Data Structure and Algorithm, Spring 2024
Homework 2

Due: 13:00:00, Tuesday, April 23, 2024

BLUE CORRECTION: 20:30:00, April 3, 2024

RED CORRECTION: 06:15:00, March 28, 2024

TA E-mail: `dsa_ta@csie.ntu.edu.tw`

Rules and Instructions

- Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

- Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources (including chatGPT) can be consulted, but not copied from.

- Since everyone needs to write the final solutions alone, there is **absolutely no need to lend your homework solutions and/or source codes to your classmates at any time.** In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

- In Homework 2, the problem set contains a special Problem 0 (see below) and 4 other problems. The set is divided into two parts, the non-programming part (Problems 0, 1, 2) and the programming part (Problems 3, 4).

- For problems in the non-programming part, you should combine your solutions in *one* PDF file. Your file should generally be legible with a white/light background—using white/light texts on a dark/black background is prohibited. Your solution must be as simple as possible. At the TAs' discretion, solutions which are too complicated can be penalized or even regarded as incorrect. If you would like to use any theorem which is not mentioned in the classes, please include its proof in your solution.

- The PDF file for the non-programming part should be submitted to Gradescope as instructed, and you should use Gradescope to tag the pages that correspond to each sub-problem to facilitate the TAs' grading. Failure to tagging the correct pages of the sub-problem can cost you a 20% penalty.

- For problems in the programming part, you should write your code in C programming language, and then submit the code via the *DSA Judge*:

  https://dsa2024.csie.org/.

  Each day, you can submit up to 5 times for each problem. The judge system will compile your code with

  ```
  gcc main.c -static -O2 -std=c11
  ```

- For debugging in the programming part, we strongly suggest you produce your own test-cases with programs and/or use tools like `gdb` or compile with flag `-fsanitize=address` to help you solve issues like illegal memory usage. For `gdb`, you are strongly encouraged to use compile flag `-g` to preserve symbols after compiling.

  Note: For students who use IDE (VScode, DevC++...) you can modify the compile command in the settings/preference section. Below are some examples:

  – `gcc main.c -O2 -std=c11 -fsanitize=address`
    `# Works on Unix-like OS, but possibly not M1/M2`

  – `gcc main.c -O2 -std=c11 -g # use it with gdb`

- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

$$\mathit{Late\ Score} = \max\left(\left(\frac{5 \times 86400 - \mathit{Delay\ Time}(\mathit{sec.})}{5 \times 86400}\right) \times \mathit{Original\ Score}, 0\right)$$

- If you have questions about HW1, please go to the Discord channel and discuss (*strongly preferred*, which will provide everyone with a more interactive learning experience). If you need an email answer, please follow the rules outlined below to get a fast response:

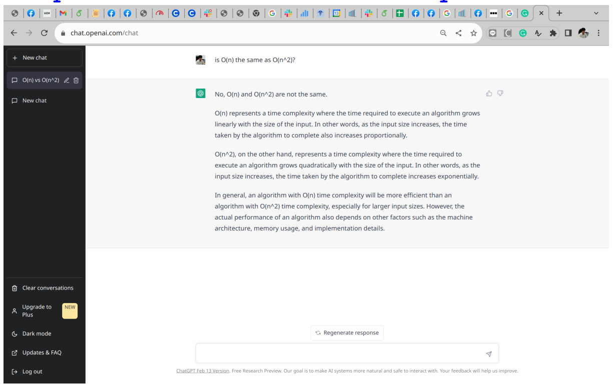
  – Please include the following tag in the subject of your email: `"[HW2.Px]"`, specifying the problem where you have questions. For example, `"[HW2.P2] Is k in subproblem 5 an integer?"`. Adding the tag allows the TAs to track the status of each email and to provide faster responses to you.

  – If you want to provide your code segments to us as part of your question, please upload it to Gist or similar platforms and provide the link. Please remember to protect your uploaded materials with proper permission settings. Screenshots or code segments directly included in the email are discouraged and may not be reviewed.

