15.082J and 6.855J and ESD.78J
October 26, 2010

Introduction to Minimum Cost Flows

Overview of lecture

- Quick review of min cost flow problem
- An application of min cost flows
- The residual network, again
- The cycle canceling algorithm for solving the min cost flow problem
- Reduced costs and optimality conditions

The Minimum Cost Flow Problem

u_{ij} = capacity of arc (i,j).

c_{ij} = unit cost of shipping flow from node i to node j on (i,j).

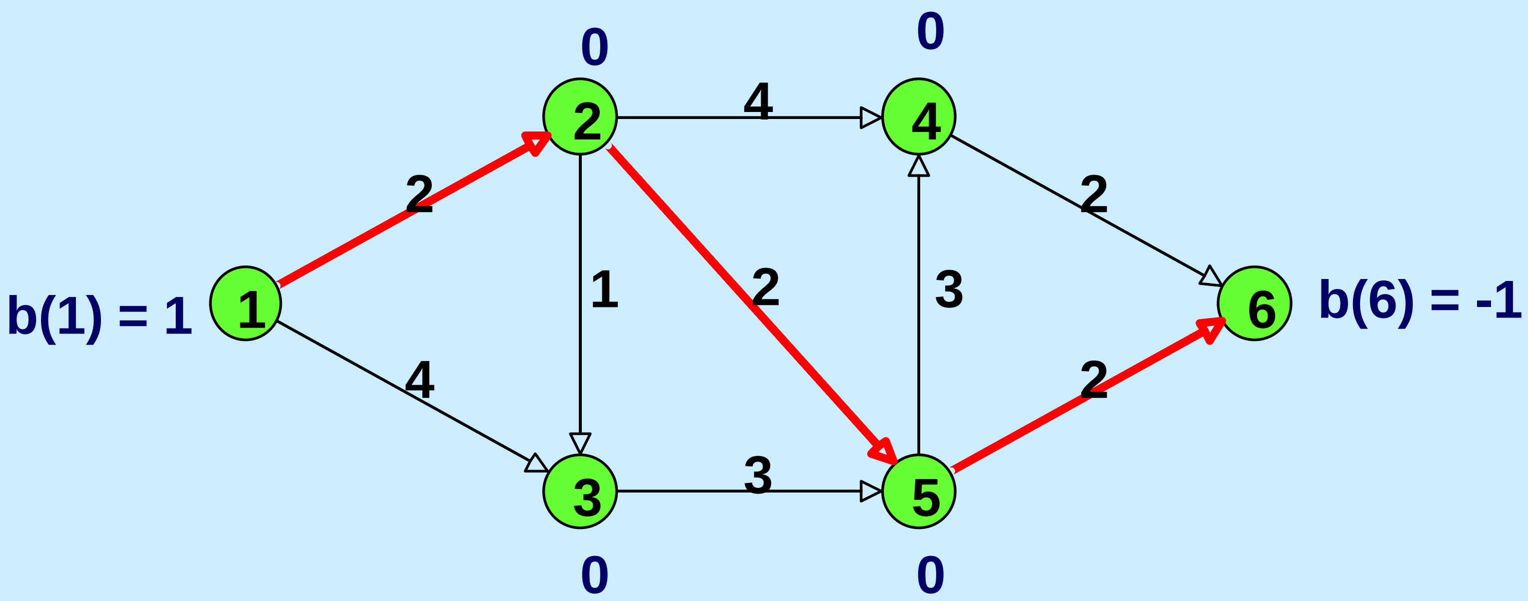
x_{ij} = amount shipped on arc (i,j)

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

$$\sum_j x_{ij} \quad - \quad \sum_j x_{ji} = b_i \qquad \text{for all } i \in N.$$

and $0 \le x_{ij} \le u_{ij}$ for all $(i,j) \in A$.

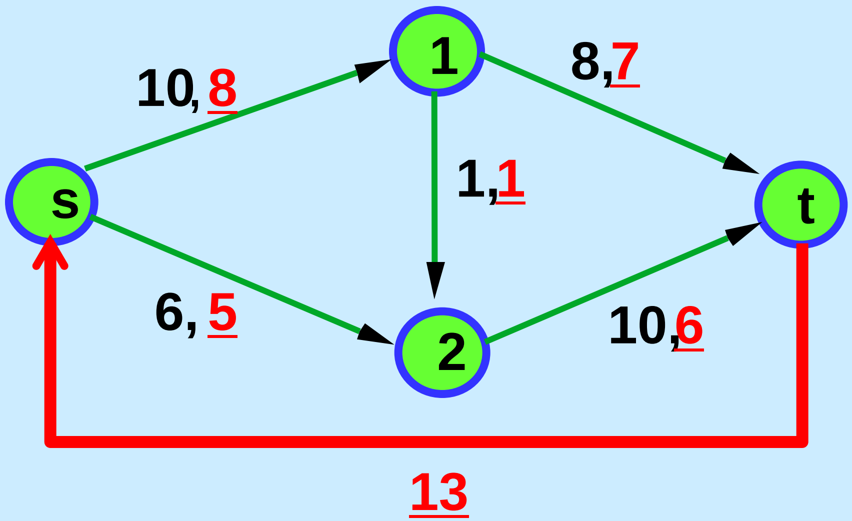
Find the shortest path from node 1 to node 6



The optimal flow is to send one unit of flow along 1-2-5-6.

This transformation works so long as there are no negative cost cycles in G.
(What if there are negative cost cycles?)

Find the Maximum Flow from s to t



b(i) = 0 for all i;

add arc (t,s) with a cost of -1 and large capacity.

The cost of every other arc is 0.

The optimal solution in the corresponding minimum cost flow problem will send as much flow in (t,s) as possible.

Transshipment Problems

Plants with given production capabilities for a product.

