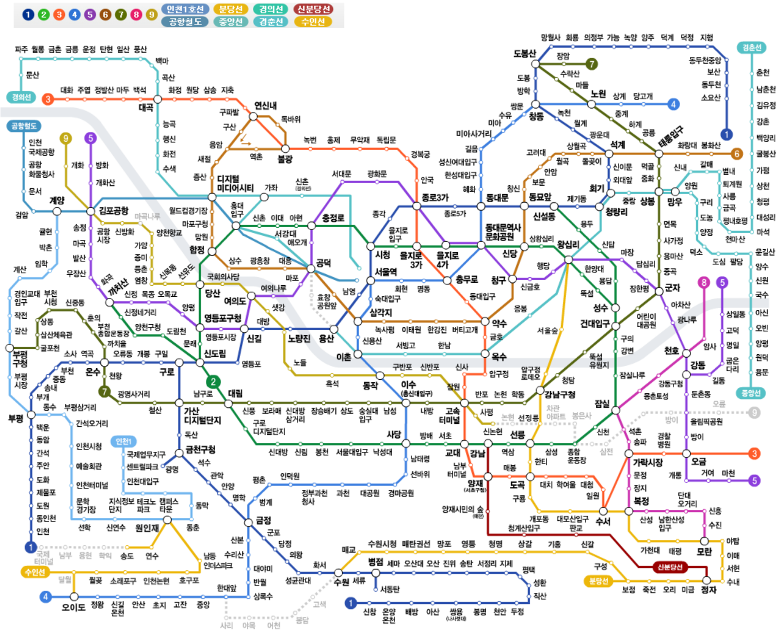
[CSED233-01] Data Structure

Shortest Path

Jaesik Park

POSTECH



Shortest Path Problems

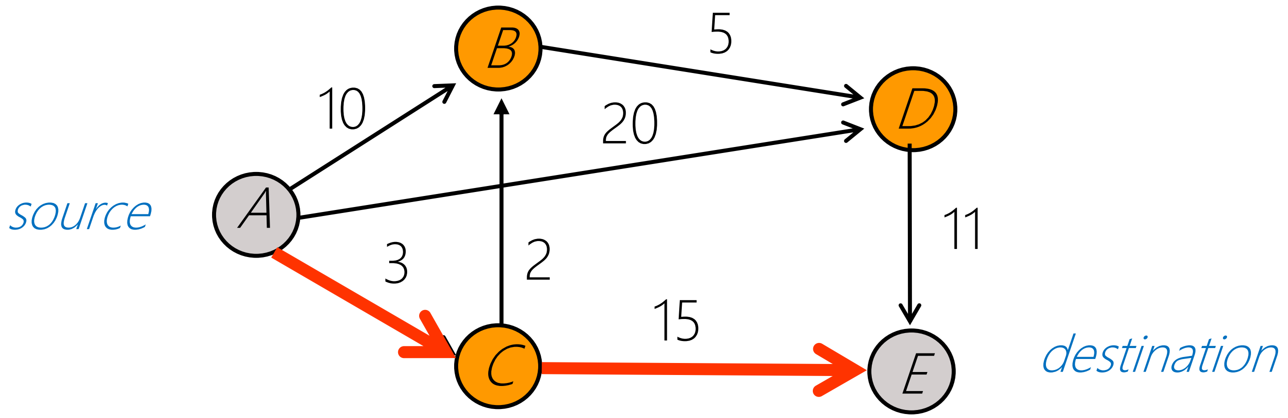
- Weighted digraph
  - Each edge *(i,j)* has a <u>nonnegative</u> cost *C[i,j]*
  - Also called weights or distances
- Path length/cost
  - Sum of weights of edges on path
  - source vertex ↔ destination vertex
- Shortest path problems (SPP)
  - Finding a path b/w two vertices such that the path length is minimized

Shortest Path Problems: Types

- Single-pair SPP
  - Single source, single destination
- Single-source SPP
  - Single source, all destinations
- Single-destination SPP
  - All sources, single destination
  - Can be reduced to the single-source SPP by reversing the edges in the graph
- All-pairs SPP
  - Every vertex is a source & destination

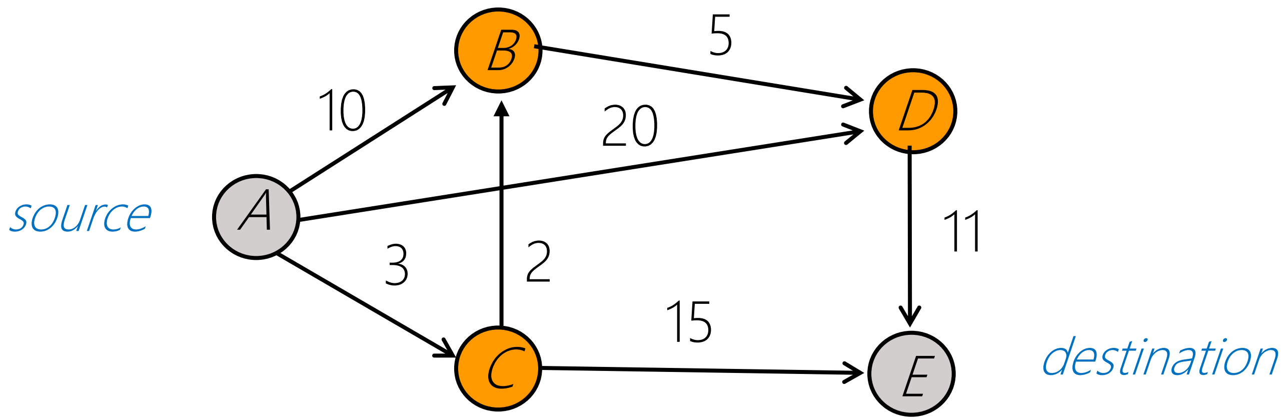
Single-Pair SPP: Single Source, Single Destination

- What is the shortest path from *A* to *E*?



