Course Project: Combinatorics & Graph Theory

Vo Ngoc Tram Anh

July 23, 2025

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject

Contents

1 Integer Partition

1.1 Bài toán 1

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/1_IntegerPartition/Baitoan1

Mục tiêu: Cho 2 số nguyên dương n, k. Liệt kê tất cả các phân hoạch của n thành đúng k số nguyên dương sao cho:

- Tổng các phần tử bằng n
- Có đúng k phần tử
- Các phần tử không giảm (để đảm bảo các phân hoạch là phân biệt)

Số lượng các phân hoạch thoả mãn là $p_k(n)$.

Giải thuật: Gọi hàm đệ quy `genPartitions(cur, k, nRemain, part, res)`

- *cur*: chỉ số hiện tại trong phân hoạch (từ 0 đến $k-1$)
- *nRemain*: tổng còn lại cần chia
- *part*: mảng tạm lưu phân hoạch hiện tại
- *res*: danh sách chứa các phân hoạch hoàn chỉnh

Điều kiện dừng: Nếu đã điền đủ k phần tử ($cur = k$):

$$\text{Nếu } nRemain = 0 \Rightarrow \text{thêm phân hoạch vào danh sách kết quả}$$

Tại vị trí cur, ta xét các giá trị nguyên dương x sao cho:

$$x \in [\max(1, part[cur-1]),\ nRemain - (k - cur - 1)]$$

Ý nghĩa:

- Mỗi phần tử phải ≥ 1, nên $x \geq 1$
- Để đảm bảo dãy không giảm, ta cần $x \geq part[cur-1]$
- Để không bị thiếu tổng, cần chừa lại ít nhất $k - cur - 1$ đơn vị cho các phần tử còn lại (mỗi phần tử ít nhất là 1)

Với mỗi giá trị hợp lệ của x, thực hiện:

$$part[cur] \leftarrow x, \quad \texttt{genPartitions}(cur+1, k, nRemain - x, part, res)$$

Biểu diễn Ferrers và Ferrers chuyển vị: Với mỗi phân hoạch, ta biểu diễn Ferrers và Ferrers chuyển vị như sau

1. Ferrers: Cho phân hoạch $(a_1, a_2, \ldots, a_k)$. Ta vẽ k dòng, dòng thứ i có a_i dấu $*$.
2. Ferrers chuyển vị: Xét lại biểu đồ Ferrers theo cột. Dòng i của biểu đồ chuyển vị chứa số lượng dấu $*$ bằng số phần tử trong phân hoạch lớn hơn hoặc bằng i.

1.2 Bài toán 2

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/1_IntegerPartition/Baitoan2

Mục tiêu: Cho 2 số nguyên dương n, k. Cần tính:

- $p(n,k)$: Số phân hoạch của n mà tất cả các phần tử không lớn hơn k.
- $p_{\max}(n,k)$: Số phân hoạch của n mà phần tử lớn nhất bằng chính xác k.
- $p_k(n)$: Số phân hoạch của n thành đúng k phần tử.

Giải thuật:

1. Tính $p(n,k)$ Gọi $p(n,k)$ là số phân hoạch của n sử dụng các phần tử không lớn hơn k. Công thức quy hoạch động được sử dụng là:

$$p(n,k) = \begin{cases} 1 & \text{n} = 0 \\ 0 & \text{n} < 0 \text{ hoặc k} = 0 \\ p(n,k-1) + p(n-k,k) & \text{n, k} > 0 \end{cases}$$

   Giải thích:

   - $p(n,k-1)$: không sử dụng số k, chỉ sử dụng các số nhỏ hơn k.
   - $p(n-k,k)$: sử dụng ít nhất một số k, còn lại phân hoạch $n-k$ với phần tử không vượt quá k.

2. Tính $p_k(n)$ Gọi $pk(n,k)$ là số phân hoạch của n thành đúng k phần tử.

   Công thức quy hoạch động là:

$$pk(n,k) = \begin{cases} 1 & \text{n} = 0, \text{k} = 0 \\ 0 & \text{n} = 0, \text{k} > 0 \text{ hoặc k} = 0, \text{n} > 0 \\ pk(n-1,k-1) + pk(n-k,k) & \text{n, k} > 0 \end{cases}$$

   Giải thích:

   - $pk(n-1,k-1)$: thêm phần tử 1 vào phân hoạch $n-1$ có $k-1$ phần tử.
   - $pk(n-k,k)$: tăng mỗi phần tử trong phân hoạch $n-k$ thành k phần tử lên 1.

3. Tính $p_{\max}(n,k)$ Đây là số phân hoạch của n trong đó phần tử lớn nhất đúng bằng k.

   Ta có:

$$p_{\max}(n,k) = p(n,k) - p(n,k-1)$$

   Giải thích:

   - $p(n,k)$: tất cả phân hoạch với phần tử không vượt quá k.
   - $p(n,k-1)$: tất cả phân hoạch với phần tử không vượt quá $k-1$.
   - $p(n,k) - p(n,k-1)$ chính là những phân hoạch có phần tử lớn nhất là đúng k.

