Status	Finished
Started	Sunday, 20 October 2024, 1:33 PM
Completed	Monday, 21 October 2024, 8:01 PM
Duration	1 day 6 hours
Marks	7.00/7.00
Grade	**10.00** out of 10.00 (**100**%)

Question 1

Correct

Mark 1.00 out of 1.00

Implement all methods in class **Queue** with template type **T**. The description of each method is written as comment in frame code.

```
#ifndef QUEUE_H
#define QUEUE_H
#include "DLinkedList.h"
template<class T>
class Queue {
protected:
    DLinkedList<T> list;
public:
    Queue() {}
    void push(T item) ;
    T pop() ;
    T top() ;
    bool empty() ;
    int size() ;
    void clear() ;
};

#endif /* QUEUE_H */
```

You can use all methods in class **DLinkedList** without implementing them again. The description of class **DLinkedList** is written as comment in frame code.

```
template <class T>
class DLinkedList
{
public:
    class Node;     //forward declaration
protected:
    Node* head;
    Node* tail;
    int count;
public:
    DLinkedList() ;
    ~DLinkedList();
    void add(const T& e);
    void add(int index, const T& e);
    T removeAt(int index);
    bool removeItem(const T& removeItem);
    bool empty();
    int size();
    void clear();
    T get(int index);
    void set(int index, const T& e);
    int indexOf(const T& item);
    bool contains(const T& item);
};
```

For example:

Test	Result

`Queue<int> queue;`
`assert(queue.empty());`
`assert(queue.size() == 0);`	

Answer: (penalty regime: 0 %)

Reset answer

```
void push(T item) {
    // TODO: Push new element into the end of the queue
    list.add(item);
}

T pop() {
    // TODO: Remove an element in the head of the queue
    T ret=list.get(0);
    list.removeAt(0);
    return ret;
}

T top() {
    // TODO: Get value of the element in the head of the queue
    T ret=list.get(0);
    return ret;
}

bool empty() {
    // TODO: Determine if the queue is empty
    return list.empty();

}

int size() {
    // TODO: Get the size of the queue
    return list.size();
}

void clear() {
    // TODO: Clear all elements of the queue
    list.clear();
}
```

	Test

✓	`Queue<int> queue;`
	`    assert(queue.empty());`
	`    assert(queue.size() == 0);`

✓	`Queue<int> queue;`
	
	`    int item[] = { 3, 1, 4, 5, 2, 8, 10, 12 };`
	`    //index:  0  1  2  3  4  5  6   7`
	`    for (int idx = 0; idx < 8; idx++) queue.push(item[idx]);`
	`    assert(queue.empty() == false);`
	`    assert(queue.size() == 8);`

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **2**

Correct

Mark 1.00 out of 1.00

Given an array of integers.
Your task is to implement a function with following prototype:

```
int sumOfMaxSubarray(vector<int>& nums, int k);
```

The function returns the sum of the maximum value of every consecutive subarray of `nums` with fixed length `k`.

Note:

- The `iostream`, `vector`, `queue` and `deque` libraries have been included and `namespace std` is being used. No other libraries are allowed.
- You can write helper functions and classes.

For example:

Test	Result

`vector<int> nums {1, 2, 4, 3, 6};`
`int k = 3;`
`cout << sumOfMaxSubarray(nums, k);`
	14

Answer: (penalty regime: 0 %)

Reset answer

```
int sumOfMaxSubarray(vector<int>& nums, int k) {
    deque<int> d; // deque to store INDICES of Maximums (usually in the front of deque)

    // deque elements would be arranged:
    // Max --- Min

    int sum = 0;

    // Push in maximum of the first subarray
    int i = 0;
    while (i < k)
    {
        while (!d.empty() && nums.at(i) >= nums.at(d.back()))
            d.pop_back();

        d.push_back(i);
        ++i;
    }
    sum += nums.at(d.front()); // Front is always a maximum value of the current subarray

    // Continue with other elements
    // nums[i] is new element added
    while (i < (int)nums.size())
    {
        // Pop old elements
        if (!d.empty() && d.front() < i - k + 1)
            d.pop_front();

        while (!d.empty() && nums.at(i) >= nums.at(d.back()))
            d.pop_back();

        d.push_back(i);
        ++i;

        sum += nums.at(d.front());
    }

    return sum;
```

```
39 }
```