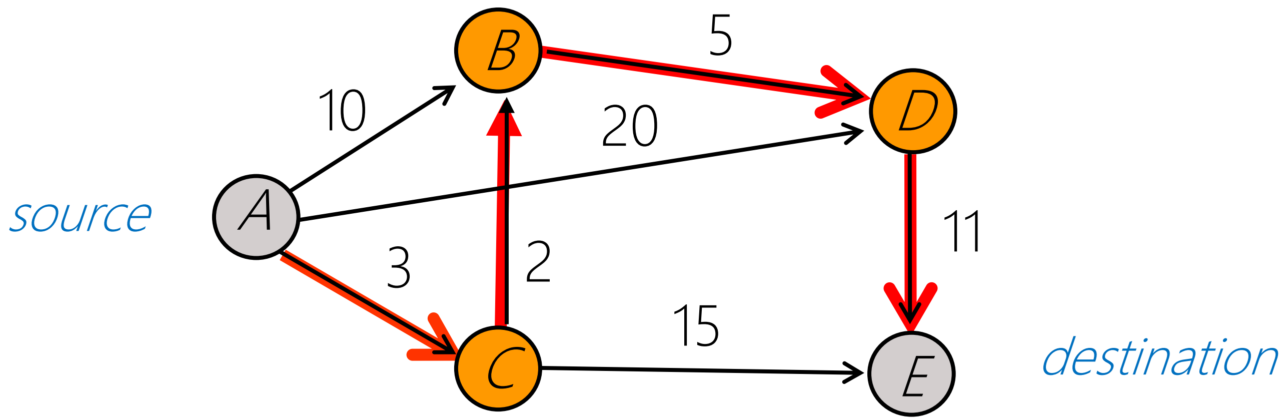
Shortest path length = 18

Single-Pair SPP: Single Source, Single Destination



- Greedy algorithm
  - Making the locally-optimal choice at each stage with the hope of finding a global optimum

Greedy Shortest A-to-E Path



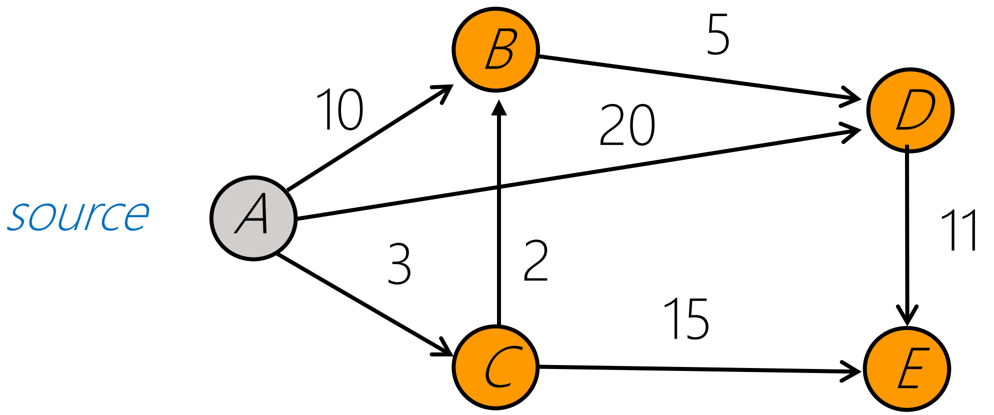
- Possible algorithm
  - Start with source vertex
  - Select a vertex using the cheapest edge subject to the constraint that a new vertex is reached
  - Continue until destination is reached
- Path length = 21 (not shortest path)
- Algorithm doesn't work

Single-Source SPP: Single Source, All Destinations

- Need to generate *n shortest paths*
  - From a given source to all destinations (*n*: # of vertices)

- Dijkstra's greedy method:
  - Generate the shortest paths in stages
    - Each stage generates a shortest path to a new destination
  - Greedy criterion at each stage
    - Select the destination (for the next shortest path) *in increasing order of its length*

Greedy: Single Source All Destinations



Length	Shortest Path
0	A
3	A → C
5	A → C → B
10	A → C → B → D
18	A → C → E

As a consequence, each path (other than first) is one-edge extension of an already generated shortest path

Greedy: Single Source All Destinations

- *D[i]* (distance from source to vertex *i*)
  - The length of the shortest one-edge extension of an already generated path, which ends at vertex *i*
  - Alternatively, *D[i]* can be defined to be the length of the "current" shortest path to *i* (that passes only through vertices in *S*)
    - where *S* is a set of vertices whose shortest path (distance) from source is already known
- *P(i)* (predecessor of *i*)
  - The vertex just before *i* on the shortest one-edge extension to *i*

Dijkstra's Alg. (Only to Compute Distance)

```
S := {1};
for   i := 2   to   n   do   begin
  D[i] := C[1, i]
for   i := 1   to   n-1  do   begin
  choose a vertex w ∈ V-S  whose D[.] value is least;
  add w to S;
  for    each vertex v ∈ V-S    do
      D[v] := min(D[v],  D[w] + C[w,v])
end;
```

One-edge extension

Update of D[.]

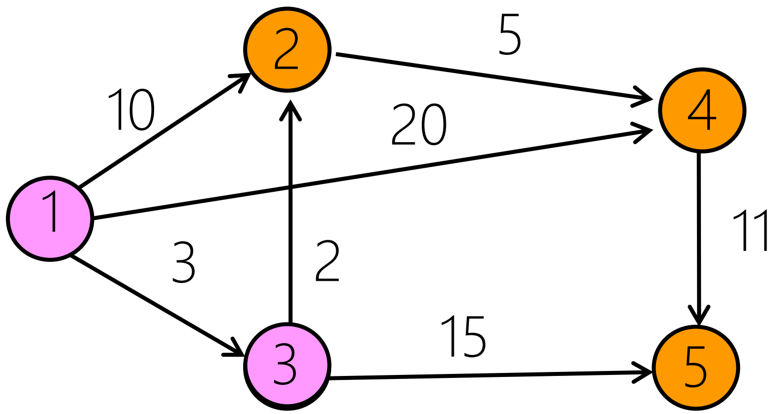
Dijkstra's Alg. (To Record Actual Path)

```
S := {1};
for  i := 2  to  n  do   begin
   D[i] := C[1, i];
   if  C[1, i] ≠ ∞  then  P[i] := 1      end;
for  i := 1  to  n-1  do  begin
   choose a vertex w ∈ V-S whose D[.] value is least;
   add w to S;
   for  each vertex v ∈ V-S  do
        if  (D[w] + C[w,v]) < D[v]  then  begin
             D[v] := D[w] + C[w,v];
             P[v] := w      end;
end;
```

Initialize P[.]

