Algorithm Design & Analysis: Homework #5

Due on 2015/01/11

王冠鈞 (b03902027)

Problem 1

(0) References: Mein Kampf

(1) According to $S(l(v)) \leq \alpha \cdot S(v)$, we get $S(v) \leq \alpha \cdot S(f(V))$, and the following:

$$S(v) \leq \alpha \cdot S(f(V))$$
$$S(f(v)) \leq \alpha \cdot S(f^2(v))$$
$$\vdots$$
$$S(f^{n-1}(v)) \leq \alpha \cdot S(f^n(v))$$

By multiplying these n inequalities, we get:

$$S(v) \leq \alpha^n \cdot S(f^n(v))$$
$$S(f^n(v)) \geq (\frac{1}{\alpha})^n \cdot S(v)$$

, which is the answer.

(2) Let v be the leave node, then $S(v) = 1$. From (1), we know that $N = S(f^n(v)) \geq (\frac{1}{\alpha})^n \cdot 1$. Then we get $height = \log_2(\frac{1}{\alpha})^n \leq \log_2 N$, and finally $h = O(log(N))$.

(3) Analyse the operation via the steps written in the hints.

(a) For an 1/2-balanced tree, $S(l(v)) = S(r(v)) \rightarrow \Delta(v) = |S(l(v)) - S(r(v))| = 0$, thus it has potential 0.

(b) By observation, we can find that for every level, the maximum potential is $N(2\alpha - 1)$, so it may contain $N \log N \cdot (2\alpha - 1)$ potential at most. It means that in average, an insert operation needs $\log N(2\alpha - 1) = O(\log N)$ time. Thus the potential function may be:

$$\Phi(T) = (2\alpha - 1) \sum_{v \in T, \Delta(v) \geq 2} \Delta(v)$$

,whewe c can be replaced as $(2\alpha - 1)$.

(4) From (3), when a node has size m, to reconstruct a balanced tree, it will use up at most $m(2\alpha - 1)$ potential, which is $O(m)$ time.

Problem 2

(0) References:

[1] `http://www.cs.princeton.edu/~wayne/cs423/lectures/reductions-poly-4up.pdf`

[2] `http://cgi.csc.liv.ac.uk/~igor/COMP309/3CP.pdf`

[3] Michael R. Garey; David S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*, pp. 84-86

(1) According to the hint, we can add literals to every clause until each clause reaches k. For instance, we have $(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6)$ initially, then we can extend it into 4-literal clauses with the following way:

$$(y_1 \vee x_1 \vee x_2 \vee x_3) \wedge (\neg y_1 \vee x_1 \vee x_2 \vee x_3) \wedge (y_2 \vee x_4 \vee x_5 \vee x_6) \wedge (\neg y_2 \vee x_4 \vee x_5 \vee x_6)$$

Keep doing the process until it reaches k-literal clauses and thus reduced to a K-CNF-SAT. The simple pseudocode is listed below:

Data: Input for 3-CNF-SAT problem
Result: Input for k-CNF-SAT problem, $k > 3$
while *there are clauses whose number of literals is less than* k **do**
    duplicate the clause into two clauses with AND ($\wedge$) operator and assign y and $\neg y$ to the clauses with OR ($\vee$) respectively
end

Algorithm 1: How 3-CNF-SAT is reduced to k-CNF-SAT

(2) Similar with the previous problem, according to the hint, we can split a clause of size > 4 by adding literals until all the clauses have size 3. For instance, if we have $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5)$, representing a clause in a 5-CNF-SAT, then we can split it into 3-CNF-SAT with the following way:

$$(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5)$$
$$= (y_1 \vee x_1 \vee x_2) \wedge (\neg y_1 \vee x_3 \vee x_4 \vee x_5)$$
$$= (y_1 \vee x_1 \vee x_2) \wedge (y_2 \vee \neg y_1 \vee x_3) \wedge (\neg y_2 \vee x_4 \vee x_5)$$

The reduction process it roughly does is listed below:

Data: Input for k-CNF-SAT problem, $k > 3$
Result: Input for 3-CNF-SAT problem
while *there are clauses whose number of literals is more than 3* **do**
    split the clause of size s into $\lfloor \frac{s}{2} \rfloor$ and $(s - \lfloor \frac{s}{2} \rfloor)$ and add literal y and $\neg y$ to the two clauses respectively
end

Algorithm 2: How k-CNF-SAT is reduced to 3-CNF-SAT

(3) This is a *3-colorability problem* (3-COLOR), which has been proven to be a NP-complete problem by *Karp* by reducing 3-CNF-SAT to it. For this problem, we have to prove that 3-CNF-SAT $\leq_p$ 3-COLOR.

