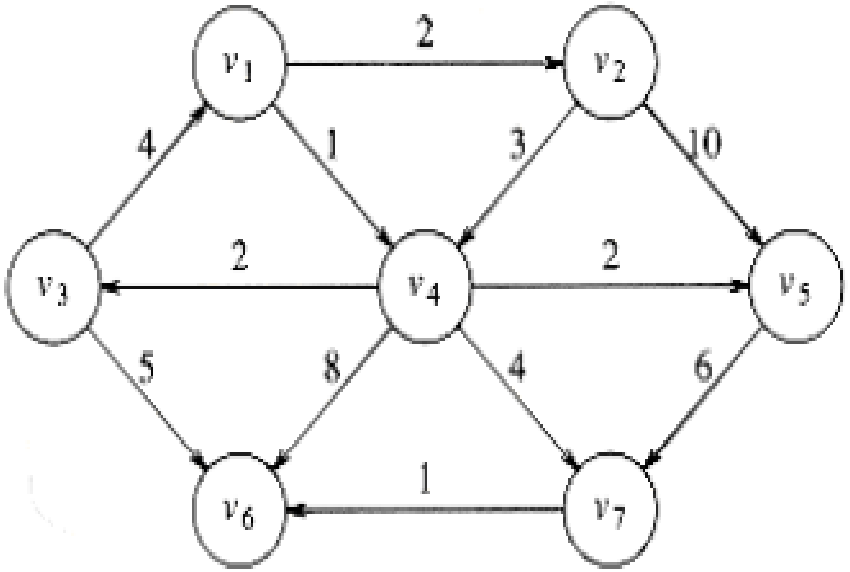
Graph (VI)

Bruce Nan

Shortest-Path Algorithms

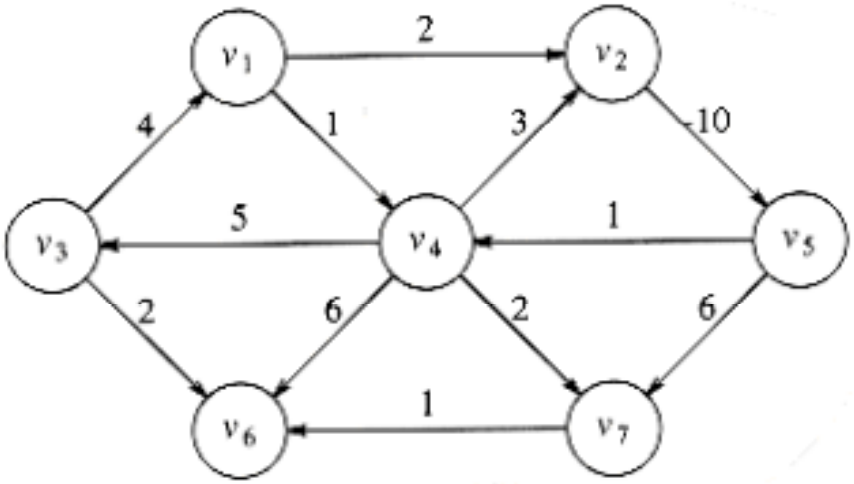
- The input is a weighted graph: associated with each edge (v_i, v_j) is a cost $c_{i,j}$ to traverse the arc. The cost of a path $v_1 v_2 \ldots v_n$ is sum $(c_{i,i+1})$.
- This is referred to as the weighted path length . The unweighted path length is merely the number of edges on the path, namely, n - 1.

- For example, in the right graph, the shortest weighted path from v1 to v6 has a cost of 6 and goes from v1 to v4 to v7 to v6.



