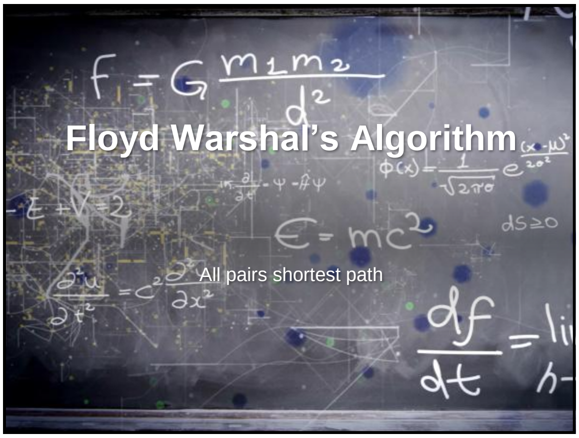


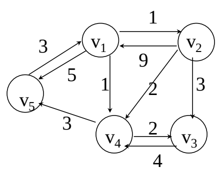
1

All pairs shortest path

- ***The problem:*** find the shortest path between every pair of vertices of a graph
- ***The graph*:** may contain negative edges but no negative cycles
- ***A representation*:** a weight matrix where
  W(i,j)=0 if i=j.
  W(i,j)=∞ if there is no edge between i and j.
  W(i,j)="weight of edge"
- Note: we have shown principle of optimality applies to shortest path problems

2

The weight matrix and the graph



	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0

3

The subproblems

- How can we define the shortest distance $d_{i,j}$ in terms of "smaller" problems?
- One way is to restrict the paths to only include vertices from a restricted subset.
- Initially, the subset is empty.
- Then, it is incrementally increased until it includes all the vertices.

4

The subproblems

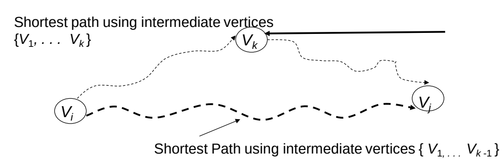
- Let $D^{(k)}[i,j]$=weight of a shortest path from v_i to v_j using only vertices from $\{v_1,v_2,\ldots,v_k\}$ as intermediate vertices in the path
  - $D^{(0)}=W$
  - $D^{(n)}=D$ which is the goal matrix
- How do we compute $D^{(k)}$ from $D^{(k-1)}$?

5

The Recursive Definition:

Case 1: A shortest path from v_i to v_j restricted to using only vertices from $\{v_1,v_2,\ldots,v_k\}$ as intermediate vertices does not use v_k. Then $D^{(k)}[i,j]= D^{(k-1)}[i,j]$.

Case 2: A shortest path from v_i to v_j restricted to using only vertices from $\{v_1,v_2,\ldots,v_k\}$ as intermediate vertices does use v_k. Then $D^{(k)}[i,j]= D^{(k-1)}[i,k]+ D^{(k-1)}[k,j]$.



6

The recursive definition

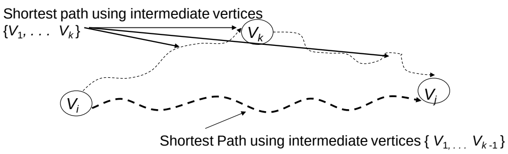
- Since

  $D^{(k)}[i,j] = D^{(k-1)}[i,j]$ or

  $D^{(k)}[i,j] = D^{(k-1)}[i,k] + D^{(k-1)}[k,j]$.

  We conclude:

  $D^{(k)}[i,j] = \min\{ D^{(k-1)}[i,j], D^{(k-1)}[i,k] + D^{(k-1)}[k,j] \}$.



