Shortest Path Problems I

AU4606: Network Optimization

AI4702: Network Intelligence and Optimization

Xiaoming Duan
Department of Automation
Shanghai Jiao Tong University

September 25, 2023

Last time

- General search algorithms
  - Forward search
  - Reverse search
- Particular search algorithms
  - Breadth-first search
  - Depth-first search
- Applications
  - Strong connectivity
  - Topological ordering
  - Determining bipartite graphs
  - Finding Eulerian circuits in undirected graphs

Today

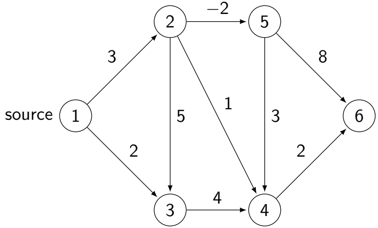
1. Shortest path problems: formulation
   - What is a shortest path problem
   - An “unusual” application
2. Algorithms for shortest path problems
   - Properties
   - Algorithms for acyclic graphs
   - Dijkstra’s algorithm

Today

1. Shortest path problems: formulation
   - What is a shortest path problem
   - An "unusual" application

2. Algorithms for shortest path problems
   - Properties
   - Algorithms for acyclic graphs
   - Dijkstra's algorithm

What is a shortest path problem?



Given

- A directed network $G = (N, A)$
- Each arc $(i, j) \in A$ has an arc length c_{ij}
- A source node $s \in N$

Define

- Length of a directed path as the sum of the lengths of arcs in the path

Determine

- A shortest length directed path from s to every other node

Why do we study shortest path problems?

- They arise frequently in practice in a wide variety of applications
  1. Network routing
  2. Transportation logistics
  3. Social network analysis
- They are easy to solve efficiently
  - Solvable in polynomial time with low degrees
- They capture many of core ingredients of network optimization
  1. Network modeling
  2. Algorithm design and analysis
- They arise frequently as subproblems when solving other problems
  - Network flow optimization

Types of shortest path problems and assumptions

Different algorithms are developed to solve

1. Problems with nonnegative arc lengths (our focus today)
2. Problems with negative arc lengths (upcoming next)

Types of shortest path problems and assumptions

Different algorithms are developed to solve

1. Problems with nonnegative arc lengths (our focus today)
2. Problems with negative arc lengths (upcoming next)

Assumptions

- All arc lengths are integers
- Network contains a directed path from node s to every other node
- The network does not contain a negative cycle
- The network is directed

An application: approximating piecewise linear functions

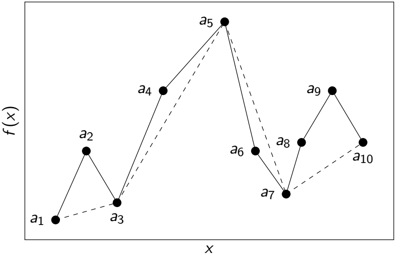
- Let $f(x)$ be a piecewise linear function of a scalar variable x
- It passes through n points $\{a_i = (x_i, f(x_i))\}_{i \in \{1,\ldots,n\}}$

Find "best" approximation $f'(x)$ with some of $\{a_i\}_{i \in \{2,\ldots,n-1\}}$?

1. Whenever a point a_i is used, a fixed cost α is incurred
2. Define approximation error as

$$\text{error} = \sum_{i=2}^{n-1} (f(x_i) - f'(x_i))^2$$

3. Total penalty is the sum of the fixed costs plus approximation error



Today

1. Shortest path problems: formulation
   - What is a shortest path problem
   - An “unusual” application

2. Algorithms for shortest path problems
   - Properties
   - Algorithms for acyclic graphs
   - Dijkstra’s algorithm

Two properties of shortest path problems

Subpaths are shortest

If the path $s = i_l - i_2 - \cdots - i_h = k$ is a shortest path from node s to node k, then for every $q \in \{2, 3, ..., h-1\}$, the subpath $s = i_l - i_2 - \cdots - i_q$ is a shortest path from the source node to node i_q.