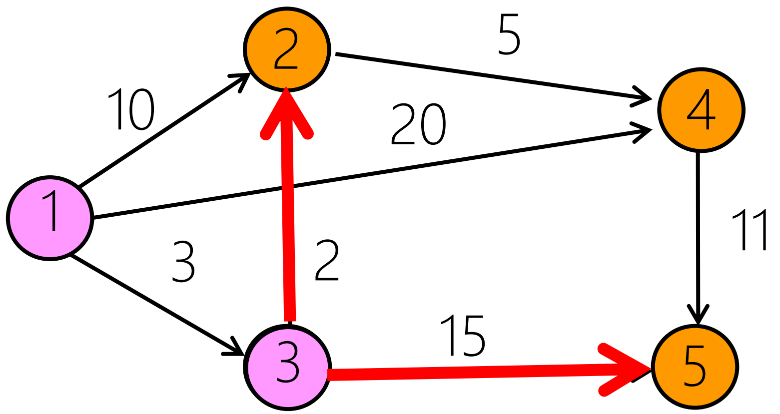
Path recording

Progress of Dijkstra’s Algorithm



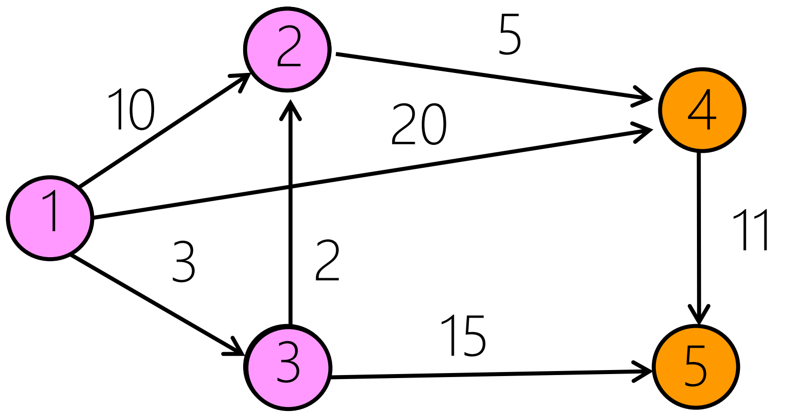
Stage	S	D/P[2]	D/P[3]	D/P[4]	D/P[5]
Initial	{1}	10/1	3/1	20/1	∞ / -
1					

Progress of Dijkstra's Algorithm



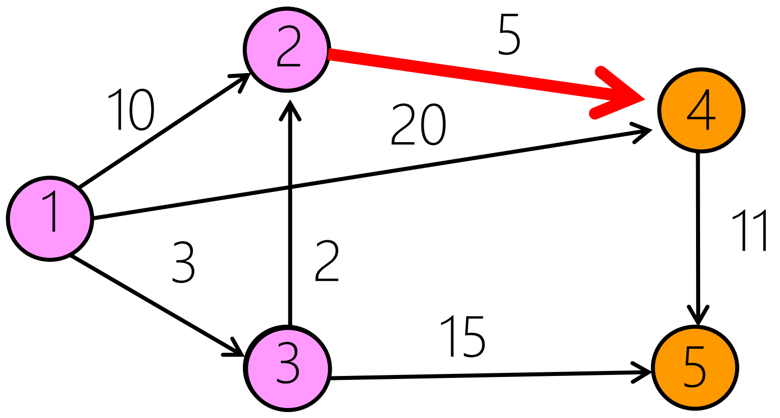
Stage	S	D/P[2]	D/P[3]	D/P[4]	D/P[5]
Initial	{1}	10/1	3/1	20/1	∞ / -
1	{1, 3}	5/3	3/1	20/1	18/3

Progress of Dijkstra's Algorithm



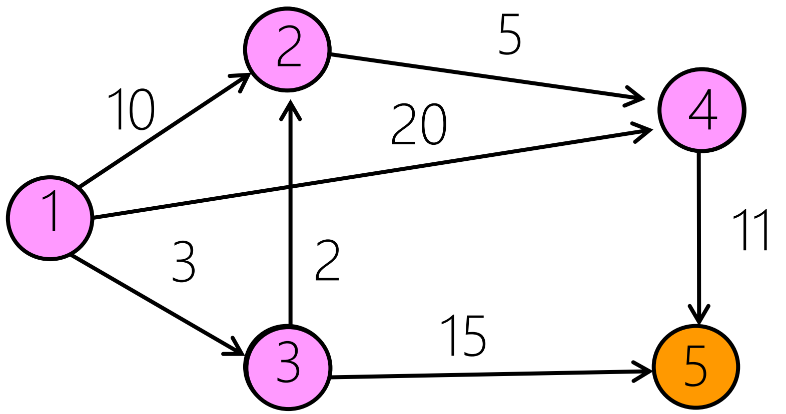
Stage	S	D/P[2]	D/P[3]	D/P[4]	D/P[5]
Initial	{1}	10/1	3/1	20/1	∞ / -
1	{1, 3}	5/3	3/1	20/1	18/3
2					

Progress of Dijkstra's Algorithm



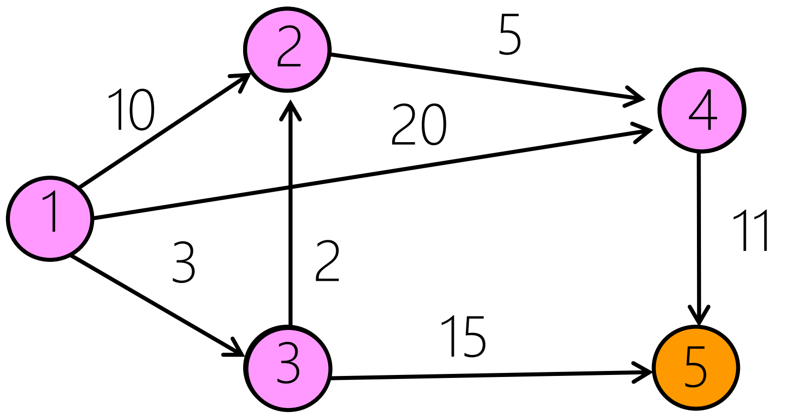
Stage	S	D/P[2]	D/P[3]	D/P[4]	D/P[5]
Initial	{1}	10/1	3/1	20/1	∞ / -
1	{1, 3}	5/3	3/1	20/1	18/3
2	{1, 3, 2}	5/3	3/1	10/2	18/3

