**1.**

축구 결승전에서 승부차기를 하게 되었습니다.
**oxox** 이면 첫번째/세번째 사람이 공을 넣은 것입니다.
**승부차기 할 사람수를 입력 받고, 가능한 경우수를 출력** 하세요.
(재귀호출로 구현해주세요)

ex)
[입력]
3
[출력]
ooo
oox
oxo
oxx
xoo
xox
xxo
xxx

# 입력 예제

```
4
```

# 출력 결과

```
oooo
ooox
ooxo
ooxx
oxoo
oxox
oxxo
```

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

int people = 0;
int* result = nullptr;
const int goalCase = 2;

void printResult()
{
	for (int i = 1; i < people + 1; ++i)
	{
		if (result[i] == 0)
			cout << 'o';
		else
			cout << 'x';
	}
	cout << endl;
}

void dfs(int _level, int _goal)
{
	if (_level == people + 1)
	{
		printResult();
		return;
	}

	for (int i = 0; i < goalCase; ++i)
	{
		if (result[0] == 1)
			break;
		result[_level] = i;
		dfs(_level + 1, i);
		result[_level] = 3;
	}
}
void init()
{
	cin >> people;
	result = new int[people + 1];
	for (int i = 0; i < people + 1; ++i)
		result[i] = 3;
}

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

```
        dfs(0, 0);

        return 0;
}
```

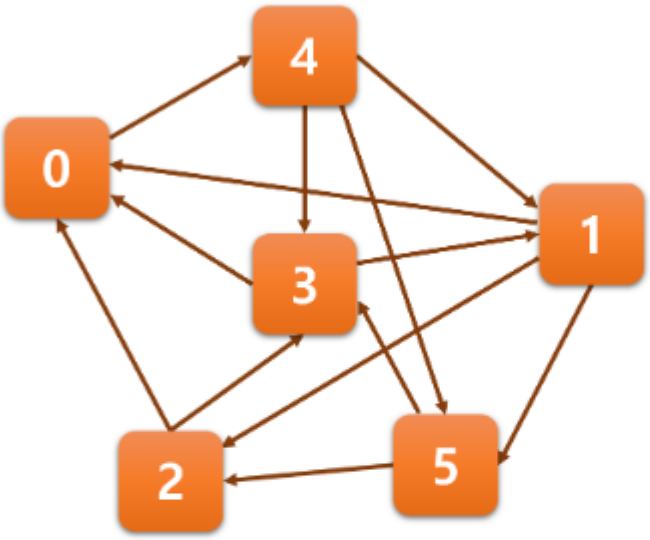
**2.**

아래에 그래프가 놓여져 있습니다.

입력 받은 출발 지점에서 너비 우선 탐색법으로 그래프를 탐색해주세요.
한번 방문했던 노드는 다시 방문할 수 없습니다.

시작 지점부터 BFS가 끝날때까지 방문한 노드를 출력해주세요.

[그래프]




[하드코딩용 테이블]

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | | | | | 1 | |
| 1 | 1 | | 1 | | | 1 |
| 2 | 1 | | | 1 | | |
| 3 | 1 | 1 | | | | |
| 4 | | 1 | | 1 | | 1 |
| 5 | | | 1 | 1 | | |

## 입력 예제

```
0
```

## 출력 결과

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

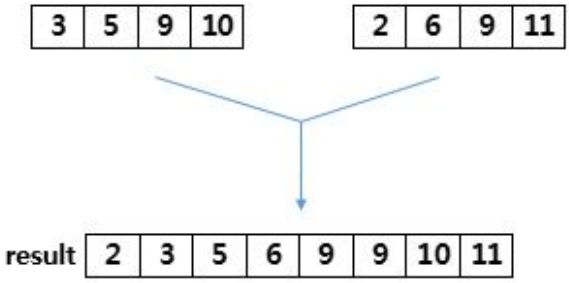
```cpp
#include <iostream>
#include <queue>
using namespace std;

const int g_size = 6;
int matrix[g_size][g_size] =
{
	//	0 1 2 3 4 5
		0,0,0,0,1,0,
		1,0,1,0,0,1,
		1,0,0,1,0,0,
		1,1,0,0,0,0,
		0,1,0,1,0,1,
		0,0,1,1,0,0,
};
queue<int> que;
int isVisit[g_size] = {};

void bfs()
{

	while(que.empty() == false)
	{
		int node = que.front();
		for (int i = 0; i < g_size; ++i)
		{
			if (matrix[node][i] == 1 && isVisit[i] == 0)
			{
				que.push(i);
				isVisit[i] = 1;
			}
		}
		cout << node << endl;
		que.pop();
	}
}

int main(void)
{
	int startPoint = 0;
	cin >> startPoint;
	isVisit[startPoint] = 1;
	que.push(startPoint);
	bfs();

	return 0;
}
```

**3.**

숫자 4개씩 2개의 배열에 숫자를 입력 받아주세요.