1.3 Bài toán 3

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/1_IntegerPartition/Baitoan3

Lý thuyết

- Phân hoạch tự liên hợp là phân hoạch có biểu đồ Ferrers đối xứng qua đường chéo chính.
- Phân hoạch thành các phần lẻ (odd parts) là phân hoạch mà mỗi phần là số lẻ.
- Số phân hoạch tự liên hợp của n đúng bằng số phân hoạch của n thành các phần lẻ phân biệt (Ref: Wolfram MathWorld).

Giải thuật

- Sinh phân hoạch tự liên hợp:
  - Kiểm tra tự liên hợp: Tạo phân hoạch liên hợp bằng cách đếm số phần tử lớn hơn hoặc bằng $i+1$ trong từng cột của biểu đồ Ferrers. Nếu phân hoạch gốc bằng phân hoạch liên hợp $\Rightarrow$ tự liên hợp.
  - Backtrack sinh phân hoạch: thêm vào danh sách kết quả nếu là tự liên hợp.
- Đếm và sinh phân hoạch thành các phần lẻ:
  - Gọi $dp[i]$ là số cách phân hoạch số i thành tổng các số lẻ, $dp[0] = 1$. Duyệt qua từng số lẻ odd từ 1 đến n, với mỗi odd: $dp[i] += dp[i - odd]$ (phân hoạch i bằng cách thêm odd vào phân hoạch $i - odd$, vẫn đảm bảo phân hoạch i thành các số lẻ)
  - Duyệt qua các số lẻ $odd = minodd, minodd + 2, ...$, với mỗi odd, nếu thêm vào phân hoạch hiện tại thì gọi đệ quy phân hoạch $n - odd$. Dừng khi $n == 0$ (tổng các số trong phân hoạch đang xét bằng đúng ban đầu). Sau khi đệ quy thì `pop_back()` để backtrack xét tiếp các phân hoạch khác.
- Sinh phân hoạch thành các phần lẻ phân biệt: Tương tự như sinh phân hoạch thành các phần lẻ, tuy nhiên ở mỗi bước ta luôn tăng $start$ lên $odd + 2$ để đảm bảo tính phân biệt.

2 Graph & Tree Traversing Problems

2.1 Bài toán 4

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/2_GraphAndTree/Baitoan4

- Graph:
  - From adjacency matrix: https://github.com/vntanh1406/Graph_SUM2025/blob/main/FinalProject/2_GraphAndTree/Baitoan4/FromAdjMat.cpp
  - From adjacency list: https://github.com/vntanh1406/Graph_SUM2025/blob/main/FinalProject/2_GraphAndTree/Baitoan4/FromAdjList.cpp
  - From edge list: https://github.com/vntanh1406/Graph_SUM2025/blob/main/FinalProject/2_GraphAndTree/Baitoan4/FromEdgeList.cpp
  - From adjacency map: https://github.com/vntanh1406/Graph_SUM2025/blob/main/FinalProject/2_GraphAndTree/Baitoan4/FromAdjMap.cpp

- Tree:
  - From array of parents: https://github.com/vntanh1406/Graph_SUM2025/blob/main/FinalProject/2_GraphAndTree/Baitoan4/Tree_FromArrOfParents.cpp
  - From first-child & next-sibling: https://github.com/vntanh1406/Graph_SUM2025/blob/main/FinalProject/2_GraphAndTree/Baitoan4/Tree_FromFCNS.cpp
  - From graph-based representation (adjacency list): https://github.com/vntanh1406/Graph_SUM2025/blob/main/FinalProject/2_GraphAndTree/Baitoan4/Tree_FromGP.cpp

2.2 Bài toán 5

2.2.1 Problem 1.1

- Complete graph K_n: Với n đỉnh, mỗi cặp đỉnh được nối với nhau bởi đúng một cạnh. Số cặp đỉnh là:

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

  Vậy số cạnh của K_n là: $\dfrac{n(n-1)}{2}$

- Complete bipartite graph $K_{p,q}$: Gồm hai tập đỉnh rời có p và q đỉnh, mỗi đỉnh ở tập p nối với tất cả đỉnh ở tập q, nên:

$$\text{số cạnh} = p \cdot q$$

  Vậy số cạnh của $K_{p,q}$ là: $p \cdot q$

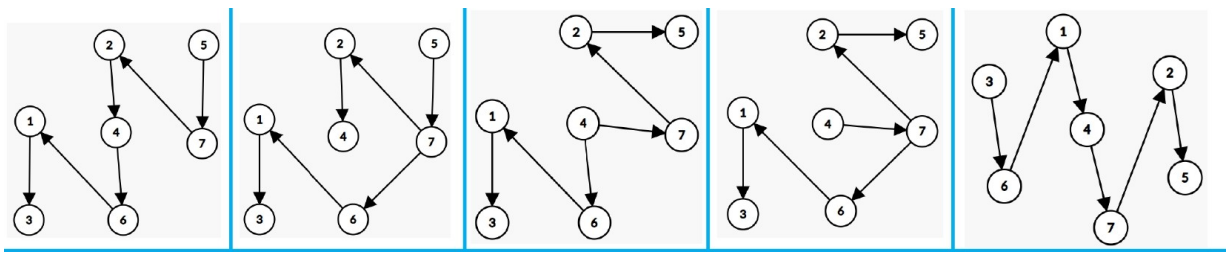
2.2.2 Problem 1.2

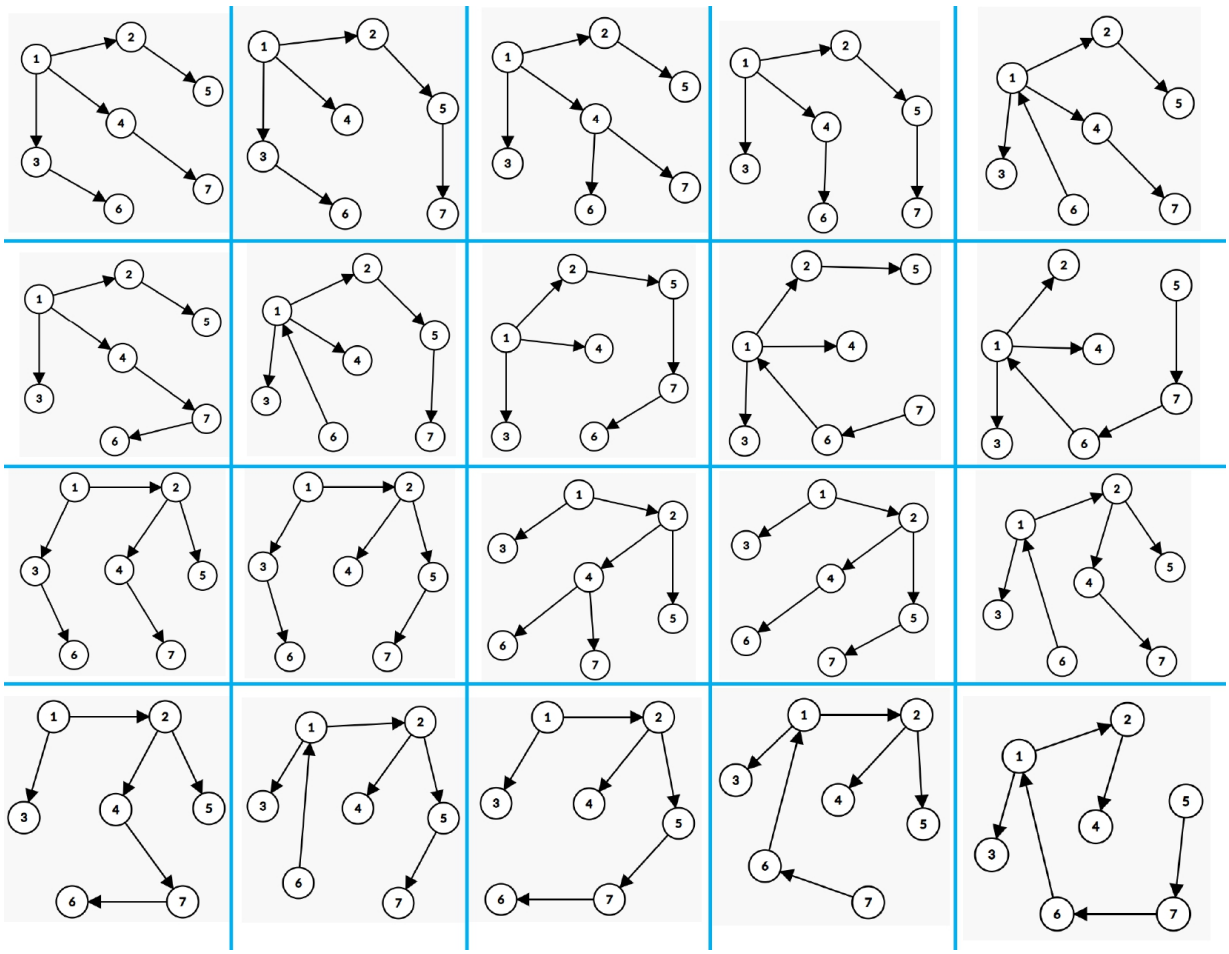
Một đồ thị là bipartite $\iff$ tập đỉnh của nó có thể chia thành hai tập con rời nhau sao cho mọi cạnh đều nối một đỉnh ở tập này với một đỉnh ở tập kia. Vì vậy , bipartite graph không chứa chu trình nào có số cạnh lẻ.