The above property implies that if we store a shortest path $s = i_l - i_2 - \cdots - i_h = k$, then we also store shortest paths from s to all nodes i_q for $q \in \{2, 3, ..., h-1\}$

Two properties of shortest path problems

Subpaths are shortest

If the path $s = i_l - i_2 - \cdots - i_h = k$ is a shortest path from node s to node k, then for every $q \in \{2, 3, ..., h-1\}$, the subpath $s = i_l - i_2 - \cdots - i_q$ is a shortest path from the source node to node i_q.

The above property implies that if we store a shortest path $s = i_l - i_2 - \cdots - i_h = k$, then we also store shortest paths from s to all nodes i_q for $q \in \{2, 3, ..., h-1\}$

- Let $d(i)$ be the shortest distance from s to node i

Distance labels

Let the vector d represent the shortest path distances. Then a directed path P from the source node to node k is a shortest path if and only if $d(j) = d(i) + c_{ij}$ for every arc $(i, j) \in P$.

Two properties of shortest path problems

Subpaths are shortest

If the path $s = i_l - i_2 - \cdots - i_h = k$ is a shortest path from node s to node k, then for every $q \in \{2, 3, ..., h-1\}$, the subpath $s = i_l - i_2 - \cdots - i_q$ is a shortest path from the source node to node i_q.

The above property implies that if we store a shortest path $s = i_l - i_2 - \cdots - i_h = k$, then we also store shortest paths from s to all nodes i_q for $q \in \{2, 3, ..., h-1\}$

- Let $d(i)$ be the shortest distance from s to node i

Distance labels

Let the vector d represent the shortest path distances. Then a directed path P from the source node to node k is a shortest path if and only if $d(j) = d(i) + c_{ij}$ for every arc $(i, j) \in P$.

Distance labels are the key to finding shortest distance paths

Shortest path trees

How do we store shortest paths in a space efficient way?

- Naively: store every shortest path to each node, $O(n^2)$ space

Shortest path trees

How do we store shortest paths in a space efficient way?

- Naively: store every shortest path to each node, $O(n^2)$ space
- Store a "shortest path tree"
  1. There is a shortest path from source to every node

Shortest path trees

How do we store shortest paths in a space efficient way?

- Naively: store every shortest path to each node, $O(n^2)$ space
- Store a "shortest path tree"
  1. There is a shortest path from source to every node
  2. Can assign each node a distance label indicating shortest path distance

Shortest path trees

How do we store shortest paths in a space efficient way?

- Naively: store every shortest path to each node, $O(n^2)$ space
- Store a "shortest path tree"
  1. There is a shortest path from source to every node
  2. Can assign each node a distance label indicating shortest path distance
  3. Consider the spanning subgraph $G' = (N, A')$ where

$$A' = \{(i, j) \in A \mid d(j) = d(i) + c_{ij}\}$$

Shortest path trees

How do we store shortest paths in a space efficient way?

- Naively: store every shortest path to each node, $O(n^2)$ space
- Store a "shortest path tree"
  1. There is a shortest path from source to every node
  2. Can assign each node a distance label indicating shortest path distance
  3. Consider the spanning subgraph $G' = (N, A')$ where

$$A' = \{(i,j) \in A \,|\, d(j) = d(i) + c_{ij}\}$$

  4. G' must contain paths from source to every other node

Shortest path trees

How do we store shortest paths in a space efficient way?

