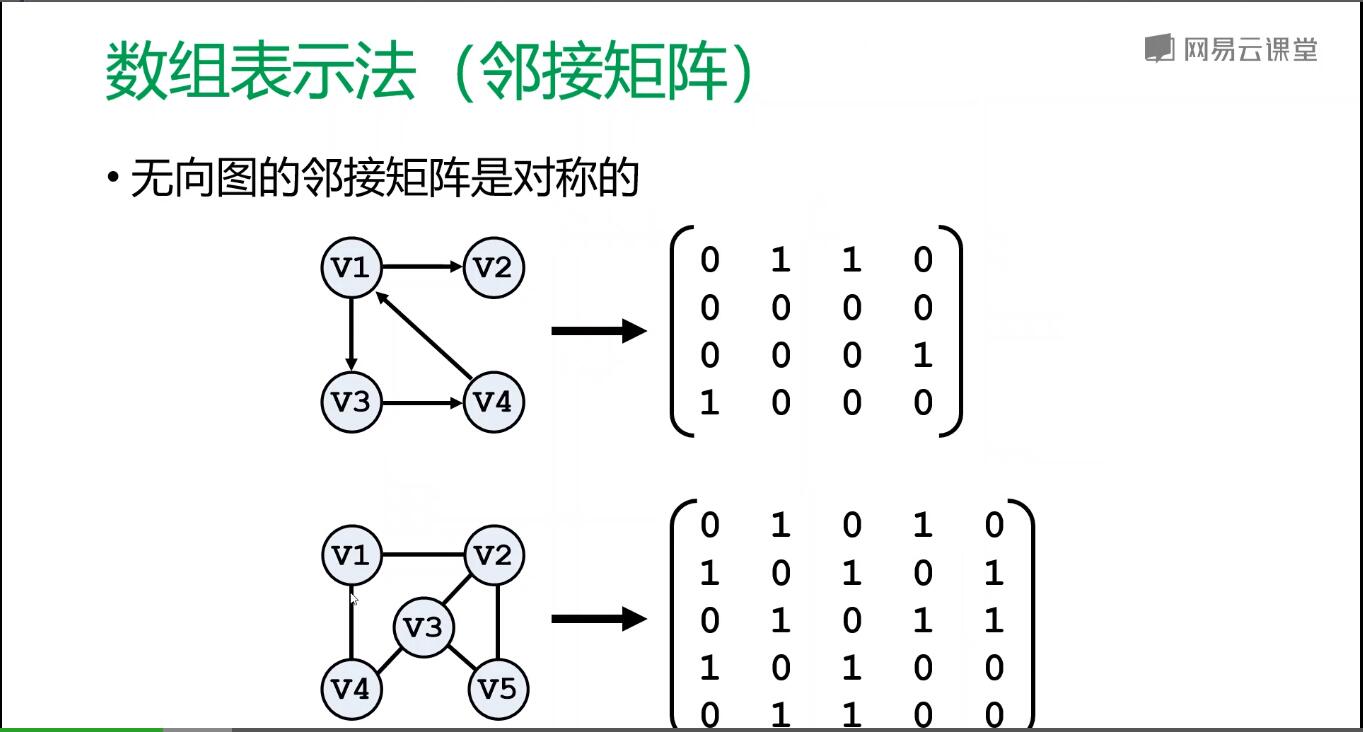
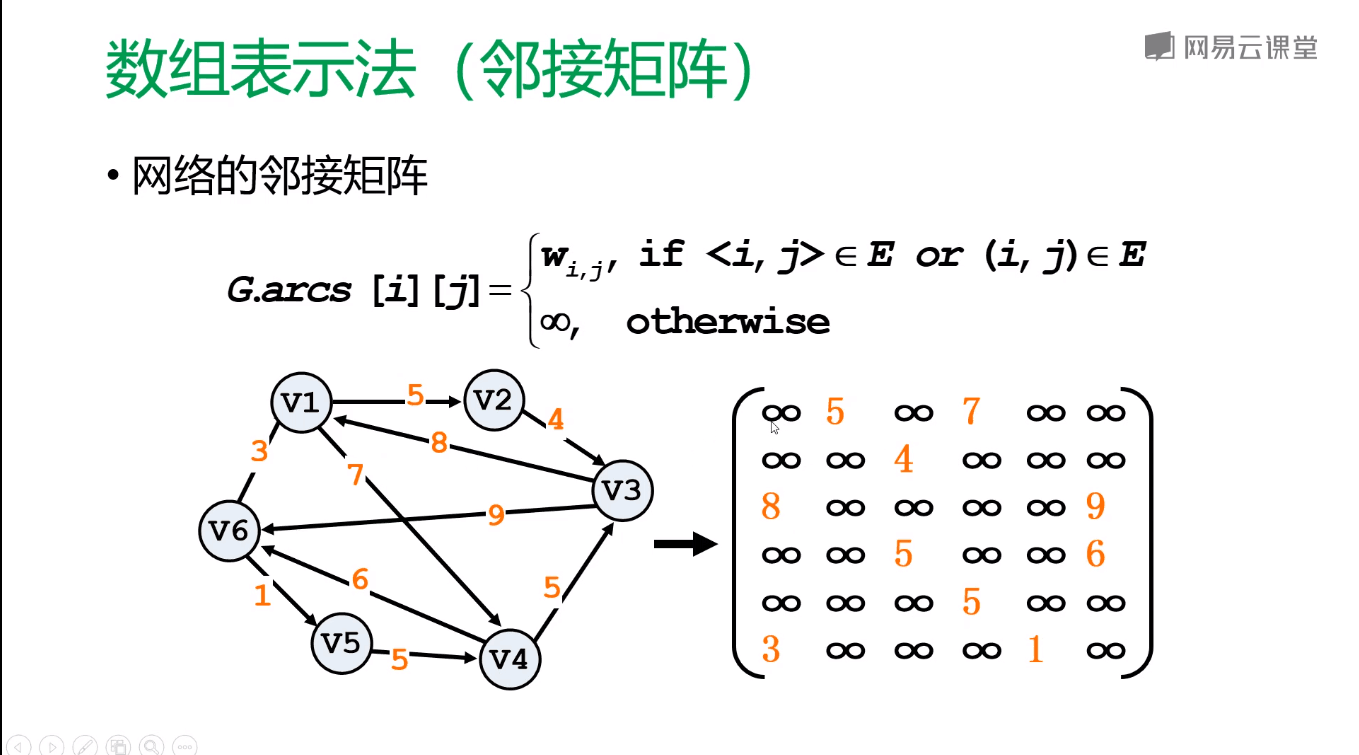
**图的邻接矩阵表示**