Non-Programming Part

Problem 0 - Proper References (0 pts)

For each problem below, please specify the references (the Internet URL you consulted with or the classmates/friends you discussed with) in your PDF submitted to Gradescope. If you have used chatGPT or similar tools, please provide a quick snapshot of your interaction with the tools. *You do not need to list the TAs/instructors.* If you finished any problem all by yourself (or just with the help of TAs/instructors), just say "all by myself." While we did not allocate any points to this problem, failure to complete this problem can lead to penalty (i.e. negative points). Examples are

- Problem 1: Alice (B86506002), Bob (B86506054), and
  https://stackoverflow.com/questions/982388/
  
  chat.openai.com/chat

  is O(n) the same as O(n^2)?

  No, O(n) and O(n^2) are not the same.

  O(n) represents a time complexity where the time required to execute an algorithm grows linearly with the size of the input. In other words, as the input size increases, the time taken by the algorithm to complete also increases proportionally.

  O(n^2), on the other hand, represents a time complexity where the time required to execute an algorithm grows quadratically with the size of the input. In other words, as the input size increases, the time taken by the algorithm to complete increases exponentially.

  In general, an algorithm with O(n) time complexity will be more efficient than an algorithm with O(n^2) time complexity, especially for larger input sizes. However, the actual performance of an algorithm also depends on other factors such as the machine architecture, memory usage, and implementation details.

  ChatGPT Feb 13 Version. Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.

- Problem 2: all by myself

- ...

Listing the references in this problem does *not* mean you could copy from them. We cannot stress this enough: *you should always write the final solutions alone and understand them fully.*

Problem 1 - Traveling to Tohoku (100 pts)

For all sub-problems listed below, unless stated otherwise, please ensure that your answers consist of the following components:

- Please describe your algorithm using pseudocode and provide a **brief explanation** of why it works. Your pseudocode for each sub-problem should be concise and easy to read, **consisting of no more than 30 lines**. Please note that overly complex or lengthy pseudocode may result in some penalties.
- Analyze the time complexity and extra-space complexity of your algorithm. Please note that if your algorithm is not efficient in terms of time or space, you may not receive full credit for your solution.

Discovering Sendai Adventures (DSA) is a renowned tourism group organized by the *National Travel Union* (NTU) in the Tohoku area of Japan. Every year, hundreds of students sign up for the tour. When a student wanted to buy the tour ticket, ze discovered that the ticket booking website had implemented a complex reCAPTCHA system to prevent ticket scalping by bots. To successfully purchase the ticket, ze must first solve the following two sub-problems.

1. (15 pts) Convert the infix expression below into a postfix expression, with the precedence of () over $\sim$ (bitwise not) over & (bitwise and) over $\wedge$ (bitwise xor, exclusive or) over | (bitwise or). All operations above are left-associative. Note that $\sim$ (bitwise not) is a unary operation and others are binary operations. Briefly explain your human evaluation procedure and its execution steps *with plain text*, without any complexity analysis. :-)

$$\sim (\mathrm{A}\ \&\ 5) \mid (6 \wedge 9\ \&\ \sim 1) \wedge ((4 \wedge \mathrm{C}) \mid 8)$$

2. (15 pts) Evaluate the outcome of the following postfix expression, where each number is a four-bit unsigned integer represented in the usual hexadecimal form

$$0, 1, 2, 3, \cdots, 9, \mathrm{A}, \mathrm{B}, \cdots, \mathrm{F}.$$

The bitwise operations manipulate the four bits. For example,

$$\sim \mathrm{A} = \sim (1010)_2 = (0101)_2 = 5$$

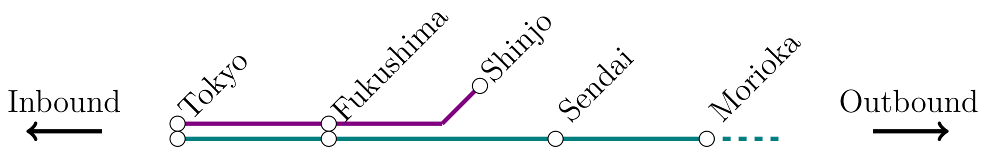
and

$$9 \wedge 5 = (1001)_2 \wedge (0101)_2 = (1100)_2 = \mathrm{C}.$$

Briefly explain your human evaluation procedure and its execution steps *with plain text*, without any complexity analysis. :-)

$$7, \mathrm{C}, 8, \;\&\;, 5, \wedge, \sim, \mathrm{C}, \sim, 9, |, \mathrm{D}, 4, \sim, \wedge, \;\&\;, |, \wedge$$

With your assistance, the student successfully booked the ticket for *Discovering Sendai Adventures* and embarked on zir journey through Tohoku area. One thing that ze wants to figure out is when the Shinkansen (bullet train) will depart from **Morioka**. Ze only has the map below, the outbound Shinkansen timetable for **Fukushima**, and the following rules.