Graph with Negative Weight

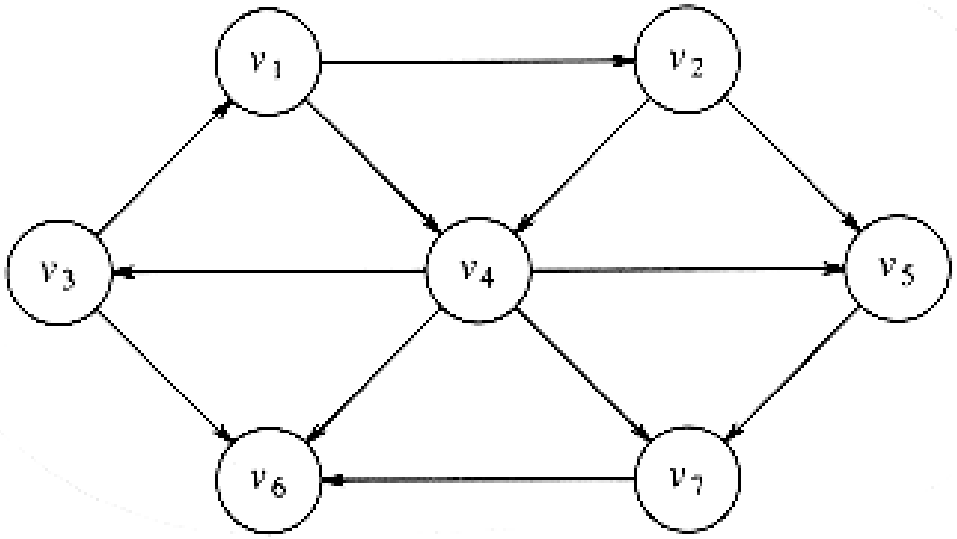
- Some problem will cause because of negative weight.
- The path from v5 to v4 has cost 1, but a shorter path exists by following the loop v5, v4, v2, v5, v4, which has cost -5. This path is still not the shortest, because we could stay in the loop arbitrarily long. Thus, the shortest path between these two points is undefined.



Unweighted Shortest Path

- Using some vertex, s, which is an input parameter, we would like to find the shortest path from s to all other vertices.
- We are only interested in the number of edges contained on the path, so there are no weights on the edges. This is clearly a special case of the weighted shortest-path problem, since we could assign all edges a weight of 1.

How to get the unweighted shortest path from V3 to other vertices?



Code for Unweighted Shortest Path

```
void unweighted( TABLE T )
{
    QUEUE Q;
    vertex v, w;
    Q = create_queue( NUM_VERTEX ); make_null( Q );
    enqueue( s, Q );
    while( !is empty( Q ) )
    {
        v = dequeue( Q );
        T[v].known = TRUE; /* not really needed anymore */
        for each w adjacent to v
            if( T[w].dist = INT_MAX )
            {
                T[w].dist = T[v].dist + 1;
                T[w].path = v;
                enqueue( w, Q );
            }
    }
dispose_queue( Q );
```

