Date handed out: 28 December 2020, Monday
Date submission due: 7 January 2020, Thursday 23:59

Assignment 3: A Simple Scanpath Analyser

This assignment aims to help you practice ***graph data structure*** *and* ***basic graph operations***. Your main task in this assignment is to develop a simple scanpath analyser using **C programming language**.

Overview

Eye tracking does not only allow us to understand which areas of interest (AOIs) of visual stimuli take the most attention, but also allows us to understand in which order these AOIs are used. Figure 1 shows an example of an eye movement sequence (i.e. scanpath) of a particular person on the home page of the Babylon website. The rectangles show the AOIs of the web page, and the circles show where the person looked at. The radius of each circle is directly proportional to the duration of its corresponding fixation, and the numbers in the circles show the sequence of the fixations.

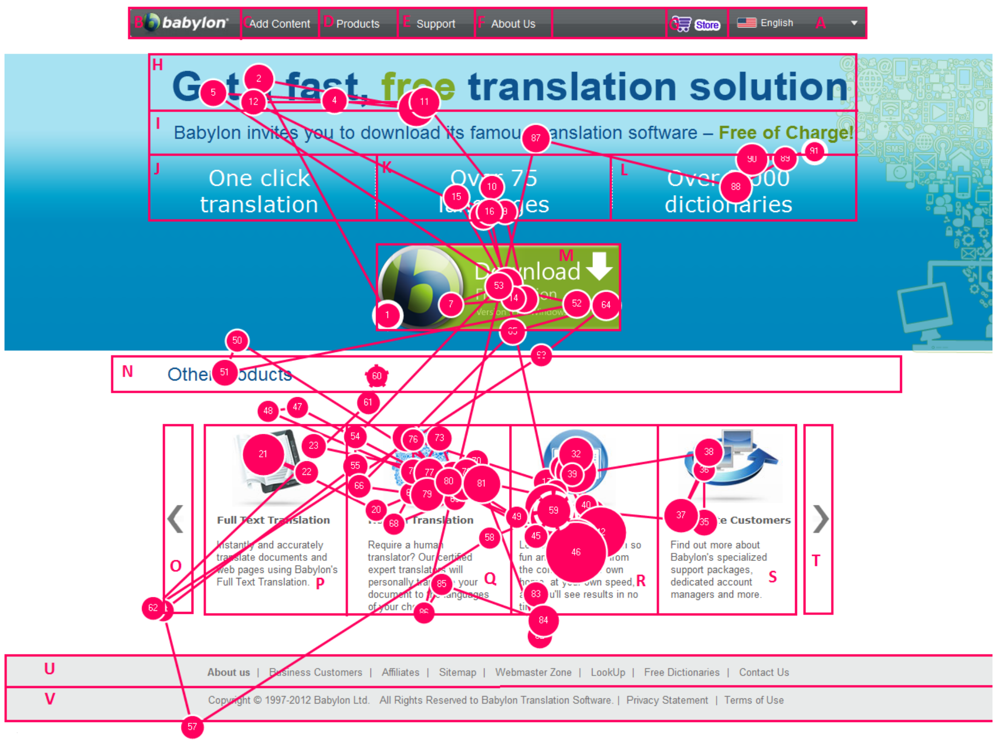
babylon
Add Content
Products
Support
About Us
Store
English
Get fast, free translation solution
Babylon invites you to download its famous translation software – Free of Charge!
One click translation
Over 75 languages
Over 1,000 dictionaries
Download
Other Products
Full Text Translation
Translation
Customers
About us | Business Customers | Affiliates | Sitemap | Webmaster Zone | LookUp | Free Dictionaries | Contact Us
Copyright © 1997-2012 Babylon Ltd. All Rights Reserved to Babylon Translation Software. | Privacy Statement | Terms of Use
A
B
C
D
E
F
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V

Figure 1. A scanpath of a single user on the home page of the Babylon website

Scanpaths are typically analysed based on the AOIs of visual stimuli. They are first represented as string sequences. For example, if a person looks at the AOIs M, P, Q, R and S respectively, his/her scanpath is represented as MPQRS.

In this assignment, you will need to develop a simple scanpath analyser that supports the following functionalities.

1. Read a number of scanpaths which are represented as string sequences from a text file
2. Compute the similarities between these scanpaths

3. Create an undirected weighted graph where a vertex is created for each scanpath and an edge is created between two vertices when the corresponding scanpaths have at least 50% similarity. The weight of the edge will be equal to the similarity score between two scanpaths. An example of a graph is shown below where Scanpath 1 and Scanpath 4 have 57.1% similarity, Scanpath 2 and Scanpath 3 have 71.4% similarity, and Scanpath 3 and Scanpath 5 have 60% similarity.

