Data Structure and Algorithm, Spring 2018

Homework 4

Release: Tuesday, June 5, 2018

Due: 23:59:59, Sunday, June 17, 2018

TA email: dsa1@csie.ntu.edu.tw

Rules and Instructions

- In homework 4, the problem set contains 7 problems and is divided into two parts, non-programming part (problem 1, 2, 3, 4, 5) and programming part (problem 6, 7). This homework set is worthy of 125 points in total including 25 bonus points. You can get more than 100 points in this final homework set. Good Luck.
- Please go to *DSA Judge* (`https://dsa.csie.org`) to complete this homework. If you have any problem, check out the uploading tutorial in problem 0.
- For problems in non-programming part, you should combine your solutions in ONE pdf file, and then hand it in via the judge system. You can either type them or write on papers and scan them. If you choose the latter, please make sure that your writing is recognizable and clear enough, otherwise you might receive some penalties. The pdf file should contain your STUDENT ID and your NAME. If not, you would get minor penalties. Your solution must be as simple as possible. At the TAs' discretion, too complicated solutions will be counted as wrong.
- For problems in programming part, you should write your code in C programming language, and then hand them in via the judge system. You have 5 chances per day for each problem. The compilation command of the judge system will be `gcc main.c -static -std=c11 -O2 -lm`.
- Discussions with others are encouraged. However, you should write down your solutions in your own words. In addition, for each 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.
- Do NOT lend your judge system account to others or borrow other students' accounts. Both cases would be regarded as plagiarism.
- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

$$LateScore = (\frac{86400 - DelayTime(sec.)}{86400})OriginalScore$$

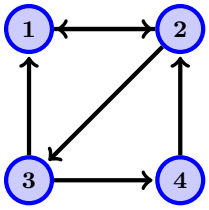
Non-Programming Part

Problem 1. Adjacency matrix (13% + Bonus 4%)

We've learned adjacency matrix in class. It is a method to store an n-node graph G into an n by n matrix M, satisfying

$$M_{i,j} = \begin{cases} 1, & \text{if } \langle i,j \rangle \in E \\ 0, & \text{otherwise} \end{cases}$$

In this problem, we will discuss some applications of adjacency matrix and its generalization. Note that we only consider **simple unweighted graph** in this problem. A graph is simple means that it does not contain any self-loops or multi-edges.



1. (a) (1%) Please write down M, the adjacency matrix of the graph above.

   (b) (2%) Please calculate M^3.

   (c) (2%) For every node i in the previous graph, please list all paths of length 3 from node 1 to node i.

   (d) (4%) Please prove that $(M^k)_{i,j} =$ (number of paths of length k from node i to node j) for every $k \in \mathbb{N}$.
   Note that those paths are not necessarily simple.
   You can use mathematical induction to prove the statement.

2. Now let's modify the adjacency matrix.

$$M'_{i,j} = \begin{cases} 1, & \text{if } \langle i,j \rangle \in E \\ 0, & \text{if } i = j \\ \infty, & \text{otherwise} \end{cases}$$

Also, the product of two matrix A and B are redefined like below.

$$(AB)_{i,j} = \min_{1 \le k \le n} (A_{i,k} + B_{k,j})$$

(a) (4%) Please prove that

$$(M'^k)_{i,j} = \begin{cases} \text{(length of shortest path from node } i \text{ to node } j\text{)}, & \text{if such length} \le k \\ \infty, & \text{otherwise} \end{cases}$$

for every $k \in \mathbb{N}$.
You can use mathematical induction to prove the statement.

(b) (Bonus 4%) Let the time complexity to calculate the redefined product of two n by n matrices be $O(T(n))$, please design a $O(T(n)\lg n)$-time algorithm to solve all-pair-shortest-path problem. That is, calculate length of shortest path from node i to node j for every $1 \leq i, j \leq n$.

(Hint: Exponentiation by squaring)

Problem 2. Rabbit Talent Contest (10%)

After 2 homework assignments, you finally found that unhonest rabbits are actually useless. Thus you ~~eliminated~~ released all the rabbits except one honest and smart rabbit, and you keep it as your beloved pet.

Now, you and your rabbit are preparing for a "pet talent contest". There will be 3 rounds in the contest. In each round, there will be 5 phases. They are:

- *Announcing Phase:* The host will announce the rule of this round.
- *Discussing Phase:* You and your rabbit can discuss the strategy for this round. After then, you and your rabbit will be taken into two different rooms respectively.
- *Revealing Phase:* The host will reveal an n-node, undirected, simple graph G to you, and reveal the value of n to the rabbit. Note that a graph is simple if it does not contain any self-loops or multi-edges. The nodes are labeled with $\{0, 1, 2, ..., (n-1)\}$. No two nodes share the same label.
- *Communicating Phase:* You can write down a string s, but s should only contain '0' and '1'. After then, the host will give s to the rabbit, who knows nothing about the n-node graph.
- *Reconstructing Phase:* Based on the string s and the strategy you've discussed, the rabbit needs to reconstruct a graph G' in $O(n^2)$-time.

