#define MAX_NODES 1024  /* maximum number of nodes */
#define INFINITY 1000000000 /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES];  /* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{
    struct state { /* the path being worked on */
        int predecessor;  /* previous node */
        int length;      /* length from source to this node */
        enum {permanent, tentative} label; /* label state */
    } state[MAX_NODES];

    int i, k, min;
    struct state *p;

    for (p = &state[0]; p < &state[n]; p++) { /* initialize state */
        p->predecessor = -1;
        p->length = INFINITY;
        p->label = tentative;
    }

    state[t].length = 0; state[t].label = permanent;
    k = t;  /* k is the initial working node */
    do { /* Is there a better path from k? */
        for (i = 0; i < n; i++) /* this graph has n nodes */
            if (dist[k][i] != 0 && state[i].label == tentative) {
                if (state[k].length + dist[k][i] < state[i].length) {
                    state[i].predecessor = k;
                    state[i].length = state[k].length + dist[k][i];
                }
            }

        /* Find the tentatively labeled node with the smallest label. */
        k = 0; min = INFINITY;
        for (i = 0; i < n; i++)
            if (state[i].label == tentative && state[i].length < min) {
                min = state[i].length;
                k = i;
            }

        state[k].label = permanent;
    } while (k != s);

    /* Copy the path into the output array. */
    i = 0; k = s;
    do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);