# Lecture: shortest path problems

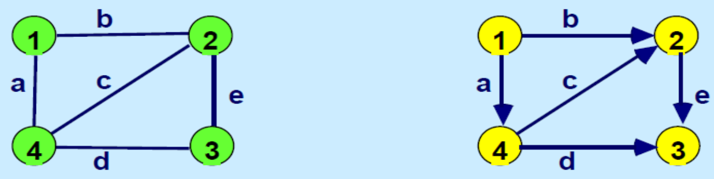
http://bicmr.pku.edu.cn/~wenzw/bigdata2016.html

Acknowledgement: this slides is based on Prof. James B. Orlin's lecture notes of "15.082/6.855J, Introduction to Network Optimization" at MIT

Textbook: Network Flows: Theory, Algorithms, and Applications by Ahuja, Magnanti, and Orlin referred to as AMO

# Notation and Terminology

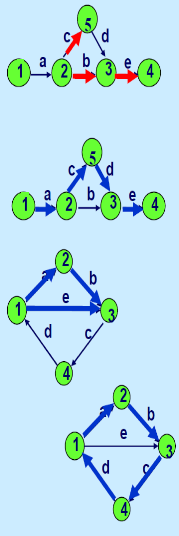
Network terminology as used in AMO.



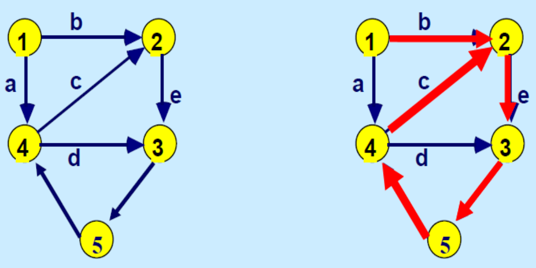

Left: an undirected graph, Right: a directed graph

- Network G = (N, A)
- Node set N = {1, 2, 3, 4}
- Arc set A = {(1,2), (1,3), (3,2), (3,4), (2,4)}
- In an undirected graph, (i,j) = (j,i)

- Path: a finite sequence of nodes: $i_1, i_2, \ldots, i_t$ such that $(i_k, i_{k+1}) \in A$ and all nodes are not the same. Example: 5, 2, 3, 4. (or 5, c, 2, b, 3, e, 4). No node is repeated. Directions are ignored.
- Directed Path. Example: 1, 2, 5, 3, 4 (or 1, a, 2, c, 5, d, 3, e, 4). No node is repeated. Directions are important.
- Cycle (or circuit or loop) 1, 2, 3, 1. (or 1, a, 2, b, 3, e). A path with 2 or more nodes, except that the first node is the last node. Directions are ignored.
- Directed Cycle: (1, 2, 3, 4, 1) or 1, a, 2, b, 3, c, 4, d, 1. No node is repeated. Directions are important.



# Walks

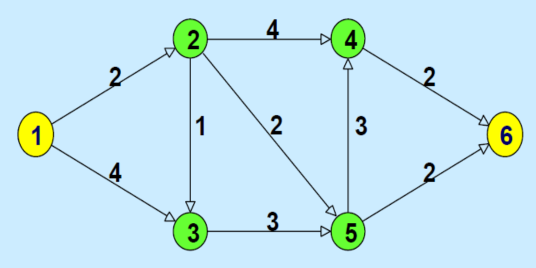



- Walks are paths that can repeat nodes and arcs
- Example of a directed walk: 1-2-3-5-4-2-3-5
- A walk is closed if its first and last nodes are the same.
- A closed walk is a cycle except that it can repeat nodes and arcs.

# Three Fundamental Flow Problems

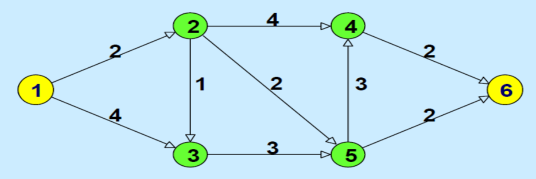
- The shortest path problem
- The maximum flow problem
- The minimum cost flow problem

# The shortest path problem




- Consider a network G = (N, A) with cost $c_{ij}$ on each edge $(i,j) \in A$. There is an origin node s and a destination node t.
- Standard notation: n = |N|, m = |A|
- cost of of a path: $c(P) = \sum_{(i,j)\in P} c_{ij}$
- What is the shortest path from s to t?