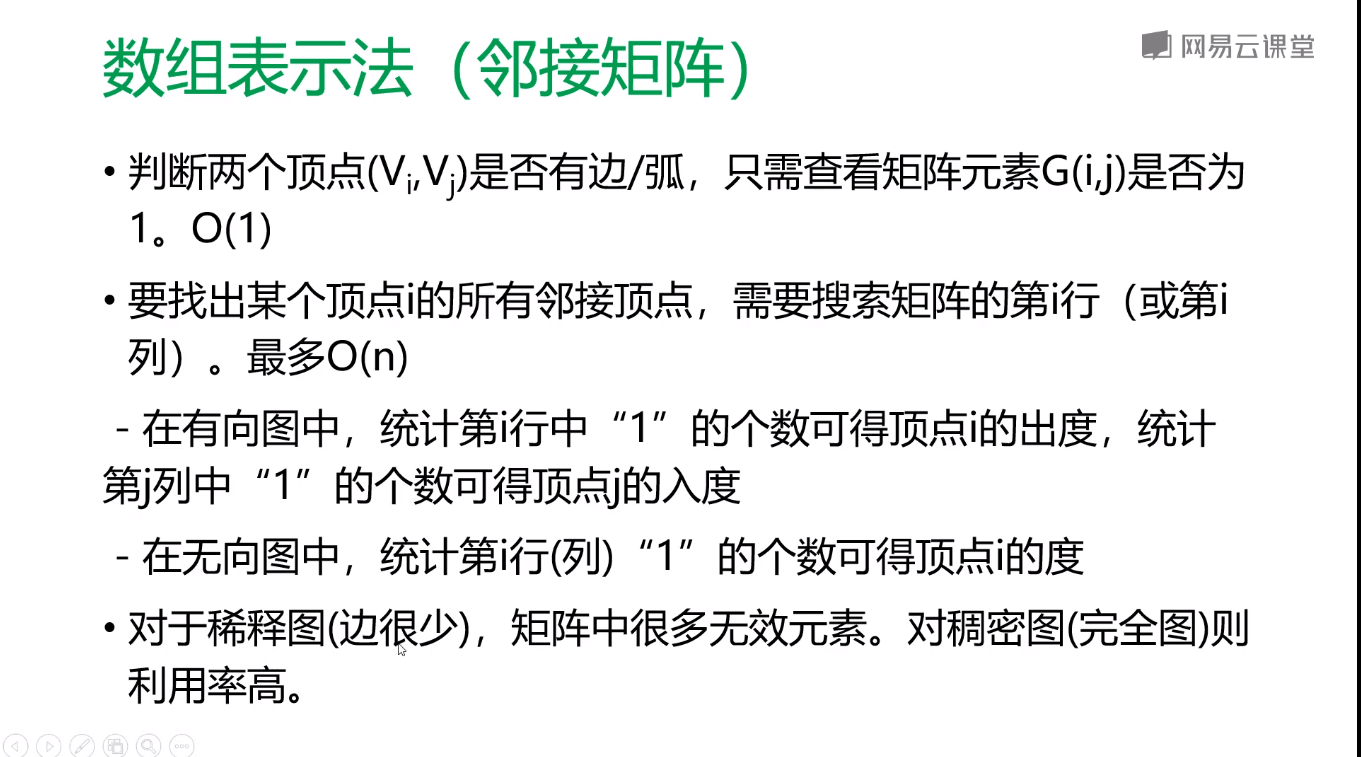
1. 我们可以表示有向图和无向图，就是有相连，我们把值设为1，不想连我们呢可以设为0或着一个取不到的值
2. 例如下面