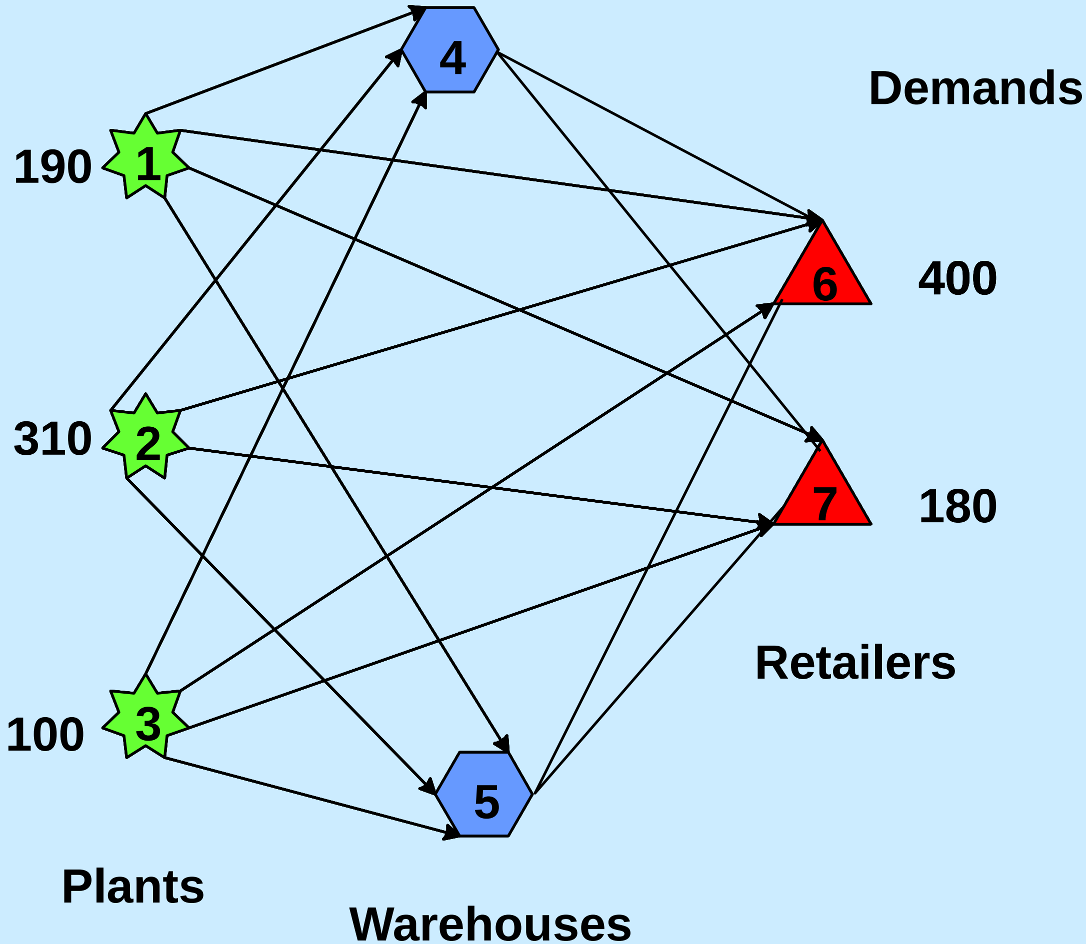
One can ship directly from the plants to retailers, or from plants to warehouses, and then from warehouses to retailers.

There is a given demand for each retailer.

Costs of shipment are given.

What is the minimum cost method for satisfying demands?

A Network Representation



The Caterer Problem

Demand for d_i napkins on day i for i = 1 to 7 (so, j ∈ [1..7]).

Cost of new napkins: *a* cents each,

2-day laundry: b cents per napkin

1-day laundry: c cents per napkin.

Minimize the cost of meeting demand.

Assumptions:

- **there is no initial inventory of napkins**
- **at the end, no clean napkins remain**

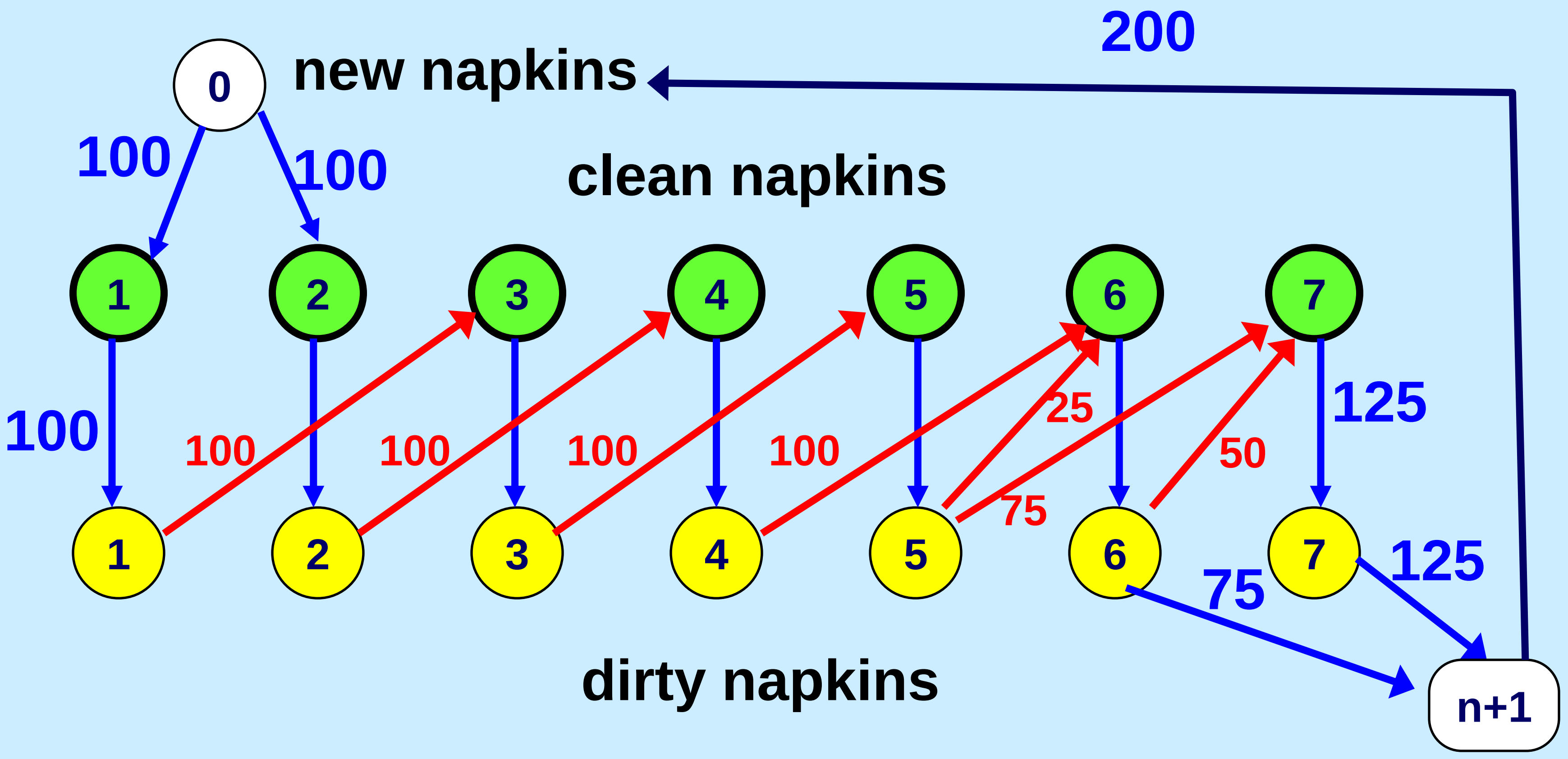
Application to airplane maintenance.

