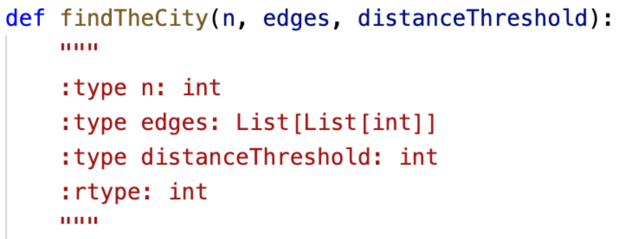
Programming

Part1. Python test

(leetcode 1334. Find the City With the Smallest Number of Neighbors at a Threshold Distance)

Implement the function '**findTheCity(n, edges, distanceThreshold)**'.



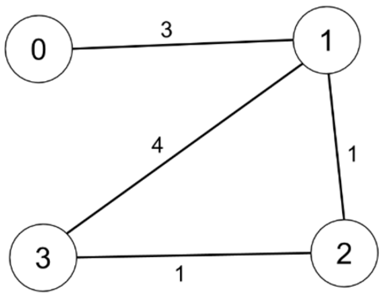
```
def findTheCity(n, edges, distanceThreshold):
    """
    :type n: int
    :type edges: List[List[int]]
    :type distanceThreshold: int
    :rtype: int
    """
```

There are **n** cities numbered from 0 to n-1. Given the array **edges** where edges[i] = [fromi, toi, weighti] represents a bidirectional and weighted edge between cities fromi and toi, and given the integer **distanceThreshold**.

The function 'findTheCity' should return the city with the smallest number of cities that are reachable through some path and whose distance is at most **distanceThreshold**. If there are multiple such cities, return the city with the greatest number.

Notice that the distance of a path connecting cities **I** and **j** is equal to the sum of the edges' weights along that path.

Example :



Input: n = 4, edges = [[0,1,3],[1,2,1],[1,3,4],[2,3,1]], distanceThreshold = 4

Output: 3

Explanation: The figure above describes the graph.

The neighboring cities at a distanceThreshold = 4 for each city are:

City 0 -> [City 1, City 2]

City 1 -> [City 0, City 2, City 3]

City 2 -> [City 0, City 1, City 3]

City 3 -> [City 1, City 2]

Cities 0 and 3 have 2 neighboring cities at a distanceThreshold = 4, but we have to return city 3 since it has the greatest number.

Part2. C test

(leetcode14. Longest Common Prefix)

Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string "".

```
char * longestCommonPrefix(char ** strs, int strsSize){

}
```

Example 1:

Input: strs = ["flower","flow","flight"], strsSize = 3

Output: "fl"

Example 2:

Input: strs = ["dog","racecar","car"], strsSize = 3

Output: ""

Explanation: There is no common prefix among the input strings.

Part2. C++ test

Implement class MyQueue and class MyDeque using class BaseArray. Class BaseArray is given as follows. The rearrangement of elements within an array is not considered.

```
class BaseArray {
    private:
        int capacity;
        int* mem;

    protected:
        BaseArray(int capacity = 100) {     // constructor
            this->capacity = capacity;
            mem = new int[capacity];
        }
        ~BaseArray() { delete[] mem; }     // destructor
        void put(int index, int val) { mem[index] = val; }
        int get(int index) { return mem[index]; }
        int getCapacity() { return capacity; }
};
```

Int capacity : size of array

Int* mem : Pointer to the starting point of the array

Put(index, val) : Inserts a value val at the index position of the array.

Get(index) : Gets the value at the index position of the array.

Getcapacity() : return size of the array

1) Implement a class MyQueue that inherits the BaseArray class and works as a queue.

```
class MyQueue : public BaseArray {
protected:
    int start;   // queue 시작지점 index
    int end;     // queue 의 끝지점 index

public:
    MyQueue(int capacity) : BaseArray(capacity) { start = 0; end = 0; } // constructor
    int length();            // queue 의 길이 return.
    void enqueue(int n);     // queue 에 n 을 넣는다. 불가능한 경우 'cannot push' print
    int dequeue();           // dequeue 를 시행한다. 불가능한 경우 -1 return.
};
```

2) Implement a MyDeque class that inherits the MyQueue class and works as a deque.

```
class MyDeque : protected MyQueue {
public:
    MyDeque(int capacity) : MyQueue(capacity) { int start = 0; int end = 0; } // constructor
    int length();            // 길이 return.
    void push_back(int n);   // deque 의 뒤에 원소를 push. 불가능한 경우 'cannot push' print
    int pop_front();         // deque 의 앞의 원소를 pop. 불가능한 경우 -1 return.
    void push_front(int n);  // deque 의 앞에 원소를 push. 불가능한 경우 'cannot push' print
    int pop_back();          // deque 의 뒤의 원소를 pop. 불가능한 경우 -1 return.
};
```