Rule for each round is listed as follows. Please describe your strategy for each round.
(Two n-node, undirected, labeled graphs G_1 and G_2 are identical, if $E_1 = E_2$.
$E_i = \{(u, v) | u < v,$ in G_i, the node labeled with u is adjacent to the node labeled with $v.\}$, that is, the edge list of G_i.)

Round 1 (2%)

Rules:

- The length of s must $\leq n^2$.
- You can pass this round if G' is identical to G.

Round 2 (4%)

Rules:

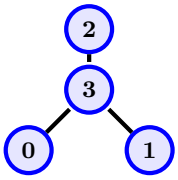
- G will be a tree.
- The length of s must $\leq n\lceil \lg n \rceil$.
- You can pass this round if G' is identical to G.

Round 3 (4%)

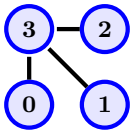
Rules:

- G will be a tree.
- The length of s must $\leq 2n$.

- You can pass this round if there is a way to relabel G', such that the relabeled graph is identical to G. (To relabel means to modify the labels on the nodes.) (The host will check whether such way exists or not.)

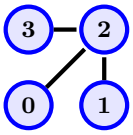
To make it more clear, when $G =$ 

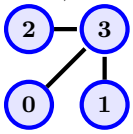
:

In round 1 or round 2, you can pass the round by reconstructing $G' =$ 

.

Note that G' is identical to G since $E' = E = \{(0,3),(1,3),(2,3)\}$

However, in round 3, $G' =$ 

is also a valid solution, because there is a way to relabel it into 

, which is identical to G.

For example, write **edge list** down as s is a strategy.

Every edge (u, v) can be represented as a 0/1-string of length $(2\lceil \lg n \rceil)$:

$$\{\text{Binary representation of } u\}\{\text{Binary representation of } v\}$$

Recall that you can always represent an integer between 0 and $n-1$ in $\lceil \lg n \rceil$ bits.

Thus, you can represent the whole E as a 0/1-string of length $(2|E|\lceil \lg n \rceil)$, write it down as s, and give it to your rabbit.

In the other hand, the rabbit can reconstruct the graph by parsing s into a list of edges in the same way, which may cost $O(n^2 \lg n)$ time.

Also, the maximum length of s in this strategy could be $n(n-1)\lceil \lg n \rceil$.

Thus, this strategy does not meet the length limit and the time limit in any of the three rounds, so you have to come up with some better strategies.

Problem 3. More about Red-Black Tree (17%)

In the lecture, you learned about insertion for Red-Black Tree (RB Tree) which involves a lot of case analysis. In this problem set, you will derive a similar operation.

It is the `meld` operation (yes, again)!

Given 2 RB Trees T_1, T_2 and a single element e with all keys in T_1 are less than $e.key$ and all keys in T_2 are bigger than $e.key$ ($T_1 < e < T_2$). `meld(T1, e, T2)` returns an RB Tree consists of $V(T_1) \cup \{e\} \cup V(T_2)$, where $V(T_i)$ denotes the vertices of T_i. An naïve way is to write down all elements from both trees and build an RB Tree back in $O(n \log n)$ time where n is the total number of nodes.

But in fact the operation can be done in $O(h_1 + h_2) = O(\log n)$ where h_1 (respectively, h_2) is height of T_1 (T_2).

To make things simpler, we assume that $h_1 \geq h_2$.

1. (1%) Given an RB Tree T, describe a $O(h(T))$ time algorithm to compute the `black-height` (`bh`) of root of T. $h(T)$ is height of T.

2. (2%) Given an RB Tree T with `bh` k and a non-negative integer k' which is less than k. Describe a $O(h(T))$ time algorithm to return the pointer to a node in T which has biggest key among all nodes with `bh` $k' + 1$.

3. (2%) In the previous sub-problem, prove that the node we get has a black right child node (it may be an external node) with `bh` k'.

4. (2%) Describe how to implement `meld(T1, e, T2)` in $O(1)$ time when T_1, T_2 have same `bh` for their root. Briefly explain the correctness (why the result is still a valid RB Tree).

5. (4%) Let k' be `bh` of T_2, y be black right child of the biggest node with `bh` $k' + 1$ and T_y be the sub-tree rooted at y (with `bh` k'). Then replace T_y by `meld(Ty, e, T2)` using 4.. What color should we put on e so that the entire tree satisfying RULE 1, 2, 3, 5 (see lecture slides for definition of RULEs)? and why your choice satisfies RULE 1, 2, 3, 5?