A·如果要求V1到V2是否右边

B·那么我们就需要遍历整个边表，得到0-1是个否有

1. 邻接矩阵的情况



A·缺点就是如果是稀疏图，我们无效的值太多，浪费太多空间了

1. 代码实现

// 无向图的灵界矩阵.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include <iostream>

#include <vector>

#include <stack>

using namespace std;

template <typename T>

class MGraph {

vector<T> vertices;

vector<vector<int>> edges;

vector<bool> flag;

bool undirected{ true };

int inv = -1;

public:

MGraph(bool undirected = true) :undirected(undirected) {};

void add\_vertice(const T& v) {

vertices.push\_back(v);

}

//初始化二位数组，如果输出4个的字母，则生成4x4二维数组

void clear\_edges(const int value) {

if (vertices.size() != edges.size()) {

const int vn = vertices.size();

edges.resize(vn, vector<int>(vn, value));

}

}

void add\_edges(int x, int y, int value) {

if (vertices.size() != edges.size()) {

throw "先生成二位数组";

}

edges[x][y] = value;

}

void print\_matrix() {

for (auto& row : edges) {

for (auto& col : row) {

cout << col << " ";

}

cout << endl;

}

}

void print() {

cout << "顶点信息：" << endl;

for (auto& e : vertices) {

cout << e << " ";

}

cout << endl;

print\_matrix();

}

};

int main()

{

MGraph<char> G;

G.add\_vertice('A');

G.add\_vertice('B');

G.add\_vertice('C');

G.add\_vertice('D');

G.clear\_edges(-1);

G.add\_edges(0, 0, 1);

G.add\_edges(0, 1, 1);

G.add\_edges(0, 3, 2);

G.print();

}