	Test	Expected	Got	

✓	`vector<int> nums {1, 2, 4, 3, 6};`
	`int k = 3;`
	`cout << sumOfMaxSubarray(nums, k);`
		14	14	✓

✓	`vector<int> nums {8016};`
	`int k = 1;`
	`cout << sumOfMaxSubarray(nums, k);`
		8016	8016	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **3**

Correct

Mark 1.00 out of 1.00

Research **queue** which is implemented in C library at http://www.cplusplus.com/reference/queue/queue/. You can use library **queue** in c++ for this question.

Using **queue**, complete function **bool isBipartite(vector<vector<int>> graph)** to determine if a graph is bipartite or not (the graph can be disconnected). In caat https://en.wikipedia.org/wiki/Bipartite_graph.

You can use below liberaries in this question.

```
#include <iostream>
#include <vector>
#include <queue>
```

For example:

Test	Result

`int G[6][6] = { {0, 1, 0, 0, 0, 1},`
`{1, 0, 1, 0, 0, 0},`
`{0, 1, 0, 1, 0, 0},`
`{0, 0, 1, 0, 1, 0},`
`{0, 0, 0, 1, 0, 1},`
`{1, 0, 0, 0, 1, 0} };`
`int n = 6;`
 	
`vector<vector<int>> graph(n, vector<int>());`
`for (int i = 0; i < n; ++i) {`
`for (int j = 0; j < n; ++j) {`
`if (G[i][j]) graph[i].push_back(j);`
`}`
`}`
 	
`isBipartite(graph) ? cout << "Yes" : cout << "No";`
	Yes

Answer: (penalty regime: 0 %)

Reset answer

```
bool isBipartite(vector<vector<int>> graph) {
    int n = graph.size();
    vector<int> colors(n, -1); // Initialize colors of all nodes to -1 (unvisited)

    // Perform BFS on each connected component of the graph
    for (int i = 0; i < n; ++i) {
        if (colors[i] == -1) { // If the node is unvisited
            queue<int> q;
            q.push(i); // Start BFS from this node
            colors[i] = 0; // Color the node with color 0

            while (!q.empty()) {
                int node = q.front();
                q.pop();

                // Assign alternate color to its neighbors
                for (int neighbor : graph[node]) {
                    if (colors[neighbor] == -1) { // If neighbor is unvisited
                        colors[neighbor] = 1 - colors[node]; // Assign alternate color
                        q.push(neighbor);
                    } else if (colors[neighbor] == colors[node]) {
                        // If neighbor has same color as current node, graph is not bipartite
                        return false;
                    }
```

```
                }
            }
        }
    }

    // If all connected components are bipartite, the graph is bipartite
    return true;
}
```

	Test	Expected	Got	

✓	int G[6][6] = { {0, 1, 0, 0, 0, 1},
	{1, 0, 1, 0, 0, 0},
	{0, 1, 0, 1, 0, 0},
	{0, 0, 1, 0, 1, 0},
	{0, 0, 0, 1, 0, 1},
	{1, 0, 0, 0, 1, 0} };
	int n = 6;
	
	vector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (G[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	isBipartite(graph) ? cout << "Yes" : cout << "No";
		Yes	Yes	✓

✓	int G[6][6] = { {0, 1, 1, 1, 0, 1},
	{1, 0, 1, 0, 0, 0},
	{1, 1, 0, 1, 0, 0},
	{1, 0, 1, 0, 1, 0},
	{0, 0, 0, 1, 0, 1},
	{1, 0, 0, 0, 1, 0} };
	
	int n = 6;
	
	vector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (G[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	isBipartite(graph) ? cout << "Yes" : cout << "No";
		No	No	✓

	Test	Expected	Got	

✓	int G[6][6] = { {0, 1, 1, 0, 0, 1},
	{1, 0, 1, 0, 0, 0},
	{1, 1, 0, 1, 0, 0},
	{0, 0, 1, 0, 1, 0},
	{0, 0, 0, 1, 0, 1},
	{1, 0, 0, 0, 1, 0} };
	
	int n = 6;
	
	vector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (G[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	isBipartite(graph) ? cout << "Yes" : cout << "No";
		No	No	✓