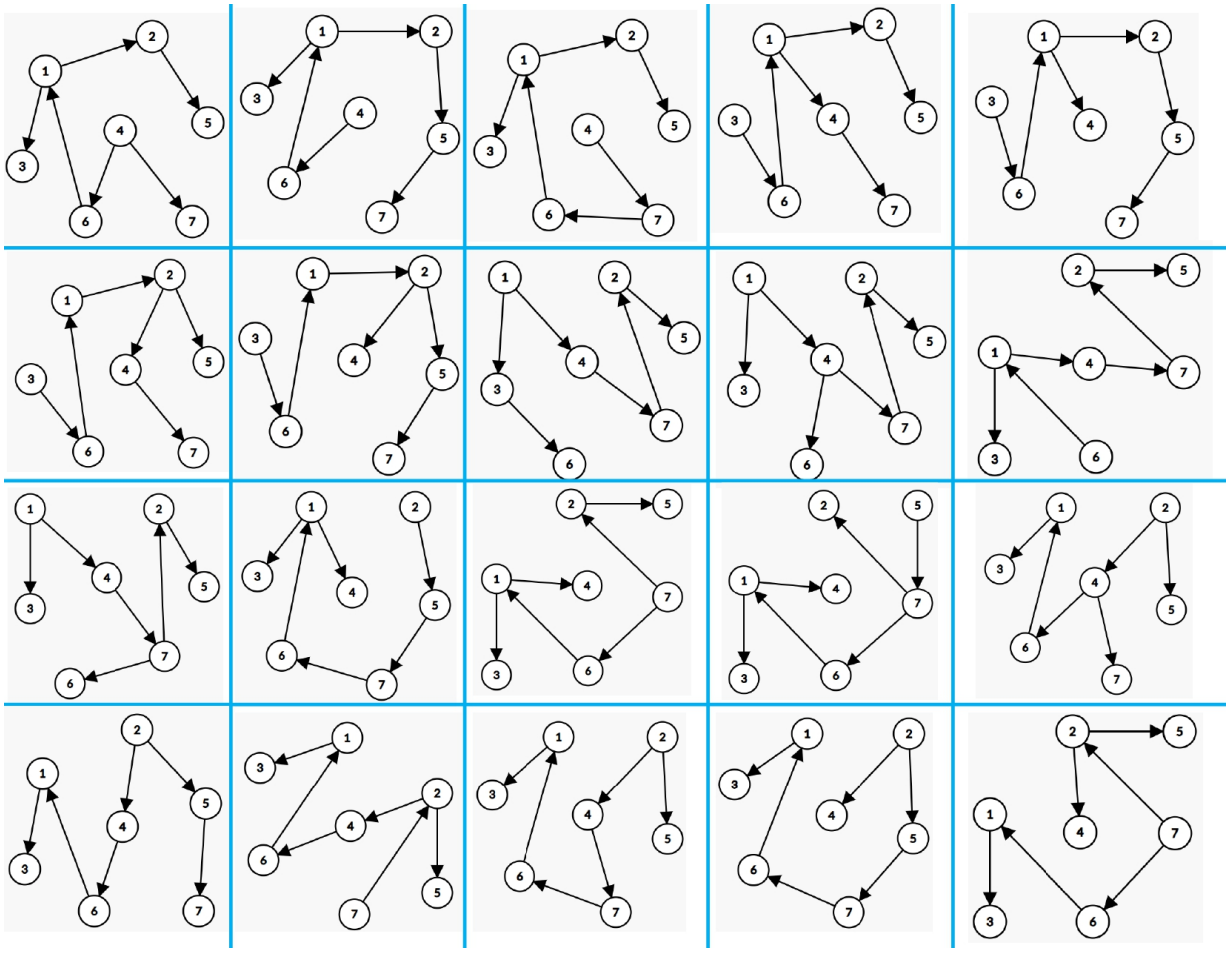
- Circle graph C_n gồm n đỉnh là một cycle n cạnh. Vì vậy, C_n là bipartite $\iff$ n chẵn.
- Complete graph K_n:
  - K_1: chỉ có một đỉnh, không có cạnh $\Rightarrow$ bipartite.
  - K_2: chia hai đỉnh vào hai tập khác nhau $\Rightarrow$ bipartite.
  - K_3: có ba đỉnh và ba cạnh tạo thành chu trình lẻ (tam giác) $\Rightarrow$ không phải bipartite.
  - Với $n \geq 3$: tồn tại chu trình lẻ trong K_n $\Rightarrow$ không phải bipartite.

  Vậy K_n là bipartite graph khi và chỉ khi $n \leq 2$

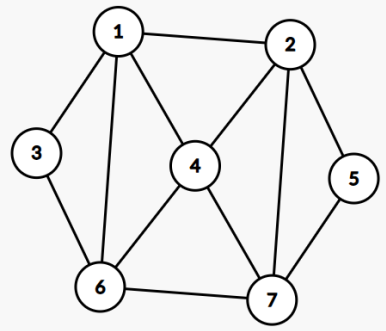
2.2.3 Problem 1.3







Tổng số lượng spanning tree (với đồ thị vô hướng): (Ref: Wiki - Kirchhoff's Theorem)



- Xét ma trận bậc $D = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 4 \end{bmatrix}$,ma trận kề $A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$
- Xét ma trận Laplacian

$$L = D - A = \begin{bmatrix} 4 & -1 & -1 & -1 & 0 & -1 & 0 \\ -1 & 4 & 0 & -1 & -1 & 0 & -1 \\ -1 & 0 & 2 & 0 & 0 & -1 & 0 \\ -1 & -1 & 0 & 4 & 0 & -1 & -1 \\ 0 & -1 & 0 & 0 & 2 & 0 & -1 \\ -1 & 0 & -1 & -1 & 0 & 4 & -1 \\ 0 & -1 & 0 & -1 & -1 & -1 & 4 \end{bmatrix}$$