# The shortest path problem




$$\min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

s.t. $\sum_j x_{sj} = 1$

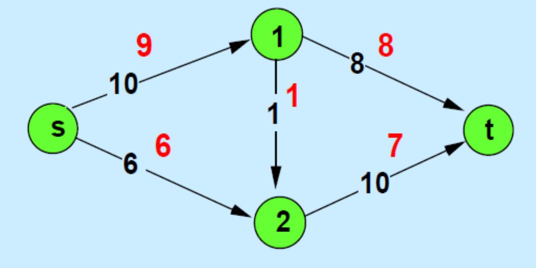
$$\sum_j x_{ij} - \sum_j x_{ji} = 0, \text{ for each i } \neq s \text{ or } t$$

$$-\sum_i x_{it} = -1$$

$x_{ij} \in \{0, 1\}$ for all $(i, j)$

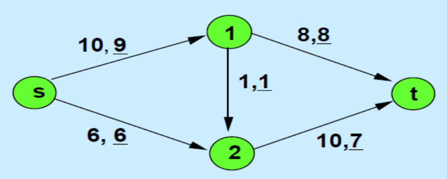
# The Maximum Flow Problem

- Directed Graph G = (N, A).
  - Source s
  - Sink t
  - Capacities $u_{ij}$ on arc (i,j)
  - Maximize the flow out of s, subject to
- Flow out of i = Flow into i, for $i \neq s$ or t.




A Network with Arc Capacities (and the maximum flow)

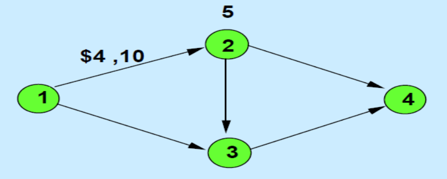
# Representing the Max Flow as an LP




Flow out of i = Flow into i, for $i \neq s$ or t.

$$\begin{aligned}
\max \quad & v \\
\text{s.t.} \quad & \sum_j x_{sj} = v \\
& \sum_j x_{ij} - \sum_j x_{ji} = 0, \ \text{for each i } \neq s \text{ or } t \\
& -\sum_i x_{it} = -v \\
& 0 \leq x_{ij} \leq u_{ij} \text{ for all } (i,j)
\end{aligned}$$

# Min Cost Flows




Flow out of i - Flow into i = b(i).
Each arc has a linear cost and a capacity

$$\min \sum_{ij} c_{ij} x_{ij}$$

$$\text{s.t. } \sum_{j} x_{ij} - \sum_{j} x_{ji} = b(i), \text{ for each i}$$

$$0 \le x_{ij} \le u_{ij} \text{ for all } (i,j)$$

Covered in detail in Chapter 1 of AMO

# Where Network Optimization Arises

- Transportation Systems
  - transportation of goods over transportation networks
  - Scheduling of fleets of airplanes
- Manufacturing Systems
  - Scheduling of goods for manufacturing
  - Flow of manufactured items within inventory systems
- Communication Systems
  - Design and expansion of communication systems
  - Flow of information across networks
- Energy Systems, Financial Systems, and much more

# Applications in social network: shortest path

2014 ACM SIGMOD Programming Contest

`http://www.cs.albany.edu/~sigmod14contest/task.html`

- Shortest Distance Over Frequent Communication Paths
  定义社交网络的边: 相互直接至少有x条回复并且相互认识。给定网络里两个人p1和p2 以及另外一个整数x，寻找图中p1 和p2之间数量最少节点的路径
- Interests with Large Communities
- Socialization Suggestion
- Most Central People (All pairs shorted path)
  定义网络：论坛中有标签t的成员，相互直接认识。给定整数k和标签t,寻找k个有highest closeness centrality values的人

