(훈련반2) Level28

그래프의 일종인 트리에 대해 배워봅니다.
단순히 **부모-자식관계**로 되어있는 그래프라면 **트리**라고 보면 됩니다.

트리를 이해하고, 트리를 DFS로 탐색하는 것까지 이번 Level의 목표입니다.
그래프, 트리, DFS를 혼동하지 않도록 합시다.

1. **그래프** : 자료구조, 링크드리스트와 달리 **각 노드들의 관계**까지 저장하는 구조
2. **트리** : 자료구조, 그래프의 일종, **Cycle이 없고 부모 자식 구조**인 그래프
3. **DFS** : 그래프(트리)를 탐색하는 방법(알고리즘), 깊이 우선 방식 (Level 28 과정)
4. **BFS** : 그래프(트리)를 탐색하는 방법(알고리즘), 너비 우선 방식 (Level 29 과정)

Level28 보스와 부하들

문제 1번 [숙제 목록보기]

현수는 다른 조직으로 이직에 성공했습니다.

이 그룹의 조직도를 **인접행렬(N x N 사이즈)**로 전달 받으면,

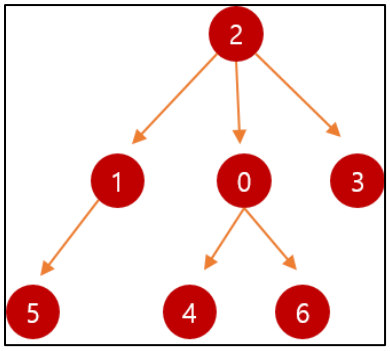
현수의 직속 보스와 직속 부하 들이 누군지 출력 해 주세요.

문제 조건

1. 현수는 0번 노드입니다.
2. 부하들끼리 번호 순서대로 출력 해 주세요

예시

만약 아래와 같은 인접행렬을 입력 받았다면,



boss:2

under:4 6

입력 예제

```
7
0 0 0 0 1 0 1
0 0 0 0 0 1 0
1 1 0 1 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

출력 결과

```
boss:2
under:4 6
```

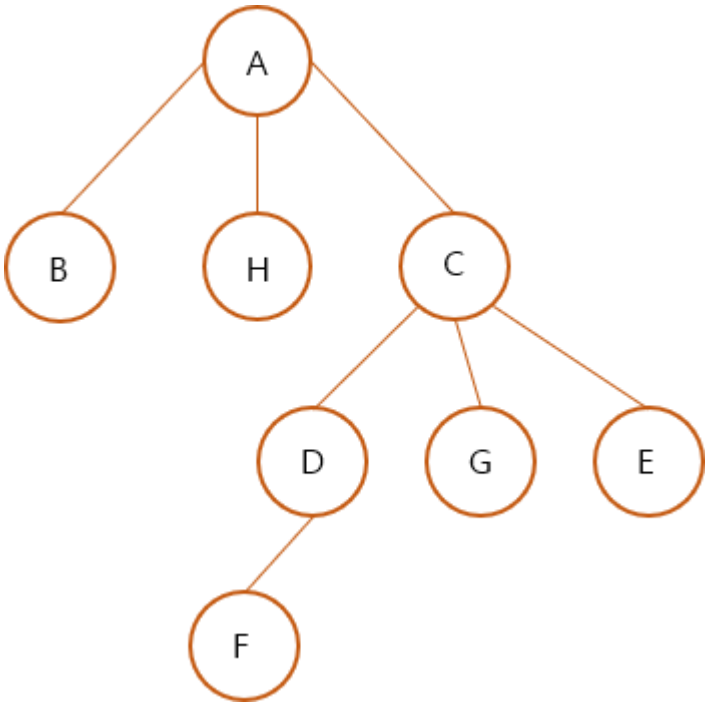
```cpp
#include <iostream>
using namespace std;
/*
	[문제]
	- 인접행렬로 값을 입력받는다.
	- 인접행렬의 크기도 입력받는다.
	- 현수의 번호는 0번이다.
*/
int** matrix = nullptr;  // 2차원 배열이기에 이차원 포인터 변수사용
int arrSize = 0;         // 인접행렬의 사이즈
int targetIdx = 0;               // 현수의 Idx