7

The parent array Pi

Used to enable finding a shortest path

Initially the array contains 0

Each time that a shorter path from *i* to *j* is found the *k* that provided the minimum is saved (highest index node on the path from *i* to *j*)

To print the intermediate nodes on the shortest path a recursive procedure that print the shortest paths from *i* and *k*, and from *k* to *j* can be used

8

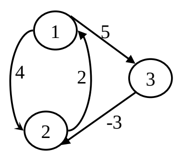
Floyd's Algorithm Using n+1 *D* matrices

Floyd//Computes shortest distance between all pairs of
//nodes, and saves P to enable finding shortest paths

1. $D^0 \leftarrow W$ // initialize *D* array to *W* [ ]
2. $P \leftarrow 0$ // initialize P array to [0]
3. **for** $k \leftarrow 1$ **to** n
4. **do for** $i \leftarrow 1$ **to** n
5. **do for** $j \leftarrow 1$ **to** n
6. **if** $(D^{k-1}[i, j] > D^{k-1}[i, k] + D^{k-1}[k, j])$
7. **then** $D^k[i, j] \leftarrow D^{k-1}[i, k] + D^{k-1}[k, j]$
8. $P[i, j] \leftarrow k$;
9. **else** $D^k[i, j] \leftarrow D^{k-1}[i, j]$

9

Example



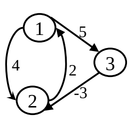
$W = D^0 =$

	1	2	3
1	0	4	5
2	2	0	∞
3	∞	-3	0

P =

	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

10



$D^0 =$

	1	2	3
1	0	4	5
2	2	0	∞
3	∞	-3	0

k = 1
Vertex 1 can be intermediate node

$D^1 =$

	1	2	3
1	0	4	5
2	2	0	7
3	∞	-3	0

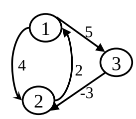
$D^1[2,3] = \min(D^0[2,3], D^0[2,1]+D^0[1,3])$
$= \min(\infty, 7)$
$= 7$

P =

	1	2	3
1	0	0	0
2	0	0	1
3	0	0	0

$D^1[3,2] = \min(D^0[3,2], D^0[3,1]+D^0[1,2])$
$= \min(-3, \infty)$
$= -3$

11



$D^1 =$

	1	2	3
1	0	4	5
2	2	0	7
3	∞	-3	0

k = 2
Vertices 1, 2 can be intermediate

$D^2 =$

	1	2	3
1	0	4	5
2	2	0	7
3	-1	-3	0

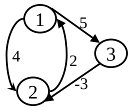
$D^2[1,3] = \min(D^1[1,3], D^1[1,2]+D^1[2,3])$
$= \min(5, 4+7)$
$= 5$

P =

	1	2	3
1	0	0	0
2	0	0	1
3	2	0	0

$D^2[3,1] = \min(D^1[3,1], D^1[3,2]+D^1[2,1])$
$= \min(\infty, -3+2)$
$= -1$

12



$D^2 =$

	1	2	3
1	0	4	5
2	2	0	7
3	-1	-3	0

k = 3
Vertices 1, 2, 3 can be intermediate

$D^3 =$

	1	2	3
1	0	2	5
2	2	0	7
3	-1	-3	0

$D^3[1,2] = \min(D^2[1,2], D^2[1,3]+D^2[3,2])$
$= \min(4, 5+(-3))$
$= 2$

$P =$

	1	2	3
1	0	3	0
2	0	0	1
3	2	0	0

$D^3[2,1] = \min(D^2[2,1], D^2[2,3]+D^2[3,1])$
$= \min(2, 7+(-1))$
$= 2$

13

Floyd's Algorithm: Using 2 D matrices

Floyd
1. ***D* ← *W*** // initialize *D* array to *W* [ ]
2. ***P*** ← 0 // initialize P array to [0]
3. **for *k* ← 1 to *n***
   // Computing D' from D
4. **do for *i* ← 1 to *n***
5. **do for *j* ← 1 to *n***
6. **if (*D*[*i*, *j*] > *D*[*i*, *k*] + *D*[*k*, *j*])**
7. **then *D'*[*i*, *j*] ← *D*[*i*, *k*] + *D*[*k*, *j*]**
8. ***P*[*i*, *j*] ← *k*;**
9. **else *D'*[*i*, *j*] ← *D*[*i*, *j*]**
10. ***Move D' to D.***

14

Can we use only one D matrix?

- ***D*[*i,j*] depends only on elements in the *k*th column and row of the distance matrix.**
- **We will show that the *k*th row and the *k*th column of the distance matrix are unchanged when D^k is computed**
- **This means *D* can be calculated *in-place***

15

The main diagonal values

- Before we show that *k*th row and column of *D* remain unchanged we show that the main diagonal remains 0
- $D^{(k)}[\, j,j\,] = \min\{ D^{(k-1)}[\, j,j\,],\ \ D^{(k-1)}[\, j,k\,] + D^{(k-1)}[\, k,j\,] \}$
  $= \min\{ 0,\ \ D^{(k-1)}[\, j,k\,] + D^{(k-1)}[\, k,j\,] \}$
  $= 0$
- Based on which assumption?

