Homework 3

[2024-2] 데이터사이언스 응용을 위한 컴퓨팅 (001)

Due: **2024년 11월 24일 23:59**

1. Max-heap with tree-based structure [25pts]

Implement a binary max-heap with tree-based structure which has the following properties:

- Structure property: It is a complete binary tree.
- Heap property: The value of every non-root node is equal to or smaller than the value of its parent.

The binary max-heap should use linked `Node` instances with pointers rather than an array-based indexing approach.

You need to implement the `enqueue` and `dequeue` method in `MaxHeap` class provided in `functions.cpp`.

- The `enqueue` method takes an integer `k` as input and returns `void`. It creates a new node with the value `k` and appends this node to the heap as the last element. After adding the new node, the method restores the heap property.
- The `dequeue` method takes no input and returns `void`. It removes the value at the root node, which is the **maximum** value in the heap. The root node is then replaced with the value of the last node, and the last node is removed from the heap. Finally, the method restores the heap property.
- Ensure that each operation in the `enqueue` and `dequeue` methods is completed in O(log *n*) time.

Several helper methods are provided, each described as follows:

- `getMax()`: Returns a pointer to the root node.
- `printHeap()`: Prints the values of all nodes in the heap from the root to the leaves, traversing each level from left to right.
- `swap(Node* a, Node* b)`: Swaps the values of the two input nodes, `a` and `b`.

Example

```
Maxheap h;
h.enqueue(10);
h.enqueue(30);
h.enqueue(40);
h.enqueue(500);
h.enqueue(80);
h.dequeue();
h.getMax()->val;
>> 80
```

2. Minimum distance [25pts]

Given an array of points representing integer coordinates on a 2D plane, where each point is represented as (x_i, y_i), your goal is to find the minimum distance required to connect all points. The distance between any two points (x_i, y_i) and (x_j, y_j) is calculated using the Manhattan distance: $|x_i - x_j| + |y_i - y_j|$.
Implement the `minDistance` function in `functions.cpp` following the instructions below.

2.1 Input

- `vector<vector<int>>& points`: A reference to a vector of integer pairs, where each pair represents the (x, y) coordinates of a point on a 2D plane.

2.2 Output

- The `minDistance` returns an integer representing the minimum distance required to connect all points.

2.3 Constraints

- $1 \leq$ `points.size()` ≤ 1000
- $-10^6 \leq x_i, y_i \leq 10^6$
- All points (x_i, y_i) are unique.

2.4 Example

```
minDistance (vector({vector({3,12}), vector({-2,5}), vector({-4,1})}))
>> 18
```

3. Flight routes [25pts]

You are planning to travel from San Francisco to New York by plane.
Your goal is to determine the following details:

1. **Minimum Price**
   a. Find the minimum price for a route from San Francisco to New York.What is the minimum price for this route?
2. **Distinct Minimum-Price Routes Count**
   a. Calculate the number of distinct routes that achieve this minimum price. Return the count modulo INT_MAX (2,147,483,647).
3. **Minimum Flights for Minimum Price**
   a. Among all minimum-price routes, find the minimum number of flights required in any route that achieves the minimum price.
4. **Maximum Flights for Minimum Price**
   a. Find the maximum number of flights used in any route that achieves the minimum price.

Implement the `flightRoute` function in `functions.cpp` according to the following specifications.

3.1 Input

The function takes three input arguments:

1. `const int n`: The number of cities.
   - The cities are numbered from 1 to n, where city 1 is San Francisco and city n is New York.
2. `const int m`: The number of flights.
3. `const vector<vector<int>>& route_info`: A vector containing `m` elements, where each element represents information about a flight route. Each element is a vector of three integers: `a`, `b`, and `c`, where:
   - `a` and `b` represent the starting and destination cities, respectively.
   - `c` is the cost of the flight from city `a` to city `b`.

Each flight is one-way. You can assume there exists at least one route from San Francisco ($city_1$) to New York ($city_n$).

3.2 Output

The function should print four space-separated integers corresponding to the following values:

1. The minimum price for a route from San Francisco to New York
2. Distinct Minimum-Price Routes Count (return modulo INT_MAX, which is 2147483647)
3. Minimum Flights for Minimum Price
4. Maximum Flights for Minimum Price

Ensure the output does not include any newline characters or std::endl at the end. The function should print the result but does not return any value.

3.3 Constraints

$1 \leq \mathtt{n} \leq 10^5$

$1 \leq \mathtt{m} \leq 2 \times 10^5$

$1 \leq \mathtt{a}, \mathtt{b} \leq \mathrm{n}$

$1 \leq \mathtt{c} \leq 10^9$

3.4 Example

```
flightRoute(4, 5, vector({{1,4,5}, {1,2,4}, {2,4,5}, {1,3,2}, {3,4,3}}))
>> 5 2 1 2
```

4. Coin combinations[25pts]

Consider a money system consisting of n types of coins, each with a unique positive integer value. Write a function `coinCombinations` that calculates the number of distinct ways to achieve a target sum. You may use the same coin multiple times in a combination, and the order of coins matters if they are of different types. For example, if the coins are {1, 2, 3} and the target sum is 4, there are 7 possible combinations:

- 1+1+1+1
- 1+1+2
- 1+2+1
- 2+1+1
- 2+2
- 1+3
- 3+1

Implement the `coinCombinations` function in `functions.cpp` as specified below.

4.1 Input

The function takes three input arguments:

1. `int n`: The number of distinct coins.
2. `int sum`: The desired target sum.
3. `vector<int>& coins`: A vector containing the values of the `n` distinct coins(c_1, c_2, . . . c_n), where each value is a positive integer.

4.2 Output

The function returns an integer representing the number of distinct ways to reach the sum using the available coins. Return this count modulo `INT_MAX` (2,147,483,647) to handle large values.

4.3 Constraints

$1 \leq$ `n` ≤ 100

$1 \leq$ `sum` $\leq 10^6$

$1 \leq c_i \leq 10^6$ (where c_i is the value of each coin)

4.4 Example

```
coinCombinations (3, 4, vector({1, 2, 3}))
>> 7
```

Submission Requirements

- For each problem, write your functions in the designated `functions.cpp` file, then submit this file to Gradescope for grading. You may modify other files for testing purposes, but keep in mind that only `functions.cpp` will be graded, using the original header files and `main.cpp`.

- Grading will be conducted using the C++11 standard. The `main` function in the autograder, which includes test cases, may differ from the `main` function released to students. Compilation will follow this process:
  - `$ g++ -c main.cpp -o main.o -std=c++11`
  - `$ g++ -c functions.cpp -o functions.o -std=c++11` (using your functions.cpp)
  - `$ g++ main.o functions.cpp -o main`
- Before submitting, remove all debugging code, including any `stdout` print statements, from `functions.cpp`, except those explicitly required by the problem. Grading will rely solely on `stdout`, and any unrelated output may impact your score.
- Do not share your code with other students. Similarly, students must not use AI tools to generate code directly. High similarity levels in code, as detected by our similarity detection tools, may lead to serious penalties.
- You have 5 grace days for homework submissions throughout the semester, counted in 24-hour increments from the original due time. For instance, if an assignment is due on 11/24 at 11:59 PM, submitting it 30 minutes late will use one grace day. Submitting on 11/26 at 9:00 PM will use two grace days. Late submissions are only accepted if grace days are available.