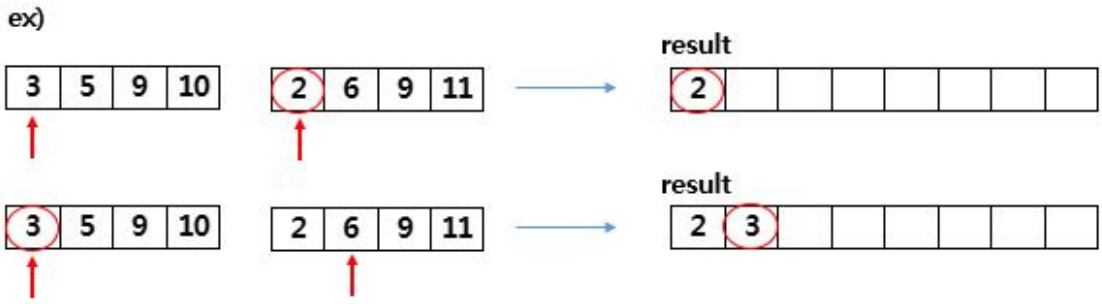

입력값은 정렬된 상태로 숫자가 들어옵니다.

이 두배열을 합쳐 정렬된 8개의 숫자를 저장 하려고 합니다.




이렇게 합치기 위한 알고리즘은

비교를 한 후에 작은 숫자를 result배열에 넣고 화살표를 옆으로 옮김니다.



위와 같은 동작을 반복하면, 정렬된 result 배열을 만들 수 있습니다.

위 알고리즘대로 코딩하여 result 배열을 만들고 출력 해주세요.

---

## 입력 예제

```
3 5 9 10
2 6 9 11
```

## 출력 결과

```
2 3 5 6 9 9 10 11
```

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

const int g_len = 4;
int arr1[g_len] = {};
int arr2[g_len] = {};
int result[g_len * 2] = {};

void input(int* arr)
{
	for (int i = 0; i < g_len; ++i)
		cin >> arr[i];
}

void sortArr()
{
	for (int i = 0; i < g_len; ++i)
	{
		int num1 = arr1[i];
		int num2 = arr2[i];
		int idx = i * 2;
		if (num1 > num2)
		{
			result[idx] = num2;
			result[idx + 1] = num1;
		}
		else
		{
			result[idx] = num1;
			result[idx + 1] = num2;
		}
	}
}

void print()
{
	for (int i = 0; i < g_len * 2; ++i)
		cout << result[i] << " ";
}

// 입력 값 : 3 5 9 10 2 6 9 11
int main(void)
{
	input(arr1);
	input(arr2);
	sortArr();
	print();
```

```
        return 0;
}
```

**4.**

**4x4 배열**에 **병정개미 4마리를 입력하여 배치 합니다.**

**숫자 1이 병정개미**라고 했을 때

만약, 아래와 같이 입력하면 **붙어있는 개미가 있어 위험한 상태** 입니다.

**1 0 0 0**

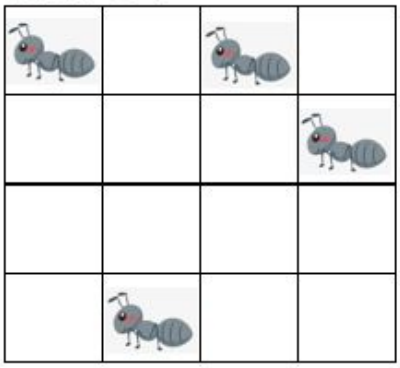
**0 1 1 0**

**0 0 0 1**

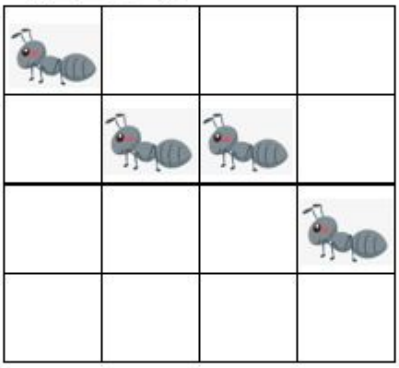
**0 0 0 0**

**Ex)**

안전한 상태



위험한 상태



**4x4 배열에 병정개미 4마리를 배치한 값을 입력받고,**

개미의 상태가 안전하면 **"안전한 상태"**,

안전하지 않으면 **"위험한 상태"**라고 판단하여 출력 해주는 프로그램을 만들어 주세요.

## 입력 예제