- Xóa 1 hàng và 1 cột của L (chọn xóa hàng 1, cột 1) được ma trận L_{minor}. Sau đó tính $det(L_{minor})$ ta được số spanning trees của đồ thị ban đầu là 288.

```
import numpy as np
L = np.array([
    [ 4, -1, -1, -1,  0, -1,  0],
    [-1,  4,  0, -1, -1,  0, -1],
    [-1,  0,  2,  0,  0, -1,  0],
    [-1, -1,  0,  4,  0, -1, -1],
    [ 0, -1,  0,  0,  2,  0, -1],
    [-1,  0, -1, -1,  0,  4, -1],
    [ 0, -1,  0, -1, -1, -1,  4]
])
L_minor = np.delete(np.delete(L, 0, axis=0), 0, axis=1)
print(round(np.linalg.det(L_minor)))
```

- Code tìm spanning tree của đồ thị có hướng và vô hướng: https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/2_GraphAndTree/Baitoan5/Prob13

2.2.4 Problem 1.4

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/2_GraphAndTree/Baitoan5/Prob14

Với đồ thị được biểu diễn bằng ma trận kề, không tồn tại đối tượng cạnh riêng biệt. Do đó, mọi thao tác có liên quan đến cạnh (edge object) sẽ được chuyển đổi sang dạng thao tác với cặp đỉnh (v, w).

Ta xây dựng lớp Graph cho đồ thị n đỉnh như sau:

- Sử dụng ma trận kề $adj[n][n]$ với $adj[v][w] = 1$ nếu tồn tại cạnh nối từ v đến w.
- `del_edge`(v, w): Xóa cạnh nối từ v đến w bằng cách đặt $adj[v][w] := 0$.
- `edges`(): Duyệt toàn bộ ma trận kề, thêm mọi cặp đỉnh (v, w) mà $adj[v][w] = 1$ vào danh sách kết quả. Kết quả trả về là danh sách các cạnh dưới dạng các cặp đỉnh (v, w).
- `incoming`(v): Tìm các đỉnh u sao cho có cạnh nối từ u đến v, tức $adj[u][v] = 1$.Kết quả trả về là danh sách các cạnh dưới dạng các cặp đỉnh (u, v) (hoặc danh sách các đỉnh u).
- `outgoing`(v): Tìm các đỉnh w sao cho có cạnh nối từ v đến w, tức $adj[v][w] = 1$. Kết quả trả về là danh sách các cạnh dưới dạng các cặp đỉnh (v, w) (hoặc danh sách các đỉnh w).
- `source`(v, w): Với cạnh được xác định bằng cặp đỉnh (v, w) thì đỉnh nguồn chính là v nếu cạnh tồn tại.
- `target`(v, w): Với cạnh được xác định bằng cặp đỉnh (v, w) thì đỉnh đích chính là w nếu cạnh tồn tại.

2.2.5 Problem 1.5

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/2_GraphAndTree/Baitoan5/Prob15

Giải thích và ý tưởng: Cấu trúc *First-Child, Next-Sibling* biểu diễn mỗi nút trong cây bằng hai con trỏ:

- $first\_child$: trỏ tới con đầu tiên của nút.
- $next\_sibling$: trỏ tới anh em kế tiếp của nút.

Tuy nhiên, trong biểu diễn truyền thống, để lấy số con của nút v hoặc tập con của v, ta phải duyệt danh sách các anh em kế tiếp, mất thời gian $\mathcal{O}(k)$ với k là số con của v. Để đạt được phép toán $\mathcal{O}(1)$, ta mở rộng như sau:

- Dùng 1 biến riêng lưu *root* cho cây, cho phép lấy *root* trong $\mathcal{O}(1)$.
- Mỗi node v được mở rộng thêm trường $num\_children$ lưu số lượng con hiện tại của v. Việc cập nhật số con được thực hiện mỗi khi thêm hoặc xóa con.
- Giữ nguyên con trỏ $first\_child$ và $next\_sibling$: Giúp truy xuất tập con của nút v bằng cách truy cập con trỏ $first\_child$ (trả về danh sách con dạng liên kết). Việc trả về con trỏ này là $\mathcal{O}(1)$.

Kết quả:

- T.`root`(): Trả về biến *root* ngay lập tức, thời gian $\mathcal{O}(1)$.
- T.`number_of_children(v)`: Trả về $num\_children$ của node v mà không phải duyệt danh sách con, thời gian $\mathcal{O}(1)$.
- T.`children(v)`: Trả về con trỏ $first\_child$ của nút v, cho phép duyệt hoặc truy cập tập con theo liên kết anh em kế tiếp, thời gian trả về con trỏ là $\mathcal{O}(1)$.

2.2.6 Problem 1.6

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/2_GraphAndTree/Baitoan5/Prob16

Phân tích: Để kiểm tra xem một đồ thị có phải là cây hay không, ta dựa vào định nghĩa:

- Đồ thị phải liên thông.
- Không chứa chu trình.
- Có đúng $n-1$ cạnh với n đỉnh.
- Với đồ thị có hướng (cây có gốc - rooted directed tree) thì cần xét thêm:
  - Có đúng một đỉnh gốc (*root*) với $indegree = 0$.
  - Mọi đỉnh còn lại đều có $indegree = 1$.

Giải thuật: Độ phức tạp $\mathcal{O}(n+m)$, tuyến tính theo kích thước đồ thị (với n là số đỉnh, m là số cạnh).

