọi cạnh $(v, w), (v, z) \in E$.

As in the case of adjacency matrices, edges are implicit in the adjacency list representation of a graph, & those operations having an edge as argument for giving an edge as result cannot be implemented using an adjacency list representation. These operations are `G.edges(), G.incoming(v), G.outgoing(v), G.source(e), G.target(e)`, & `G.del_edge(e)`. Furthermore, `G.new_vertex(), G.del_vertex()` cannot be implemented unless the list of lists can be dynamically resized, & `G.new_edge(v, w)` can be implemented by appending vertex w to the list corresponding to vertex v, & takes $O(1)$ time, but the operation cannot give the new edge as result. The only remaining operation, `G.adjacent(v, w)`, can be implemented by scanning the list corresponding to vertex v & takes $O(\text{outdeg}(v))$ time.

– Giống như trong trường hợp của ma trận kề, các cạnh được ngầm định trong biểu diễn danh sách kề của đồ thị, & các phép toán có cạnh làm đối số để đưa ra cạnh làm kết quả không thể được triển khai bằng cách sử dụng biểu diễn danh sách kề. Các phép toán này là `G.edges(), G.incoming(v), G.outgoing(v), G.source(e), G.target(e)`, & `G.del_edge(e)`. Hơn nữa, `G.new_vertex(), G.del_vertex()` không thể được triển khai trừ khi danh sách các danh sách có thể được thay đổi kích thước động, & `G.new_edge(v, w)` có thể được triển khai bằng cách thêm đỉnh w vào danh sách tương ứng với đỉnh v, & mất $O(1)$ thời gian, nhưng phép toán không thể đưa ra cạnh mới làm kết quả. Hoạt động duy nhất còn lại, `G.adjacent(v, w)`, có thể được thực hiện bằng cách quét danh sách tương ứng với đỉnh v & mất $O(\text{outdeg}(v))$ thời gian.

6.3.1.3 Extended adjacency list – Danh sách kề mở rộng

The adjacency list representation can be extended by making edges explicit. The extended adjacency list representation of a graph consists of a list of vertices & a list of edges. Associated with each vertex are 2 lists of incoming & outgoing edges. Associated with each edge are its source & target vertices.

– Biểu diễn danh sách kề có thể được mở rộng bằng cách làm cho các cạnh trở nên rõ ràng. Biểu diễn danh sách kề mở rộng của đồ thị bao gồm 1 danh sách các đỉnh & 1 danh sách các cạnh. Liên kết với mỗi đỉnh là 2 danh sách các cạnh đến & đi. Liên kết với mỗi cạnh là các đỉnh nguồn & đích của nó.

Definition 54 (Extended adjacency list, [Val21], Def. 1.28, pp. 29–30)**.** *Let $G = (V, E)$ be a graph with $n \in \mathbb{N}^\star$ vertices & $m \in \mathbb{N}$ edges. The* extended adjacency list *representation of G consists of a list of n elements (the vertices of the graph), a list of m elements (the edges of the graph), & 2 lists of n lists of a total of m elements (the edges of the graph). The* incoming *list corresponding to vertex v contains all edges $(u, v) \in E$ coming into vertex v, for all vertices $v \in V$. The* outgoing *list corresponding to vertex v contains all edges $(v, w) \in E$ going out of vertex v, for all vertices $v \in V$. The source vertex v & the target vertex w are associated with each edge $(v, w) \in E$.*

Định nghĩa 45 (Danh sách kề mở rộng)**.** *Cho $G = (V, E)$ là 1 đồ thị có $n \in \mathbb{N}^\star$ đỉnh & $m \in \mathbb{N}$ cạnh. Biểu diễn* danh sách kề mở rộng *của G bao gồm 1 danh sách n phần tử (các đỉnh của đồ thị), 1 danh sách m phần tử (các cạnh của đồ thị), & 2 danh sách gồm n danh sách có tổng cộng m phần tử (các cạnh của đồ thị). Danh sách* đến *tương ứng với đỉnh v chứa tất cả các cạnh $(u, v) \in E$ đi vào đỉnh v, với mọi đỉnh $v \in V$. Danh sách* đi *tương ứng với đỉnh v chứa tất cả các cạnh $(v, w) \in E$ đi ra khỏi đỉnh v, với mọi đỉnh $v \in V$. Đỉnh nguồn v & đỉnh đích w được liên kết với mỗi cạnh $(v, w) \in E$.*

Edges are explicit in the extended adjacency list graph representation. Therefore, all of the previous operations can be implemented using the extended adjacency list representation & take $O(1)$ time, with the exception of `G.del_model(v)`, which takes $O(\deg v)$ time. The operations can be implemented as follows:

1. `G.vertices(), G.edges()` are, resp., the list of vertices & the list of edges of graph G.

   – `G.vertices(), G.edges()` tương ứng là danh sách các đỉnh & danh sách các cạnh của đồ thị G.

2. `G.incoming(v), G.outgoing(v)` are, resp., the list of edges coming into vertex v & the list of edges going out of vertex v.

   – `G.incoming(v), G.outgoing(v)` tương ứng là danh sách các cạnh đi vào đỉnh v & danh sách các cạnh đi ra khỏi đỉnh v.

3. `G.source(e), G.target(e)` are, resp., the source & the target vertex associated with edge e.

   – `G.source(e), G.target(e)` tương ứng là đỉnh nguồn & đỉnh đích được liên kết với cạnh e.

4. `G.new_vertex()` is implemented by appending a new vertex v to the list of vertices of graph G & returning vertex v.

   – `G.new_vertex()` được thực hiện bằng cách thêm 1 đỉnh mới v vào danh sách các đỉnh của đồ thị G & trả về đỉnh v.

5. `G.new_edge(v, w)` is implemented by appending a new edge e to the list of edges of graph G, setting to v the source vertex associated with edge e, setting to w the target vertex associated with edge e, appending e to the list of edges going out of vertex v & to the list of edges coming into vertex w, & returning edge e.

   – `G.new_edge(v, w)` được triển khai bằng cách thêm 1 cạnh mới e vào danh sách các cạnh của đồ thị G, đặt v là đỉnh nguồn được liên kết với cạnh e, đặt w là đỉnh đích được liên kết với cạnh e, thêm e vào danh sách các cạnh đi ra khỏi đỉnh v & vào danh sách các cạnh đi vào đỉnh w, & trả về cạnh e.

6. `G.del_vertex()` is implemented by performing `G.del_edge(e)` for each edge e in the list of edges coming into vertex v & for each edge e in the list of edges going out of vertex v & then deleting vertex v from the list of vertices of graph G.

   – `G.del_vertex()` được thực hiện bằng cách thực hiện `G.del_edge(e)` cho mỗi cạnh e trong danh sách các cạnh đi vào đỉnh v & cho mỗi cạnh e trong danh sách các cạnh đi ra khỏi đỉnh v & sau đó xóa đỉnh v khỏi danh sách các đỉnh của đồ thị G.

7. `G.del_edge(e)` is implemented by deleting edge e from the list of edges of graph G, from the list of edges coming into vertex `G.target(e)`, & from the list of edges going out of vertex `G.source(e)`.

   – `G.del_edge(e)` được thực hiện bằng cách xóa cạnh e khỏi danh sách các cạnh của đồ thị G, khỏi danh sách các cạnh đi vào đỉnh `G.target(e)`, & khỏi danh sách các cạnh đi ra khỏi đỉnh `G.source(e)`.

The extended adjacency list representation of a graph $G = (V, E)$ with n vertices & m edges takes $\Theta(m + n)$ space.

6.3.1.4 Adjacency map – Bản đồ kề

The adjacency list representation can also be extended by making edges explicit in a different way. The lists of incoming & outgoing edges can be replaced with dictionaries of source vertices to incoming edges & target vertices to outgoing edges. This allows for a more efficient adjacency test, although adding a logarithmic factor to the cost to all of the operations (when dictionaries are implemented using balanced trees) or turning the worst-case cost for all of the operations into the expected cost (when dictionaries are implemented using hashing).

– Biểu diễn danh sách kề cũng có thể được mở rộng bằng cách làm cho các cạnh rõ ràng theo 1 cách khác. Danh sách các cạnh đến & đi có thể được thay thế bằng các từ điển của các đỉnh nguồn đến các cạnh đến & các đỉnh đích đến các cạnh đi. Điều này cho phép kiểm tra kề hiệu quả hơn, mặc dù thêm 1 hệ số logarit vào chi phí cho tất cả các hoạt động (khi các từ điển được triển khai bằng cách sử dụng cây cân bằng) hoặc biến chi phí trường hợp xấu nhất cho tất cả các hoạt động thành chi phí dự kiến (khi các từ điển được triển khai bằng cách sử dụng băm).

The adjacency map representation of a graph consists of a dictionary D of vertices to a pair of dictionaries of vertices to edges: a 1st dictionary I of source vertices to incoming edges, & a 2nd dictionary O target vertices to outgoing edges.

– Biểu diễn danh sách kề cũng có thể được mở rộng bằng cách làm cho các cạnh rõ ràng theo 1 cách khác. Danh sách các cạnh đến & đi có thể được thay thế bằng các từ điển của các đỉnh nguồn đến các cạnh đến & các đỉnh đích đến các cạnh đi. Điều này cho phép kiểm tra kề hiệu quả hơn, mặc dù thêm 1 hệ số logarit vào chi phí cho tất cả các hoạt động (khi các từ điển được triển khai bằng cách sử dụng cây cân bằng) hoặc biến chi phí trường hợp xấu nhất cho tất cả các hoạt động thành chi phí dự kiến (khi các từ điển được triển khai bằng cách sử dụng băm).

Definition 55 (Adjacency map, [Val21], Def. 1.29, p. 31)**.** *Let $G = (V, E)$ be a graph with $n \in \mathbb{N}^\star$ & $m \in \mathbb{N}$ edges. The* adjacency map *representation of G consists of a dictionary of n elements (the vertices of the graph) to a pair of dictionaries of m elements (the source & target vertices for the edges of the graph, resp.). The* incoming *dictionary corresponding to vertex v contains the mappings $(u, (u, v))$ for all edges $(u, v) \in E$ coming into vertex v, for all vertex $v \in V$. The* outgoing *dictionary corresponding to vertex v contains the mappings $(v, (v, w))$ for all edges $(v, w) \in E$ going out of vertex v, for all vertices $v \in V$.*

Định nghĩa 46 (Bản đồ kề)**.** *Cho $G = (V, E)$ là 1 đồ thị có $n \in \mathbb{N}^\star$ & $m \in \mathbb{N}$ cạnh. Biểu diễn* adjacency map *của G bao gồm 1 từ điển gồm n phần tử (các đỉnh của đồ thị) tới 1 cặp từ điển gồm m phần tử (các đỉnh nguồn & đích cho các cạnh của đồ thị, tương ứng). Từ điển* incoming *tương ứng với đỉnh v chứa các ánh xạ $(u, (u, v))$ cho mọi cạnh $(u, v) \in E$ đi vào đỉnh v, cho mọi đỉnh $v \in V$. Từ điển* outgoing *tương ứng với đỉnh v chứa các ánh xạ $(v, (v, w))$ cho mọi cạnh $(v, w) \in E$ đi ra khỏi đỉnh v, cho mọi đỉnh $v \in V$.*

Both vertices & edges are explicit in the adjacency map representation of a graph. The vertices are the keys of the dictionary. The edges are the values in the dictionary of outgoing edges, & also the values in the dictionary of incoming edges, for all vertices of the graph. Therefore, all of the previous abstract operations can be implemented using the adjacency map representation & take expected $O(1)$ time, with the exception of `G.del_vertex()`, which takes $O(\deg v)$ expected time. The operations can be implemented as follows:

– Cả hai đỉnh & cạnh đều được nêu rõ trong biểu diễn bản đồ kề của đồ thị. Các đỉnh là các khóa của từ điển. Các cạnh là các giá trị trong từ điển của các cạnh đi ra, & cũng là các giá trị trong từ điển của các cạnh đến, cho mọi đỉnh của đồ thị. Do đó, tất cả các phép toán trừu tượng trước đó có thể được triển khai bằng cách sử dụng biểu diễn bản đồ kề & mất thời gian dự kiến là $O(1)$, ngoại trừ `G.del_vertex()`, mất thời gian dự kiến là $O(\deg v)$. Các phép toán có thể được triển khai như sau:

1. `G.vertices()` are the keys in dictionary D.

   – `G.vertices()` là các khóa trong từ điển D.

2. `G.edges()` are the values $D[v].O[w]$ for all keys v in dictionary D & for all keys w in dictionary $D[v].O$.

   – `G.edges()` là các giá trị $D[v].O[w]$ cho tất cả các khóa v trong từ điển D & cho tất cả các khóa w trong từ điển $D[v].O$.

3. `G.incoming(v)` are the values $D[v].I[u]$ for all keys u in dictionary $D[v].I$.

   – `G.incoming(v)` là các giá trị $D[v].I[u]$ cho tất cả các khóa u trong từ điển $D[v].I$.

4. `G.outgoing(v)` are the values $D[v].O[w]$ for all keys w in dictionary $D[v].O$.

   – `G.outgoing(v)` là các giá trị $D[v].O[w]$ cho tất cả các khóa w trong từ điển $D[v].O$.

5. `G.source(e), G.target(e)` are, resp., the source & the target vertex associated with edge e.

   – `G.source(e), G.target(e)` tương ứng là đỉnh nguồn & đỉnh đích được liên kết với cạnh e.

6. `G.new_vertex()` is implemented by inserting an entry in dictionary D, with key a new vertex v & value a pair of empty dictionaries $D[v].I, D[v].O$, & returning vertex v.

   – `G.new_vertex()` được thực hiện bằng cách chèn 1 mục vào từ điển D, với khóa là 1 đỉnh mới v & giá trị là 1 cặp từ điển rỗng $D[v].I, D[v].O$, & trả về đỉnh v.

7. `G.new_edge(v, w)` is implemented by setting to v the source vertex associated with a new edge e, setting to w the target vertex associated with edge e, inserting an entry in dictionary $D[v].O$ with key w & value e, inserting an entry in dictionary $D[w].I$ with key v & value e, & returning edge e.