Progress of Dijkstra’s Algorithm



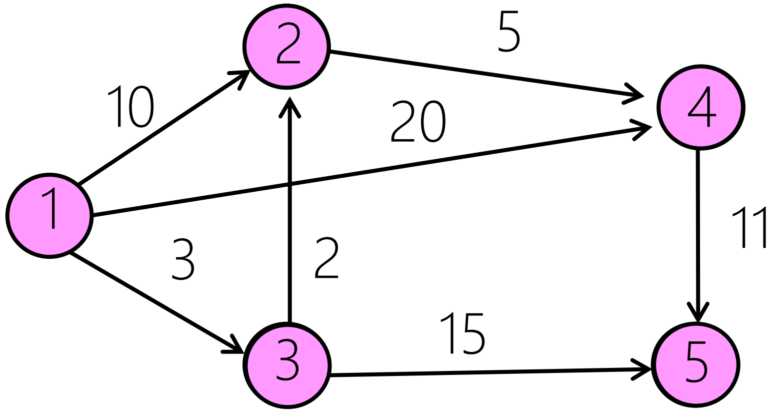
Stage	S	D/P[2]	D/P[3]	D/P[4]	D/P[5]
Initial	{1}	10/1	3/1	20/1	∞ / -
1	{1, 3}	5/3	3/1	20/1	18/3
2	{1, 3, 2}	5/3	3/1	10/2	18/3
3					

Progress of Dijkstra's Algorithm



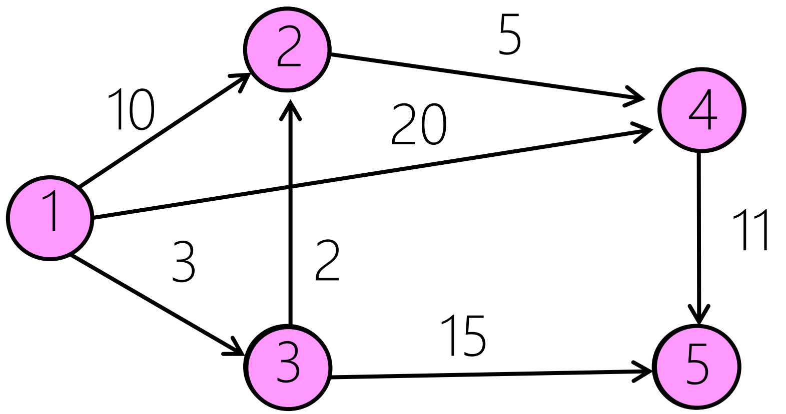
Stage	S	D/P[2]	D/P[3]	D/P[4]	D/P[5]
Initial	{1}	10/1	3/1	20/1	∞ / -
1	{1, 3}	5/3	3/1	20/1	18/3
2	{1, 3, 2}	5/3	3/1	10/2	18/3
3	{1, 3, 2, 4}	5/3	3/1	10/2	18/3

Progress of Dijkstra's Algorithm



Stage	S	D/P[2]	D/P[3]	D/P[4]	D/P[5]
Initial	{1}	10/1	3/1	20/1	∞ / -
1	{1, 3}	5/3	3/1	20/1	18/3
2	{1, 3, 2}	5/3	3/1	10/2	18/3
3	{1, 3, 2, 4}	5/3	3/1	10/2	18/3
4					

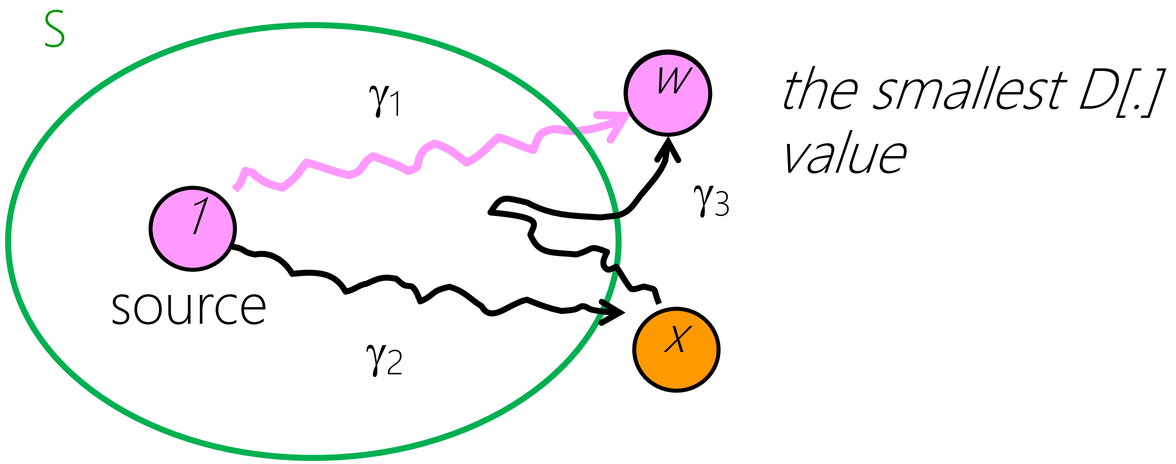
Progress of Dijkstra's Algorithm



Stage	S	D/P[2]	D/P[3]	D/P[4]	D/P[5]
Initial	{1}	10/1	3/1	20/1	∞ / -
1	{1, 3}	5/3	3/1	20/1	18/3
2	{1, 3, 2}	5/3	3/1	10/2	18/3
3	{1, 3, 2, 4}	5/3	3/1	10/2	18/3
4	{1, 3, 2, 4, 5}	5/3	3/1	10/2	18/3

Why Dijkstra's Algorithm Works (1)

- Locally-best choice turns out to be the best over all
  - Called greedy choice property



