Cơ sở lý thuyết và
Hướng dẫn sử dụng chương trình

LTDT-TTTH

Phần 1: Cơ sở lý thuyết

Yêu cầu 1: Nhận diện một số dạng đồ thị đặc biệt

a. Kiểm tra đồ thị cối xay gió Wd(k,n) (windmill graph) với k = 3 (cố định). Xác định tham số n.

- Định nghĩa: Đồ thị cối xay gió Wd(k,n)
  - Đồ thị vô hướng
  - Gồm n cánh quạt (n ≥ 2)
  - Mỗi cánh quạt là 1 đồ thị đầy đủ K_k (k ≥ 2)
  - Các cánh quạt được nối chung với nhau tại 1 đỉnh ở trung tâm
- Thuộc tính
  - Số đỉnh = n (k – 1) + 1
  - Số cạnh = nk (k − 1)/2
  - Đỉnh trung tâm là 1 đỉnh khớp, có bậc = (số đỉnh - 1)
  - Bậc của các đỉnh còn lại là k - 1
- Trường hợp đặc biệt với k = 3 (cố định)
  - Đồ thị cối xay gió Wd(3, n) là còn được gọi là đồ thị tình bạn (Friendship graph) F_n
  - Số đỉnh = 2n + 1
  - Số cạnh = 3n
  - Mỗi cánh quạt là 1 đồ thị vòng C_3 , hoặc bậc của tất cả các đỉnh ngoại trừ đỉnh khớp đều bằng 2
- Hướng tiếp cận: Từ định nghĩa và các thuộc tính trên, ta xây dựng thuật toán kiểm tra đồ thị Wd(3,n) đồng thời xác định tham số n như sau:
  - Từ file input ta có được số đỉnh và số cạnh => tìm được n
  - Xác nhận lại n bằng cách kiểm tra số đỉnh khớp, bậc của đỉnh khớp, bậc của các đỉnh còn lại

b. Kiểm tra đồ thị Barbell bậc k (Barbell graph). Xác định tham số k.

- Định nghĩa: Đồ thị thanh đòn (đồ thị tạ) k-Barbell
  - Đồ thị vô hướng
  - Gồm 2 đồ thị đầy đủ K_k ,kết nối với nhau bằng 1 cạnh cầu
- Thuộc tính
  - Chỉ có duy nhất 1 cạnh cầu
  - 2 đỉnh của cạnh cầu đều có bậc = k

    - Bậc của các đỉnh không thuộc cạnh cầu = k -1
    - Số đỉnh = 2k
    - 1 Đồ thị đầy đủ có k*(k-1)/2 cạnh
    - => Đồ thị thanh đòn có Số cạnh = 2*(số cạnh của đồ thị đầy đủ) =2*(k*(k-1)/2)+1
    - => Số cạnh = k*(k-1) + 1
- Hướng tiếp cận: Từ định nghĩa và các thuộc tính trên, ta xây dựng thuật toán kiểm tra đồ thị k-Barbell đồng thời xác định tham số k như sau:
    - Từ file input ta có được số đỉnh => tìm được k
    - Xác nhận lại k bằng cách kiểm tra số lượng cạnh, số cạnh cầu, bậc của 2 đỉnh thuộc cạnh cầu, bậc của các đỉnh còn lại

c. Kiểm tra đồ thị k-phân (k-partite graph) với k > 2 . Xác định tham số k và xác định chỉ mục của các đỉnh nằm trong từng tập hợp con không giao nhau.

- Đồ thị k-phân là khái niệm được mở rộng từ đồ thị lưỡng phân, trong đó tập hợp đỉnh được chia thành k tập hợp con không giao nhau, thay vì chỉ 2 tập hợp con, và các đỉnh thuộc cùng một tập hợp không có liên kết với nhau.
- Hướng tiếp cận:
    - Kiểm tra điều kiện đề bài k > 2, nghĩa là số lượng đỉnh tối thiểu > 2 (điều kiện đã phù hợp với file input đầu vào)
    - Giả sử cho 1 đỉnh đầu tiên vào 1 tập hợp con
    - Duyệt qua lần lượt các đỉnh
    - Kiểm tra Danh Sách Đỉnh Kề (DS_Ke) của đỉnh đang duyệt với các tập hợp con
    - Nếu không có phần tử nào trùng nhau giữa DS_Ke và tập con đang duyệt thì thêm đỉnh đó vào tập con hiện tại
    - Nếu phát hiện có phần tử trùng nhau với tập hợp con hiện tại thì kiểm tra tập con tiếp theo
    - Nếu không còn tập con nào để kiểm tra, nghĩa là đỉnh đang duyệt đang tồn tại ít nhất 1 liên kết với mỗi tập con, thì tạo thêm 1 tập con mới để chứa đỉnh đó.