- Naively: store every shortest path to each node, $O(n^2)$ space
- Store a "shortest path tree"
  1. There is a shortest path from source to every node
  2. Can assign each node a distance label indicating shortest path distance
  3. Consider the spanning subgraph $G' = (N, A')$ where

$$A' = \{(i,j) \in A \,|\, d(j) = d(i) + c_{ij}\}$$

  4. G' must contain paths from source to every other node
  5. A breadth-first search on G' results in a "shortest path tree"

Shortest paths in acyclic graphs: pulling algorithms

Recall definition of topological ordering

- A labeling "order" that satisfies $\text{order}(i) < \text{order}(j)$ for $(i, j) \in A$

A graph is acyclic if and only if it possesses a topological ordering

- Assume the nodes are already labeled in this fashion

Shortest paths in acyclic graphs: pulling algorithms

Recall definition of topological ordering

- A labeling "order" that satisfies $\text{order}(i) < \text{order}(j)$ for $(i, j) \in A$

A graph is acyclic if and only if it possesses a topological ordering

- Assume the nodes are already labeled in this fashion

Suppose we already found the shortest paths to nodes $1, \ldots, k-1$, consider node k

- All incoming arcs to node k can only emanate from $\{1, \ldots, k-1\}$
- All paths to k must pass through one of its in-neighbors

Thus, shortest path distance to k is simply $\min_{(j,k)\in A}\{d(j) + c_{jk}\}$

1. Set $d(s) = 0$ and $d(i) = \infty$ for $i \in N \setminus \{s\}$
2. Follow the topological ordering, examine node k: set $d(k) = \min_{(j,k)\in A}\{d(j) + c_{jk}\}$

Shortest paths in acyclic graphs: pulling algorithms

Recall definition of topological ordering

- A labeling "order" that satisfies order$(i) <$ order(j) for $(i, j) \in A$

A graph is acyclic if and only if it possesses a topological ordering

- Assume the nodes are already labeled in this fashion

Suppose we already found the shortest paths to nodes $1, \ldots, k-1$, consider node k

- All incoming arcs to node k can only emanate from $\{1, \ldots, k-1\}$
- All paths to k must pass through one of its in-neighbors

Thus, shortest path distance to k is simply $\min_{(j,k) \in A}\{d(j) + c_{jk}\}$

1. Set $d(s) = 0$ and $d(i) = \infty$ for $i \in N \setminus \{s\}$
2. Follow the topological ordering, examine node k: set $d(k) = \min_{(j,k) \in A}\{d(j) + c_{jk}\}$

Need to store all in-neighbors

Shortest paths in acyclic graphs: reaching algorithms

Considering the following procedure

1. Set $d(s) = 0$ and $d(i) = \infty$ for $i \in N \setminus \{s\}$
2. Following the topological ordering, examine node i: for $(i, j) \in A$, set $d(j) = d(i) + c_{ij}$ when $d(j) > d(i) + c_{ij}$

The reaching algorithm finds optimal distance labels

The reaching algorithm solves the shortest path problem on acyclic networks in $O(m)$ time.

Shortest paths trees of pulling/reaching algorithms

For pulling

- When updating $d(k) = \min_{(j,k)\in A}\{d(j) + c_{jk}\}$, set $\text{pred}(k) = j^*$ where

$$j^* \in \underset{j}{\text{argmin}}\{d(j) + c_{jk}\}$$

- A shortest path tree can be built inductively
  1. The second node must only have source node as its in-neighbor
  2. Suppose for first $k-1$ nodes, a path exists to them
  3. For the k-th node, a path exists since predecessor is one of in-neighbors
- No cycles exist since connected and only $n-1$ edges

For reaching

- When examine j and update $d(k) = d(j) + c_{kj}$, set $\text{pred}(k) = j$
- A shortest path tree can be built inductively
  1. When examine the source node, we have path to the second node
  2. Suppose when examining first $k-1$ nodes, a path exists to them
  3. For k-th node, path exists since its in-neighbors were already examined
- No cycles exist since connected and only $n-1$ edges

Shortest paths in general graphs: Dijkstra's algorithm

Why is finding distance labels more difficult in general graphs?

- No natural order to update the distance labels of nodes

Why is finding distance labels more difficult in general graphs?

- No natural order to update the distance labels of nodes

How about this?

