Problem 3 - Despair Sans Absolution (100 pts)

- Problem ID: 11
- Time Limit: 1s
- Memery Limit: **16384 KB**

> This problem has a **strict memory limit of 16384 KB**, which differs from the default limit of 1048576 KB. Ensure your solution adheres to this constraint to avoid exceeding the limit, which may result in a **Runtime Error**.

Problem Description

In this problem, we ask you to design a system to emulate Tien's todo list, currently containing thousands of jobs `:-(` which leaves zir in Despair sans Absolution (DSA). This system needs to support the following three operations:

- Add Operation: Insert a job into the todo list with some given priority.
- Complete Operation: Remove the job with the highest priority, which will then be attended by Tien.
- Drop Operation: Remove the job with the lowest priority, which will then be ignored and dropped after Tien drinks a few bottles of beer.

The max-heap data structure, as covered in class, efficiently supports Insert and RemoveMax operations but does not handle RemoveMin efficiently. Similarly, a min-heap supports Insert and RemoveMin but struggles with RemoveMax. While it is possible to construct both a max-heap and a min-heap with some cross-link mechanism to support all these operations efficiently, such an approach is complex to implement and requires additional space.

To design the system efficiently, we suggest you to learn related data structures that can handle both RemoveMax and RemoveMin operations efficiently. One possibility is the min-max heap. The following description of the min-max heap is modified from Wikipedia (https://en.wikipedia.org/wiki/Min-max_heap):

The min-max heap property is: each node at an even level in the tree is less than all of its descendants, while each node at an odd level in the tree is greater than all of its descendants.
The pseudo code on Wikipedia appears pretty clear, and hence we do not list them here. In this problem, we suggest you implement the min-max heap to conquer the tasks.

Input

Assuming that Tien's job queue is initially empty, the input begins with an integer N, representing the total number of operations. Each of the next N lines is given in one of the following formats:

- `1 job_id priority`: add operation
  Insert a job with `job_id` into the job queue, assigning it the priority `priority`.
- `2`: complete operation
  Complete the job with the highest priority, and remove it from the job queue.
- `3`: drop operation
  Drop the job with the lowest priority, and remove it from the job queue.

Output

- For each **add** operation, print a line that indicates the number of jobs in the job queue with the format:

  `[num_of_jobs] jobs waiting`

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

  `job [job_id] with [priority] completed`

  We will accept both `priority [priority]` and `[priority]` as correct output here.

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

  `no job in queue`

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

  `job [job_id] with [priority] dropped`

  We will accept both `priority [priority]` and `[priority]` as correct output here.

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

  `no job in queue`

Constraints

- $1 \leq N \leq 10^6$
- $1 \leq$ `job_id`, `priority` $\leq 10^9$
- It is guaranteed that all `priority` and `job_id` values will be distinct, respectively, at any given time.
- All operations will be valid, ensuring that all IDs remain within the specified range.

Subtasks

Subtask 1 (20 pts)

- only **add** and **complete** operations are implemented.

Subtask 2 (20 pts)

- $1 \leq N \leq 10^5$
- all operations are implemented
- $0 \leq$ **drop** operations $\leq 10^4$

Subtask 3 (60 pts)

- no other constraints

Sample Testcases

Sample Input 1

```
7
1 1 1126
1 2 314
1 3 520
2
2
2
2
```

Sample Output 1

```
1 jobs waiting
2 jobs waiting
3 jobs waiting
job 1 with priority 1126 completed
job 3 with priority 520 completed
job 2 with priority 314 completed
no job in queue
```

Sample Input 2

```
15
1 1 1126
2
3
1 2 10
1 3 11
1 4 8
1 5 12
1 6 1
1 7 13
3
1 8 3
1 9 2
1 10 7
3
2
```