Yêu cầu 2: Xác định thành phần liên thông mạnh

a. Xác định đồ thị được cho là đồ thị liên thông mạnh, liên thông một phần, liên thông yếu hay không liên thông.
b. Xác định các thành phần liên thông mạnh có trong đồ thị. Với mỗi thành phần liên thông mạnh, cho biết chỉ mục của các đỉnh thuộc về thành phần liên thông đó.

- Định nghĩa:
    - Đồ thị có hướng gọi là **liên thông mạnh** (strongly connected) nếu có đường đi từ a tới b và từ b tới a với mọi cặp đỉnh a và b của đồ thị.
    - Đồ thị có hướng được gọi là **liên thông một phần** (unilaterally connected) nếu với mọi cặp đỉnh a, b bất kỳ, có ít nhất một đỉnh đến được đỉnh còn lại.
    - Đồ thị có hướng gọi là **liên thông yếu** (weakly connected) nếu có đường đi giữa 2 đỉnh bất kỳ của đồ thị vô hướng nền.

    - Đồ thị vô hướng nền không liên thông thì đồ thị có hướng tương ứng cũng **không liên thông**
    - **Thành phần liên thông mạnh** là thành phần có đường đi từ mọi đỉnh đến mọi đỉnh khác trong cùng 1 thành phần đó
- Hướng tiếp cận:
    - Dùng thuật toán Kosaraju để xác định các thành phần liên thông mạnh của đồ thị trước, sau đó mới kiểm tra ngược lại là đồ thị đã cho thuộc dạng đồ thị nào
    - Đồng thời tạo 1 đồ thị vô hướng nền (UG) từ đồ thị có hướng (G) của đề bài. (Chỉ cần tạo danh sách kề của UG để đếm số thành phần liên thông)

    - Nếu UG không liên thông
        - Thì G cũng không liên thông
    - Ngược lại, nếu UG liên thông, thì chia thành các trường hợp:
        - Nếu G chỉ có duy nhất 1 thành phần liên thông mạnh
            - Thì G là đồ thị liên thông mạnh
        - Ngược lại, nếu G có nhiều hơn 1 thành phần liên thông mạnh
            - Nếu tồn tại 1 cặp đỉnh bất kỳ mà không có đường đi (chỉ cần có đường đi 1 chiều) thì G là đồ thị liên thông yếu
            - Ngược lại, là đồ thị liên thông từng phần

- Thuật toán Kosaraju để tìm thành phần liên thông mạnh
    - Chạy DFS trên đồ thị gốc, lưu thứ tự duyệt đỉnh vào stack
    - Lật ngược tất cả các cung trên đồ thị gốc (tạm gọi là reverse graph - RG). Trên đồ thị ngược RG, các thành phần liên thông mạnh sẽ không bị thay đổi.
    - Chạy DFS trên đồ thị ngược, với thứ tự duyệt đỉnh được lấy từ trong stack ra, ta được các thành phần liên thông mạnh (Strongly Connected Components (SCC))

Tham khảo: Kosaraju uing DFS
https://www.geeksforgeeks.org/connectivity-in-a-directed-graph/
https://www.geeksforgeeks.org/strongly-connected-components/

Yêu cầu 3: Tìm cây khung lớn nhất

a. Tìm cây khung lớn nhất trên đồ thị đã cho bằng cách hiệu chỉnh giải thuật Prim.

Sử dụng thuật toán Prim có sẵn, hiệu chỉnh bước 2.

