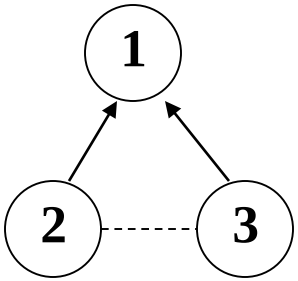
4.1.2 Kruskal's Algorithm

- **Checking a cycle (using merge and find)**



select (v_1, v_2)

select (v_1, v_3)

add (v_2, v_3)?

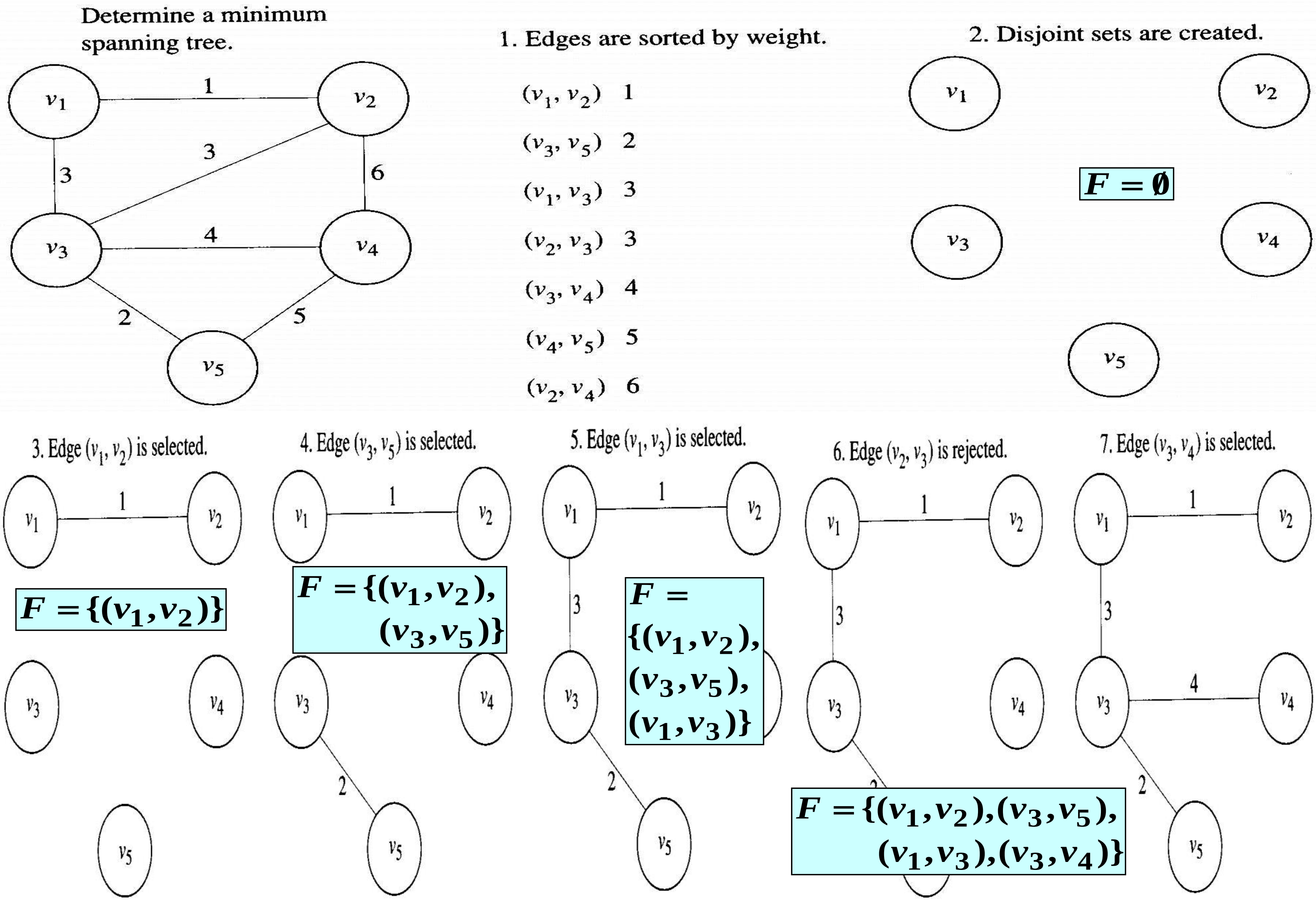
(find(2) = 1) = (find(3) = 1) → cycle

(v_2, v_3) is rejected

High-level algorithm

```
F = ∅;            // Initialize set of edges to empty.
create disjoint subsets of V, one for each vertex
   and containing only that vertex;
sort the edges in E in nondecreasing order;
while (the instance is not solved) {
   select next edge;                          // selection procedure
   if (the edge connects two vertices in disjoint subsets){
                                              // feasibility check
      merge the subsets;
      add the edge to F;
   }
   if (all the subsets are merged)            // solution check
      the instance is solved;
}
```

