ÇİZGELER (GRAPHS)
PART - III

Shortest Path Algorithms

1) Dijkstra’s 2) Floyd-Warshall

Minimum Spanning Tree Algorithms

1)Prim’s 2)Kruskal’s

En Kısa Yol
Shortest Path

- Bir Çizgede, iki köşe arasındaki en kısa yol. Geçtiği kenarlardaki ağırlıklar toplamının en az olması isteniyor.

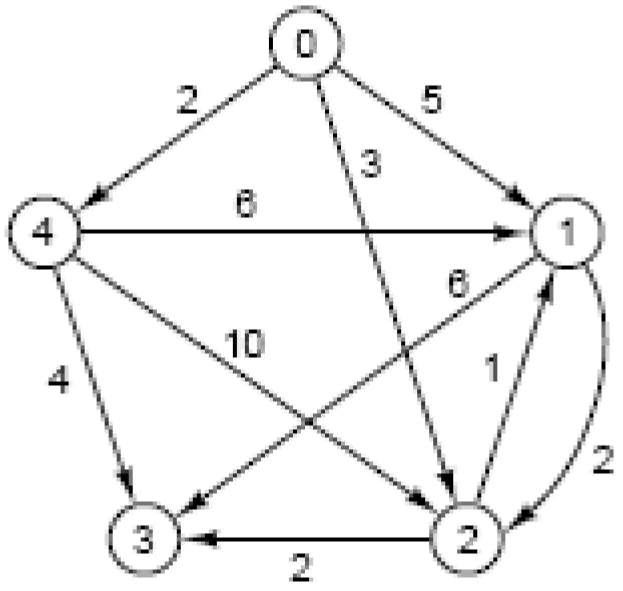
- Dijkstra's algorithm solves the single-source shortest path problems. Tek kaynaktan bütün köşelere.
- Floyd–Warshall algorithm solves all pairs shortest paths. Tüm köşe çiftleri arasında.

Dijkstra’s Algorithm

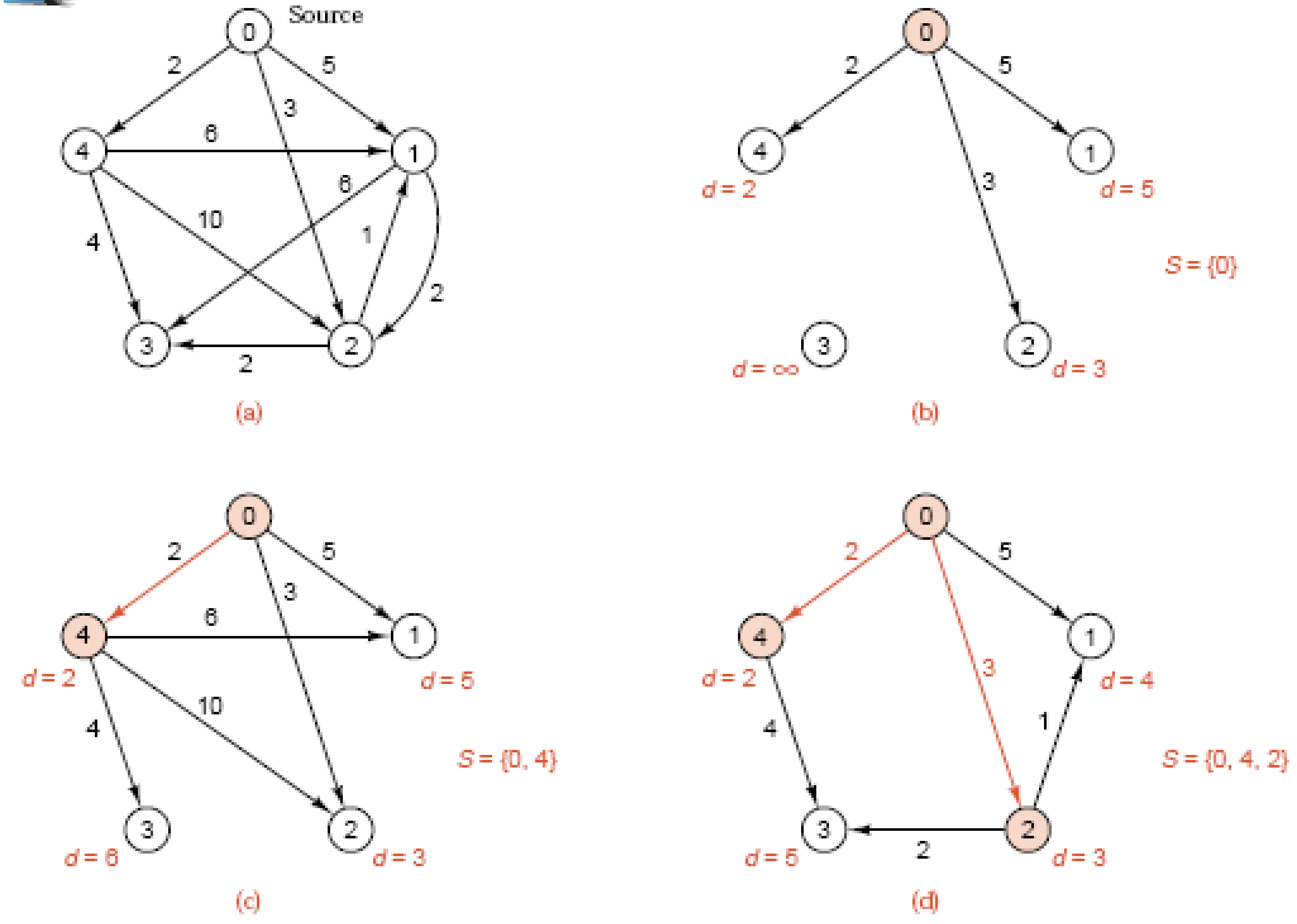
- Tek kaynaktan bütün köşelere en kısa yolları bulur. Örnek : 0-1, 0-2, 0-3, 0-4
- Bir Greedy Algoritmasıdır. Açgözlü bir yaklaşım izler. Kısa vadede en iyi veya sonuca en çok yaklaştıran.