Bước 1. Khởi tạo danh sách các cạnh thuộc cây khung $E_T = \emptyset$ và khởi tạo mảng used để đánh dấu 1 đỉnh đang thuộc tập V hay tập V\Y, trong đó:

- V là tập danh sách các đỉnh của đồ thị được cung cấp ban đầu
- Y là tập danh sách các đỉnh của cây khung lần lượt được tìm thấy từ việc chạy thuật toán
- V\Y là tập danh sách các đỉnh của đồ thị nhưng chưa được thêm vào danh sách đỉnh của cây khung
- Nếu used[i] == true nghĩa là đỉnh i đang thuộc tập V
- Nếu used[i] == false nghĩa là đỉnh i đang thuộc tập V\Y
- Đỉnh bắt đầu do người dùng nhập từ bàn phím

Bước 2. Trong số những cạnh e = {u, v}, trong đó $u \in Y$ và $v \in V\backslash Y$, ta chọn cạnh $e_i = \{u_i, v_i\}$ có trọng số lớn nhất.

Bước 3. Gán Y = Y ∈ {v_i}, $E_T = E_T \cup \{e_i\}$

Bước 4. Nếu E_T đủ n – 1 phần tử thì dừng (n là số lượng đỉnh), ngược lại làm tiếp bước 2.

b. Tìm cây khung lớn nhất trên đồ thị đã cho bằng cách hiệu chỉnh giải thuật Kruskal.

Sử dụng thuật toán Kruskal có sẵn, hiệu chỉnh bước 1.

Bước 1. Sắp xếp tất cả các cạnh của đồ thị theo thứ tự trọng số giảm dần và khởi tạo E_T =∅.

Bước 2. Lần lượt lấy từng cạnh e trong danh sách đã sắp xếp. Nếu $E_T \cup \{e\}$ không tạo thành chu trình trong tập danh sách cây khung T thì gán $E_T = E_T \cup \{e\}$

Bước 3. Nếu E_T đủ n – 1 phần tử thì dừng, ngược lại làm tiếp bước 2.

Yêu cầu 4: Tìm đường đi ngắn nhất bằng thuật toán Floyd-Warshall

- Thuật toán Floyd-Warshall được dùng để tìm ra đường đi ngắn nhất giữa tất cả các cặp đỉnh bất kỳ của một đồ thị với các cạnh có trọng lượng dương
- Dữ liệu nhập cho thuật toán là ma trận trọng lượng
- Cách hoạt động của thuật toán:
    - **Khởi đầu với ma trận trọng lượng:** Khởi tạo ma trận trọng lượng ban đầu sao cho đường đi trực tiếp giữa các đỉnh có trọng số là trọng số của cạnh, và nếu không có cạnh nào nối trực tiếp giữa hai đỉnh thì trọng số của cạnh đó được đặt là vô cùng (đại diện cho không có đường đi giữa hai đỉnh).
    - **Chép ma trận trọng lượng ra một ma trận trọng lương mới** `khoangCach[i, j]`**:** Ma trận mới có kích thước mảng bằng với ma trận trọng lượng để lưu trữ lại dữ liệu của đồ thị, những giá trị mỗi phần tử (P[i,j]) đại diện cho việc từ đỉnh i đến đỉnh j sẽ đi qua trước đỉnh j sẽ là đỉnh nào
    - **Cập nhật ma trận mới** `khoangCach[i, j]`**:** Thực hiện n lần lặp trên ma trận trọng lượng mới `khoangCach[i, j]`**.** Duyệt qua lần lượt tất cả các cặp đỉnh i, j trong đồ thị và thử cập nhật giá trị của ma trận trong sao cho nếu có một đỉnh k nào đó mà đường đi từ i đến j thông qua k ngắn hơn so với đường đi trực tiếp từ i đến j, thì ta cập nhật giá trị đó. Mô tả bằng biểu thức như sau: `khoangCach[i, k] + khoangCach[k, j] < khoangCach[i, j]`. Nếu `khoangCach[i, k]` được cập nhật giá trị mới, thì phần tử `p[i, j]` của ma trận truy vết cũng được cập nhật giá trị là k, nghĩa là từ đỉnh i đến đỉnh j sẽ thông qua đỉnh k.
    - **Lặp lại quá trình:** Lặp lại quá trình cập nhật ma trận trọng lượng mới `khoangCach[i, j]` cho đến khi không còn bất kỳ cập nhật nào nữa. Khi không còn bất kỳ cập nhật nào được thực hiện, ta đã tìm được đường đi ngắn nhất giữa tất cả các cặp đỉnh trong đồ thị.

