Laboratory work # 2

Student: *Zhanghao Chen*
Student ID: 21321205
Timus Name: *BennyChan*

Mail: 1824362950@qq.com

Problem # 1401 *Gamers*

Screenshot from Timus:

ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
10241085	18:40:39 14 Apr 2023	BennyChan	1401. Gamers	G++ 9.2 x64	Accepted		0.015	1 172 KB

Explanation of algorithm:

In this program, I first enter three integers n, x, and y. Then, generate a matrix M of $2^n \times 2^n$ by calling the paveSquare function. Among them, the paveSquare function takes a recursive way to continuously divide the matrix until the matrix size is 2×2. The function findFake is used to find the fake cell of each matrix block, where, x and y specify the upper-left position of the matrix block currently being processed, and *fakex* and *fakey* specify the position of the deleted cell, and the value of each position in the matrix is successively a multiple of 3. The resulting matrix M is output at the end of the program.

Computational complexity of algorithm:

$$O(N^2)$$

Source code:

```
#include <cmath>
#include <iostream>
using namespace std;
int a = 3, M[512][512];

void findFake(int n, int x, int y, int fakex, int fakey){
    for (int i = 0; i < 2; i++){
        for (int j = 0; j < 2; j++){
```

```
            if (x + i * n / 2 > fakex || fakex >= x + i * n / 2 + n / 2 || y + j * n / 2 > fakey || fakey >= y + j * n  + n / 2){
                M[x + n / 2 - 1 + i][y + n / 2 - 1 + j] = a++ / 3;
            }
        }
    }
}

void paveSquare(int n, int x, int y, int fakex, int fakey) {
    if (n == 2){
        for (int i = 0; i < 2; i++){
            for (int j = 0; j < 2; j++){
                if (x + i != fakex || y + j != fakey){
                    M[x + i][y + j] = a++ / 3;
                }
            }
        }
        return;
    }
    findFake(n ,x, y, fakex, fakey);
    for (int i = 0; i < 2; i++){
        for (int j = 0; j < 2; j++){
            if (x + i * n / 2 <= fakex && fakex < x + i * n / 2 + n / 2 && y + j * n / 2 <= fakey && fakey < y + j * n /2 + n / 2){
                paveSquare(n /2, x + i * n / 2, y + j * n / 2, fakex, fakey);
            }
            else {
                paveSquare(n / 2, x + i * n / 2, y + j * n / 2, x + n / 2 - 1 + i, y + n / 2 - 1 + j);
            }
        }
    }
}

int main(){
    int n, x, y;
    cin >> n >> x >> y;
    int d = pow(2 , n);
    paveSquare(d, 0, 0, x - 1, y - 1);
    for(int i = 0; i < d; i++){
        for(int j = 0; j < d; j++){
            cout << M[i][j] << " ";
        }
        cout << endl;
    }
}
```

Problem # 2025 *Line Fighting*

Screenshot from Timus:

10239909	15:58:06 13 Apr 2023	BennyChan	2025. Line Fighting	G++ 9.2 x64	Accepted		0.031	404 KB

Explanation of algorithm:

We can divide the players into equal k groups of $\frac{n}{k}$ due to the method of Lagrange multipliers. If we randomly group players, then the number of games within each group may vary greatly. So equal players in each group must be the best solution. Thus, there will be $\frac{n}{k} \times (n - \frac{n}{k})$ matches between each group. This is because players within each group do not

play against each other, so the number of matches only relates to players between different groups.

Computational complexity of algorithm:

$$O(N)$$

Source code:

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
    int T;
    int total_number = 0;
    cin >> T;
    int n[T], k[T];
    for (int i = 0; i < T; i++) {
        cin >> n[i] >> k[i];
    }
    for (int i = 0; i < T; i++) {
        while (k[i] >= 2) {
            total_number += (n[i] - n[i] / k[i]) * (n[i] / k[i]);
            n[i] = n[i] - n[i] / k[i];
            k[i] = k[i] - 1;
        }
        cout << total_number << endl;
        total_number = 0;
    }
    return 0;
}
```