Tarea 3 - Graphs

CC3101: Discrete Mathematics for Computer Science

Submitted To:
Pablo Barcelo

Submitted By :
Alexandre Poupeau

June 23, 2019

Exercice 1

1.1) Demonstrate that if G is a oriented and strongly connected graph where $\forall u \in V, in(u) = out(u)$ then G has a eulerian circuit. In order to demonstrate that property, we need to firstly show that $\forall u \in V$ there is a way to go from u to u. Let us choose an adjacent node v of u connected by (u,v) and remove this edge. Then let us choose another adjacent node x of v connected by (v,x) and remove this edge again. Doing this double step we have that all nodes p in G' do respect the condition $in(p) = out(p)$ except our first node u ($in(u) = out(u) + 1$ and x, $in(x) + 1 = out(x)$. Doing that on the finite graph we will eventually reach the node u since as it had at least one out-edge, it has one in-edge.

In order to understand the fact that we can create a eulerian circuit from our node u we have to visualize the fact that the graph G is a set of connected oriented cycles. Therefore the idea is to start form u, each time we use a edge, we delete it. When we find a node p such that $in(p) = out(p) > 1$, we pass through all the edges formed by the cycle structure from p until we go back to p where now $in(p) = 0, out(p) = 1$. Since the graph is strongly connected we can not end up on a way where we can not go back. We do that on all the graph and we deleted all edges when we finally u.

Thus if G is a oriented and strongly connected graph where $\forall u \in V, in(u) = out(u)$ then G has a eulerian circuit.

I had another more straightforward idea so I write it here but I am not that convinced of the demonstration. Since $\forall u \in V, in(u) = out(u)$ then we can deduce the $\forall u \in V, degree(u) = in(u) + out(u) = 2in(u) = 2out(u)$ which means all nodes have an even degree. We know by definition that G has a eulerian circuit.

1.2) Model the problem given a k using a graph G_k. In order to visualize how to construct the G_k we are going to start with the graph G_2.

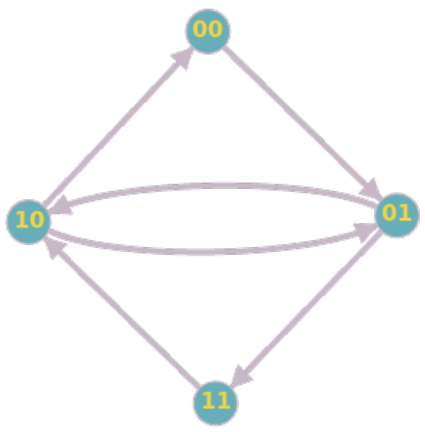

Figure 1: G_2

The construction of the graph is simple : two nodes $\mathcal{N}_1$ and $\mathcal{N}_2$ are connected by a oriented edge if the string they respectively describe is a and b and a = x.a' (x=0 or x=1) and b = a'.0 or b =

a'.1.

With this construction, each node $u \in V$ in the graph has the following property : $in(u) = out(u) = 2$ except the nodes with only zeros and ones which respect this property $in(u) = out(u) = 1$.

1.3) Demonstrate that there is a k−complete sentence which length is smaller than $2^{k+1} + k - 1$. Now that we know how our graph is constructed for any k, we can use the property we saw in 1.1). Since our G_k graph is a oriented and strongly connected graph such that $\forall u \in V, in(u) = out(u)$, we know there is a eulerian circuit.

Using the eulerian circuit to construct a k-complete sentence gives us the result we want. Indeed, let us choose the node with k zeros $\mathcal{N}$ in G_k, then get the circuit from $\mathcal{N}$ to $\mathcal{N}$. This circuit construction by concatenating the string with the node's symbol after we pass through all nodes gives necessarily a k-complete sentence. Indeed, we pass through all nodes by passing by all the edges so all elements in $\sum^k$ are in the final string.

Since this circuit pass through every nodes twice (except for the node with only zeros and ones) the length of the sentence is strictly less than : 2×2^k plus k for the beginning length (string of length k) and -1 in order to not count the node $\mathcal{N}$ one again at the end. Precision : 2^k is the number of nodes in the graph G_k.

