ADA homework 5 b03902089 林良翰

Problem 1

1.(1) prove $S(f^n(v)) \geq S(v) \cdot (\alpha^{-1})^n$

$n = 1 : S(f(v)) \cdot \alpha \geq S(v)$ *by conditions*

$n = 2 : S(f^2(v)) \cdot \alpha^2 \geq S(v)$

…

$n = k : S(f^k(v)) \cdot \alpha^k \geq S(v) \Rightarrow S(f^k(v)) \geq (\alpha^{-1})^k \cdot S(v) \Rightarrow$ *proved.*

1.(2)

Set n = *the height of the tree*. By **1.(1)**, we have

$N = S(f^n(v)) \geq (\alpha^{-1})^n S(v)$

$\log(N) \geq \log((\alpha^{-1})^n) + \log(S(v))$

$n \leq (\log(N) - \log(S(v))) / \log(\alpha^{-1}) \Rightarrow n = O(\log(N))$

1.(3) reference : https://en.wikipedia.org/wiki/Scapegoat_tree

Define the deference of the subtrees of node t as this function : $D(t)=max(|S(l(t))-S(r(t))|-1,\ 0)$

We can assert that after rebuilding a subtree, $D(t) = 0$.

Thus, before rebuilding a subtree, $D(t) = m = \Omega(S(t))$, since we assume that there are $\Omega(S(t))$ degenerate insertions. And we need $O(\log(N))$ for every operations($\Omega(S(t))$ times) to rebuild a subtree because by **1.(2)**, the height of the tree is $O(\log(N))$. The final insertions that causes rebuilding cost $O(S(t))$.

As a result, we need $\frac{\Omega(S(t))O(\log N)+\Omega(S(t))}{\Omega(S(t))}=O(\log N)$ for every insertions of a *scapegoat tree*.

1.(4)

Set $S(t)$ = *the size of subtree* = m. Assume that before breaking the rule of proportion α (when it happens, we start to rebuild subtree), we have $D(t) = O(m)$ since there would at most $O(m)$ elements to form a "*line-tree*" (the tree or subtree that every nodes with only one child). By **1.(3)**, the total insertion costs = $\log(m) + \frac{1}{2}\log(m) + \ldots + \frac{1}{2^k}\log(m) = 2\log(m)$ Last, we have $O(2\log(m)+m) = O(m)$ for rebuilding a subtree.

Problem 2

2.(1)

Set A to be the initial clause $A = (a_1 \vee a_2 \vee a_3)$.

We have know that : if A is satisfiable if and only if $(y \vee A) \wedge (\neg y \vee A)$ is satisfiable.

Thus we can add the elements into clause one by one until the number of clause = m

$A \Leftrightarrow (y_1 \vee A) \wedge (\neg y_1 \vee A) \Leftrightarrow (y_2 \vee y_1 \vee A) \wedge (y_2 \vee \neg y_1 \vee A) \wedge (\neg y_2 \vee y_1 \vee A) \wedge (\neg y_2 \vee \neg y_1 \vee A) \Leftrightarrow \ldots$

The time we transfer one *3-CNF-SAT* to *k-CNF-SAT* needs $O(m) = O(2^k)$ time since each iterations will double the number of the clauses.

2.(2)

Set $C = C_1 \wedge \ldots \wedge Cn$ to be the initial clauses.

We have know that : $(A \vee B)$ is satisfiable if and only if $(y \wedge A) \wedge (\neg y \vee B)$is satisfiable.

Thus we separate each clauses one by one until the number of elements in each clauses = 3.

Demonstrate one of the clause C_i in C :

$C_i \Leftrightarrow (y_1 \vee C_i/2) \wedge (\neg y_1 \vee C_i/2) \Leftrightarrow (y_2 \vee y_1 \vee C_i/4) \wedge (y_2 \vee \neg y_1 \vee C_i/4) \wedge (\neg y_2 \vee y_1 \vee C_i/4) \wedge (\neg y_2 \vee \neg y_1 \vee C_i/4) \Leftrightarrow \ldots$

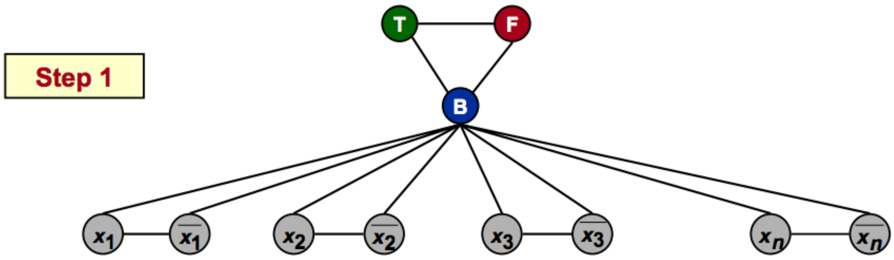
The time we transfer one *k-CNF-SAT* to *3-CNF-SAT* needs $O(\log(k))$ since each iterations will divide the number of element in clauses.

2.(3) reference : http://www.cs.princeton.edu/~wayne/cs423/lectures/reductions-poly-4up.pdf

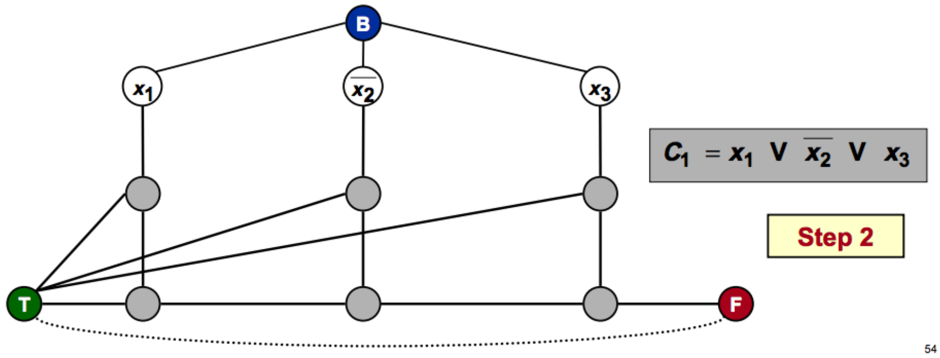
We can proved this problem is *NP-complete* by reducing *3-CNF-SAT* to *3-colorability* problem.

