BÀI TẬP TUẦN 9, 10

Môn học: Cấu Trúc Dữ Liệu và Giải Thuật.
GV hướng dẫn thực hành: Nguyễn Khánh Toàn (ktoan271199@gmail.com).
Nội dung chính:

- Đồ thị.

Thời gian thực hiện: **2** tuần.

Bài tập được biên soạn lại và tham khảo từ tài liệu thực hành môn Cấu Trúc Dữ Liệu và Giải Thuật của quý Thầy Cô Trợ Giảng khóa trước của bộ môn Khoa Học Máy Tính và quý Thầy Cô của bộ môn Công Nghệ Tri Thức. Trân trọng cảm ơn quý Thầy Cô của các bộ môn.

1 Đồ thị

1.1 Biểu diễn đồ thị

1.1.1 Đọc dữ liệu đồ thị

Tập tin **graph.txt** là tập tin chứa thông tin của một đồ thị. Tập tin có thể có nội dung như sau:

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

Hình 1: Tập tin đầu vào mô tả đồ thị.

Trong đó:

- Dòng đầu tiên chứa số nguyên n, cho biết số đỉnh của đồ thị.
- n dòng tiếp theo, mỗi dòng chứa n số nguyên ứng với các phần tử trong ma trận kề.

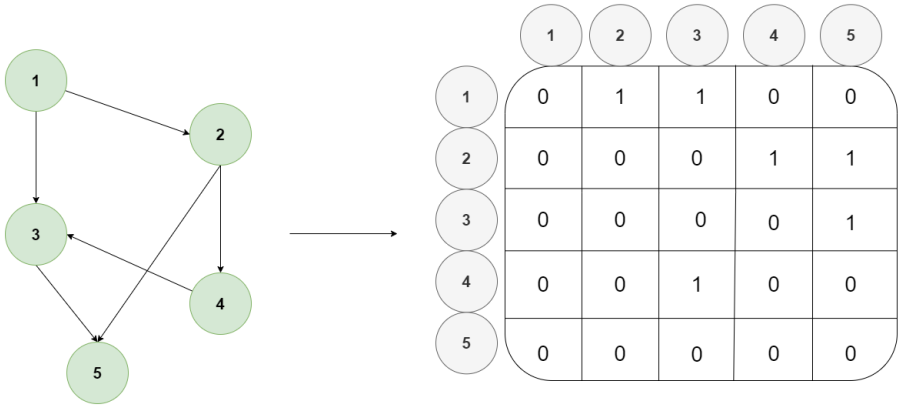
Sinh viên được yêu cầu đọc thông tin của đồ thị từ tập tin trên vào cấu trúc dữ liệu **Graph** và thực hiện một số thao tác trên đồ thị đó.

Lưu ý: Tập tin **graph.txt** sinh viên tự tạo ra theo mẫu trên.

1.1.2 Biểu diễn đồ thị

Có hai cách để các bạn cài đặt cấu trúc dữ liệu đồ thị:

- Sử dụng **ma trận kề** (Hình 2).



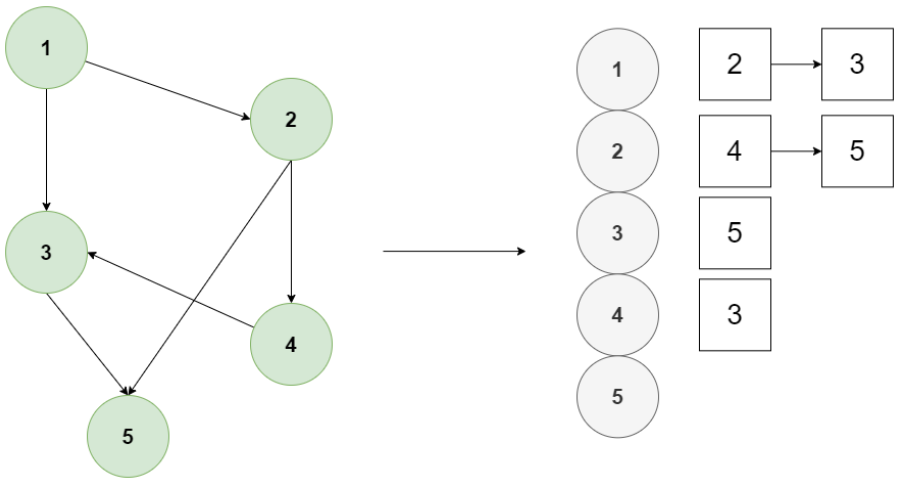
Hình 2: Biểu diễn đồ thị sử dụng ma trận kề.

Để biểu diễn đồ thị theo cách này, ta cần cài đặt cấu trúc **Graph** như sau (Hình 3):

```
#define MAX 100
struct Graph
{
    int num_vertices; // number of vertices
    int adjacency_matrix[MAX][MAX]; // adjacency matrix
};
```

Hình 3: Cài đặt đồ thị sử dụng ma trận kề.

- Sử dụng **danh sách kề**. (Hình 4).



Hình 4: Biểu diễn đồ thị sử dụng danh sách kề.

Để biểu diễn đồ thị theo cách này, ta cần cài đặt cấu trúc **Graph** như sau (Hình 5):

```
#define MAX 100
struct Graph
{
    int num_vertices; // number of vertices
    vector<int> adjacency_list[MAX]; // adjacency list
};
```

Hình 5: Cài đặt đồ thị sử dụng danh sách kề.

