15.082 and 6.855J
February 20, 2003

Dijkstra's Algorithm for the Shortest Path Problem

Wide Range of Shortest Path Problems

- Sources and Destinations
  - We will consider single source problems in this lecture
- Properties of the costs.
  - We will consider non-negative cost coefficients in this lecture
- Network topology.
  - We will consider all directed graphs

Assumptions for the Problem Today

- Integral, non-negative data
- There is a directed path from source node s to all other nodes.
- Objective: find the shortest path from node s to each other node.
- Applications.
  - Vehicle routing
  - Communication systems

Overview of today's lecture

- One nice application (see the book for more)

- Dijkstra's algorithm
  - animation
  - proof of correctness (invariants)
  - time bound

- Dial's algorithm (a way of implementing Dijkstra's algorithm)
  - animation
  - time bound

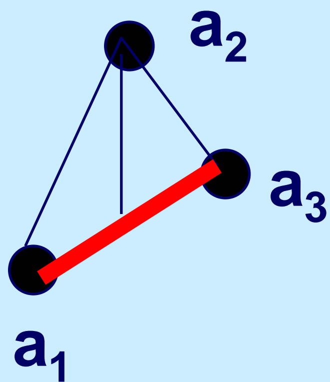
Approximating Piecewise Linear Functions

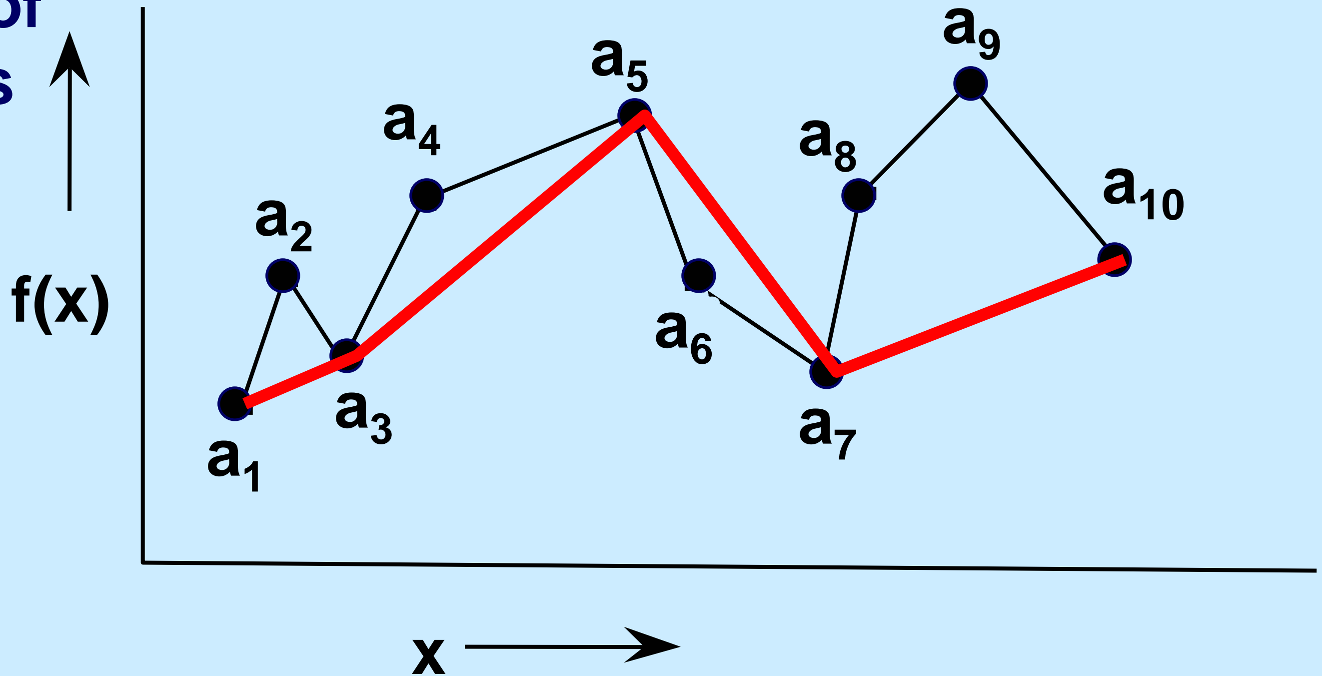
- **INPUT: A piecewise linear function**
  - n points $a_1 = (x_1,y_1)$, $a_2 = (x_2,y_2)$,..., $a_n = (x_n,y_n)$.
  - $x_1 \leq x_2 \leq \cdots \leq x_n$.