1˚ Let three brothers have their own colors(red, blue, green) to tag the islands(vertices), and the bridges are the edges between vertices.

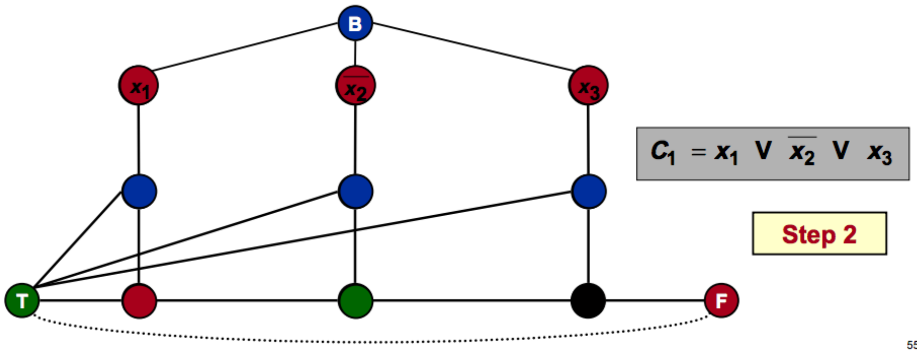
2˚ First we create a triangle graph with red(defined as false in clause), green(defined as true in clause), blue(undefined), and then connect all vertices(x_i and $\neg x_i$) with the blue vertices.



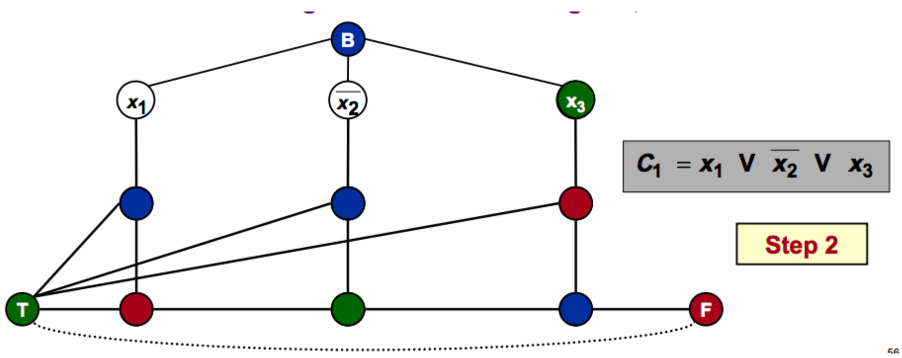
3˚ Second, for each clause, add 6 vertices and 13 edges like this :



4˚ Coloring the clause and make at least one green(true) in the clause(x_1, $\neg x_2$, x_3). If not, coloring all red(false), and we will find that it is not colorable for the rest 6 nodes. (black node shows that it can't be colored).



5˚ We must color one of the element green in clause. Repeatedly connect all clause likely with 3˚and 4˚. Thus, we finally reduced *3-CNF-SAT* to *3-colorability* problem, which is an *NP-complete* problem, too.



2.(4)
1° We can again create a graph $G(V, E)$ with islands defined as vertices V and bridges defined as edges E, and three brothers have their own colors to tag the islands.
2° Transform the graph $G(V, E)$ to be the edge complement graph G_c, and it is in polynomial time $O(V^2)$ since we have to delete the edges connected in G and add the complement edges not connected in G for every vertices in V, in other words, we have iterations of number of edges in complete graph of G, which is $V(V-1)/2$.
3° Solving the resulting graph G_c by *3-colorability* problem which had been proved to be *NP-Complete* problems in **2.(3)**. As a result, the method proposed by the youngest brother is *NP-complete* problem.

Problem 3

3.(1)
1° Set two graph, $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, which are both connected.
2° Transform G_1 and G_2 into bipartite graph with one side are components of G_1 and another side are components of G_2, the transformation will be in polynomial time.
3° Connect the vertices between two sides if the vertices are isomorphic.
4° Find maximum cardinality bipartite matching, which will be done in $O(N^3)$ time. Finally we will verify that whether G_1 and G_2 are isomorphic.
5° *Graph-isomorphism* problem, which is *GI-complete*, is turing reduced to *two-simple-graph-isomorphism* problem.

3.(2)
1° Set two complete graph K_G and K_H, respectively for $G(V_G, E_G)$ and $H(V_H, E_H)$
2° $\forall u \in V_G$, $\forall v \in V_H$, according to the result from **3.(1)** which is *GI-complete*, connect the corresponding u and v respectively to K_G and K_H, and connect the vertices between K_G and K_H which is connected respectively to u and v which are corresponded (or isomorphic). The cost of time for connecting complete graphs is $O(V)$ ($V = max(V_G, V_H)$), which is polynomial.
3° Last, we have reduced *two-simple-graph-isomorphism* problem, which is *GI-complete* problem, to *bijective-function-for-graph-isomorphism* problem.

3.(3)
We can use the method of **3.(2)** to map two regular graphs to complete graph, and finally will show the bijective is available if it correctly map all vertices of two regular graph to isomorphic pairs. Thus, it is a reduction from *bijective-function-for-graph-isomorphism* problem to *regular-graph-isomorphism* problem, which is proved to be *GI-complete* problem, too.