1. The source node has the correct distance label $d(s) = 0$

Shortest paths in general graphs: Dijkstra's algorithm

Why is finding distance labels more difficult in general graphs?

- No natural order to update the distance labels of nodes

How about this?

1. The source node has the correct distance label $d(s) = 0$
2. Update the distance label of its out-neighbors

Shortest paths in general graphs: Dijkstra's algorithm

Why is finding distance labels more difficult in general graphs?

- No natural order to update the distance labels of nodes

How about this?

1. The source node has the correct distance label $d(s) = 0$
2. Update the distance label of its out-neighbors
3. Select a node with the smallest distance label in the graph

Shortest paths in general graphs: Dijkstra's algorithm

Why is finding distance labels more difficult in general graphs?

- No natural order to update the distance labels of nodes

How about this?

1. The source node has the correct distance label $d(s) = 0$
2. Update the distance label of its out-neighbors
3. Select a node with the smallest distance label in the graph
4. Udate the distance label of its out-neighbors

Shortest paths in general graphs: Dijkstra's algorithm

Why is finding distance labels more difficult in general graphs?

- No natural order to update the distance labels of nodes

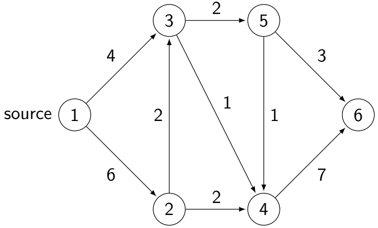
How about this?

1. The source node has the correct distance label $d(s) = 0$
2. Update the distance label of its out-neighbors
3. Select a node with the smallest distance label in the graph
4. Udate the distance label of its out-neighbors
5. Repeat from step 3

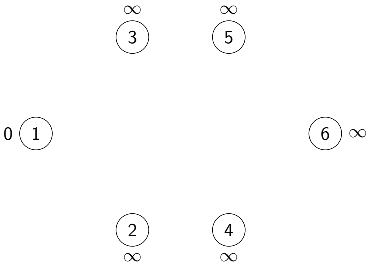
Intuition: we "believe" nodes with smallest label are correctly labeled

Algorithm Dijkstra's algorithm

$S \leftarrow \emptyset$, $\bar{S} \leftarrow N$
$d(i) \leftarrow \infty$ for each node $i \in N$
$d(s) \leftarrow 0$ and pred$(s) \leftarrow 0$
while $|S| < n$ **do**
  Let $i \in \bar{S}$ be a node for which $d(i) = \min_{j \in \bar{S}} d(j)$
  $S = S \cup \{i\}$
  $\bar{S} = \bar{S} \setminus \{i\}$
  for Each $(i, j) \in A$ **do**
    if $d(j) > d(i) + c_{ij}$ **then**
      $d(j) \leftarrow d(i) + c_{ij}$
      prd$(j) \leftarrow i$
    end if
  end for
end while



(a) A directed graph

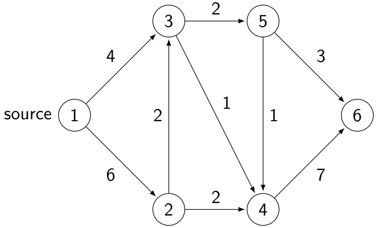


(b) Dijkstra's example

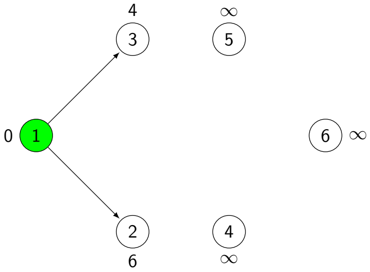
1. $d(i) \leftarrow \infty$ for each node $i \in N$
2. $d(1) \leftarrow 0$ and $\text{pred}(1) \leftarrow 0$