- **Objective: approximate f with fewer points**
  - c* is the “cost” per point included
  - c_{ij} = cost of approximating the function through points a_i, a_{i+1}, . . ., a_{jj} by a single line joining point a_i to point a_j.

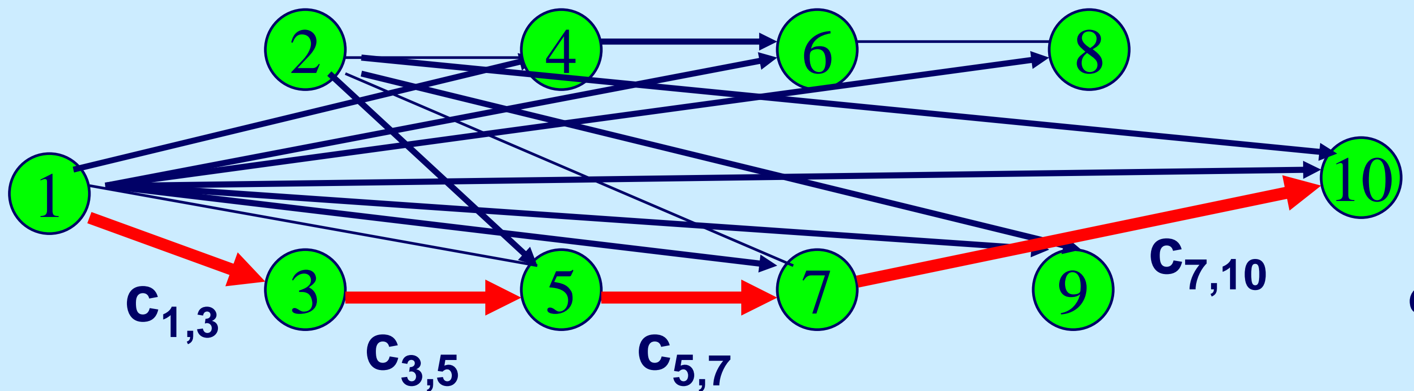
  - Find the minimum cost path from node 1 to node n.
  - Each path from 1 to n corresponds to an approximation of the data points a_1 to a_n.

$c_{i,j}$ is the cost of deleting points $a_{i+1}, \ldots, a_{j-1}$





e.g., $c_{1,3}$ = -c* + dist of a_2 to the line



A Key Step in Shortest Path Algorithms

- In this lecture, and in subsequent lectures, we let d( ) denote a vector of temporary distance labels.
- d(i) is the length of some path from the origin node 1 to node i.
- **Procedure Update(i)**