   – `G.new_edge(v, w)` được triển khai bằng cách đặt v là đỉnh nguồn được liên kết với cạnh mới e, đặt w là đỉnh đích được liên kết với cạnh e, chèn 1 mục vào từ điển $D[v].O$ với khóa w & giá trị e, chèn 1 mục vào từ điển $D[w].I$ với khóa v & giá trị e, & trả về cạnh e.

8. `G.del_vertex(v)` is implemented by performing `G.del_edge(e)` for each entry with key u & value e in dictionary $D[v].I$ & for each entry with key w & value e in dictionary $D[v].O$ & then deleting the entry with key v from dictionary D.

   – `G.del_vertex(v)` được thực hiện bằng cách thực hiện `G.del_edge(e)` cho mỗi mục có khóa u & giá trị e trong từ điển $D[v].I$ & cho mỗi mục có khóa w & giá trị e trong từ điển $D[v].O$ & sau đó xóa mục có khóa v khỏi từ điển D.

9. `G.del_edge(e)` is implemented by deleting the entry with key w from dictionary $D[w].I$, where $v = G.source(e), w = G.target(e)$.

   – `G.del_edge(e)` được thực hiện bằng cách xóa mục có khóa w khỏi từ điển $D[w].I$, trong đó $v = G.source(e), w = G.target(e)$.

The adjacency test `G.adjacent(v,w)` can be defined in terms of the small set of abstract operations on graphs, using the abstract operations on basic data structures, by scanning the list of outgoing edges $G.outgoing(v)$ for an edge e s.t. $G.target(e) = w$. Now, besides the low space requirement, the main advantage of the adjacency map representation is the support of adjacency test in expected $O(1)$ time, when dictionaries are implemented using hashing, as follows:

– Kiểm tra kề `G.adjacent(v,w)` có thể được định nghĩa theo tập hợp nhỏ các phép toán trừu tượng trên đồ thị, sử dụng các phép toán trừu tượng trên các cấu trúc dữ liệu cơ bản, bằng cách quét danh sách các cạnh đi ra $G.outgoing(v)$ cho 1 cạnh e s.t. $G.target(e) = w$. Bây giờ, bên cạnh yêu cầu không gian thấp, lợi thế chính của biểu diễn bản đồ kề là hỗ trợ kiểm tra kề trong thời gian $O(1)$ dự kiến, khi các từ điển được triển khai bằng cách sử dụng băm, như sau:

- $G.adjacent(v, w)$ is true if $(w, e) \in D[v].O$ (where $e = D[v].O[w]$), & false otherwise.

  – $G.adjacent(v, w)$ là đúng nếu $(w, e) \in D[v].O$ (với $e = D[v].O[w]$), & false nếu không.

The adjacency map representation of a graph $G = (V, E)$ with n vertices & m edges takes $\Theta(m + n)$ space.

– Biểu diễn bản đồ kề của đồ thị $G = (V, E)$ với n đỉnh & m cạnh chiếm không gian $\Theta(m + n)$.

6.3.2 Representation of trees – Biểu diễn cây

As in the case of graphs, a tree representation consists of a collection of abstract operations, a concrete representation of trees by appropriate data structures, & implementation of the abstract operations using the concrete data structures.

– Giống như trong trường hợp đồ thị, biểu diễn cây bao gồm 1 tập hợp các phép toán trừu tượng, 1 biểu diễn cụ thể của cây theo các cấu trúc dữ liệu thích hợp và triển khai các phép toán trừu tượng bằng cách sử dụng các cấu trúc dữ liệu cụ thể.

The following collection of 13 abstract operations covers the needs of most of the algorithms on trees presented in [Val21].

– Bộ sưu tập 13 phép toán trừu tượng sau đây đáp ứng nhu cầu của hầu hết các thuật toán trên cây được trình bày trong [Val21].

1. `T.number_of_nodes()` gives the number of nodes $|V|$ of tree $T = (V, E)$.

   – `T.number_of_nodes()` cho biết số nút $|V|$ của cây $T = (V, E)$.

2. `T.root()` gives `root[T]`.

3. `T.is_root(v)` is true if node $v = root[T]$, & false otherwise.

4. `T.number_of_children(v)` gives `children[v]`, i.e., the number of nodes $w \in V$ s.t. $(v, w) \in E$.

5. $T.parent(v)$ gives $parent[v]$, i.e., the parent in T of node v.

6. $T.children(v)$ gives a list of the children of node v in T.

7. `T.is_leaf(v)` is true if node v is a leaf of T, & false otherwise.

8. `T.first_child(v), T.last_child(v)` give, resp., the 1st & the last of the children of node v, according to the order on the children of v fixed by the representation of T, or nil if v is a leaf node of T.

   – `T.first_child(v), T.last_child(v)` trả về, tương ứng, con thứ nhất & con cuối cùng của nút v, theo thứ tự các con của v được xác định bởi biểu diễn của T, hoặc nil nếu v là nút lá của T.

9. `T.previous_sibling(v), T.next_sibling(v)` give, resp., the previous child of $T.parent(v)$ before vertex v & the next child of $T.parent(v)$ after vertex v, according to the order on the children of $T.parent(v)$ fixed by the representation of T, or *nil* if v is a 1st child or a last child, resp.

   – `T.previous_sibling(v), T.next_sibling(v)` trả về, tương ứng, con trước của $T.parent(v)$ trước đỉnh v & con tiếp theo của $T.parent(v)$ sau đỉnh v, theo thứ tự các con của $T.parent(v)$ được cố định bởi biểu diễn của T, hoặc *nil* nếu v là con đầu tiên hoặc con cuối cùng, tương ứng.

10. `T.is_first_child(v), T.is_last_child(v)` are true if node v is, resp., the 1st & the last of the children of node $T.parent(v)$, according to the order on the children of $T.parent(v)$ fixed by the representation of T, & false otherwise.

    – `T.is_first_child(v), T.is_last_child(v)` là đúng nếu nút v là nút con thứ nhất & là nút con cuối cùng của nút $T.parent(v)$, theo thứ tự các nút con của $T.parent(v)$ được xác định bởi biểu diễn của T, & false nếu không.

The previous operations support iteration over the children of a node in a tree, much in the same way that the graph operations support iteration over the vertices & edges of a graph. In the following procedure, the children of node v in tree T are assigned in turn to variable w.

– Các hoạt động trước đó hỗ trợ lặp lại trên các con của 1 nút trong cây, tương tự như cách các hoạt động đồ thị hỗ trợ lặp lại trên các đỉnh & cạnh của đồ thị. Trong quy trình sau, các con của nút v trong cây T lần lượt được gán cho biến w.

```
let w be the 1st child of node v in T // w = T.first_child(v)
while w != nil do
    process node w
    let w be the next sibling of node w in T // w = T.next_sibling(w)
```

The following procedure is an alternative form of iteration over the children w of a node v in a tree T.

```
if v is not a leaf node of T then // not T.is_leaf(v)
    let w be the 1st child of node v in T // w = T.first_child(v)
    process node w
    while w is not the last child of node v in T do // w != T.last_child(v)
        let w be the next sibling of node w in T // w = T.next_sibling(w)
        process node w
```

The following procedure is still another, more compact form of iteration over the children w of a node v in a tree T.

```
for all children w of node v in T do // w in T.children(v)
    process node w
```

6.3.2.1 Array of parents – Mảng cha mẹ

The data structures most often used for representing trees are the array-of-parents representation & the 1st-child, next-sibling representation. The array-of-parents representation of a tree is an array of nodes, with an entry for each node in the tree, where the entry corresponding to node v has the value $parent[v]$ if v is not the root of the tree, & it has the value *nil* otherwise.

– Cấu trúc dữ liệu thường được sử dụng nhất để biểu diễn cây là biểu diễn mảng cha mẹ & biểu diễn con thứ nhất, anh chị em kế tiếp. Biểu diễn mảng cha mẹ của cây là 1 mảng các nút, với 1 mục nhập cho mỗi nút trong cây, trong đó mục nhập tương ứng với nút v có giá trị $parent[v]$ nếu v không phải là gốc của cây, & nó có giá trị *nil* nếu không.

Definition 56 (Array-of-parents representation of trees, [Val21], Def. 1.30, p. 34)**.** *Let $T = (V, E)$ be a tree with n nodes. The* array-of-parents *representation of T is an array P of n nodes, indexed by the nodes of the tree, where $P[v] = parent[v]$ if $v \neq root[T]$, & $P[v] = nil$ otherwise, for all nodes $v \in V$.*

Định nghĩa 47 (Biểu diễn mảng cha mẹ của cây)**.** *Cho $T = (V, E)$ là 1 cây có n nút. Biểu diễn* array-of-parents *của T là 1 mảng P gồm n nút, được lập chỉ mục theo các nút của cây, trong đó $P[v] = parent[v]$ nếu $v \neq root[T]$, & $P[v] = nil$ nếu không, đối với tất cả các nút $v \in V$.*

The array of parent nodes is not necessarily arranged in any particular order although there is, as a matter of fact, an order on the nodes fixed by the representation. Given the array-of-parents representation of a tree, node precedence is given by the order of the nodes as array indices, & precedence between sibling nodes is also given by the order of array indices.

– Mảng các nút cha không nhất thiết phải được sắp xếp theo bất kỳ thứ tự cụ thể nào mặc dù trên thực tế, có 1 thứ tự trên các nút được cố định bởi biểu diễn. Với biểu diễn mảng cha của 1 cây, thứ tự ưu tiên của nút được đưa ra theo thứ tự của các nút dưới dạng chỉ số mảng, & thứ tự ưu tiên giữa các nút anh chị em cũng được đưa ra theo thứ tự của chỉ số mảng.

Let $T = (V, E)$ be a tree with n nodes, & let P be the array-of-parents representation of T. Operations

`T.root()`, `T.number_of_children(v)`, `T.children(v)`, `T.is_leaf(v)`, `T.first_child(v)`, `T.last_child(v)`, `T.previous_sibling(v)`, `T.next_sibling(v)`, `T.is_first_child(v)`, `T.is_last_child(v)`

require scanning the whole array & thus take $O(n)$ time. The remaining operations can be implemented to take $O(1)$ time as follows:

- `T.number_of_nodes()` is the size of array P.
- `T.is_root(v)` is true if $P[v] = nil$, & false otherwise.
- $T.parent(v)$ is equal to $P[v]$.

– Cho $T = (V, E)$ là 1 cây có n nút, & cho P là biểu diễn mảng cha mẹ của T. Các phép toán

`T.root()`, `T.number_of_children(v)`, `T.children(v)`, `T.is_leaf(v)`, `T.first_child(v)`, `T.last_child(v)`, `T.previous_sibling(v)`, `T.next_sibling(v)`, `T.is_first_child(v)`, `T.is_last_child(v)`

yêu cầu quét toàn bộ mảng & do đó mất $O(n)$ thời gian. Các phép toán còn lại có thể được triển khai để mất $O(1)$ thời gian như sau:

- `T.number_of_nodes()` là kích thước của mảng P.
- `T.is_root(v)` là đúng nếu $P[v] = nil$, & false nếu không.
- $T.parent(v)$ bằng $P[v]$.

6.3.2.2 1st-child, next-sibling – Con thứ nhất, anh chị em ruột tiếp theo

The 1st-child, next-sibling representation, on the other hand, consists of 2 arrays of nodes, each of them with an entry for each node in the tree, where the entries corresponding to node v have, resp., the value $first[v]$, or nil if v is a leaf node, & $next[v]$, or nil if v is a last sibling.

– Mặt khác, biểu diễn con thứ nhất, anh chị em kế cận bao gồm 2 mảng các nút, mỗi mảng có 1 mục cho mỗi nút trong cây, trong đó các mục tương ứng với nút v có giá trị tương ứng là $first[v]$ hoặc nil nếu v là nút lá, & $next[v]$ hoặc nil nếu v là anh chị em cuối cùng.

Definition 57 (1st-child next-sibling representation of trees, [Val21], Def. 1.31, p. 35)**.** *Let $T = (V, E)$ be a tree with n nodes. The* 1st-child next-sibling *representation of T is a pair (F, N) of arrays of n nodes, indexed by the nodes of the tree, where $F[v] = first[v]$ if v is not a leaf node, $F[v] = nil$ otherwise, $N[v] = next[v]$ if v is not a last child node, & $N[v] = nil$ otherwise, for all nodes $v \in V$.*

Định nghĩa 48 (Biểu diễn cây của con thứ nhất & anh chị em tiếp theo)**.** *Cho $T = (V, E)$ là 1 cây có n nút. Biểu diễn* 1st-child next-sibling *của T là 1 cặp (F, N) mảng của n nút, được lập chỉ mục theo các nút của cây, trong đó $F[v] = first[v]$ nếu v không phải là nút lá, $F[v] = nil$ nếu không, $N[v] = next[v]$ nếu v không phải là nút con cuối cùng, & $N[v] = nil$ nếu không, đối với mọi nút $v \in V$.*

As with the array-of-parents representation, the arrays of 1st children & next siblings are not necessarily arranged in any particular order, as long as they are arranged in the same order. Anyway, there is an order on the nodes fixed by the representation. Given the 1st-child, next-sibling representation of a tree, node precedence is given by the order to the nodes as array indices, & precedence between sibling nodes is also given by the order of array indices.

– Giống như biểu diễn mảng cha mẹ, các mảng con thứ nhất & anh chị em tiếp theo không nhất thiết phải được sắp xếp theo bất kỳ thứ tự cụ thể nào, miễn là chúng được sắp xếp theo cùng thứ tự. Dù sao thì, có 1 thứ tự trên các nút được cố định bởi biểu diễn. Với biểu diễn con thứ nhất, anh chị em tiếp theo của 1 cây, thứ tự ưu tiên của nút được đưa ra theo thứ tự cho các nút như các chỉ số mảng, & thứ tự ưu tiên giữa các nút anh chị em cũng được đưa ra theo thứ tự của các chỉ số mảng.