An example with a feasible solution

new napkins:	$10
1 day cleaning	$2
2-day cleaning	$1

Demand:

M-F	100 napkins,
Sa-Su	125 napkins



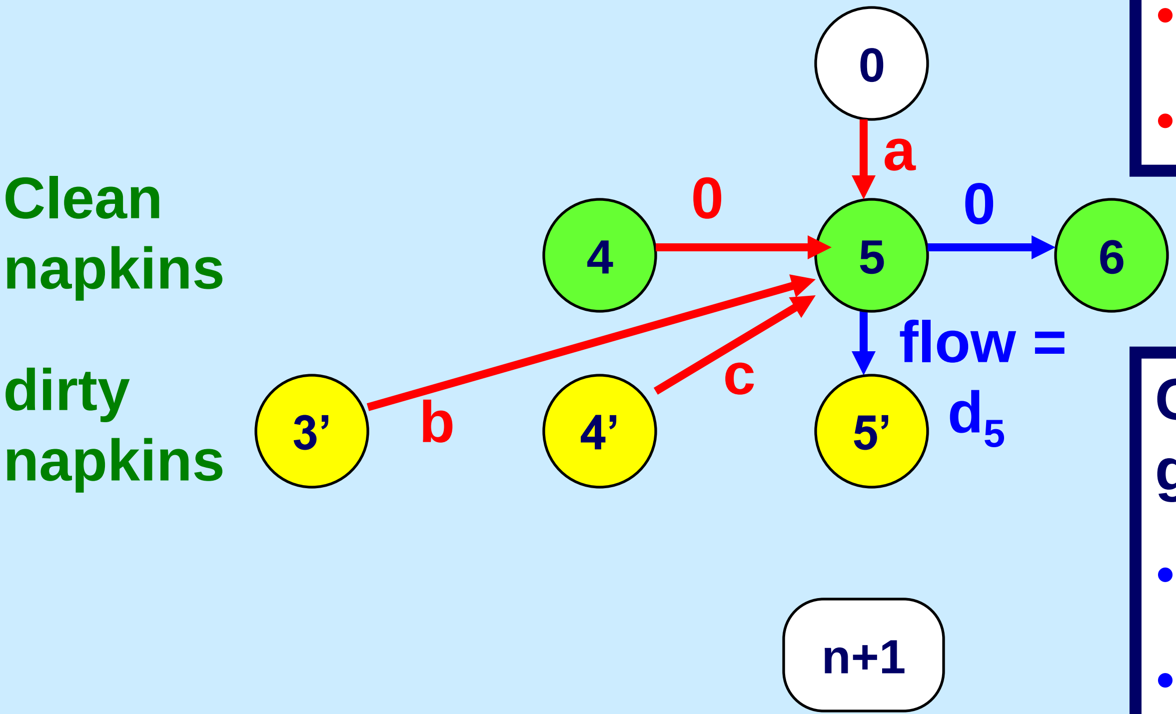
Nodes and Arcs

Node j : Clean napkins on day j

Node j' : Dirty napkins on day j

Node 0: Where napkins originate

Node n+1: Where napkins go to stay dirty



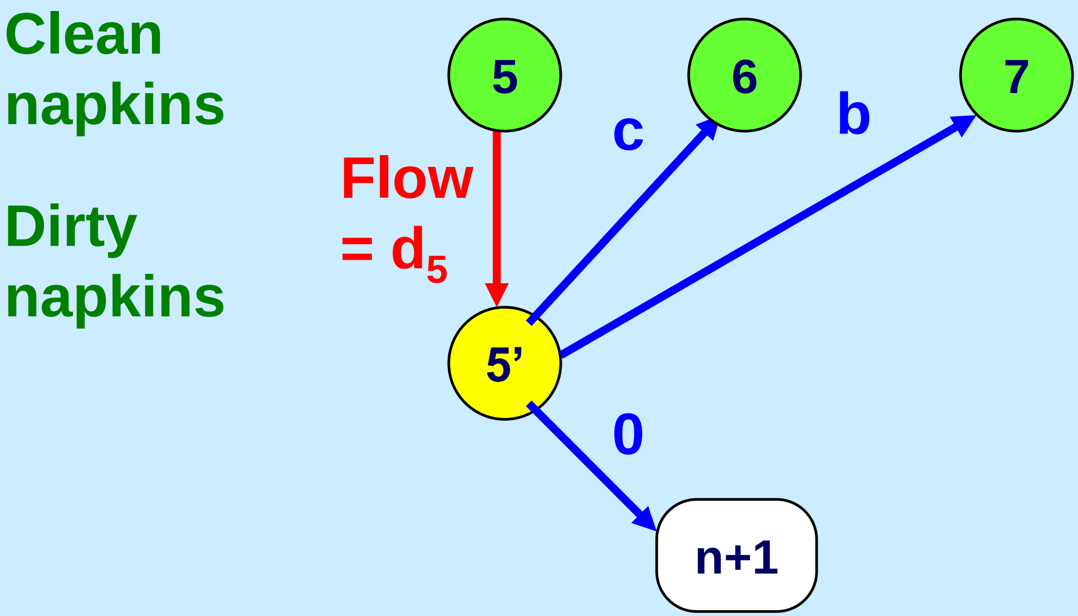
Clean napkins on day 5 come from

- purchases (0, 5)
- leftover clean napkins (4, 5)
- 2-day laundry (3', 5)
- 1-day laundry (4', 5)

Clean napkins on day 5 go to

- being used (5, 5')
- being stored (5, 6)