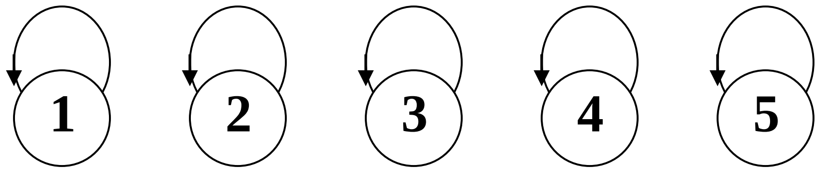
Figure 4.7 A weighted graph (in upper left corner) and the steps in Kruskal's Algorithm for that graph.



merge and find operation (1/2)

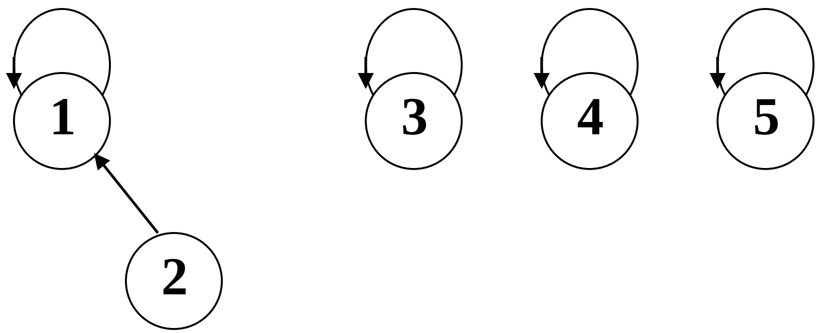
- **1. Initial**

  $\boldsymbol{F = \emptyset}$



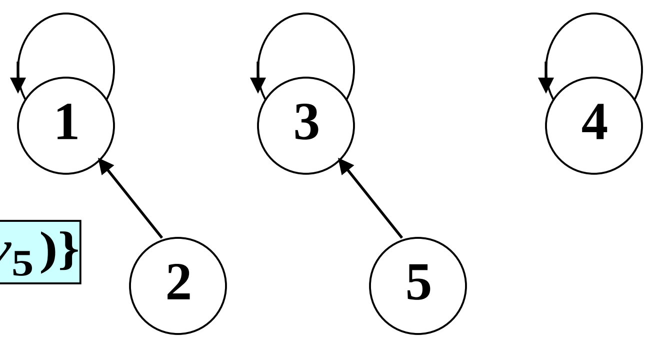
- **2. (v_1, v_2) selected**

  $\boldsymbol{F = \{(v_1, v_2)\}}$



- **3. (v_3, v_5) selected**

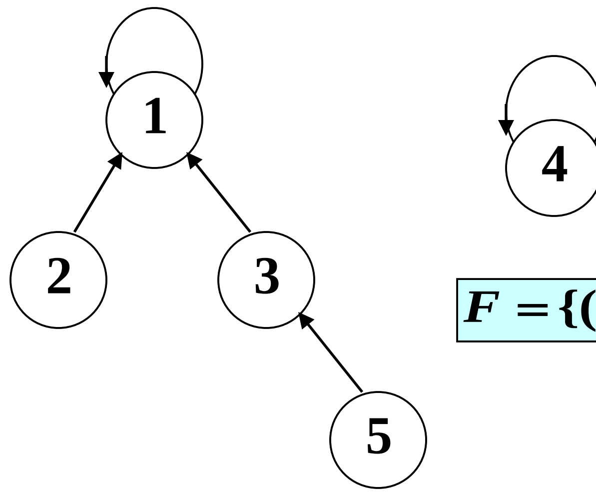
  $\boldsymbol{F = \{(v_1, v_2), (v_3, v_5)\}}$



$$
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{bmatrix}
0 & 1 & 3 & \infty & \infty \\
1 & 0 & 3 & 6 & \infty \\
3 & 3 & 0 & 4 & 2 \\
\infty & 6 & 4 & 0 & 5 \\
\infty & \infty & 2 & 5 & 0
\end{bmatrix}
\end{array}
$$

merge and find operation (2/2)

