Homework 3 Graph Algorithms - 3 January, 2020

Constantinescu M.G. George-Gabriel, group E3, year II
Bogatu D. Daniel-Marian, group E3, year II

1 Problem 1

proof: (a)

Let T be a tree.

Given the restriction in the problem, T doesn't have a perfect matching. Hence, we can say that at least one vertex $u \in V(T)$ won't be a part of the final matching for any matching possible in T.

Let M be the matching of T, with $|M| = k$. Now, we choose x be the exposed vertex of T.

Hence, we divide the problem in two situations: x represents a pendant vertex(a leaf, $d_G(v) = 1$); and x doesn't represent a pendant vertex(is not a leaf);

For the first case, this is obvious. We can say that there exists a matching of the same cardinality k which exposes at least a pendant vertex of T.

In the second case, if x is not a pendant vertex. Hence, the only problem would be that if we have all the pendant vertices being in matching M.

Proving the second statement : We can assume that $\forall u \in V_p(T)$, where V_p represents the set of all pendant vertices in T, if we traverse from every pendant vertex to the root of the tree, we will find the vertex x in any case because vertex x(which is exposed) will be part of the path to the root.

Let P be the path who contains the exposed vertex x. First edge of this path will contain one pendant vertex(due to the fact that we start traversing the tree from to last level up to the root). In order to keep the property of a matching safe, the next edge is not a part of M. We repeat this steps until we find two consecutive edges that are not a part of M. In this case, we found the vertex x which connects the two edges who are not in M.

Let e_{lower} and e_{higher} the two edges.

The intersection between these 2 edges is : $e_{lower} \cap e_{higher} =$ x, where x is the exposed vertex.

Now, we use e_{lower} as follows: all the edges which belongs to the path P which are under e_{lower}, we increase the level by one. Now, we choose in the place of the first edge from the path P another edge which doesn't appear in the matching.

Hence, we can say that the number of edges in the path remains the same and our pendant vertex is now an exposed one.

So, we proved that there exists a matching of the same cardinality k which exposes at least a pendant vertex of T.

proof: (b)

Using what we've proved at point (a), we can say that for a tree T who doesn't have a perfect matching there exists T', a similar tree such that has at least one if its vertices exposed. In the case when we have all the pendant vertices in the matching there is a contradiction. Hence, we can say that in order to have a perfect matching for T we have to be sure that all the leaves(all the pendant vertices) are in the maximum matching.

So, we proved that T has a perfect matching if and only if any maximum matching of it saturates all the pendant vertices.

2 Problem 2

We know from the requirment of the problem that all the arcs are directed.

In the variable x[uv] we calculate the maximum flow in the network and flowOut represents the sum of all flows in R.

We can say that the flow would not be greater or equal with the capacity because x represents a flow in R because in every edge uv we have: $x[uv] \neq c[uv]$ and the difference between the sum of the flows which enter the vertex and the sum of the flows which exists the vertex is equal with 0. This is guaranteed by the recursive instruction in the algorithm:

x[uv]=Flow(v,min(c[uv],α).

When the algorithm is called, it chooses in every case min(c[uv],α).

In this way, the algorithm finds the maximum value for the flow, starting in the root. All the arcs are computed and we can say that the time-complexity of this algorithm is $\mathcal{O}(M^2)$, where M represents the number of arcs in the network. We can see that the dominant operation is the recursive one, which is inside a for loop. We can see that an edge can be visited multiple times if it has the flow different than the capacity.(this is why our time-complexity should be higher than M)

3 Problem 3

proof: a)

Let $|X| = r$. From requirment of the problem, we can say that X is the set of all vertices x_i, with $i \in [1, r]$.

Let A be the set of all vertices x'_j, with $j \in [1, d]$.

We know that S = $X \cup A$. Hence, $|X| = |X + A| = |X| + |A| = r + n - r = n$.

We know that $T = V/S$ and $V = W' \cup W$. Hence, $T = (W' \cup W)/(X \cup A)$.

Using the modulo we got this equation: $|T| = |(W' \cup W)|/|(X \cup A)| => |T| = (|W'| \cup |W|)/(|X| \cup |A|)$. So, $T = (n + n) - (r + n - r) = 2n - n = n$.