Let $T = (V, E)$ be a tree with n nodes, & let (F, N) be the 1st-child, next-sibling representation of T. Operations `T.root()`, `T.is_roo` require scanning both of arrays F, N, & `T.previous_sibling(v)`, `T.is_first_child(v)` require scanning array N, & thus take $O(n)$ time. Further, `T.number_of_children(v)`, `T.last_child(v)` require following up to $children[v]$ next-sibling links, where the former also requires following a 1st-child link, & take $O(children[v])$ time. The remaining operations can be implemented to take $O(1)$ time as follows:

1. `T.number_of_nodes()` is the size of array F, or the size of array N.
2. `T.is_leaf(v)` is true if $F[v] = nil$, & false otherwise.
3. `T.first_child(v)` is given by $F[v]$.
4. `T.next_sibling(v)` is given by $N[v]$.

5. `T.is_last_child(v)` is true if $N[v] = nil$, & false otherwise.

– Cho $T = (V, E)$ là 1 cây có n nút, & cho (F, N) là biểu diễn con thứ nhất, anh chị em kế cận của T. Các thao tác `T.root(), T.is_root(v), T.parent(v), T.children(v)` yêu cầu quét cả hai mảng F, N, & `T.previous_sibling(v), T.is_first_` yêu cầu quét mảng N, & do đó mất $O(n)$ thời gian. Hơn nữa, `T.number_of_children(v), T.last_child(v)` yêu cầu theo dõi tới $children[v]$ liên kết anh chị em kế cận, trong khi thao tác trước cũng yêu cầu theo dõi 1 liên kết con thứ nhất, & mất $O(children[v])$ thời gian. Các phép toán còn lại có thể được triển khai để mất $O(1)$ thời gian như sau:

1. `T.number_of_nodes()` là kích thước của mảng F hoặc kích thước của mảng N.
2. `T.is_leaf(v)` là true nếu $F[v] = nil$, & false nếu không.
3. `T.first_child(v)` được chỉ định bởi $F[v]$.
4. `T.next_sibling(v)` được chỉ định bởi $N[v]$.
5. `T.is_last_child(v)` là true nếu $N[v] = nil$, & false nếu không.

6.3.2.3 Graph-based representation of trees

Trees can also be represented using the small set of abstract operations on graphs, which can be implemented using either the extended adjacency list representation or the adjacency map representation. The following operations are defined in terms of the graph representation, using the abstract operations on basic data structures presented as follows:

1. `T.number_of_nodes()` is given by $T.vertices().size()$.
2. $T.root()$ requires following the path of parent nodes starting off with any node of the tree & then storing the root of the tree in the attribute *root*.

   $T.root()$ yêu cầu phải theo đường dẫn của các nút cha bắt đầu bằng bất kỳ nút nào của cây & sau đó lưu trữ gốc của cây trong thuộc tính *root*.

```
function root(T)
    if T.root != nil then
        return T.root
    if T is empty then
        return nil
    v <= T.vertices().front()
    while T.incoming(v) is not empty do
        v <- T.parent(v)
    T.root <- v
    return v
```

3. `T.is_root(v)` is given by $T.incoming(v).empty()$.
4. `T.number_of_children(v)` is given by $T.outgoing(v).size()$.
5. $T.parent(v)$ is given by $T.incoming(v).front().source()$.
6. $T.children(v)$ is given by the list $[T.target(e) : e \in T.outgoing(v)]$.
7. `T.is_leaf(v)|` is given by $T.outgoing(v).empty()$.
8. `T.first_child(v)` is given by $T.outgoing(v).front(v).target()$.
9. `T.last_child(v)` is given by $T.outgoing(v).back(v).target()$.
10. `T.previous_sibling(v)` requires accessing the previous edge in the list of edges coming out of $parent[v]$.

```
function previous_sibling(T, v)
    if T.is_root(v) then
        return nil
    e <- T.incoming(v).front(v)
    u <- T.source(e)
    if T.outgoing(u).prev(e) = nil then
        return nil
    else
        return T.outgoing(u).prev(e).target()
```

11. `T.next_sibling(v)` requires accessing the next edge in the list of edges coming out of $parent[v]$.

```
function previous_sibling(T, v)
    if T.is_root(v) then
        return nil
    e <- T.incoming(v).front()
    u <- T.source(e)
    if T.outgoing(u).next(e) = nil then
        return nil
    else
        return T.outgoing(u).next(e).target()
```

12. `T.is_first_child(v)` is true if `T.previous_sibling(v)` is equal to *nil*, & false otherwise.

13. `T.is_last_child(v)`| is true if `T.next_sibling(v)` is equal to *nil*, & false otherwise.

All of the operations take $O(1)$ time with the only exception of $T.root()$, which takes time linear in depth or height of the tree. These are worst-case time bounds when using extended adjacency lists, & expected time bounds when using adjacency maps. The representation uses $O(n)$ space.

– Tất cả các hoạt động đều mất $O(1)$ thời gian ngoại trừ $T.root()$, mất thời gian tuyến tính theo chiều sâu hoặc chiều cao của cây. Đây là giới hạn thời gian trường hợp xấu nhất khi sử dụng danh sách kề mở rộng, & giới hạn thời gian dự kiến khi sử dụng bản đồ kề. Biểu diễn sử dụng không gian $O(n)$.

Note 1. *Ordered or embedded graphs are the subject of topological graph theory. Adjacency matrices are most often used in combinatorics & graph theory. Adjacency lists are the preferred graph representation in computer science, though, because a large number of graph algorithms can be implemented to run in time linear in the number of vertices & edges in the graph using an adjacency list representation, while no graph algorithm can be implemented to run in linear time using an adjacency matrix representation. The adjacency map representation was adopted in the NetworkX package for complex network analysis in Python. Implicit representations of static trees include the Prüfer sequences, which are used for counting & generating trees.*

– Đồ thị có thứ tự hoặc nhúng là chủ đề của lý thuyết đồ thị tôpô. Ma trận kề thường được sử dụng nhất trong tổ hợp & lý thuyết đồ thị. Tuy nhiên, danh sách kề là biểu diễn đồ thị được ưa chuộng trong khoa học máy tính, vì có thể triển khai nhiều thuật toán đồ thị để chạy tuyến tính theo thời gian theo số đỉnh & cạnh trong đồ thị bằng cách sử dụng biểu diễn danh sách kề, trong khi không có thuật toán đồ thị nào có thể được triển khai để chạy tuyến tính theo thời gian bằng cách sử dụng biểu diễn ma trận kề. Biểu diễn bản đồ kề được áp dụng trong gói NetworkX để phân tích mạng phức tạp trong Python. Các biểu diễn ngầm định của cây tĩnh bao gồm chuỗi Prüfer, được sử dụng để đếm & tạo cây.

Problem 85 ([Val21], 1.1., p. 39)**.** *Determine the size of the complete graph K_n on n vertices & the complete bipartite graph $K_{p,q}$ on $p+q$ vertices.*

– Xác định kích thước của đồ thị đầy đủ K_n trên n đỉnh & đồ thị hai phần đầy đủ $K_{p,q}$ trên $p+q$ đỉnh.

Problem 86 ([Val21], 1.2., p. 39)**.** *Determine the values of n for which the circle graph C_n on n vertices is bipartite, & also the values of n for which the complete graph K_n is bipartite.*

– Xác định các giá trị của n để đồ thị tròn C_n trên n đỉnh là 2 phần, & các giá trị của n để đồ thị hoàn chỉnh K_n là 2 phần.

Problem 87 ([Val21], 1.3., p. 39)**.** *Give all the spanning trees of the graph in* Fig. 1.30*, & also the number of spanning trees of the underlying undirected graph.*

– Cung cấp tất cả các cây khung của đồ thị trong Fig. 1.30*, & số lượng cây khung của đồ thị vô hướng cơ bản.*

Problem 88 ([Val21], 1.4., p. 39)**.** *Extended the adjacency matrix graph representation by replacing those operations having an edge as argument or giving an edge or a list of edges as result, by corresponding operations having as argument or giving as result the source & target vertices of the edge or edges:* `G.del_edge(v, w)`*, $G.edges()$, $G.incoming(v)$, $G.outgoing(v)$, $G.source(v,w)$, $G.target$*

– Mở rộng biểu diễn đồ thị ma trận kề bằng cách thay thế các phép toán có 1 cạnh làm đối số hoặc đưa ra 1 cạnh hoặc danh sách các cạnh làm kết quả bằng các phép toán tương ứng có đối số hoặc đưa ra kết quả là các đỉnh nguồn & đích của cạnh hoặc các cạnh: `G.del_edge(v, w)`*, $G.edges()$, $G.incoming(v)$, $G.outgoing(v)$, $G.source(v,w)$, $G.target(v,w)$.*

Problem 89 ([Val21], 1.5., p. 39)**.** *Extended the 1st-child, next-sibling tree representation, in order to support the collection of basic operations but* `T.root()`*,* `T.number_of_children(v)`*,* `T.children(v)` *in $O(1)$ time.*

– Mở rộng biểu diễn cây con thứ nhất, anh chị em tiếp theo, nhằm hỗ trợ việc thu thập các phép toán cơ bản nhưng `T.root()`*,* `T.number_of_children(v)`*,* `T.children(v)` *trong thời gian $O(1)$.*

Problem 90 ([Val21], 1.6., p. 39)**.** *Show how to double check that the graph-based representation of a tree is indeed a tree, in time linear in the size of the tree.*

– Hiển thị cách kiểm tra lại xem biểu diễn dựa trên đồ thị của 1 cây có thực sự là 1 cây hay không, theo thời gian tuyến tính với kích thước của cây.

Chương 7

Algorithms on Graphs – Các Thuật Toán Trên Đồ Thị

Contents

7.1 Preliminaries

Phần này nêu sơ lược 1 số khái niệm cơ bản & bài toán cơ bản trong lý thuyết đồ thị.

7.1.1 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 & 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 & 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 & tối ưu hóa tổ hợp, ví dụ như tìm kiếm các phép khớp, các nhóm tùy chọn & 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 58 (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).

7.1.2 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 & 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 & 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 & 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 & 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.

7.1.2.1 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

7.1.2.2 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 & 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"), & 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 & 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; & 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 & 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 & 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 & 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 & tìm kiếm theo chiều rộng, cũng như nhiều phương pháp cắt tỉa cây tìm kiếm dựa trên phương pháp heuristic như quay lui & rẽ nhánh & 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 & 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 & 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 & 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 & 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 & 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.

7.2 Algorithmic Techniques – Các Kỹ Thuật Thuật Toán

Resources – Tài nguyên.

1. [Val02; Val21]. Gabriel Valiente. *Algorithms on Trees & Graphs With Python Code*. 2e. Chap. 2: Algorithmic Techniques.

7.2.1 Tree edit distance problem – Bài toán khoảng cách chỉnh sửa cây

The problem of transforming or *editing* trees is a primary means of tree comparison, with practical applications in combinatorial pattern matching, pattern recognition, chemical structure search, computational molecular biology, & other areas of engineering & life sciences. In the most general sense, trees can be transformed by the application of elementary edit operations, which have a certain cost or weight associated with them, & the edit distance between 2 trees is the cost of a least-cost sequence of elementary edit operations, or the length of the shortest sequence of elementary edit operations, that allows to transform 1 of the trees into the other.

– Vấn đề biến đổi hoặc chỉnh sửa cây là phương tiện chính để so sánh cây, với các ứng dụng thực tế trong việc khớp mẫu tổ hợp, nhận dạng mẫu, tìm kiếm cấu trúc hóa học, sinh học phân tử tính toán, & các lĩnh vực kỹ thuật khác & khoa học sự sống. Theo nghĩa chung nhất, cây có thể được biến đổi bằng cách áp dụng các thao tác chỉnh sửa cơ bản, có 1 chi phí hoặc trọng số nhất định liên quan đến chúng, & khoảng cách chỉnh sửa giữa 2 cây là chi phí của chuỗi thao tác chỉnh sửa cơ bản có chi phí thấp nhất hoặc độ dài của chuỗi thao tác chỉnh sửa cơ bản ngắn nhất, cho phép biến đổi 1 trong các cây thành cây kia.

Rooted ordered trees are considered in [Val21, Chap. 2], with the following elementary edit operations: deletion of a leaf node from a tree, insertion of a new leaf node into a tree, & substitution or replacement of a node in a tree with a node in another tree.

– Cây có thứ tự có gốc được xem xét trong [Val21, Chap. 2], với các hoạt động chỉnh sửa cơ bản sau: xóa 1 nút lá khỏi cây, chèn 1 nút lá mới vào cây, & thay thế hoặc thay thế 1 nút trong cây bằng 1 nút trong cây khác.

Definition 59 (Elementary edit operation: deletion, substitution, insertion, [Val21], Def. 2.1, p. 45)**.** *Let* $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ *be ordered trees. An* elementary edit operation *on* T_1, T_2 *is either the* deletion *from* T_1 *of a leaf node* $v \in V_1$*, denoted by* $v \mapsto \lambda$ *or* (v, λ)*; the* substitution *or replacement of a node* $v \in V_1$ *with a node* $w \in V_2$*, denoted by* $v \mapsto w$ *or* (v, w)*; or the* insertion *into* T_2 *of a node* $w \notin V_2$ *as a new leaf, denoted by* $\lambda \mapsto w$ *or* (λ, w)*. Deletion of a non-root node* $v \in V_1$ *implies the deletion of the edge* $(parent[v], v) \in E_1$*, & insertion of a non-root node* $w \notin V_2$ *as a child of a leaf node* $parent[w] \in V_2$ *implies insertion of an edge* $(parent[w], w \notin E_2)$*.*

Định nghĩa 49 (Thao tác chỉnh sửa cơ bản: xóa, thay thế, chèn)**.** *Cho* $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ *là các cây có thứ tự. Một* thao tác chỉnh sửa cơ bản *trên* T_1, T_2 *là* xóa *khỏi* T_1 *1 nút lá* $v \in V_1$*, ký hiệu là* $v \mapsto \lambda$ *hoặc* (v, λ)*;* thay thế *hoặc thay thế 1 nút* $v \in V_1$ *bằng 1 nút* $w \in V_2$*, ký hiệu là* $v \mapsto w$ *hoặc* (v, w)*; hoặc* chèn *vào* T_2 *của 1 nút* $w \notin V_2$ *như 1 nút lá mới, ký hiệu là* $\lambda \mapsto w$ *hoặc* (λ, w)*. Việc xóa 1 nút không phải gốc* $v \in V_1$ *ngụ ý việc xóa cạnh* $(parent[v], v) \in E_1$*, & việc chèn 1 nút không phải gốc* $w \notin V_2$ *làm nút con của 1 nút lá* $parent[w] \in V_2$ *ngụ ý việc chèn 1 cạnh* $(parent[w], w \notin E_2)$*.*

Deletion & insertion operations are thus made on leaves only. The deletion of a non-leaf node requires 1st the deletion of the whole subtree rooted at the node, & the same applies to the insertion of non-leaves.

