Homework #5

Due Time: 2015/01/11 (Mon.) 12:00
Contact TAs: ada@csie.ntu.edu.tw

Instructions and Announcements

- For the first time submitting your ADA 2015 HW, please create a repository for ADA on bitbucket (https://bitbucket.org) and share the repository with user ada2015 with read permission.
- You have to login to the judge system (http://ada2015.csie.org) to bind the bitbucket repository to your account. For programming problems (those containing "**Programming**" in the problem topic), push your source code to your bitbucket repository. After you push the source code to repository, you can run the judge and get the score. Please refer to the guide of the judge system on its index page for further details.
- For other problems (also known as "hand-written problems"), submit the answers in an electronic copy via the git repository before the due time. Please combine the solutions of all these problems into **only ONE file** in the PDF format, with the file name in the format of "hw[# of HW].pdf" (e.g. "hw1.pdf"), all in **lowercase**; otherwise, you might only get the score of one of the files (the one that the grading TA chooses) or receive penalty because of incorrect filename.
- Discussions with others are strongly encouraged. However, you should write down your solutions in your own words. In addition, for **each and every** problem you have to specify the references (the Internet URL you consulted with or the people you discussed with) on the first page of your solution to that problem. You may get zero point for problems with no specified references.
- **NO LATE SUBMISSION ONE DAY AFTER THE DUE TIME IS ALLOWED.** For all submissions, up to one day of delay is allowed; however, penalty will be applied to the score according to the following rule (the time will be in seconds):

$$\text{LATE\_SCORE} = \text{ORIGINAL\_SCORE} \times (1 - \text{DELAY\_TIME}/86400)$$

Note that late submission of partial homework is NOT allowed. The penalty will be applied to the entire homework in the case of late submission.

Problem 1 (20+5%)

Binary search tree is a useful data structure, but one big problem is that its worst-case time complexity is bad. For example, if we insert a sequence of N increasing numbers into a binary search tree, the binary search tree degenerates to a link list, so any subsequent insertion takes $\mathcal{O}(N)$ time.

Self-balancing binary trees solve the problem by preforming some transformation on the tree, so the height of the tree would always be in $\mathcal{O}(\log N)$ order, and hence an insertion or a query takes $\mathcal{O}(\log N)$ time no matter what the inputs are.

In this problem, we are dealing with a kind of self-balancing binary tree, called *scapegoat tree*.

Let $S(v)$ denote the size of v, that is, the number of nodes that are in the subtree of v, including the root node v. Let $l(v)$ to be the left child of v, and $r(v)$ to be the right child of v.

A node v is α-balanced if

$$S(l(v)) \leq \alpha \cdot S(v) \quad \text{and} \quad S(r(v)) \leq \alpha \cdot S(v),$$

where α is a constant satisfied $0.5 \leq \alpha < 1$. (Intuitively, a node is balance means that the size of its left subtree is almost equal to the size of its right subtree.)
A tree v is α-balanced if every node in the tree is α-balanced.

(1) Let $f(v)$ be the father node of v, $f^2(v) = f(f(v))$ and so on. Prove that in a α-balanced tree, $S(f^n(v)) \geq S(v) \cdot (1/\alpha)^n$ for all n. (4%)

(2) Prove that the height of an α-balanced tree is of $\mathcal{O}(\log N)$, where N is the size of the tree. (4%)

Now pick an α such that $0.5 < \alpha < 1$ (usually, we pick $\alpha = 0.7$) , the scapegoat tree maintain itself to be an α-balanced tree by doing the following operations after each insertion.

1. Check if the tree is still an α-balanced tree.
2. If not, find the highest node that is not α-balanced. Rebuild the subtree of the node to be an 1/2-balanced tree (i.e, fully balanced tree).

(3) Use amortize analysis to prove that an insertion, together with the operations after the insertion, takes $\mathcal{O}(\log N)$ time, where N is the final size of the tree. Assume that rebuilding a subtree of size m to be an 1/2-balanced tree only takes $\mathcal{O}(m)$ time. (12%)

Hint: Define the potential of the tree T to be

$$\Phi(T) = c \sum_{v \in T, \Delta(v) \geq 2} \Delta(v), \quad \textit{where } \Delta(v) = |S(l(v)) - S(r(v))|$$

If you have no idea, you could follow these steps.