# Applications in social network: max flow and etc

Community detection in social network

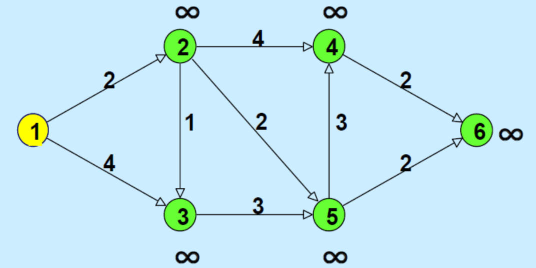
- Social network is a network of people connected to their “friends”
- Recommending friends is an important practical problem
- solution 1: recommend friends of friends
- solution 2: detect communities
  - idea1: use max-flow min-cut algorithms to find a minimum cut
  - it fails when there are outliers with small degree
  - idea2: find partition A and B that minimize conductance:

$$\min_{A,B} \frac{c(A,B)}{|A|\ |B|},$$

where $c(A,B) = \sum_{i \in A} \sum_{j \in B} c_{ij}$

# Dijkstra's Algorithm for the Shortest Path Problem

# Single source shortest path problem




Find the shortest path from a source node to each other node.

- Assume: all arc lengths are non-negative
- the network is directed
- there is a path from the source node to all other nodes

# 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

# The shortest path problem: LP relaxation

LP Relaxation: replace $x_{ij} \in \{0,1\}$ by $x_{ij} \geq 0$

**Primal**

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

s.t. $-\sum_j x_{sj} = -1$

$$\sum_j x_{ji} - \sum_j x_{ij} = 0, i \neq s \text{ or } t$$

$$\sum_i x_{it} = 1$$

$x_{ij} \geq 0$ for all $(i,j)$

**Dual**

$$\max \quad d(t) - d(s)$$

s.t. $d(j) - d(i) \leq c_{ij}, \forall (i,j) \in A$

Signs in the constraints in the primal problem

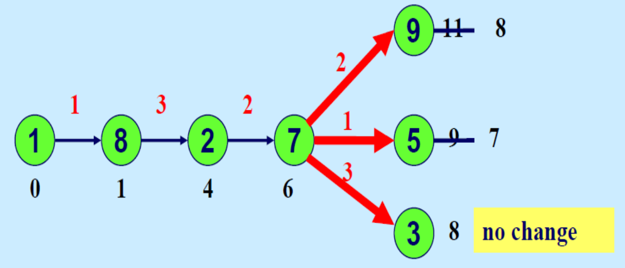
# Dual LP

Claim: When $G = (N, A)$ satisfies the no-negative-cycles property, the indicator vector of the shortest s-t path is an optimal solution to the LP.

- Let $x^*$ be the indicator vector of shortest s-t path
  - $x^*_{ij} = 1$ if $(i, j) \in P$, otherwise $x^*_{ij} = 0$
  - Feasible for primal
- Let $d^*(v)$ be the shortest path distance from s to v
  - Feasible for dual (by triangle inequality)
- $\sum_{(i,j)\in A} c_{ij}x^*_{ij} = d^*(t) - d^*(s)$
- Hence, both $x^*$ and $d^*$ are optimal

# Update(7)

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

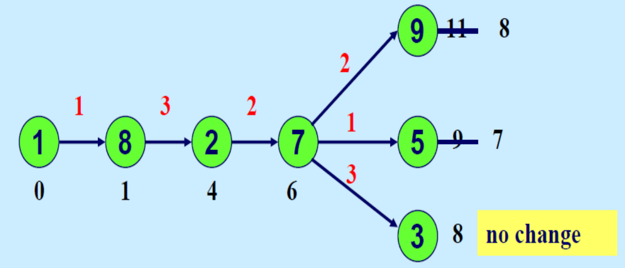



Suppose 7 is incident to nodes 9, 5, 3, with temporary distance labels as shown.

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 = N$\backslash$S denotes the set of temporarily labeled nodes.