From $|S| = n$ and $|T| = n => |S| = |T|$;(S and T have the same cardinality).

From $V = X \cup Y => Y = V/X => T \cap Y \neq \emptyset$. Hence, we can say that $X \in S$ and $Y \in T$ (because (X,Y) is a bipartition of W).

We know that $|[xy \in F : x \in X, y \in Y]| \geq k$;we have the subset X with elements x_i, where $i \in [1, n - r]$ and the subset Y with elements x_j, where $j \in [n - r + 1, n]$. From this we can obtain this new statement: $|[xy \in E : x \in S, y \in T]| = (n - r) * (r - 1)$. (1)

We have the following equation: $(n - r)(r - 1) \leq n^2 - k$. (2)

We also know that $n^2 - k = p$. (3)

From (1),(2) and (3) $=> |[xy \in E : x \in S, y \in T]| \leq p$.

In conclusion, we've proved that $|S| = |T|$ and $|[xy \in E : x \in S, y \in T]| \leq p$.

proof: (b)

We can say that the property of the bipartition is safe because we know that $|S| = |T|$ and $E = [xy : x \in V, y \in V, x \neq y, xy \notin F]$. (1)

We also know that $X = W \cap S$ and $Y = W \cap T$. (2)

From (1) and (2) we can conclude that $X \neq \emptyset$ and $Y \neq \emptyset$.

Using a) we can say that from $|[xy \in E \cup F : x \in S, y \in T]| = n^2$ and $|[xy \in E : x \in S, y \in T]| = p$ and $p = n^2 - k$.

Hence, if we use the operation of difference on this 2 equations we obtain:

$|[xy \in E \cup F : x \in S, y \in T]| - |[xy \in E : x \in S, y \in T]| = |[xy \in F : x \in S, y \in T]|$

and

$n^2 - (n^2 - k) = n^2 - n^2 + k = k$

So, $|[xy \in F : x \in S, y \in T]| \geq k$. (1)

But, we also know from point a) that $x \in X$ and $y \in Y$($x \in W \cap S$=X and $y \in W \cap T = Y$; because of the bipartition property: (X,Y) is a bipartition of W). (2)

From (1) and (2) $=> |[xy \in F : x \in X, y \in Y]| \geq k$.

In conclusion, we've proved that that X and Y are non-empty

and $|[xy \in F : x \in X, y \in Y]| \geq k$.

proof: (c)

In order to show that a problem is **NP**-complete we have to show that the problem is in the class NP which means: there exists a non-deterministic algorithm which solves the problem in polynomial time complexity and we must polynomially reduce a known NP-complete problem to our decision problem.

We know that MBC $\in$ NP because there exists an algorithm which solves this problem in a polynomial-time complexity.

Now we have to prove that there exists another NP-complete problem that reduces at MBC. We can use simple-MAX-CUT in order to prove this.

We can use some of the statements from point (a):

Let G=(V,E) be a graph with bipartition (S,T) having the property proved at point (a). (1)

(we have $|S| = |T|$ and $|[xy \in E : x \in S, y \in T]| \leq p$)

Let G'=(V',E') be a graph with bipartion (X,Y) having the property proved at point (b). (2)

(we have $X \neq \emptyset$ and $Y \neq \emptyset$ and $|[xy \in F : x \in X, y \in Y]| \geq k$)

Using the instance for simple-MAX-CUT from the requirment of the problem, we can obtain an instance for MBC problem:

$W' = \{x'_1, x'_2, ..., x'_n\}$, $V = W \cup W'$ and $E = [xy : x \in V, y \in V, x \neq y, xy \notin F]$

Let G=(V,E) be a graph with p=$n^2 - k$.

Using these statements about simple-MAX-CUT and the statements we've got above (1 and 2) we can prove that simple-MAX-CUT reduces at MBC.

From $|[xy \in F : x \in X, y \in Y]| \geq k$ (because we have a bipartition of W), using (1) we can say that there exists a bipartition (S,T) of V such that $|S| = |T|$ and $|[xy \in E : x \in X, y \in Y]| \leq p$.

