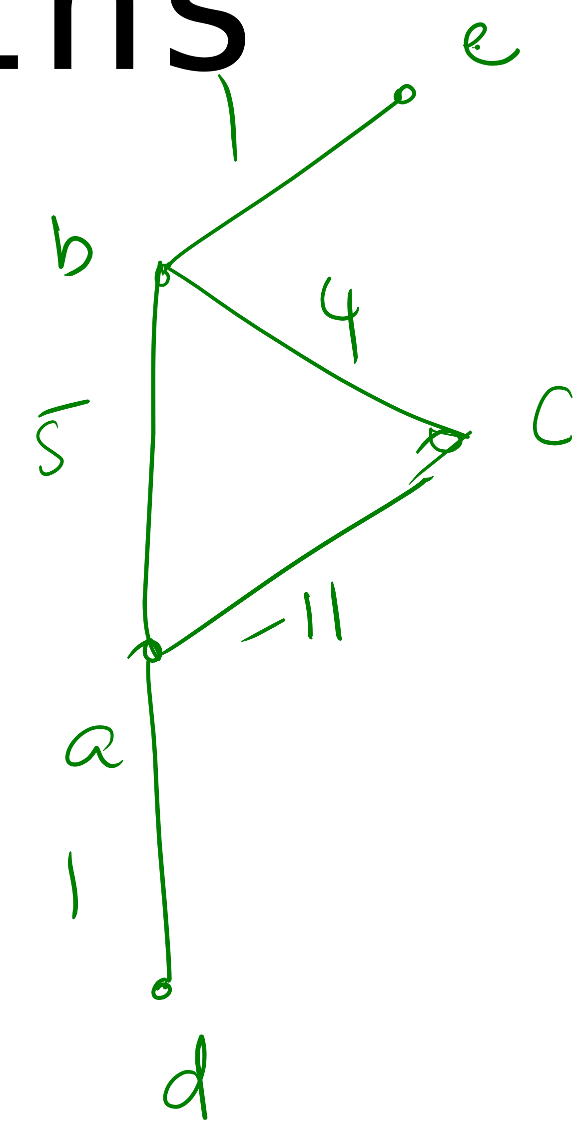
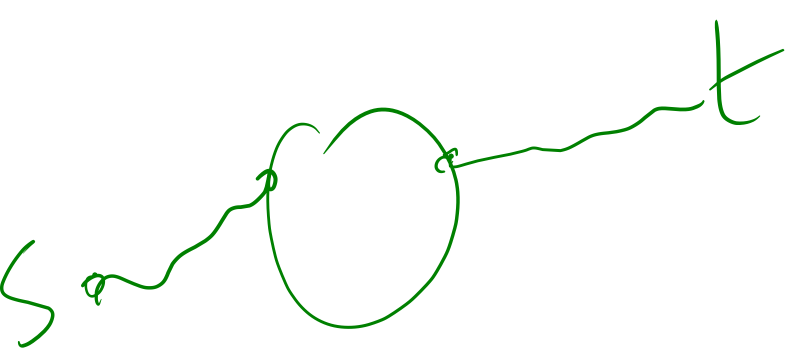
Shortest Path Algorithms