Ağırlıklı Yönlü Çizge

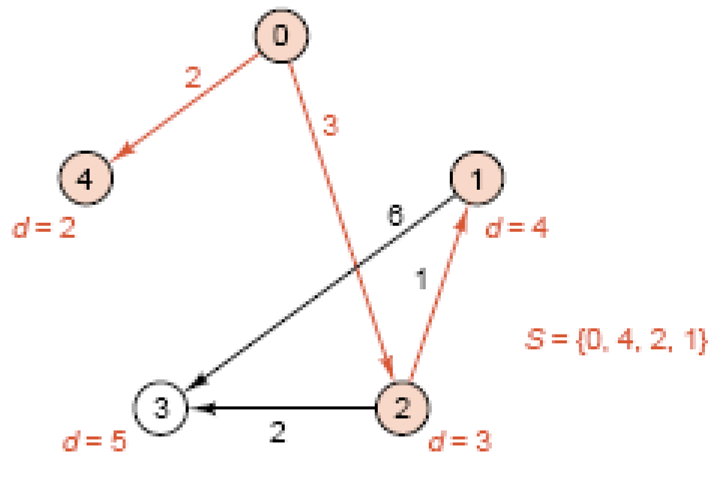
(Weighted Digraph) :



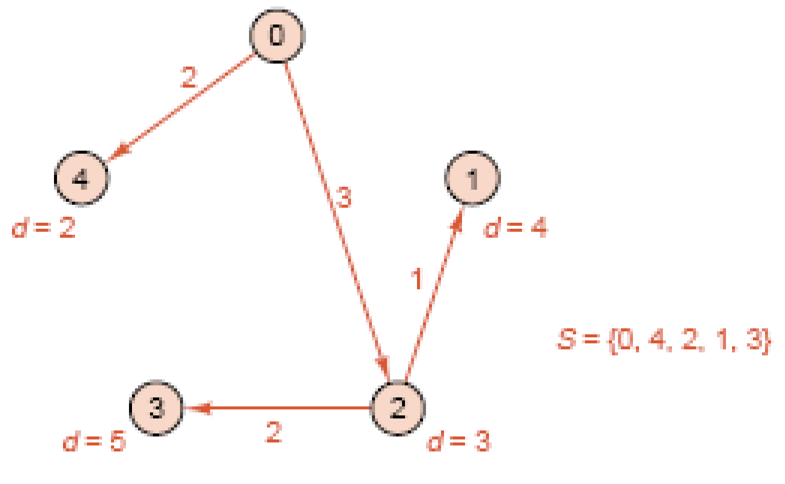
Method : First Steps



Method : Last Steps



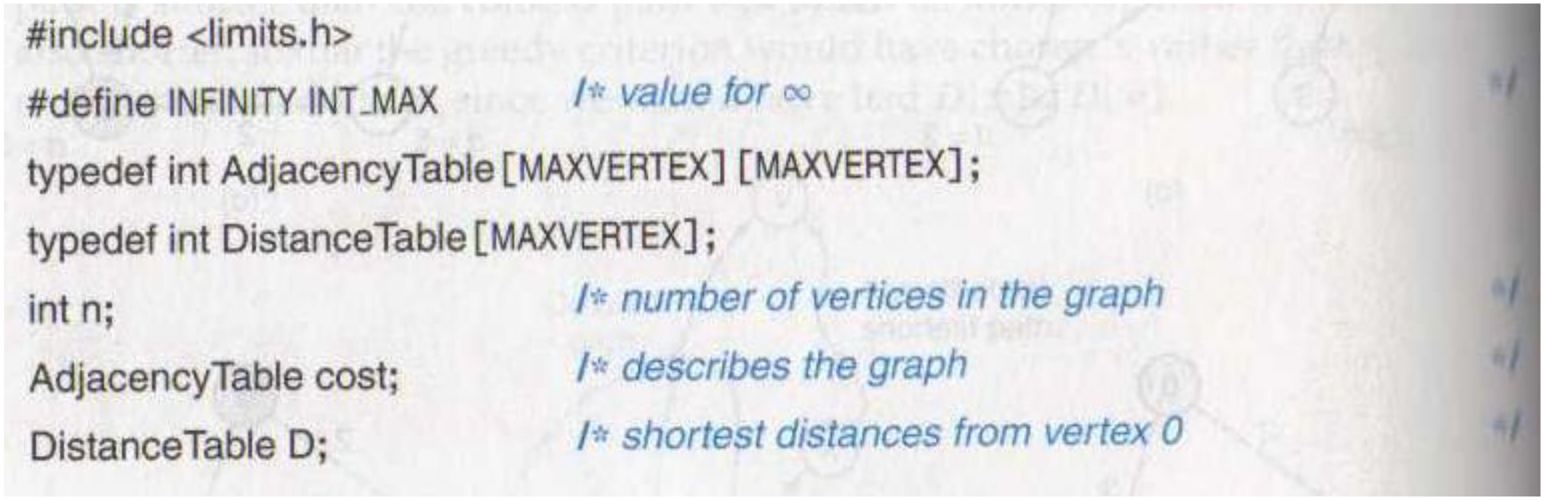
(e)