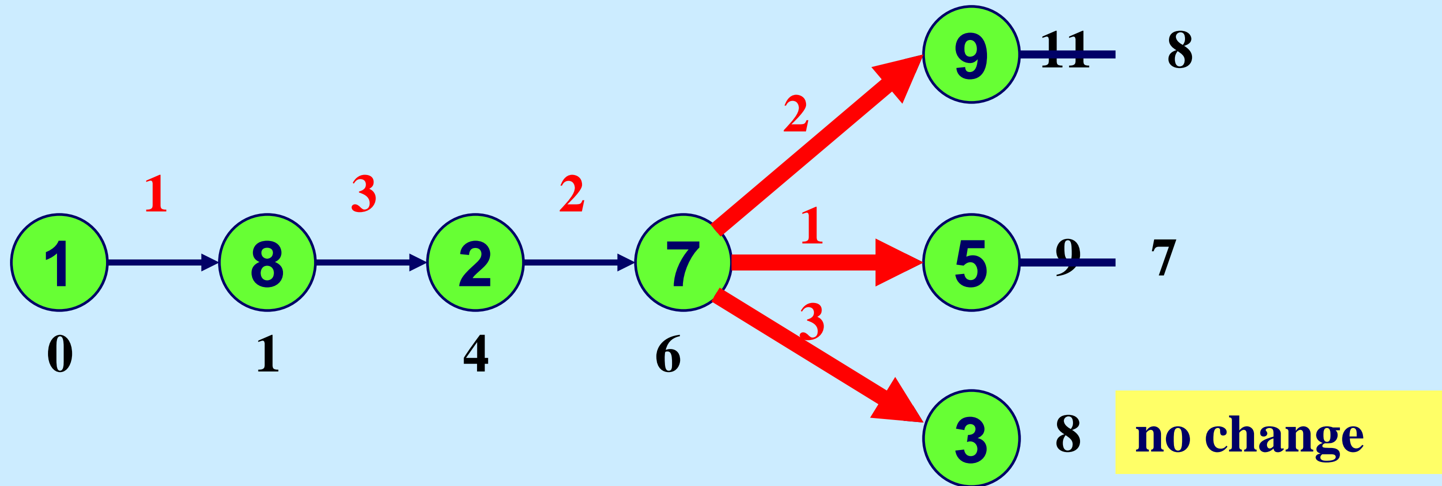
  for each $(i,j) \in A(i)$ ***do***

  if $d(j) > d(i) + c_{ij}$ ***then*** $d(j) := d(i) + c_{ij}$ and pred(j) : = i;

- Update(i)
- used in Dijkstra's algorithm and in the label correcting algorithm

Update(7)

d(7) = 6 at some point in the algorithm, because of the path 1-8-2-7

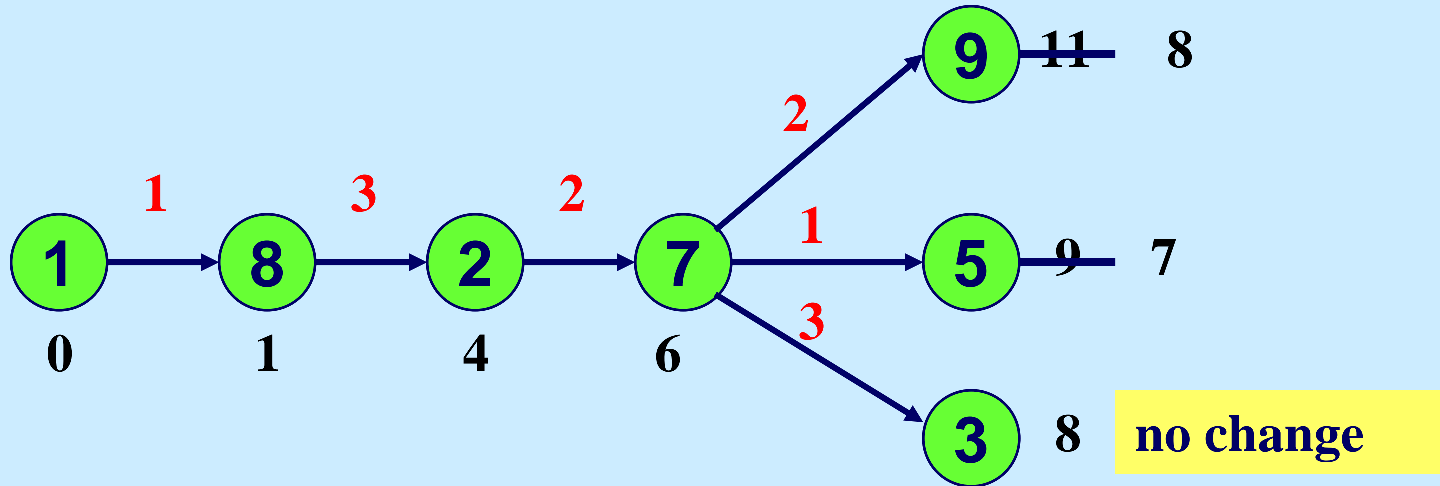


Suppose 7 is incident to nodes 9, 5, 3, with temporary distance labels as shown.

We now perform Update(7).

On Updates

Note: distance labels cannot increase in an update step. They can decrease.



We do not need to perform Update(7) again, unless d(7) decreases. Updating sooner could not lead to further decreases in distance labels.

In general, if we perform Update(j), we do not do so again unless d(j) has decreased.

Dijkstra's Algorithm

Let $d^*(j)$ denote the shortest path distance from node 1 to node j.

Dijkstra's algorithm will determine $d^*(j)$ for each j, in order of increasing distance from the origin node 1.

S denotes the set of *permanently labeled* nodes.

That is, $d(j) = d^*(j)$ for $j \in S$.