✓	int G[6][6] = { {0, 0, 0, 0, 0, 0},
	{0, 0, 1, 0, 0, 0},
	{0, 1, 0, 1, 0, 0},
	{0, 0, 1, 0, 1, 0},
	{0, 0, 0, 1, 0, 1},
	{0, 0, 0, 0, 1, 0} };
	
	int n = 6;
	
	vector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (G[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	isBipartite(graph) ? cout << "Yes" : cout << "No";
		Yes	Yes	✓

✓	int G[6][6] = { {0, 1, 0, 1, 0, 0},
	{1, 0, 1, 0, 0, 0},
	{0, 1, 0, 1, 0, 0},
	{1, 0, 1, 0, 0, 0},
	{0, 0, 0, 0, 0, 0},
	{0, 0, 0, 0, 0, 0} };
	
	int n = 6;
	
	vector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (G[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	isBipartite(graph) ? cout << "Yes" : cout << "No";
		Yes	Yes	✓

	Test	Expected	Got	

✓	int G[6][6] = { {0, 1, 0, 1, 0, 0},
	{1, 0, 1, 0, 0, 0},
	{0, 1, 0, 1, 0, 0},
	{1, 0, 1, 0, 0, 0},
	{0, 0, 0, 0, 0, 1},
	{0, 0, 0, 0, 1, 0} };
	
	int n = 6;
	
	vector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (G[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	isBipartite(graph) ? cout << "Yes" : cout << "No";
		Yes	Yes	✓

✓	int G[6][6] = { {0, 1, 0, 1, 1, 0},
	{1, 0, 1, 0, 0, 0},
	{0, 1, 0, 1, 0, 0},
	{1, 0, 1, 0, 0, 0},
	{1, 0, 0, 0, 0, 0},
	{0, 0, 0, 0, 0, 0}};
	
	int n = 6;
	
	vector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (G[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	isBipartite(graph) ? cout << "Yes" : cout << "No";
		Yes	Yes	✓

✓	int G[6][6] = { {0, 1, 0, 1, 1, 1},
	{1, 0, 1, 0, 0, 0},
	{0, 1, 0, 1, 0, 0},
	{1, 0, 1, 0, 0, 0},
	{1, 0, 0, 0, 1, 0},
	{1, 0, 0, 0, 0, 1}};
	
	int n = 6;
	
	vector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (G[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	isBipartite(graph) ? cout << "Yes" : cout << "No";
		No	No	✓

	Test	Expected	Got	

✓	int G[6][6] = { {0, 1, 0, 1, 1, 0},
	{1, 0, 1, 0, 0, 0},
	{0, 1, 0, 1, 0, 0},
	{1, 0, 1, 0, 0, 0},
	{1, 0, 0, 0, 1, 0},
	{0, 0, 0, 0, 0, 1}};
	
	int n = 6;
	
	vector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (G[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	isBipartite(graph) ? cout << "Yes" : cout << "No";
		No	No	✓

✓	int G[6][6] = { {0, 1, 0, 0, 0, 0},
	{1, 0, 0, 0, 0, 0},
	{0, 0, 0, 0, 0, 0},
	{0, 0, 0, 0, 1, 1},
	{0, 0, 0, 1, 0, 1},
	{0, 0, 0, 1, 1, 0} };
	int n = 6;
	
	vector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (G[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	isBipartite(graph) ? cout << "Yes" : cout << "No";
		No	No	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **4**

Correct

Mark 1.00 out of 1.00

Research **queue** which is implemented in C library at: http://www.cplusplus.com/reference/queue/queue/. You can use library **queue** in c++ for this question.

Using **queue**, complete function **void bfs(vector<vector<int>> graph, int start)** to traverse all the nodes of the graph from given start node using Breadth First Search algorithm and data structure **queue**, and print the order of visited nodes.