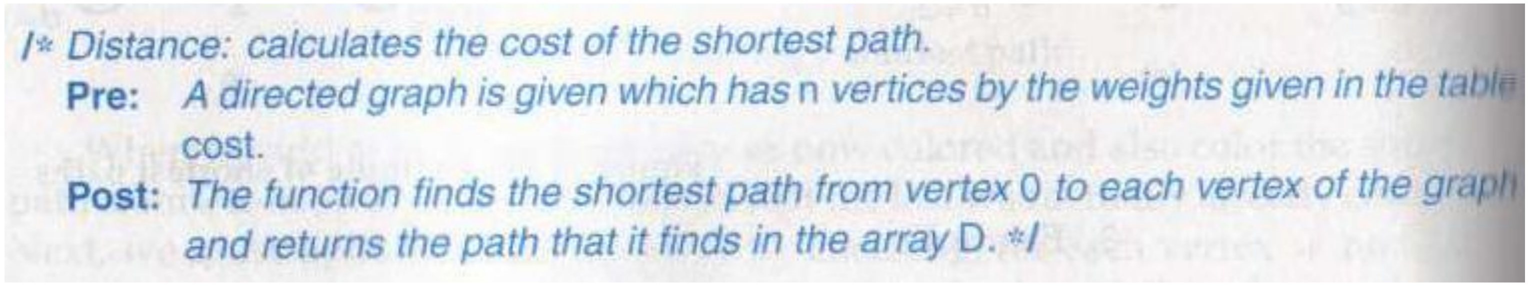
(f)

C Program
for Dijkstra’s Shortest Path Algorithm

from Data Structures and Program Design In C, Robert L. Kruse

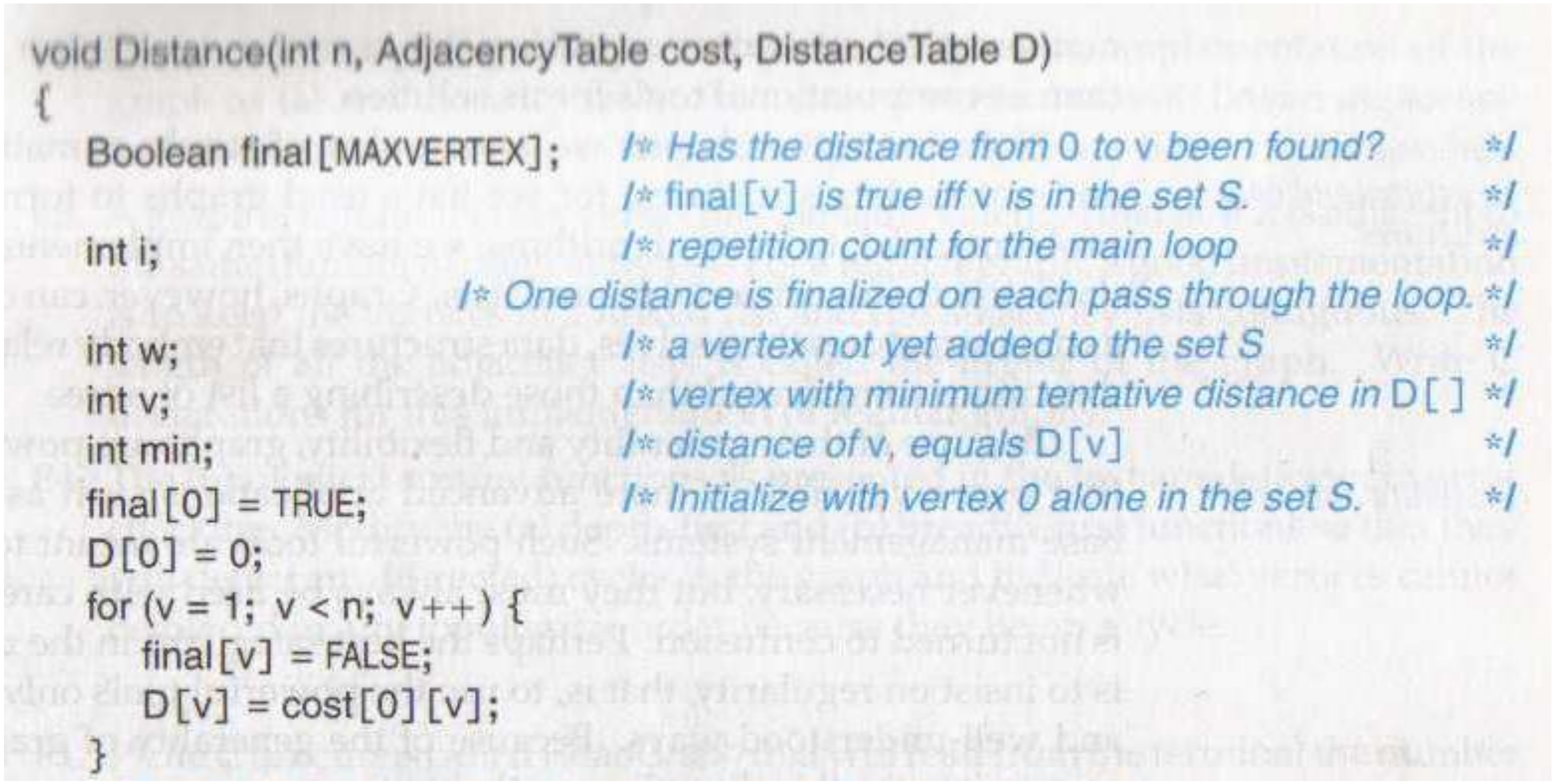


```
#include <limits.h>
#define INFINITY INT_MAX          /* value for ∞                          */
typedef int AdjacencyTable [MAXVERTEX] [MAXVERTEX];
typedef int DistanceTable [MAXVERTEX];
int n;                            /* number of vertices in the graph      */
AdjacencyTable cost;              /* describes the graph                  */
DistanceTable D;                  /* shortest distances from vertex 0     */
```