$S = \emptyset,\ \bar{S} = \{1, 2, 3, 4, 5, 6\}$

Dijkstra's algorithm: running example



(a) A directed graph

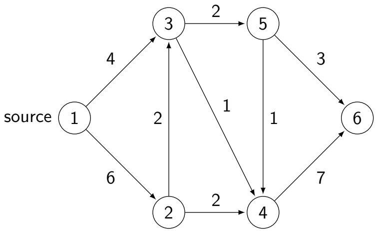


(b) Dijkstra's example

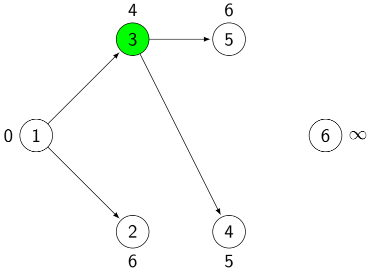
$S = \emptyset,\ \bar{S} = \{1,2,3,4,5,6\}$

1. Pick node 1 since $d(1) = \min_{j \in \bar{S}} d(j)$
2. $S \leftarrow S\{1\},\ \bar{S} \leftarrow \bar{S} \setminus \{1\}$
3. Update $d(2) \leftarrow d(1) + c_{12} = 6,\ \text{pred}(2) = 1$
4. Update $d(3) \leftarrow d(1) + c_{13} = 4,\ \text{pred}(3) = 1$

$S = \{1\},\ \bar{S} = \{2,3,4,5,6\}$



(a) A directed graph



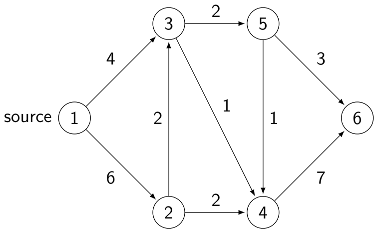
(b) Dijkstra's example

$S = \{1\}$, $\bar{S} = \{2, 3, 4, 5, 6\}$

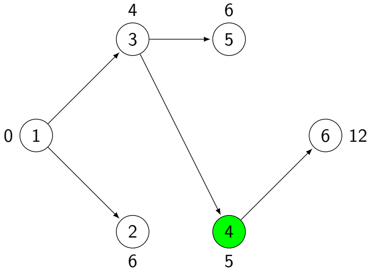
1. Pick node 3 since $d(3) = \min_{j \in \bar{S}} d(j)$
2. $S \leftarrow S\{3\}$, $\bar{S} \leftarrow \bar{S} \setminus \{3\}$
3. Update $d(4) \leftarrow d(3) + c_{34} = 5$, $\text{pred}(4) = 3$
4. Update $d(5) \leftarrow d(3) + c_{35} = 6$, $\text{pred}(5) = 3$

$S = \{1, 3\}$, $\bar{S} = \{2, 4, 5, 6\}$

Dijkstra's algorithm: running example



(a) A directed graph



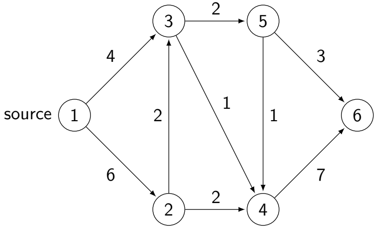
(b) Dijkstra's example

$S = \{1, 3\}$, $\bar{S} = \{2, 4, 5, 6\}$

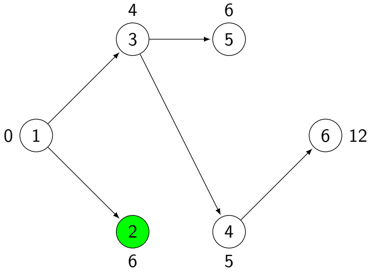
1. Pick node 4 since $d(4) = \min_{j \in \bar{S}} d(j)$
2. $S \leftarrow S\{4\}$, $\bar{S} \leftarrow \bar{S} \setminus \{4\}$
3. Update $d(6) \leftarrow d(4) + c_{46} = 12$, $\text{pred}(6) = 4$