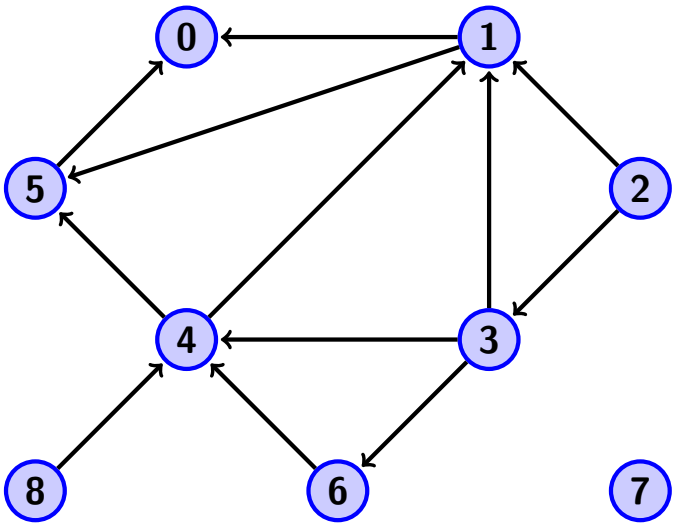
6. (3%) How should we fix the tree we built in 5. so that RULE 4 is satisfied? (Hint: Check how we fix e if it is inserted into T_1 along.) Briefly explain your approach, both pseudocode and figures are allowed.

7. (3%) Describe how to implement `meld(T1, e, T2)`, and show that `meld(T1, e, T2)` can be done in $O(h_1 + h_2)$ time.

Problem 4. Depth-First Search and Topological Sort (20%)

In this problem, we will play some tricks on the topological sort algorithm. In addition, in the following sub-problems, the graphs are stored in *adjacency lists*.

1. (4%) The following directed graph illustrated the relationship between the vertices. If there exists an directed edge from vertex x to vertex y, then it means that in the topologically sorted order, x must be ranked in front of y. Please give an topologically sorted sequence of the graph using DFS algorithm, and show the algorithm process by labeling $u.v$ and $u.d$ on each vertex u. Next, according to your $u.v$, $u.d$ labels, please mark all the *tree edges* as T, *back edges* as B, *forward edges* as F, and cross edges as C.



The followings are the procedures of DFS and DFS-VISIT.

```
1  DFS(G):
2      u = [sequence of all vertices]
3      for(int i=0;i<n;i++)
4          u[i].color = WHITE
5          u[i].pi = NIL
6      time = 0
7      cnt = 0
8      for(int j=0;j<n;j++)
9          if u[j].color == WHITE
10             DFS-VISIT(G, u[j])
11             cnt++;
```

```
1  DFS-VISIT(G, u):
2      time += 1
3      u.d = time
4      u.color = GRAY
5      for each v in G.adj[u]
6          if v.color == WHITE
7              v.pi = u
```

```
8               DFS-VISIT(G, v)
9       u.color = BLACK
10      time += 1
11      u.f = time
```

2. (4%) When solving the topological sort problem, we might call `DFS-VISIT` from `DFS` several times. The frequency of the function calls of `DFS-VISIT` triggered by `DFS` is stored in the variable `cnt` in line 7 of `DFS`. For example, it is possible for `cnt` to be 5 or more to finish the whole DFS process in the graph above. The final value of `cnt` highly depends on the sequence of u in line 2 of `DFS` function. Are there any approaches to find out a sequence of u in order to minimize the final value of `cnt` on the previous graph? Please give such sequence of u, describe how you found it, briefly justify it, and show the minimum final value of `cnt` of the previous graph.

3. (2%) Please give a directed graph with less than 5 nodes which cannot be topologically sorted, and briefly explain why.

4. (5%) Based on the idea of DFS and topological sort, please design an algorithm to decide whether an **undirected** graph with V vertices contains a *loop*. Your algorithm should run in $O(V)$ time, and you should explain why your algorithm works and why it satisfies the time complexity constraint. Solutions without explanations will **NOT** get any credits.

The following problems are *Prove or Disprove* problems. If you think the statement is correct, please prove it; if you think the statement is wrong, please give a counter example to disprove that statement. Answers without any proof or counter example will **NOT** get any credits.

5. (3%) Given a directed binary tree T_1, with all the edges from parent to child, the preorder traversal sequence of T_1 will be a topologically sorted sequence of T_1.

6. (2%) Given a directed complete binary tree T_2 with V vertexes sequentially indexed from top to bottom, from left to right, and the indexes start from 0, then the sequence $(0, 1, 2, ..., V-1)$ will be a topologically sorted sequence of T_2.

Problem 5. Course Suggestion and Evaluation (Bonus 20%)

The semester finally comes to an end. As mentioned in class, please provide some constructive suggestions and feedback for us. All kinds of feedback are welcome. Both English and Mandarin are accepted. Your suggestions can make us improve in the DSA course next year!