- If there were a shorter path, passing through a vertex *x* outside of *S*
  - that is, $|\gamma_2| + |\gamma_3| < |\gamma_1|$
- Then $|\gamma_2| < |\gamma_1|$
  - thus the vertex *x* should have been selected before *w*

➔ Contradiction

Why Dijkstra's Algorithm Works (2)

- Update of *D[v]* correctly keeps track of the shortest path to vertex *v*

  - [See Aho83]

Complexity

- Depending on how to select the minimum D value?
- Option-1: Scanning the list of all vertices
  - O(n x n) to select next destination for all n vertices
  - O(e) to update D & P values for all e edges with non-infinity cost C[,] (using adjacency lists)

  ➔ Total time: $O(n^2 + e) = O(n^2)$

- Option-2: Using a min-heap of D values
  - O(n log n) for n *DeleteMin* operations
  - O(e log n) for e *PriorityUpdate* operations

➔ Total time: $O((n + e) \log n) = O(e \log n)$

- Better for sparse graph: $O(n \log n)$

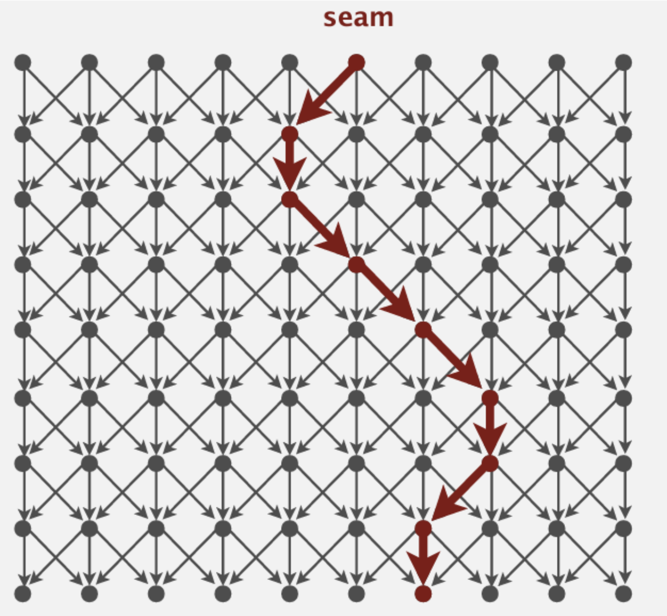
Further Questions

- Dijkstra's algorithm works with cyclic graphs?
  - Yes, why?
- What about negative edges?
  - No, why?
- Could we do better for acyclic graphs?
  - Use topological sort

Edsger W. Dijkstra
Turing award 1972

Application

- Content-aware resizing



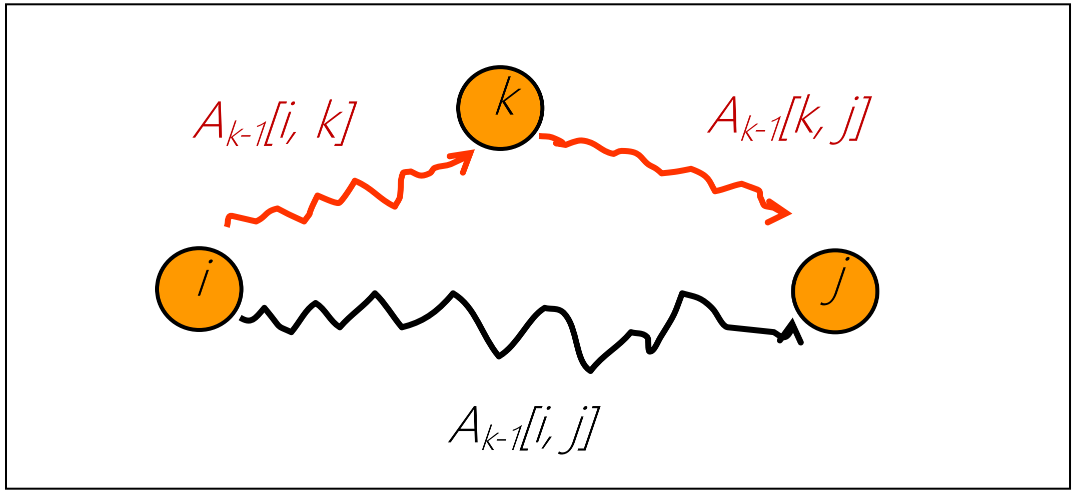
All-Pairs SPP

- One solution is to run Dijkstra's greedy algorithm n times
  - If G is sparse (e = O(n)), this is a good solution
  - Total time (of the min-heap version)
    - $O(n\,[e \log n]) = O(n^2 \log n)$

- Dynamic-programming solution by Floyd
  - Let $A_k[i,j]$ to be the shortest length of any path from vertex i to vertex j, whose intermediate vertices all have indices $\le k$

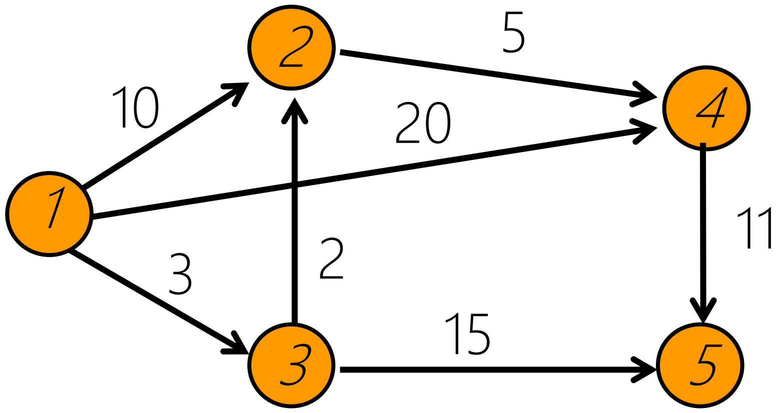
Floyd's Algorithm

- At the *k*-th iteration, we compute $A_k[,]$
  - $A_k[i, j] := \min\{ A_{k-1}[I, j]\ ,\ \ A_{k-1}[i, k] + A_{k-1}[k, j] \}$