$S = \{1, 3, 4\}$, $\bar{S} = \{2, 5, 6\}$

Dijkstra's algorithm: running example



(a) A directed graph

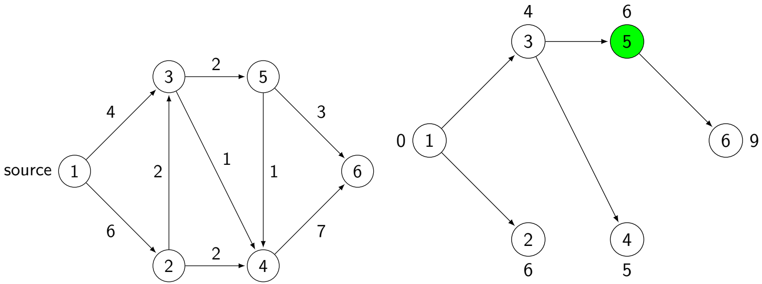


(b) Dijkstra's example

$S = \{1, 3, 4\}$, $\bar{S} = \{2, 5, 6\}$

1. Pick node 2 since $d(2) = \min_{j \in \bar{S}} d(j)$
2. $S \leftarrow S\{2\}$, $\bar{S} \leftarrow \bar{S} \setminus \{2\}$

$S = \{1, 2, 3, 4\}$, $\bar{S} = \{5, 6\}$

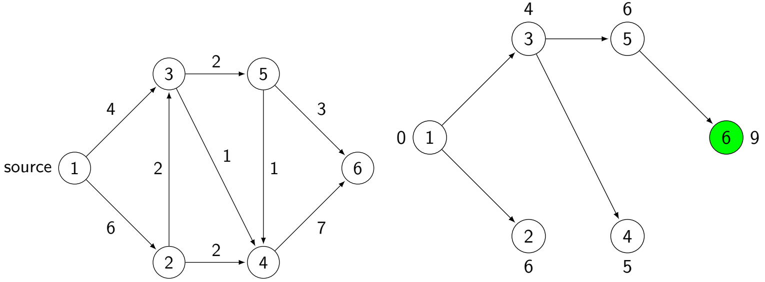


(a) A directed graph (b) Dijkstra's example

$S = \{1,2,3,4\}$, $\bar{S} = \{5,6\}$

1. Pick node 5 since $d(5) = \min_{j \in \bar{S}} d(j)$
2. $S \leftarrow S\{5\}$, $\bar{S} \leftarrow \bar{S} \setminus \{5\}$
3. Update $d(6) \leftarrow d(5) + c_{56} = 9$, $\text{pred}(6) = 5$

$S = \{1,2,3,4,5\}$, $\bar{S} = \{6\}$



(a) A directed graph (b) Dijkstra's example

$S = \{1, 2, 3, 4, 5\}$, $\bar{S} = \{6\}$

1. Pick node 6 since $d(6) = \min_{j \in \bar{S}} d(j)$
2. $S \leftarrow S\{6\}$, $\bar{S} \leftarrow \bar{S} \setminus \{6\}$

$S = \{1, 2, 3, 4, 5, 6\}$, $\bar{S} = \emptyset$

Dijkstra's algorithm: analysis

Correctness of Dikstra algorithm

Given a directed network $G = (N, A)$ with nonnegative arc costs, Dijkstra's algorithm correctly determines the shortest path distances from the source s to every node in the network.

Dijkstra's algorithm: analysis

Correctness of Dikstra algorithm

Given a directed network $G = (N, A)$ with nonnegative arc costs, Dijkstra's algorithm correctly determines the shortest path distances from the source s to every node in the network.