Shortest Paths

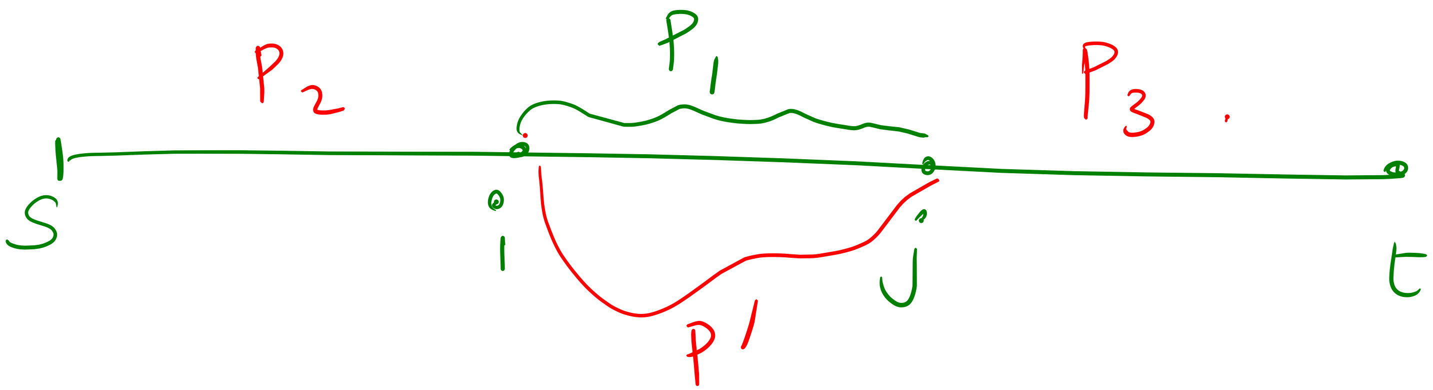


- Negative edges and cycles
- Optimal substructure



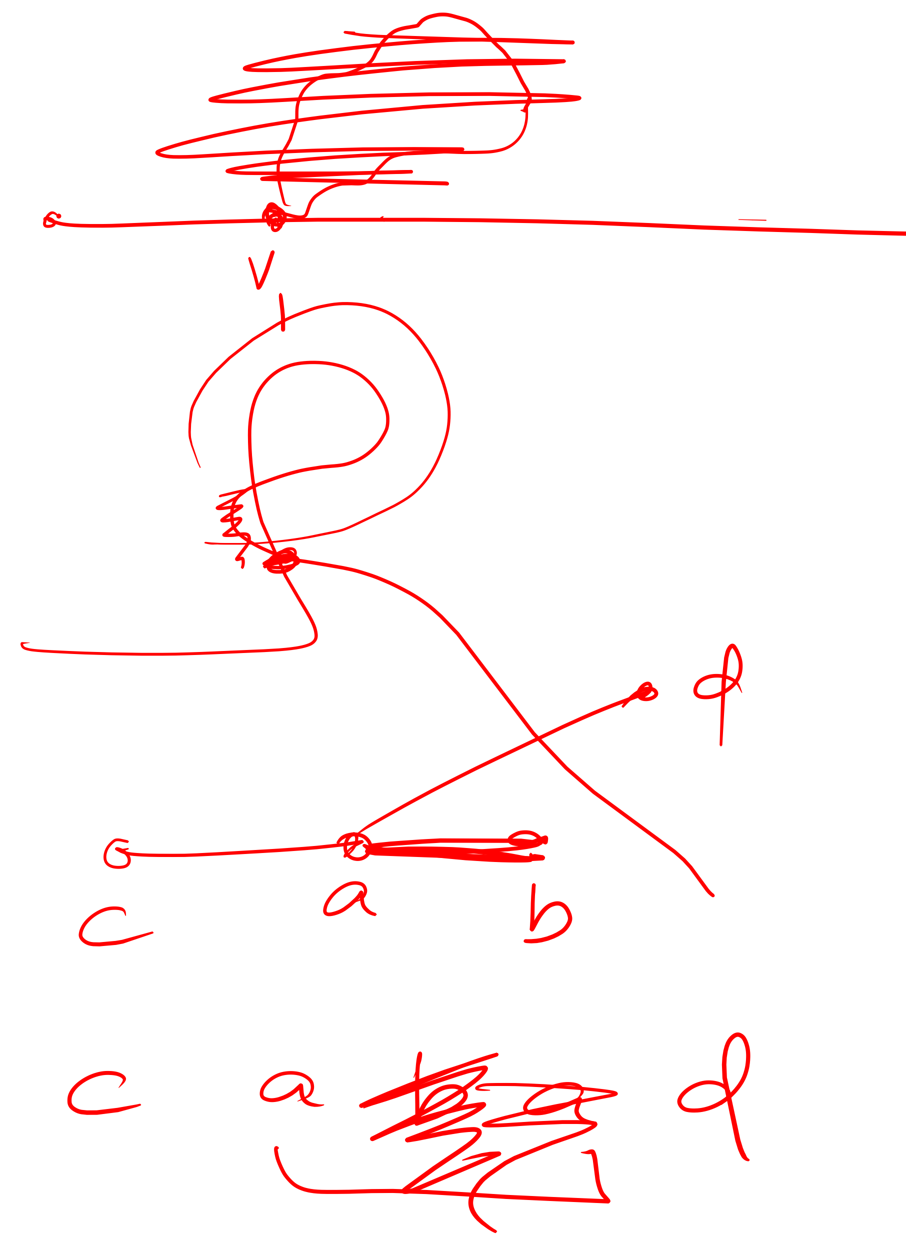
dacbe -5.