The route map of Shinkansen

- The outbound Shinkansen arriving in **Fukushima** will split to two Shinkansen's. One of them will head to **Shinjo** and the other will head to **Morioka**. The Shinkansen will take T_s and T_m minutes, respectively, to arrive **Shinjo** and **Morioka**. Note that $T_s < T_m$ because **Shinjo** is closer to **Fukushima** than **Morioka**.

- The outbound Shinkansen arriving at each destination (**Shinjo** or **Morioka**) will immediately turn back. The returning (inbound) Shinkansen will take the same amount of time as the outbound Shinkansen.

- Every Shinkansen returning from **Shinjo** must connect with one and only one Shinkansen returning from **Morioka** at **Fukushima** before leaving. If there is already a Shinkansen from **Morioka** arriving at exactly the same time at **Fukushima**, then the returning Shinkansen from **Shinjo** will connect with the arriving train and leave. Somehow because the Shinkansen from **Morioka** takes longer to return, it is possible that some Shinkansen returning from **Shinjo** cannot connect with any Shinkansen returning from **Morioka** immediately, causing complaints from unnecessary passenger waiting.

  To resolve the complaints, the Shinkansen operator will call an additional Shinkansen to depart from **Morioka** T_m minutes earlier than the returning time of the Shinkansen from **Shinjo** to the **Fukushima** station. Then, the additional Shinkansen from **Morioka** and the returning Shinkansen from **Shinjo** can connect and leave immediately.

  For example, if $T_m = 1$, and the returning time of a Shinkansen at **Fukushima** is `11:26`. Unless there is a Shinkansen returning from **Morioka** at `11:26` exactly, an additional Shinkansen will be called to depart from **Morioka** at time `11:25` to connect with the Shinkansen returning from **Shinjo** at time `11:26`.

Given the outbound Shinkansen timetable for **Fukushima**, please help the student determine the timetable for the Shinkansen's that depart from **Morioka**.

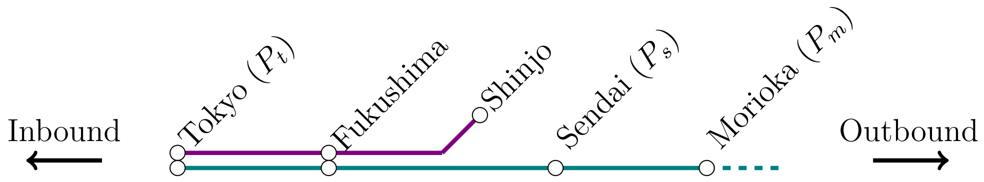
3. (10 pts) Given the outbound Shinkansen timetable for **Fukushima** as leaving at time steps (minutes) $0, 2, 5, 8$, and the knowledge of $T_m = 6$ and $T_s = 2$, calculate the timetable for the Shinkansen's that depart from **Morioka**. Briefly explain your human evaluation procedure and its execution steps *with plain text*, without any complexity analysis. :-)

4. (20 pts) Assume that the outbound Shinkansen leaves **Fukushima** at time steps $S[i]$, where S is an ordered array of size n. Please design an $O(n)$ time and $O(n)$ extra-space algorithm to calculate the timetable for the Shinkansen's that depart from **Morioka**, sorted by departure time.

The largest snowstorm in a century has struck the Tohoku area of Japan. Outside, the wind and snow are raging, leaving only one operational track for the inbound Shinkansen. The one operational track prevents any Shinkansen from passing any other Shinkansen before reaching **Sendai**. Trapped in the slower Shinkansen, the student wants to calculate the fastest Shinkansen that brings zir inbound from **Morioka** to **Tokyo** (after passing **Sendai**) using the position and speed of every Shinkansen, as well as the map that we provide below (which is the same as the previous map).