- Total time: $O(n^3)$

Example



	1	2	3	4	5
1	0	10	3	20	-
2	-	0	-	5	-
3	-	2	0	-	15
4	-	-	-	0	11
5	-	-	-	-	0

$A_0[i, j]$

	1	2	3	4	5
1					
2					
3					
4					
5					

$A_5[i, j]$

Space Reduction

- $A_k[i, j] := \min\{ A_{k-1}[i, j] , A_{k-1}[i, k] + A_{k-1}[k, j] \}$
  - 3-dimensional space

- In all cases, $A_k[i, j]$ can overwrite $A_{k-1}[i, j]$
  - When i equals k, $A_k[k, j] = A_{k-1}[k, j]$
  - When j equals k, $A_k[i, k] = A_{k-1}[i, k]$
  - When neither i nor j equals k,
    $A_{k-1}[i, j]$ is used only in the computation of $A_k[i, j]$

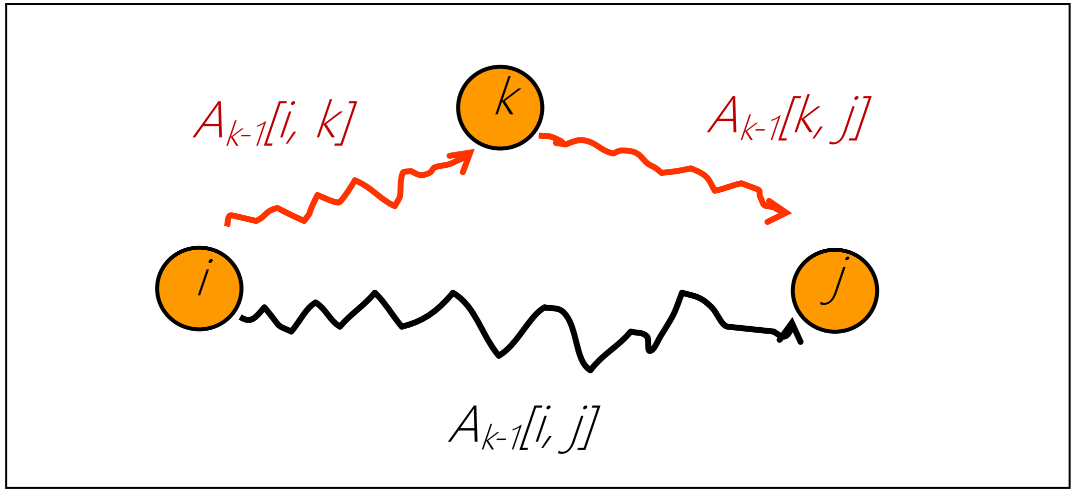
Building The Shortest Paths

[See Aho83]

Building The Shortest Paths

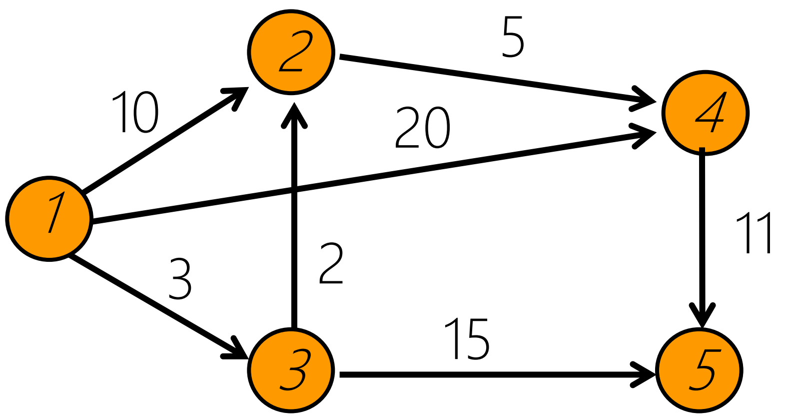
- If the path exists between two nodes then Next[u][v] = v
  else we set Next[u][v] = -1

- if(A[i][j] > A[i][k] + A[k][j])
  {
    A[i][j] = A[i][k] + A[k][j];
    Next[i][j] = Next[i][k];
  }



- path = [u]
  while u != v:
    u = Next[u][v]
    path.append(u)

Example



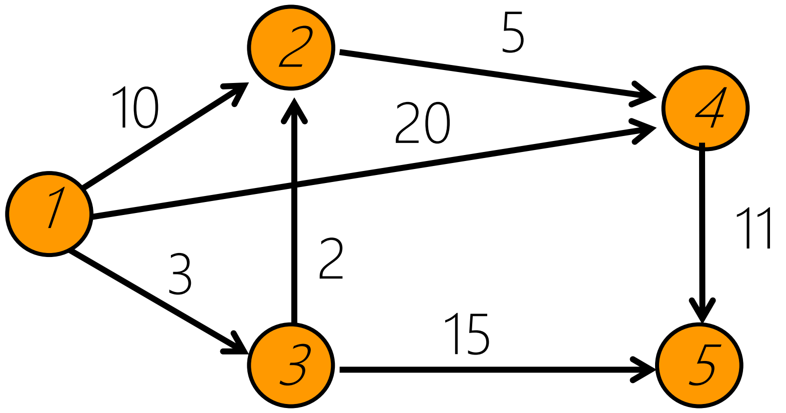
	1	2	3	4	5
1	0	10	3	20	-
2	-	0	-	5	-
3	-	2	0	-	15
4	-	-	-	0	11
5	-	-	-	-	0

$A_0[i, j]$

	1	2	3	4	5
1	-1	2	3	4	-1
2	-1	-1	-1	4	-1
3	-1	2	-1	-1	5
4	-1	-1	-1	-1	5
5	-1	-1	-1	-1	-1

$Next_0[i, j]$

Example



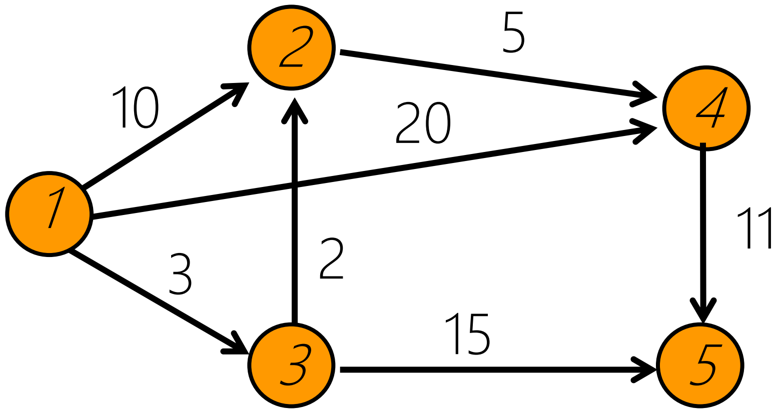
	1	2	3	4	5
1	0	10	3	20	-
2	-	0	-	5	-
3	-	2	0	-	15
4	-	-	-	0	11
5	-	-	-	-	0