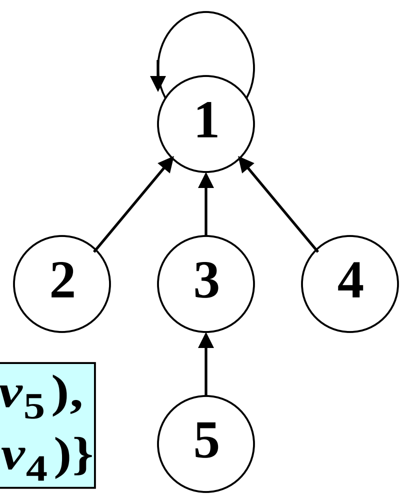
- **4. (v_1, v_3) selected**



$$F = \{(v_1, v_2), (v_3, v_5), (v_1, v_3)\}$$

- **5. (v_2, v_3) rejected**

- **6. (v_3, v_4) selected**



$$F = \{(v_1, v_2), (v_3, v_5), (v_1, v_3), (v_3, v_4)\}$$

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 3 & \infty & \infty \\ 1 & 0 & 3 & 6 & \infty \\ 3 & 3 & 0 & 4 & 2 \\ \infty & 6 & 4 & 0 & 5 \\ \infty & \infty & 2 & 5 & 0 \end{bmatrix} \end{array}$$

Algorithm 4.2 Kruskal's Algorithm (1/2)

- **Problem: Determine an MST.**
- **Inputs: integer $n \geq 2$, positive integer m, and a connected, weighted, undirected graph containing n vertices and m edges. The graph is represented by a set E that contains the edges in the graph along with their weights.**
- **Outputs: F, a set of edges in an MST.**

Algorithm 4.2 Kruskal's Algorithm (2/2)

```
void kruskal (int n, int m, set_of_edges E, set_of_edges& F)
{
    index i, j;       set_pointer p, q;       edge e;
    Sort the m edges in E by weight in nondecreasing order;
    F = Ø ;
    initial(n);                               // Initialize n disjoint subsets.
    while (number of edges in F is less than n - 1) {
        e = edge with least weight not yet considered;
        i, j = indices of vertices connected by e;
        p = find(i);          q = find(j);
        if (! equal(p, q)) {
            merge(p, q);          add e to F;
        }
    }
}
```

T(n) of Algorithm 4.2

- **Basic operation: comparison**
- **Input size:** $n = |V|,\ m = |E|$

- **1. Sort edges** $\Theta(m \log m)$
- **2. While loop** $\Theta(m \log m)$
- **3. Initialize n sets** $\Theta(n)$

$$\left.\begin{array}{l}\Theta(m\log m)\\ \Theta(m\log m)\\ \Theta(n)\end{array}\right\}\Theta(m \log m)$$

- **Since** $n-1 \le m \le n(n-1)/2$
- $$m \log m = \begin{cases} n^2 \log n^2 = n^2 \log n & \Theta(n^2 \log n) \\ (n-1)\log(n-1) \approx n \log n & \Theta(n \log n) \end{cases}$$

$cf.$ Prim's algorithm $\Theta(n^2)$

- **Correctness proof: similarly as Lemma 4.1 and Theorem 4.1**

4.2 Dijkstra's Algorithm for Single-Source Shortest Paths

Problem: Determine the shortest paths from v_1 to all other vertices in a weighted, directed graph.