Nodes and Arcs



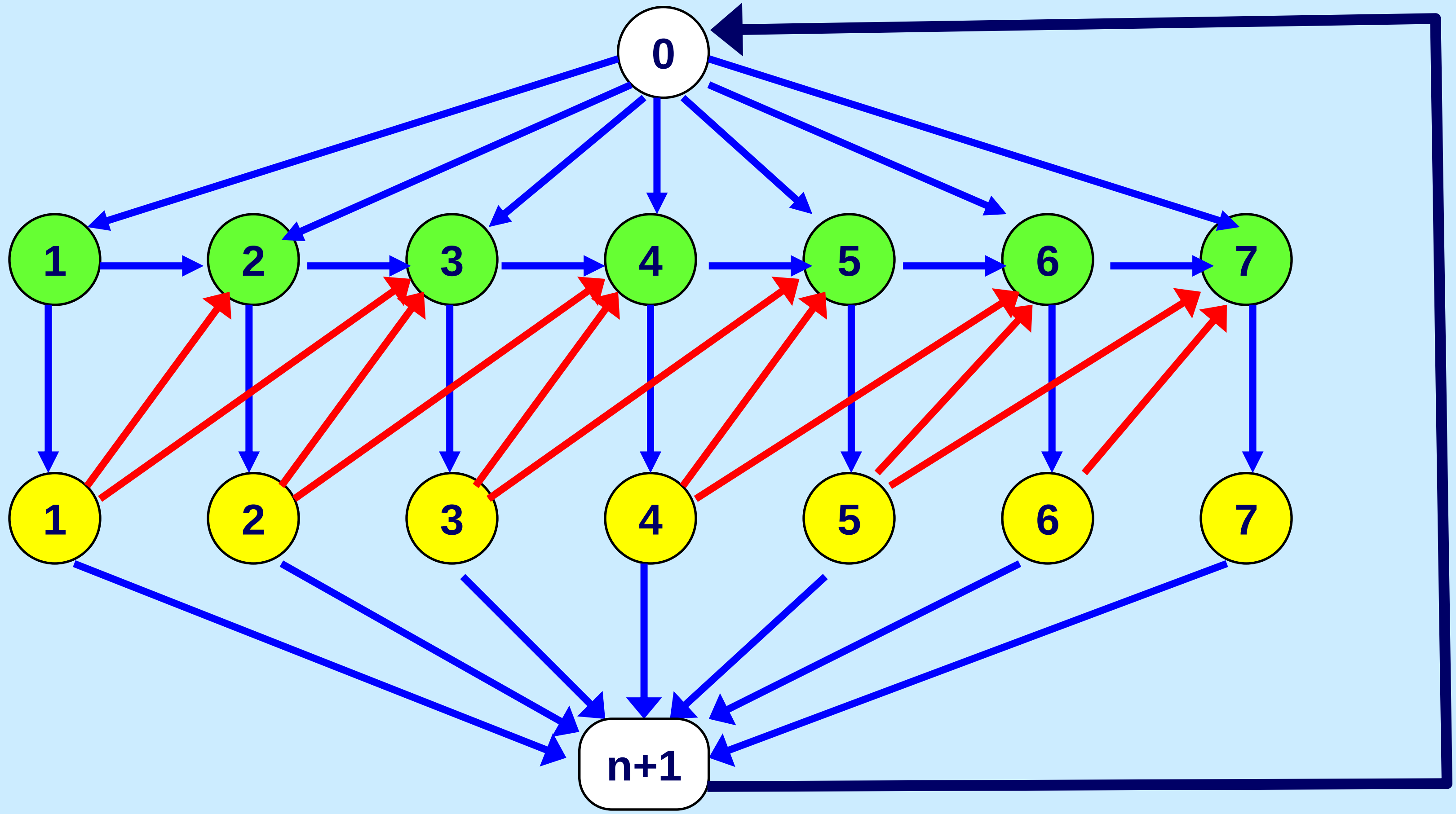
Dirty napkins on day 5 come from

- being used on day 5 (5, 5')

Dirty napkins on day 5 go to

- never cleaned (5, n+1)
- 1-day cleaning (5', 6)
- 2- day cleaning (5', 7)

The network for the caterer problem



Find a minimum cost circulation such that the flow on (j, j') = d_j on arcs (j, j') for j = 1 to n. Lower bound = upper bound = d_j

Arc (n+1, 0) : each purchased napkin ends up dirty.

The minimum cost flow problem

- Simplifying assumptions
- Finding a feasible flow
- The residual network
- The cycle canceling algorithm
- Reduced costs and optimality conditions

Some Assumptions

1. **All data is integral. (Needed for some proofs, and some running time analysis).**

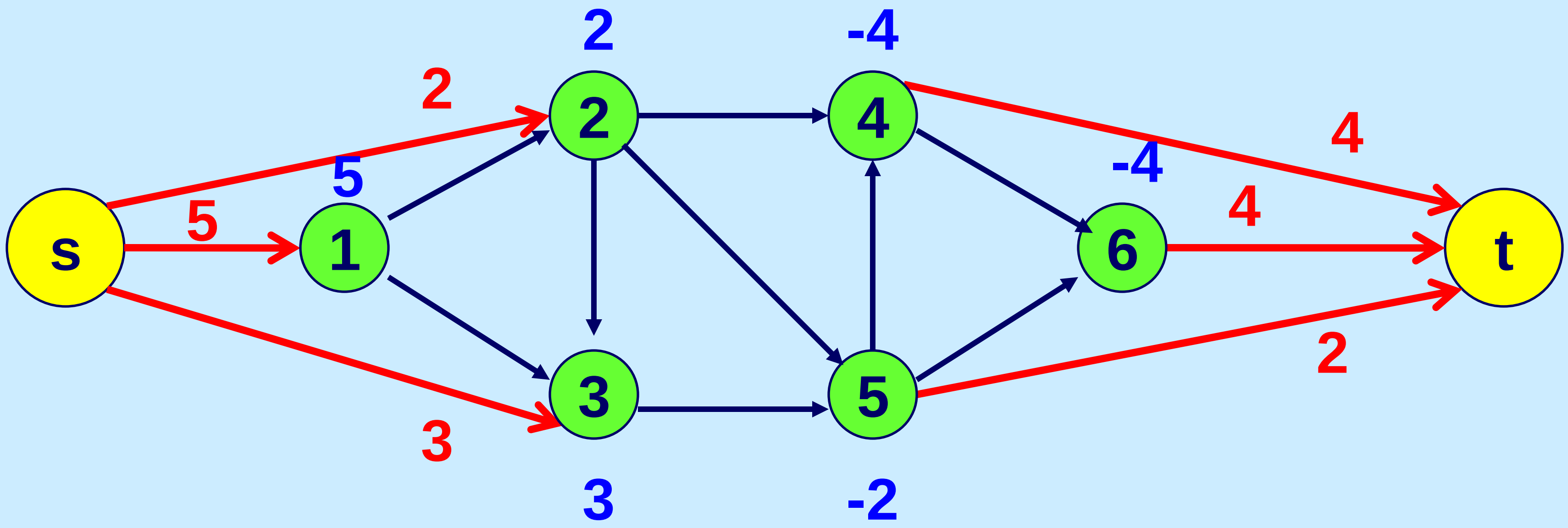
2. **The network is directed and connected**

3. $\sum_{i=1 \text{ to } n} b(i) = 0.$
(Otherwise, there cannot be a feasible solution.)

Finding a feasible solution

One can find a feasible solution when all lower bounds are 0 by solving a single max flow problem.