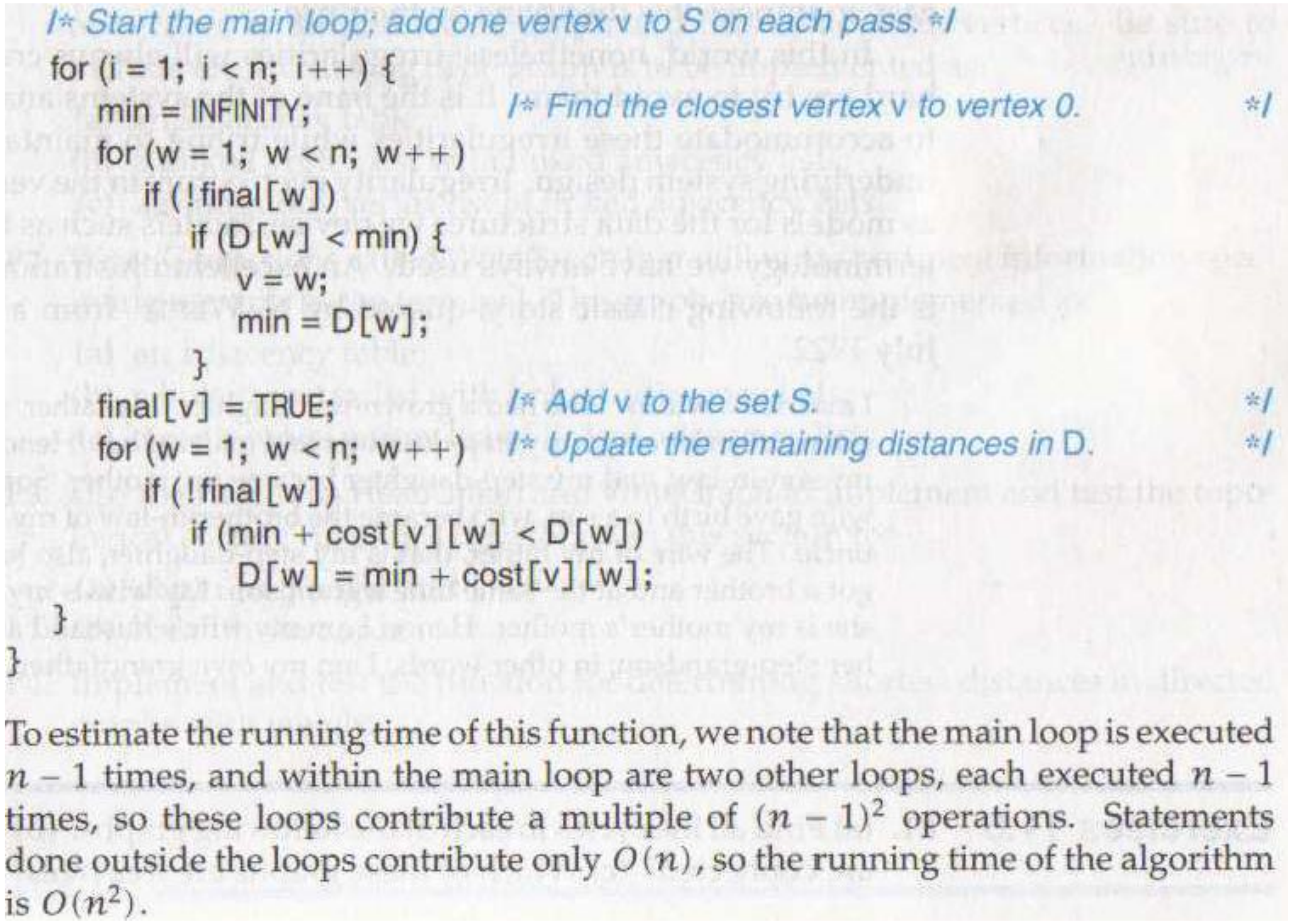
```
/* Distance: calculates the cost of the shortest path.
   Pre:  A directed graph is given which has n vertices by the weights given in the table
         cost.
   Post: The function finds the shortest path from vertex 0 to each vertex of the graph
         and returns the path that it finds in the array D. */
```