Time complexity

- $O(n)$ iterations and each iteration finds a minimum among nodes, examine $O(m)$ total edges, $O(n^2 + m) = O(n^2)$
- Using d-heaps $O(nd \log_d n + m \log_d n)$
  1. Build d-heap with n elements $O(n \log_d n)$ (adding elements and sift up)
  2. Delete n mins, each takes $O(d \log_d n)$ (replace with leaf, sift down)
  3. Decrease keys $O(m)$ times, each takes $\log_d n$ (sift up)

Dijkstra's algorithm: analysis

Correctness of Dikstra algorithm

Given a directed network $G = (N, A)$ with nonnegative arc costs, Dijkstra's algorithm correctly determines the shortest path distances from the source s to every node in the network.

Time complexity

- $O(n)$ iterations and each iteration finds a minimum among nodes, examine $O(m)$ total edges, $O(n^2 + m) = O(n^2)$
- Using d-heaps $O(nd \log_d n + m \log_d n)$
  1. Build d-heap with n elements $O(n \log_d n)$ (adding elements and sift up)
  2. Delete n mins, each takes $O(d \log_d n)$ (replace with leaf, sift down)
  3. Decrease keys $O(m)$ times, each takes $\log_d n$ (sift up)

Shortest path tree

Given a directed network $G = (N, A)$ with nonnegative arc costs, Dijkstra's algorithm builds a shortest path tree.

Similarities between Dijkstra's algorithm and BFS

What if all the arc lengths are ones???

Similarities between Dijkstra's algorithm and BFS

What if all the arc lengths are ones???

Dijkstra's algorithm becomes BFS!

Parameter balancing

Recall the time complexity of Dijkstra's algorithm using d-heaps

$$O(nd \log_d n + m \log_d n)$$

What choice of d is optimal? Differentiating?

Recall the time complexity of Dijkstra's algorithm using d-heaps

$$O(nd \log_d n + m \log_d n)$$

What choice of d is optimal? Differentiating?

Parameter balancing

Suppose the running time of some algorithm is $O(f(m,n,k) + g(m,n,k))$ where k is a parameter. Further suppose $f(m,n,k) \geq 0$ and $g(m,n,k) \geq 0$ are increasing and decreasing functions of k.

$$f(m,n,k^*) + g(m,n,k^*) \leq 2 \min_k \{f(m,n,k) + g(m,n,k)\}$$

where k^* is such that $f(m,n,k^*) = g(m,n,k^*)$.

In the Dijkstra's algorithm case, $d^* = \max\{\frac{m}{n}, 2\}$.

Reverse Dijkstra's and bidirectional Dijkstra's algorithms

Suppose want to find shortest paths to a destination node t

1. Initialize $d(t) = 0$
2. When node j with smallest label is selected, scan incoming arcs
3. Update distance label of node i to $\min\{d(i), c_{ij} + d(j)\}$ for $(i, j) \in A$

Reverse Dijkstra's and bidirectional Dijkstra's algorithms

Suppose want to find shortest paths to a destination node t

1. Initialize $d(t) = 0$
2. When node j with smallest label is selected, scan incoming arcs
3. Update distance label of node i to $\min\{d(i), c_{ij} + d(j)\}$ for $(i, j) \in A$

Suppose want to find shortest paths from node s to node t

- Run forward Dijkstra's algorithm form s and reverse one from t
- Stop when there exists a node k selected by both algorithm
- The shortest distance can be
  1. s to k + k to t
  2. s to some i labeled by forward alg. + (i, j) + some j labeled by reverse alg. to t

Upcoming

Week 1-8 (AU4606 & AI4702):

- Introduction (1 lecture)
- Preparations (3 lectures)
  - basics of graph theory
  - algorithm complexity and data structure
  - graph search algorithm
- Shortest path problems (this & next lecture)
- Maximum flow problems (5 lectures)
- Minimum cost flow problems (3 lectures)
- Introduction to multi-agent systems (1 lecture)
- Introduction to cloud networks (1 lecture)

Week 9-16 (AU4606):

- Simplex and network simplex methods (2 lectures)
- Global minimum cut problems (3 lectures)
- Minimum spanning tree problems (3 lectures)