void assignMemory()              // 메모리 할당
{
	matrix = new int* [arrSize];             // 이차원 행렬의 y축 할당
	for (int i = 0; i < arrSize; ++i)
		matrix[i] = new int[arrSize];    // 이차원 행렬의 x축 할당
}

void inputValue()        // 인접행렬 값 입력받기
{
	for (int y = 0; y < arrSize; ++y)
	{
		for (int x = 0; x < arrSize; ++x)
			cin >> matrix[y][x];
	}
}

const int yes = 1;
void whoBoss()
{
	cout << "boss:";
	for (int y = 0; y < arrSize; ++y)
	{
		int isBoss = matrix[y][targetIdx];
		if (isBoss == yes)
			cout << y << " ";
	}
	cout << endl;
}

void whoUnder()
{
	cout << "under:";
	for (int x = 0; x < arrSize; ++x)
	{
		int isUnder = matrix[targetIdx][x];
```

```
                if (isUnder == yes)
                        cout << x << " ";
        }
}

int main(void)
{
        cin >> arrSize;
        assignMemory();
        inputValue();
        whoBoss();
        whoUnder();

        return 0;
}
```

Level28 잃어버린 가족상봉

문제 2번 [숙제 목록보기]

마마코코 가족의 계보는 다음과 같습니다. (A ~ H)



다음과 같은 그래프를 **인접행렬 (2차배열)로 하드코딩** 해주세요.

이제 **노드이름을 입력받고, 그 노드의 형제들을 모두 출력** 해주세요.

(만약 형제들이 없다면 "**없음**" 을 출력해주시면 됩니다.)

예를들어 H의 형제는 B와 C입니다.

A의 형제는 없습니다.

C의 형제는 B와 H입니다.

입력 예제

```
H
```

출력 결과

```
B C
```

```cpp
#include <iostream>
using namespace std;
/*
	[문제]
	- 마마코코 가족의 계보를 인접행렬로 나타냄
	  ┆    A      B      C      D      E      F      G      H
	---------------------------------
	A ┆ 0  1      1      0      0      0      0      1
	B ┆    1      0      0      0      0      0      0      0
	C ┆    1      0      0      1      1      0      1      0
	D ┆    0      0      1      0      0      1      0      0
	E ┆    0      0      1      0      0      0      0      0
	F ┆    0      0      0      1      0      0      0      0
	G ┆    0      0      1      0      0      0      0      0
	H ┆    1      0      0      0      0      0      0      0
*/

char input = '\0';
const int g_size = 8;
char names[g_size + 1] = "ABCDEFGH";
bool isCheck[g_size] = {};
const bool yes = true;
const bool no = false;
char path[g_size + 1] = {};
int pathIdx = 0;                // path에 다음 값이 들어갈 위치를 나타냄
const int startLevel = 0;       // 족보 시작 레벨 : 0
const int referIdx = 0;         // 족보 시작 인물의 인덱스 : 0 -> names[0] = 'A'
int inputLevel = 0;
bool isBrother[g_size] = {};

int matrix[g_size][g_size] =
{
	0,1,1,0,0,0,0,1,
	1,0,0,0,0,0,0,0,
	1,0,0,1,1,0,1,0,
	0,0,1,0,0,1,0,0,
	0,0,1,0,0,0,0,0,
	0,0,0,1,0,0,0,0,
	0,0,1,0,0,0,0,0,
	1,0,0,0,0,0,0,0,
};

void recursiveForInputLevel(int level, int referIdx)
// 입력한 사람의 족보상 위치를 찾아준다.
{
	int isFamily = 0;
	for (int i = 0; i < g_size; ++i)
```

```
        {
                isFamily = matrix[referIdx][i];
                if (isFamily == 1 && isCheck[i] == no)
                {
                        if (names[i] == input)
                                inputLevel = pathIdx;
                        else
                        {
                                isCheck[i] = yes;
                                path[pathIdx] = names[i];
                                ++pathIdx;

                                recursiveForInputLevel(level + 1, i);