You can use below liberaries in this question.

```
#include <iostream>
#include <vector>
#include <queue>
```

For example:

Test	Result

`int init_graph[10][10] = { {0, 1, 1, 0, 1, 0, 1, 0, 1, 0},`
`{0, 0, 1, 1, 0, 0, 0, 1, 0, 0},`
`{0, 1, 0, 0, 0, 1, 1, 0, 1, 1},`
`{1, 0, 0, 0, 0, 0, 0, 1, 0, 0},`
`{0, 1, 0, 0, 0, 0, 0, 1, 0, 0},`
`{1, 0, 1, 0, 1, 0, 0, 0, 1, 0},`
`{0, 0, 1, 1, 0, 1, 0, 0, 0, 0},`
`{1, 0, 0, 0, 0, 1, 1, 0, 1, 0},`
`{0, 0, 0, 0, 0, 1, 0, 1, 0, 1},`
`{1, 0, 1, 0, 1, 0, 0, 0, 1, 0} };`
`int n = 10;`
`vector<vector<int>> graph(n, vector<int>());`
`for (int i = 0; i < n; ++i) {`
`for (int j = 0; j < n; ++j) {`
`if (init_graph[i][j]) graph[i].push_back(j);`
`}`
`}`
 	
`bfs(graph, 0);`
	`0 1 2 4 6 8 3 7 5 9`

Answer: (penalty regime: 0 %)

Reset answer

```
void bfs(vector<vector<int>> graph, int start) {
        int n = graph.size();
    vector<bool> visited(n, false);
    queue<int> q;

    // Enqueue the start node and mark it as visited
    q.push(start);
    visited[start] = true;

    // Continue traversal until queue is empty
    while (!q.empty()) {
        // Dequeue a node from the queue
        int current = q.front();
        q.pop();
        cout << current << " ";

        // Enqueue all adjacent nodes of the dequeued node
        for (int neighbor : graph[current]) {
            if (!visited[neighbor]) {
                q.push(neighbor);
                visited[neighbor] = true;
            }
```

```
        }
    }
}
```

	Test	Expected	Got	

✓	\tint init_graph[10][10] = { {0, , 1, 1, 0, 1, 0, 1, 0, 1, 0},
	\t\t\t\t {0, 0, 1, 1, 0, 0, 0, 1, 0, 0},
	\t\t\t\t {0, 1, 0, 0, 0, 1, 1, 0, 1, 1},
	\t\t\t\t {1, 0, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t {0, 1, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t {1, 0, 1, 0, 1, 0, 0, 0, 1, 0},
	\t\t\t\t {0, 0, 1, 1, 0, 1, 0, 0, 0, 0},
	\t\t\t\t {1, 0, 0, 0, 0, 1, 1, 0, 1, 0},
	\t\t\t\t {0, 0, 0, 0, 0, 1, 0, 1, 0, 1},
	\t\t\t\t {1, 0, 1, 0, 1, 0, 0, 0, 1, 0} };
	\tint n = 10;
	\tvector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (init_graph[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	\tbfs(graph, 0);
		0 1 2 4 6 8 3 7 5 9	0 1 2 4 6 8 3 7 5 9	✓

✓	\tint init_graph[10][10] = { {0, 1, 1, 0, 1, 0, 1, 0, 1, 0},
	\t\t\t {0, 0, 1, 1, 0, 0, 0, 1, 0, 0},
	\t\t\t\t {0, 1, 0, 0, 0, 1, 1, 0, 1, 1},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 1, 0, 0, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 1, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 0, 0, 0, 1, 0, 1, 0, 1},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0} };
	\tint n = 10;
	\tvector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (init_graph[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	\tbfs(graph, 1);
		1 2 3 7 5 6 8 9 0 4	1 2 3 7 5 6 8 9 0 4	✓

	Test	Expected	Got	

✓	\tint init_graph[10][10] = { {0, 1, 1, 0, 1, 0, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 1, 1, 0, 1, 1},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 1, 0, 0, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 1, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 0, 0, 0, 1, 0, 1, 0, 1},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0} };
	\tint n = 10;
	\tvector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (init_graph[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	\tbfs(graph, 2);
		2 1 5 6 8 9 3 7 0 4	2 1 5 6 8 9 3 7 0 4	✓