C Program for Dijkstra's Shortest Path Algorithm



```
void Distance(int n, AdjacencyTable cost, DistanceTable D)
{
    Boolean final[MAXVERTEX];    /* Has the distance from 0 to v been found?        */
                                 /* final[v] is true iff v is in the set S.         */
    int i;                       /* repetition count for the main loop              */
                        /* One distance is finalized on each pass through the loop. */
    int w;                       /* a vertex not yet added to the set S             */
    int v;                       /* vertex with minimum tentative distance in D[]   */
    int min;                     /* distance of v, equals D[v]                      */
    final[0] = TRUE;             /* Initialize with vertex 0 alone in the set S.    */
    D[0] = 0;
    for (v = 1; v < n; v++) {
        final[v] = FALSE;
        D[v] = cost[0][v];
    }
```

Main Loop



```
/* Start the main loop; add one vertex v to S on each pass. */
for (i = 1;  i < n;  i++) {
    min = INFINITY;           /* Find the closest vertex v to vertex 0.         */
    for (w = 1;  w < n;  w++)
        if (!final[w])
            if (D[w] < min) {
                v = w;
                min = D[w];
            }
    final[v] = TRUE;          /* Add v to the set S.                            */
    for (w = 1;  w < n;  w++) /* Update the remaining distances in D.           */
        if (!final[w])
            if (min + cost[v][w] < D[w])
                D[w] = min + cost[v][w];
    }
}
```

To estimate the running time of this function, we note that the main loop is executed $n - 1$ times, and within the main loop are two other loops, each executed $n - 1$ times, so these loops contribute a multiple of $(n - 1)^2$ operations. Statements done outside the loops contribute only $O(n)$, so the running time of the algorithm is $O(n^2)$.

Floyd Pseudocode

An algorithm to solve the *all pairs shortest path* problem in a weighted, directed graph

```
for i := 1 to n do
   for j := 1 to n do
      cost[i, j] := c[i, j];  // Let c[u, u] := 0
for k := 1 to n do
   for i := 1 to n do
      for j := 1 to n do
         sum = cost[i, k] + cost[k, j];
         if(sum < cost[i, j]) then cost[i, j] := sum;
```

Floyd’s Code

```
for (int k = 0; k < N; k++)
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            if (M[i][k]+M[k][j] < M[i][j]) then
                M[i][j] = M[i][k]+M[k][j]
```

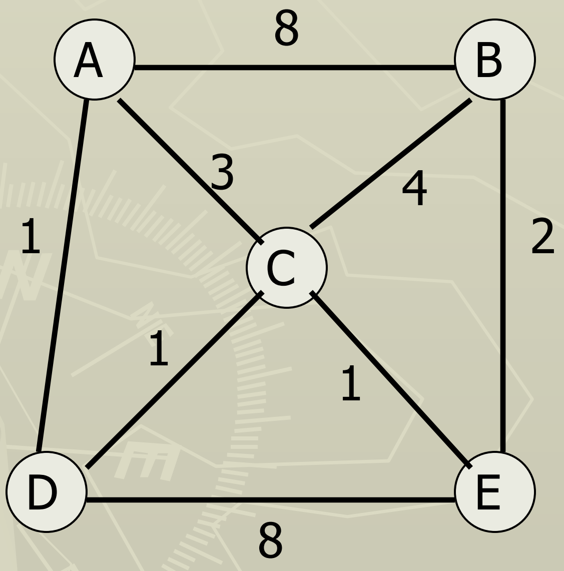
Dynamic Programming

- Floyd, bir dinamik programlama algoritmasıdır.

- Dynamic Programming
  - Solves one step of the problem
  - Stores it, so it doesn’t have to be recomputed
  - Uses stored step to solve the next step
  - Avoids re-computing progressive steps

Floyd's Algorithm

- Back to the shortest-path problem.

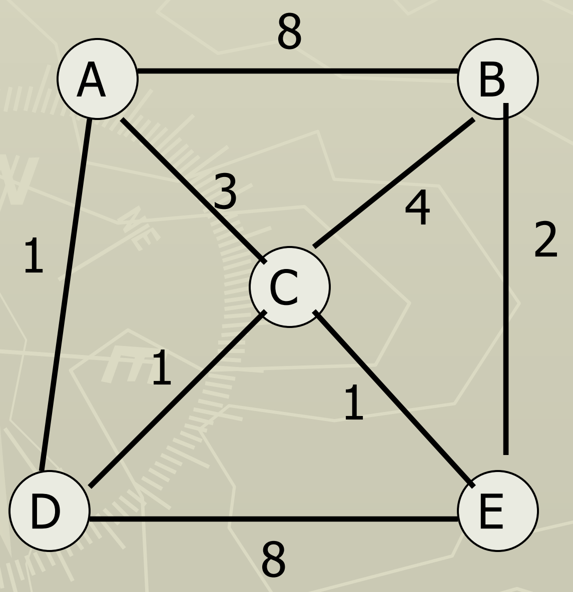


	A	B	C	D	E
A	0	8	3	1	∞
B	8	0	4	∞	2
C	3	4	0	1	1
D	1	∞	1	0	8
E	∞	2	1	8	0

Floyd's Algorithm

- First step, where can we go without using an intermediate vertex
- M^k, where k the set of intermediate vertices

