Data Structures and Algorithms II

Assignment 4

Wachmann Elias

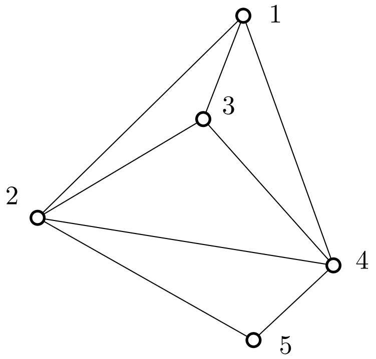
January 23, 2022

1 Task Description

Triangulation 3-coloring

Your are given a triangulation of a point set. Your task is to design an efficient algorithm that constructs a valid 3-coloring of the points of the triangulation or determines that such a 3-coloring does not exist. A 3-coloring of the points is valid if any two points that are connected with an edge have different colors. The n points of the triangulation are labeled with the integers $\{1, \ldots, n\}$. The triangulation is given by a list of edges with additional triangle points (see Figure 1 for an example):

- Every edge is given by the labels of its two end points (first the smaller point label, then the larger one).
- For every edge, the labels of the point(s) with which the edge forms a triangle (a bounded triangular face) in the triangulation is given (two labels for interior edges and one label for edges on the boundary of the convex hull).



edges	**triangle points**
$1-2$	3
$2-4$	$3, 5$
$3-4$	$1, 2$
$1-3$	$2, 4$
$4-5$	2
$2-5$	4
$2-3$	$1, 4$
$1-4$	3

Figure 1: Example of a triangulation and a list of its edges with triangle points.

Explain and describe your algorithm in detail, analyze its runtime and memory requirements, and give reasons for the correctness of your solution.

2 Description of algorithm

Note: It is assumed from the example, that the edges are always given with the lower vertex number in the first place e.g. 1-5 instead of 5-1. If this is not a given, the edge representation could be changed to conform to the above constraint in linear time, thus not increasing the asymptotic runtime.

General: The algorithm takes a list of edges with additional triangle points `EDGES` (as described above) as input and outputs a 3-coloring of the graph `COLORS` if it is possible. If no valid 3-coloring of the given graph exists, the algorithm exits and indicates that no such coloring is possible (`COLORS` is not returned in this case).
The algorithm starts with the **Setup** step, then the Points on the first edge and it's triangle points are colored in **Start**, after which **Loop** takes care of the remaining Points. Afterwards the validity of the coloring is checked.

1. **Setup**:
   First get number of Points. The Maximum is given, by the maximum (number) in triangle points. Use this number to create an array `COLORS` with this size for the colors of the vertices. Store edges (keys) and triangle points (values) in a hashmap `EDGE_DIC` for fast access. Setup counter `counter` which tracks how many vertices need to be colored in. Setup empty queue `NEXTEDGES` for neighboring edges.

2. **Start**:
   An arbitrary start edge, for example the first in `EDGES` is chosen as `cur_edge`. The two endpoints p_1 and p_2 of `cur_edge` are colored with c_1 and c_2 respectively. Then the triangle points are colored [1] in. Finally the `counter` is decreased by the number of vertices which were colored in and `cur_edge` is deleted from `EDGE_DIC`.

3. **Loop**: The Loop is executed while `counter` is not 0.
   All neighbors of `cur_edge` which were not visited before (meaning they were never `cur_edge`) are added to the queue `NEXTEDGES`. Neighbors to the `cur_edge` are edges which form a triangle with the `cur_edge` if the two endpoints of the so given path are connected. [2] Now `cur_edge` is set to the last edge in `NEXTEDGES` (according to the FIFO-principle). The colors c_1 and c_2 of p_1 and p_2 of `cur_edge` are checked. If they are the same, algorithm terminates because no valid 3-coloring of the given graph exists. Otherwise the algorithm tries to color the triangle points of `cur_edge` (at least one [outer edge] and at most 2 triangle points for inner edges). If the triangle point is already colored with c_3, it is check if the color is neither c_1 nor c_2, if the color would match either c_1 or c_2 the algorithm terminates as in the aforementioned case c_1 equals c_2. If the the triangle point is properly colored nothing happens and if applicable the second triangle point is checked. In the case that the triangle point

[1] Depending on the edge type either one (outer edge e.g. 1-2 in Figure 1) or a maximum of two triangle points (inner edge e.g. 1-3 in Figure 1) are colored in.

[2] This definition of neighbors guarantees that the so formed triangle already has two colored points, thus making the coloring of the third trivial.
Example: The edge 1-2 in Figure 1 has 2 neighbors (1-3 and 2-3) according to the above definition, whereas an inner edge has 4: e.g. 3-4 has 1-3, 1-4, 2-3 and 2-4 as neighbors

isn't colored already, it is colored with c_3 such that c_3 is different than c_2 and c_1. Finally the `counter` is decreased by the number of vertices which were colored in and `cur_edge` is deleted from `EDGE_DIC`. → Loop

4. **Check validity of coloring of remaining edges**:
   For the graph in Figure 1 **without Point 5** (without edges 2-5 and 4-5) the algorithm could produce a impossible coloring and end. [3] To avoid these edge cases one could check all edges and their triangle points for matching colors as described in 'Loop' above. However color-checking only the remaining edge-triangle-point-pairs in `EDGE_DIC` is sufficient, because these are the edges which were never `cur_edge`. Asymptotically this reduction in remaining edge-triangle-point-pair color checks doesn't matter as we'll see below.

Jede Edge abchecken, damit man kein Dreieck übersieht.

1) Wähle beliebigen Startpunkt -> Färbe initial erste Edge Loop:
2) checke triangle points von current edge -> färbe falls keine farbe -) wenn triangle points farben falsch –> ABBRUCH
3) Auswahlregel für nächste Edge: Aus dem Stack nehmen wenn Stack nicht leer. Sonst Minimalste Edge nehmen z.B.: 2-4 3 (2-3) wählen restliche Edges in einen Stack falls mehrere möglich sind (Somit kann gewährleistet werden, dass jede Edge angeschaut wird und somit kein Dreieck übersehen wird.)

3 Korrektheit des Algorithmus

TODO

[3] The `counter` would reach 0 if the start edge would be 1-3 (The loop would be skipped and thus no checks would be done)