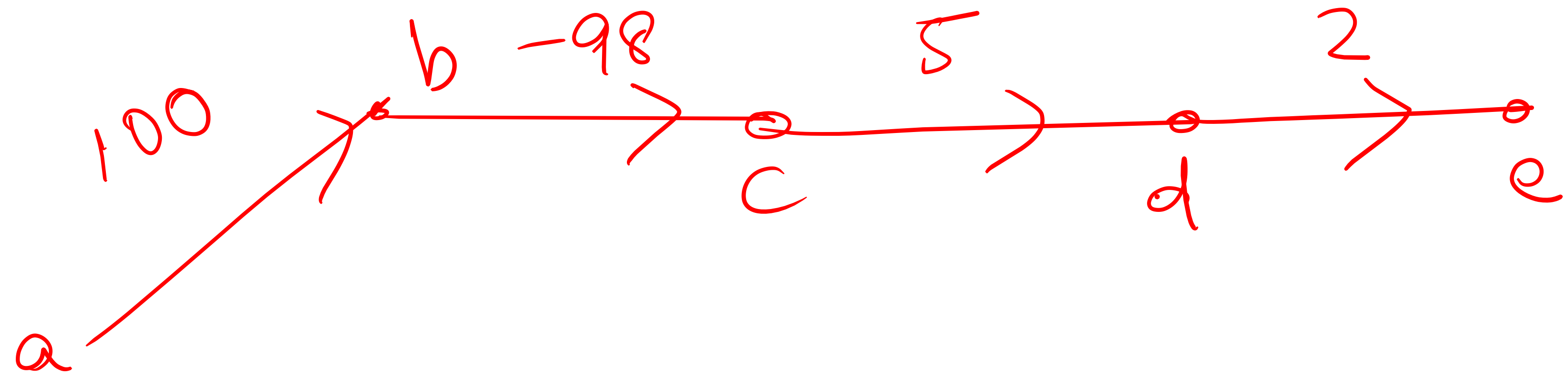
dacbacbe



$P \Rightarrow \delta(s,t)$

p' connecting i & j is a subpath in P. thus $p' := \delta(i,j)$.





100
-98
5
2
a
b
c
d
e

Shortest Paths

- If G is an unweighted graph, how do we find a shortest path from s to t?

Bellman – Ford Algorithm

- Graph G
- Given a source s and a destination t, find a shortest path from s to t
- G may have negative edges but no negative cycles

Bellman – Ford Algorithm

- If G has no negative cycles, then the shortest path from s to t is simple and hence has at most n-1 edges

Bellman – Ford Algorithm

- If G has no negative cycles, then the shortest path from s to t is simple and hence has at most n-1 edges

- dynamic programming algorithm

Bellman – Ford Algorithm

- OPT(i, v) - minimum cost of a v - t path using at most i edges

Bellman – Ford Algorithm

- OPT(i, v) - minimum cost of a v - t path using at most i edges
- OPT(n-1, s) is the desired solution

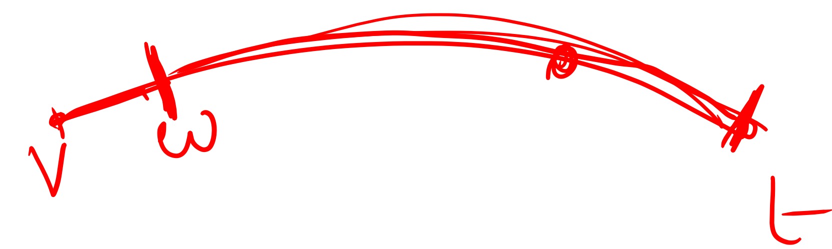
Bellman – Ford Algorithm

Let P be the shortest path of at most i edges between v and T

Bellman – Ford Algorithm

Let P be the shortest path of at most i edges between v and T

- P has at most i-1 edges



$$OPT(i, v) = OPT(i-1, v)$$

-

Bellman – Ford Algorithm

Let P be the shortest path of at most i edges between v and T

- P has at most i-1 edges

$$OPT(i, v) = OPT(i-1,v)$$

- P has i edges. Guess the first edge in P.

$$OPT(i, v) = \min_{w \in N(v)} c(v, w) + OPT(i-1, w)$$

Bellman – Ford Algorithm

$$OPT(i, v) =$$

$$\min(\underbrace{OPT(i-1, \overset{v}{\cancel{w}})}, \min_{w \in N(v)} c(v, w) + OPT(i-1, w))$$

Bellman – Ford Algorithm

OPT(i, v) =

min(OPT(i-1,w), $\min_{w \in N(v)}$ c(v, w) + OPT(i-1, w))

Proof of correctness by induction on i, using optimal substructure property of shortest paths

Bellman – Ford Algorithm

Running Time :

Bellman – Ford Algorithm

Running Time : $O(n^3)$

Bellman - Ford Algorithm

→ for i = 0 to n − 1
  for v ∈ V
    for w ∈ Adj[v]
      OPT(i, v) = ___

Running Time : $O(nm)$

For a fixed i, v, the value of OPT(i, v) is used for the computation of OPT(i+1, w) for all w ∈ N(v). Summing for all vertices $= \sum_{v \in V} \deg(v) = O(m)$ times.

