Computing Techniques
Escola Politècnica Superior, University of Lleida
Fall 2022
Jesús Ojeda

Assignment 2 – Radiance and Color

1 Introduction

Lets tackle a couple of problems that, among other fields, have an application to vision. And lets do it in style, using the algorithmic paradigms that we have seen in the theory sessions.

1. When analysing images, one usual thing to do is to locate the most brilliant areas, those that receive most radiance and thus will have greater values. Depending on the problem at hand, this can be used to locate some artifact in the image to compute its area, for example (see Figure 1).

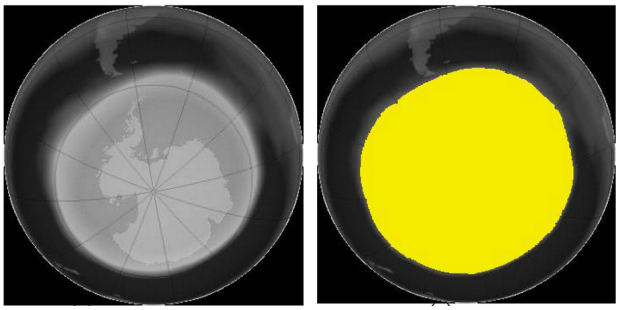

Figure 1: An image of the Ozone Hole over Antarctica (left), the segmentation of the hole by selecting the brightest pixels in order to compute the area of the hole (right).

2. In segmentation contexts, when objects within images have been found, in order to ease visual inspection for the engineers, each region is colored with a different color (see Figure 2).

Figure 2: Autonomous driving cars have to *see* their environment, in order to decide what to do (steer, brake, etc.). They detect each part of the image and label it as person, car, etc. To debug these systems, the developers usually assign a different color to each detected part.

2 Problems

2.1 Computing high intensity areas in images

The problem in the original description in the introduction would have required to have one value per each pixel of the image (so black and white images), representing the radiance in a common image type using for example 1 byte per pixel (values between 0 and 255). Then we should transform the image to have values positive and negative (values between -127 and 128, for example). And then we should work it out in two dimensions.

In this assignment we will simplify the problem a little bit. First, we will work only on the 1D problem. Furthermore, we will assume that the data is already in the whole integer realm (so we have already positive and negative values, but can't assume the sequence has any of them... maybe all values are negative or positive or are mixed).
In particular, you have to consider that the input for your algorithm is a sequence of length n of integer values.
Then, the objective is to find the **contiguous sub-sequence** within the original input, such that **the sum of its elements has the maximum possible value**.

As an example, lets assume we have as input the sequence `[2, -4, 1, 9, -6, 7, -3]`. The contiguous sub-sequence that has a maximum value is the one from index 2 to index 5, and resulting sum of the elements is 11. We want our function to return those indices and the value of the sum.

This problem can be solved in multiple ways, but in this assignment we will solve it twice:

- **A Divide and Conquer solution** (with cost $\mathcal{O}(n \log n)$). Implement your solution in the function `maxsum_subseq_dnc(lst)`.
- **A Dynamic Programming solution** (the best solution can achieve a cost $\mathcal{O}(n)$). Implement your solution in the function `maxsum_subseq_dp(lst)`.

2.2 Coloring an image segmentation

To compute the required colors to assign to each detected object in an image, a graph representation is used. The graph is a mathematical data structure with vertices or nodes, and edges that connect those vertices. The vertices may represent real-world objects, and the edges represent an existing relation between the objects.
Getting back to our context, in our graph each vertex will represent a detected object in the image, and an edge will connect two vertices such that the objects represented by the vertices are touching in the image (see Figure 3).
So, the problem consists on assigning different colors to vertices that share an edge [1]. The naive solution would be to assign a different color per each vertex, such that, globally, all vertices would be different. But we want to be a little bit more intelligent about the problem and try to assign a lower number of colors that vertices exist. There are ways to compute the minimum number of colors needed, but that is more difficult. Instead we

[1] You may think that this is an artificial problem, but this very same problem is solved when assigning the radio-frequencies used by the cellphone towers, the radio and tv broadcasting towers, etc. – We don't want two towers that have an intersected area of influence with the same radio-frequency because that would cause interference in the communications...

will be happy with a not-optimal-but-not-bad solution using the greedy and backtracking paradigms.