                                --pathIdx;
                                path[pathIdx] = '\0';
                                isCheck[i] = no;
                        }
                }
        }
}

void findInputLevel()
{
        path[pathIdx] = 'A';                    // 시작점 : A
        ++pathIdx;                              // A가 들어갔기에 idx+1 해준다.
        isCheck[0] = true;                      // A가 들어갔기에 체크
        recursiveForInputLevel(startLevel, referIdx);
}

void recursiveForBrother(int level, int referIdx) // 입력한 사람의 형재를 찾아준다.
{
        int isFamily = 0;
        for (int i = 0; i < g_size; ++i)
        {
                isFamily = matrix[referIdx][i];
                if (isFamily == 1 && isCheck[i] == no)
                {
                        if (pathIdx == inputLevel && names[i] != input)
                                isBrother[i] = yes;
                        isCheck[i] = yes;
                        path[pathIdx] = names[i];
                        ++pathIdx;

                        recursiveForBrother(level + 1, i);

                        --pathIdx;
```

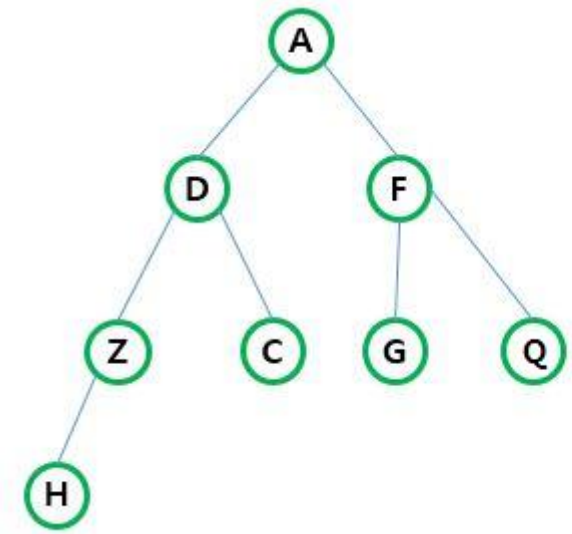
```
                        path[pathIdx] = '\0';
                        isCheck[i] = no;
                }
        }
}

void findBrother()
{
        for (int i = 0; i < g_size; ++i)  // path 배열 초기화
                path[i] = '\0';
        pathIdx = 0;                      // 초기화
        for (int k = 0; k < g_size; ++k)  // isCheck 배열 초기화
                isCheck[k] = false;
        path[pathIdx] = 'A';              // 시작점 : A
        ++pathIdx;                        // A가 들어갔기에 idx+1 해준다.
        isCheck[0] = true;                // A가 들어갔기에 체크
        recursiveForBrother(startLevel, referIdx);
}

void printBrother()
{
        for (int i = 0; i < g_size; ++i)
        {
                if (isBrother[i] == yes)
                        cout << names[i];
        }
}

int main(void)
{
        cin >> input;

        findInputLevel();
        findBrother();
        printBrother();

        return 0;
}
```

Level28 부모자식관계 판별

문제 3번 [숙제 목록보기]

아래 이진 트리를 **1차원 배열**에 저장하세요.

(**Root 노드인 'A'를 1번 Index에 두는 것을 잊지마세요.**)



이제 문자 2개를 입력 받으세요.

그 문자에 해당하는 노드가 서로 부모자식 관계인지 아닌지 출력 하세요.

ex)

G F -> 부모자식관계

ex)

Z C -> 아님

입력 예제

```
G F
```

출력 결과

```
부모자식관계
```

```cpp
#include <iostream>
using namespace std;