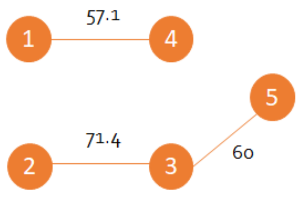

Figure 2. A graph with five scanpaths where similar ones are connected

4. Print similar scanpaths in descending order based on their similarity scores as follows:

```
Scanpath 2 and Scanpath 3 – Similarity: 71.4%
Scanpath 3 and Scanpath 5 – Similarity: 60%
Scanpath 1 and Scanpath 4 – Similarity: 57.1%
```

Process Model

The process model of this program is as follows (see Figure 3):

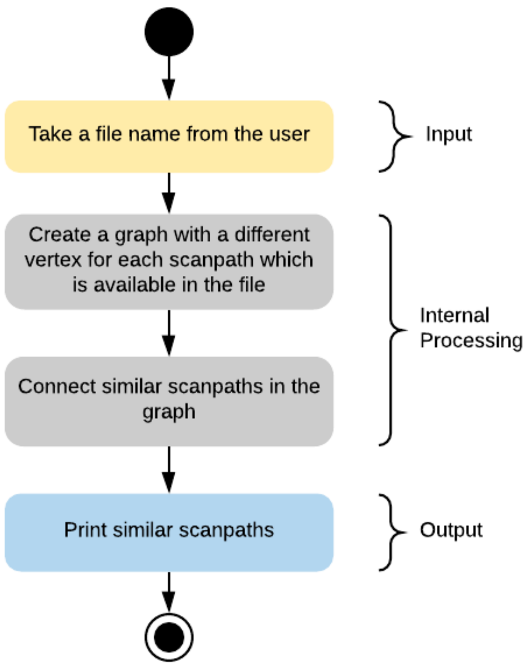
Take a file name from the user
Input
Create a graph with a different vertex for each scanpath which is available in the file
Connect similar scanpaths in the graph
Internal Processing
Print similar scanpaths
Output

Figure 2. Process model

1. Input

This program will have a single input which is the name of the file containing multiple scanpaths represented as string sequences, such as MVQARS and BFMCQRS. You can assume that the maximum length of each scanpath is 50. The input should be taken as follows:

```
Enter a file name > scanpaths.txt
```

2. Internal Processing

Once the name of the file is received from the user, the internal processing will be performed in two main steps to create a graph. You will need to implement a separate function for each of these steps.

- **createVertices:** This function takes the name of the file. It reads the string representation of the scanpaths from the file, and creates a graph with a different vertex for each scanpath. This function returns the graph without any edges. **The adjacency list method must be used for graph representation.**

- **createEdges:** This function takes the graph created in **createVertices**. It computes the similarity between each pair of the scanpaths. If two scanpaths have at least 50% similarity, the corresponding vertices will be connected with an undirected weighted edge where the weight will be the similarity score. The function returns the updated graph.

  The edit distance algorithm will be used to compute the similarity between two scanpaths. This algorithm is used to compute the minimum number of editing operations (insertion, deletion, and substitution) to transform one string into another one. For example, the distance between ABCD and ABCX is computed as 1 (one) because only one substitution operation can transform ABCD into ABCX or vice versa. Since scanpaths are represented as string sequences in our case, the similarities between these scanpaths can be computed with the edit distance algorithm. The C code of this algorithm is provided below where m is the length of s1 and n is the length of s2.

```
int editDist(char s1[], char s2[], int m, int n) {
    if (m == 0)
        return n;
    if (n == 0)
        return m;
    if (s1[m - 1] == s2[n - 1])
        return editDist (s1, s2, m - 1, n - 1);
    return 1 + min(editDist(s1, s2, m, n - 1), editDist(s1, s2, m - 1, n),
editDist(s1, s2, m - 1, n - 1));
}
```

  The edit distance can then be used to compute the similarity score between two string sequences as follows where S, d, and n represents the similarity score out of 100, the edit distance, the number of characters in the longer string respectively.

```
S = 100 x (1 – (d/n))
```

3. Output

Once the graph is constructed, the pairs of similar scanpaths are printed in descending order based on their similarity scores, as shown in Item 4 in Overview. For this operation, you will need to implement a function called **printSimilarScanpaths** that will take the graph and print similar scanpaths in the format demonstrated above.

Incremental and Modular Development

You are expected to follow an incremental development approach. Each time you complete a function or module, you are expected to test it to make sure that it works properly. You are also expected to follow modular programming which means that you are expected to divide your program into a set of meaningful functions.

Professionalism and Ethics

You are expected to complete the assignments on your own. Sharing your work with others, uploading the assignment to online websites to seek solutions, and/or presenting someone else's work as your own work will be considered as cheating.

Rules

- You need to write your program by using **C programming language.**
- You need to name your file with **your student id**, e.g. 1234567.c, and submit it to ODTUCLASS.
- **Code quality, modularity, efficiency, maintainability, and appropriate comments** will be part of the grading.

Grading Policy

The assignment will be graded as follows:

Grading Item	**Mark (out of 100)**
Graph Representation	15
Main Function – coordinate the function calls	10
createVertices	25
createEdges	25
printSimilarScanpaths	25