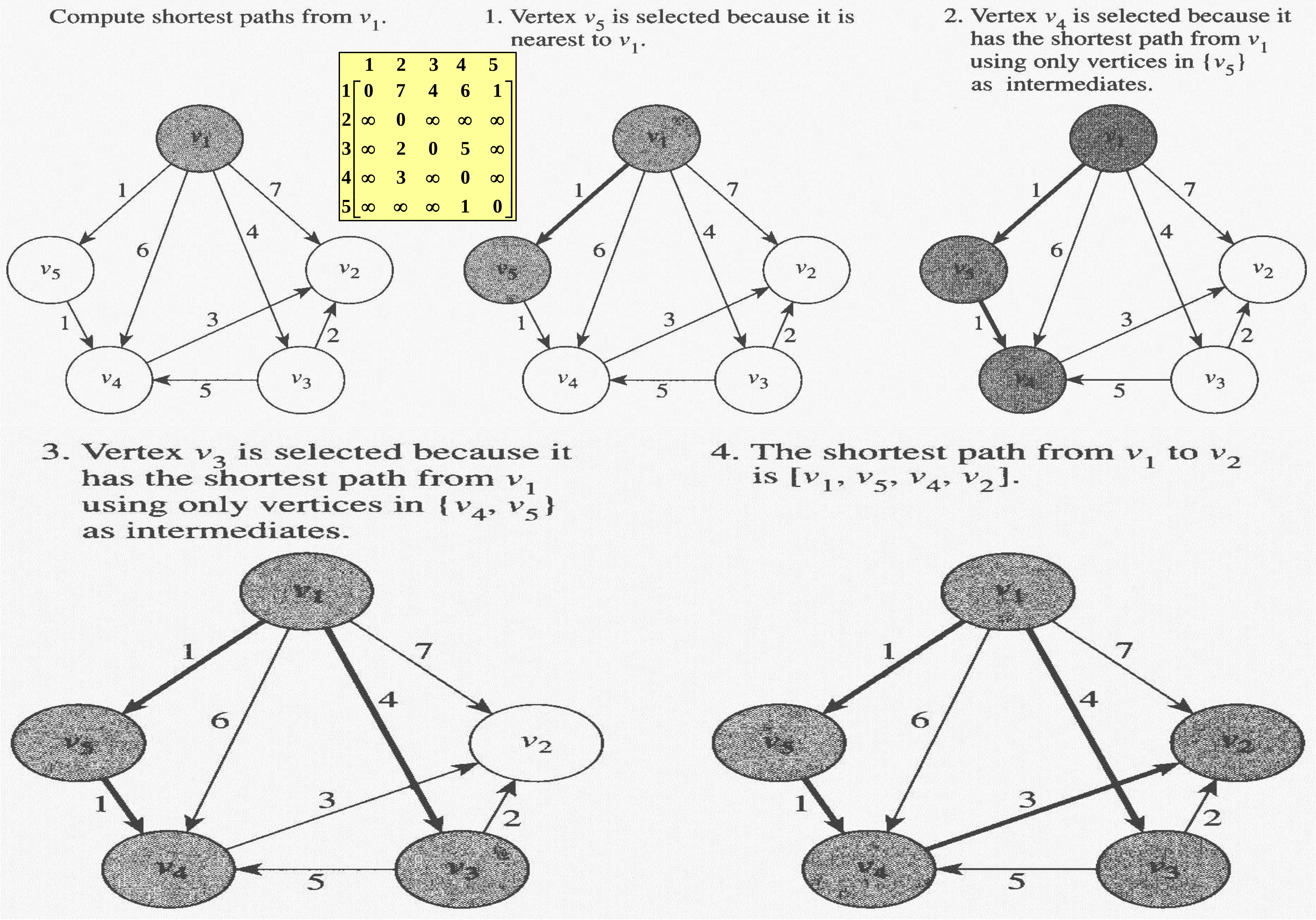
- **High-level algorithm**

$Y = \{v_1\}$;

$F = \emptyset$;

```
while (the instance is not solved) {
    select a vertex v in V – Y,                   // selection procedure
        that has a shortest path from v1,         // and feasibility check
        using only vertices in Y as intermediates;
    add the new vertex v to Y;
    add the edge (on the shortest path) that touches v to F;
    if (Y == V)                                   // solution check
        the instance is solved;
}
```

Figure 4.8 A weighted, directed graph (in upper left corner) and the steps in Dijkstra's Algorithm for that graph. The vertices in Y and the edges in F are shaded at each step.



$$
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{bmatrix}
0 & 7 & 4 & 6 & 1 \\
\infty & 0 & \infty & \infty & \infty \\
\infty & 2 & 0 & 5 & \infty \\
\infty & 3 & \infty & 0 & \infty \\
\infty & \infty & \infty & 1 & 0
\end{bmatrix}
\end{array}
$$

Variables for Dijkstra's algorithm

- ***touch*[*i*] = index of the vertex *v* in *Y* s.t. < *v*, v_i > is the last edge on the current shortest path from v_1 to v_i using only vertices in *Y* as intermediates. (cf.: *nearest*[*i*] in Prim's)**
- ***length* [*i*] = length of the current shortest path from v_1 to v_i using only vertices in *Y* as intermediates. (cf.: *distance*[*i*] in Prim's)**