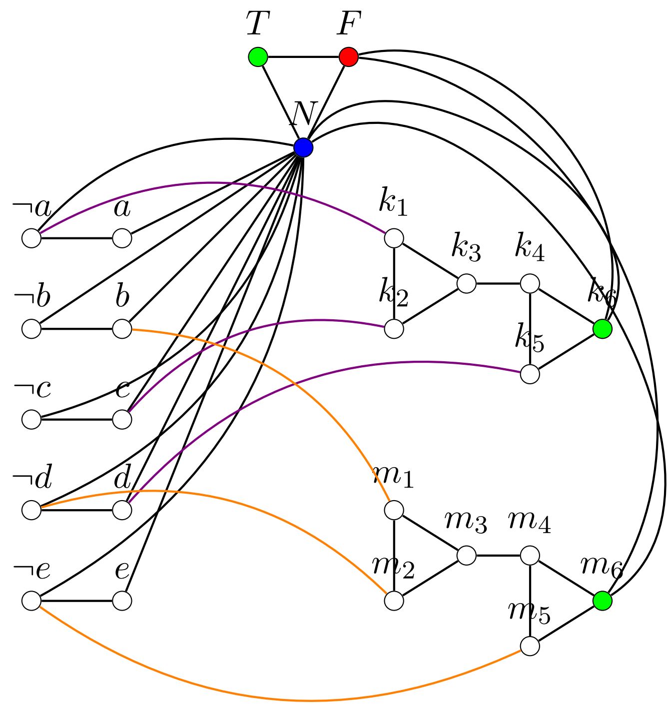

Figure 1: A graph G_ϕ where $\phi = (\neg a \vee c \vee d) \wedge (b \vee \neg d \vee \neg e)$

According to reference [2], for every 3-SAT equation ϕ, we can construct a graph G_ϕ such that if ϕ is satisfiable, then G_ϕ is 3-colorable.

To construct such a graph, we at first need three vertices, T, F, N forming a cycle, which are green (true), red (false), blue, respectively. Then we construct every variable in a boolean equation with a pair, y and $\neg y$, and connect them together. Then, connect N to all these vertices, meaning that when a vertex is colored *true*, its complement must be *false*. After that, construct two triangle graphs and connect them together with an edge, which will be functioned as *OR-gate*.(See Fig. 1, $k_1, k_2, \cdots, k_6$ form an OR-gate). See Figure 1, connect F, N with the end (k_6, m_6) of the OR-gates, restricting that the output should be *true*, and connect the input with the vertices to the left, whose boolean value will be "sent" into the OR-gate and make an output. Since the bi-triangular graph acts as an OR-gate, the output are restricted to be *true*, and the inputs are restricted to be *true* or *false*, it implies that if all the inputs are *false*, it will contradict with the restricted output and thus isn't 3-colorable, implying that the 3-SAT is not satisfiable.

For the entire process, to solve a 3-colorable problem, we can modify the graph to make it become equivalent with a certain 3-SAT equation (or k-SAT), and then if such equation is satisfiable, there will be a solution for the 3-colorable problem. Modifying the graph to map a k-SAT problem can be done in polynomial time.

(4) This time, not a 3-colorability problem, but to keep all the vertices of same color connected.

Problem 3

(0) References:

[1] Consulted with B03902045

[2] `https://en.wikipedia.org/wiki/Graph_isomorphism_problem`

(1) To prove whether two simple connected graphs are isomorphic, we need to verify:

(a) It is in GI.

(b) It is in GI-hard.

For the first subtask, in this case it is trivial since it's also a kind of isomorphic problem, which is the definition of GI.
As for the second subtask, we can consider two arbitrary graphs G_1, G_2, which are not necessarily connected. Examine their connected components, if the number of the connected components are not the same, they are definitely not isomorphic. After that, if they have a same number of sub-components, let it be k, then we will call the connected-isomorphic problem at most k^2, polynomial times to do the mapping between these two graphs, and if the sub-components can be successfully mapped, G_1, G_2 are found to be isomorphic. In the total reduction process, we called the connected-isomorphic problem polynomial times to do the mapping, which means that the problem is also in GI-hard.
In conclusion, the problem is also an GI-complete problem.

(2) The fact that the problem is in GI also holds trivially.
Since the determination of two connected graphs is GI-complete, we can force a vertex $u \in G_1(V_1, E_1), v \in G_2(V_2, E_2)$, to connect with a large perfrct graph K_m, and we can examine whether it's isomorphic after adding the graph, and we will memorize the results for every $u \in G_1, v \in G_2$. in this process, it will call the connected graph isomorphic problem at most $V_1 \times V_2$ times, which is polynomial. After that, we'll create a bipartite graph with respect to the previous result. After connecting all the mapping possibilities, we can perform a perfect match and thus we can finally find the exact mapping between the two isomorphic graphs. For the entire process, we can also verify that the problem is also GI-hard, and then it is GI-complete, in conclusion.

(3) The fact that the problem is in GI also holds trivially.
As for GI-hard, we can consider a arbitrary graph G. Then we can calculate the degrees and verify the vertex with the largest degree, and let it be k. For all vertices whose degree is

less than k, draw graphs of degree k and connect to the original vertices in order to make all the vertices k to become regular. However, one can verify that when k is even, the graphs we added can never attain degree k for all vertices added. In this case, we can extend the degree to $k + 1$ so as to make it possible to regularize. Then we can verify that, for any two graphs G_1, G_2, the two graphs are isomorphic if and only if their regularized graph, G_1^r, G_2^r, are determined to be isomorphic via the regular-isomorphism algorithm. That is, we can reduce the isomorphic problem to the regular isomorphic problem by checking, adding vertices and edges, and calling the problem once, which in total takes polynomial time. Thus, the regular-isomorphism problem is also in GI-hard, and then it is a GI-complete problem.