# Dijkstra's Algorithm

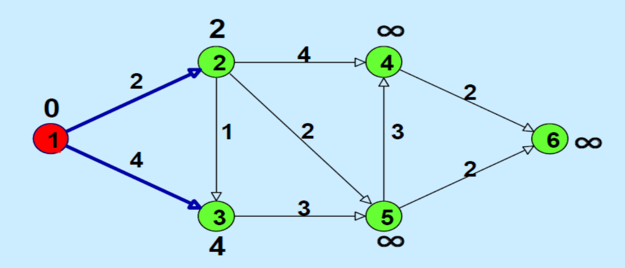
- S:= {1}; T = N - {1};
  d(1):= 0 and pred(1):= 0; d(j) = $\infty$ for j = 2 to n;
  update(1);
- while $S \neq N$ do
  - (node selection, also called FINDMIN)
    let i$\in$T be a node for which
    d(i) = min {d(j) : j $\in$T};
    S : = S $\cup$ {i}; T: = T - {i};
  - Update(i)
    for each (i,j)$\in$ A(i) do
    if $d(j) > d(i) + c_{ij}$ then
    $d(j) := d(i) + c_{ij}$ and pred(j) : = i;

# Invariants for Dijkstra's Algorithm

1. If $j \in S$, then $d(j) = d^*(j)$ is the shortest distance from node 1 to node j.
2. (after the update step) If $j \in T$, then $d(j)$ is the length of the shortest path from node 1 to node j in $S \cup \{j\}$, which is the shortest path length from 1 to j of scanned arcs.

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

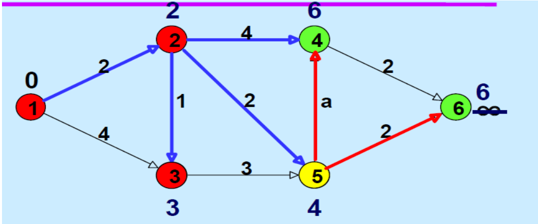
# Verifying invariants when S ={ 1 }




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

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

- d(5) = min {d(j) : j∈ T}.
- Consider any path from 1 to 5 passes through a node k of T. The path to node k has distance at least d(5). So d(5) = d*(5).
- Suppose 5 is transferred to S and we carry out Update(5). Let P be the shortest path from 1 to j with j ∈ T.
- If 5 ∉ P, then invariant 2 is true for j by induction. If 5 ∈P, then invariant 2 is true for j because of Update(5).

# 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.
- Can be improved to $O(m + nlogC)$

# The Label Correcting Algorithm

# Overview

- A generic algorithm for solving shortest path problems
  - negative costs permitted
  - but no negative cost cycle (at least for now)
- The use of reduced costs
- All pair shortest path problem
- INPUT G = (N, A) with costs c
- Node 1 is the source node
- There is no negative cost cycle
  - We will relax that assumption later

# Optimality Conditions

Lemma. Let d*(j) be the shortest path length from node 1 to node j, for each j. Let d( ) be node labels with the following properties:

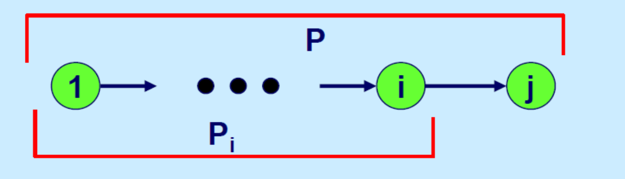
$$d(j) \leq d(i) + c_{ij} \text{ for i } \in \text{ N for j } \neq 1 \quad (1)$$
$$d(1) = 0 \quad (2)$$

Then d(j) $\leq$ d*(j) for each j.

- Proof. Let P be the shortest path from node 1 to node j.

# Completion of the proof

- If P = (1, j), then $d(j) \leq d(1) + c_{1j} = c_{1j} = d^*(j)$.
- Suppose |P| > 1, and assume that the result is true for paths of length |P| - 1. Let i be the predecessor of node j on P, and let $P_i$ be the subpath of P from 1 to i.




- $P_i$ is the shortest path from node 1 to node i. So, $d(i) \leq d^*(i) = c(P_i)$by inductive hypothesis. Then, $d(j) \leq d(i) + c_{ij} \leq c(P_i) + c_{ij} = c(P) = d^*(j)$.

# Optimality Conditions

Theorem. Let d(1), . . . , d(n) satisfy the following properties for a directed graph G = (N,A):

1. d(1) = 0.
2. d(i) is the length of some path from node 1 to node i.
3. $d(j) \leq d(i) + c_{ij}$ for all (i,j) $\in$A.