In particular, assume that the stations of **Morioka**, **Sendai**, and **Tokyo** are located at P_m, P_s, P_t with $0 < P_m < P_s < P_t$. Consider n trains that starts at positions p_i with a constant speed v_i heading towards **Tokyo** (P_t). That is, the original arrival time to **Tokyo** should be $(P_t - p_i)/v_i$. Because of the single-track operation, some faster Shinkansen j may catch up with a slower Shinkansen k at a position within $[0, P_s]$ (i.e. before **Sendai**). If that happens, Shinkansen j will need to slow down from speed v_j to speed v_k, and will keep the new speed before reaching P_t (**Tokyo**).

Every Shinkanshen will stop at P_m, P_s, P_t because they are all large stations. The student is interested in the time required between P_m (**Morioka**) and P_t (**Tokyo**).



The route map of Shinkansen

5. (40 pts) Given $\{(p_i, v_i)\}_{i=1}^{n}$ for the n Shinkansen's sorted by their initial position p_i, as well as P_m, P_s, P_t, please design an $O(n)$ time and $O(n)$ extra-space algorithm to calculate the fastest train from **Morioka** to **Tokyo**, given the one-track-operation constraint before **Sendai**. It is guaranteed that the student can arrive **Tokyo** by the Shinkansen.

Problem 2 - Bob the Builder (100 pts)

Bob is a skillful tree-house builder at *National Tree-house Union* (NTU). He was entrusted with a crucial mission, one that demanded his leadership to guide the team in constructing the magnificent *Twin Star Tree-House*. However, Bob went to the *DSA Night* (Didn't Sleep All Night) and spent the entire night without sleeping, leaving him drained and unable to to build the tree-house. Can you, who fortunately got away from the *DSA Night* because you always start working on your homework earlier :-), help Bob build the tree-house?

1. (10 pts) To ensure that you are capable of leading the *Twin Star Tree-House* team, Sam, the head-boss of NTU, first challenges you the basic understanding of tree-house. For this sub-problem and the next one, please construct all possible binary trees from the given two traversals and draw them out. No other explanations are needed.
    - Pre-order traversal: EABGCDHI
    - In-order traversal: BGACEDIH
2. (10 pts)
    - Pre-order traversal: ABECD
    - Post-order traversal: EDCBA

For all sub-problems listed below, unless stated otherwise, please ensure that your answers consist of the following components:

- Please describe your algorithm using pseudocode and provide a **brief explanation** of why it works. Your pseudocode for each sub-problem should be concise and easy to read, **consisting of no more than 30 lines**. Please note that overly complex or lengthy pseudocode may result in some penalties.
- Analyze the time complexity and extra-space complexity of your algorithm. Please note that if your algorithm is not efficient in terms of time or space, you may not receive full credit for your solution.

3. (15 pts) Sam claimed that:*Given an in-order traversal, and either a pre-order or a post-order traversal, a unique binary tree can be constructed.* Is Sam correct? If so, design an algorithm to do so (without analyzing the time complexity); if not, provide a counter-example.

4. (25 pts) Given the pre-order and the post-order traversals of some binary tree with n nodes, design an $O(n^2)$-time and $O(1)$-extra-space algorithm to calculate the number of all possible binary trees that satisfy those traversals. You can assume that the given traversals are valid (*i.e.* can at least construct one binary tree) and have no duplicated key.

After finishing the challenges above, Sam, the head-boss of NTU, finally recognizes your capabilities of building the tree-house. Ze then introduces the details of the *Twin Star Tree-House* to you. *Twin Star Tree-House* contains two ***mirrored*** trees, which are defined as follows.

Two binary trees T_1 and T_2 are regarded as ***mirrored*** iff

1. The root values of both trees are the same.
2. If either the left sub-tree of T_1 or the right sub-tree of T_2 is non-empty, then both must be non-empty and ***mirrored***.
3. If either the right sub-tree of T_1 or the left sub-tree of T_2 is non-empty, then both must be non-empty and ***mirrored***.

And yes, the termination condition! Two leaves are regarded as ***mirrored*** iff the values of both leaves are the same.

However, some part of the magnificent *Twin Star Tree-House* are misconstructed. Ze asked if you can help zir fix it. Representing the *Twin Star Tree-House* as a mirrored tree T_1 and T_2 with nodes and edges, can you modify a node in T_1 while also maintaining T_2 as a mirrored tree of T_1?