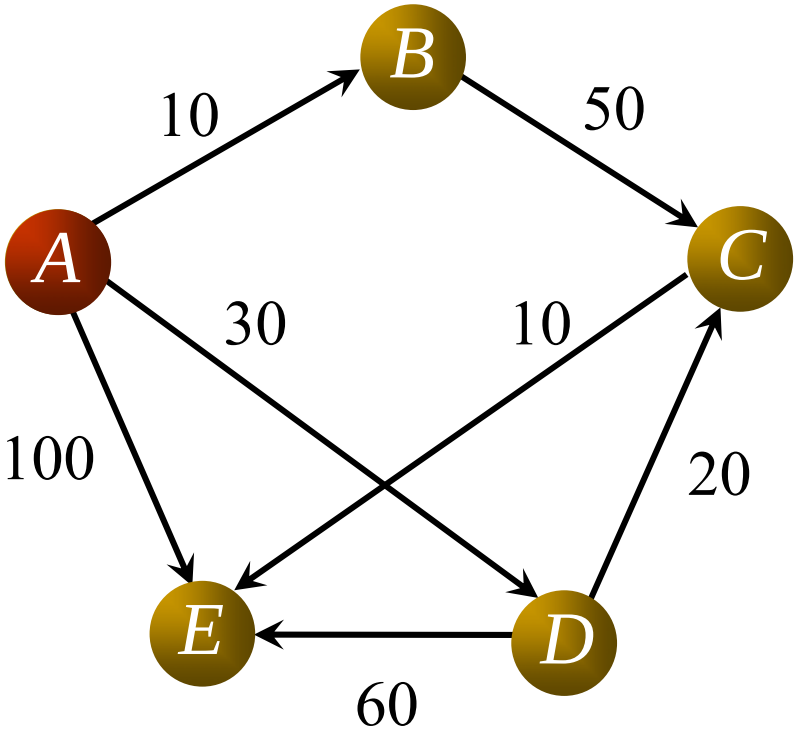
Dijkstra's Algorithm

- The general method to solve the single-source shortest-path problem is known as Dijkstra's algorithm . This solution is a prime example of a greedy algorithm.
- Greedy algorithms generally solve a problem in stages by doing what appears to be the best thing at each stage. For example, to make change in U.S. currency, most people count out the quarters first, then the dimes, nickels, and pennies. This greedy algorithm gives change using the minimum number of coins.

Dijkstra's Algorithm

- Dijkstra's algorithm proceeds in stages, just like the unweighted shortest-path algorithm. At each stage, Dijkstra's algorithm selects a vertex v, which has the smallest d_v among all the unknown vertices, and declares that the shortest path from s to v is known. The remainder of a stage consists of updating the values of d_w.