We have that there is a k−complete sentence which length is smaller than $2^{k+1} + k - 1$

1.4) Demonstrate that a minimal k−complete sentence has a length of $2^k + k - 1$. The idea we came out with is the following : since we want a way to obtain a minimal k-complete is to construct the same graph G_k with more constraints. The idea is that when we have G_k, we delete edges until $\forall u \in V, in(u) = out(u) = 1$. We begin by keeping the edges that come in and out of the nodes $u \in V$ such that we already have $in(u) = out(u) = 1$. If all nodes are still not connected, in the case of symmetric edges we force that one of (u,v) or (v,u) is kept. With this construction the graph is still strongly connected but the good news is that the eulerian circuit passes through the nodes only once.

Here is the graph G_3 where the edges are marked with a one to show how to start the transformation (first the keep the edges that come in and out of 000 and 111 and then keep an edge (010, 101) or (101, 010)) :

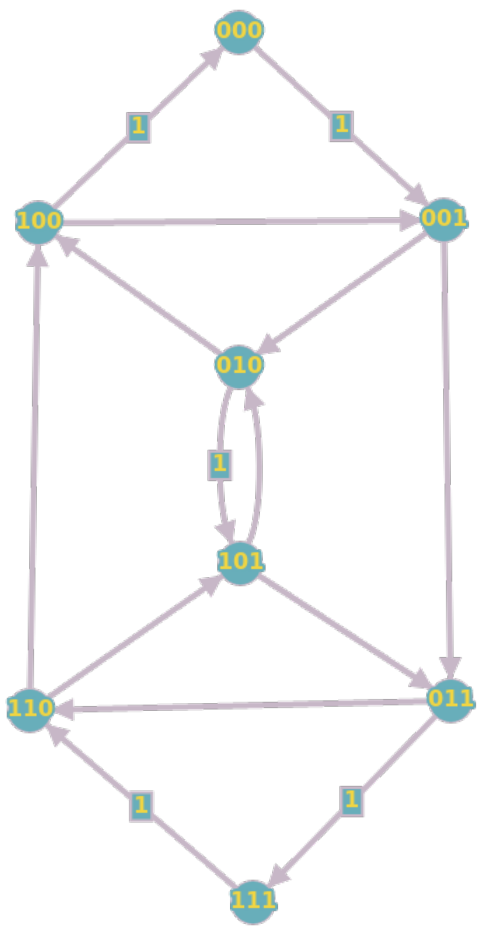

Figure 2: G_3 transformation

How to demonstrate that property ? To show that the transformed graph is still a strongly connected graph ? Since we first keep the edges that connect the only zeros and only ones nodes, we are left with choosing one input and output edge for the rest of the nodes of the graph. Since the second step is to keep one of the edges for all symmetric edges we avoid creating a non-connected graph. (This is maybe not the best demonstration but at least the idea is original !)

This is not a property that works for all strongly connected graph ! It works because our graphs G_k are peculiar (only one symmetric edge, only two nodes with the property $in(u) = out(u) = 1$...).

Then with that transformed graph, like before we start with the node with k zeros $\mathcal{N}$, then we pass through all the nodes using the unique cycle and we stop when we reach $\mathcal{N}$ once again. We know that the sentence formed from using the nodes is a k-complete sentence since it is formed by using all the elements in $\sum^k$. Moreover, the sentence formed uses only one time each element in $\sum^k$ so it is minimal.

The length of that minimal sentence is $2^k + k - 1$ because k is the length of the string describe by the first node. We add the number $2^k - 1$ to k since we concatenate the string with 0 or 1 after passing through each node and the concatenation adds only one letter by construction of the graph. -1 is because we do not count the node $\mathcal{N}$ for which we already count the number k.

Thus we demonstrate that a minimal k$-$complete sentence has a length of $2^k + k - 1$.

1.5) Create a program to produce a minimal k-complete sentence given k. We begin with a string of k zeros. We simply create a list L_{symb} that contains all the possible combinations of length k with 0 and 1 (except only full zeros) and then at each step we verify if adding a 1 make a symbol in L_{symb}. If that is the case, we remove this symbol from the list, we add a 1 to the string and we continue. If this is not the case, we add a 0 to the string.

Exercice 2 - Hogwarts

Let us define n the number of mages and T the set that contains the travel-time of all the mages.

2.1) Show a possible way to make all the mages arrive in the Dark Lord's room. There are various form to make this. However, without thinking too much about the problem, one can consider this simple strategy : always take the mage that has the smallest travel-time to escort the other mages.

In our case, the mage with the smallest travel-time is Hermione with a travel-time of 30 minutes.