Yêu cầu 5: Tìm chu trình hoặc đường đi Euler

Định nghĩa:

- Chu trình Euler: Là chu trình đơn chứa tất cả các cạnh của đồ thị (nghĩa là chu trình đi qua mỗi cạnh của đồ thị đúng 1 lần).
- Đồ thị Euler: Là đồ thị có chu trình Euler.
- Đường đi Euler: Là đường đi đơn chứa mọi cạnh của đồ thị (nghĩa là đường đi qua mỗi cạnh của đồ thị đúng 1 lần).

- Đồ thị nữa Euler: Là đồ thị có đường đi Euler.

Tính chất (xét trên đồ thị vô hướng):

- Một đa đồ thị liên thông có chu trình Euler khi và chỉ khi mỗi đỉnh của nó đều có bậc chẵn.
- Một đồ thị vô hướng liên thông có đường đi Euler khi và chỉ khi nó có không quá 2 đỉnh bậc lẻ (nghĩa là chỉ được phép có 0 hoặc 2 đỉnh bậc lẻ). Khi đó đường đi Euler sẽ bắt đầu từ đỉnh bậc lẻ thứ nhất và kết thúc ở đỉnh bậc lẻ thứ 2.

Hướng tiếp cận:

- Kiểm tra đồ thị vô hướng, liên thông
- Đếm số lượng đỉnh bậc lẻ của đồ thị
- Nếu không có đỉnh bậc lẻ => Đồ thị Euler
    - Nhập đỉnh bắt đầu => chạy thuật toán tìm chu trình Euler
- Nếu có đúng 2 đỉnh bậc lẻ => Đồ thị nữa Euler
    - Nhập đỉnh bắt đầu
    - Nếu nhập khác 2 đỉnh bậc lẻ bên trên => Không có lời giải
    - Ngược lại => chạy thuật toán tìm đường đi Euler
- Ngược lại => Đồ thị không Euler

Phần 2: Hướng dẫn sử dụng chương trình

I. Mô tả các hàm chính của chương trình

0. Mô tả các hàm chính trong Class “Graph”

Tên hàm	**Chức năng**
SetFilePath	Thiết lập đường dẫn đọc file input
GetNumberVertex	Lấy số lượng đỉnh
GetNumberEdge	Lấy số lượng cạnh
GetArrayNumberAdjacentVertices	Lấy mảng số đỉnh kề (là cột đầu tiên trong file input)
GetArrayDegree	Lấy mảng bậc của đỉnh
GetNumberAdjacentVertices	Lấy số lượng đỉnh kề của 1 đỉnh bất kỳ được truyền qua tham số
GetAdjacencyMatrix	Lấy ma trận kề, tùy chọn hiển thị trọng số hoặc không
GetEdgeList	Lấy danh sách cạnh, mỗi cạnh được định nghĩa bằng struct có đỉnh đầu, đỉnh cuối và trọng số
GetAdjacencyList	Lấy danh sách đỉnh kề
GetAdjacencyEdgeList	Lấy danh sách cạnh kề
IsUndirectedGraph	Kiểm tra đồ thị là vô hướng hay không

IsGraphHasLoops	Kiểm tra có cạnh khuyên hay không
IsGraphHasParallel	Kiểm tra có cạnh bội (song song) hay không
IsWeightNegative	Kiểm tra có trọng số âm hay không
DFS	Duyệt đồ thị theo chiều sâu từ đỉnh u
GetNumberConnectedComponents	Lấy số thành phần liên thông của đồ thị hoặc kiểm tra đồ thị có liên thông hay không
GetArticulationPointList	Lấy danh sách các đỉnh khớp của đồ thị
GetBridgeList	Lấy ra danh sách các cạnh cầu của đồ thị