Dijkstra Algorithm



S={A}

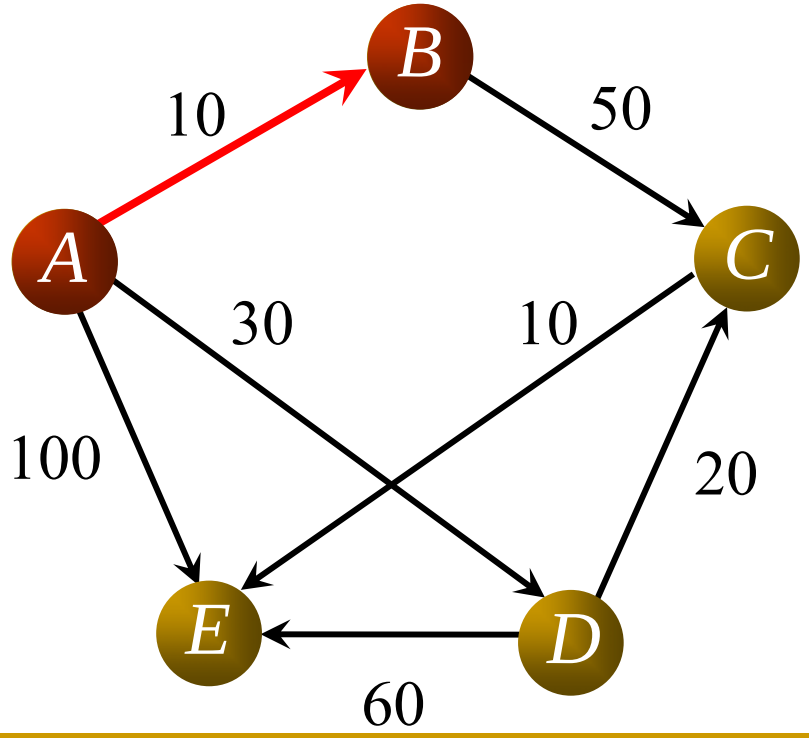
A→B:(A, B)10

A→C:(A, C)∞

A→D: (A, D)30

A→E: (A, E)100

Dijkstra Algorithm



S={A, B}

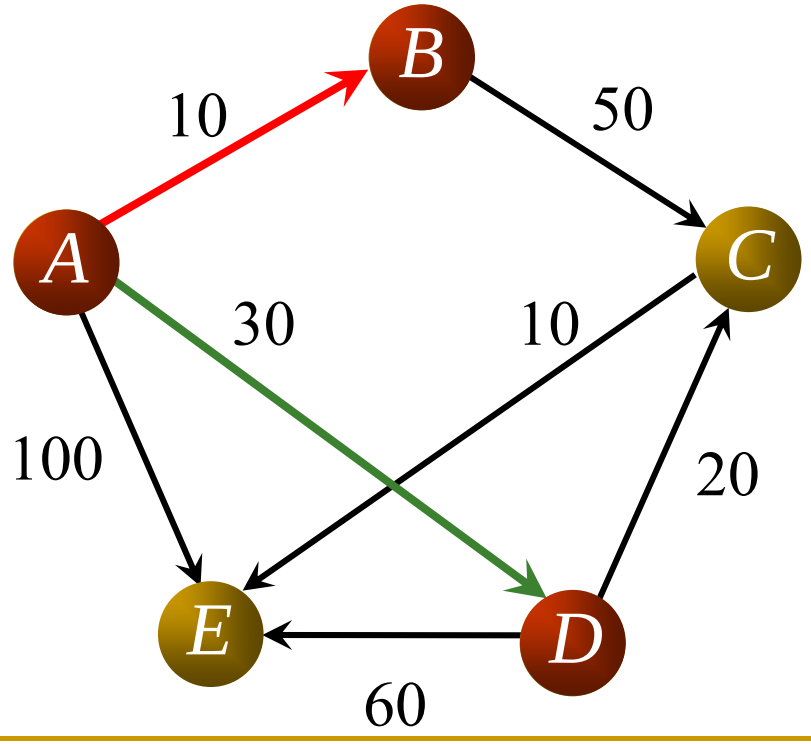
A→B:(A, B)10

A→C:(A, B, C)60

A→D: (A, D)30

A→E: (A, E)100

Dijkstra Algorithm



S={A, B, D}

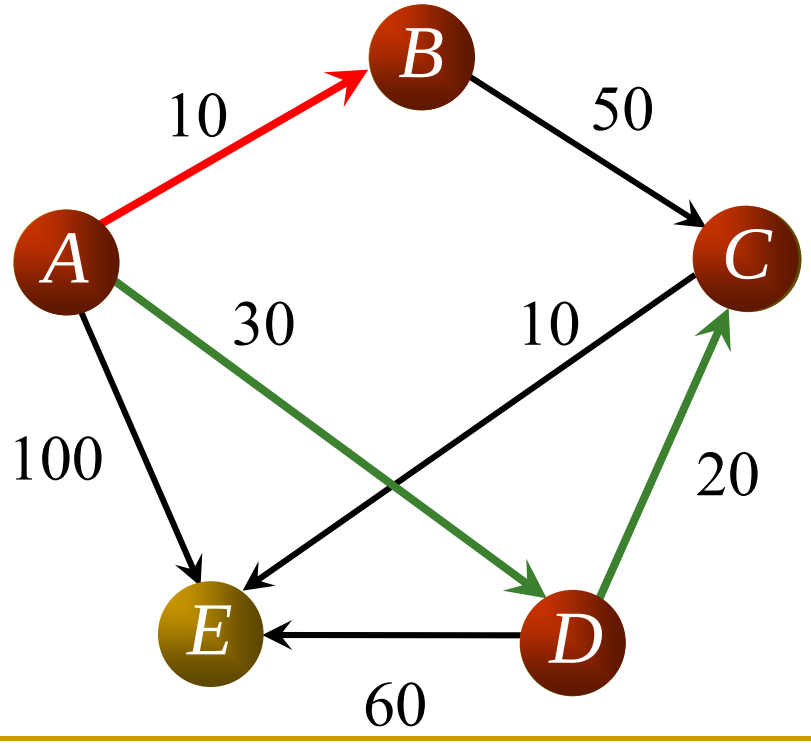
A→B:(A, B)10

A→C:(A, D, C)50

A→D: (A, D)30

A→E: (A, D, E)90

Dijkstra Algorithm



S={A, B, D, C}

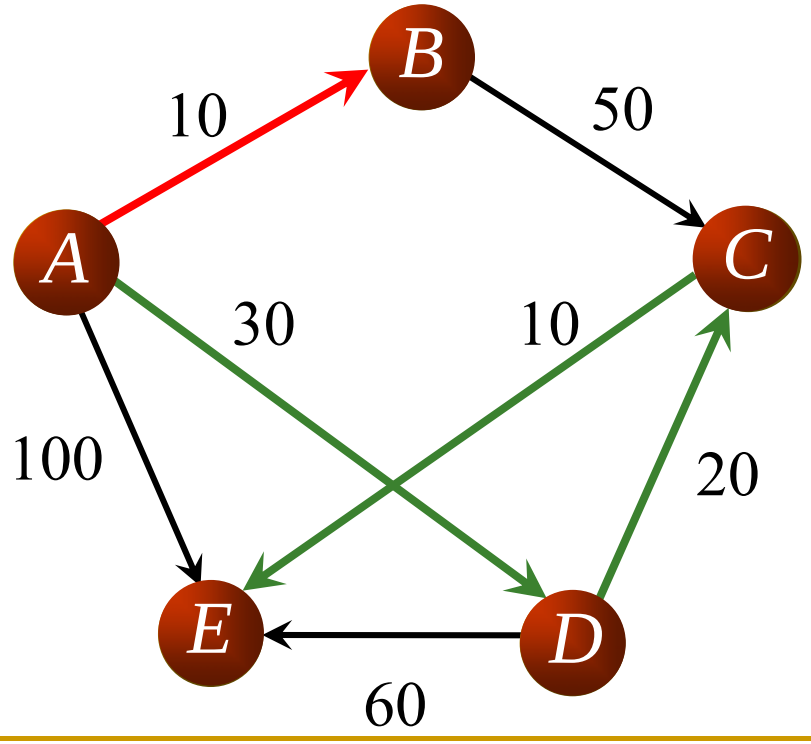
A→B:(A, B)10

A→C:(A, D, C)50

A→D: (A, D)30

A→E: (A, D, C, E)60

Dijkstra Algorithm



S={A, B, D, C, E}

A→B:(A, B)10

A→C:(A, D, C)50

A→D: (A, D)30

A→E: (A, D, C, E)60

Floyd Algorithm

- Many applications need to know the length of the shortest path between all pairs of vertices in a given graph.
- Calling Dijkstra's algorithm from each of the n possible starting vertices.
- Or using Floyd's all-pairs shortest-path algorithm.

- Very easy to implement!!!

Floyd Algorithm

```
1 /* Assume a function edgeCost(i,j) which returns the cost of the edge from i to j
2    (infinity if there is none).
3    Also assume that n is the number of vertices and edgeCost(i,i) = 0
4 */
5
6 int path[][];
7 /* A 2-dimensional matrix. At each step in the algorithm, path[i][j] is the shortest
     path
8    from i to j using intermediate vertices (1..k−1).  Each path[i][j] is initialized to
9    edgeCost(i,j).
10 */
11
12 procedure FloydWarshall ()
13    for k := 1 to n
14       for i := 1 to n
15          for j := 1 to n
16             path[i][j] = min ( path[i][j], path[i][k]+path[k][j] );
```

Floyd's algorithm

- Floyd's algorithm is best employed on an adjacency matrix data structure, which is no extravagance since we have to store all n2 pairwise distances anyway

```
typedef struct {
    int weight[MAXV+1][MAXV+1];    /* adjacency/weight info */
    int nvertices;      /* number of vertices in graph */
} adjacency_matrix;
```

Thank You