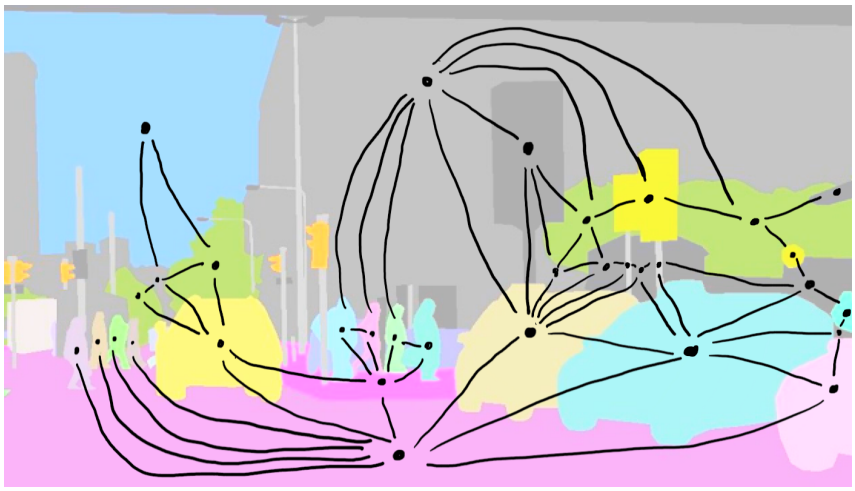

Figure 3: Partial graph (there are some vertices and edges missing) representing the connections between touching parts of the segmented image. Each object is represented by a vertex in the graph, and each edge connects two vertices if and only if the two objects represented by them share a common frontier.

So, in this assignment we will solve the problem twice:

- **A Greedy solution**. If the graph has maximum degree k [2], we know that with a greedy algorithm we will always have a possible (not necessarily optimal) solution if we have $k + 1$ possible colors. The algorithm should compute first the maximum degree of the graph (as to know the numbers of color possibly needed), and then try to assign a color to each node in the graph, greedily. Implement your solution in the function `coloring_greedy(graph)`.
- **A Backtracking solution**. Lets spice the problem a little bit. With backtracking we are searching for a solution in a brute-force orderly way. Lets add an additional input parameter n that specifies the number of colors to be used. The algorithm will return a color assignment (if there is one) or an empty list. Implement your solution in the function `coloring_backtrack(graph, n)`.

To ease your work, a `Graph` class will be provided. This class will be initialized from a list of tuples like

```
[(0, 1), (0, 4), (0, 5), (4, 5), (1, 4), (1, 3), (2, 3), (2, 4)]
```

This list represents the edges that connect each pair of vertices, while each vertex is represented by a number. This means that, implicitly, the greatest number (+1) in this data structure corresponds to the number of vertices that the graph will have. In particular, the previous list corresponds to the graph from Figure 4.

[2]The maximum degree of a graph is the degree of the vertex in the graph that has the maximum number of edges connected to it. And the degree of a vertex is precisely the number of edges connected to it.

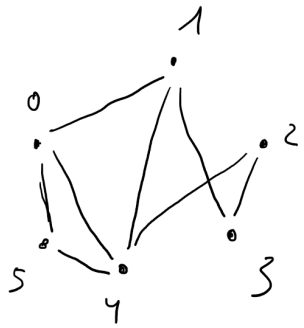

Figure 4: A graph.

Then, from the graph instance and for a given vertex, you will be able to get the list of adjacent vertices.

3 Code starting point

The provided code for this assignment is just one Python file:

paradigms.py – This code defines the signature for the functions you have to implement, as well as the provided util code, like the `Graph` class used in Problem 2.2. **You will have to complete your code in this file** and can write any additional functions that you may need.

4 Scores per task

- **2pt** Correctly implement the maxsum_subseq with divide and conquer.
- **3pt** Correctly implement the maxsum_subseq with dynamic programming.
- **2pt** Correctly implement the graph_coloring with a greedy approach.
- **2pt** Correctly implement the graph_coloring with the backtracking approach.
- **1pt** Comment the code (a doc-string per function plus special comments through the algorithms).

5 Delivery

When? We will work through this assignment in the PRALAB sessions and the delivery date is due Thursday, 22nd of December of 2022, at 23:55. You have this info in the Virtual Campus.

What? The **paradigms.py** Python file with your code, thoroughly commented. It will be tested against **Python 3.7 or greater**, be sure to use an appropriate version!

Additionally: This assignment can done individually or in pairs. It accounts for 25% of the final score of the course. Be extra careful **including the names of the participants as a comment in the first line of the uploaded file**.