/*
	[문제]
	이진 트리를 1차원 배열에 저장

	idx		value
	 0		NULL(0)
	 1		a = 1
	 2		d = a * 2 = 2
	 3		f = a * 2 + 1 = 3
	 4		z = d * 2 = 4
	 5		c = d * 2 + 1 = 5
	 6		g = f * 2 = 6
	 7		q = f * 2 + 1 = 7
	 8		h = z * 2 = 8
*/

const int g_size = 9;                    // 배열의 크기
const char arr[g_size + 1] = "AADFZCGQH"; // 이진트리의 값을 저장하는 1차원 배열
char input1 = '\0';                      // 입력 값1
char input2 = '\0';                      // 입력 값2

int findIdx(char name)
{
	for (int i = 1; i < g_size + 1; ++i)
	{
		if (arr[i] == name)
			return i;
	}

	return g_size;   // 일치 대상이 없으면 이상한 값 출력
}

void compare(int _parent, int _child, bool* _result)
{
	int child = _child;
	while (child > 0)
	{
		if (child % 2 == 1)
			child = (child - 1) / 2;
		else
			child /= 2;

		if (child == _parent)
		{
```

```
                        *_result = true;
                        break;
                }
        }
}

void isFamily()
{
        int idx1 = findIdx(input1);       // input1의 arr배열에서의 idx값
        int idx2 = findIdx(input2);       // input2의 arr배열에서의 idx값
        bool result = false;              // 가족여부

        if (idx1 == idx2)
                __noop;                   // result = false;
        else if (idx1 < idx2)
                compare(idx1, idx2, &result);
        else if (idx1 > idx2)
                compare(idx2, idx1, &result);
        else
                __noop;

        if (result)
                cout << "부모자식관계";
        else
                cout << "아님";
}

int main(void)
{
        cin >> input1 >> input2;
        isFamily();

        return 0;
}
```

Level28 인접행렬 DFS 시작

문제 4번 [숙제 목록보기]

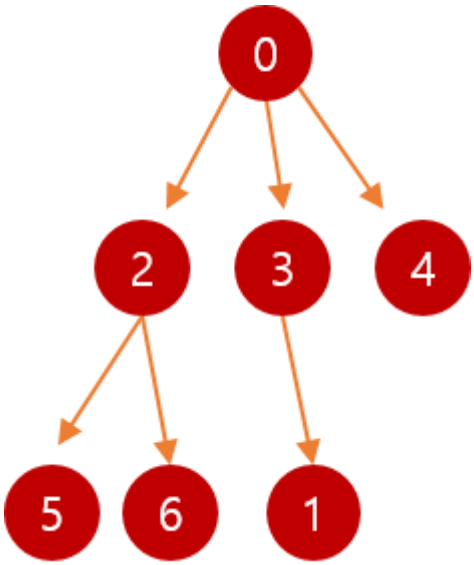
DFS는 그래프 / 트리를 탐색하는 방법입니다. 배열 or 링크드리스트 자료구조는 for문으로 쉽게 탐색이 가능하지만,

그래프 같은 자료구조는 for문으로 탐색이 어렵습니다. DFS로 탐색을 하곤합니다.

N x N 그래프를 인접행렬로 입력받고,

DFS 탐색 순서대로 출력 해 주세요.

ex) 만약 아래와 같은 인접행렬을 입력받았다면,



출력결과 : 0 2 5 6 3 1 4

입력 예제

```
7
0 0 1 1 1 0 0
0 0 0 0 0 0 0
0 0 0 0 0 1 1
0 1 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