5. (20 pts) Given a **complete binary tree** T_1 and its mirrored tree T_2 (which is not necessarily complete), each with n nodes. As illustrated in our class, the **complete binary tree** can be indexed by numbering the root node as index 1, its left child at index 2, the root's right child at index 3, and so on. We will assume that T_1 is represented with those indexed nodes. Given m operations (a_i, v_i), each changing the a_i-th node of T_1 to value v_i, design a $O(m \log n)$-time and $O(\log n)$-extra-space algorithm that modifies T_2 to maintain its ***mirroring*** property to T_1.

The definition of the mirrored tree can be extended to k-ary trees, where each tree has k **indexed** children instead of 2.

Two k-ary trees T_1 and T_2 are regarded as ***mirrored*** iff

1. The root values of both trees are the same.
2. For any integer i with $1 \leq i \leq k$, if either the i-th sub-tree of T_1 or the $(k-i+1)$-th sub-tree of T_2 is non-empty, then both must be non-empty and ***mirrored***.

Two leaves are regarded as ***mirrored*** iff the values of both leaves are the same.

6. (20 pts) The city now decides to build the magnificent *Twin Star Tree-House* by k-ary trees T_1 and T_2 instead of binary. They will consider T_1 to be a **complete** k-ary tree, whose nodes can be indexed like complete binary trees. In particular, the root will be of index 1, and its children will be of indices 2 to $k+1$, and the children of the root's first child will be of index $k+2$ to $2k+1$, and so on. Consider a complete k-ary tree T_1 and its mirrored tree T_2 (which is not necessarily complete), each with n nodes. Assume that the nodes are stored in a linked manner by the following C-like structure.

```
struct Node {
    int value;
    Node* children[k];
}
```

Given m operations (a_i, v_i), each changing the a_i-th node of T_1 to value v_i, design a $O(m \log n)$-time and $O(1)$-extra-space algorithm that modifies T_2 to maintain its ***mirroring*** property to T_1.

Programming Part

Problem 3 - Printing Queue (100 pts)

Problem Description

In this problem, we ask you to design a system to emulate the job queues of several printers within the *Computer Science and Information Engineering* (CSIE) department of *National Taiwan University* (NTU). Yes, we have more natural acronyms too! :-) This system needs to support the following three operations:

- Add Operation: Insert a job into the printer's queue.
- Print Operation: Print the document with the highest priority from the queue. This is a typical need for the priority queue (i.e. what the binary heap is designed for).
- Move Operation: Transfer all jobs from one printer to another. This happens when one printer "fails" for some reason, such as running out of paper or toner.

Although the add and print operations can be done efficiently with the binary heap data structure that we taught in class, the binary heap cannot execute the move operation very efficiently. In particular, the move operation takes $O(n \log(m+n))$ with the binary heap when moving n tasks to a printer that carries m tasks by running n insertions naïvely.

To design the system efficiently, we suggest you to learn related data structures that can handle the move operation efficiently. One possibility is the binomial heap (but there are also other possibilities that you can consider). The following description of the binomial heap is modified from Wikipedia (`https://en.wikipedia.org/wiki/Binomial_heap`):

A binomial min-heap is implemented as a set of binomial min-trees. Each binomial tree is defined recursively as follows:

- *(termination) binomial tree of order* 0 *is a single node.*
- *(recursion) A binomial tree of order* k *has a root node whose children are roots of binomial trees of orders* $k-1$, $k-2$, …, 2, 1, 0, *usually sorted reversely by its order.*

Then, the binomial min-heap can be defined as a set of binomial trees, as follows:

- *There can be at most one binomial tree for each order, including the zero-th order.*
- *Each binomial tree is a min-tree. That is, the key of the root of any sub-tree is smaller than or equal to the key of any of its descendants.*

The second property ensures that the root of each binomial tree contains the smallest key in the tree. It follows that the smallest key in the entire heap is one of the roots.