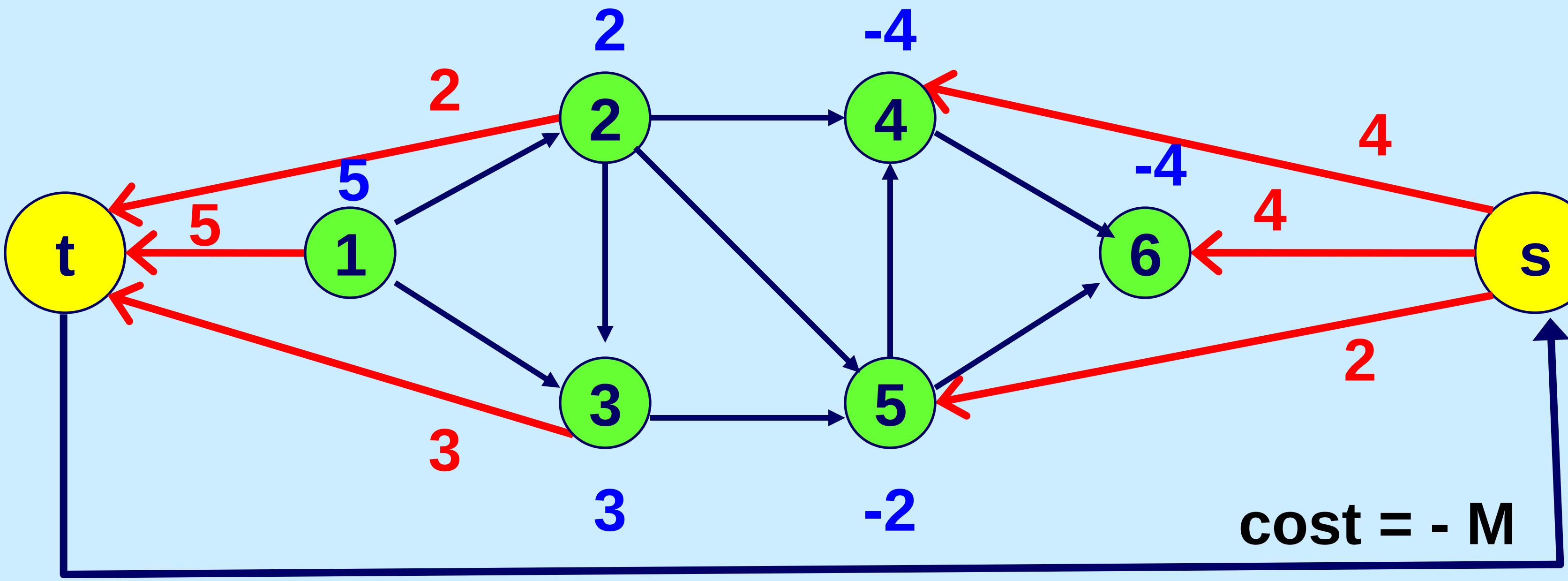
1. If b(j) > 0, create arc (s, j) with u_{sj} = b(j),
2. If b(j) < 0, create arc (j, t) with u_{jt} = - b(j)



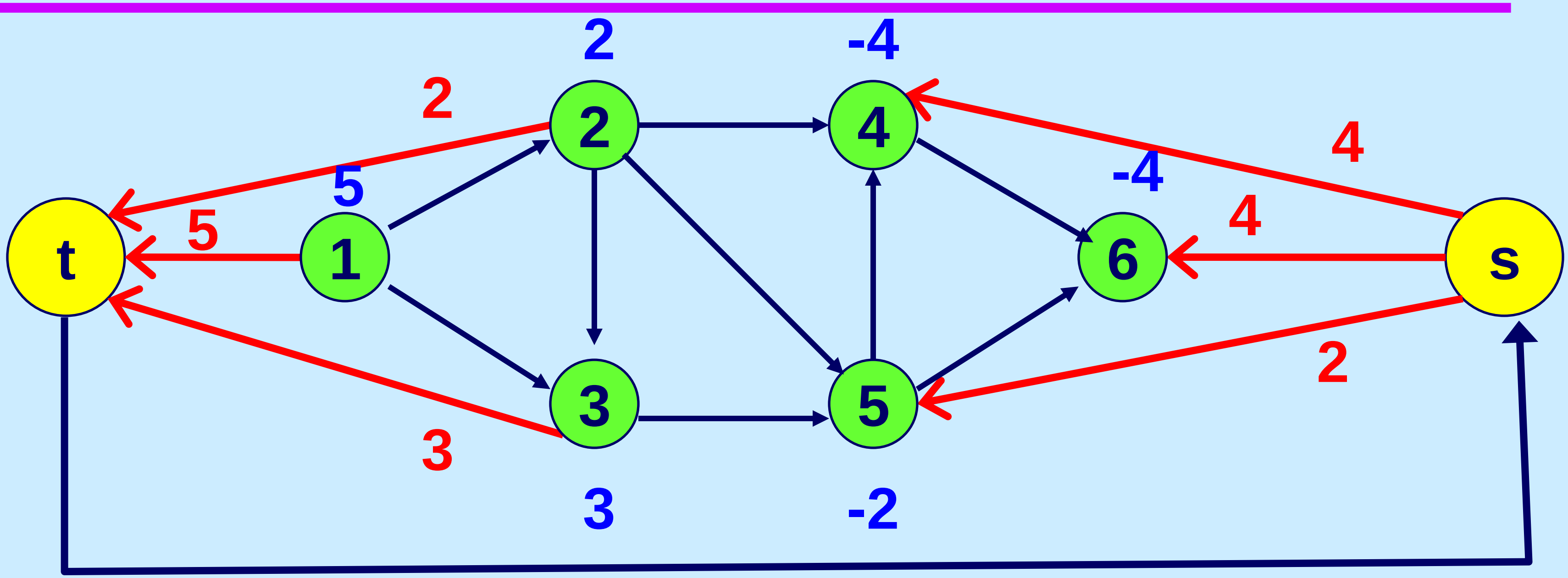
Finding an artificial feasible solution

One can start with an "artificial" feasible solution with large cost. The flow in these arcs will be 0 at the end.

1. Add nodes s and t with b(s) = b(t) = 0
2. If b(j) > 0, create arc (j, t) with u_{jt} = b(j) and c_{jt} = 0
3. If b(j) < 0, create arc (s, j) with u_{sj} = - b(j) and c_{jt} = 0
4. Add an arc (t, s) with u_{ts} = mu_{max} and c_{ts} = mc_{max}



Finding an artificial feasible solution



Initial solution.

- If b(j) > 0, then x_{jt} = b(j) (supplies are satisfied)
- If b(j) < 0, then x_{sj} = - b(j) (demands are satisfied)
- $x_{ts} = \sum_{b(j)>0}$ b(j) (flow into t = flow out of t)
- x_{ij} = 0 otherwise

In an optimal feasible solution, x_{ts} = 0. There is no flow in any of the artificial arcs)

Mental Break

Why is the word “ring” part of “boxing ring”?

They used to be round.

How many dimples are in a regulation golf ball?

336

Was “tug of war” ever an Olympic event?

Yes. Between 1900 and 1920.

Mental Break

Where did karate originate?

In India.