If we apply this strategy, the total time will be : $\sum\limits_{t \in T, t \neq \tilde{t}} t + (n-2) \times \tilde{t}$. In our case, this method provides the total time $40 + 40 + 50 + 50 + 75 + 120 + 40 + 6 \times 30 = 595$. As we already know the best solution, we can say that this method does not provide a horrible solution. Nonetheless this is clearly not the best one ! The very first idea that one can think about is to make the "big time travellers" to travel together so that we save time by not making them travel one by one. This idea can be used to make Fred (2 hours) and Ginny (1 hour and 15 minutes) travel together and Luna and Neville as well (both 50 minutes).

2.2) Model this problem using a graph and describe it. First, we had the basic idea of describing every single state in a graph where a state would describe two elements : the mages present in the Dark Lord's room and where is the invisibility cloak. However, when I tried to think about this representation I could not figure out easily a way to program automatically the connections between the nodes and the weight for each edge. So I started to think differently.

The model I made is a graph that describe every possible state where the invisible cloak is never in the Dark Lord's room (except for the last node).

A way to visualize easily the graph construction is to visualize n layers of nodes. The first layer is the beginning or starting point so this is just a single node ($1 = \binom{n}{0}$). This first node describes the state where all the mages are all outside of the Dark Lord's room.

The second layer has $\binom{n}{1}$ nodes, where each node corresponds to a state where only one mage is in the Dark Lord's room. The edge weight from the first node to each node $\mathcal{N}$ in the second layer is just the minimum travel-time to make the mage that corresponds to the state $\mathcal{N}$ travels in the Dark Lord's room.

Then it goes on, the third layer has $\binom{n}{3}$ nodes, the fourth layer has $\binom{n}{4}$ nodes... The penultimate layer has $\binom{n}{n-2}$ nodes and the last layer has only 1 node ($1 = \binom{n}{n}$). The edge weight from each node $\mathcal{N}$ in the penultimate layer to the last node is simply the maximum value of the two travel-times of the two mages still outside of the Dark Lord's room in the state described by the node $\mathcal{N}$.

In the end, we have $\sum_{k=0}^{n} \binom{n}{k} - \binom{n}{n-1} = 2^n - n$ nodes in our graph and n layers (from layer $k = 0$ to layer $k = n - 1$).

Now that we explained the particular cases (edge weight for the first to second layer and penultimate to last layer), we now have to describe how to get the weight for every edge between two layers in this graph. In order to explain it simply, let us consider two nodes, one in the layer k ($\mathcal{N}_k$) and another in the layer $k+1$ ($\mathcal{N}_{k+1}$) with $k \in [\![1, n-3]\!]$. $\mathcal{N}_k$ corresponds to a state where k mages are in the Dark Lord's room. Let us define L_1 the list of mages that are both in the state $\mathcal{N}_k$ and $\mathcal{N}_{k+1}$.

- If $|L_1| < k-1$: this means that the edge can not exist since this transition can not happen. Therefore there is nothing to calculate. (example : (0, 1, 2) $\longrightarrow$ (0, 3, 4, 5))
- If $|L_1| = k-1$: this means that two mages m_1 and m_2 have to come and one mage m_3 that was previously in the state $\mathcal{N}_k$ has to go out of the Dark Lord's room. Thus the weight of this edge can be computed as follows : $max(t_{m_1}, t_{m_2}) + m_3$. (example : (0, 1, 2) $\longrightarrow$ (0, 1, 4, 5))
- If $|L_1| > k-1$: (only possible when $|L_1| = k$ in reality) this means that all mages in $\mathcal{N}_k$ still are in $\mathcal{N}_{k+1}$ except a new one m_i. The weight of the edge is the minimum total travel-time $max(t_{m_e}, t_{m_i}) + t_{m_i}$ where m_e is a escort mage. This escort mage is a mage still outside of Dark Lord's room. This means that the escort mage is not in $\mathcal{N}_{k+1}$. (example : (0, 1, 2) $\longrightarrow$ (0, 1, 2, 3))

I know it might not be super clear right now so I made two examples : one with three mages labeled (0, 1, 2) which travel-times are respectively (1, 2, 3) and one with four mages but no travel-times.