– Do đó, các thao tác xóa & chèn chỉ được thực hiện trên các lá. Việc xóa 1 nút không phải lá trước tiên yêu cầu xóa toàn bộ cây con có gốc tại nút, & tương tự cũng áp dụng cho việc chèn các nút không phải lá.

Now, a tree can be transformed into another tree by application of a sequence of elementary edit operations. Recall that a relation $R \subseteq A \times B$ is a set of ordered pairs $\{(a, b); a \in A,\ b \in B\}$. An *ordered relation* $R \subseteq A \times B$ is an ordered set, or sequence, of ordered pairs $[(a, b); a \in A,\ b \in B]$.

– Bây giờ, 1 cây có thể được chuyển đổi thành 1 cây khác bằng cách áp dụng 1 chuỗi các phép chỉnh sửa cơ bản. Nhớ lại rằng 1 quan hệ $R \subseteq A \times B$ là 1 tập hợp các cặp có thứ tự $\{(a, b); a \in A,\ b \in B\}$. Một *quan hệ có thứ tự* $R \subseteq A \times B$ là 1 tập hợp có thứ tự, hoặc chuỗi, các cặp có thứ tự $[(a, b); a \in A,\ b \in B]$.

Definition 60 (Transformation of trees, [Val21], Def. 2.2, p. 46)**.** *Let* $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ *be ordered trees. A* transformation *of* T_1 *into* T_2 *is an ordered relation* $E \subseteq (V_1 \cup \{\lambda\}) \times (V_2 \cup \{\lambda\})$ *s.t.*

- $\{v \in V_1; (v, w) \in E,\ w \in V_2 \cup \{\lambda\}\} = V_1$.
- $\{v \in V_2; (v, w) \in E,\ w \in V_1 \cup \{\lambda\}\} = V_2$.
- $(v_1, w), (v_2, w) \in E \Leftarrow v_1 = v_2$*, for all nodes* $v_1, v_2 \in V_1 \cup \{\lambda\}$ *&* $w \in V_2$.
- $(v, w_1), (v, w_2) \in E \Rightarrow w_1 = w_2$*, for all nodes* $v \in V_1$ *&* $w_1, w_2 \in V_2 \cup \{\lambda\}$.

Định nghĩa 50 (Phép biến đổi của cây)**.** *Cho* $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ *là các cây có thứ tự. Một* phép biến đổi *của* T_1 *thành* T_2 *là 1 quan hệ có thứ tự* $E \subseteq (V_1 \cup \{\lambda\}) \times (V_2 \cup \{\lambda\})$ *s.t.*

- $\{v \in V_1; (v, w) \in E,\ w \in V_2 \cup \{\lambda\}\} = V_1$.
- $\{v \in V_2; (v, w) \in E,\ w \in V_1 \cup \{\lambda\}\} = V_2$.
- $(v_1, w), (v_2, w) \in E \Leftarrow v_1 = v_2$*, cho tất cả các nút* $v_1, v_2 \in V_1 \cup \{\lambda\}$ *&* $w \in V_2$.
- $(v, w_1), (v, w_2) \in E \Rightarrow w_1 = w_2$*, cho tất cả các nút* $v \in V_1$ *&* $w_1, w_2 \in V_2 \cup \{\lambda\}$.

Remark 15 ([Val21], Rmk. 2.1, p. 46). *In a transformation $E \subset (V_1 \cup \{\lambda\}) \times (V_2 \cup \{\lambda\})$ of an ordered tree $T_1 = (V_1, E_1)$ into an ordered tree $T_2 = (V_2, E_2)$, the relation $E \cap V_1 \times V_2$ is a bijection. In fact, condition $(v_1, w), (v_2, w) \in E \Leftarrow v_1 = v_2$ establishes injectivity, & $(v, w_1), (v, w_2) \in E$ implies $w_1 = w_2$ establishes the surjectivity of $E \cap V_1 \times V_2$.*

– Trong phép biến đổi $E \subset (V_1 \cup \{\lambda\}) \times (V_2 \cup \{\lambda\})$ của 1 cây có thứ tự $T_1 = (V_1, E_1)$ thành 1 cây có thứ tự $T_2 = (V_2, E_2)$, quan hệ $E \cap V_1 \times V_2$ là 1 song ánh. Trên thực tế, điều kiện $(v_1, w), (v_2, w) \in E \Leftarrow v_1 = v_2$ thiết lập tính đơn ánh, & $(v, w_1), (v, w_2) \in E$ ngụ ý $w_1 = w_2$ thiết lập tính toàn ánh của $E \cap V_1 \times V_2$.

Not every sequence of elementary edit operations corresponds to a valid transformation between 2 ordered trees, though. On the 1 hand, deletions & insertions must appear in a bottom-up order – e.g., according to a postorder traversal of the trees – in order to ensure that deletions & insertions are indeed made on leaves. In the postorder traversal of an ordered tree, which will be introduced in Sect. 3.2, a postorder traversal of the subtree rooted in turn at each of the children of the root is performed 1st, & the root of the tree is visited last, where the subtree rooted at the 1st child is traversed 1st, followed by the subtree rooted at the next sibling, etc.

– Tuy nhiên, không phải mọi trình tự của các thao tác chỉnh sửa cơ bản đều tương ứng với 1 phép biến đổi hợp lệ giữa 2 cây có thứ tự. Mặt khác, các phép xóa & chèn phải xuất hiện theo thứ tự từ dưới lên – ví dụ, theo phép duyệt hậu thứ tự các cây – để đảm bảo rằng các phép xóa & chèn thực sự được thực hiện trên các lá. Trong phép duyệt hậu thứ tự của 1 cây có thứ tự, sẽ được giới thiệu trong Mục 3.2, phép duyệt hậu thứ tự của cây con có gốc lần lượt tại mỗi con của gốc được thực hiện trước, & gốc của cây được truy cập cuối cùng, trong đó cây con có gốc tại con thứ nhất được duyệt trước, tiếp theo là cây con có gốc tại anh chị em tiếp theo, v.v.

On the other hand, for a tree transformation to be valid, both parent & sibling order must be preserved by the transformation, in order to ensure that the result of the transformation is indeed an ordered tree. I.e., in a valid transformation of an ordered tree T_1 into another ordered tree T_2, the parent of a non-root node of T_1 which is substituted or replaced with a non-root node of T_2 must be substituted or replaced with the parent of the node of T_2. Further, whenever sibling nodes of T_1 are substituted or replaced with sibling nodes of T_2, the substitution must preserve their relative order.

– Mặt khác, để 1 phép biến đổi cây có hiệu lực, cả thứ tự cha & anh chị em phải được bảo toàn bởi phép biến đổi, để đảm bảo rằng kết quả của phép biến đổi thực sự là 1 cây có thứ tự. Tức là, trong 1 phép biến đổi hợp lệ của 1 cây có thứ tự T_1 thành 1 cây có thứ tự khác T_2, cha của 1 nút không phải gốc của T_1 được thay thế hoặc thay thế bằng 1 nút không phải gốc của T_2 phải được thay thế hoặc thay thế bằng cha của nút T_2. Hơn nữa, bất cứ khi nào các nút anh chị em của T_1 được thay thế hoặc thay thế bằng các nút anh chị em của T_2, phép thay thế phải bảo toàn thứ tự tương đối của chúng.

The latter requirement for a transformation between 2 ordered trees to be valid is formalized by the notion of *mapping*, also known as *trace*.

– Yêu cầu sau để phép biến đổi giữa 2 cây có thứ tự là hợp lệ được chính thức hóa bằng khái niệm ánh xạ, còn được gọi là vết.

Definition 61 (Mapping, [Val21], Def. 2.3, p. 47). *Let $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ be ordered trees, let $W_1 \subseteq V_1, W_2 \subseteq V_2$. A* mapping *$M$ of T_1 to T_2 is a bijection $M \subseteq W_1 \times W_2$ s.t.*

- *$(root[T_1], root[T_2]) \in M$ if $M \neq \emptyset$.*
- *$(v, w) \in M$ only if $(parent[v], parent[w]) \in M$, for all non-root nodes $v \in W_1, w \in W_2$.*
- *v_1 is a left sibling of v_2 iff w_1 is a left sibling of w_2, for all nodes $v_1, v_2 \in W_1$ & $w_1, w_2 \in W_2$ with $(v_1, w_1), (v_2, w_2) \in M$.*

Định nghĩa 51 (Phép ánh xạ). *Cho $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ là các cây có thứ tự, cho $W_1 \subseteq V_1, W_2 \subseteq V_2$. Một* mapping *$M$ của T_1 tới T_2 là 1 song ánh $M \subseteq W_1 \times W_2$ s.t.*

- *$(root[T_1], root[T_2]) \in M$ nếu $M \neq \emptyset$.*
- *$(v, w) \in M$ chỉ nếu $(parent[v], parent[w]) \in M$, đối với mọi nút không phải gốc $v \in W_1, w \in W_2$.*
- *v_1 là 1 nút anh em bên trái của v_2 nếu và chỉ nếu w_1 là 1 nút anh em bên trái của w_2, đối với mọi nút $v_1, v_2 \in W_1$ & $w_1, w_2 \in W_2$ với $(v_1, w_1), (v_2, w_2) \in M$.*

Lemma 5 ([Val21], Lem. 2.1, p. 48). *Let M be a mapping of an ordered tree $T_1 = (V_1, E_1)$ to another ordered tree $T_2 = (V_2, E_2)$. Then, $depth[v] = depth[w]$, $\forall (v, w) \in M$.*

A transformation is thus valid if node deletions & insertions are made on leaves only, & node substitutions constitute a mapping.

– Do đó, 1 phép biến đổi là hợp lệ nếu việc xóa và chèn nút chỉ được thực hiện trên các nút lá, còn việc thay thế nút tạo thành 1 phép ánh xạ.

Definition 62 (Valid transformation, [Val21], Def. 2.4, p. 48). *Let $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ be ordered trees. A transformation $E \subseteq (V_1 \cup \{\lambda\}) \times (V_2 \cup \{\lambda\})$ of T_1 into T_2 is said to be a* valid transformation *if*

- (v_j, λ) *occurs before* (v_i, λ) *in* E*, for all* $(v_i, \lambda), (v_j, \lambda) \in E \cap V_1 \times \{\lambda\}$ *s.t. node* v_j *is a descendant of node* v_i *in* T_1.
- (λ, w_i) *occurs before* (λ, w_j) *in* E*, for all* $(\lambda, w_i), (\lambda, w_j) \times E \cap \{\lambda\} \times V_2$ *s.t. node* w_j *is a descendant of node* w_i *in* T_2.
- $E \cap V_1 \times V_2$ *is a mapping of* T_1 *to* T_2.

Định nghĩa 52 (Biến đổi hợp lệ)**.** *Cho* $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ *là các cây có thứ tự. Phép biến đổi* $E \subseteq (V_1 \cup \{\lambda\}) \times (V_2 \cup \{\lambda\})$ *của* T_1 *thành* T_2 *được gọi là phép biến đổi* hợp lệ *nếu*

- (v_j, λ) *xảy ra trước* (v_i, λ) *trong* E*, với mọi* $(v_i, \lambda), (v_j, \lambda) \in E \cap V_1 \times \{\lambda\}$ *nút s.* v_j *là con cháu của nút* v_i *trong* T_1.
- (λ, w_i) *xảy ra trước* (λ, w_j) *trong* E*, với mọi* $(\lambda, w_i), (\lambda, w_j) \times E \cap \{\lambda\} \times V_2$ *nút s.* w_j *là con cháu của nút* w_i *trong* T_2.
- $E \cap V_1 \times V_2$ *là ánh xạ của* T_1 *tới* T_2.

The elementary edit operations of deletion, substitution, and insertion are, as a matter of fact, sufficient for the transformation of any tree into any other tree.

– Các thao tác chỉnh sửa cơ bản như xóa, thay thế và chèn đã đủ để biến đổi bất kỳ cây nào thành bất kỳ cây nào khác.

Lemma 6 ([Val21], Def. 2.4, p. 48)**.** *Let* $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ *be ordered trees. Then, there is a sequence of elementary edit operations transforming* T_1 *into* T_2.

Proof. Let $[v_i, \ldots, v_j]$ be the set of nodes V_1 in the order in which they are 1st visited during a postorder traversal of T_1, & let $[w_k, \ldots, w_l]$ be the set of nodes V_2 in the order in which they are 1st visited during a postorder traversal of T_2. Then, the sequence of elementary edit operations $[v_i \mapsto \lambda, \ldots, v_j \mapsto \lambda, \lambda \mapsto w_k, \ldots, \lambda \mapsto w_l]$ allows to transform T_1 into T_2.

– Giả sử $[v_i, \ldots, v_j]$ là tập hợp các nút V_1 theo thứ tự mà chúng được truy cập lần đầu tiên trong quá trình duyệt theo thứ tự sau của T_1, & giả sử $[w_k, \ldots, w_l]$ là tập hợp các nút V_2 theo thứ tự mà chúng được truy cập lần đầu tiên trong quá trình duyệt theo thứ tự sau của T_2. Khi đó, trình tự các thao tác chỉnh sửa cơ bản $[v_i \mapsto \lambda, \ldots, v_j \mapsto \lambda, \lambda \mapsto w_k, \ldots, \lambda \mapsto w_l]$ cho phép biến đổi T_1 thành T_2. □

It follows from the proof of Lem. 2.2 that the elementary edit operation of substitution is not necessary for the transformation of a tree into another tree. However, substitutions are convenient because they allow to find shortest or, in general, least-cost transformations between trees.