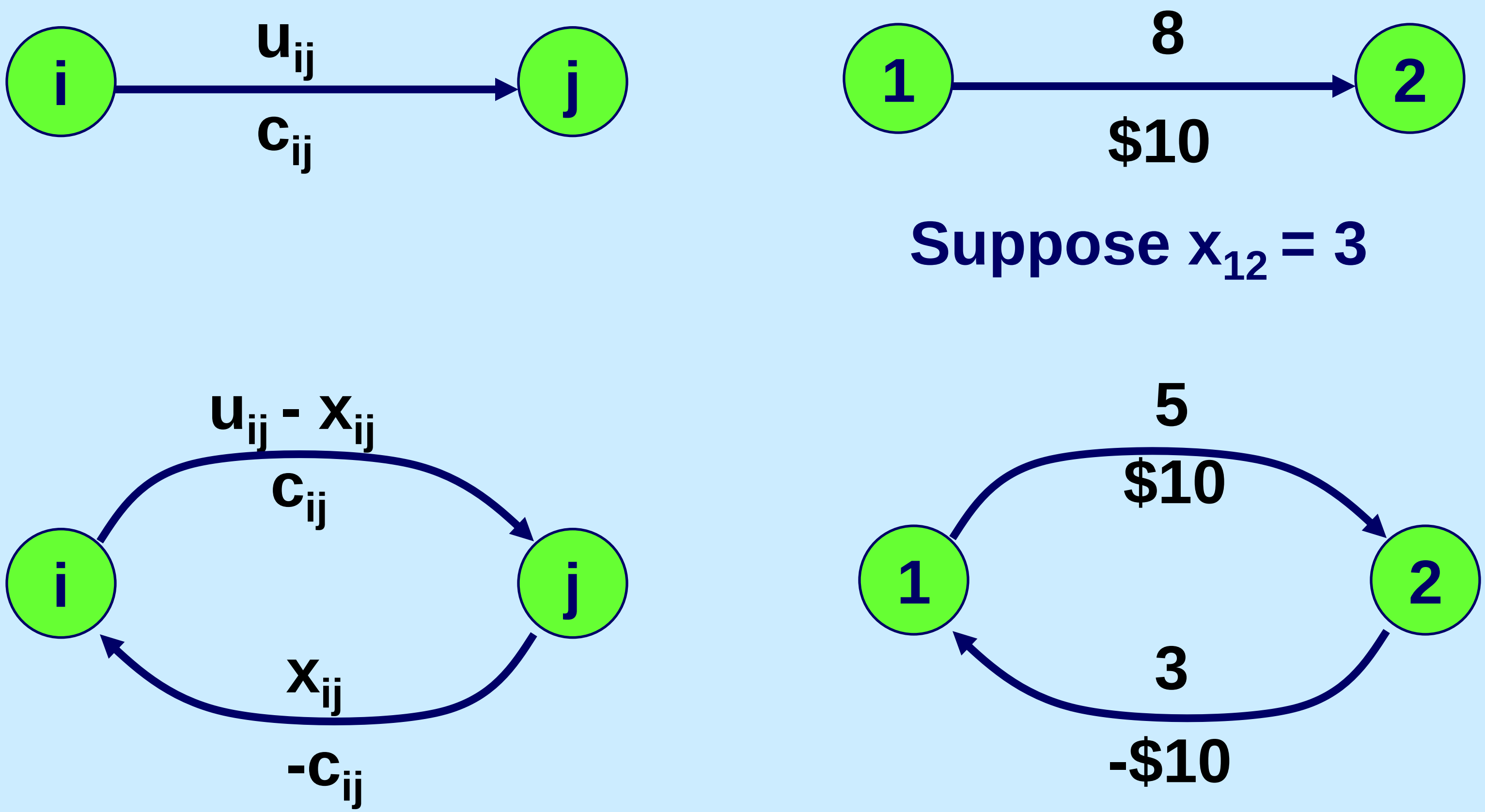
What do the following nicknames for sports teams have in common: the Miami Heat, the Minnesota Wild, the Utah Jazz, the Boston Red Sox, and the Chicago White Sox ?

None of them ends in the letter s.

In which country is kite flying a professional sport?

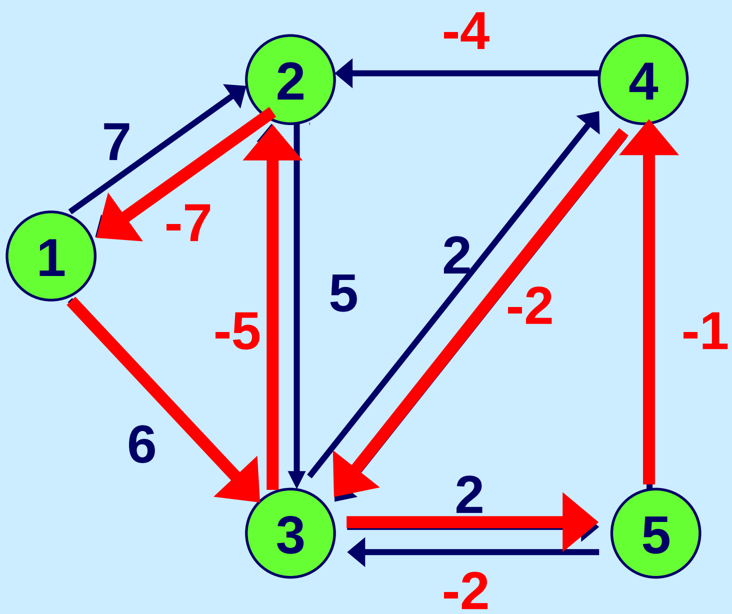
Thailand.

The Residual Network G(x)



Reducing the flow in (i, j) by 1 is equivalent to sending one unit of flow from j to i. It reduces the cost by c_{ij}.

Negative cost cycles and augmentations



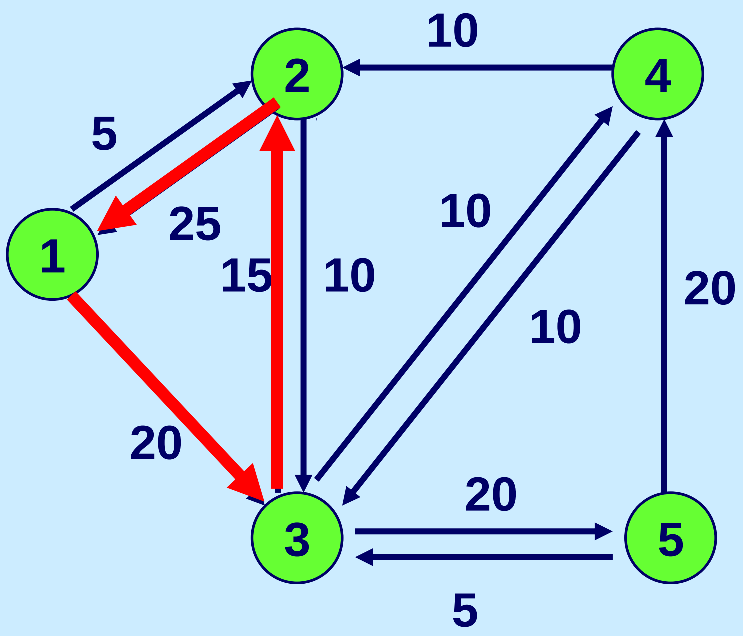
A residual network G(x) and its <u>arc costs</u>.

Note: each arc of G(x) has a cost and a capacity.

Typically, we will only show one of them.