16

The *k*th column

- *k*th column of D^k is equal to the *k*th column of D^{k-1}
- *Intuitively true* - a path from i to k will not become shorter by adding k to the allowed subset of intermediate vertices
- For all i, $D^{(k)}[i,k] =$
  $= \min\{ D^{(k-1)}[i,k],\ \ D^{(k-1)}[i,k]+ D^{(k-1)}[k,k] \}$
  $= \min \{ D^{(k-1)}[i,k],\ D^{(k-1)}[i,k]+0 \}$
  $= D^{(k-1)}[i,k]$

17

The *k*th row

- *k*th row of D^k is equal to the *k*th row of D^{k-1}

  For all *j*, $D^{(k)}[k,j] =$
  $= \min\{ D^{(k-1)}[k,j],\ D^{(k-1)}[k,k]+ D^{(k-1)}[k,j] \}$
  $= \min\{ D^{(k-1)}[\, k,j\,],\ 0+D^{(k-1)}[k,j\,] \}$
  $= D^{(k-1)}[\, k,j\,]$

18

Floyd's Algorithm using a single *D*

Floyd
1. $D \leftarrow W$ // initialize D array to W [ ]
2. $P \leftarrow 0$ // initialize P array to [0]
3. for $k \leftarrow 1$ to n
4. do for $i \leftarrow 1$ to n
5. do for $j \leftarrow 1$ to n
6. if ($D[i, j] > D[i, k] + D[k, j]$)
7. then $D[i, j] \leftarrow D[i, k] + D[k, j]$
8. $P[i, j] \leftarrow k$;

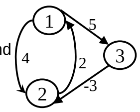
19

Printing intermediate nodes on shortest path from q to r

```
path(index q, r)
  if (P[ q, r ]!=0)
      path(q, P[q, r])
      println( "v"+ P[q, r])
      path(P[q, r], r)
     return;
  //no intermediate nodes
  else return
```

P =

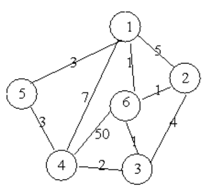
	1	2	3
1	0	3	0
2	0	0	1
3	2	0	0



Before calling path check $D[q, r] < \infty$, and print node q, after the call to path print node r

20

Example



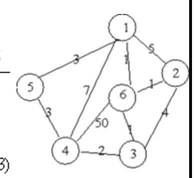
21

The final distance matrix and P

The values in parenthesis are the non zero P values.

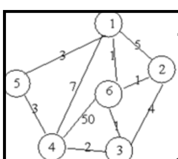
$D^6 =$

	1	2	3	4	5	6
1	0	2(6)	2(6)	4(6)	3	1
2	2(6)	0	2(6)	4(6)	5(6)	1
3	2(6)	2(6)	0	2	5(4)	1
4	4(6)	4(6)	2	0	3	3(3)
5	3	5(6)	5(4)	3	0	4(1)
6	1	1	1	3(3)	4(1)	0



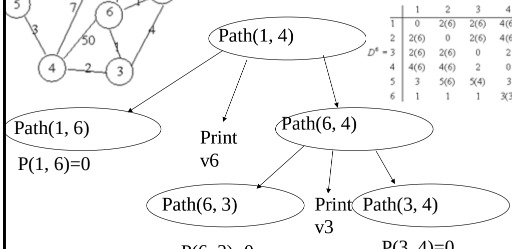
22

The call tree for Path(1, 4)



$D^6 =$

	1	2	3	4	5	6
1	0	2(6)	2(6)	4(6)	3	1
2	2(6)	0	2(6)	4(6)	5(6)	1
3	2(6)	2(6)	0	2	5(4)	1
4	4(6)	4(6)	2	0	3	3(3
5	3	5(6)	5(4)	3	0	4(1
6	1	1	1	3(3)	4(1)	0



The intermediate nodes on the shortest path from 1 to 4 are v6, v3.
The shortest path is v1, v6, v3, v4.

23