(a) Briefly explain that an 1/2-balanced (sub)tree has potential 0.

(b) Observe that the potential decrease after a subtree is rebuilt. The potential difference should be large enough to "pay" for the time to rebuild a subtree.

(c) Since rebuilding a subtree of size m takes $\mathcal{O}(m)$ time, there is a constant β so that the operation takes no more than βm time. Find c such that (b) is satisfied.

(d) prove that every insertion increases the potential by $\mathcal{O}(\log N)$.

(e) summarize the above and finish the proof.

(4*) Prove that rebuilding the subtree of the node to be an 1/2-balanced tree takes $\mathcal{O}(m)$ time, where m is the size of the subtree. (5%)

Problem 2 (20%)

(1) (4%) To prove that k-CNF-SAT is NP-hard, we want to reduce 3-CNF-SAT to k-CNF-SAT ($k > 3$). Please give a reduction algorithm that runs in polynomial time. The k-conjunctive normal form (k-CNF) is $= C_1 \wedge C_2 \wedge \cdots \wedge C_m$. Where $C_i = (x_1 \vee x_2 \vee \cdots \vee x_k)$.
Hint. A is satisfiable if and only if $(y \vee A) \wedge (\neg y \vee A)$ is satisfiable.

(2) (4%) Please give an algorithm that reduces k-CNF-SAT to 3-CNF-SAT ($k > 3$) in polynomial time.
Hint. $(A \vee B)$ is satisfiable if and only if $(y \vee A) \wedge (\neg y \vee B)$ is satisfiable.

(3) (8%) In a country far far away from Taiwan there lives a father and his 3 sons. The father is very rich and owns a group of islands. Some islands are connected with other islands via bridges and some do not. However, the father wishes to pass his wealth, including these islands, evenly to his 3 sons. But here comes the problem: how to evenly split those islands into 3 groups for his sons to inherit ?

The oldest son suggests that the islands should be split according the following rule: each island can be assigned to one of the 3 sons, and the two islands connected by each and every bridge cannot be assigned to the same son. Any set of assignments of islands to sons that satisfies this rule is valid.

However, the agency that tries to come up with a valid assignment finds it very time-consuming. Your task is to prove that, even to know whether there exists a valid set of assignment, is a NP-complete problem.
Hint: can (1) & (2) help us on this question?

(4) (4%) However, the youngest son has a different opinion. He suggests that the islands should be split according to the following rule: each island can be assigned to one of the 3 sons, and the islands belonging to the same son should all be directly connected (i.e., via one bridge) to one another.

The agency again finds that coming up with a valid set of assignments is very time-consuming. Your task again is to prove that knowing whether a valid set of assignments of islands to sons exists is a NP-complete problem.
Hint: can problem (3) help us on this question?

Problem 3 (30+10%)

Graph isomorphism problem is the problem of determining whether two graphs are isomorphism.

Two simple graphs G, H are *isomorphic* if and only if there is an bijection $f : V(G) \mapsto V(H)$ such that if $(u, v) \in E(G)$, than $(f(u), f(v)) \in E(H)$. Recall that a bijection is an mapping that is one-to-one and onto. We assume that all the graphs in this problem are simple graphs.

The graph isomorphism problem is currently not known to be in P nor to be in NP-complete (NPC). So, if a problem can be reduced to a graph isomorphism problem in polynomial time, we say the problem is in GI.

We also define GI-hard to be the set of all problems X such that any problems in GI can be reduced to X in polynomial time, and GI-complete to be the set of problems that are both in GI and GI-hard.

Usually, reduction in "polynomial-time reduction" refers to *many-one reduction*, but complexity classes GI, GI-complete are defined using *turing reduction*, which is just a little different.