– Từ chứng minh của Lem. 2.2 suy ra rằng phép toán chỉnh sửa cơ bản của phép thay thế không cần thiết cho việc biến đổi 1 cây thành 1 cây khác. Tuy nhiên, phép thay thế rất tiện lợi vì chúng cho phép tìm các phép biến đổi ngắn nhất hoặc nói chung là ít tốn kém nhất giữa các cây.

Definition 63 (Cost of elementary edit operations, [Val21], Def. 2.5, p. 49)**.** *Let* $T_1 = (V_1, E_1), T_2 = (E_2, V_2)$ *be ordered trees. The* cost *of an elementary edit operation on* T_1 *&* T_2 *is given by a function* $\gamma : V_1 \cup V_2 \cup \{\lambda\} \times V_1 \cup V_2 \cup \{\lambda\} \to \mathbb{R}$ *s.t.* $\forall v, w, z \in V_1 \cup V_2 \cup \{\lambda\}$,

(i) $\gamma(v, w) \geq 0$,

(ii) $\gamma(v, w) = 0 \Leftrightarrow v \neq \lambda \wedge w \neq \lambda$,

(iii) $\gamma(v, w) = \gamma(w, v)$,

(iv) $\gamma(v, w) \leq \gamma(v, z) + \gamma(z, w)$.

Định nghĩa 53 (Chi phí cho các thao tác chỉnh sửa cơ bản)**.** *Cho* $T_1 = (V_1, E_1), T_2 = (E_2, V_2)$ *là các cây có thứ tự.* cost *của 1 thao tác chỉnh sửa cơ bản trên* T_1 *&* T_2 *được đưa ra bởi 1 hàm* $\gamma : V_1 \cup V_2 \cup \{\lambda\} \times V_1 \cup V_2 \cup \{\lambda\} \to \mathbb{R}$ *s.t.* $\forall v, w, z \in V_1 \cup V_2 \cup \{\lambda\}$,

(i) $\gamma(v, w) \geq 0$,

(ii) $\gamma(v, w) = 0 \Leftrightarrow v \neq \lambda \wedge w \neq \lambda$,

(iii) $\gamma(v, w) = \gamma(w, v)$,

(iv) $\gamma(v, w) \leq \gamma(v, z) + \gamma(z, w)$.

The 1st 2 conditions in Def. 2.5 are known as *nonnegative definiteness*, the 3rd condition is the *symmetry* of the cost function, & the 4th condition is known as *triangularity* or *triangular inequality*. Together, these conditions on the cost of elementary edit operations turn ordered trees into a metric space.

– 2 điều kiện đầu tiên trong Định nghĩa 2.5 được gọi là *là xác định không âm*, điều kiện thứ 3 là *là đối xứng* của hàm chi phí, & điều kiện thứ 4 được gọi là *là tính tam giác* hoặc *là bất đẳng thức tam giác*. Cùng nhau, các điều kiện này về chi phí của các hoạt động chỉnh sửa cơ bản biến cây có thứ tự thành không gian metric.

Definition 64 (Cost of transformations, [Val21], Def. 2.6, p. 49)**.** *Let* $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ *be ordered trees. The* cost *of a transformation* $E \subseteq (V_1 \cup \{\lambda\}) \times (V_2 \cup \{\lambda\})$ *of* T_1 *into* T_2 *is given by* $\gamma(E) = \sum_{(v,w) \in E} \gamma(v, w)$.

Định nghĩa 54 (Chi phí của phép biến đổi). *Cho $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ là các cây có thứ tự.* cost *của phép biến đổi $E \subseteq (V_1 \cup \{\lambda\}) \times (V_2 \cup \{\lambda\})$ của T_1 thành T_2 được đưa ra bởi $\gamma(E) = \sum_{(v,w)\in E} \gamma(v, w)$.*

Now, the edit distance between 2 ordered trees is the cost of a least-cost valid transformation between these trees.

– Bây giờ, khoảng cách chỉnh sửa giữa 2 cây có thứ tự là chi phí của phép chuyển đổi hợp lệ có chi phí thấp nhất giữa các cây này.

Definition 65 (Edit distance, [Val21], Def. 2.7, p. 49). *The* edit distance *between ordered trees T_1 & T_2 is $\delta(T_1, T_2) = \min\{\gamma(E); E$ is a valid transformation of T_1 to $T_2\}$.*

Problem 91. *Assume that the cost of elementary edit operations is s.t.*

$$\gamma(v, w) = \begin{cases} 1 & \text{if either } v = \lambda \text{ or } w = \lambda, \\ 0 & \text{otherwise,} \end{cases}$$

i.e., node deletions & insertions have cost equal to 1, & node substitutions have zero cost. Compute the cost of transformations of 2 trees T_1 to T_2 input from file, determine a least-cost valid transformation of T_1 to T_2, & compute the edit distance between T_1 to T_2.

An alternative formulation of the tree edit problem consists of building a kind of grid graph on the nodes of the ordered trees, called the *edit graph* of the trees, & then reducing the problem of finding a valid transformation of 1 tree into the other to that of finding a path in the edit graph from 1 corner down to the opposite corner.

– 1 công thức thay thế cho bài toán chỉnh sửa cây bao gồm việc xây dựng 1 loại đồ thị lưới trên các nút của các cây có thứ tự, được gọi là *edit graph* của các cây, & sau đó giảm bài toán tìm phép biến đổi hợp lệ của 1 cây thành cây kia thành bài toán tìm đường đi trong đồ thị chỉnh sửa từ góc này xuống góc đối diện.

Recall that in the preorder traversal of an ordered tree, the root of the tree is visited 1st, followed by a preorder traversal of the subtree rooted in turn at each of the children of the root, where the subtree rooted at the 1st child is traversed 1st, followed by the subtree rooted at the next sibling, etc.

– Nhớ lại rằng trong phép duyệt tiền thứ tự của 1 cây có thứ tự, gốc của cây được thăm trước, theo sau là phép duyệt tiền thứ tự của cây con có gốc lần lượt tại mỗi cây con của gốc, trong đó cây con có gốc tại cây con thứ nhất được duyệt trước, theo sau là cây con có gốc tại cây anh chị em tiếp theo, v.v.

Definition 66 (Edit graph, [Val21], Def. 2.8, p. 50). *Let $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ be ordered trees. The* edit graph *of T_1 & T_2 has a vertex of the form vw for each pair of nodes $v \in \{v_0\} \cup V_1, w \in \{w_0\} \cup V_2$, where $v_0 \notin V_1, w_0 \notin V_2$ are dummy nodes. Further,*

(i) $(v_i w_j, v_{i+1} w_j) \in E \Leftrightarrow \text{depth}[v_{i+1}] \geq \text{depth}[w_{j+1}]$,

(ii) $(v_i w_j, v_{i+1} w_{j+1}) \in E \Leftrightarrow \text{depth}[v_{i+1}] = \text{depth}[w_{j+1}]$,

(iii) $(v_i w_j, v_i w_{j+1}) \in E \Leftrightarrow \text{depth}[v_{i+1}] \leq \text{depth}[w_{j+1}]$,

for $i \in \overline{0, n_1 - 1}, j \in \overline{0, n_2 - 1}$, where nodes are numbered according to the order in which they are visited during a preorder traversal of the trees. Moreover, $(v_i w_{n_2}, v_{i+1} w_{n_2}) \in E$ for $i \in \overline{0, n_1 - 1}$, & $(v_{n_1} w_j, v_{n_1} w_{j+1}) \in E$ for $j \in \overline{0, n_2 - 1}$.

Định nghĩa 55 (Đồ thị chỉnh sửa). *Cho $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ là các cây có thứ tự.* edit graph *của T_1 & T_2 có đỉnh có dạng vw cho mỗi cặp nút $v \in \{v_0\} \cup V_1, w \in \{w_0\} \cup V_2$, trong đó $v_0 \notin V_1, w_0 \notin V_2$ là các nút giả. Hơn nữa,*

(i) $(v_i w_j, v_{i+1} w_j) \in E \Leftrightarrow \text{depth}[v_{i+1}] \geq \text{depth}[w_{j+1}]$,

(ii) $(v_i w_j, v_{i+1} w_{j+1}) \in E \Leftrightarrow \text{depth}[v_{i+1}]$,

(iii) $(v_i w_j, v_i w_{j+1}) \in E \Leftrightarrow \text{depth}[v_{i+1}] \leq \text{depth}[w_{j+1}]$,

đối với $i \in \overline{0, n_1 - 1}, j \in \overline{0, n_2 - 1}$, trong đó các nút là được đánh số theo thứ tự mà chúng được truy cập trong quá trình duyệt cây theo thứ tự trước. Hơn nữa, $(v_i w_{n_2}, v_{i+1} w_{n_2}) \in E$ cho $i \in \overline{0, n_1 - 1}$, & $(v_{n_1} w_j, v_{n_1} w_{j+1}) \in E$ cho $j \in \overline{0, n_2 - 1}$.

In the edit graph of ordered trees T_1, T_2, a *vertical* edge of the form $(v_i w_j, v_{i+1} w_j)$ represents the deletion of node v_{i+1} from T_1, where the condition $\text{depth}[v_{i+1}] \geq \text{depth}[w_{j+1}]$ ensures that node w_{j+1} does not belong to the subtree of T_2 rooted at node w_j, i.e., node w_{j+1} does not have to be inserted into T_2 before node v_{i+1} can be deleted from T_1. Further, a *diagonal* edge of the form $(v_i w_j, v_{i+1} w_{j+1})$ represents the substitution or replacement of node v_{i+1} of T_1 with node w_{j+1} of T_2, where the condition $\text{depth}[v_{i+1}] = \text{depth}[w_{j+1}]$ follows from Lem. 2.1. A *horizontal* edge, on the other hand, of the form $(v_i w_j, v_i w_{j+1})$ represents the insertion of node w_{j+1} into T_2, where the condition $\text{depth}[v_{i+1}] \leq \text{depth}[w_{j+1}]$ ensures that node v_{i+1} does not belong to the subtree of T_1 rooted at node v_i, i.e., node v_{i+1} does not have to be deleted from T_1 before node w_{j+1} can be inserted into T_2.

– Trong đồ thị chỉnh sửa của cây có thứ tự T_1, T_2, 1 cạnh *dọc* có dạng $(v_i w_j, v_{i+1} w_j)$ biểu diễn việc xóa nút v_{i+1} khỏi T_1, trong đó điều kiện $\text{depth}[v_{i+1}] \geq \text{depth}[w_{j+1}]$ đảm bảo rằng nút w_{j+1} không thuộc về cây con của T_2 có gốc tại nút w_j, tức là, nút w_{j+1} không cần phải được chèn vào T_2 trước khi có thể xóa nút v_{i+1} khỏi T_1. Hơn nữa, 1 cạnh *đường chéo* có dạng $(v_i w_j, v_{i+1} w_{j+1})$ biểu diễn sự thay thế hoặc thay thế nút v_{i+1} của T_1 bằng nút w_{j+1} của T_2, trong đó điều kiện $\text{Øsu}[v_{i+1}] = \text{Øsu}[w_{j+1}]$ tuân

theo Bổ đề 2.1. Mặt khác, 1 cạnh *nằm ngang* có dạng $(v_i w_j, v_i w_{j+1})$ biểu diễn việc chèn nút w_{j+1} vào T_2, trong đó điều kiện $\text{depth}[v_{i+1}] \leq \text{depth}[w_{j+1}]$ đảm bảo rằng nút v_{i+1} không thuộc về cây con của T_1 có gốc tại nút v_i, tức là, nút v_{i+1} không cần phải bị xóa khỏi T_1 trước khi nút w_{j+1} có thể được chèn vào T_2.

Missing horizontal & diagonal edges ensure thus that once a path in the edit graph traverses a vertical edge, corresponding to the deletion of a certain hole, that path can only be extended by traversing further vertical edges, corresponding to the deletion of all the nodes in the subtree rooted at that node. Conversely, missing vertical & diagonal edges ensure that once a path in the edit graph traverses a horizontal edge, corresponding to the insertion of a certain node, that path can only be extended by traversing further horizontal edge, corresponding to the insertion of all the nodes in the subtree rooted at that node.

– Thiếu các cạnh ngang & chéo đảm bảo rằng khi 1 đường dẫn trong đồ thị chỉnh sửa đi qua 1 cạnh dọc, tương ứng với việc xóa 1 lỗ nhất định, đường dẫn đó chỉ có thể được mở rộng bằng cách đi qua các cạnh dọc tiếp theo, tương ứng với việc xóa tất cả các nút trong cây con có gốc tại nút đó. Ngược lại, thiếu các cạnh dọc & chéo đảm bảo rằng khi 1 đường dẫn trong đồ thị chỉnh sửa đi qua 1 cạnh ngang, tương ứng với việc chèn 1 nút nhất định, đường dẫn đó chỉ có thể được mở rộng bằng cách đi qua các cạnh ngang tiếp theo, tương ứng với việc chèn tất cả các nút trong cây con có gốc tại nút đó.

The correspondence of valid transformations between ordered trees T_1, T_2 & paths in the edit graph of T_1, T_2 from the top-left to the bottom-right corner is given by the fact that node substitutions (diagonal edges) along a path in the edit graph of T_1, T_2 from the top-left to the bottom-right corner constitute a mapping of T_1 to T_2, which can thus be extended to a valid transformation of T_1 to T_2.

– Sự tương ứng của các phép biến đổi hợp lệ giữa các cây có thứ tự T_1, T_2 & các đường dẫn trong đồ thị chỉnh sửa của T_1, T_2 từ góc trên bên trái xuống góc dưới bên phải được đưa ra bởi thực tế là các phép thay thế nút (các cạnh chéo) dọc theo 1 đường dẫn trong đồ thị chỉnh sửa của T_1, T_2 từ góc trên bên trái xuống góc dưới bên phải tạo nên 1 ánh xạ từ T_1 tới T_2, do đó có thể mở rộng thành 1 phép biến đổi hợp lệ từ T_1 tới T_2.