$A_1[i, j]$

	1	2	3	4	5
1	-1	2	3	4	-1
2	-1	-1	-1	4	-1
3	-1	2	-1	-1	5
4	-1	-1	-1	-1	5
5	-1	-1	-1	-1	-1

$Next_1[i, j]$

Example



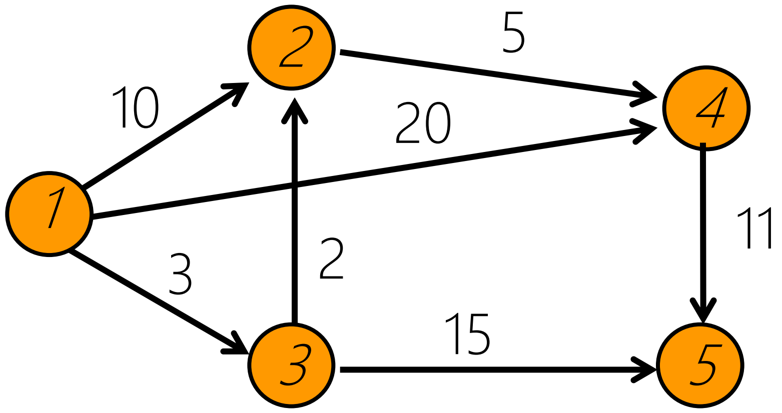
	1	2	3	4	5
1	0	10	3	15	-
2	-	0	-	5	-
3	-	2	0	7	15
4	-	-	-	0	11
5	-	-	-	-	0

$A_2[i, j]$

	1	2	3	4	5
1	-1	2	3	2	-1
2	-1	-1	-1	4	-1
3	-1	2	-1	2	5
4	-1	-1	-1	-1	5
5	-1	-1	-1	-1	-1

$Next_2[i, j]$

Example



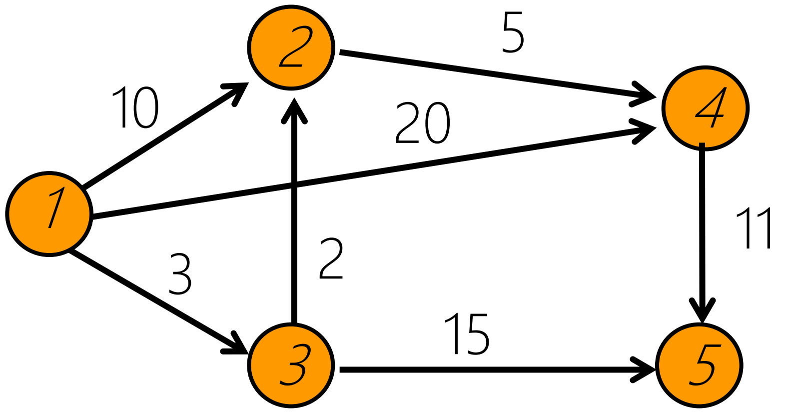
	1	2	3	4	5
1	0	5	3	10	18
2	-	0	-	5	-
3	-	2	0	7	15
4	-	-	-	0	11
5	-	-	-	-	0

$A_3[i, j]$

	1	2	3	4	5
1	-1	3	3	3	3
2	-1	-1	-1	4	-1
3	-1	2	-1	2	5
4	-1	-1	-1	-1	5
5	-1	-1	-1	-1	-1

$Next_3[i, j]$

Example



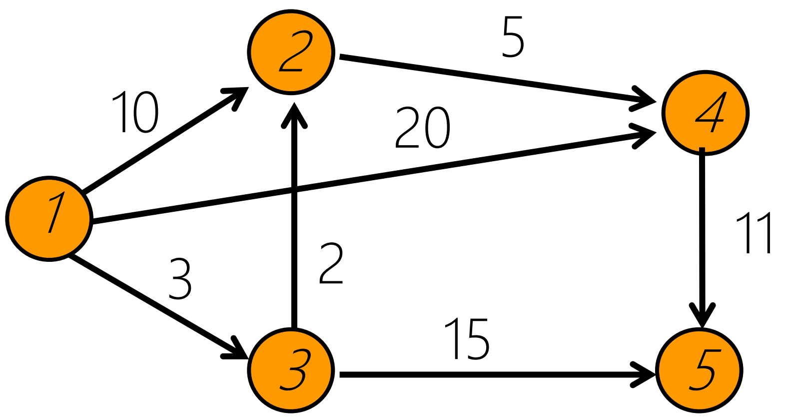
	1	2	3	4	5
1	0	5	3	10	18
2	-	0	-	5	16
3	-	2	0	7	15
4	-	-	-	0	11
5	-	-	-	-	0

$A_4[i, j]$

	1	2	3	4	5
1	-1	3	3	3	3
2	-1	-1	-1	4	4
3	-1	2	-1	2	5
4	-1	-1	-1	-1	5
5	-1	-1	-1	-1	-1