✓	\tint init_graph[10][10] = { {0, 1, 1, 0, 1, 0, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 1, 1, 0, 1, 1},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 1, 0, 0, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 1, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 0, 0, 0, 1, 0, 1, 0, 1},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0} };
	\tint n = 10;
	\tvector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (init_graph[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	\tbfs(graph, 3);
		3 0 7 1 2 4 6 8 5 9	3 0 7 1 2 4 6 8 5 9	✓

✓	\tint init_graph[10][10] = { {0, 1, 1, 0, 1, 0, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 1, 1, 0, 1, 1},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 1, 0, 0, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 1, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 0, 0, 0, 1, 0, 1, 0, 1},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0} };
	\tint n = 10;
	\tvector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (init_graph[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	\tbfs(graph, 4);
		4 1 7 2 3 0 5 6 8 9	4 1 7 2 3 0 5 6 8 9	✓

	Test	Expected	Got	

✓	\tint init_graph[10][10] = { {0, 1, 1, 0, 1, 0, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 1, 1, 0, 1, 1},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 1, 0, 0, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 1, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 0, 0, 0, 1, 0, 1, 0, 1},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0} };
	\tint n = 10;
	\tvector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (init_graph[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	\tbfs(graph, 5);
		5 0 2 4 8 1 6 9 7 3	5 0 2 4 8 1 6 9 7 3	✓

✓	\tint init_graph[10][10] = { {0, 1, 1, 0, 1, 0, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 1, 1, 0, 1, 1},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 1, 0, 0, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 1, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 0, 0, 0, 1, 0, 1, 0, 1},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0} };
	\tint n = 10;
	\tvector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (init_graph[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	\tbfs(graph, 6);
		6 2 3 5 1 8 9 0 7 4	6 2 3 5 1 8 9 0 7 4	✓

✓	\tint init_graph[10][10] = { {0, 1, 1, 0, 1, 0, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 1, 1, 0, 1, 1},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 1, 0, 0, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 1, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 0, 0, 0, 1, 0, 1, 0, 1},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0} };
	\tint n = 10;
	\tvector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (init_graph[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	\tbfs(graph, 7);
		7 0 5 6 8 1 2 4 3 9	7 0 5 6 8 1 2 4 3 9	✓

	Test	Expected	Got	

✓	\tint init_graph[10][10] = { {0, 1, 1, 0, 1, 0, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 1, 1, 0, 1, 1},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 1, 0, 0, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 1, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 0, 0, 0, 1, 0, 1, 0, 1},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0} };
	\tint n = 10;
	\tvector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (init_graph[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	\tbfs(graph, 8);
		8 5 7 9 0 2 4 6 1 3	8 5 7 9 0 2 4 6 1 3	✓

✓	\tint init_graph[10][10] = { {0, 1, 1, 0, 1, 0, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 1, 1, 0, 1, 1},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{0, 1, 0, 0, 0, 0, 0, 1, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 1, 1, 0, 1, 0, 0, 0, 0},
	\t\t\t\t\t\t\t\t{1, 0, 0, 0, 0, 1, 1, 0, 1, 0},
	\t\t\t\t\t\t\t\t{0, 0, 0, 0, 0, 1, 0, 1, 0, 1},
	\t\t\t\t\t\t\t\t{1, 0, 1, 0, 1, 0, 0, 0, 1, 0} };
	\tint n = 10;
	\tvector<vector<int>> graph(n, vector<int>());
	\tfor (int i = 0; i < n; ++i) {
	\t\tfor (int j = 0; j < n; ++j) {
	\t\t\tif (init_graph[i][j]) graph[i].push_back(j);
	\t\t}
	\t}
	
	\tbfs(graph, 9);
		9 0 2 4 8 1 6 5 7 3	9 0 2 4 8 1 6 5 7 3	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **5**

Correct

Mark 1.00 out of 1.00

A nice number is a positive integer that contains only 2's and 5's.
Some nice numbers are: 2, 5, 22, 25, 52, 55, ...
Number 2 is the first nice number.
Given an integer N, return the Nth nice number.
Note: iostream, vector, queue are already included for you.