Lemma 7 ([Val21], Lem. 2.3, p. 51)**.** *Let P be a path in the edit graph of ordered trees $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ from the top-left to the bottom-right corner, & let $M = \{(v_{i+1}, w_{j+1}) \in V_1 \times V_2; (v_i w_j, v_{i+1} w_{j+1}) \in P\}$. Then, M is a mapping of T_1 to T_2. Conversely, let $M \subseteq V_1 \times V_2$ be a mapping of T_1 to T_2. Then, there is a path P in the edit graph of T_1, T_2 from the top-left to the bottom-right corner s.t. $\{(v_{i+1}, w_{j+1}) \in V_1 \times V_2; (v_i w_j, v_{i+1} w_{j+1}) \in P\} = M$.*

Remark 16 ([Val21], Rmk. 2.2, p. 52)**.** *A valid transformation of T_1 into T_2 can be obtained from a path in the edit graph of T_1, T_2 from the top-left to the bottom-right corner, by just reordering the edit operations between each pair of successive node substitutions according to the preorder traversal of the trees. I.e., in such a way that deletion & insertion of children nodes do not precede deletion & insertion of their parent or their left sibling nodes.*

– Có thể lấy được phép biến đổi hợp lệ của T_1 thành T_2 từ đường dẫn trong đồ thị chỉnh sửa của T_1, T_2 từ góc trên bên trái xuống góc dưới bên phải, chỉ bằng cách sắp xếp lại các hoạt động chỉnh sửa giữa mỗi cặp thay thế nút liên tiếp theo phép duyệt trước của cây. Tức là theo cách mà việc xóa & chèn các nút con không diễn ra trước việc xóa & chèn các nút cha hoặc các nút anh chị em bên trái của chúng.

Now, computing the edit distance between 2 ordered trees can be reduced to the problem of finding the shortest path in the edit graph of the trees, with edges in the edit graph labeled or weighted by the cost of the respective tree edit operation.

– Bây giờ, việc tính toán khoảng cách chỉnh sửa giữa 2 cây có thứ tự có thể được giảm xuống thành bài toán tìm đường đi ngắn nhất trong đồ thị chỉnh sửa của các cây, với các cạnh trong đồ thị chỉnh sửa được gắn nhãn hoặc được tính trọng số theo chi phí của thao tác chỉnh sửa cây tương ứng.

Lemma 8 ([Val21], Lem. 2.4, p. 52)**.** *Let G be the edit graph of ordered trees $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$, with edges of the form $(v_i w_j, v_k w_l) \in E$ weighted by $\gamma(v_k, \lambda)$ if $k = i+1, l = j$, weighted by $\gamma(v_k, w_l)$ if $k = i+1, l = j+1$, & weighted by $\gamma(\lambda, w_l)$ if $k = i, l = j+1$. Let also P be a shortest path in G from the top-left to the bottom-right corner, & let $E = \{(v_{i+1}, \lambda) \in V_1 \times \{\lambda\}; (v_i w_j, v_{i+1} w_j) \in P\} \cup \{(v_{i+1}, w_{j+1}) \in V_1 \times V_2; (v_i w_j, v_{i+1} w_{j+1}) \in P\} \cup \{(\lambda, w_{j+1}) \in \{\lambda\} \times V_2; (v_i w_j, v_i w_{j+1}) \in P\}$. Then, $\delta(T_1, T_2) = \sum_{(v,w) \in E} \gamma(v, w)$.*

Bổ đề 1. *Cho G là đồ thị chỉnh sửa của cây có thứ tự $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$, với các cạnh có dạng $(v_i w_j, v_k w_l) \in E$ được trọng số bởi $\gamma(v_k, \lambda)$ nếu $k = i+1, l = j$, được trọng số bởi $\gamma(v_k, w_l)$ nếu $k = i+1, l = j+1$, & được trọng số bởi $\gamma(\lambda, w_l)$ nếu $k = i, l = j+1$. Giả sử P cũng là đường đi ngắn nhất trong G từ góc trên bên trái đến góc dưới bên phải, & cho $E = \{(v_{i+1}, \lambda) \in V_1 \times \{\lambda\}; (v_i w_j, v_{i+1} w_j) \in P\} \cup \{(v_{i+1}, w_{j+1}) \in V_1 \times V_2; (v_i w_j, v_{i+1} w_{j+1}) \in P\} \cup \{(\lambda, w_{j+1}) \in \{\lambda\} \times V_2; (v_i w_j, v_i w_{j+1}) \in P\}$. Khi đó, $\delta(T_1, T_2) = \sum_{(v,w) \in E} \gamma(v, w)$.*

The following function builds the edit graph G of ordered trees T_1, T_2. The vertices of the edit graph are labeled by the preorder number of the nodes of T_1, T_2 they correspond to, & the edges are labeled by a string indicating the tree edit operation they correspond to: "del" for deletion of a node from T_2, "sub" for substitution or replacement of a node of T_1 with a node of T_2, & "ins" for insertion of a node into T_2.

```
1  function tree_edit_graph(T1, T2)
2      let G be an empty graph
```

```
    let n1, n2 be the number of nodes of T1, T2
    preorder_tree_traversal(T1)
    preorder_tree_traversal(T2)
    preorder_tree_depth(T1)
    preorder_tree_depth(T2)
    let d1[1..n1], d2[1..n2] be new arrays (of integers)
    for all nodes v of T1 do
        d1[v.order] <-- v.depth
    for all nodes w of T2 do
        d2[w.order] <-- w.depth
    let A[0..n1][0..n2] be a new matrix (of nodes)
    for i = 0 to n1 do
        for j = 0 to n2 do
            add a new vertex A[i, j] to G labeled [i, j]
    for i = 0 to n1 - 1 do
        add a new edge to G from A[i, n2] to A[i + 1, n2] labeled ``del''
    for j = 0 to n2 - 1 do
        add a new edge to G from A[n1, j] to A[n1, j + 1] labeled ``ins''
    for i = 0 to n1 - 1 do
        for j = 0 to n2 - 1 do
            if d1[i + 1] >= d2[j + 1] then
                add a new edge to G from A[i, j] to A[i + 1, j] labeled ``del''
            if d1[i + 1] = d2[j + 1] then
                add a new edge from A[i, j] to A[i + 1, j + 1] labeled ``sub''
            if d1[i + 1] <= d2[j + 1] then
                add a new edge to G from A[i, j] to A[i, j + 1] labeled ``ins''
    return G
```

The cost of elementary tree edit operations can be set to $\gamma(v, w) = 1$, for all edit operations of the form $v \mapsto w$ with $v \in V_1 \cup \{\lambda\}, w \in V_2 \cup \{\lambda\}$, by defining the weight or cost of the respective edges in the edit graph of the trees. However, since all of the edges in the edit graph correspond to these elementary tree edit operations, it suffices to find the shortest path in the edit graph without edge weights, by means of breadth-1st traversal.

– Chi phí của các hoạt động chỉnh sửa cây cơ bản có thể được đặt thành $\gamma(v, w) = 1$, cho tất cả các hoạt động chỉnh sửa có dạng $v \mapsto w$ với $v \in V_1 \cup \{\lambda\}, w \in V_2 \cup \{\lambda\}$, bằng cách xác định trọng số hoặc chi phí của các cạnh tương ứng trong đồ thị chỉnh sửa của các cây. Tuy nhiên, vì tất cả các cạnh trong đồ thị chỉnh sửa tương ứng với các hoạt động chỉnh sửa cây cơ bản này, nên chỉ cần tìm đường đi ngắn nhất trong đồ thị chỉnh sửa mà không có trọng số cạnh, bằng cách duyệt theo chiều rộng thứ nhất.

The actual computation of the shortest path P in the edit graph G of ordered trees T_1, T_2 is done by the following function. The shortest path P is recovered from the predecessor edges *pred* computed by the acyclic shortest path function, starting off with the last vertex of G (the bottom-right corner of the edit graph) & following up the thread of predecessor edges until reaching the 1st vertex of G (the top-left corner of the edit graph), which has no predecessor edge. The shortest path P is represented by a list of edges (of graph G).

– Tính toán thực tế của đường đi ngắn nhất P trong đồ thị chỉnh sửa G của các cây có thứ tự T_1, T_2 được thực hiện bằng hàm sau. Đường đi ngắn nhất P được phục hồi từ các cạnh tiền nhiệm *pred* được tính bằng hàm đường đi ngắn nhất không có chu trình, bắt đầu bằng đỉnh cuối cùng của G (góc dưới bên phải của đồ thị chỉnh sửa) & theo dõi luồng các cạnh tiền nhiệm cho đến khi đạt đến đỉnh thứ nhất của G (góc trên bên trái của đồ thị chỉnh sửa), không có cạnh tiền nhiệm. Đường đi ngắn nhất P được biểu diễn bằng danh sách các cạnh (của đồ thị G).

```
function tree_edit(T1, T2, P)
    G <-- tree_edit_graph(T1, T2)
    let s, t be the 1st vertex & the last vertex of G
    (W, S) <-- breath_1st_spanning_subtree(G, s)
    return acyclic_shortest_path(G, s, t, W, S)
```

Lemma 9 ([Val21], Lem. 2.5, p. 54)**.** *The tree edit algorithm based on shortest paths in the edit graph for finding a least-cost transformation of an ordered tree T_1 to an ordered tree T_2 with, resp., n_1, n_2 nodes runs in $O(n_1 n_2)$ time using $O(n_1 n_2)$ additional space.*

Bổ đề 2. *Thuật toán chỉnh sửa cây dựa trên đường dẫn ngắn nhất trong đồ thị chỉnh sửa để tìm phép biến đổi có chi phí thấp nhất từ cây có thứ tự T_1 sang cây có thứ tự T_2 với tương ứng n_1, n_2 nút chạy trong thời gian $O(n_1 n_2)$ bằng cách sử dụng không gian bổ sung $O(n_1 n_2)$.*

Proof. Let $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ be ordered trees with, resp., n_1, n_2 nodes. The edit graph of T_1, T_2 has $(n_1 + 1)(n_2 + 1)$ vertices & at most $3n_1n_2$ edges, & building the edit graph takes thus $O(n_1n_2)$ time & uses $O(n_1n_2)$ additional space. Further, the breadth-1st spanning tree procedure & the acyclic shortest path produce also take $O(n_1n_2)$ time. Therefore, the tree edit algorithm runs in $O(n_1n_2)$ time using $O(n_1n_2)$ additional space.

– Cho $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ là cây có thứ tự với, tương ứng, n_1, n_2 nút. Đồ thị chỉnh sửa của T_1, T_2 có $(n_1+1)(n_2+1)$ đỉnh & nhiều nhất là $3n_1n_2$ cạnh, & việc xây dựng đồ thị chỉnh sửa do đó mất thời gian $O(n_1n_2)$ & sử dụng $O(n_1n_2)$ không gian bổ sung. Hơn nữa, quy trình cây bao trùm thứ nhất theo chiều rộng & đường dẫn ngắn nhất phi chu trình cũng mất thời gian $O(n_1n_2)$. Do đó, thuật toán chỉnh sửa cây chạy trong thời gian $O(n_1n_2)$ bằng cách sử dụng $O(n_1n_2)$ không gian bổ sung. □

The representation of the trees is assumed to be arranged in such a way that the order of the nodes fixed by the representation of each of the trees coincides with the order in which they are visited during a preorder traversal of the tree, as required by the formulation of the tree edit problem. Further, for any trees T_1, T_2, the preorder number of all nodes in the trees is assumed to be stored in the attributes $v.order, w.order$, for all nodes v of T_1 & w of T_2.

– Biểu diễn của các cây được cho là được sắp xếp theo cách mà thứ tự của các nút được cố định bởi biểu diễn của từng cây trùng với thứ tự mà chúng được truy cập trong quá trình duyệt trước thứ tự của cây, như yêu cầu của việc xây dựng bài toán chỉnh sửa cây. Hơn nữa, đối với bất kỳ cây nào T_1, T_2, số thứ tự trước của tất cả các nút trong các cây được cho là được lưu trữ trong các thuộc tính $v.order, w.order$, đối với tất cả các nút v của T_1 & w của T_2.

7.3 Backtracking – Quay lui

The solution to combinatorial problems on trees & graphs often requires an exhaustive search of the set of all possible solutions. As a matter of fact, most combinatorial problems on trees & graphs can be stated in the general framework of assigning values of a finite domain to each of a series of variables, where the value for each variable must be taken from a corresponding finite domain. An ordered set of values satisfying some specified constraints represents a solution to the problem, & an exhaustive search for a solution must consider the elements of the cartesian product of variable domains as potential solutions.

– Giải pháp cho các bài toán tổ hợp trên cây & đồ thị thường đòi hỏi phải tìm kiếm toàn diện tập hợp tất cả các giải pháp khả thi. Trên thực tế, hầu hết các bài toán tổ hợp trên cây & đồ thị có thể được nêu trong khuôn khổ chung của việc gán các giá trị của 1 miền hữu hạn cho mỗi biến trong 1 loạt các biến, trong đó giá trị cho mỗi biến phải được lấy từ 1 miền hữu hạn tương ứng. Một tập hợp các giá trị có thứ tự thỏa mãn 1 số ràng buộc cụ thể biểu diễn 1 giải pháp cho bài toán, & tìm kiếm toàn diện cho 1 giải pháp phải xem xét các phần tử của tích Descartes của các miền biến như các giải pháp tiềm năng.

Consider, e.g., the tree edit problem. Let $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ be ordered trees. A potential solution to the problem of finding a transformation of T_1 into T_2 is given by a bijection $M \subseteq W_1 \times W_2$, where $W_1 \subseteq V_1, W_2 \subseteq V_2$, which can also be seen as an assignment of a node $w \in V_2 \cup \{\lambda\}$ to each node $v \in V_1$. The dummy node λ represents the deletion of a node from T_1, & the insertion of a node into T_2 remains implicit, i.e., all nodes $w \in V_2 \backslash W_2$ are inserted into T_2.