$Next_4[i, j]$

Example



	1	2	3	4	5
1	0	5	3	10	18
2	-	0	-	5	16
3	-	2	0	7	15
4	-	-	-	0	11
5	-	-	-	-	0

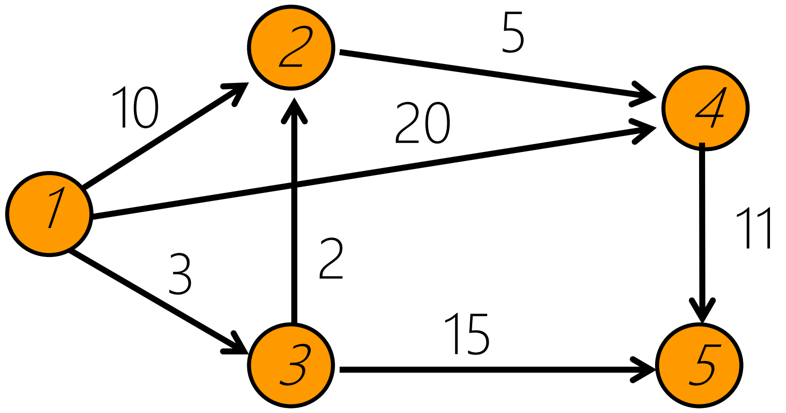
$A_5[i, j]$

	1	2	3	4	5
1	-1	3	3	3	3
2	-1	-1	-1	4	4
3	-1	2	-1	2	5
4	-1	-1	-1	-1	5
5	-1	-1	-1	-1	-1

$Next_5[i, j]$

Example

```
path = [u]
while u != v:
    u = Next[u][v]
    path.append(u)
```



Shortest path from 1 to 4?
Path = [1]

u=Next[1,4]=3
Path = [1,3]

u=Next[3,4]=2
Path = [1,3,2]

u=Next[2,4]=4
Path = [1,3,2,4]

	1	2	3	4	5
1	-1	3	3	3	3
2	-1	-1	-1	4	4
3	-1	2	-1	2	5
4	-1	-1	-1	-1	5
5	-1	-1	-1	-1	-1

$A_5[i, j]$

$Next_5[i, j]$

Comparison of Floyd's with Dijkstra's

- Floyd's is better for a dense graph
  - $O(n^3)$ vs. $O(n^3 \log n)$

- Floyd's algorithm
  - Works even when the graph has a negative edge cost (if there are no negative-length cycles)

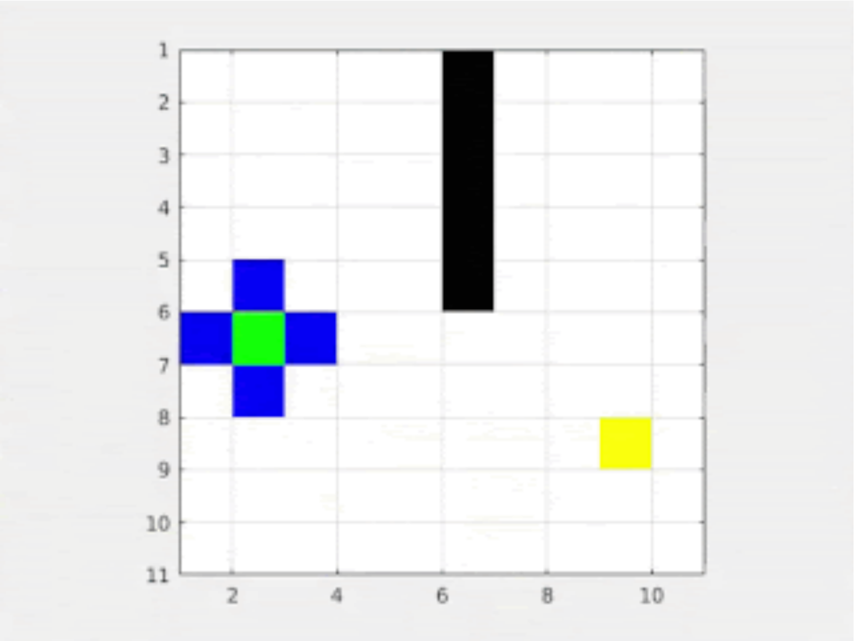
Other Shortest Path Algorithms (1)

- Bellman-Ford algorithm
  - Solves single-source SPP (allowing negative edge costs)
  - Time complexity: O(ne)
    - Slower than Dijkstra’s

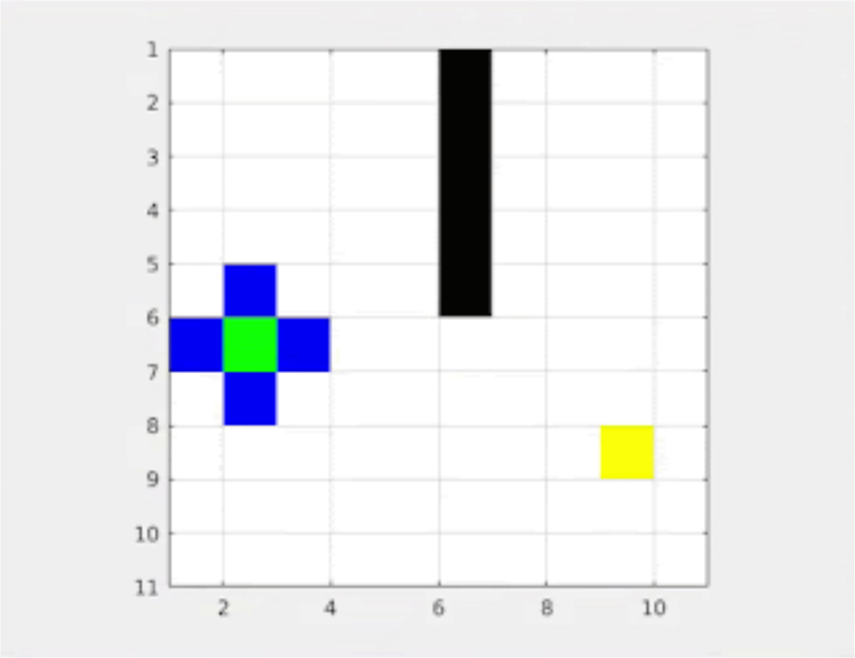
- Johnson's algorithm
  - Solves all-pairs SPP (allowing negative edge costs, but no negative-length cycles)
  - Time complexity: O($n^2 \log n + ne$)
    - May be faster than Floyd’s on sparse graphs

Other Shortest Path Algorithms (2)

- A* search algorithm
  - Solves single-pair SPP
  - Uses heuristics to try to speed up the search



Dijkstra



A*

References

- Further reading list and references
  - https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
  - https://www.geeksforgeeks.org/a-search-algorithm/

- Slide credit
  - Jaesik Park
  - Seung-Hwan Baek
  - Jong-Hyeok Lee