Programming Part

Problem 6. Super Typhoon (20% + Bonus 1%)

Arvin is the mayor of DerTian City. Looking downward from the sky, this city is a $N \times M$ grid divided into N rows and M columns, and in each grid there is a house.

Last month, the super typhoon *Joey* destroyed all the houses in the city, and Arvin is planning to rebuild them. To avoid the city looks too different from the original, he wants to make sure that for every two houses A and B in the same row or same column,

- if A is originally taller than B, then A is still taller than B after the rebuild
- if A is originally as tall as B, then A is still as tall as be after the rebuild.

Also, to make things simple, he hopes that the number of different house heights are as few as possible.

Given the original heights of all houses, please find out the least possible number of different house heights after the rebuild.

Input Format

The first line is two integers N and M, indicating the number of rows and columns respectively ($N \times M \leq 10^5$).

The i^{th} row of the following N rows contains M integers $H_{i1}, ..., H_{iM}$, where H_{ij} indicates the original heights of house on the i^{th} row and the j^{th} column. It's guaranteed that $0 \leq H_{ij} \leq 2147483647$ for all (i, j).

Output Format

Output an integer that is the least possible number of different house heights after the rebuild.

Subtasks

15%	$N = 1$ or $M = 1$.
20%	$H_{ij} > H_{(i-1)j}$ and $H_{ij} > H_{i-1(j-1)}$.
30%	$N \leq 100$, $M \leq 100$, and $H_{i_1 j_1} \neq H_{i_2 j_2}$ for every $(i_1, j_1) \neq (i_2, j_2)$.
35%	$H_{i_1 j_1} \neq H_{i_2 j_2}$ for every $(i_1, j_1) \neq (i_2, j_2)$.
5%	(Bonus) no additional constraints.

Sample Input

```
2 3
8 12 17
16 20 7
```

Sample Output

```
3
```

Explaination

In the sample input/output, one of the best approaches to rebuild the city is as follows:

```
8 12 17
12 17 8
```

This approach ends up with 3 different house heights (8, 12 and 17), so the answer is 3.

Hints

Before reading the following hints, we encourage you to think about this problem on your own for a while, and at least get some feeling of what makes it difficult.

Consider building a graph as follows:

- Every cell (i, j) in the $N \times M$ grid is a vertice v_{ij}.
- In every row, say the i^{th}, if the the original house heights, sorted in descending order, are $H_{ij_1}, H_{ij_2}, ..., H_{ij_M}$, then we add $M - 1$ edges $(v_{ij_1}, v_{ij_2}), (v_{ij_2}, v_{ij_3}), ..., (v_{ij_{M-1}}, v_{ij_M})$ to the graph.

It turns out that the answer will be the length of the **longest path** on such a graph!

It's not hard to find that there won't be any cycle in this graph, so there must be some vertices that have **NO** incoming edge. We may label each of them with number 0, meaning that it takes **at most** 0 steps to get from other vertices to it. We then eliminate all these vertices, and after that some of the other vertices may become incoming-edge-free. For each of such vertices, it takes **at most** 1 step to get from some other vertice to it (think why). Thus similarly, we may label them with number 1, eliminate them all and goes on. After all the vertices have been labeled, the length of the longest path is the largest label assigned in this process.

***Problem** 7.* World Lines Again (20%)

Scientists aim high, while mad scientists aim higher. After your help last time, Kyōma wants to learn more about the mystery of world lines in hope to fully wield his powers.

To achieve this, Kyōma seeks your help once again. This time he provides you a list of events along with the maximum times each can be triggered. Now he wants you to figure out whether a certain world line is reachable given the current world line. If possible, he would also like you to calculate the least number of events need to be triggered to reach target world line.

Formula

The next world line = Current world line $\oplus$ the event triggered. ($\oplus$ means the steins;gate's choice, more commonly referred to as XOR.)

Input Format

The first line is an integer N, indicating the number of triggerable events.

Following N lines are the integers representing these events and the maximum times it can be triggered.

The next line contains an integer Q, indicating the number of queries.

In the last Q lines, each line contains two integers, indicating the current world line and the target world line respectively.

Input Constraint

- $1 \leq N \leq 50$
- $1 \leq Q \leq 10^6$
- $0 \leq worldline\&event \leq 2^{19}$

Output Format

For each query, please output one line containing the minimum count of events needed to be triggered to reach target.

If target world line is not reachable, output UNREACHABLE instead.

Score Distribution

- for 20% testdata: $1 \leq N \leq 11$
- No additional constraint otherwise

Sample Input

```
2
5 2
4 1
3
0 5
1 0
```

```
1 2
```

Sample Output

```
1
2
UNREACHABLE
```