T denotes the set of *temporarily labeled* nodes.

That is, $d(j) \geq d^*(j)$ for $j \in T$.

Dijkstra's Algorithm

begin

S : = {1} ;   T = N – {1};

d(1) : = 0 and pred(1) : = 0; d(j) = ∞ for j = 2 to n;

update(1);

while S ≠ N ***do***

begin (***node selection**, **also called FINDMIN***)

let i ∈ T be a node for which

d(i) = min {d(j) : j ∈ T};

S : = S ∪ {i}; T: = T – {i};

Update(i)

end;

end;

Dijkstra's Algorithm Animated

Why Does Dijkstra's Algorithm Work?

A standard method for proving correctness.

1. **Determine things that are true at each iteration. These are called *invariants*.**
2. **Prove invariants using induction**
3. **Prove that the algorithm is finite**
4. **Choose invariants so that the algorithm's correctness follows directly from the invariants and the fact that the algorithm terminates.**

Why Does Dijkstra's Algorithm Work?

Invariants for Dijkstra's Algorithm

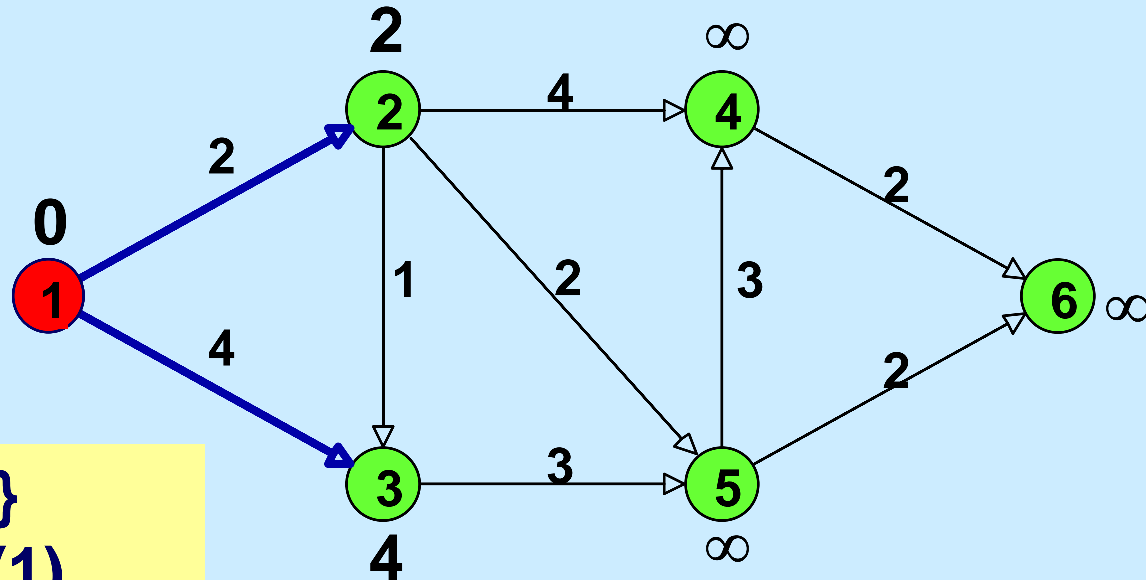
1. If $j \in S$, then $d(j)$ is the shortest distance from node 1 to node j.

2. If $i \in S$, and $j \in T$, then $d(i) \leq d(j)$.

3. If $j \in T$, then $d(j)$ is the length of the shortest path from node 1 to node j in $S \cup \{j\}$.

Note: S increases by one node at a time. So, at the end the algorithm is correct by invariance 1.