출력 결과

```
0 2 5 6 3 1 4
```

```cpp
#include <iostream>
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
using namespace std;

// ascii -> '0' = 48 , '6' = 54
int g_size = 0;
int** matrix = nullptr;          // 인접행렬
const int startLevel = 0;
const int startIdx = 0;
char* path = nullptr;            // DFS경로
int* isVisit = nullptr;
const int offset = 48;
int pathIdx = 0;

void assignMemory()              // 입접행렬 메모리 할당
{
        matrix = new int* [g_size];
        for (int i = 0; i < g_size; ++i)
                matrix[i] = new int[g_size];
}

void deleteMemory()              // 인접행렬 메모리 해제
{
        for (int i = 0; i < g_size; ++i)
        {
                delete[] matrix[i];
                matrix[i] = nullptr;
        }
        delete[] matrix;
        matrix = nullptr;
}

void init()
{
        cin >> g_size;   // 인접행렬 사이즈 입력받기
        assignMemory();
        // 인접행렬 값 입력받기
        for (int y = 0; y < g_size; ++y)
        {
                for (int x = 0; x < g_size; ++x)
                        cin >> matrix[y][x];
        }

        path = new char[g_size + 1];             // path 배열 메모리 할당
        for (int i = 0; i < g_size + 1; ++i)     // path 배열 초기화
```

```
                path[i] = '\0';
        isVisit = new int[g_size];              // isVisit 배열 메모리 할당
        for (int k = 0; k < g_size; ++k)        // isVisit 배열 초기화
                isVisit[k] = 0;
}

void end()
{
        delete[] isVisit;
        isVisit = nullptr;
        delete[] path;
        path = nullptr;
        deleteMemory();
}

void recursive(int _level, int _referIdx)
{
        for (int i = 0; i < g_size; ++i)
        {
                if (matrix[_referIdx][i] == 1 && isVisit[i] != 1)
                {
                        isVisit[i] = 1;
                        path[pathIdx] = i + offset;
                        ++pathIdx;
                        recursive(_level + 1, i);
                }
        }
}

void proc()
{
        int level = startLevel;
        path[pathIdx] = 0 + offset;
        ++level;
        ++pathIdx;
        isVisit[startIdx] = true;
        recursive(level, startIdx);
        //cout << path;
        for (int i = 0; i < g_size; ++i)
                cout << path[i] << " ";
}

int main(void)
{
        init();
        proc();
        end();
```

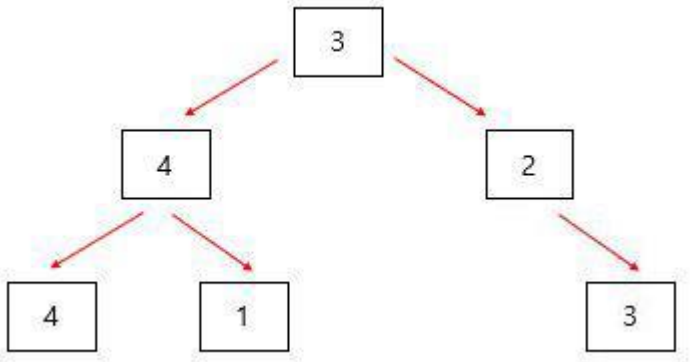
```
        _CrtDumpMemoryLeaks();
        return 0;
}
```

Level28 DFS 깊이우선탐색

문제 5번 [숙제 목록보기]

다음을 하드코딩 해주세요.

위 1차배열은 다음 이진 트리를 나타냅니다.



이제 변경할 index와 값을 입력받아주세요.

만약 4 7 을 입력 받았다면,

배열의 4번 index의 값을 7로 바꾼 후 DFS를 돌리면 됩니다.

DFS 돌린 결과를 출력 해 주세요.

입력 예제

```
4 7
```

출력 결과

```
3 4 7 1 2 3
```

```cpp
#include <iostream>
//#define _CRTDBG_MAP_ALLOC
//#include <stdlib.h>
//#include <crtdbg.h>
using namespace std;

const int nodeCnt = 7;
// 이진트리 -> 1차원 배열
// 1차원 배열의 1번 인덱스부터 값을 넣어준다.
int arr[nodeCnt + 1] = { 0,3,4,2,4,1,0,3 };
int path[nodeCnt] = {};
int startArrIdx = 1;
int pathIdx = 0;

/*
	leftChildIdx	= parentIdx * 2
	rightChildIdx	= parentIdx * 2 + 1
*/

void dfs(int _idx)
{
	int parentIdx = _idx;
	int rChildIdx = parentIdx * 2;		// left
	int lChildIdx = parentIdx * 2 + 1;	// right
	if (parentIdx < nodeCnt + 1)
	{
		int val = arr[parentIdx];
		if (val == 0)
		{
			--pathIdx;
			return;
		}
		path[pathIdx] = val;
		++pathIdx;
		dfs(rChildIdx);
		++pathIdx;
		dfs(lChildIdx);
	}
	else
	{
		--pathIdx;
		return;
	}
}
```

