컴퓨터알고리즘과실습 실습11

2020112046 이재원

11-1. 실행 결과

20번째 피보나치 수는 top-down식의 재귀함수로,

10000번째 피보나치 수는 bottom-up 방식의 반복문으로 구현했다.

10000번째 피보나치 수는 굉장히 커서 원시 자료형에 담기지 않기 때문에, int형의 배열을 이용하여 담았다.

코드

```
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int fib(int n) {
    if (n == 0) return 1;
    if (n == 1) return 1;
    return fib(n - 1) + fib(n - 2);
}

int dp[300][1000000] = {
    0,
};

void bottom_up_fib(int n) {
    dp[0][0] = 1;
    dp[0][1] = 1;

    for (int i = 2; i <= n - 1; i++) {
        // big number addition with int array
```

```
        int carry = 0;
        for (int j = 0; j < 300; j++) {
            dp[j][i] = dp[j][i - 1] + dp[j][i - 2] + carry;
            if (dp[j][i - 1] == 0) break;   // if previous number is 0, break
            carry = dp[j][i] / 1000000000;   // int형의 범위가 2,147,483,647이므로 1,000,000,000으로 나눠준다. (10^9 자리까지만 사용)
            dp[j][i] -= carry * 1000000000;
        }
    }
}

int main() {
    cout << "피보나치 20번째 수 (top-down) : " << fib(20) << endl;
    int N = 10000;
    bottom_up_fib(N);
    // print dp[9999]
    cout << "피보나치 10000번째 수 (bottom-up) : ";
    for (int i = 299; i >= 0; i--) {
        if (dp[i][N - 1] != 0) cout << dp[i][N - 1];
    }
}
```

11-2. 실행 결과

```
-------- 4x4 예시 행렬에 대한 경로 검색 --------
40
6 13 25 30
11 14 25 43
18 31 28 31
26 36 42 40
-------- 랜덤 200x200 행렬에 대한 경로 검색 --------
bottom-up 방식 : 21352
bottom-up 방식 연산 횟수 : 39800
top-down 방식 : 21352
top-down 방식 함수 호출 횟수 : 79999
```

4x4 행렬과 랜덤 200x200행렬에 대해 top-down, bottom-up 방식으로 경로 탐색을 해 보았다.

재귀로 구현한 Top-down 방식의 함수 호출 횟수만 해도 bottom-up 방식의 연산 횟수보다 큰 것을 확인할 수 있는데, 연산 횟수로 비교할 것을 생각하면, top-down 방식이 확연히 느리다는 것을 확인할 수 있다.

코드

```
#include <algorithm>
#include <iostream>
#include <random>
#include <vector>
```

```
using namespace std;

#define N 200

int matrix[4][4] = {
    {6, 7, 12, 5},
    {5, 3, 11, 18},
    {7, 17, 3, 3},
    {8, 10, 14, 9},
};
int minMatrix[4][4];

int findMinRoute_exampleMatrix() {
    minMatrix[0][0] = matrix[0][0];
    for (int i = 1; i < 4; i++) {
        minMatrix[0][i] = minMatrix[0][i - 1] + matrix[0][i];
        minMatrix[i][0] = minMatrix[i - 1][0] + matrix[i][0];
    }
    for (int i = 1; i < 4; i++) {
        for (int j = 1; j < 4; j++) {
            minMatrix[i][j] = min(minMatrix[i - 1][j], minMatrix[i][j - 1]) +
matrix[i][j];
        }
    }
    return minMatrix[3][3];
}

int BottomUpCount = 0;
int TopDownCount = 0;

int minRandomMatrix[N][N];
int findMinRoute_randomMatrix_bottomUp(int randomMatrix[N][N]) {
    minRandomMatrix[0][0] = randomMatrix[0][0];
    for (int i = 1; i < N; i++) {
        BottomUpCount++;
        minRandomMatrix[0][i] = minRandomMatrix[0][i - 1] + randomMatrix[0][i];
        minRandomMatrix[i][0] = minRandomMatrix[i - 1][0] + randomMatrix[i][0];
    }
    for (int i = 1; i < N; i++) {
        for (int j = 1; j < N; j++) {
            BottomUpCount++;
            minRandomMatrix[i][j] = min(minRandomMatrix[i - 1][j], minRandomMatrix[i][j
- 1]) + randomMatrix[i][j];
        }
    }
    return minRandomMatrix[199][199];
}

int minRandomMatrix2[N][N];
int findMinRoute_randomMatrix_topDown(int randomMatrix[N][N], int i, int j) {
    TopDownCount++;
    if (i == 0 && j == 0) {
        return randomMatrix[0][0];
```

```
    }
    if (i < 0 || j < 0) {
        return 10000;
    }
    if (minRandomMatrix2[i][j] != 0) {
        return minRandomMatrix2[i][j];
    }

    minRandomMatrix2[i][j] = min(findMinRoute_randomMatrix_topDown(randomMatrix, i - 1,
j), findMinRoute_randomMatrix_topDown(randomMatrix, i, j - 1)) + randomMatrix[i][j];
    return minRandomMatrix2[i][j];
}


int main() {
    int minRoute = findMinRoute_exampleMatrix();
    cout << "-------- 4x4 예시 행렬에 대한 경로 검색 --------" << endl;
    cout << minRoute << endl;
    // print minMatrix
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            cout << minMatrix[i][j] << " ";
        }
        cout << endl;
    }
    cout << "-------- 랜덤 200x200 행렬에 대한 경로 검색 --------" << endl;
    // random 200x200 matrix (1 ~ 200)
    random_device rd;
    int randomMatrix[N][N];
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            randomMatrix[i][j] = rd() % N;
        }
    }

    minRoute = findMinRoute_randomMatrix_bottomUp(randomMatrix);
    cout << "bottom-up 방식: " << minRoute << endl;
    cout << "bottom-up 방식 연산 횟수: " << BottomUpCount << endl;
    minRoute = findMinRoute_randomMatrix_topDown(randomMatrix, N - 1, N - 1);
    cout << "top-down 방식: " << minRoute << endl;
    cout << "top-down 방식 함수 호출 횟수: " << TopDownCount << endl;
}
```