```
1 0 0 0
0 1 1 0
0 0 0 1
0 0 0 0
```

## 출력 결과

```
위험한 상태
```

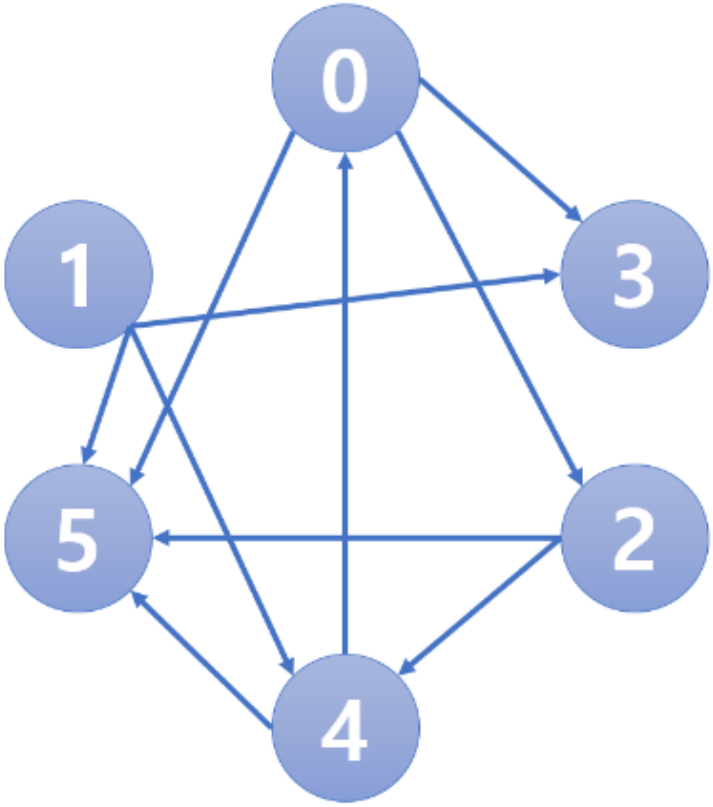
```cpp
#include <iostream>
using namespace std;

const int mapSize = 4;
int map[mapSize][mapSize] = {};

void input()
{
	for (int y = 0; y < mapSize; ++y)
	{
		for (int x = 0; x < mapSize; ++x)
			cin >> map[y][x];
	}
}

const int checkPoint = 4;
const int directionCnt = 2;
int direction[checkPoint][directionCnt] =
{
	-1,0,	// top
	0,1,	// right
	1,0,	// bottom
	-1,0,	// left
};

const bool danger = false;
const bool save = true;
bool mapCodition = save;

bool isSafe(int y, int x)
{
	for (int i = 0; i < checkPoint; ++i)
	{
		int c_Y = y + direction[i][0];
		int c_X = x + direction[i][1];
		if (map[c_Y][c_X] == 1)
			return danger;
	}
	return save;
}

void checkCodition()
{
	for (int y = 0; y < mapSize; ++y)
	{
		for (int x = 0; x < mapSize; ++x)
		{
```

```
                        if (map[y][x] == 1)      // 해당 좌표에 개미가 존재할 때
                        {
                                mapCodition = isSafe(y, x);
                                if (mapCodition == danger)
                                        break;
                        }
                }
                if (mapCodition == danger)
                        break;
        }
}

void printMapCondition()
{
        if (mapCodition == save)
                cout << "안전한 상태";
        else
                cout << "위험한 상태";
}

/*
        입력값1 :
        1 0 0 0
        0 1 1 0
        0 0 0 1
        0 0 0 0

        입력값2 :
        1 0 0 0
        0 1 0 0
        0 0 1 0
        0 0 0 1
*/
int main(void)
{
        input();
        checkCodition();
        printMapCondition();

        return 0;
}
```

**5.**

인접 행렬 그래프를 깊이 우선 탐색법으로 탐색 해봅시다.

[그래프]




그래프를 나타내는 인접 행렬을 하드코딩 하세요.

출발지점의 노드 값을 입력 받으세요.

입력 받는 노드 부터 탐색을 시작할때,
DFS로 노드들을 방문할 때마다 노드의 값을 출력해 주세요.

한 노드에 여러 노드로 갈 수 있다면,
숫자가 작은 노드부터 탐색 해주세요.

Ex) 0에서 세개의 노드로 갈 수 있는데, 작은 숫자부터 차례로 2, 3, 5 순으로 탐색이 진행되어야 합니다.

**[하드코딩 용 테이블]**

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | | | 1 | 1 | | 1 |
| 1 | | | | 1 | 1 | 1 |
| 2 | | | | | 1 | 1 |
| 3 | | | | | | |
| 4 | 1 | | | | | 1 |
| 5 | | | | | | |

# 입력 예제

```
0
```

# 출력 결과

```
0 2 4 5 3
```

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

const int nodeCnt = 6;
int matrix[nodeCnt][nodeCnt] =
{
//      0 1 2 3 4 5
        0,0,1,1,0,1,
        0,0,0,1,1,1,
        0,0,0,0,1,1,
        0,0,0,0,0,0,
        1,0,0,0,0,1,
        0,0,0,0,0,0,
};
int isVisit[nodeCnt] = {};
int path[nodeCnt] = {};
int level = 0;

void dfs(int _node)
{
        if (level == nodeCnt)
                return;

        for (int i = 0; i < nodeCnt; ++i)
        {
                if (matrix[_node][i] == 1 && isVisit[i] == 0)
                {
                        path[++level] = i;
                        isVisit[i] = 1;
                        dfs(i);
                }
        }
}

void init()
{
        int startNode = 0;
        cin >> startNode;
        isVisit[startNode] = 1;
        path[level] = startNode;
        dfs(startNode);
}

void printPath()
{
        for (int i = 0; i < level+1; ++i)
                cout << path[i] << " ";
```

```
}

int main(void)
{
        init();
        printPath();

        return 0;
}
```