Then d(j) = d*(j).

Proof. $d(j) \leq d^*(j)$ by the previous lemma. But, $d(j) \geq d^*(j)$ because d(j) is the length of some path from node 1 to node j. Thus d(j) = d*(j).

# A Generic Shortest Path Algorithm

Notation.

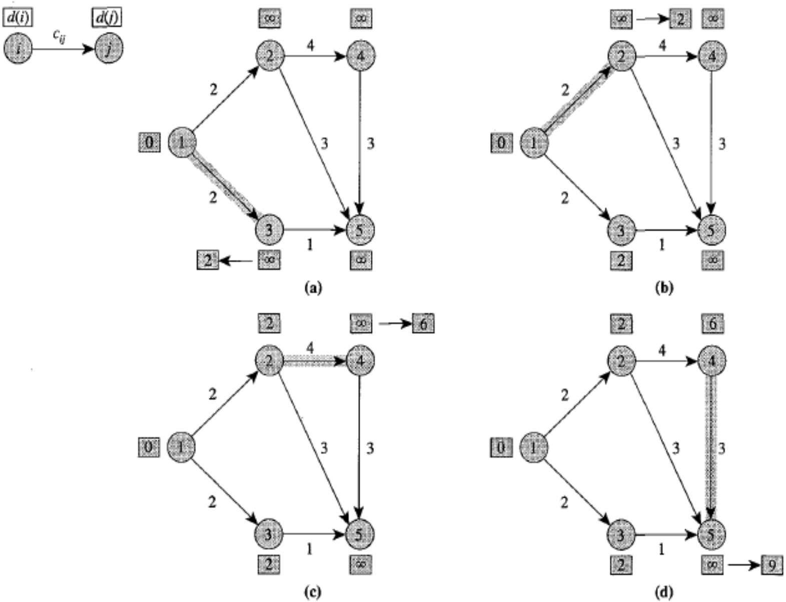
- d(j) = “temporary distance labels”.
  - At each iteration, it is the length of a path (or walk) from 1 to j.
  - At the end of the algorithm d(j) is the minimum length of a path from node 1 to node j.
- Pred(j) = Predecessor of j in the path of length d(j) from node 1 to node j.
- $c_{ij}$ = length of arc (i,j).

# A Generic Shortest Path Algorithm

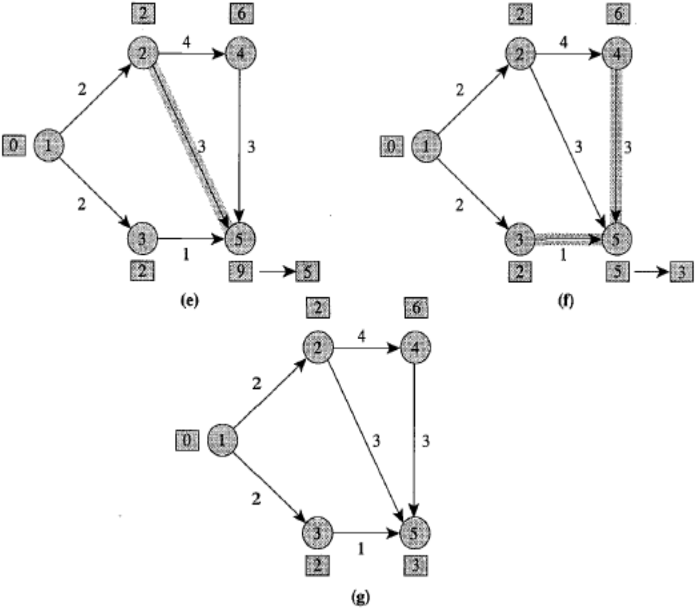
Algorithm LABEL CORRECTING;

- d(1) : = 0 and Pred(1) := $\emptyset$;
  d(j) : = $\infty$ for each j$\in$N - {1};
- while some arc (i,j) satisfies $d(j) > d(i) + c_{ij}$ do
  $d(j) := d(i) + c_{ij}$;
  Pred(j) : = i;