The first property implies that a binomial heap with n nodes consists of at most $1 + \log_2 n$ binomial trees (so locating the minimum key takes $O(\log n)$ time). The number and orders of these trees are uniquely determined by the number of nodes n: there is one binomial tree for each nonzero bit in the binary representation of the number n. For example, the decimal number 13 *is* $(1101)_2$*, and thus a binomial heap with* 13 *nodes will consist of three binomial trees of orders* 3*,* 2*, and* 0*.*

The core of the binomial heap is an efficient BinomialHeapUnion algorithm that merges two binomial heaps. The pseudo code (modified from an earlier version of the textbook) of the algorithm is listed in the next page. The algorithm contains an additional step of BinomialHeapMerge that merges the ordered linked list of binomial trees within binomial heaps H_1 and H_2 into another ordered linked list. The step is similar to the sparse vector summing algorithm that you have learned in Lecture 6. :-) Once you have implemented BinomialHeapUnion, you can conduct the following operations efficiently.

BinomialHeapInsert(H, x)

To insert a node x into a binomial heap H, you can create a new binomial heap H' containing only the node x, and then run BinomialHeapUnion(H, H').

BinomialHeapExtractMin(H)

To extract the node with the minimum priority, you can first find the binomial tree T with the minimum key in the root list of H. Then, you can remove T from the root list of H, remove $T.root$ from T, and then run BinomialHeapUnion$(H, \text{Reverse}(T.children))$.

```
BinomialHeapUnion($H_1, H_2$)
1  $H$ = MakeEmptyBinomialHeap()
2  $head[H]$ = BinomialHeapMerge($H_1, H_2$)
3  if $head[H]$ = NIL
4      return $H$
5  $prev$ = NIL
6  $x = head[H]$
7  $next = sibling[x]$
8  while $next \neq$ NIL
9      if ($degree[x] \neq degree[next]$) or ($sibling[next] \neq$ NIL and $degree[sibling[next]] = degree[x]$)
10         $prev = x$
11         $x = next$
12     else
13         if $key[x] \leq key[next]$
14             $sibling[x] = sibling[next]$
15             insert $next$ as a child of $x$, and update $degree[x]$
16         else
17             if $prev$ = NIL
18                 $head[H] = next$
19             else
20                 $sibling[prev] = next$
21             insert $x$ as a child of $next$, and update $degree[next]$
22             $x = next$
23     $next = sibling[x]$
24 return $H$
```

In this problem, we suggest you implement the binomial *max-heap* instead of the min-heap. That is, the key of the root of any sub-tree is greater than or equal to the key of any of its descendants.

Input

Assuming that the printers' queues are initially empty, the input begins with two space-separated integers N and M, representing the number of printers and the total number of operations, respectively. Printers are 1-indexed. Each of the next M lines is given in one of the following formats:

- `1 job_id priority printer_id`: add operation
  Insert a job with `job_id` into the queue of printer `printer_id`, assigning it the priority `priority`.
- `2 printer_id`: print operation
  Instruct printer `printer_id` to print the document with the highest priority from its queue, removing the job from the queue in the process.
- `3 printer_id1 printer_id2`: move operation
  Transfer all jobs from printer `printer_id1` to printer `printer_id2`, maintaining their respective priorities.

Output

- For each **add** operation, print a line that indicates the number of jobs waiting on printer `printer_id` in the format:

  ```
  [num_of_jobs] jobs waiting on printer [printer_id]
  ```

- For each **print** operation, if there is a job to be printed, print a line of

  ```
  [job_id] printed
  ```

  Otherwise (the queue is empty), print a line of

  ```
  no documents in queue
  ```

- For each **move** operation, print a line that indicates the number of jobs waiting on printer `printer_id2` after moving.

  ```
  [num_of_jobs] jobs waiting on printer [printer_id2] after moving
  ```

Constraints

- $1 \leq N, M \leq 10^6$
- $1 \leq$ `printer_id`, `printer_id1`, `printer_id2` $\leq N$
- $1 \leq$ `job_id`, `priority` $\leq 10^9$
- For the **move** operation, `printer_id1` $\neq$ `printer_id2`
- It is guaranteed that all `priority` and `job_id` will be distinct, and all operations will be legal (i.e., all IDs will be within the specified range).

Subtasks

Subtask 1 (20 pts)

- $N = 1$
- Naturally, there are no move operations when $N = 1$. That is, the task can be solved with the binary heap taught in class.