1. Kiểm tra số cạnh: không phải $n-1$ thì không phải cây.
2. Kiểm tra tính liên thông và chu trình:
   - Với đồ thị vô hướng:
     - Duyệt DFS hoặc BFS từ một đỉnh bất kỳ.
     - Tổng số đỉnh đã thăm phải bằng n (đảm bảo liên thông).
     - Nếu trong quá trình duyệt gặp lại đỉnh đã thăm mà không phải là cha thì có chu trình.
   - Với đồ thị có hướng:
     - Kiểm tra tồn tại đúng một đỉnh gốc ($indegree = 0$), các đỉnh còn lại có $indegree = 1$.
     - Duyệt DFS từ đỉnh gốc.
     - Nếu tồn tại chu trình (phát hiện trong DFS bằng phương pháp 3 trạng thái), hoặc không thăm hết các đỉnh thì không phải cây.

2.2.7 Exercise 1.1

Định dạng DIMACS: Dùng cho đồ thị vô hướng.

- Dòng định nghĩa vấn đề: `p edge n m` với n là số đỉnh, m là số cạnh.
- m dòng miêu tả các cạnh: `e i j`, với i, j là chỉ số các đỉnh (từ 1 đến n).
- Các dòng bắt đầu bằng `c` là dòng chú thích, có thể bỏ qua.

Yêu cầu: Cài đặt hai hàm.

- `read_dimacs()` để đọc dữ liệu đồ thị từ định dạng DIMACS và lưu vào cấu trúc dữ liệu.
- `write_dimacs()` để xuất đồ thị hiện tại ra định dạng DIMACS.

Giải pháp :

- Khi đọc bỏ qua các dòng comment, lấy số đỉnh và số cạnh từ dòng `p edge n m`.
- Lưu các cạnh vào danh sách kề, chuyển chỉ số đỉnh từ 1-based sang 0-based để thuận tiện xử lý trong code.
- Khi ghi, in ra dòng `p edge n m` và các dòng cạnh `e i j`, chuyển lại sang 1-based index.

2.2.8 Exercise 1.2

Định dạng SGB: Định dạng Stanford GraphBase (SGB) biểu diễn đồ thị (có hướng hoặc vô hướng) theo cấu trúc:

- Dòng 1: `* GraphBase graph (utiltypes..., nV, mA)`
  Với nV số đỉnh, mA số cung (arcs).
- Dòng 2: Chuỗi nhận diện đồ thị (tên hoặc mô tả).
- Dòng 3: `* Vertices` đánh dấu bắt đầu phần đỉnh.
- Dòng 4 đến $nV + 3$: mỗi dòng mô tả một đỉnh: `label, Ai, 0, 0`
  Trong đó:
  - `label` là nhãn đỉnh (chuỗi).
  - `Ai` là chỉ số cạnh đầu tiên đi ra từ đỉnh, trong khoảng $[0, mA - 1]$. Nếu đỉnh không có cạnh đi ra, `Ai` $= 0$.
  - Hai số 0 cuối dòng là các trường không dùng.
- Dòng $nV + 4$: `* Arcs` đánh dấu bắt đầu phần cung.
- Dòng $nV + 5$ đến $nV + mA + 4$: mỗi dòng mô tả một cung: `Vj, Ai, label, 0`
  Trong đó:
  - `Vj` là chỉ số đỉnh đích (từ 0 đến $nV - 1$).
  - `Ai` là chỉ số cạnh kế tiếp cùng xuất phát từ đỉnh nguồn; nếu đây là cung cuối cùng từ đỉnh nguồn thì `Ai` $= 0$.
  - `label` là nhãn cung (thường là số nguyên).
  - Số 0 cuối dòng là trường không dùng.
- Dòng cuối: `* Checksum ...` để kiểm tra tính toàn vẹn định dạng.

Yêu cầu: Cài đặt hai hàm

- `read_sgb()` để đọc dữ liệu đồ thị từ file văn bản theo định dạng SGB chuẩn, bao gồm các phần:
  - Dòng đầu: `* GraphBase graph (utiltypes..., nV, mA)`
  - Dòng nhận diện đồ thị
  - Dòng `* Vertices` và n dòng mô tả đỉnh: `label, Ai, 0, 0`
  - Dòng `* Arcs` và m dòng mô tả cung: `Vj, Ai, label, 0`
  - Dòng `* Checksum ...`
- `write_sgb()` để ghi dữ liệu đồ thị ra file văn bản theo đúng cấu trúc định dạng SGB chuẩn như trên.

Giải pháp:

- Trong hàm `read_sgb()`, tiến hành đọc tuần tự từng dòng từ file hoặc luồng nhập, phân tích cú pháp các phần theo định dạng đã mô tả (dòng đầu, đỉnh, cung, checksum).
- Lưu thông tin đỉnh và cung vào cấu trúc dữ liệu thích hợp, ví dụ: danh sách đỉnh với nhãn và chỉ số cạnh đầu tiên, danh sách cung với đỉnh đích, chỉ số cạnh kế tiếp cùng nguồn, và nhãn cạnh.
- Trong hàm `write_sgb()`, xuất lần lượt các phần gồm: dòng tiêu đề, dòng nhận diện đồ thị, phần `* Vertices` với mô tả đỉnh, phần `* Arcs` với mô tả cung, và cuối cùng dòng `* Checksum` để hoàn thiện định dạng.