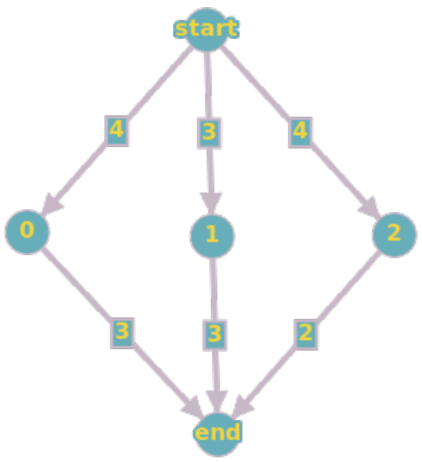

Figure 3: Graph for three mages

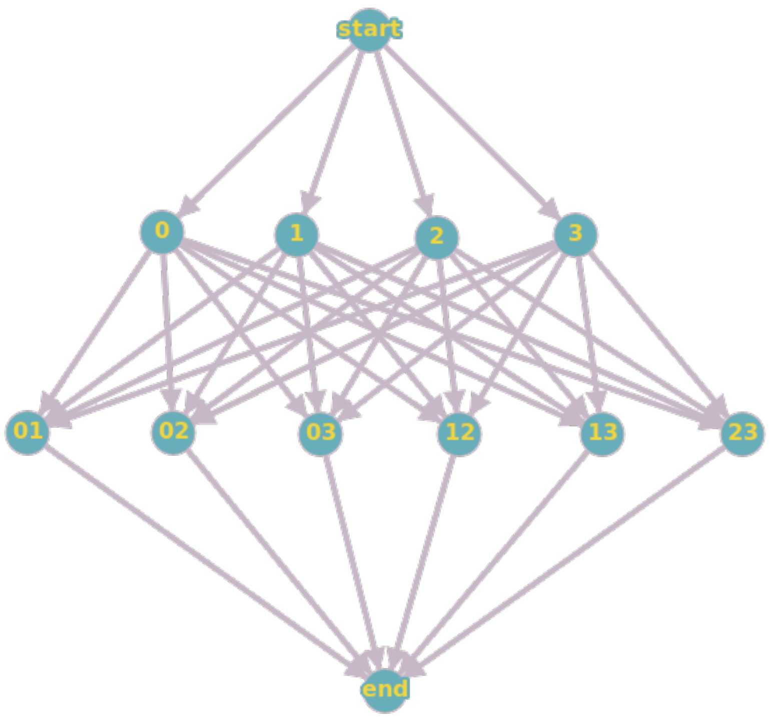

Figure 4: Graph for four mages

Now an essential question is : how to associate an unique number to each node ? We need to encode every possible node into a single number so that we can create our graph. We used the result saw in the first tarea : the unique binary representation of numbers. So each node of state $\mathcal{N}_k = (m_0, m_1, ..., m_{k-1})$ has the id $ID_{\mathcal{N}_k} = \sum_{i=0}^{k-1} 2^{m_i}$. By the uniqueness property of the binary representation, we know that two nodes can not have the same id. We define $ID_{\mathcal{N}_0} = 0$.

How can we associate an id to the last node ? By our graph construction, we know that in the last layer the biggest id is necessarily $\sum_{i=0}^{n-1} 2^i - 2^1 - 2^0 = (2^n - 1) - 2^1 - 2^0 = 2^n - 4$. Thus we just have to define $ID_{\mathcal{N}_{n-1}} = 2^n - 3$ (the maximum id value that we have plus one). The creation of the graph requires in the end a list of $2^n - 2$ elements (+1 for the first node). The number of empty spaces in this list is $2^n - 2 - (2^n - n) = n - 2$ which is not that important since $2^n >> n$ (lineal cost).

With this graph construction we can solve the problem.

2.3) Give an optimal solution of the given problem. As we explained in the part 2.1), a better idea is to make the "big time travellers" travel together and not separately.

ONe optimal way of solving the puzzle is :

- Harry and Hermione travel and Hermione go back : 70

- Ginny and Fred travel and Harry go back : 160
- Harry and Hermione travel and Hermione go back : 70
- Luna and Neville travel and Harry go back : 90
- Hermione and Harry travel and Hermione go back : 70
- Hermione and Ron travel and Hermione go back : 70
- Hermione and George travel : 40

The total travel time is 570.

2.4) Make a program to solve the problem for any number of mages and any time-travel for each mage. We just used Dijkstra algorithm on the graph we can automatically generate out of the time-travel of each mage to get the shortest path from the first node to the last one.

We tested our algorithm for the Dark Lord's room problem and others and we obtain the correct answers.