Sample Output 2

```
1 jobs waiting
job 1 with priority 1126 completed
no job in queue
1 jobs waiting
2 jobs waiting
3 jobs waiting
4 jobs waiting
5 jobs waiting
6 jobs waiting
job 6 with priority 1 dropped
6 jobs waiting
7 jobs waiting
8 jobs waiting
job 9 with priority 2 dropped
job 7 with priority 13 completed
```

Problem 4 - Dock Spot Arboretum (100 pts)

- Problem ID: 12
- Time Limit: **1.5s**
- Memery Limit: 1048576 KB

> This problem has a **strict time limit of 1.5s**. Ensure your solution adheres to this constraint to avoid exceeding the limit, which may result in **Time Limit Exceeded**.

Problem Description

In this problem, we will provide you with a story version, and an equivalent formal version. Please feel free to decide what version(s) you want to read to solve the problem.

Story Version

NTU has a high number of bikes, but parking spaces are limited. As a result, students often squeeze others' bikes into tight spots, leading to overcrowding. This not only makes it difficult for students to locate their bikes when leaving but also increases the risk of theft. To better understand the issue, NTU's General Affairs Office has decided to analyze students' parking patterns.

To simplify the analysis, each parking slot is considered a node, and these slots together form a tree connected by trails. That is, there will be a unique path of trails from one slot to any other slot. We represent $w(x, y)$ as the travel time through the trail that connects slots x and y in the tree structure. Then, for any unique path from slot x to slot z, the total travel time is the sum of the times on the trails within the path.

There is a coordinate system $[1, c_x]$ in a parking slot x with a designed capacity c_x. Each slot is designed to accommodate one or more bikes. Students are allowed to park their bikes in any slot and move them between slots as needed. However, to maintain order and cleanliness, the university periodically clears and rearranges the parking slots, relocating some bikes to Shuiyuan. The students need to go to Shuiyuan on school buses to rescue their bikes after being notified.

There are $5 + 1$ types of operations:

1. Park s, x, p: Student s parks a bike at the parking slot x with intended position p. The parking policy is as follows.

   - If the parking slot has an empty space at p, the student parks zir bike at p.
   - Otherwise, if there is another vacancy (i.e. an integer location p' that is not occupied) in the same parking slot. Ze will select the nearest empty space in x. If there are two nearest vacancies, ze will choose the one with a smaller value.
   - Otherwise, p (and other integer positions) must be occupied. If the left-most bike is not parked at p, ze will insert zir bike in the middle of two bikes that are to the left of p and p. For example, if the parking slot with capacity 3 has `1 2 3` occupied, and one wants to park at position `3`, ze will park at `5/2`.
   - Otherwise, the left-most bike is parked at p. Then, ze will insert zir bike the middle of two bikes that are to the right of p and p. For example, if the parking slot with capacity 3 has `1 3/2 2 3` occupied, and one wants to park at position `1`, ze will park at `5/4`.

2. Move s, y, p: Student s moves zir bike from the parking slot that ze has parked (called x) to another parking slot y with intended position p. Please calculate the travel time from x to y. The parking policy is the same as the type-1 operation. If $x = y$, the bike **will not be moved at all** (i.e. the position will not be changed) and the travel time is 0.

3. Clear x, t: At time t, clear a parking slot x and relocate all bikes to Shuiyuan. Each student s of the relocated bikes will be notified at time $t + \ell_s$ that zir bike has been relocated.

4. Rearrange x, t: At time t, relocate all **rule-violating** bikes in slot x to Shuiyuan. The rule-violating bikes are those parked in non-integer positions of x. Each student s of the relocated bikes will be notified at time $t + \ell_s$ that zir bike has been relocated.

5. Fetch t: At time t, there is a school bus from the NTU to Shuiyuan. Students who have been notified to fetch their bikes, including those who have been notified at time t, will go to Shuiyuan by this bus together to rescue their bikes. After rescuing, the student will wander around on zir bike without parking them. That is, those bike will not be parked immediately—they will be parked with type-1 operation later.