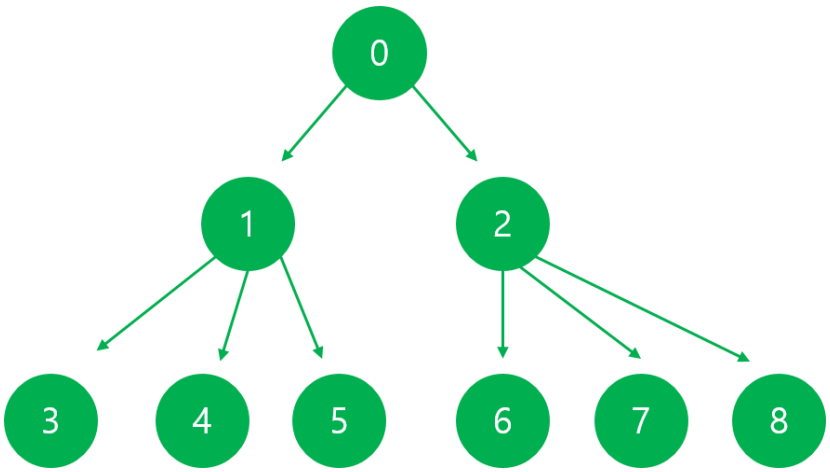
```
void printPath()
{
	for (int i = 0; i < nodeCnt; ++i)
	{
		if (path[i] == 0)
			break;
		cout << path[i] << " ";
	}
}

int main(void)
{
	int inputIdx = 0;
	int inputVal = 0;
	cin >> inputIdx >> inputVal;
	arr[inputIdx] = inputVal;

	dfs(startArrIdx);

	printPath();

	//_CrtDumpMemoryLeaks();
	return 0;
}
```

Level28 2층에서 경로 출력

문제 6번 [숙제 목록보기]

N x N인접행렬 트리를 입력받아주세요.

Level 2에 도착했을 때 마다 경로를 출력하며 됩니다.



만약 위와 같은 트리를 입력받았다면, 다음과 같이 출력하면 됩니다.

```
0 1 3

0 1 4

0 1 5

0 2 6

0 2 7

0 2 8
```

입력 예제

```
9
0 1 1 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

출력 결과

```
0 1 3
0 1 4
0 1 5
0 2 6
0 2 7
0 2 8
```

```cpp
#include <iostream>
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
using namespace std;

int g_size = 0;
int** matrix = nullptr;
const bool yes = true;
const bool no = false;
bool* isVisit = nullptr;
char* path = nullptr;
const int maxPath = 3;
const int offset = 48;   // ascii -> '0' = 48

void init()
{
	matrix = new int* [g_size];
	for (int i = 0; i < g_size; ++i)
		matrix[i] = new int[g_size];
	for (int y = 0; y < g_size; ++y)
	{
		for (int x = 0; x < g_size; ++x)
		{
			cin >> matrix[y][x];
		}
	}
	isVisit = new bool[g_size];
	for (int k = 0; k < g_size; ++k)
		isVisit[k] = false;
	path = new char[maxPath + 1];
	for (int o = 0; o < maxPath + 1; ++o)
		path[o] = '\0';
}

void end()
{
	delete[] path;
	path = nullptr;
	delete[] isVisit;
	isVisit = nullptr;
	for (int i = 0; i < g_size; ++i)
	{
		delete[] matrix[i];
		matrix[i] = nullptr;
	}
	delete[] matrix;
```

```
        matrix = nullptr;
}

void recursive(int _level, int _referIdx)
{
        if (_level == maxPath)
        {
                //cout << path << endl;
                for (int k = 0; k < maxPath; ++k)
                        cout << path[k] << " ";
                cout << endl;
                return;
        }
        for (int i = 0; i < g_size; ++i)
        {
                if (matrix[_referIdx][i] == 1 && isVisit[i] == no)
                {
                        isVisit[i] = yes;
                        path[_level] = i + offset;
                        recursive(_level + 1, i);
                        path[_level] = '\0';
                }
        }
}

void proc()
{
        int startLevel = 0;
        int referIdx = 0;
        isVisit[referIdx] = yes;
        path[startLevel] = referIdx + offset;
        ++startLevel;
        recursive(startLevel, referIdx);