1. Yêu cầu 1: Nhận diện một số dạng đồ thị đặc biệt

Tên hàm	Chức năng
Run	Kiểm tra đồ thị truyền vào có thỏa điều kiện: Đồ thị vô hướng, không có cạnh bội, không có cạnh khuyên hay không? Và gọi đến các hàm bên dưới
WindmillGraph	Kiểm tra các điều kiện của đồ thị cối xay gió Wd(3,n) và in ra kết quả
BarbellGraph	Kiểm tra các điều kiện của đồ thị Barbell bậc k và in ra kết quả
K_PartiteGraph	Kiểm tra đồ thị k-phân, liệt kê và phân nhóm các đỉnh

2. Yêu cầu 2: Xác định thành phần liên thông mạnh

Tên hàm	Chức năng
Run	Kiểm tra đồ thị truyền vào có thỏa điều kiện: Đồ thị có hướng, không có cạnh bội, không có cạnh khuyên hay không? Khởi tạo các biến cần thiết và gọi đến các hàm bên dưới
SetAdjList_UUG_and_RG	Thiết lập danh sách kề của đồ thị vô hướng nền và danh sách kề của đồ thị ngược
GetListSCC	Lấy được danh sách các thành phần liên thông mạnh
GetGraphType	Kiểm tra đồ thị thuộc loại nào: Liên thông mạnh, liên thông từng phần, liên thông yếu hay không liên thông

InitializeStack	2 hàm để chạy thuật toán Kosaraju, tìm ra được các thành phần liên thông mạnh, được gọi trong hàm chính GetListSCC
DfsOnRG	

3. Yêu cầu 3: Tìm cây khung lớn nhất

Tên hàm	Chức năng
Run	Kiểm tra đồ thị truyền vào có thỏa điều kiện: Đồ thị vô hướng & liên thông không? Và gọi đến các hàm bên dưới
Prim	Chạy thuật toán Prim và in kết quả
Kruskal	Chạy thuật toán Kruskalvà in kết quả

4. Yêu cầu 4: Tìm đường đi ngắn nhất

Tên hàm	Chức năng
`YeuCau4.Run()`	- Gọi hàm đọc file và tạo đồ thị được truyền vào: `XuLyDoThi.DocDoThi();` - Gọi hàm để lấy được ma trận trọng lượng của đồ thị truyền vào: `XuLyDoThi.LayMaTranTrongLuong()` - Gọi hàm để kiểm tra ma trận trọng lượng có trọng số âm: `XuLyDoThi.KiemTraDoThiCoTrongSoAm()` - Gọi hàm tìm đường đi bằng thuật toán Floyd-Warshall và in ra màn hình `ThuatToanFloydWarshall.Run()`
`XuLyDoThi.DocDoThi(filePath1)`	- Gọi hàm `LuuTruDoThi.DocDoThi(filePath)`. - Ở đây truyền vào đường dẫn đến file .txt. Lưu ý rằng đường dẫn đang được là static link. Kiểm tra file có tồn tại không. Và trả về 1 thực thể đồ thị với các dữ liệu từ đồ thị như: số đỉnh, danh sách đỉnh kề của từng đỉnh và danh sách trọng số của từng đỉnh.
`XuLyDoThi.LayMaTranTrongLuong(doThi.Value)`	Truyền vào thực thể đồ thị với các dữ liệu từ đồ thị như: số đỉnh, danh sách đỉnh kề của từng đỉnh và danh sách trọng số của từng đỉnh như đã lấy ở trên. Từ danh sách đỉnh kề và danh sách trọng số lấy được ma trận trọng lượng (nếu không có cạnh giữa các đỉnh thì mặc định để trong số là `“1000000”` đại diện cho dương vô cùng).
`XuLyDoThi.KiemTraDoThiCoTrongSoAm(doThi.Value)`	Từ ma trận trọng lượng được lấy ở trên, kiểm tra có trọng số âm hay không. Nếu là ma trận