Constraint:
1 <= n <= 10^6

Example 1:
Input:
n = 5
Output:
52

Explanation:
The sequence of nice numbers is 2, 5, 22, 25, 52, 55, ...
The 5th number in this sequence is 52

Example 2:
Input:
n = 10000
Output:
2255522252225

For example:

Test	Input	Result

`int n;`
`cin >> n;`
`cout << nthNiceNumber(n) << endl;`
	5	52

`int n;`
`cin >> n;`
`cout << nthNiceNumber(n) << endl;`
	10000	2255522252225

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
// iostream, vector and queue are included
// You can write helper methods

long long nthNiceNumber(int n) {
    queue<long long> q;
    q.push(2);
    q.push(5);
    for(int i=1;i<n;i++){
        long long temp=q.front();
        q.pop();
        q.push(temp*10+2);
        q.push(temp*10+5);
    }
    return q.front();
}
```

```
}
```

	Test	Input	Expected	Got	
✓	int n; cin >> n; cout << nthNiceNumber(n) << endl;	5	52	52	✓
✓	int n; cin >> n; cout << nthNiceNumber(n) << endl;	10000	2255522252225	2255522252225	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 6

Correct

Mark 1.00 out of 1.00

Given a n*m grid where each cell in the grid can have a value of 0, 1 or 2, which has the following meaning:
1. Empty cell
2. This cell contains a fresh apple
3. This cell contains a rotten apple
After 1 second, the cell with rotten apple will rot all fresh apples in all the cells adjacent to it (i.e the cells (x+1, y), (x-1, y), (x, y+1), (x, y-1))
Determine the minimum time (in seconds) required to rot all apples. If this cannot be done, return -1.
Note: iostream, vector, and queue are already included.

Constraint:
1 <= n, m <= 500

Hint: Have you ever heard about breadth-first-search?

Example 1:
Input: grid = {{2,2,0,1}}
Output: -1
Explanation:
The grid is
2 2 0 1
The apple at (0, 3) cannot be rotten

Example 2:
Input: grid = {{0,1,2},{0,1,2},{2,1,1}}
Output: 1
Explanation:
The grid is
0 1 2
0 1 2
2 1 1
Apples at positions (0,2), (1,2), (2,0)
will rot apples at (0,1), (1,1), (2,2) and (2,1) after 1 second.

For example:

Test	Input	Result

`int rows, cols;`
`cin >> rows >> cols;`
`vector<vector<int>> grid(rows, vector<int>(cols));`
`for(int i = 0; i < rows; i++) {`
`  for(int j = 0; j < cols; j++) cin >> grid[i][j];`
`}`
`cout << secondsToBeRotten(grid);`
	1 4
	2 2 0 1
		-1

`int rows, cols;`
`cin >> rows >> cols;`
`vector<vector<int>> grid(rows, vector<int>(cols));`
`for(int i = 0; i < rows; i++) {`
`  for(int j = 0; j < cols; j++) cin >> grid[i][j];`
`}`
`cout << secondsToBeRotten(grid);`
	3 3
	0 1 2
	0 1 2
	2 1 1
		1

Answer: (penalty regime: 0 %)

Reset answer

```
// iostream, vector and queue are included
// Hint: use breadth-first-search

int secondsToBeRotten(vector<vector<int>>& grid) {
    int n = grid.size();
    int m = grid[0].size();
    int freshApples = 0;
    queue<pair<int, int>> rottenApples;

    // Count fresh apples and enqueue rotten apples
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (grid[i][j] == 1)
                freshApples++;
            else if (grid[i][j] == 2)
                rottenApples.push({i, j});
        }
    }

    int time = 0;
    vector<pair<int, int>> directions = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

    while (!rottenApples.empty() && freshApples > 0) {
        int size = rottenApples.size();
        for (int i = 0; i < size; ++i) {
            int x = rottenApples.front().first;
            int y = rottenApples.front().second;
            rottenApples.pop();

            for (const auto& dir : directions) {
                int nx = x + dir.first;
                int ny = y + dir.second;
                if (nx >= 0 && nx < n && ny >= 0 && ny < m && grid[nx][ny] == 1) {
                    grid[nx][ny] = 2;
                    rottenApples.push({nx, ny});
                    freshApples--;
                }
            }
        }
        time++;
    }