	A	B	C	D	E
A	0	8	3	1	∞
B	8	0	4	∞	2
C	3	4	0	1	1
D	1	∞	1	0	8
E	∞	2	1	8	0



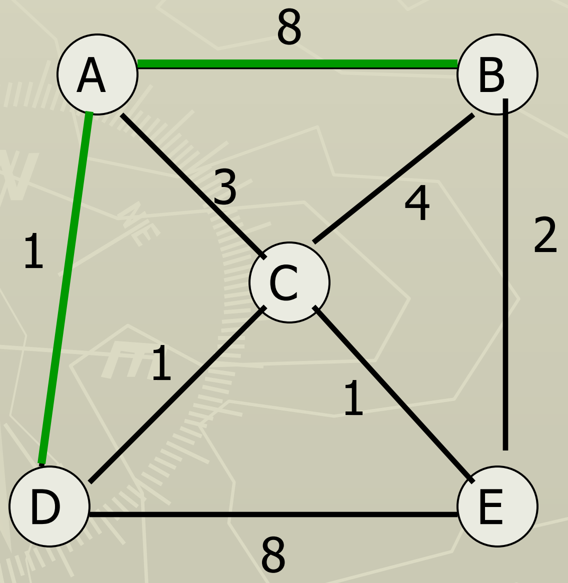
$M^{\{\}}$

	A	B	C	D	E
A	0	8	3	1	∞
B	8	0	4	∞	2
C	3	4	0	1	1
D	1	∞	1	0	8
E	∞	2	1	8	0

Floyd's Algorithm

► Second step, where can we go if we use A as an intermediate vertices

	A	B	C	D	E
A	0	8	3	1	∞
B	8	0	4	∞	2
C	3	4	0	1	1
D	1	∞	1	0	8
E	∞	2	1	8	0



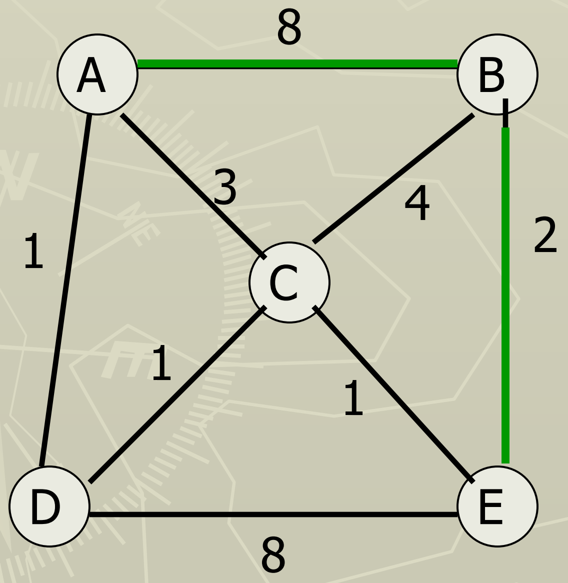
$M^{\{A\}}$

	A	B	C	D	E
A	0	8	3	1	∞
B	8	0	4	**9**	2
C	3	4	0	1	1
D	1	**9**	1	0	8
E	∞	2	1	8	0

Floyd's Algorithm

► Third step, where can we go if we use A and B as intermediate vertices

	A	B	C	D	E
A	0	8	3	1	∞
B	8	0	4	∞	2
C	3	4	0	1	1
D	1	∞	1	0	8
E	∞	2	1	8	0



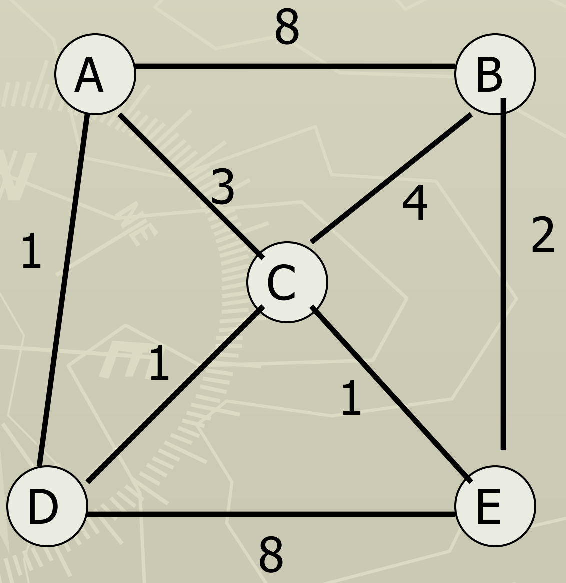
$M^{\{A,B\}}$

	A	B	C	D	E
A	0	8	3	1	**10**
B	8	0	4	9	2
C	3	4	0	1	1
D	1	9	1	0	8
E	**10**	2	1	8	0

Floyd's Algorithm

- Fourth step, where can we go if we use A, B, and C as intermediate vertices
- **You tell me!**