Summing over all i, $\Theta(nm)$.

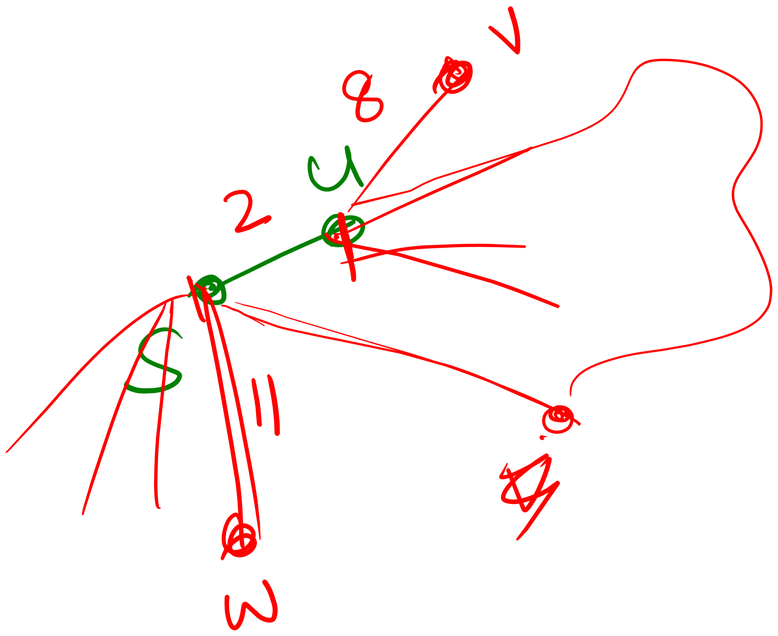
Dijkstra’s Algorithm

Given a graph G and a source s, find the shortest path from s to all vertices.

Dijkstra’s Algorithm

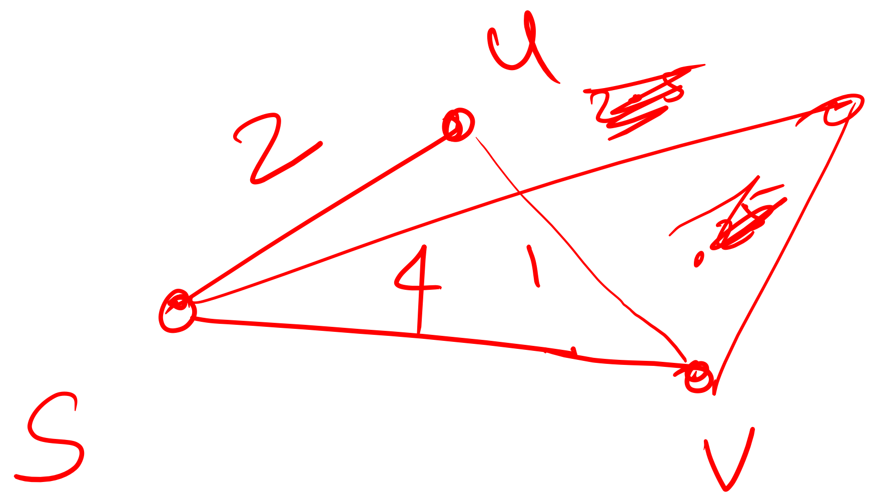
Greedy Algorithm

Dijkstra's Algorithm



Greedy Algorithm

- maintains a set S of vertices u for which we have determined a shortest path distance d(u) from s