    return freshApples == 0 ? time : -1;}
```

	Test	Input	Expected	Got	

✓	int rows, cols;
	cin >> rows >> cols;
	vector<vector<int>> grid(rows, vector<int>(cols));
	for(int i = 0; i < rows; i++) {
	for(int j = 0; j < cols; j++) cin >> grid[i][j];
	}
	cout << secondsToBeRotten(grid);
		1 4
		2 2 0 1
			-1	-1	✓

✓	int rows, cols;
	cin >> rows >> cols;
	vector<vector<int>> grid(rows, vector<int>(cols));
	for(int i = 0; i < rows; i++) {
	for(int j = 0; j < cols; j++) cin >> grid[i][j];
	}
	cout << secondsToBeRotten(grid);
		3 3
		0 1 2
		0 1 2
		2 1 1
			1	1	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **7**

Correct

Mark 1.00 out of 1.00

A group of N students in HCMUT are playing a funny game. They gather around a circle and number themselves from 1 to N clockwise. After a step of the game, a person is removed from the circle. The last person to stay in the circle is the winner.

The game's rule is as follows:

1. The game start at the person numbered 1.
2. From the current person, count k people clockwise (including the person you started at). The counting may wraps around the circle.
3. The last counted one is remove from the circle.
4. If the circle still has more than one people, the game continues from the person immediately clockwise of the person who just lost the game. Then repeat step 2
5. The last person in the game will win.

Toan really wants to win the game to impress their friends. Given the number of players, N, and an integer, k. Help Toan win the game by determine the number in which he has to be standing to certainly win the game.

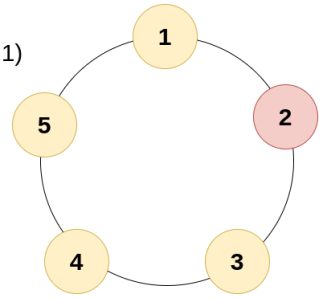
Included libraries: queue, stack, list, vector

Constraint:
1 <= k <= N <= 10^4

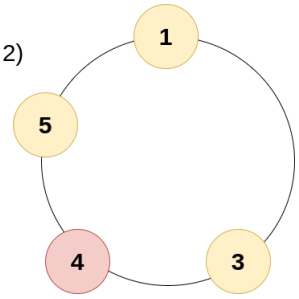
Example:
Input:
n = 5, k = 2
Output
3

Explanation: The game proceeds as follows

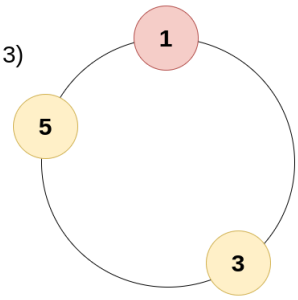
1)



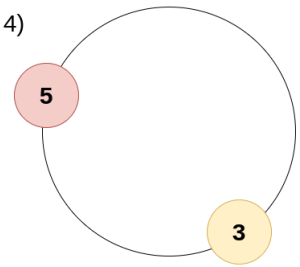
2)



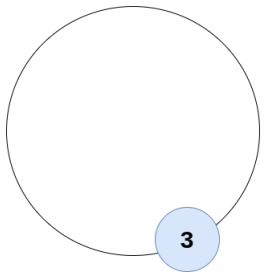
3)



4)



5)



For example:

Test	Input	Result
int N; int k; cin >> N >> k; cout << numberOfTheWinner(N, k);	5 2	3
int N; int k; cin >> N >> k; cout << numberOfTheWinner(N, k);	6 5	1

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
// iostream and queue are included
// Hint: Use a queue to simulate the process

int numberOfTheWinner(int N, int k) {
    queue<int> q;
    for(int i=1;i<=N;i++){
        q.push(i);
    }
    while(q.front()!=q.back()){
        int x=k-1;
        while(x>0){
            int temp=q.front();
            q.pop();
            q.push(temp);
```

```
            x--;
        }
        q.pop();
    }
    return q.front();
}
```

	Test	Input	Expected	Got	
✓	int N; int k; cin >> N >> k; cout << numberOfTheWinner(N, k);	5 2	3	3	✓
✓	int N; int k; cin >> N >> k; cout << numberOfTheWinner(N, k);	6 5	1	1	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.