}

int main(void)
{
        cin >> g_size;
        init();
        proc();
        end();

        _CrtDumpMemoryLeaks();
        return 0;
}
```

Level28 짝수까지 가자

문제 7번 [숙제 목록보기]

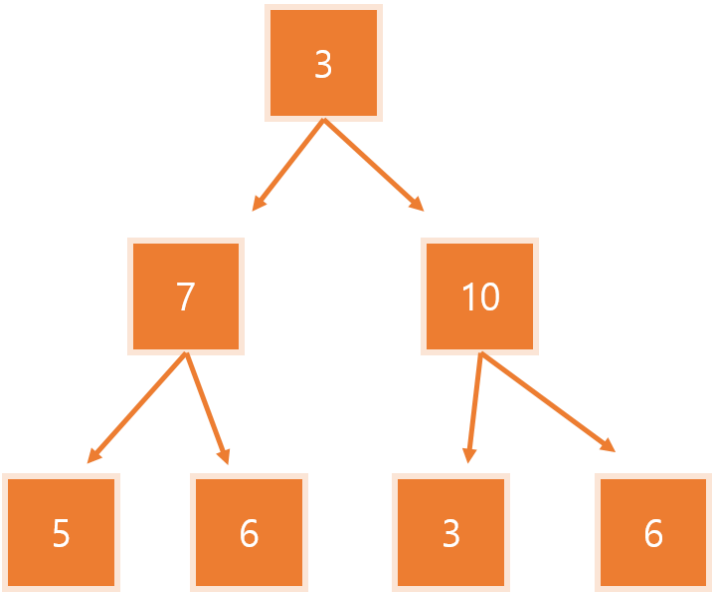
최대 Level이 2인, Binary Tree를 1차원 배열에 입력받아주세요.

(숫자 8개 입력, 숫자 0 은 없는 노드입니다.)

짝수 노드를 발견할 때마다

탐색을 멈추고, 왔었던 경로를 출력하시면 됩니다.

만약 아래 트리와 같이 입력받았다면



출력결과

3 7 6

3 10

입력 예제

```
0 3 7 10 5 6 3 6
```

출력 결과

```
3 7 6
3 10
```

```cpp
#include <iostream>
//#define _CRTDBG_MAP_ALLOC
//#include <stdlib.h>
//#include <crtdbg.h>
using namespace std;

const int nodeCnt = 7;
int arr[nodeCnt + 1] = {};
int path[10] = {};
int startArrIdx = 1;
int pathIdx = 0;

/*
        1[3]    -> 2[7.L]       -> 4[5,L]
        1[3]    -> 2[7.L]       -> 5[6,R]
        1[3]    -> 3[10.R]      -> 6[3,L]
        1[3]    -> 3[10.R]      -> 7[6,R]

        leftChildIdx    = parentIdx * 2
        rightChildIdx   = parentIdx * 2 + 1
*/

void dfs(int _idx)
{
        int parentIdx = _idx;
        int rChildIdx = parentIdx * 2;          // left
        int lChildIdx = parentIdx * 2 + 1;      // right
        if (parentIdx < nodeCnt + 1)
        {
                int val = arr[parentIdx];
                path[pathIdx] = val;
                if (val % 2 == 0)
                {
                        for (int i = 0; i < pathIdx + 1; ++i)
                                cout << path[i] << " ";
                        cout << endl;
                        path[pathIdx] = '\0';
                        --pathIdx;
                        return;
                }
                ++pathIdx;
                dfs(rChildIdx);
                ++pathIdx;
```

```
                dfs(lChildIdx);
                path[pathIdx] = '\0';
                --pathIdx;
        }
        else
        {
                --pathIdx;
                return;
        }
}

int main(void)
{
        // 이진트리 -> 1차원 배열
        // 1차원 배열의 1번 인덱스부터 값을 넣어준다.
        for (int i = 0; i < nodeCnt + 1; ++i)
                cin >> arr[i];

        dfs(startArrIdx);

        //_CrtDumpMemoryLeaks();
        return 0;
}
```