Verifying invariants when S = { 1 }



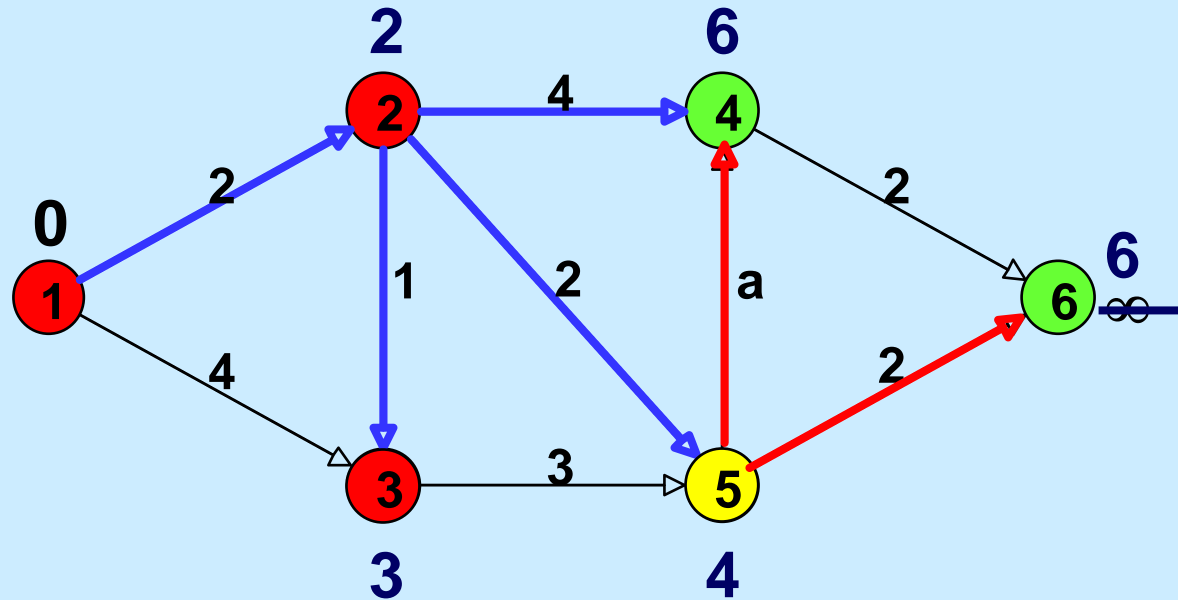
Consider S = { 1 } and after update(1)

1. If $j \in S$, then d(j) is the shortest distance from node 1 to node j.
2. If $i \in S$, and $j \in T$, then $d(i) \leq d(j)$.
3. If $j \in T$, then d(j) is the length of the shortest path from node 1 to node j in $S \cup \{j\}$.

Verifying invariants Inductively

Assume that the invariants are true before the update.

Node 5 was just selected. It is now transferred from T to S.

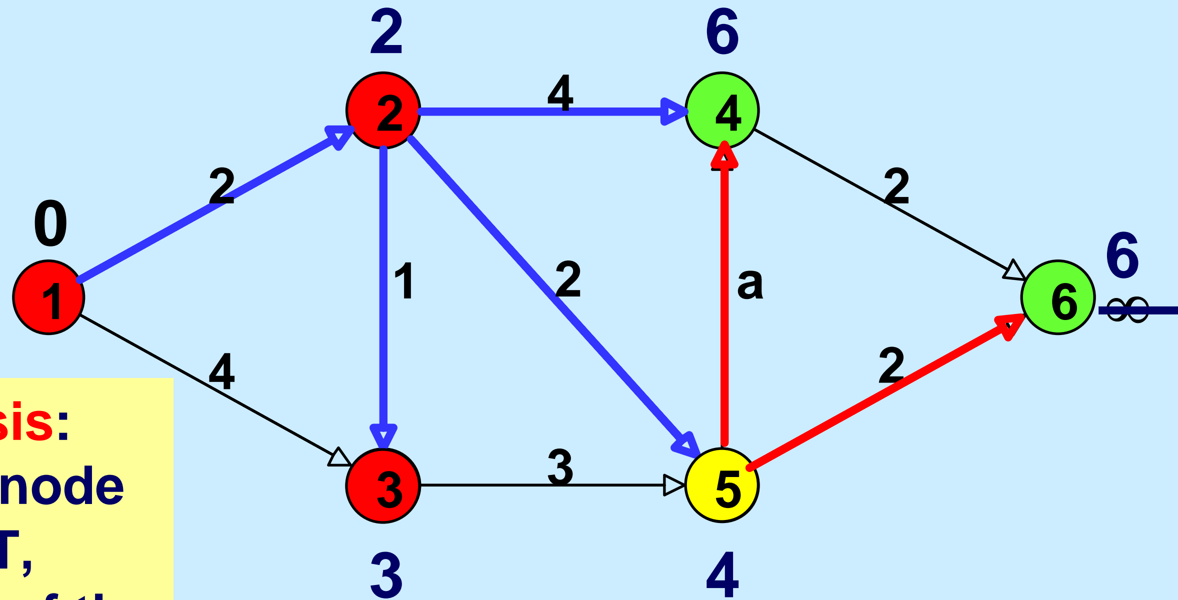


2. If i $\in$ S, and j $\in$ T, then d(i) $\leq$ d(j).

If this was true before node 5 was transferred, it remains true afterwards.