6. Rebuild x, y, d (bonus): Sometimes, the Dwellers and Student Affairs want to adjust the trail so that $w(x, y)$ is changed to $d > 0$. It is guaranteed that there is a trail between x and y within the original tree-structured slots.

Formal Version

Given a weighted tree $T(V, E), |V| = n, |E| = n - 1$, where $V = \{0, 1, \ldots, n - 1\}$ represents its nodes and E represents its edges, with some weight $w(x, y) > 0$ on each $(x, y) \in E$. For each node $x \in V$, consider some integer $c_x > 0$ as the capacity of slot x. Consider m users $\{0, 1, \ldots, m - 1\}$, where each user s comes with a parameter $\ell_s > 0$. Also, consider an empty set S. There are $5 + 1$ types of operations:

1. Park s, x, p: Insert s in node x at p with the following policy.
   - If no $(*, p)$ is in x, insert (s, p) to x.
   - Otherwise, if there is an integer $p' \in [1, c_x]$ such that no $(*, p')$ is x, insert (s, p') with the smallest $|p' - p|$ to x. Ties are broken by inserting the (s, p') with a smaller p'.
   - Otherwise, if $p \neq \min_{(*,q)\in x} q$, let $p' = \max_{(*,q)\in x, q<p} q$. Insert $(s, \frac{p+p'}{2})$ to x.
   - Otherwise, $p = \min_{(*,q)\in x} q$, let $p' = \min_{(*,q)\in x, q>p} q$. Insert $(s, \frac{p+p'}{2})$ to x.
2. Move s, y, p: Find (s, q) from some $(s, *) \in x$ for a unique node x.
   - If $x \neq y$, remove (s, q) from x. Then, calculate $\sum_{(a,b) \text{ on path between } (x,y)} w(a, b)$ and execute similar to the type-1 operation of Park (s, y, p).
   - If $x = y$, output 0 and do nothing else.
3. Clear x, t: At time t, clear every item $(s, *) \in x$ and insert $(s, t + \ell_s)$ to S.
4. Rearrange x, t: At time t, clear every item $(s, p) \in x$ where p is not an integer and insert $(s, t + \ell_s)$ to S.
5. Fetch t: At time t, remove every $(s, t') \in S$ with $t' \leq t$.
6. Rebuild x, y, d (bonus): Change the $w(x, y)$ for some $(x, y) \in E$ to d.