Subtask 2 (20 pts)

- $1 \leq N, M \leq 5000$

Subtask 3 (60 pts)

- no other constraints

Sample Testcases

Sample Input 1

```
1 5
1 1 1 1
1 2 2 1
2 1
2 1
2 1
```

Sample Output 1

```

1 jobs waiting on printer 1
2 jobs waiting on printer 1
2 printed
1 printed
no documents in queue
```

Sample Input 2

```
2 15
1 1 5 1
1 2 4 1
1 3 3 1
1 4 2 2
1 5 1 2
3 1 2
2 1
2 2
2 2
3 2 1
2 2
2 1
2 1
2 1
2 1
```

Sample Output 2

```

1 jobs waiting on printer 1
2 jobs waiting on printer 1
3 jobs waiting on printer 1
1 jobs waiting on printer 2
2 jobs waiting on printer 2
5 jobs waiting on printer 2 after moving
no documents in queue
1 printed
2 printed
3 jobs waiting on printer 1 after moving
no documents in queue
3 printed
4 printed
5 printed
no documents in queue
```

Problem 4 - Dungeons of Sacred Abyss (100 pts + 20 bonus pts)

Problem Description

Dungeons of Sacred Abyss is a brand-new game developed by the software company *Nextgen Technology Universal Company of Software* (NTUCS) in order to fill the video game market after **Dragon's Song: Awakening** has failed due to power inflation. This new game is designed to give players a unique immersive experience while discovering treasures through traversing the magnificent dungeons and the beauty of the sacred abyss.

In each game, the map consists of N dungeons and the player is initially located at dungeon 0. After the game starts, M pathways between dungeons u_i, v_i where $u_i < v_i$ with length ℓ_i, will appear one by one. In the regular version of the game, all pathways will be given before the player starts traversing, but in the bonus version of the game, some pathways might appear during the game. Interestingly, for each dungeon j except dungeon 0, at most one pathway with $v_i = j$ ever exists. That is, each dungeon has a unique "upstream" dungeon. The pathways are bi-directional and will always be accessed by moving downstream first before going back upstream. After the player goes upstream, the pathway will fall apart and disappear. At any timestamp, it is guaranteed that dungeon 0 can be reached from the player's current dungeon by repeatedly going to the upstream dungeon.

During the player's traversal, ze might find some valuable treasure in the current dungeon. Since the dungeons are buried deep underground, the game includes a transportation system to ship the treasures upstream. The system operates under the constraint that each dungeon can store at most one treasure. The constraint is maintained as follows. The system checks if the newly-found treasure can be stored in the current dungeon. If so (i.e. the dungeon is empty), the treasure is stored. Otherwise, the system places the newly-found treasure in the current dungeon, and push the originally-stored treasure to the upstream dungeon. The process (of checking whether the upstream dungeon can store the treasure) is repeated until a empty dungeon is reached, or until dungeon 0 is reached. If an empty and non-zero dungeon is reached, the last-pushed treasure is stored. Dungeon 0 is a special dungeon that cannot be used to store any treasure. Thus, if dungeon 0 is reached, the treasure will be escorted out of the system immediately.

The transportation system operates at a cost. In particular, transporting the treasure through an upstream pathway decreases its value by the length of the pathway. The value may even drop to a negative number. The transportation system will not make the pathway disappear though—only the upstream action of the player can make the pathway disappear.

The game will give player Q instructions on what to do and you, as the player, should follow them exactly. There are six kinds of instructions that may be given to you.

1. **downstream**: go to a downstream dungeon that appeared first (when the map is constructed, or in the order of any **construct** instructions below). Print out the dungeon number. If no such dungeon exists, print `-1`.

2. **upstream**: go to the unique upstream dungeon. Print out the dungeon number. If no such dungeon exists (i.e. you are in dungeon 0), print `-1`. Note that going upstream makes the pathway disappear.

3. **plan**: if a *hypothetical* treasure of value t_i appears in the current dungeon, it will gradually be pushed upstream step by step when future treasures appear. During its upstream journey, the treasure will gradually lose its value. Print out the *smallest* dungeon number where the treasure can still maintain a non-negative value along the upstream journey. In addition, given that this is an instruction to **plan**, no actual treasure gets stored in the dungeon.

4. **guess**: assume that a *hypothetical* treasure of value 0 has been transported to the current dungeon step by step along some upstream journey, without considering whether any dungeon stores a treasure or not, print out the maximum original value of the treasure. Only the pathways that appeared (when the map is constructed, or with any previous **construct** instructions) and have not disappeared (with the **upstream** instruction) need to be considered. Note that the original value of the treasure can also be 0, representing a possible treasure that appears in the current dungeon. In addition, given that this is an instruction to **guess**, no actual treasure gets stored in the dungeon.