Verifying invariants Inductively

To show: 1. If j ∈ S, then d(j) is the shortest distance from node 1 to node j.



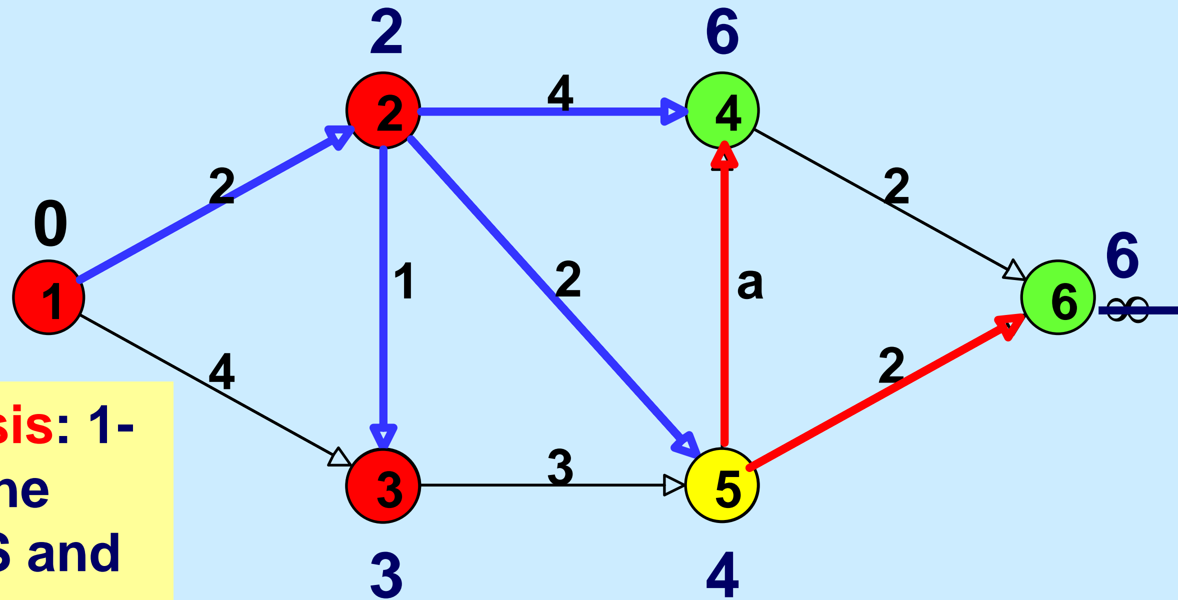
By inductive hypothesis: Before the transfer of node j* (node 5) to S, if k ∈ T, then d(k) is the length of the shortest path from node 1 to node k in S ∪ {k}.

The result is clearly true for nodes in S prior to transferring node j*. We need to prove it for j* = 5. Any path from node 1 to node j* has a first node k in T prior to the transfer, and the path from 1 to k has length at least d(k) ≥ d(j*). So, the shortest path to j* has length at least d(j*).

Verifying invariants Inductively

To show: 3. If j ∈ T, then d(j) is the length of the shortest path from node 1 to node j in S ∪ {j}.

By inductive hypothesis: 1-3 are all true prior to the transfer of node j* to S and prior to update(j*).



Consider node k ∈ T (e.g., node 4). By hypothesis, d(4) is the shortest path length from 1 to 4 in G restricted to {1, 2, 3, 4}. Let d'(4) be the value after update(5). We want to show that d'(4) is the length of the shortest path P from 1 to 4 in G as restricted to {1, 2, 3, 4, 5}.

If P does not contain node 5, then d'(4) = d(4) and the result is true.

If P does contain node 5, then 5 immediately precedes node 4 and the result is true.

A comment on invariants

It is the standard way to prove that algorithms work.

- Finding the best invariants for the proof is often challenging.

- A reasonable method. Determine what is true at each iteration (by carefully examining several useful examples) and then use all of the invariants.

- Then shorten the proof later.