A negative cost cycle refers to a directed cycle in G(x) whose total cost is negative, e.g., 1-3-2-1 and 3-5-4-3

Negative cost cycles and augmentations

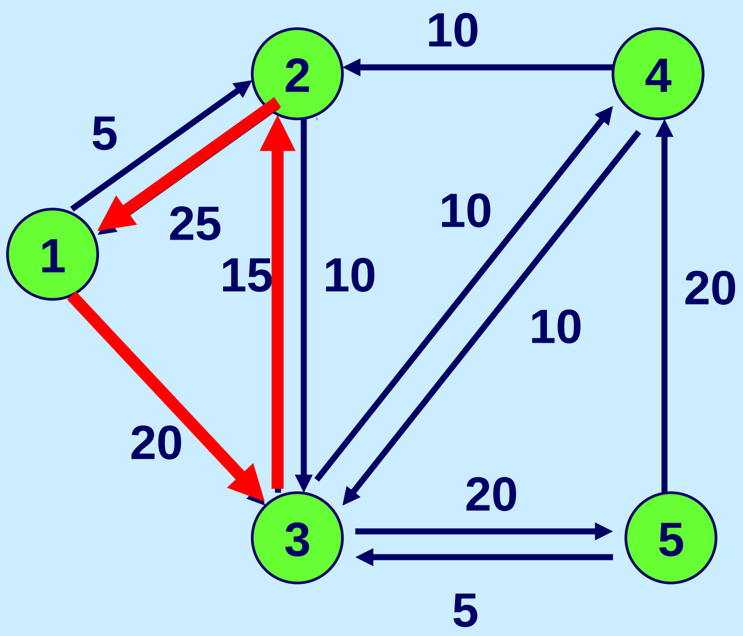


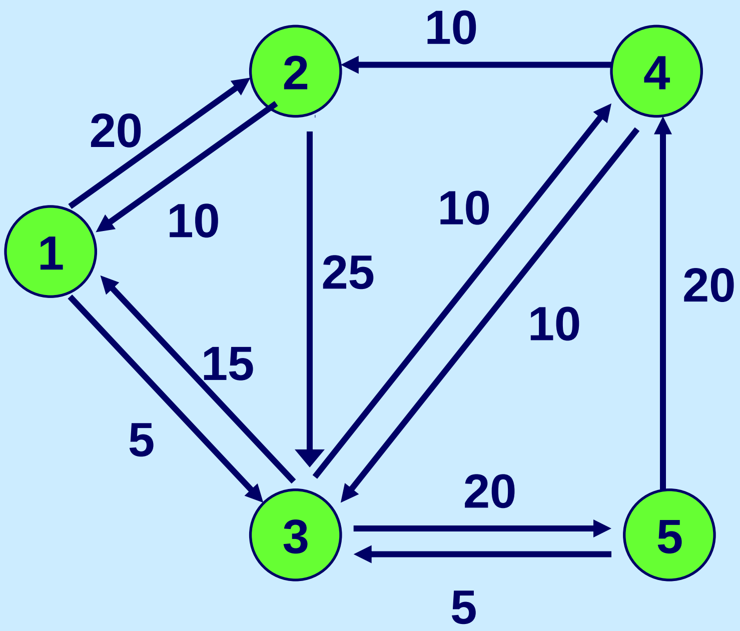
The capacities of the residual network G(x).

To **augment around a cycle** C is to send δ units of flow around C, where δ = min $\{r_{ij} : (i, j) \in C\}$

The cycle 1-3-2-1 had negative cost. Its capacity is 15.

The impact of an augmentation





The capacities of G(x) before the augmentation.

The capacities after the augmentation.

Negative Cycle Algorithm

(also known as the cycle canceling algorithm)

Algorithm NEGATIVE CYCLE;

establish a feasible flow x in the network;

while G(x) contains a negative cost cycle *do*

use some algorithm to identify a negative cost cycle C in G(x);

let $\delta := \min \{ r_{ij} : (i, j) \in C \}$;

augment δ units of flow in cycle C;

update G(x);

Negative Cycle Algorithm

More on the Negative Cycle Algorithm

Suppose that all supplies/demands are integral, and capacities are integral. Then the negative cycle algorithm maintains an integral solution at each iteration.

Finiteness

Theorem. The Negative Cycle Algorithm is finite if all data are finite and integral.

Proof. By flow decomposition, we can express the min cost flow as the sum of n+m paths and cycles. Each path and cycle has a cost bounded by nC, where $C = \max(|c_{ij}| : (i,j) \in A)$. The cost of the flow is at most $(nC)(n+m)U$, where U is the largest capacity.

At each iteration of cycle canceling, the cost improves by at least one.

Optimality

Theorem. ***The Negative Cycle Algorithm terminates with an optimal flow.***

Proof. Consider the final residual network G(x*). Suppose that x* is not optimal. Equivalently, the flow y = 0 is not optimal for the circulation problem in G(x*).

Let y* be a minimum cost circulation in G(x*). Then y* can be decomposed into flows around cycles. At least one of these cycles (say C) has negative cost. But this contradicts that the algorithm terminated.

Reduced costs and optimality conditions

Reduced costs

- recall replacing $c_{ij} - \pi_i + \pi_j$ for the shortest path problem. The same transformation is very useful for min cost flow algorithms.

Optimality conditions

- Most iterative optimization algorithms stop when "optimality conditions are satisfied". We describe optimality conditions for the min cost flow problem.

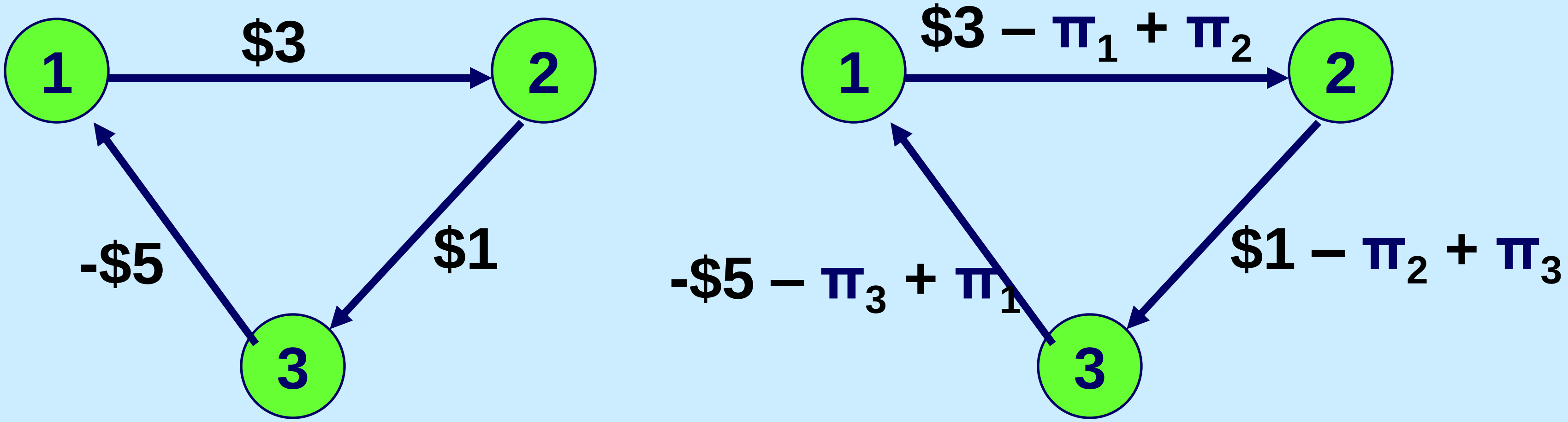
Reduced Costs in G(x)

Let π_i denote the node potential for node i.

$$c_{ij}^{\pi} = c_{ij} - \pi_i + \pi_j$$

For unit of flow out of node i, subtract π_i from the cost.