	A	B	C	D	E
A	0	8	3	1	∞
B	8	0	4	∞	2
C	3	4	0	1	1
D	1	∞	1	0	8
E	∞	2	1	8	0



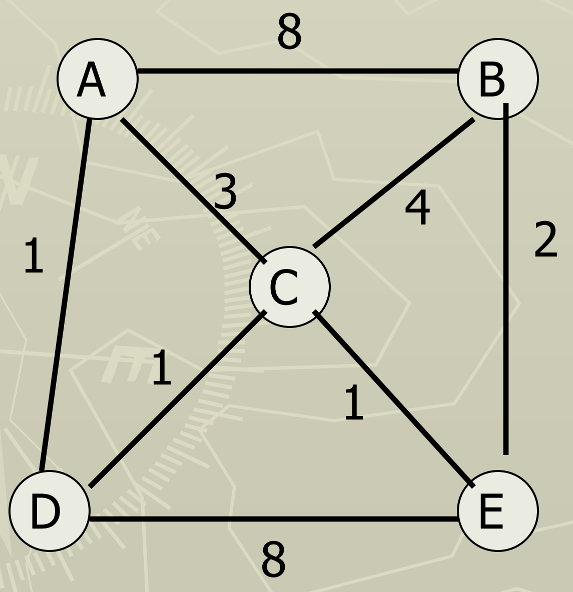
$M^{\{A,B,C\}}$

	A	B	C	D	E
A	0	8	3	1	10
B	8	0	4	9	2
C	3	4	0	1	1
D	1	9	1	0	8
E	10	2	1	8	0

Floyd's Algorithm

- Fourth step, where can we go if we use A, B, and C as intermediate vertices

	A	B	C	D	E
A	0	8	3	1	∞
B	8	0	4	∞	2
C	3	4	0	1	1
D	1	∞	1	0	8
E	∞	2	1	8	0



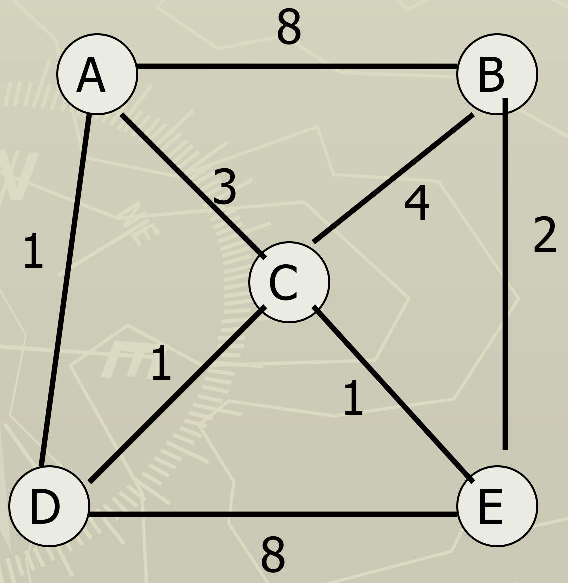
$M^{\{A,B,C\}}$

	A	B	C	D	E
A	0	**7**	3	1	**4**
B	**7**	0	4	**4**	2
C	3	4	0	1	1
D	1	**4**	1	0	**2**
E	**4**	2	1	**2**	0

Floyd's Algorithm

- Fifth step, where can we go if we use A, B, C, and D as intermediate vertices
- **You tell me!**

	A	B	C	D	E
A	0	8	3	1	∞
B	8	0	4	∞	2
C	3	4	0	1	1
D	1	∞	1	0	8
E	∞	2	1	8	0



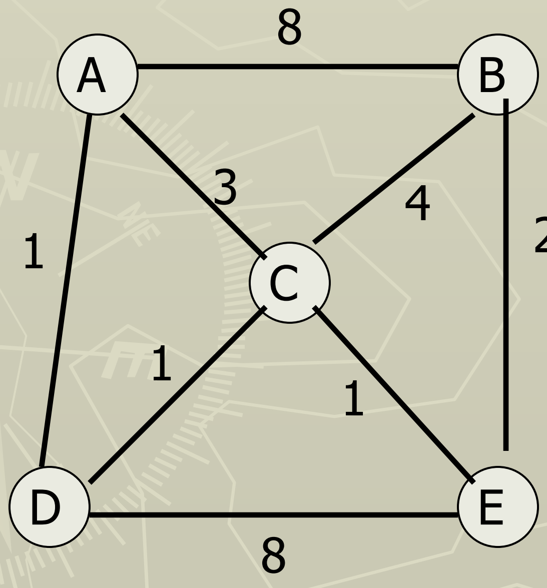
$M^{\{A,B,C,D\}}$

	A	B	C	D	E
A	0	7	2	1	4
B	7	0	4	5	2
C	2	4	0	1	1
D	1	5	1	0	2
E	4	2	1	2	0

Floyd's Algorithm

- Fifth step, where can we go if we use A, B, C, and D as intermediate vertices
- **OK, here is the answer.**

	A	B	C	D	E
A	0	8	3	1	∞
B	8	0	4	∞	2
C	3	4	0	1	1
D	1	∞	1	0	8
E	∞	2	1	8	0



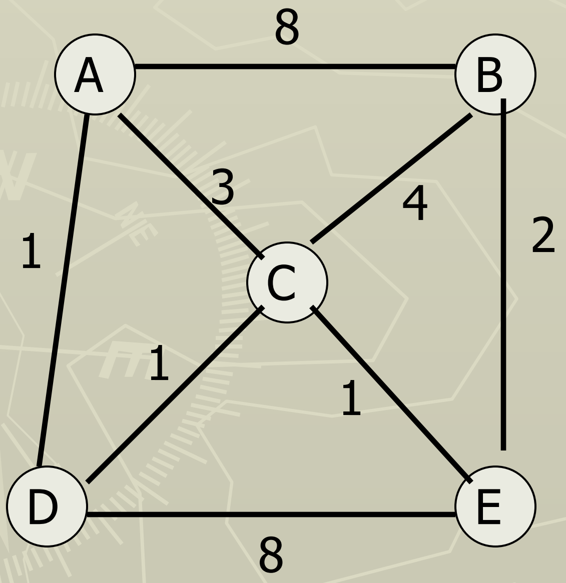
$M^{\{A,B,C,D\}}$

	A	B	C	D	E
A	0	**6**	2	1	**3**
B	**6**	0	4	4	2
C	2	4	0	1	1
D	1	4	1	0	2
E	**3**	2	1	2	0

Floyd's Algorithm

	A	B	C	D	E
A	0	8	3	1	∞
B	8	0	4	∞	2
C	3	4	0	1	1
D	1	∞	1	0	8
E	∞	2	1	8	0

- Final step, where can we go if we use all the vertices as intermediates.
- **You tell me!**