Complexity Analysis of Dijkstra's Algorithm

- **Update Time:** update(j) occurs once for each j, upon transferring j from T to S. The time to perform all updates is O(m) since the arc (i,j) is only involved in update(i).

- **FindMin Time:** To find the minimum (in a straightforward approach) involves scanning d(j) for each $j \in T$.
  - Initially T has n elements.
  - So the number of scans is n + n-1 + n-2 + … + 1 = $O(n^2)$.

- **$O(n^2)$ time in total.** This is the best possible only if the network is *dense*, that is m is about n^2.

- We can do better if the network is *sparse*.

A Simple Bucket-based Scheme

Let C = 1 + max(c_{ij} : (i,j) $\in$ A); then nC is an upper bound on the minimum length path from 1 to n.

RECALL: When we select nodes for Dijkstra's Algorithm we select them in increasing order of distance from node 1.

SIMPLE STORAGE RULE. **Create buckets from 0 to nC.**

Let BUCKET(k) = {i $\in$ T: d(i) = k}.

This technique is known as *Dial's Algorithm.*

Dial's Algorithm

- Whenever d(j) is updated, update the buckets so that the simple bucket scheme remains true.
- The FindMin operation looks for the minimum non-empty bucket.
- To find the minimum non-empty bucket, start where you last left off, and iteratively scan buckets with higher numbers.

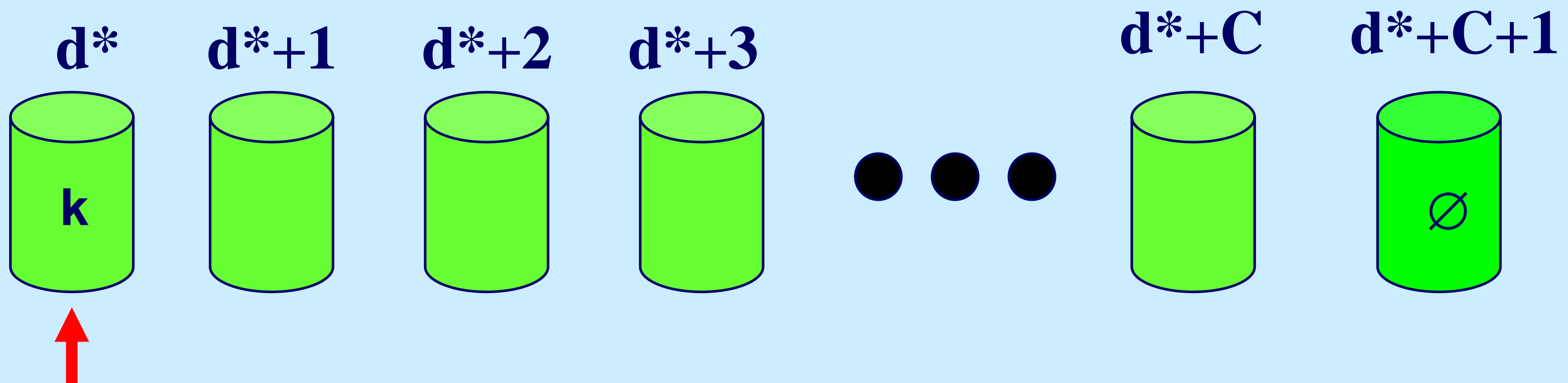
Dial's Algorithm

Running time for Dial's Algorithm

- Let C be the largest arc length (cost).
- Number of buckets needed. O(nC)
- Time to create buckets. O(nC)
- Time to update d( ) and buckets. O(m)
- Time to find min. O(nC).
- Total running time. O(m+ nC).
- This can be improved in practice.

Additional comments on Dial's Algorithm

- Create buckets when needed. Stop creating buckets when each node has been stored in a bucket.
- Let d* = max d*(j). Then the maximum bucket ever used is at most d* + C.



Suppose $j \in$ Bucket(d* + C + 1) after update(i).
But then $d(j) = d(i) + c_{ij} \leq d^* + C$

Summary

- **Shortest path problem, with**
  - **Single origin**
  - **non-negative arc lengths**

- **Dijkstra's algorithm (label setting)**
  - **Simple implementation**
  - **Dial's simple bucket procedure**

- **Next lecture: a more complex bucket procedure that reduces the time to $O(m + n \log C)$.**