Illustration

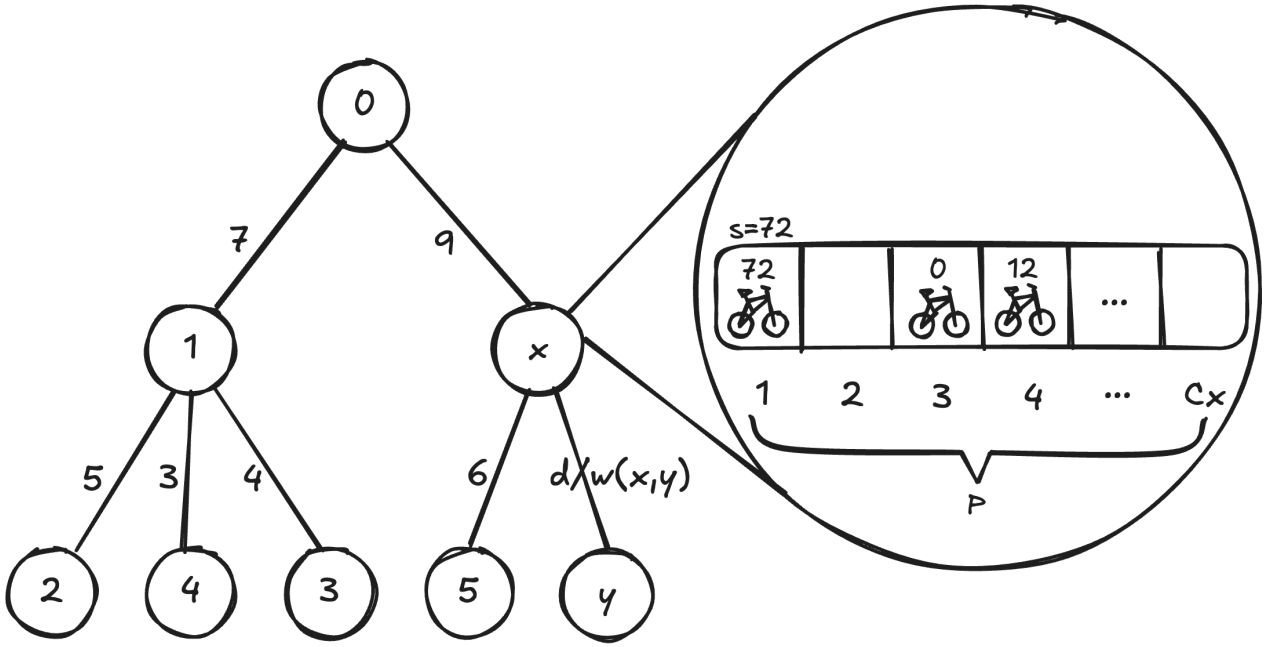

Figure 2.1: Sample Parking Tree

Input

- The first line contains three integers n, m, and q representing the number of parking slots (tree nodes), the number of students (users), and the number of operations, respectively.
- The second line contains n integers representing the capacity for each parking slot (tree nodes) c_x for $x \in \{0, 1, \ldots, n-1\}$.
- The third line contains m integers representing the fetch time ℓ_s for each student (user) s for $s \in \{0, 1, \ldots, m-1\}$.
- The next $n-1$ lines contain three integers x, y, and w representing the parking slots (tree nodes) x and y are connected by a trail (edge) with travel time (weight) w.
- The next q lines contain the operations. For each operation:

    1. Park: `0 s x p`
    2. Move: `1 s y p`
    3. Clear: `2 x t`
    4. Rearrange: `3 x t`
    5. Fetch: `4 t`
    6. Rebuild: `5 x y d`

~~Both ℓ_u and t are measured by seconds.~~ Both ℓ_u and t are measured by a time scale that is much larger than seconds. Because the scale is so different, the seconds taken to move a bike from x to y (see output format below) can be ignored. That is, you can consider the timeline of Clear, Rearrange, and Fetch separately from the timeline of Move.

Output

For each operation, output the following.

- Park:

  `[s] parked at ([x], [fp]).`

  where fp is the final location of the bike (the second part of the pair being inserted to x). Display fp by an irreducible fraction of `p/q`. If $q = 1$, you should output only `p`.

- Move:

  `[s] moved to [y] in [t] seconds.`

  You only need to output `seconds` in all cases, without changing to a singular form.

- Clear: output nothing

- Rearrange:

  `Rearranged [n] bikes in [x].`

  We will accept both `bikes` and `bicycles` as correct output here.

- Fetch:

  `At [t], [n] bikes was fetched.`

  We will accept both `bikes` and `bicycles` as correct output here.

- Rebuild: output nothing

Constraints

- $1 \leq n, m \leq 3 \times 10^5$, $1 \leq q \leq 10^5$
- $0 \leq s < m$
- $0 \leq x, y < n$
- $0 \leq w(x, y) \leq 10^5$, $0 \leq d \leq 10^5$
- $0 \leq t \leq 10^{15}$
- $1 \leq p \leq c_x$ with $2 \leq c_x \leq 15$ for all $x \in \{0, 1, \ldots, n-1\}$
- $0 \leq \ell_s \leq 10^6$ for all $s \in \{0, 1, \ldots, m-1\}$
- At any time, the number of bikes/elements in a single (parking slot/node) x will not exceed $2 \times c_x$.
- The time t in all operations are given in a strictly ascending order.
- At any time, there is at most one $(s, *)$ in $\{$ all nodes $\} \cup S$. That is, each student s owns a unique bike.
- For all (s, p') in any node x, where p' is the actual parking position of the bike of student s in x, it is guaranteed that $p' = \alpha/\beta$ can be represented by two positive integers α and β that are relatively co-prime, where $1 \leq \alpha \leq 2^{64} - 1$ and $1 \leq \beta \leq 2^{64} - 1$.