– Xem xét, e.g., bài toán chỉnh sửa cây. Giả sử $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ là các cây có thứ tự. Một giải pháp tiềm năng cho bài toán tìm phép biến đổi T_1 thành T_2 được đưa ra bởi một song ánh $M \subseteq W_1 \times W_2$, trong đó $W_1 \subseteq V_1, W_2 \subseteq V_2$, cũng có thể được xem như là phép gán một nút $w \in V_2 \cup \{\lambda\}$ cho mỗi nút $v \in V_1$. Nút giả λ biểu diễn việc xóa một nút khỏi T_1, & việc chèn một nút vào T_2 vẫn ngầm định, tức là, tất cả các nút $w \in V_2 \backslash W_2$ đều được chèn vào T_2.

However, not every assignment of a node $w \in V_2 \backslash \{\lambda\}$ to each node $v \in V_1$ corresponds to a valid transformation of T_1 into T_2. The bijection $M \subseteq W_1 \times W_2 \subseteq V_1 \times V_2$ must be a mapping of T_1 to T_2, according to Def. 2.4. Notice that in such a case, a valid transformation of T_1 into T_2 can be obtained be 1st deleting from T_1 all those nodes of T_1 which were assigned to the dummy node λ, in the order given by a preorder traversal of T_1; substituting or replacing all nodes of T_1 which were assigned to a non-dummy node of T_2 with the node they were assigned to; & inserting into T_2 all those nodes of T_2 which were not assigned to any node of T_1, in the order given by a preorder traversal of T_2.

– Tuy nhiên, không phải mọi phép gán một nút $w \in V_2 \backslash \{\lambda\}$ cho mỗi nút $v \in V_1$ đều tương ứng với một phép biến đổi hợp lệ của T_1 thành T_2. Song ánh $M \subseteq W_1 \times W_2 \subseteq V_1 \times V_2$ phải là một phép ánh xạ của T_1 thành T_2, theo Định nghĩa 2.4. Lưu ý rằng trong trường hợp như vậy, có thể thu được một phép biến đổi hợp lệ của T_1 thành T_2 bằng cách đầu tiên xóa khỏi T_1 tất cả các nút của T_1 đã được gán cho nút giả λ, theo thứ tự được chỉ ra bởi phép duyệt trước của T_1; thay thế hoặc thay thế tất cả các nút của T_1 đã được gán cho một nút không phải giả của T_2 bằng nút mà chúng đã được gán; & chèn vào T_2 tất cả các nút của T_2 không được gán cho bất kỳ nút nào của T_1, theo thứ tự được chỉ định bởi phép duyệt trước của T_2.

Backtracking is a general technique for organizing the exhaustive search for a solution to a combinatorial problem. Roughly, the technique consists of repeatedly extending a partial solution to the problem – represented as an assignment of values of a finite domain, satisfying certain constraints, to an ordered finite set of variables – to a complete solution to the problem, by extending the representation of the partial solution 1 variable at a time & shrinking the representation of the partial solution (backtracking) whenever a partial solution cannot be further extended.

– Quay lui là một kỹ thuật chung để tổ chức tìm kiếm toàn diện cho một giải pháp cho một bài toán tổ hợp. Về cơ bản, kỹ thuật này bao gồm việc mở rộng liên tục một giải pháp cục bộ cho bài toán – được biểu diễn dưới dạng phép gán các giá trị của một miền hữu hạn, thỏa mãn một số ràng buộc nhất định, thành một tập hợp hữu hạn các biến có thứ tự – thành một giải

pháp hoàn chỉnh cho bài toán, bằng cách mở rộng biểu diễn của giải pháp cục bộ 1 biến tại một thời điểm & thu hẹp biểu diễn của giải pháp cục bộ (quay lui) bất cứ khi nào một giải pháp cục bộ không thể mở rộng thêm.

The backtracking technique can be applied to those problems that exhibit the *domino principle*, i.e., if a constraint is not satisfied by any extension of the partial solution at all. The domino principle allows to stop extending a partial solution as soon as it can be established that the extension will not lead to a solution to the problem, because the partial solution already violates some constraint.

– Kỹ thuật quay lui có thể được áp dụng cho những bài toán thể hiện *nguyên lý domino*, tức là nếu một ràng buộc không được thỏa mãn bởi bất kỳ phần mở rộng nào của giải pháp một phần. Nguyên lý domino cho phép dừng việc mở rộng một giải pháp một phần ngay khi có thể xác định rằng phần mở rộng sẽ không dẫn đến giải pháp cho bài toán, vì giải pháp một phần đã vi phạm một số ràng buộc.

The procedure of extending a partial solution toward a complete solution to the problem & shrinking a partial solution that cannot be further extended can be better understood in terms of a *backtracking tree* of possible assignments of values to the variables that represent the problem. The root of the backtracking tree is a dummy node, the nodes at level 1 represent the possible values which the 2nd variable can be assigned to, given the value which the 1st variable was assigned to, the nodes at level 3 represent the possible values which the 3rd variable can be assigned to, given the values which the 1st & 2nd variables were assigned to, & so on.

– Quy trình mở rộng một giải pháp một phần thành một giải pháp hoàn chỉnh cho vấn đề & thu hẹp một giải pháp một phần không thể mở rộng thêm nữa có thể được hiểu rõ hơn theo thuật ngữ của một *cây quay lui* của các phép gán giá trị có thể cho các biến biểu diễn vấn đề. Gốc của cây quay lui là một nút giả, các nút ở cấp độ 1 biểu diễn các giá trị có thể mà biến thứ 2 có thể được gán cho, cho giá trị mà biến thứ 1 được gán cho, các nút ở cấp độ 3 biểu diễn các giá trị có thể mà biến thứ 3 có thể được gán cho, cho các giá trị mà biến thứ 1 & biến thứ 2 được gán cho, & cứ như vậy.

The domino principle allows *pruning* the backtracking tree at those nodes representing partial solutions that violate some constraint of the problem.

– Nguyên lý domino cho phép cắt tỉa cây quay lui tại các nút biểu diễn các giải pháp một phần vi phạm một số ràng buộc của bài toán.

Finding a solution to a tree edit problem by backtracking means finding a node in the backtracking tree for which all the constraints of the problem are satisfied, & finding all solutions to the tree edit problem means finding all such nodes in the backtracking tree. These solution nodes will be leaves in the backtracking tree for a tree edit problem, because of the problem formulation, but this is not always the case.

– Tìm giải pháp cho bài toán chỉnh sửa cây bằng cách quay lui có nghĩa là tìm một nút trong cây quay lui mà tất cả các ràng buộc của bài toán đều được thỏa mãn, & tìm tất cả các giải pháp cho bài toán chỉnh sửa cây có nghĩa là tìm tất cả các nút như vậy trong cây quay lui. Các nút giải pháp này sẽ là các nút lá trong cây quay lui cho bài toán chỉnh sửa cây, do cách xây dựng bài toán, nhưng không phải lúc nào cũng vậy.

The backtracking tree for a combinatorial problem is not explicit but remains implicit in the formulation of the problem. Further, the size of the backtracking tree is often exponential in the number of variables & possible values. Therefore, it is crucial for a backtracking procedure to build only that portion of the backtracking tree that is needed at each stage, during the search for 1 or more solutions to the problem.

– Cây quay lui cho một bài toán tổ hợp không phải là rõ ràng nhưng vẫn ngầm định trong việc xây dựng bài toán. Hơn nữa, kích thước của cây quay lui thường theo cấp số nhân trong số các biến & các giá trị có thể. Do đó, điều quan trọng đối với một quy trình quay lui là chỉ xây dựng phần cây quay lui cần thiết ở mỗi giai đoạn, trong quá trình tìm kiếm 1 hoặc nhiều giải pháp cho bài toán.

The following backtracking algorithm for enumerating node assignments that correspond to valid tree transformations extends an (initially empty) mapping M in all possible ways, in order to enumerate all assignments $M \subseteq V_1 \times V_2 \cup \{\lambda\}$ of nodes of an ordered tree $T_2 = (V_2, E_2)$, including a dummy node λ, to the nodes of an ordered tree $T_1 = (V_1, E_1)$. The node assignments found are collected in a list L of dictionaries of nodes (of tree T_1) to nodes (of tree T_2).

– Thuật toán quay lui sau đây để liệt kê các phép gán nút tương ứng với các phép biến đổi cây hợp lệ mở rộng một ánh xạ (ban đầu là rỗng) M theo mọi cách có thể, để liệt kê tất cả các phép gán $M \subseteq V_1 \times V_2 \cup \{\lambda\}$ của các nút của một cây có thứ tự $T_2 = (V_2, E_2)$, bao gồm một nút giả λ, tới các nút của một cây có thứ tự $T_1 = (V_1, E_1)$. Các phép gán nút được tìm thấy được thu thập trong một danh sách L các từ điển của các nút (của cây T_1) tới các nút (của cây T_2).

Further, the preorder number *order* & depth *depth* of the nodes in the trees will be used for testing the constraints at each stage, & the candidate nodes which each node can be mapped to are represented by a dictionary C of nodes (of tree T_1) to lists of nodes (of tree T_2).

– Hơn nữa, số thứ tự trước *order* & độ sâu *depth* của các nút trong cây sẽ được sử dụng để kiểm tra các ràng buộc ở mỗi giai đoạn, & các nút ứng viên mà mỗi nút có thể được ánh xạ tới được biểu diễn bằng một từ điển C gồm các nút (của cây T_1) tới danh sách các nút (của cây T_2).

```
1 function backtracking_tree_edit(T1, T2)
2     preordered_tree_traversal(T1)
3     preordered_tree_traversal(T2)
```

```
    let M be an empty dictionary (of nodes to nodes)
    let L be an empty list (of dictionaries of nodes to nodes)
    C <-- set_up_candidate_nodes(T1, T2)
    let v be the 1st node of T1 in preorder
    (M, L) <-- extend_tree_edit(T1, T2, M, L, C, v)
    return L
```

The candidate nodes which a node of T_1 can be mapped to are just those nodes of T_2 of same depth as the node of T_1, together with the dummy node λ representing the deletion of the node from T_1. I.e., $C[v] = \{w \in V_2; \text{depth}[v] = \text{depth}[w]\} \cup \{\lambda\}$.

```
function set_up_candidate_nodes(T1, T2)
    preorder_tree_depth(T1)
    preorder_tree_depth(T2)
    add a new node T2.dummy to T2
    let C be an empty dictionary (of nodes to lists of nodes)
    for all nodes v of T1 do
        let C[v] be an empty list (of nodes)
        append T2.dummy to C[v]
        for all nodes w of T2 do
            if w != T2.dummy & v.depth = w.depth then
                append w to C[v]
    return C
```

p. 59+++

7.4 Branch-&-Bound – Phân Nhánh & Giới hạn

The backtracking technique can be used for finding either 1 solution or all solutions to a combinatorial problem. When a cost can be associated with each partial solution, a least-cost solution can be found in a more efficient way by a simple variation of backtracking, called *branch-&-bound*. The technique consists of remembering the lowest-cost solution found at each stage of the backtracking search for a solution & using the cost of the lowest-cost solution found so far as a lower bound on the cost of a least-cost solution to the problem, in order to discard partial solution as soon as it can be established that they will not improve on the lowest-cost solution found so far.

– Kỹ thuật quay lui có thể được sử dụng để tìm 1 giải pháp hoặc tất cả các giải pháp cho một bài toán tổ hợp. Khi có thể liên kết chi phí với mỗi giải pháp một phần, giải pháp có chi phí thấp nhất có thể được tìm thấy theo cách hiệu quả hơn bằng một biến thể đơn giản của quay lui, được gọi là *nhánh-&-bound*. Kỹ thuật này bao gồm việc ghi nhớ giải pháp có chi phí thấp nhất được tìm thấy ở mỗi giai đoạn của quá trình tìm kiếm quay lui cho một giải pháp & sử dụng chi phí của giải pháp có chi phí thấp nhất được tìm thấy cho đến nay làm giới hạn dưới của chi phí của giải pháp có chi phí thấp nhất cho bài toán, để loại bỏ giải pháp một phần ngay khi có thể xác định rằng chúng sẽ không cải thiện giải pháp có chi phí thấp nhất được tìm thấy cho đến nay.

The branch-&-bound technique can be applied to those problems that exhibit an extension of the *domino principle*, meaning not only that if a constraint is not satisfied by a given partial solution, the constraint will not be satisfied by any extension of the partial solution at all – as in the case of backtracking – but also that the cost of any extension of a partial solution will be $\geq$ the cost of the partial solution itself. Such an extended domino principle allows to stop extending a partial solution as soon as it can be established that the extension will not lead to a solution to the problem, because the partial solution already violates some constraint, & also as soon as it can be established that the extension will not lead to a least-cost solution to the problem, because the cost of the partial solution already reaches or exceeds the cost of the lowest-cost solution found so far.

– Kỹ thuật nhánh-&-bound có thể được áp dụng cho những bài toán thể hiện sự mở rộng của *nguyên lý domino*, nghĩa là không chỉ nếu một ràng buộc không được thỏa mãn bởi một giải pháp một phần nhất định, ràng buộc đó sẽ không được thỏa mãn bởi bất kỳ sự mở rộng nào của giải pháp một phần – như trong trường hợp quay lui – mà còn có nghĩa là chi phí của bất kỳ sự mở rộng nào của một giải pháp một phần sẽ là $\geq$ chi phí của chính giải pháp một phần đó. Một nguyên lý domino mở rộng như vậy cho phép dừng việc mở rộng một giải pháp một phần ngay khi có thể xác định rằng việc mở rộng sẽ không dẫn đến một giải pháp cho bài toán, vì giải pháp một phần đã vi phạm một số ràng buộc, & cũng ngay khi có thể xác định rằng việc mở rộng sẽ không dẫn đến một giải pháp có chi phí thấp nhất cho bài toán, vì chi phí của giải pháp một phần đã đạt hoặc vượt quá chi phí của giải pháp có chi phí thấp nhất đã tìm thấy cho đến nay.

