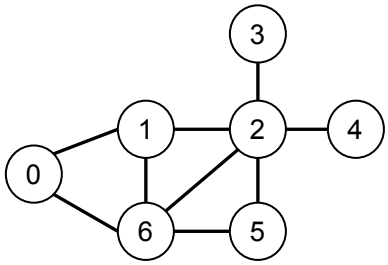
1. **グラフ構造**とは，右図のようにノード（または頂点）がリンク（またはエッジ，辺）によって連結された構造であり，木もグラフの一種である。
　数学ではグラフの連結の表現として，**隣接行列**が用いられる。これは，グラフGにおいて行列Aを定義し，Aの成分a_{ij}の値をノードv_iとv_jを直接つなぐリンクの本数とするものであり，プログラム言語では 2 次元配列で実現できる。グラフ構造のデータ表現には，他にも，木構造のように構造体とポインタを用いる方法などがある（教科書参照）。



　下記のプログラムは，右図のグラフを隣接行列で表し，あるノードから他のノードへの最短距離とその経路を**ダイクストラ法**というアルゴリズムによって求めるものである。空欄を適切に埋めてプログラムを入力して実行し，説明を読んでアルゴリズムについて理解せよ。

```
#include <stdio.h>

#define N 7       /* グラフのノード数 */
#define INF 9999  /* 距離無限大（＝非接続） */

/* 隣接行列：ノード i と j が直接つながっていれば
   adj[i][j]==1、そうでなければ 0 */
int adj[N][N] = {
    {    ,    ,    ,    ,    ,    ,    },
    {    ,    ,    ,    ,    ,    ,    },
    {    ,    ,    ,    ,    ,    ,    },
    {    ,    ,    ,    ,    ,    ,    },
    {    ,    ,    ,    ,    ,    ,    },
    {    ,    ,    ,    ,    ,    ,    },
    {    ,    ,    ,    ,    ,    ,    }};

int main(void)
{
    int start;   /* 始点ノード */
    int dist[N]; /* 各ノードまでの距離 */
    int back[N]; /* 1 つ前の経由ノード */
    int node, xnode;
    int min_dist;

    int Q[N];   /* 距離が未確定のノードの集合 */
    int nQ = N; /* Q の要素数（未確定ノード数）*/
    int count, min_elem;

    /* 初期化：全ノードへの距離を無限大に設定 */
    for (node = 0; node < N; node++) {
        dist[node] = INF;
        Q[node] = node; /* 未確定集合に登録 */
    }

    /* 始点ノードの入力 */
    printf("starting node (0-%d):", N-1);
    scanf("%d", &start);

    dist[start] = 0;   /* 始点までの距離は 0 */
    back[start] = -1;  /* 始点の 1 つ前はない */

    /* 未確定ノードがある間、繰り返す */
    while (nQ > 0) {
        /* Q（距離未確定）の中の全ノードを比較し，
           距離が最短のノードを選び出す */
        node = Q[0]; /* Q の最初（0）の要素 */
        min_dist = dist[node];
        min_elem = 0;
        for (count = 1; count < nQ; count++) {
            node = Q[count];
            if (dist[node] < min_dist) {
                min_dist = dist[node];
                min_elem = count;
            }
        }

        /* 到達できないノードが残った場合も終了 */
        if (min_dist == INF) break;

        /* 距離が最短だったノード（xnode）は，
           それより短い経路が存在し得ないので
           Q から取り除く（その距離が確定する） */
        xnode = Q[min_elem];
        Q[min_elem] = Q[--nQ];

        /* 未確定の全ノードについて，xnode を
           経由した距離を計算し，今までよりも
           短くなるなら距離と経路を更新する */
        for (count = 0; count < nQ; count++) {
            /* Q に含まれるノードを順にチェック */
            node = Q[count];
            if (adj[xnode][node] > 0) {
                /* xnode までの距離+1 のほうが，
                   node までよりも短ければ */

                if (dist[node] >                ) {

                    dist[node] =
                    back[node] = xnode;
                }
            }
        }
    }

    /* 結果表示 */
    for (node = 0; node < N; node++) {
        printf("%d->%d dist:%d path:",
               start, node, dist[node]);
        /* 経路を逆にたどって表示 */
        for (xnode = node;
             xnode != -1; xnode = back[xnode]) {
            printf("%d ", xnode);
        }
        putchar('¥n');
    }
    return 0;
}
```