(exactly what we've proved at point a))

Using what we've said in the paragraph above, having a bipartition (S,T) of V such that $|S| = |T|$ and $|[xy \in E : x \in X, y \in Y]| \leq p$, using (2) we can say that there exists a bipartition (X,Y) where X and Y are non-empty and $|[xy \in F : x \in X, y \in Y]| \geq k$.

(exactly what we've proved at point b))

Hence, we can say that simple-MAX-CUT can be polynomially reduced to MBC(our problem). So, we've proved that **MBC** is a **NP**-complete problem.

4 Problem 4

proof: (a)

We are gonna build our network flow model as follows:

I - Let x be the starting vertex in our network flow model.

II - From x leaves p arcs with capacity n_i, with $i \in [1, p]$ to vertices g_i, with $i \in [1, n]$.

(p represents the total number of freshmen groups)

(n represents the total number of maximum students in a group)

III - From every g_i vertex leaves r arcs with capacity c to the new vertices g'_j, with $j \in [1, r]$.

(r and c are given in the requirement of the problem: r groups and c the maximum number of students from the same old group)

IV - Final step: From every g'_j vertex leaves an arc with capacity m_j with $j \in [1, r]$ to the final vertex of the network flow, y.

Considering all the steps above(I,II,III and IV), we can say that we devised a network flow model for building the new organizing schema.

proof: (b)

Using the network flow model that we devised at point (a), in order to obtain a proper and correct solution we have to be sure that:

i) all the arcs from x to all vertices g_i, with $i \in [1, n]$ have their capacity equal with their flow; In natural language, when the groups are initially formed, there not exists students without a group assigned to them.

ii) the same property from i) is applied here for all the arcs from all the vertices g'_j, with $j \in [1, r]$ to the final vertex y $->$ have their capacity equal with their flow; In natural language, every new group of students that is formed has now m_j students(as the requirement (1)and(2) of the problem says). All the arcs with capacity=c have a crucial role here: we are sure that there won't be more than c students from the same old group into the new group formed(as the requirement (3) of the problem says).

From what we said before at i) and ii):

1)All the arcs leaving x have their capacity equal with their flow;

2)All the arcs entering y have their capacity equal with their flow;

From 1) and 2) we can say that $\nexists$ augmentating paths in the network flow. Hence, the flow in the network is maximum.

Let g_i be one of the initial groups in the network. From this group we have arcs to all the new group g'_j, with $j \in [1, r]$. The flow from each arc represents the number of students from the old group g_i which are now selected in the new group g'_j. So, in this way, following all the arcs between the two sets of arcs g and g', we obtain this new schema for the network.

Considering what we've said above, we've proved a characterization of the existence of a solution to this problem in terms of maximum flow in the above network.

proof: (c)

In order to solve this part of the problem, we have to choose 3 algorithms and discuss their time complexity.

In this case, we choose the following algorithms from the lectures notes:

I) Ford-Fulkerson algorithm (Lecture 8, page 24)

II) Edmonds-Karp algorithm (Lecture 8, page 32)

III) Goldberg-Tarjan algorithm (Lecture 9, page 17)

For I) algorithm, we know that the time-complexity is $\mathcal{O}(m * v)$, where m is the number of edges in our network and v is the maximum flow value. In our case(natural language), v represents the total numbers of students and m = p + r*p + r, where p represents the initials groups of freshmen formed and r represents the total number of new groups formed.

For II) algorithm, we know that the time-complexity is $\mathcal{O}(m^2 * n)$, where m is the number of edges in our network and n is the number of vertices in our network. The value of m is the same: m = p + r*p + r and the value of n = p + r + 1 + 1 = p + r + 2. (In natural language, the sum of all groups initially formed + the number of all groups which are formed after + the starting vertex x + the final vertex y).

For III) algorithm, we know that the time-complexity is $\mathcal{O}(m * n^2)$, where m is the number of edges in our network and n is the number of vertices in our network. The values for the n and m are the same as II) : m = p + r*p + r and n = p + r + 2.