- A problem X is polynomial-time **many-one** reducible to a problem Y if and only if X could be solved by a **single** call to Y in addition to some polynomial time outside of the call.
- A problem X is polynomial-time **turing** reducible to Y if and only if X could be solved by a **polynomial** number of calls to Y in addition to some polynomial time outside of these calls.

(1) Prove that determing two simple **connected** graphs are isomorphic is in GI-complete. (15%)

Hint: Notice that in the original graph isomorphism problem, the graph need not to be connected.

Hint2: If you have no idea, try to follow the steps below. To prove that X is in GI-complete, you need to prove

(a) X is reducible to graph isomorphism problem, which should be trivial.

(b) Graph isomorphism problem is reducible to X. Using turing reduction, what you need to prove is that solving graph isomorphism only needs a polynomial number of calls to X plus some polynomial time other than these calls. Note that maximum cardinality bipartite matching could be done in $\mathcal{O}\left(N^3\right)$, which is in polynomial time (you have learned it in HW4).

(2) Prove that "Not only to determine whether two graphs G, H are isomorphic, but also give the bijection (solution) f if the two graphs are isomorphic." is also in GI-complete. (15%)

(3*) Prove that to determine whether two *regular* graphs G, H are isomorphic is in GI-complete. A *regular* graph is a graph that each vertex has *the same degree*. (10%)

Problem 4 - Plagiarism (Programming, 30+6%)

Description

HanHan is the chief urban designer of HyperHacker city. This city is known for its beautiful urban design. To simplify this problem, we treat HyperHacker city as an undirected graph, G_1.

HugeHammer, the lazy and evil urban designer of HappyHippo City, is too lazy to create his own design. He decides to just copy HanHan's design, and randomly renames the vertices in the graph. We call the renamed graph G_2.

To simplify this problem again, we assume that the vertices of G_1 and G_2 are labelled as $1, 2, \ldots n$. The renaming can be viewed as a random permutation π of $1, 2, \ldots, n$, so an edge (u, v) in G_1 would be an edge $(\pi(u), \pi(v))$ in G_2.

For example, if there are three vertices and two edges in G_1 and G_2. The edges in G_1 are $(1, 2)$ and $(2, 3)$. The edges in G_2 are $(2, 3)$ and $(1, 3)$. A possible permutation is $\pi = \langle 1, 3, 2 \rangle$. Since we can map edge $(1, 2)$ in G_1 to $(\pi(1), \pi(2)) = (1, 3)$ in G_2, and $(2, 3)$ in G_1 to $(\pi(2), \pi(3)) = (3, 2)$ in G_2. Note that the graphs are undirected, so $(2, 3)$ and $(3, 2)$ are interchangeable.

Please help HanHan to find the permutation π as the evidence of plagiarism.

Input Format

The first line contains an integer T indicating the total number of test cases. Each test case starts with a line containing two integers n, m, denoting the number of nodes and edges in the undirected graphs G_1, G_2. Each the following m lines contains two integers a_i, b_i, denoting an edge (a_i, b_i) in G_1. Each the following m lines contains two integers c_i, d_i, denoting an edge (c_i, d_i) in G_2.

- $1 \leq T \leq 10$
- $3 \leq n \leq 10000$
- $0 \leq m \leq 100000$
- $1 \leq a_i, b_i, c_i, d_i \leq n$
- There **MAY** be self-loops and/or multi-edges in the graphs.

Output Format

For each test case, please output the permutation π in one line as $\pi(1), \pi(2), \ldots \pi(n)$ separated by a space. If there are more than one possible permutation, you can output any one of them.

Sample Input

```
3
3 2
1 2
2 3
2 3
1 3
3 3
1 2
2 3
3 1
1 3
3 2
1 2
3 3
1 1
2 3
3 2
2 2
3 1
1 3
```

Sample Output

```
1 3 2
1 2 3
2 1 3
```

Hint

- There are two bonus tests (3 points each) in this problem. The condition $n \leq 1000$ and $m \leq 10000$ holds for the first 10 tests.
- The top secret of HanHan is that he just random generated a graph as his design.