# llustration

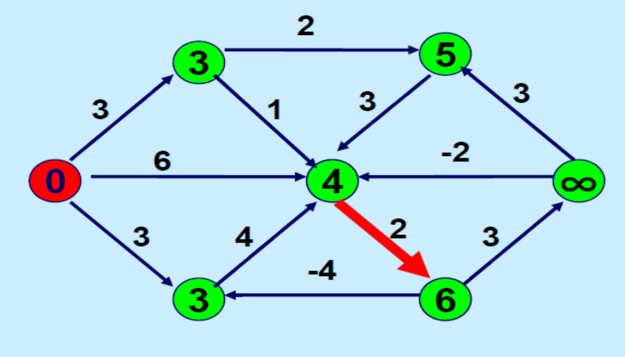


# Ilustration



# Algorithm Invariant

At each iteration, if d(j) < $\infty$, then d(j) is the length of some walk from node 1 to node j.



Theorem. Suppose all data are integral, and that there are no negative cost cycles in the network. Then the label correcting algorithm ends after a finite number of steps with the optimal solution.

- Proof of correctness. The algorithm invariant ensures that d(j) is the length of some walk from node 1 to node j. If the algorithm terminates, then the distances satisfy the optimality conditions.
- Proof of Finiteness. Consider finite distance labels. At each iteration, d(j) decreases by at least one for some j.
- Also $nC \geq d(j) \geq d^*(j) > -nC$, where C = max ($|c_{ij}|$ : (i,j) $\in$A).
- So, the number of iterations is $O(n^2C)$.

# More on Finiteness

- What happens if data are not required to be integral? The algorithm is still finite, but one needs to use a different proof. What happens if there is a negative cost cycle?
- The algorithm may no longer be finite. Possibly, d(j) keeps decreasing to -$\infty$.
- But we can stop when d(j) < -nC since this guarantees that there is a negative cost cycle.

# On computational complexity

- Proving finiteness is OK, but . . .
- Can we make the algorithm polynomial time? If so, what is the best running time?
- Can we implement it efficiently in practice?

# A Polynomial Time Version of the Label Correcting Algorithm

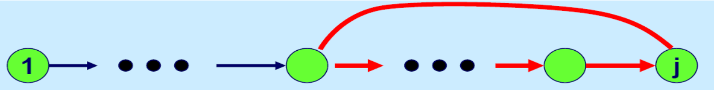
We define a pass to consist of scanning all arcs in A, and updating distance labels when d(j) > d(i) + $c_{ij}$ . (We permit the arcs to be scanned in any order).

Theorem. If there is no negative cost cycle, then the label correcting algorithm determines the shortest path distances after at most n-1 passes. The running time is O(nm).

# Proof follows from this lemma

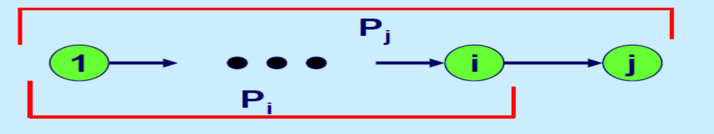
Lemma. Let $P_j$ be a shortest walk from node i to node j. (If there are multiple shortest walks, choose one with the fewest number of arcs.)
Let $d^k(j)$ be the value d(j) after k passes.
Then $d^k(j) = d^*(j)$ if $P_j$ has at most k arcs.

- Note: if there are no negative cost cycles, then $P_j$ will also be a path. If there is a negative cost cycle containing node j, then there is no shortest walk to node j.



# Proof of lemma

- To show: $d^k(j) = d^*(j)$ whenever that shortest path from 1 to j has at most k arcs.
- $d^1(j) \leq c_{1j}$ . If $P_j$ = (1, j) the lemma is true.
- Suppose $|P_j|$ > 1, and assume that the result is true for paths of length $|P_j|$ - 1. Let i be the predecessor of node j on $P_j$. Then the subpath from 1 to i is a shortest path to i.




- After pass k, $d^k(j) \leq d^{k-1}(i) + c_{ij} = c(P_i) + c_{ij} = d^*(j)$.