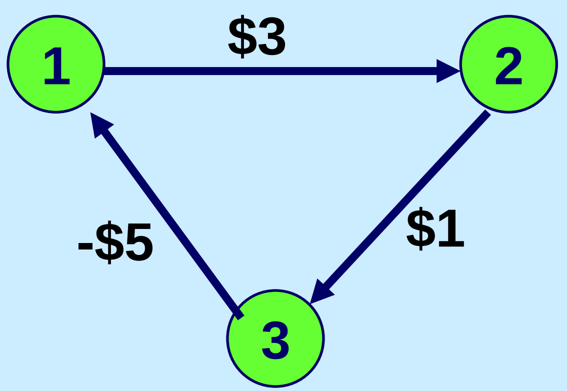
For unit of flow into node j, add π_j to the cost.

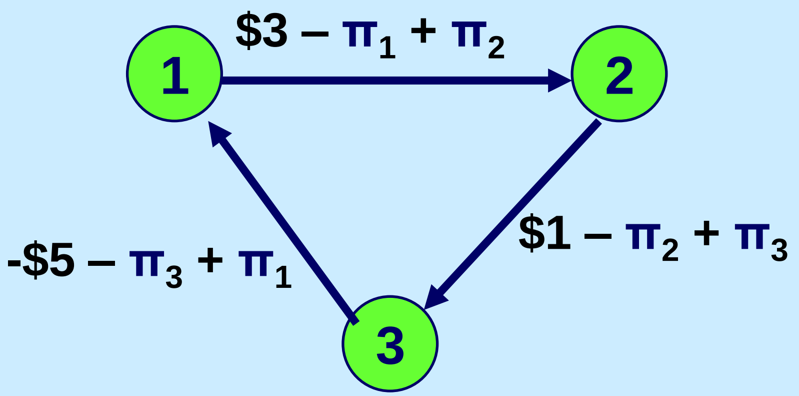


More on Reduced Costs

Lemma. **The reduced cost of a cycle is the cost of a cycle.**

Proof: $$c^{\pi}(C) = \sum_{(i,j)\in C} c_{ij} - \pi_i + \pi_j = \sum_{(i,j)\in C} c_{ij} = c(C)$$





Corollary. **Optimizing with respect to reduced costs is equivalent to optimizing with respect to the original costs.**

Optimality Conditions

Theorem. A flow x^* is optimal if and only if there is a vector π^* so that $c^{\pi^*}_{ij} \geq 0$ for all $(i, j) \in G(x^*)$.

Proof. We already know that x^* is optimal if and only if there is no negative cost cycle in $G(x^*)$. It remains to show that there is no negative cycle in $G(x^*)$ if $\exists\ \pi^*$ so that $c^{\pi^*}_{ij} \geq 0$ for all $(i, j) \in G(x^*)$.

Suppose first that there is such a vector π^*.

Then the reduced cost of every cycle in $G(x^*)$ must be non-negative

Optimality Conditions

Proof. Continued.

Suppose now that there is no negative cycle cycle in $G(x^*)$.

Let $d(i)$ be the shortest path length from node 1 to node i in $G(x^*)$ using original costs. (Assume that such a path exists).

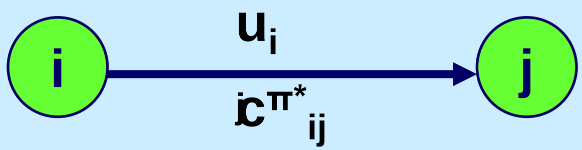
Then for all (i, j) in $G(x^*)$, $d(j) \leq d(i) + c_{ij}\ \forall\ (i, j) \in G(x^*)$

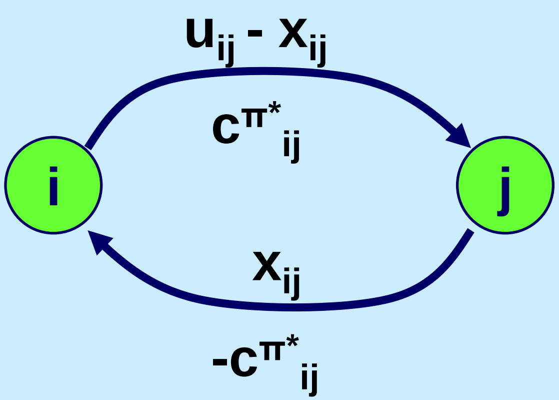
Let $\pi^*_i = - d(i)$ for all i.

Then $c^{\pi^*}_{ij} = c_{ij} + d(i) - d(j) \geq 0$ for all $(i, j) \in G(x^*)$. QED

Optimality conditions for the original network

If $c^{\pi^*}_{ij} \geq 0$ for all $(i, j) \in G(x^*)$, what is true about the original network?





Optimality conditions.

1. If $x^*_{ij} = 0$, then $(j, i) \notin G(x^*)$ and $c^{\pi^*}_{ij} \geq 0$.
2. If $x^*_{ij} = u_{ij}$, then $(i, j) \notin G(x^*)$ and $c^{\pi^*}_{ij} \leq 0$ (so that $c^{\pi^*}_{ji} \geq 0$)
3. If $0 < x^*_{ij} < u_{ij}$, then $(i, j) \in G(x^*)$ and $(j, i) \in G(x^*)$ and $c^{\pi^*}_{ij} = 0$.

Optimality conditions again

Optimality conditions 1.

1. If $x^*_{ij} = 0$, then $c^{\pi^*}_{ij} \geq 0$.
2. If $x^*_{ij} = u_{ij}$, then $c^{\pi^*}_{ij} \leq 0$.
3. If $0 < x^*_{ij} < u_{ij}$, then $c^{\pi^*}_{ij} = 0$.

Optimality conditions 2.

1. If $c^{\pi^*}_{ij} > 0$ then $x^*_{ij} = 0$,
2. If $c^{\pi^*}_{ij} < 0$ then $x^*_{ij} = u_{ij}$,
3. If $c^{\pi^*}_{ij} = 0$, then $0 \leq x^*_{ij} \leq u_{ij}$.

Opt conditions 1 are equivalent to optimality conditions 2.

Review of Cycle Canceling

Given a flow x, we look for negative cost cycles in G(x).

- If we find a negative cost cycle, we sent flow around the cycle
- If we don't find a negative cost cycle, we establish optimality.

It is a very generic algorithm for solving minimum cost flows.

Key subroutine: finding a negative cost cycle. It can be done in different ways.

How to Find a Negative Cycle

POSSIBILITY 1. Use a shortest path algorithm to determine a negative cost cycle.

POSSIBILITY 2. Find the most negative cost cycle.

POSSIBILITY 3. Augment along the cycle that minimizes COST(C)/|C|. (The cost divided by the number of arcs.)

Summary

- Some applications of the minimum cost flow problem
- Cycle Canceling Algorithm
- Integrality Property for Minimum Cost Flows
- Optimality Conditions