Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Năm, 29 tháng 2 2024, 6:13 PM
Kết thúc lúc	Thứ Ba, 5 tháng 3 2024, 4:34 PM
Thời gian thực hiện	4 Các ngày 22 giờ

Câu hỏi **1**
Đúng
Đạt điểm 1,00

[Eng] Given a queue of integers of even length, rearrange the elements by interleaving the first half of the queue with the second half of the queue.

Your task is to implement interleaveQueue function.

stack and queue are included.

[Vie] Cho 1 hàng đợi có số lượng phần tử là số chẵn, sắp xếp lại các phần tử theo quy tắc xen kẽ phần tử ở nửa đầu và nửa sau của hàng đợi.

Sinh viên cần hiện thực hàm interleaveQueue.

Thư viện stack và queue đã được thêm vào.

For example:

Test	Input	Result

`queue<int> q;`
`    int n; cin >> n;`
`    for (int i = 0; i < n; i++){`
`        int element; cin >> element;`
`        q.push(element);`
`    }`
`    interleaveQueue(q);`
`    while (!q.empty()){`
`        cout << q.front() << ' ';`
`        q.pop();`
`    }`
	4
	1 2 3 4
		1 3 2 4

`queue<int> q;`
`    int n; cin >> n;`
`    for (int i = 0; i < n; i++){`
`        int element; cin >> element;`
`        q.push(element);`
`    }`
`    interleaveQueue(q);`
`    while (!q.empty()){`
`        cout << q.front() << ' ';`
`        q.pop();`
`    }`
	6
	2 4 6 8 10 12
		2 8 4 10 6 12

Answer: (penalty regime: 0 %)

Reset answer

```
void reverseVector(vector<int>& v) {
    int start = 0;
    int end = v.size() - 1;
    while (start < end) {
        swap(v[start], v[end]);
        start++;
        end--;
    }
}

void interleaveQueue(queue<int>& q) {
    vector<int> ans;
    int size = q.size();
    for (int i = 0; i < size / 2; i++) {
        ans.push_back(q.front());
        q.pop();
    }
    reverseVector(ans);

    while (!ans.empty()) {
        q.push(ans.back());
        ans.pop_back();
        int temp = q.front();
        q.pop();
        q.push(temp);
    }
}

```

	Test	Input	Expected	Got	

✓	`queue<int> q;`
	`    int n; cin >> n;`
	`    for (int i = 0; i < n; i++){`
	`        int element; cin >> element;`
	`        q.push(element);`
	`    }`
	`    interleaveQueue(q);`
	`    while (!q.empty()){`
	`        cout << q.front() << ' ';`
	`        q.pop();`
	`    }`
		4
		1 2 3 4
			1 3 2 4	1 3 2 4	✓

✓	`queue<int> q;`
	`    int n; cin >> n;`
	`    for (int i = 0; i < n; i++){`
	`        int element; cin >> element;`
	`        q.push(element);`
	`    }`
	`    interleaveQueue(q);`
	`    while (!q.empty()){`
	`        cout << q.front() << ' ';`
	`        q.pop();`
	`    }`
		6
		2 4 6 8 10 12
			2 8 4 10 6 12	2 8 4 10 6 12	✓

Passed all tests! ✓

Câu hỏi **2**
Đúng
Đạt điểm 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, 0);
    queue<int> q;

    for (int i = 0; i < n; ++i) {
        if (colors[i] != 0) continue;
        if (colors[i] != 0) continue;

        q.push(i);
        colors[i] = 1;

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

            for (int neighbor : graph[curr]) {
                if (colors[neighbor] == 0) {
                    colors[neighbor] = -colors[curr];
                    q.push(neighbor);
                } else if (colors[neighbor] == colors[curr]) {
                    return false;
                }
            }
        }
    }

    return true;
}
```

Passed all tests! ✓

Câu hỏi **3**
Đúng
Đạt điểm 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;

    q.push(start);
    visited[start] = true;

    while (!q.empty()) {
        int curr = q.front();
        int curr = q.front();
        q.pop();
        cout << curr << " ";

        for (int neighbor : graph[curr]) {
            if (!visited[neighbor]) {
                q.push(neighbor);
                visited[neighbor] = true;
            }
        }
    }
}

```

Passed all tests! ✓

Câu hỏi **4**
Đúng
Đạt điểm 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) {
        list.add(item);
    }

    T pop() {
        if (list.empty()) {
            throw std::out_of_range("Queue is empty");
        }
        T frontElement = list.get(0);
        list.removeAt(0);
        return frontElement;
    }

    T top() {
        if (list.empty()) {
            throw std::out_of_range("Queue is empty");
        }
        return list.get(0);
    }

    bool empty() {
        return list.empty();
    }

    int size() {
        return list.size();
    }

    void clear() {
        list.clear();
    }

```

Passed all tests! ✓

Câu hỏi **5**
Đúng
Đạt điểm 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

```
long long nthNiceNumber(int n) {
    queue<long long> q;
    q.push(2);
    q.push(5);
    long long nthNiceNumber = 0;

    for (int i = 0; i < n; ++i) {
        nthNiceNumber = q.front();
        q.pop();
        q.push(nthNiceNumber * 10 + 2);
        q.push(nthNiceNumber * 10 + 5);
    }

    return nthNiceNumber;
}

```

	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! ✓

Câu hỏi 6
Đúng
Đạt điểm 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

```
int secondsToBeRotten(std::vector<std::vector<int>>& grid) {
    int n = grid.size();
    int m = grid[0].size();
    int freshCount = 0;
    std::queue<std::pair<int, int>> rottenApples;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (grid[i][j] == 1) {
                ++freshCount;
            } else if (grid[i][j] == 2) {
                rottenApples.push({i, j});
            }
        }
    }
    int seconds = 0;
    std::vector<std::pair<int, int>> directions = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
    while (!rottenApples.empty() && freshCount > 0) {
        int size = rottenApples.size();
        bool rotten = false;

        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});
                    --freshCount;
                    rotten = true;
                }
            }
        }

        if (rotten) {
            ++seconds;
        }
    }
    return freshCount == 0 ? seconds : -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! ✓

Câu hỏi 7
Đúng
Đạt điểm 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) {
    int n = nums.size();
    int sum = 0;
    std::deque<int> dq;
    for (int i = 0; i < k; ++i) {
        while (!dq.empty() && nums[i] >= nums[dq.back()]) {
            dq.pop_back();
        }
        dq.push_back(i);
    }
    sum += nums[dq.front()];
    for (int i = k; i < n; ++i) {
        while (!dq.empty() && dq.front() <= i - k) {
            dq.pop_front();
        }
        while (!dq.empty() && nums[i] >= nums[dq.back()]) {
            dq.pop_back();
        }
        dq.push_back(i);
        sum += nums[dq.front()];
    }
    return sum;
}

```

	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! ✓