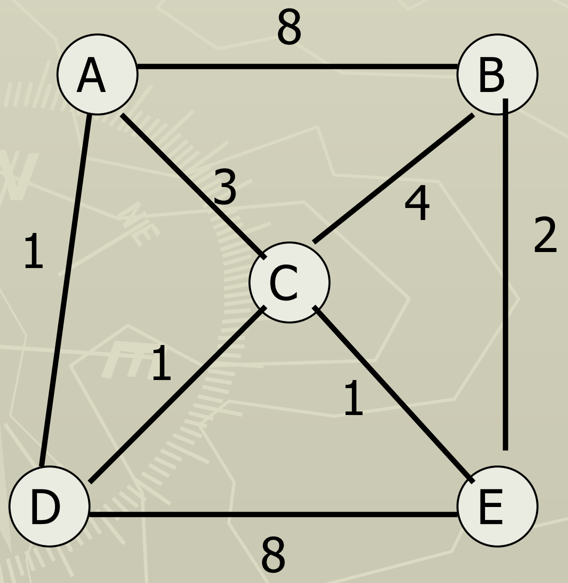
$M^{\{A,B,C,D,E\}}$

	A	B	C	D	E
A	0	6	2	1	3
B	6	0	4	4	2
C	2	4	0	1	1
D	1	4	1	0	2
E	3	2	1	2	0

Floyd's Algorithm

	A	B	C	D	E
A	0	8	3	1	∞
B	8	0	4	∞	2
C	3	4	0	1	1
D	1	∞	1	0	8
E	∞	2	1	8	0

- Final step, where can we go if we use all the vertices as intermediates.
- **You tell me!**



$M^{\{A,B,C,D,E\}}$

	A	B	C	D	E
A	0	**5**	2	1	3
B	**5**	0	**3**	4	2
C	2	**3**	0	1	1
D	1	4	1	0	2
E	3	2	1	2	0

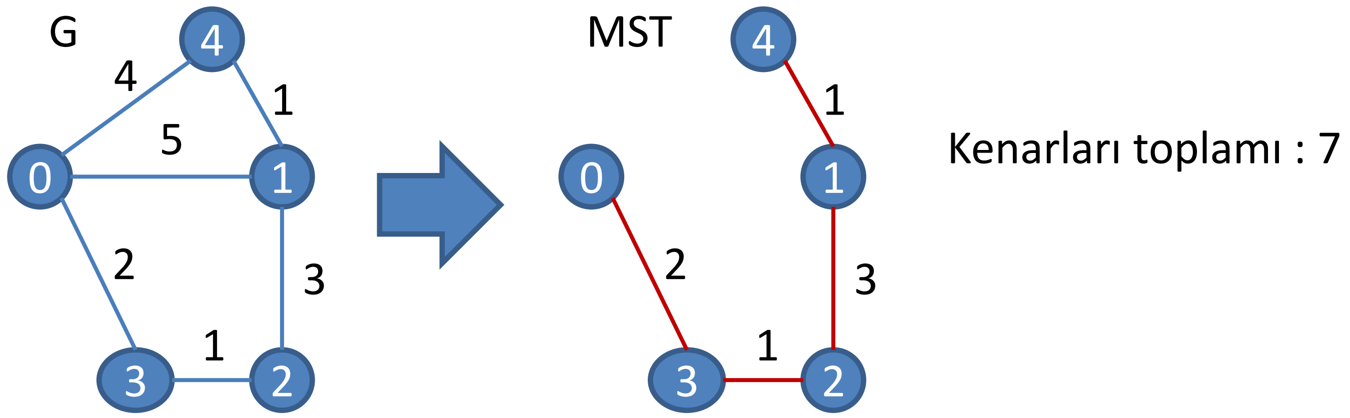
Warshall’s Algorithm

```
for (int k = 0; k < N; k++)
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            $w_{ij} = w_{ij} \vee (w_{ik} \wedge w_{kj})$
```

- Warshall's algorithm is an efficient method for computing the transitive closure of a relation.
- Komşuluk Matrisi için!
- Her ikisine Floyd-Warshall da denilebiliyor.

En Küçük Kapsayan Ağaç Minimum Spanning Tree (MST)

- Bağlı yönsüz bir çizgede Kapsayan Ağaç (Spanning Tree), ağaç şeklindeki bir alt çizgedir ve tüm köşeleri birbirine bağlar.
- En küçük kapsayan ağaç ise, kenarlarındaki maliyet toplamı en küçük olanıdır:



- Bir çizge, birden çok farklı MST içerebilir.

Prim’s Algorithm

Start with a vertex, and put it in the tree.

Then repeatedly do the following:

1) Find all the edges from the newest vertex to other vertices that aren’t in the tree. Put these edges in the Priority Queue (PQ).
2) Pick the edge from the PQ with the lowest weight and add this edge and its destination to the tree.

Repeat these steps until all the vertices are in the tree.

Newest means most recently installed in the tree.

In Prim’s algorithm, the set A forms a single tree.

Kruskal’s Algorithm

The safe edge added to A is always a least-weight edge in the graph that connects two distinct components.

- Scan all the edges in increasing weight order. If an edge is safe, add it F (Forest)

Kruskal’s algorithm is a greedy algorithm, because at each step it adds to the forest an edge of least possible weight.