	trọng số âm thì trả về `true` nếu không thì trả về `false`.
`ThuatToanFloydWarshall.Run(MT_TrongLuong, doThi.Value.SoLuongDinh);`	Truyền ma trận trọng lượng và đồ thị ở trên, chạy thuật toán FloyWarshall và in kết quả ra màn hình: - Gọi hàm `FloydWarshall()`: Chạy thuật toán FloysWarshall - In kết quả ra màn hình: Với từng cặp đỉnh từ i đến j, gọi hàm `InDuongDiNganNhat()` với thông số truyền vào là đỉnh i, đỉnh j và `p[i, j]` để in ra đường đi. Nếu khoảng cách ngắn nhất từ đỉnh i đến j là "`1000000`" đại diện cho dương vô cùng thì in ra "`Khong co duong di`". Nếu tồn tại khoảng cách ngắn nhất thì in ra khoảng cách ngắn nhất ứng với từng đường đi.

5. Yêu cầu 5: Tìm chu trình hoặc đường đi Euler

Tên hàm	Chức năng
Run	Kiểm tra đồ thị truyền vào có thỏa điều kiện: Đồ thị vô hướng liên thông không? Và gọi đến các hàm bên dưới
Euler	Chạy thuật toán Euler và in ra đường đi Euler

II. Cách chạy chương trình

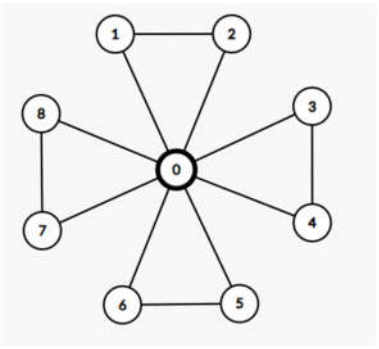
1. Cài đặt file input

- Đưa dữ liệu đầu vào vào trong 5 file định dạng ".txt" tương ứng với đề bài của 5 yêu cầu, có tên lần lượt:
  - MT1.txt
  - MT2.txt
  - MT3.txt
  - MT4.txt
  - MT5.txt
- Các file này được đặt trực tiếp trong ổ đĩa C (đường dẫn của file có dạng C:\MT1.txt).
- Người dùng có thể thay đổi đường dẫn mặc định trên bằng cách sửa file "Program.cs".

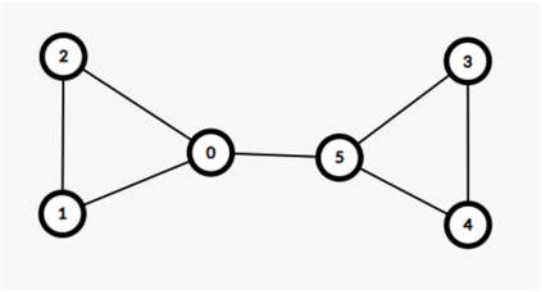
2. Dữ liệu mẫu:

Yêu cầu 1:

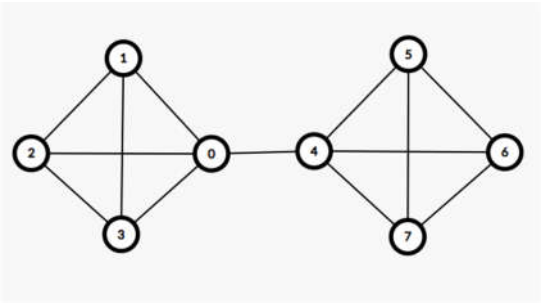
Mẫu 1A:

9 8 1 1 2 1 3 1 4 1 5 1 6 1 7 1 8 1 2 0 1 2 1 2 0 1 1 1 2 0 1 4 1 2 0 1 3 1 2 0 1 6 1 2 0 1 5 1 2 0 1 8 1 2 0 1 7 1	

Mẫu 1B:

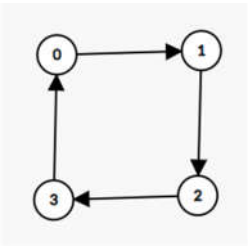
6 3 1 1 2 1 5 1 2 0 1 2 1 2 0 1 1 1 2 4 1 5 1 2 3 1 5 1 3 0 1 3 1 4 1	

Mẫu 1C:

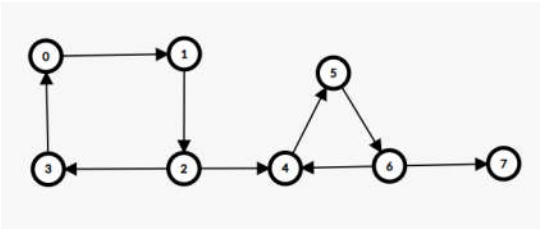
8 4 1 1 2 1 3 1 4 1 3 0 1 2 1 3 1 3 0 1 1 1 3 1 3 0 1 1 1 2 1 4 0 1 5 1 6 1 7 1 3 4 1 6 1 7 1 3 4 1 5 1 7 1 3 4 1 5 1 6 1	

Yêu cầu 2:

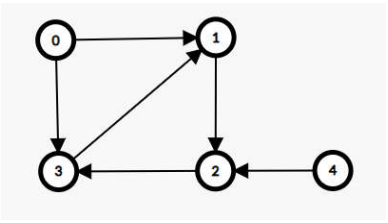
Mẫu 2A:

4 1 1 1 1 2 1 1 3 1 1 0 1	

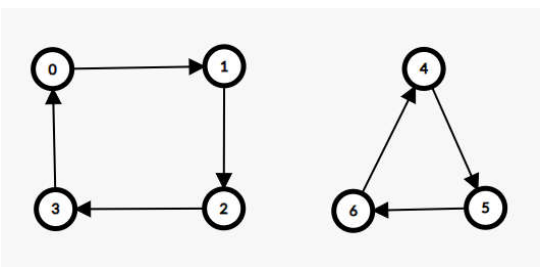
Mẫu 2B:

8 1 1 1 1 2 1 2 3 1 4 1 1 0 1 1 5 1 1 6 1 2 4 1 7 1 0	

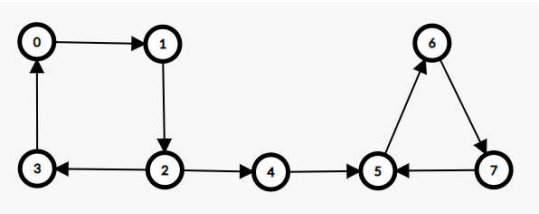
Mẫu 2C:

5 2 1 1 3 1 1 2 1 1 3 1 1 1 1 1 2 1	

Mẫu 2D:

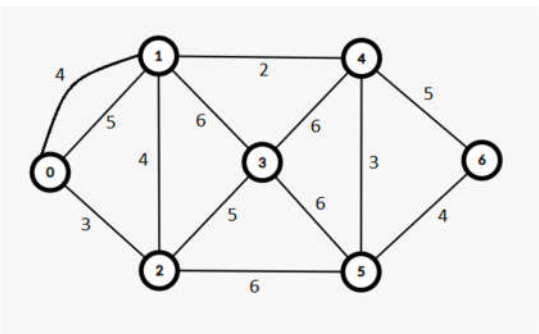
7 1 1 1 1 2 1 1 3 1 1 0 1 1 5 1 1 6 1 1 4 1	

Mẫu 2E:

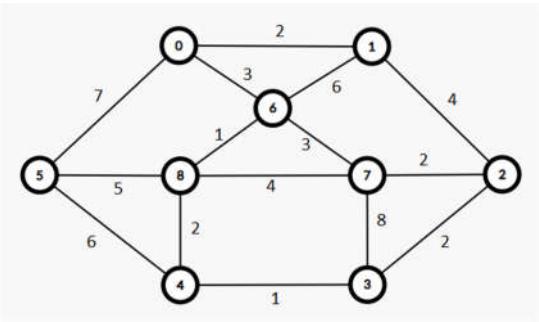
8 1 1 1 1 2 1 2 3 1 4 1 1 0 1 1 5 1 1 6 1 1 7 1 1 5 1	

Yêu cầu 3:

Mẫu 3A:

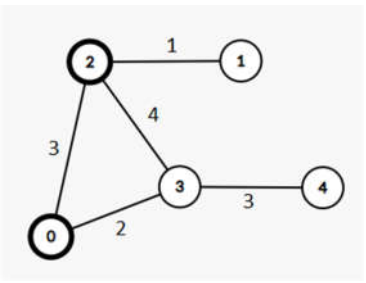
7 3 1 5 1 4 2 3 5 0 5 0 4 2 4 3 6 4 2 4 0 3 1 4 3 5 5 6 4 1 6 2 5 4 6 5 6 4 1 2 3 6 5 3 6 5 4 2 6 3 6 4 3 6 4 2 4 5 5 4	