# What if there is a negative cost cycle?

- If in the n-th pass, there is no update, then the algorithm has found the shortest path distances.
- If in the nth pass, there is an update, then there must be a negative cost cycle.

# Solving all pairs shortest problems

- Note: Dijkstra's algorithm is much faster in the worst case than label correcting.
  - $O(m + n \log nC)$ vs $O(mn)$
- To solve the all pairs shortest path problem we will solve it as
  - one shortest path problem using label correcting
  - n-1 shortest path problems using Dijkstra
  - Technique: create an equivalent optimization problem with nonnegative arc lengths.

# Reduced Costs

- Suppose that is any vector of node potentials.
  Let $c^{\pi}_{ij} = c_{ij} - \pi_i + \pi_j$ be the reduced cost of arc (i,j)
- For a path P, let c(P) denote the cost (or length) of P. Let $c^{\pi}(P)$ denote the reduced cost (or length) of P

$$c(P) = \sum_{(i,j)\in P} c_{ij}, \quad c^{\pi}(P) = \sum_{(i,j)\in P} c^{\pi}_{ij}$$

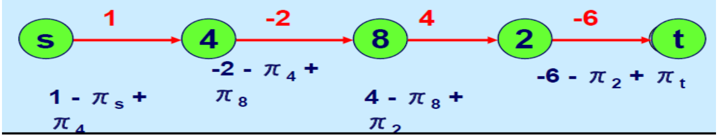
Lemma: For any path P from node s to node t,

$$c^{\pi}(P) = c(P) - \pi_s + \pi_t$$

# Proof: $c^\pi(P) = c(P) - \pi_s + \pi_t$

- Proof: When written as a summation, the terms in $c^\pi(P)$ involving $\pi$ for some i all cancel, except for the term $-\pi_s$ and the term $\pi_t$.
- Note: for fixed vector $\pi$ of multipliers and for any pair of nodes s and t, $c^\pi(P) - c(P)$ is a constant for every path P from s to t.

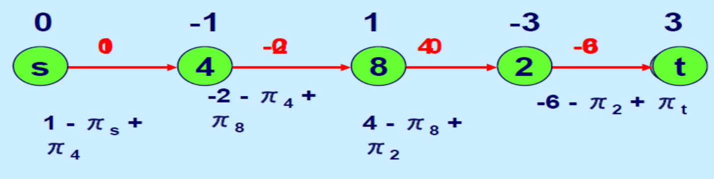
Corollary. A shortest path P from s to t with respect $c^\pi$ is also the shortest path with respect to c.



# Using reduced costs

Lemma. Let d(j) denote the shortest path from node s to node j. Let $\pi_j$ = -d(j) for all j. Then $c^{\pi}_{ij} \geq 0$ for all (i,j) $\in$ A.

Proof: $d(j) \leq d(i) + c_{ij} \Longrightarrow c_{ij} + d(i) - d(j) \geq 0 \Longrightarrow c^{\pi}_{ij} \geq 0$



# Solving the all pair shortest path problem

- Step 1. Find the shortest path from node 1 to all other nodes. Let d(j) denote the shortest path from 1 to j for all j.
- Step 2. Let $\pi_j$ = -d(j) for all j.
- Step 3. For i = 2 to n, compute the shortest path from node i to all other nodes with respect to arc lengths $c^\pi$.

Running time using Radix Heaps**.

- O(nm) for the first shortest path tree
- O(m + n log C) for each other shortest path tree.
- O(nm + $n^2$ log C) in total.
- One can choose a slightly faster approach.

# Detecting Negative Cost Cycles

- Approach 1. Stop if d(j) is sufficiently small, say d(j) $\geq$ -nC.
- Approach 2. Run the FIFO modified label correcting algorithm, and stop after n passes.
- Approach 3. Run the FIFO label correcting algorithm, and keep track of the number of arcs on the "path" from s to j. If the number of arcs exceeds n-1, then quit.
- Approach 4. At each iteration of the algorithm, each node j (except for the root) has a a temporary label d(j) and a predecessor pred(j). The predecessor subgraph consists of the n-1 arcs {(pred(j),j) : j $\neq$ s}. It should be a tree. If it has a cycle, then the cost of the cycle will be negative, and the algorithm can terminate.