u
2
4
1
s
v

Dijkstra's Algorithm

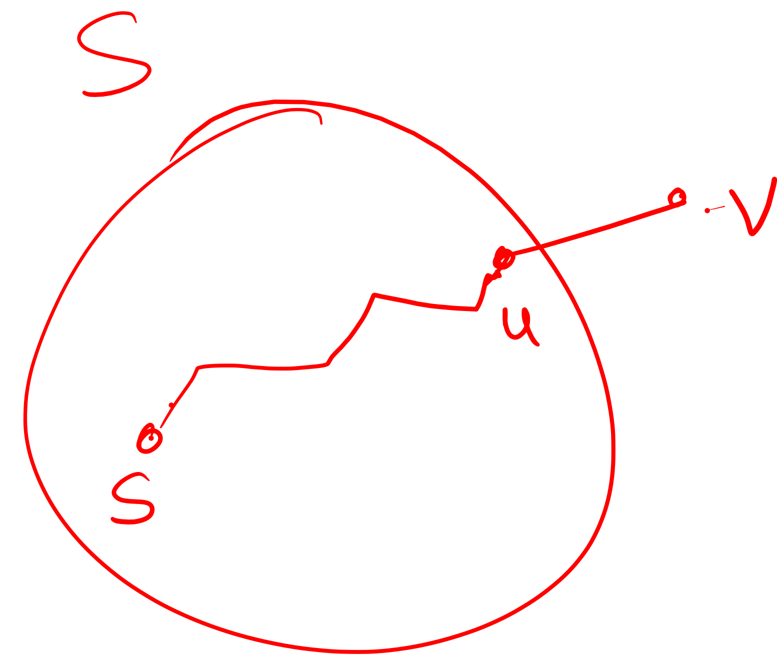
Greedy Algorithm

 - maintains a set S of vertices u for which we have determined a shortest path distance d(u) from - "explored"

Dijkstra’s Algorithm

Initially, $S = \{s\}$, $d(s) = 0$

Dijkstra's Algorithm



Initially, S = {s}, d(s) = 0

For $v \in V \backslash S$, we determine the shortest path that can be constructed by traveling along a path through S to some $u \in S$, followed by a single edge (u, v)

Dijkstra's Algorithm

S = {s}, d(s) = 0

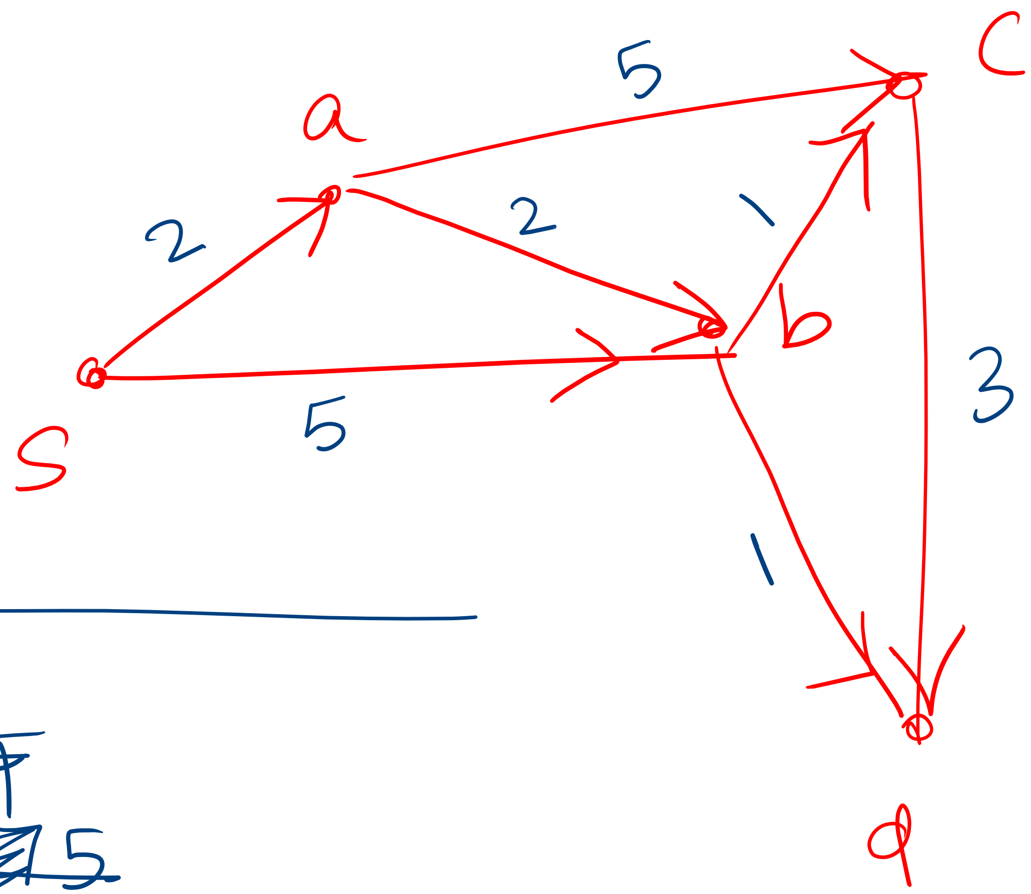
while $S \neq V$,

Select a node $v \notin S$ such that

$d'(v) = \min_{e=(u,v), u \in S} d(u) + l_e$ is minimum

$S = S \cup \{v\}$

d(v) = d'(v)