5. **discover**: A treasure with value p_i is discovered in the current dungeon—the transportation system will work on storing it.

6. **construct** (the bonus instruction): a new pathway of length ℓ_i from the current dungeon to dungeon v_i appears.

Input

The first line consists of three space-separated integers, N, M, Q. The next M lines each consists of three space-separated integers u_i, v_i, ℓ_i, representing the initial map.

The final Q lines each consists of space-separated integers depending on the instruction given.

- 1, indicating instruction 1 is given.
- 2, indicating instruction 2 is given.
- 3 t_i, indicating instruction 3 is given.
- 4, indicating instruction 4 is given.
- 5 p_i, indicating instruction 5 is given.
- 6 v_i ℓ_i, indicating instruction 6 is given.

Output

- For instructions 1, 2, 3 and 4, print a line as discussed above.
- For instruction 5, if some treasure is escorted out of the system at dungeon 0, print a line as follows.
  - If the value of the treasure has decreased to a negative number when being escorted, print
    `value lost at [dungeon where treasure value first became negative]`
  - If the value of the treasure is still non-negative, print
    `value remaining is [value]`

  Otherwise, nothing needs to be printed out.
- For instruction 6, no printing is needed.

Constraints

- $2 \leq N \leq 10^6$, $0 \leq M \leq N - 1$
- $1 \leq Q \leq 10^6$
- $0 \leq u_i < v_i < N$, and all the v_i throughout the M initial pathways and the pathways given by instruction 6 are unique.
- $0 < \ell_i \leq 10^9$
- $0 \leq t_i \leq 10^{18}$
- $0 \leq p_i \leq 10^{18}$
- $0 \leq$ current dungeon index $< v_i < N$
- $M +$ (number of type-6 instructions) $< N$

Subtasks

Subtask 1 (10 pts)

- $M = N - 1$
- Only instructions 1 and 2 are given.

Subtask 2 (15 pts)

- $M = N - 1$
- Only instructions 1, 2 and 3 are given.

Subtask 3 (25 pts)

- $M = N - 1$
- Only instructions 1, 2 and 4 are given.

Subtask 4 (25 pts)

- $M = N - 1$
- Only instructions 1, 2, 3 and 5 are given.

Subtask 5 (25 pts)

- $M = N - 1$
- Only instructions 1, 2, 3, 4 and 5 are given.

Subtask 6 (20 bonus pts)

- No other constraints.

Sample Testcases

Sample Input 1

```
4 3 8
0 1 2
0 2 5
1 3 2
1
3 3
1
1
3 4
3 0
2
2
```

Sample Output 1

```
1
0
3
-1
0
3
1
0
```

Sample Input 2

```
4 3 11
0 2 3
0 1 7
0 3 5
4
1
2
4
1
2
4
1
2
4
1
```

Sample Output 2

```
7
2
0
7
1
0
5
3
0
0
-1
```

Sample Input 3

```
6 5 11
0 1 5
1 2 4
2 3 3
3 4 2
4 5 1
1
5 10
1
5 5
1
1
5 30
5 25
5 50
2
5 15
```

Sample Output 3

```
1
2
3
4
value remaining is 5
3
value lost at 0
```

Sample Input 4 (Bonus)

```
5 2 13
0 1 4
1 4 5
4
6 2 11
4
1
5 10
3 4
5 8
1
5 49
5 0
2
5 1
4
```

Sample Output 4

```
9
11
1
0
value remaining is 6
4
value remaining is 4
1
value remaining is 40
0
```

Hints

- Storing some extra values per dungeon may help you conquer instruction 3 (and instruction 5).
- The maximum possible value in instruction 4 may be related to the maximum possible value of some neighboring dungeons. For instance, for any dungeon with only one downstream dungeon, the answer is trivial, right? :-) Consider extending this idea to other dungeons.
- Fun fact: The TA who proposed this problem suffered from a 6-hour long painful debug session in this problem. Another TA validating this problem also suffered miserably for about 10 hours. *Bugs are everywhere and it is our destiny to keep debugging.* Good luck!