Steps of Dijkstra's algorithm

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 7 & 4 & 6 & 1 \\ 2 & \infty & 0 & \infty & \infty & \infty \\ 3 & \infty & 2 & 0 & 5 & \infty \\ 4 & \infty & 3 & \infty & 0 & \infty \\ 5 & \infty & \infty & \infty & 1 & 0 \end{array}$$

length[i] = length[vnear] + W[vnear][i]

	$Y=\{v_1\}$		$Y=\{v_1,v_5\}$		$Y=\{v_1,v_4,v_5\}$		$Y=\{v_1,v_3,v_4,v_5\}$	
i	t	l	t	l	t	l	t	l
2	1	7	1	7	4	5	4	$\underline{\underline{5}}$
3	1	4	1	4	1	$\underline{\underline{4}}$	1	-1
4	1	6	5	$\underline{\underline{2}}$	5	-1	5	-1
5	1	$\underline{\underline{1}}$	1	-1	1	-1	1	-1
	vnear = 5		vnear = 4		vnear = 3		vnear = 2	
	add(v_1,v_5)		add(v_5,v_4)		add(v_1,v_3)		add(v_4,v_2)	

Algorithm 4.3 Dijkstra's Algorithm (1/3)

- **<u>Problem</u>: Determine the shortest paths from v_1 to all other vertices in a weighted, directed graph.**
- **<u>Inputs</u>: integer $n \geq 2$, and a connected, weighted, directed graph containing n vertices. The graph is represented by a 2D array W, which has both its rows and columns indexed from 1 to n, where $W[i][j]$ is the weight on the edge from the i-th vertex to the j-th vertex.**
- **<u>Outputs</u>: set of edges F containing edges in shortest paths.**

Algorithm 4.3 Dijkstra's Algorithm (2/3)

```
void dijkstra (int n,
               const number W[ ][ ],
               set_of_edges& F)
{
   index i, vnear;
   edge e;
   index touch[2..n];
   number length[2..n];

   F = Ø ;
   for (i = 2; i <=n; i++) {     // For all vertices, initialize v1 to be the last
      touch[i] = 1;              // vertex on the current shortest path from v1,
      length[i] = W[1][i];       // and initialize length of that path
   }                             // to be the weight on the edge from v1.
```

Algorithm 4.3 Dijkstra's Algorithm (3/3)

```
repeat (n – 1 times) {                  // Add all n – 1 vertices to Y.
   min = ∞;
   for (i = 2; i <= n; i++)             // Check each vertex for having shortest path.
      if (0 ≤ length[i] < min) {
         min = length[i];
         vnear = i;
      }
   e = edge from vertex indexed by touch[vnear] to vertex indexed by vnear;
   add e to F;
   for (i = 2; i <= n; i++)
      if (length[vnear] + W[vnear][i] < length[i]) {
         length[i] = length[vnear] + W[vnear][i];
         touch[i] = vnear; // For each vertex not in Y,
      }                    // update its shortest path.
   length[vnear] = -1;     // Add vertex indexed by vnear to Y.
}
}
```

- $T(n)$: similarly as Alg. 4.1

$$T(n) = 2(n-1)(n-1) \in \Theta(n^2)$$

	$Y=\{v_1\}$		$Y=\{v_1,v_5\}$		$Y=\{v_1,v_4,v_5\}$		$Y=\{v_1,v_3,v_4,v_5\}$	
i	t	l	t	l	t	l	t	l
2	1	7	1	7	4	5	4	$\underline{\underline{5}}$
3	1	4	1	4	1	$\underline{\underline{4}}$	1	−1
4	1	6	5	$\underline{\underline{2}}$	5	−1	5	−1
5	1	$\underline{\underline{1}}$	1	−1	1	−1	1	−1
	vnear = 5		vnear = 4		vnear = 3		vnear = 2	
	add(v_1,v_5)		add(v_5,v_4)		add(v_1,v_3)		add(v_4,v_2)	

Correctness proof

- By induction on the size of Y that for each v in Y, *length*(v) is equal to the length of a shortest path from v_1 to v.
- Induction step:
  - Suppose v_i is chosen as vnear.
  - If *length*(i) is not the length of a shortest path from v_1 to v_i , then there must exist a shortest path P and P must contain some vertex other than v_i which is not in Y.
  - Let v_j be the first such vertex on P.
  - But then the distance v_1 to v_j is shorter than *length*(i) and moreover, the shortest path to v_j lies wholly within Y, except for v_j itself.
  - Thus, *length*(j) < *length*(i) when v_i was selected
  - → a contradiction