图的两种表示方法

- 邻接矩阵与邻接表
  - 图论中绝大多数算法用邻接表的效率高
    - BFS、DFS、Dijkstra、SPFA
    - 例外：Floyd-Warshall算法一般基于邻接矩阵
    - 朴素的Bellman-Ford算法只用一个包含所有边的大数组
  - 有时候输入数据的格式决定了哪种表示法更方便
    - 网络流图中有平行边，需要累积流量，而这些平行边在输入数据中的位置是随意的，则要用邻接矩阵
  - 图的规模比较小时，可以用邻接矩阵来模拟邻接表
    - 对稀疏图来说并不省内存，但是可以达到邻接表的效率

邻接表和邻接矩阵

- 邻接表：图中有n个节点，每个节点的出边用单独一行给出
  - vector<vector<int> > v(n);
  - 第i个顶点有边指向第j个顶点：v[i].push_back(j);
  - 二维数组中第一维是预先分配好的，第二维是动态增加的
- 邻接矩阵
  - vector<vector<int> > v(n, vector<int>(n));
  - 第i个顶点有边指向第j个顶点：v[i][j] = 1;
- 注意：
  - 早期版本的编译器要求> >之间必须有空格
  - 如果顶点编号是从1到n，那么分配内存时把n改为n+1

为什么不用new来分配内存？

- 用new很难分配出同时满足以下条件的邻接矩阵空间
  - 矩阵的维度是运行时间决定的
    - int (*adj)[n] = new int[m][n];要求n在编译时是常量
  - 能够用下标操作符adj[i][j]访问矩阵中元素
    - int *adj = new int[m*n];需要把二维下标转换成一维
  - 用单个new语句而非循环完成（内存碎片）
    - 先分配一个指针数组，再给每个指针分配行空间
  - 有些高级技巧可以绕过以上限制，但是远不如使用vector方便
- 用new分配的内存需要用delete释放，而且无法初始化（除非只分配单个元素）