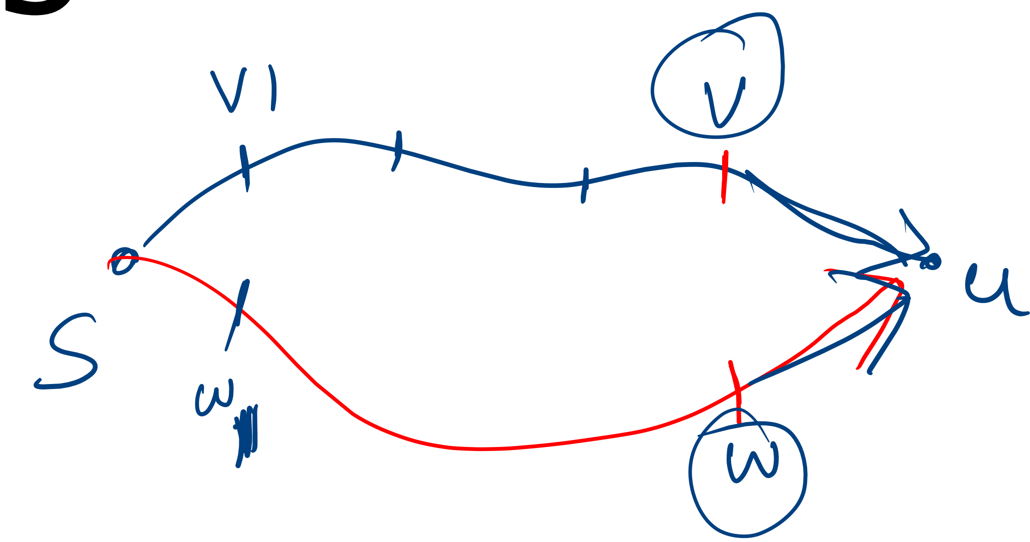


	d'
~~a~~	~~2~~
~~b~~	~~5~~ ~~4~~
~~c~~	[illegible] ~~7~~ ~~5~~
d	[illegible] 5

		d
S –	s	0
	a	2
	b	4
	c	5
	d	5.

Dijkstra's Algorithm

Proof of Correctness :



Dijkstra's Algorithm

Running Time :

Dijkstra’s Algorithm

Running Time :

- n ExtractMin()
- m Updates

Dijkstra’s Algorithm

Running Time :

- n ExtractMin()
- m Updates

(n+m) log n - using binary heaps

n log n + m - using Fibonacci heaps

Dijkstra’s Algorithm

What happens when there are negative edges?

Floyd–Warshall Algorithm

<u>All-Pairs Shortest Path problem :</u>

Input : Directed weighted Graph G

Output : Shortest path between every pair of vertices

Floyd–Warshall Algorithm

- uses Dynamic Programming

Floyd–Warshall Algorithm

- uses Dynamic Programming

- Let the set of vertices be {1,2,……, n}

Floyd–Warshall Algorithm

D[i,j,k] - weight of the shortest path between i and j, for which all intermediate vertices are from the set {1,2,….,k}

Floyd–Warshall Algorithm

D[i,j,k] - weight of the shortest path between i and j, for which all intermediate vertices are from the set {1,2,….,k}

{D[i,j,n]} - final solution

Floyd–Warshall Algorithm

recurrence for D[i,j,k]:

- k does not belong to the shortest path from i to j

D[i,j,k] = D[i,j,k-1]

Floyd-Warshall Algorithm

recurrence for D[i,j,k]:

- k does not belong to the shortest path from i to j

  D[i,j,k] = D[i,j,k-1]

- k belongs to the shortest path from i to j

  D[i,j,k] = D[i,k,k-1] + D[k,j,k-1]

Floyd–Warshall Algorithm

```
recurrence for D[i,j,k]:

D[i,j,0] = w(i,j)

D[i,j,k] =

min{D[i,j,k-1],D[i,k,k-1] + D[k,j,k-1]}
```

Floyd–Warshall Algorithm

Running Time :