2.2.9 Exercise 1.3

Mục tiêu: Sử dụng bộ 32 phép toán trừu tượng trên đồ thị (phần 1.3) để xây dựng các đồ thị tiêu chuẩn gồm:

- Path graph P_n với n đỉnh.
- Circle graph C_n với n đỉnh.
- Wheel graph W_n với n đỉnh.

Ý tưởng giải thuật:

- Khởi tạo đồ thị rỗng.
- Dùng phép `new_vertex()` thêm lần lượt n đỉnh.
- Với P_n, thêm cạnh nối liền các đỉnh liên tiếp: $\forall i = 0 \dots n-2$, `new_edge`(v_i, v_{i+1}).
- Với C_n, xây dựng P_n rồi thêm cạnh nối từ đỉnh cuối về đỉnh đầu: `new_edge`(v_{n-1}, v_0).
- Với W_n, thêm đỉnh trung tâm v_0, tạo vòng tròn trên các đỉnh còn lại $v_1 \dots v_{n-1}$ như C_{n-1}, rồi nối đỉnh trung tâm tới tất cả các đỉnh trên vòng tròn.

2.2.10 Exercise 1.4

Mục tiêu: Sử dụng bộ 32 phép toán trừu tượng trên đồ thị (phần 1.3) để xây dựng các đồ thị sau:

- Complete graph K_n gồm n đỉnh, trong đó mỗi cặp đỉnh đều có cạnh nối.
- Complete bipartite graph $K_{p,q}$ gồm hai tập đỉnh rời nhau P và Q với kích thước p và q, trong đó mỗi đỉnh của tập P nối với mọi đỉnh của tập Q.

Ý tưởng giải thuật:

- Khởi tạo đồ thị rỗng.
- Dùng `new_vertex()` thêm các đỉnh cần thiết:
  - Với K_n, thêm n đỉnh.
  - Với $K_{p,q}$, thêm $p+q$ đỉnh, chia làm 2 tập P và Q.
- Với K_n, duyệt tất cả các cặp đỉnh (v_i, v_j) với $0 \le i < j < n$, thêm cạnh `new_edge`(v_i, v_j) và nếu là đồ thị vô hướng, thêm cả `new_edge`(v_j, v_i) nếu cần thiết.
- Với $K_{p,q}$, với mỗi đỉnh $u \in P$ và $v \in Q$, thêm cạnh `new_edge`(u, v).

2.2.11 Exercise 1.5

Mục tiêu: Triển khai lớp *Graph* biểu diễn đồ thị sử dụng ma trận kề mở rộng (theo Promblem 1.4), sử dụng Python lists để lưu trữ và quản lý các đỉnh, cạnh theo số thứ tự nội bộ (internal numbering).

Mô tả và yêu cầu:

- Sử dụng ma trận kề adj kích thước $n \times n$, với $adj[v][w] = 1$ nếu tồn tại cạnh từ đỉnh v đến đỉnh w, ngược lại 0.
- Cài đặt các phép toán chính:
  - `del_edge(v, w)`: xóa cạnh nối từ v đến w.
  - `edges()`: trả về danh sách các cạnh dưới dạng cặp đỉnh (v, w).
  - `incoming(v)`: trả về danh sách các đỉnh có cạnh đi vào v.
  - `outgoing(v)`: trả về danh sách các đỉnh có cạnh đi ra từ v.
  - `source(v, w)`: trả về đỉnh nguồn v của cạnh (v, w) nếu tồn tại, ngược lại `None`.
  - `target(v, w)`: trả về đỉnh đích w của cạnh (v, w) nếu tồn tại, ngược lại `None`.
- Quản lý đỉnh với số thứ tự từ 0 đến $n - 1$ (internal numbering).
- Cài đặt: Problem 1.4

2.2.12 Exercise 1.6

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/2_GraphAndTree/Baitoan5/Ex16

Mục tiêu: Liệt kê tất cả các perfect matching trong complete bipartite graph $K_{p,q}$.

Phân tích:

- Để tồn tại perfect matching, hai tập đỉnh phải có cùng số lượng: $p = q$.
- Mỗi perfect matching tương ứng với một hoán vị của tập Q theo thứ tự tập P.
- Do đó, nhiệm vụ liệt kê các perfect matching tương đương với việc liệt kê tất cả các hoán vị của tập Q.

Giải thuật: Sử dụng sinh hoán vị (permutation) của tập Q. Với mỗi hoán vị π, tạo ra tập cạnh $\{(P_i, Q_{\pi(i)})\}_{i=1}^{p}$.

2.2.13 Exercise 1.7

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/2_GraphAndTree/Baitoan5/Ex17

Mục tiêu: Sử dụng bộ 13 phép toán trừu tượng trên cây để tạo complete binary tree n node.

Ý tưởng:

- Gán số thứ tự các node từ 0 đến $n-1$ theo thứ tự BFS.
- 0 là root.
- Với node i, nếu tồn tại, node con trái là $2i+1$, node con phải là $2i+2$.
- Thêm cạnh từ node cha i đến các con trái và phải nếu chỉ số con nhỏ hơn n.