Mẫu 3B:

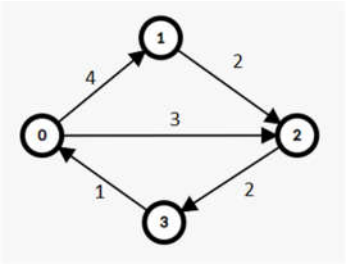
9 3 1 2 5 7 6 3 3 0 2 2 4 6 6 3 1 4 3 2 7 2 3 2 2 4 1 7 8 3 3 1 5 6 8 2 3 0 7 4 6 8 5 4 0 3 1 6 7 3 8 1 4 2 2 3 8 6 3 8 4 4 4 2 5 5 6 1 7 4	

Yêu cầu 4:

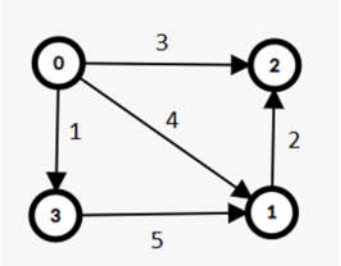
Mẫu 4A:

5 2 2 3 3 2 1 2 1 3 0 3 1 1 3 4 3 0 2 2 4 4 3 1 3 3	

Mẫu 4B:

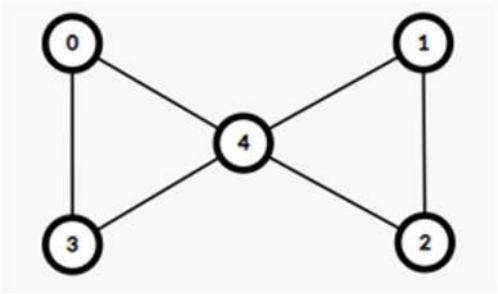
4 2 1 4 2 3 1 2 2 1 3 2 1 0 1	

Mẫu 4C:

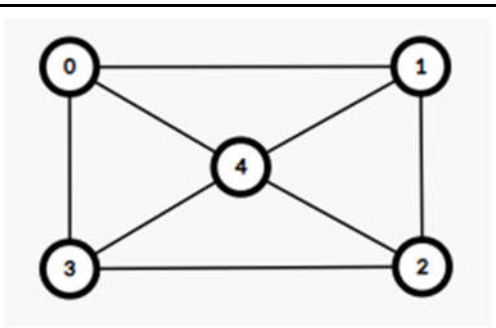
4 3 1 4 2 3 3 1 1 2 2 0 1 1 5	

Yêu cầu 5:

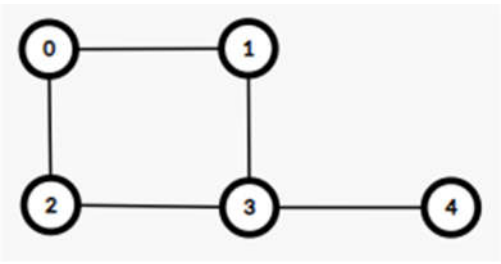
Mẫu 5A:

5 2 3 1 4 1 2 2 1 4 1 2 1 1 4 1 2 0 1 4 1 4 0 1 1 1 2 1 3 1	

Mẫu 5B:

5 3 1 1 3 1 4 1 3 0 1 2 1 4 1 3 1 1 3 1 4 1 3 0 1 2 1 4 1 4 0 1 1 1 2 1 3 1	

Mẫu 5C:

5 2 1 1 2 1 2 0 1 3 1 2 0 1 3 1 3 1 1 2 1 4 1 1 3 1	

Mẫu 5D:

5 5 0 1 1 1 1 1 1 1 4 1 6 0 1 0 1 0 1 2 1 3 1 4 1 5 1 1 2 1 3 1 3 1 3 1 5 1 1 2 1 2 1 2 1 4 1 3 0 1 1 1 3 1