Không bắt buộc nhưng khuyến khích: Sinh viên cài đặt cấu trúc dữ liệu trên và hàm yêu cầu tương ứng sử dụng Lập Trình Hướng Đối Tượng (OOP).

1.2 Cài đặt các hàm chức năng

Các bạn chọn một cách thể hiện đồ thị trong hai cách trên và cài đặt các hàm chức năng của nó như sau:

1. Hàm tạo đồ thị từ tập tin:
   - **Graph** createGraphFromFile(const **string** &filename);
   - *Đầu vào*: filename là tập tin đầu vào chứa dữ liệu của đồ thị (trong bài tập này là file graph.txt do các bạn tự tạo).
   - *Đầu ra*: Đồ thị đọc từ tập tin, kiểu dữ liệu **Graph.**
2. Hàm hiển thị đồ thị dưới dạng danh sách kề (Hình 4)
   - **void** DisplayGraph(**Graph** g)
   - *Đầu vào*: g là đồ thị muốn hiển thị.
3. Kiểm tra đồ thị có phải là đồ thị vô hướng không?
   - **bool** IsUndirectedGraph(**Graph** g)
   - *Đầu vào*: g là đồ thị muốn kiểm tra.
   - *Đầu ra*: Trả về *true* nếu g là đồ thị vô hướng, *false* nếu ngược lại.
4. Đếm số lượng cạnh trong đồ thị:
   - **int** CountEdge(**Graph** g)
   - *Đầu vào*: g là đồ thị muốn thực hiện.
   - *Đầu ra*: số lượng cạnh trong đồ thị.
5. Liệt kê thông tin bậc đầu vào, đầu ra của các đỉnh trong đồ thị.
   - **void** listDegreeOfVertices(**Graph** g)
   - *Đầu vào*: g là đồ thị muốn thực hiện.
6. Duyệt đồ thị theo chiều sâu, in ra đường đi.
   - **void** DFS(**Graph** g, **int** start_vextex)
   - *Đầu vào*: g là đồ thị muốn thực hiện, start_vextex là đỉnh bắt đầu duyệt.
7. Duyệt đồ thị theo chiều rộng, in ra đường đi.
   - **void** BFS(**Graph** g, **int** start_vextex)
   - *Đầu vào*: g là đồ thị muốn thực hiện, start_vextex là đỉnh bắt đầu duyệt.
8. Kiểm tra đồ thị có chu trình hay không?
   - **bool** isCyclicGraph(**Graph** g)

- *Đầu vào*: g là đồ thị muốn kiểm tra.
- *Đầu ra*: Trả về *true* nếu g có chu trình, *false* nếu ngược lại.

Các bài tập dưới đây các bạn sẽ thực hiện ở tuần tiếp theo.

9. Kiểm tra đồ thị có liên thông không?
   - **bool** IsConnectedGraph(**Graph** g)
   - *Đầu vào*: g là đồ thị muốn thực hiện.
   - *Đầu ra*: Trả về *true* nếu g là đồ thị liên thông, *false* nếu ngược lại.
10. Đếm số thành phần liên thông của đồ thị.
    - **int** CountConnectedComponents(**Graph** g)
    - *Đầu vào*: g là đồ thị muốn thực hiện.
    - *Đầu ra*: Số lượng thành phần liên thông trong đồ thị.
11. Tìm đường đi ngắn nhất từ một đỉnh đến tất cả các đỉnh còn lại:
    - Thuật toán Dijkstra:
      - void FindShortestPathDijkstra(Graph g, int start_vertex, <tham số bổ sung tùy ý>)
      - *Đầu vào*: g là đồ thị muốn thực hiện, start_vextex là đỉnh bắt đầu duyệt.
      - *Đầu ra*: mảng thể hiện chi phí nhỏ nhất để di chuyển từ đỉnh đầu vào đến các đỉnh còn lại.
    - Thuật toán Floyd:
      - void FindShortestPathFloyd(Graph g, int start_vertex, <tham số bổ sung tùy ý>)
      - *Đầu vào*: g là đồ thị muốn thực hiện, start_vextex là đỉnh bắt đầu duyệt.
      - *Đầu ra*: mảng thể hiện chi phí nhỏ nhất để di chuyển từ đỉnh đầu vào đến các đỉnh còn lại.
    - Thuật toán Bellman:
      - void FindShortestPathBellman(Graph g, int start_vertex, <tham số bổ sung tùy ý>)
      - *Đầu vào*: g là đồ thị muốn thực hiện, start_vextex là đỉnh bắt đầu duyệt.
      - *Đầu ra*: mảng thể hiện chi phí nhỏ nhất để di chuyển từ đỉnh đầu vào đến các đỉnh còn lại.
12. Tìm cây khung nhỏ nhất của đồ thị (*):
    - Thuật toán Prim:
      - long long prim(Graph g, int x, <tham số bổ sung tùy ý>)

  - *Đầu vào*: g là đồ thị muốn thực hiện, x là đỉnh bắt đầu cây khung.
  - *Đầu ra*: Chi phí nhỏ nhất của cây khung.
- Thuật toán Kruskal:
  - long long kruskal(Graph g, <tham số bổ sung tùy ý>)
  - *Đầu vào*: g là đồ thị muốn thực hiện.
  - *Đầu ra*: Chi phí nhỏ nhất của cây khung.
- Link tham khảo cho các bạn về các thuật toán này: Dữ liệu

Hết