As in the case of backtracking, the procedure of extending a partial solution toward a least-cost solution to the problem & shrinking a partial solution that cannot be further extended to a solution of lesser cost can be better understood in terms of a *branch-&-bound tree* of possible assignments of values to the variables that represent the problem. The root of the branch-&-bound tree is a dummy node of zero cost, the nodes at level 1 represent the possible values which the 1st variable can be assigned to, the nodes at level 2 represent the possible values which the 2nd variable can be assigned to, given the value which the 1st

variable was assigned to, the nodes at level 3 represent the possible values which the 3rd variable can be assigned to, given the values which the 1st & 2nd variables were assigned to, & so on. Further, $\text{cost}[v] \geq \text{cost}[w]$ for all leaves $v \in V_1$ & all successor nodes w of node v in a preorder traversal of the branch-&-bound tree. I.e., subtrees in the backtracking tree rooted at nodes of cost greater than the cost of a previous leaf node are pruned off the branch-&-bound tree.

– Giống như trong trường hợp quay lui, quy trình mở rộng một giải pháp một phần thành một giải pháp có chi phí thấp nhất cho vấn đề & thu hẹp một giải pháp một phần không thể mở rộng thêm thành một giải pháp có chi phí thấp hơn có thể được hiểu rõ hơn theo thuật ngữ của một *nhánh-&-bound tree* của các phép gán giá trị có thể có cho các biến biểu diễn vấn đề. Gốc của cây nhánh-&-bound tree là một nút giả có chi phí bằng không, các nút ở cấp độ 1 biểu diễn các giá trị có thể có mà biến thứ 1 có thể được gán cho, các nút ở cấp độ 2 biểu diễn các giá trị có thể có mà biến thứ 2 có thể được gán cho, với giá trị mà biến thứ 1 được gán cho, các nút ở cấp độ 3 biểu diễn các giá trị có thể có mà biến thứ 3 có thể được gán cho, với giá trị mà biến thứ 1 & thứ 2 được gán cho, & cứ như vậy. Hơn nữa, $\text{cost}[v] \geq \text{cost}[w]$ cho tất cả các lá $v \in V_1$ & tất cả các nút kế thừa w của nút v trong một phép duyệt trước của cây liên kết nhánh. Tức là, các cây con trong cây quay lui có gốc tại các nút có chi phí lớn hơn chi phí của một nút lá trước đó sẽ bị cắt khỏi cây liên kết nhánh.

p. 62+++

7.5 Basic Graph Theory: Coding & Programming Aspects – Lý Thuyết Đồ Thị Cơ Bản: Các Khía Cạnh Mã Hóa & Lập Trình

Resources – Tài nguyên.

1. VNTA's basic Graph Theory – coding: https://github.com/vntanh1406/Graph_SUM2025/tree/main/BasicGraphTheory.

7.5.1 Graph representation – Biểu diễn đồ thị

Bài toán 45 (Graph representation – Biểu diễn đồ thị)**.** *Cho đồ thị đơn vô hướng hữu hạn $G = (V, E)$. Có 3 dạng biểu diễn đồ thị chính: adjacency list (abbr., al) – danh sách kề, adjacency matrix (abbr., am) – ma trận kề, edge list (abbr., el) – danh sách cạnh. Viết $C_3^2 = 6$ thuật toán & chương trình* C/C++, Pascal, Python *chuyển đổi (converter programs among graph representations) giữa các dạng biểu diễn này.*

Bài toán 46 (Adjacency list $\mapsto$ adjacency matrix converter – chuyển đổi danh sách kề $\mapsto$ ma trận kề)**.** *Cho đồ thị đơn vô hướng hữu hạn $G = (V, E)$. Viết thuật toán & chương trình* C/C++, Pascal, Python *để chuyển đổi dạng biểu diễn của đồ thị G từ dạng danh sách kề sang dạng ma trận kề.*

Solution. C++:

1. VNTA's C++: adjacency list $\mapsto$ adjacency matrix converter: URL:

- https://github.com/vntanh1406/Graph_SUM2025/blob/main/BasicGraphTheory/AdjacencyListToAdjacencyMatrix.cpp.
- https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/C%2B%2B/VNTA_adjacency_list_to_adjacenc matrix.cpp.

```
#include <bits/stdc++.h>
using namespace std;
/*
Cho đồ thị vô hướng G = <V,E> được biểu diễn dưới dạng danh sách kề.
Chuyển đổi biểu diễn đồ thị dưới dạng ma trận kề
*/
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    int n;
    cin >> n;
    cin.ignore();
    int a[n + 1][n + 1];

    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= n; ++j) a[i][j] = 0;

    for (int i = 1; i <= n; ++i) {
```

```
        string s, num;
        getline(cin, s);
        stringstream ss(s);
        while (ss >> num) a[i][stoi(num)] = 1;
    }

    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) cout << a[i][j] << ' ';
        cout << '\n';
    }
}
```

□

Bài toán 47 (Adjacency matrix ↦ adjacency list converter – chuyển đổi ma trận kề ↦ danh sách kề)**.** *Cho đồ thị đơn vô hướng hữu hạn $G = (V, E)$. Viết thuật toán & chương trình* C/C++, Pascal, Python *để chuyển đổi dạng biểu diễn của đồ thị G từ dạng ma trận kề sang dạng danh sách kề.*

Solution. C++:

1. VNTA's C++: adjacency matrix ↦ adjacency list converter: URL:
   - https://github.com/vntanh1406/Graph_SUM2025/blob/main/BasicGraphTheory/AdjacencyMatrixToAdjacencyList.cpp.
   - https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/C%2B%2B/VNTA_adjacency_matrix_to_adjacency_list.cpp.

```
#include <bits/stdc++.h>
using namespace std;
/*
Cho đồ thị vô hướng G = <V,E> được biểu diễn dưới dạng ma trận kề.
Chuyển đổi biểu diễn đồ thị dưới dạng danh sách cạnh
*/
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    int n;
    cin >> n;
    int a[n + 1][n + 1];
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j) cin >> a[i][j];
    vector<int> adj[n + 1];
    vector<pair<int, int>> edges;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (a[i][j] == 1) edges.push_back({i + 1, j + 1});

    for (int i = 0; i < n; ++i)
        for (auto e : edges) if (e.first == i + 1) adj[i + 1].push_back(e.second);

    for (int i = 0; i < n; ++i) {
        cout << i + 1 << " : ";
        for (auto a : adj[i + 1]) cout << a << ' ';
        cout << '\n';
    }
}
```

□

Bài toán 48 (Edge list ↦ adjacency matrix converter – chuyển đổi danh sách cạnh ↦ ma trận kề)**.** *Cho đồ thị đơn vô hướng hữu hạn $G = (V, E)$. Viết thuật toán & chương trình* C/C++, Pascal, Python *để chuyển đổi dạng biểu diễn của đồ thị G từ dạng danh sách cạnh sang dạng ma trận kề.*

Solution. C++:

1. VNTA's C++: edge list ↦ adjacent matrix: URL:

- https://github.com/vntanh1406/Graph_SUM2025/blob/main/BasicGraphTheory/EdgeListToAdjacencyMatrix.cpp.

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

/*
Cho đồ thị vô hướng G = <V,E> được biểu diễn dưới dạng danh sách cạnh.
Chuyển đổi biểu diễn đồ thị dưới dạng ma trận kề
*/

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    int n, m, u, v;
    cin >> n >> m;
    int a[n][n];
    for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j) a[i][j] = 0;

    for (int e = 0; e < m; e++) {
        cin >> u >> v;
        a[u - 1][v - 1] = 1;
        a[v - 1][u - 1] = 1;
    }
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) cout << a[i][j] << ' ';
        cout << '\n';
    }
}
```

□

Bài toán 49 (Edge list ↦ adjacency list converter – chuyển đổi danh sách cạnh ↦ danh sách kề)**.** *Cho đồ thị đơn vô hướng hữu hạn $G = (V, E)$. Viết thuật toán & chương trình* C/C++, Pascal, Python *để chuyển đổi dạng biểu diễn của đồ thị G từ dạng danh sách cạnh sang dạng danh sách kề.*

Solution. C++:

1. VNTA's C++: edge list ↦ adjacent list:

- https://github.com/vntanh1406/Graph_SUM2025/blob/main/BasicGraphTheory/EdgeListToAdjacencyList.cpp.

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

/*
Cho đồ thị vô hướng G = <V,E> được biểu diễn dưới dạng danh sách cạnh.
Chuyển đổi biểu diễn đồ thị dưới dạng danh sách kề
*/

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    int n, m, u, v;
    cin >> n >> m;
    vector<int> adj[n + 1];
    for (int i = 0; i < m; ++i) {
        cin >> u >> v;
```

```
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    for (int i = 1; i <= n; ++i) {
        cout << i << " : ";
        for (int x : adj[i]) cout << x << ' ';
        cout << '\n';
    }
}
```

□

Bài toán 50 (Adjacency list ↦ adjacency matrix converter – chuyển đổi danh sách kề ↦ ma trận kề)**.** *Cho đồ thị đơn vô hướng hữu hạn $G = (V, E)$. Viết thuật toán & chương trình* C/C++, Pascal, Python *để chuyển đổi dạng biểu diễn của đồ thị G từ dạng danh sách kề sang dạng ma trận kề.*

Solution. C++:

1. VNTA's C++: adjacency list ↦ adjacency matrix converter: URL:

- https://github.com/vntanh1406/Graph_SUM2025/blob/main/BasicGraphTheory/AdjacencyListToAdjacencyMatrix.cpp.
-

```
#include <bits/stdc++.h>
using namespace std;
/*
Cho đồ thị vô hướng G = <V,E> được biểu diễn dưới dạng danh sách kề.
Chuyển đổi biểu diễn đồ thị dưới dạng ma trận kề
*/
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    int n;
    cin >> n;
    cin.ignore();
    int a[n + 1][n + 1];

    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= n; ++j) a[i][j] = 0;

    for (int i = 1; i <= n; ++i) {
        string s, num;
        getline(cin, s);
        stringstream ss(s);
        while (ss >> num) a[i][stoi(num)] = 1;
    }

    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) cout << a[i][j] << ' ';
        cout << '\n';
    }
}
```

□

Bài toán 51 (Adjacency list ↦ edge list converter – chuyển đổi danh sách kề ↦ danh sách cạnh)**.** *Cho đồ thị đơn vô hướng hữu hạn $G = (V, E)$. Viết thuật toán & chương trình* C/C++, Pascal, Python *để chuyển đổi dạng biểu diễn của đồ thị G từ dạng danh sách kề sang dạng danh sách cạnh.*

Solution. C++:

1. VNTA's C++: adjacency list ↦ edge list: URL:

- https://github.com/vntanh1406/Graph_SUM2025/blob/main/BasicGraphTheory/AdjacencyListToEdgeList.cpp.
-

```
#include <bits/stdc++.h>
using namespace std;
/*
Cho đồ thị vô hướng G = <V,E> được biểu diễn dưới dạng danh sách kề.
Chuyển đổi biểu diễn đồ thị dưới dạng danh sách cạnh
*/
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    int n;
    cin >> n;
    cin.ignore();
    vector<pair<int, int>> edges;
    for (int i = 1; i <= n; ++i) {
        string s, num;
        getline(cin, s);
        stringstream ss(s);
        while (ss >> num)
            if (i < stoi(num)) edges.push_back({i, stoi(num)});
    }
    for (auto e : edges) cout << e.first << ' ' << e.second << '\n';
}
```

□

Bài toán 52 (Course project: Graph representations converters – Đồ án môn học: chuyển đổi giữa các dạng biểu diễn đồ thị)**.** *Viết các thuật toán & chương trình chuyển đổi giữa* ≥ 3 *dạng biểu diễn (adjacency list, adjacency matrix, edge list, extended edge list, etc.) của đồ thị G gồm tổ hợp của các loại đồ thị đã học: vô hướng vs. có hướng, đồ thị đơn vs. đa vs. tổng quát. Tạo cấu trúc dữ liệu struct hoặc class để mỗi vertex có thể chứa thêm m giá trị thực, còn mỗi cạnh edge có thể chứa thêm n giá trị thực để sẵn sàng làm toán tối ưu trên kiểu dữ liệu đã được tự định dạng.*

7.6 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).

7.6.1 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 & 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.

7.6.2 Analysis of BFS – Phân tích BFS

7.6.2.1 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).

7.6.2.2 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.*

7.6.3 BFS ordering – Thứ tự BFS

Definition 67 (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 $\sigma = (v_1, \ldots, v_m)$ be a list of distinct elements of V, for $v \in V \backslash \{v_1, \ldots, v_m\}$, 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 $\sigma = (v_1, \ldots, v_n)$ be an enumeration of the vertices of V. The enumeration σ is said to be a *BFS ordering* (with source v_1) if, $\forall i = 2, 3, \ldots, n$, v_i is the vertex $w \in V \backslash \{v_1, \ldots, v_{i-1}\}$ s.t. $\nu_{(v_1,\ldots,v_{i-1})}(w)$ 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. $m < i$.

7.6.4 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.

7.7 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) & 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.

7.7.1 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 92 (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 93. *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 94 (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.*

7.7.2 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.

7.7.2.1 Vertex orderings – Đánh số đỉnh

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.

7.7.2.2 Pseudocode of DFS - Mã giả của 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()
```

7.7.3 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.

7.7.4 Complexity of DFS

The computational complexity of DFS was investigated by John Reif. More precisely, given a graph G, let $O = (v_1, \dots, v_n)$ 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 $O = (v_1, \dots, v_n)$ 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. 1 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.

7.8 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 $P = (v_1, v_2, \dots, v_n) \in V^n = V \times V \times \cdots \times V$ s.t. v_i is adjacent to v_{i+1}, $\forall i = 1, \dots, n-1$. 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 $P = (v_1, \dots, v_n)$ (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.

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

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.

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

- 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.

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

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

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

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.

7.8.1 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.

7.9 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. E.g., 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 8

CSES Problem Set/Graph Algorithms

Problem 95 (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 96 (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 97 (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 98 (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 99 (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 100 (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 101 (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 102 (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 103 (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 104 (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 105 (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 106 (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 9

CSES Problem Set/Tree Algorithms

Chương 10

Posets, Kết Nối, Lưới Boolean

Chương 11

Miscellaneous

11.1 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.