Thuật toán:

1. Tạo mảng $nodes[0 \dots n-1]$.
2. Với $i = 0 \to n-1$, thực hiện: $nodes[i] = T.\texttt{new\_node()}$
3. Gán node gốc: $T.\texttt{root()} = nodes[0]$.
4. Với $i = 0 \to \lfloor \frac{n-2}{2} \rfloor$:
   - Nếu $2i+1 < n$: $T.\texttt{children}(nodes[i]).\texttt{append}(nodes[2i+1])$
   - Nếu $2i+2 < n$: $T.\texttt{children}(nodes[i]).\texttt{append}(nodes[2i+2])$

2.2.14 Exercise 1.8

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/2_GraphAndTree/Baitoan5/Ex18

Mục tiêu: Sử dụng bộ 13 phép toán trừu tượng trên cây để tạo cây ngẫu nhiên n node.

Ý tưởng: Dựa trên một tính chất cơ bản: Mỗi cây có n đỉnh thì có đúng $n-1$ cạnh. Để sinh một cây ngẫu nhiên với n đỉnh, ta gán lần lượt mỗi đỉnh từ 1 đến $n-1$ làm con của một đỉnh đã được tạo trước đó (từ 0 đến $i-1$). Điều này đảm bảo không tạo chu trình và tạo nên một cây hợp lệ.

Thuật toán:

1. Khởi tạo cây rỗng T và gán $T.\texttt{root()} \leftarrow 0$.
2. Với mỗi i từ 1 đến $n-1$:
   - Gọi $T.\texttt{number\_of\_nodes()} = i$.
   - Sinh ngẫu nhiên một node $p \in \{0, 1, \dots, i-1\}$.
   - Thêm node i làm con của node p thông qua phép toán trừu tượng: $T.\texttt{children}(p).\texttt{append}(i)$.
   - Gán ngược lại: $T.\texttt{parent}(i) \leftarrow p$.

Độ phức tạp:

- Thời gian: Thuật toán lặp qua từng node từ 1 đến $n-1$, mỗi bước thực hiện các thao tác như sinh ngẫu nhiên, thêm con, gán cha đều là $\mathcal{O}(1)$. Do đó tổng thời gian là $\mathcal{O}(n)$.
- Bộ nhớ: Cần lưu thông tin cha con cho tất cả n node, tổng cộng bộ nhớ dùng là $\mathcal{O}(n)$.

2.2.15 Exercise 1.9

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/2_GraphAndTree/Baitoan5/Ex19

Mô tả bài toán: Cho cây T được lưu trữ dưới dạng mảng cha *parent*, trong đó $parent[v]$ là cha của node v (nếu v là *root* thì $parent[v]$ là `None` hoặc giá trị đặc biệt). Yêu cầu cài đặt phép toán `T.previous_sibling(v)` trả về previous sibling của v trong cây, nếu tồn tại; ngược lại trả về `None`.

Ý tưởng:

- Tìm node cha $p = parent[v]$.
- Tìm tất cả các node con của p, tức các node u sao cho $parent[u] = p$.
- Xác định vị trí của v trong danh sách các con của p.
- Nếu v không phải là node con đầu tiên, trả về node đứng ngay trước nó trong danh sách, ngược lại trả về `None`.

2.2.16 Exercise 1.10

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/2_GraphAndTree/Baitoan5/Ex110

Mục tiêu: Xây dựng cấu trúc cây theo mô hình *First-Child, Next-Sibling* mở rộng, cho phép truy cập số con của mỗi node trong thời gian $\mathcal{O}(1)$ và duyệt tập con theo danh sách liên kết.

Ý tưởng:

- Mỗi node được biểu diễn bằng ba trường chính:
  - $first\_child$: trỏ đến con đầu tiên của node (hoặc `None` nếu không có con).
  - $next\_sibling$: trỏ đến anh em kế tiếp của node (hoặc `None` nếu là anh em cuối).
  - $num\_children$: lưu số lượng con hiện tại của node, cập nhật khi thêm hoặc xóa con.
- Bổ sung trường *parent* để thuận tiện trong các phép toán với cây.
- Cây có biến *root* lưu chỉ số nút gốc, cho phép truy cập root nhanh chóng.
- Việc thêm con được thực hiện bằng cách chèn node con vào đầu danh sách con của node cha, cập nhật liên kết và số con.
- Duyệt con của một nút sẽ bắt đầu từ $first\_child$ và theo chuỗi $next\_sibling$ cho đến hết.

2.3 Bài toán 6

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/2_GraphAndTree/Baitoan6

2.4 Bai toán 7

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/2_GraphAndTree/Baitoan7

2.5 Bài toán 8 - 9 - 10

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/2_GraphAndTree/Baitoan8_9_10

2.6 Bài toán 11 - 12 - 13

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/2_GraphAndTree/Baitoan11_12_13

3 Shortest Path Problems on Graphs

3.1 Bài toán 14 - 15 - 16

https://github.com/vntanh1406/Graph_SUM2025/tree/main/FinalProject/3_ShortestPath/Baitoan14_15_16