Báo cáo: Tô màu đồ thị sử dụng thuật toán DSatur

Nguyễn Bùi Tuấn Linh, Nguyễn Bùi Việt Linh

Ngày 6 tháng 6 năm 2023

1 Yêu cầu bài toán

Tô màu đồ thị sao cho không có hai đỉnh kề nhau cùng màu

Input: Từ file dothi.txt trong đó

- Dòng đầu tiên ghi hai số nguyên n và m là số đỉnh và số cạnh của đồ thị;
- Ở m dòng tiếp theo, mỗi dòng ghi 2 số nguyên tương ứng với một cạnh của đồ thị.

Output: File dothitomau.txt với

- Dòng đầu tiên là số màu tối đa bạn dùng để tô (các màu đánh số từ 1,2,...);
- Ở n dòng tiếp theo, dòng thứ i là màu để tô đỉnh i

2 Thuật toán DSatur

Thuật toán DSatur là một thuật toán tô màu đồ thị chọn đỉnh tiếp theo để tô màu dựa trên chỉ số bão hoà (saturation degree).

2.1 Cấu trúc dữ liệu đồ thị

Đầu tiên, chương trình khởi tạo đồ thị bằng cách đọc thông tin từ tệp đầu vào. Đồ thị được biểu diễn bằng một mảng các đỉnh (vertices), trong đó mỗi đỉnh chứa thông tin về chỉ số, màu, số lượng đỉnh kề, danh sách đỉnh kề và chỉ số bão hoà.

```
struct Vertex {
    int index;
    int color;
    int numNeighbors;
    struct Vertex** neighbors;
    int saturation;
};
```

2.2 Tô màu đồ thị

Chương trình sử dụng thuật toán DSatur để tô màu đồ thị. Quá trình tô màu được thực hiện theo các bước sau:

1. Khởi tạo số lượng màu sử dụng (numColors) ban đầu là 0.
2. Lặp cho đến khi tất cả các đỉnh đều được tô màu:
   (a) Chọn đỉnh tiếp theo để tô màu bằng cách tìm đỉnh có chỉ số bão hoà (saturation) lớn nhất.
   (b) Tô màu cho đỉnh được chọn bằng cách chọn một màu chưa được sử dụng bởi các đỉnh kề.
   (c) Nếu màu được chọn có chỉ số lớn hơn numColors, cập nhật numColors.
   (d) Cập nhật chỉ số bão hoà của tất cả các đỉnh.
3. In kết quả tổng số màu sử dụng và màu được gán cho mỗi đỉnh.

3 Các hàm chính

3.1 `createVertex(int index, int numVertices)`

Hàm này tạo một đỉnh mới và cấp phát bộ nhớ cho các trường dữ liệu của đỉnh.

Đầu vào:

- `index` (int): Chỉ số (index) của đỉnh.
- `numVertices` (int): Số lượng đỉnh của input.

Đầu ra:

- `vertex` (Vertex): Đỉnh mới đã được tạo.

Cách hoạt động của hàm:

1. Tạo một biến đỉnh (`vertex`) và cấp phát bộ nhớ cho nó .
2. Gán giá trị của `index` cho trường `index` của `vertex`.
3. Khởi tạo trường `color` của `vertex` là -1 (đại diện cho màu chưa được gán).
4. Khởi tạo trường `numNeighbors` là 0 (đại diện cho chưa có số lượng đỉnh kề)
5. Tạo một mảng `'neighbors'` chứa các đỉnh kề của đỉnh hiện tại.
6. Khởi tạo trường `saturation` là 0 (đại diện cho độ bão hòa bằng 0.
7. Hàm trả về đỉnh mới được tạo.

3.2 `updateSaturation(struct Vertex** vertices, int numVertices)`

Hàm này được sử dụng để cập nhật chỉ số bão hoà (saturation degree) của tất cả các đỉnh.

Đầu vào:

- `vertices` (struct Vertex**): Danh sách các đỉnh của đồ thị.

Cách hoạt động của hàm:

1. Kiểm tra xem đỉnh đã tô màu chưa, chỉ xử lí các đỉnh chưa tô màu
2. Khởi tạo mảng `'usedColors'`, duyệt qua tất cả các đỉnh kề của đỉnh hiện tại, nếu đã tô màu, đánh dấu vào mảng `'usedColors'`.
3. Đếm số lượng màu đã sử dụng trong mảng `'usedColors'` (độ bão hòa).
4. Cập nhật độ bão hòa cho phần từ `vertex`.