For each operation:

1. Park: It is guaranteed that the bike of student s is not in any of the parking slots.
2. Move: It is guaranteed that the bike of student s is in a parking slot.
3. Clear: It is guaranteed that the parking slot x is not empty.
4. Rearrange: It is guaranteed that the parking slot x is not empty.
5. Fetch: It is guaranteed that the fetch time is in a strictly ascending order.
6. Rebuild: It is guaranteed that the parking slot x and y are connected by an original trail.

Subtasks

Subtask 1 (10 pts)

- $1 \leq n, m \leq 3 \times 10^2$
- $1 \leq q \leq 3 \times 10^2$
- Only operations 1 and 2 are present.

Subtask 2 (20 pts)

- $1 \leq n, m \leq 3 \times 10^2$
- $1 \leq q \leq 10^3$
- Only operations 1, 2, 3, 4, and 5 are present.

Subtask 3 (20 pts)

- $1 \leq n \leq 3 \times 10^2$
- Only operations 1, 2, 3, 4, and 5 are present.

Subtask 4 (50 pts)

- Only operations 1, 2, 3, 4, and 5 are present.

Subtask 5 (Bonus: Free Beverage)

- All operations are present.

Sample Testcases

Sample Input 1

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

Sample Output 1

```
0 parked at (0, 2).
1 parked at (0, 1).
2 parked at (0, 3).
```

Sample Input 2

```
1 5 6
3
0 0 0 0 0
0 0 0 1
0 1 0 2
0 2 0 3
0 3 0 3
3 0 1
0 4 0 3
```

Sample Output 2

```
0 parked at (0, 1).
1 parked at (0, 2).
2 parked at (0, 3).
3 parked at (0, 5/2).
Rearranged 1 bikes in 0.
4 parked at (0, 5/2).
```

Sample Input 3

```
1 5 6
3
3 4 5 6 7
0 0 0 1
0 1 0 1
0 2 0 1
0 3 0 1
2 0 1
4 6
```

Sample Output 3

```
0 parked at (0, 1).
1 parked at (0, 2).
2 parked at (0, 3).
3 parked at (0, 3/2).
At 6, 3 bikes was fetched.
```

Sample Input 4

```
6 5 6
3 3 3 4 4 4
3 4 5 6 7
0 2 1
0 5 4
1 5 7
2 3 2
4 5 3
0 0 0 1
1 0 1 1
1 0 4 1
1 0 2 1
1 0 3 1
1 0 2 1
```

Sample Output 4

```
0 parked at (0, 1).
0 moved to 1 in 11 seconds.
0 moved to 4 in 10 seconds.
0 moved to 2 in 8 seconds.
0 moved to 3 in 2 seconds.
0 moved to 2 in 2 seconds.
```

Sample Input 5

```
5 10 10
5 10 6 11 2
0 0 0 0 0 0 0 0 0 0
3 0 49410
3 2 54898
2 1 76874
4 1 14829
0 6 4 1
5 3 0 315398
0 0 4 1
0 3 4 2
0 2 4 1
2 4 18337236
0 7 2 4
1 7 2 5
0 4 0 3
2 2 37134602
```

Sample Output 5

```
6 parked at (4, 1).
0 parked at (4, 2).
3 parked at (4, 3/2).
2 parked at (4, 5/4).
7 parked at (2, 4).
7 moved to 2 in 0 seconds.
4 parked at (0, 3).
```