Quá trình này đảm bảo cập nhật đúng chỉ số bão hoà cho mỗi đỉnh trong đồ thị.

3.3 `findNextVertex(struct Vertex** vertices, int numVertices)`

Hàm này được sử dụng để tìm đỉnh tiếp theo để tô màu dựa trên chỉ số bão hoà.

Đầu vào:

- `vertices` (struct Vertext**): Danh sách các đỉnh của đồ thị.
- `numVertices` (int): Số lượng đỉnh của input

Đầu ra:

- `nextVertex` (Vertex): Đỉnh được chọn để tô màu tiếp theo.

Hàm này hoạt động như sau:

1. Khởi tạo một biến `maxSaturation` và gán giá trị ban đầu là -1 (đại diện cho giá trị không hợp lệ).
2. Lặp qua danh sách các đỉnh và kiểm tra xem chỉ số bão hoà của đỉnh chưa tô màu đó có lớn hơn `maxSaturation` hay không.
3. Nếu có, cập nhật `maxSaturation` và gán đỉnh đó cho biến `nextVertex`.
4. Trả về `nextVertex` được chọn.

3.4 `graphColoring`

Hàm này được sử dụng để thực hiện việc tô màu đồ thị

Đầu vào:

- **'struct Vertex** vertices'**: Một mảng các con trỏ tới cấu trúc '**Vertex**' đại diện cho các đỉnh trong đồ thị.
- **'int numVertices'**: Số lượng đỉnh trong đồ thị.

Đầu ra:

- Kết quả được ghi vào tệp "dothitomau.txt".
- Tệp văn bản chứa các dữ liệu sau:
  - Dòng đầu tiên: Số lượng màu sử dụng để tô màu các đỉnh.
  - Các dòng tiếp theo: Màu của từng đỉnh trong đồ thị.

Cách hoạt động của hàm:

1. Khởi tạo biến `'numColors = 0'` đại diện cho số màu được sử dụng để tô màu các đỉnh.
2. Lặp lại cho đến khi không còn đỉnh chưa được tô màu
   - Sử dụng hàm `'findNextVertex'` để tìm đỉnh chưa được tô màu có độ bão hòa cao nhất.
   - Nếu không tìm thấy đỉnh thỏa mãn, thoát khỏi vòng lặp.
   - Nếu tìm thấy đỉnh thỏa mãn:
     - Duyệt qua tất cả đỉnh kề với đỉnh hiện tại.
     - Tô màu cho đỉnh hiện tại màu nhỏ nhất có thể.
3. Ghi kết quả vào tệp văn bản "dothitomau.txt":
   - Mở tệp văn bản để ghi kết quả.
   - Ghi số lượng màu `numColors` vào dòng đầu tiên của tệp.
   - Ghi màu của từng đỉnh vào các dòng tiếp theo của tệp.

4 Độ phức tạp

4.1 Độ phức tạp thời gian

Độ phức tạp thời gian của thuật toán DSatur là $O(|V|^2)$, trong đó $|V|$ là số lượng đỉnh của đồ thị. Quá trình tô màu đòi hỏi duyệt qua tất cả các đỉnh và các đỉnh kề, do đó có thể có tối đa $|V|$ lần duyệt và xử lý. Mỗi lần duyệt, ta phải tìm đỉnh có chỉ số bão hoà lớn nhất, điều này có độ phức tạp tuyến tính $O(|V|)$. Vì vậy, tổng độ phức tạp là $O(|V|^2)$.

4.2 Độ phức tạp không gian

Độ phức tạp không gian của thuật toán DSatur phụ thuộc vào số lượng đỉnh và số lượng cạnh của đồ thị. Trong trường hợp xấu nhất, khi đồ thị là đồ thị đầy đủ, số lượng cạnh là $|V|(|V|-1)/2$, và độ phức tạp không gian của thuật toán là $O(|V|^2)$, vì ta cần lưu trữ thông tin về mỗi đỉnh và mỗi cạnh. Tuy nhiên, trong trường hợp trung bình, khi số lượng cạnh ít hơn, độ phức tạp không gian có thể thấp hơn $O(|V|^2)$.