CS 170 Final Review

Allen Tang, Raymond Chan

05/04/2017

Linear Programming & Max Flow

Linear Program

Formulating LPs

- Decision Variables
  - Things you have control over.
- Objective Function
  - Optimize a goal. Minimize or maximize.
- Constraints
  - Optimize objective while satisfying restrictions to variables.

Duality

- Suppose we have a maximization LP with an optimal value of z^*. (Primal)
- How do the constraints affect z^*?
- Suppose we have an upper bound of the LP in z_u, $z^* \leq z_u$.
- We can minimize z_u such that $z^* = z_u$.
- The smallest upper bound must be the optimal solution to the primal LP.
- We can weight the constraints in such a way to minimize the upper bound. (Dual)
- These weights will upper bound the coefficients

Duality

Find the dual of the following primal LP:

$$\max -2x_1 + x_2$$
$$x_2 \leq x_1 + 1$$
$$x_1 + x_2 \leq 5$$
$$x_1, x_2 \geq 0$$

Duality

Rewriting and multiplying by *y* multipliers:

$$\max -2x_1 + x_2$$

y_1 $x_2 - x_1 \leq 1$

y_2 $x_1 + x_2 \leq 5$

$$x_1, x_2 \geq 0$$

Duality

Rewriting the multipliers and constraints:

$$y_1 (x_2 - x_1) + y_2 (x_1 + x_2) \leq y_1 + 5y_2$$

$$\max -2x_1 + x_2$$

$$x_2 \leq x_1 + 1$$

$$x_1 + x_2 \leq 5$$

$$x_1, x_2 \geq 0$$

Duality

Rewriting the multipliers and constraints:

$$y_1 (x_2 - x_1) + y_2 (x_1 + x_2) \leq y_1 + 5y_2$$

$$(-y_1 + y_2) x_1 + (y_1 + y_2) x_2 \leq y_1 + 5y_2$$

$$\max -2x_1 + x_2$$

$$x_2 \leq x_1 + 1$$

$$x_1 + x_2 \leq 5$$

$$x_1, x_2 \geq 0$$

Duality

Adding upper bound to primal objective

$$y_1 (x_2 - x_1) + y_2 (x_1 + x_2) \leq y_1 + 5y_2$$

$$(-y_1 + y_2) x_1 + (y_1 + y_2) x_2 \leq y_1 + 5y_2$$

$$-2 x_1 + x_2 \leq (-y_1 + y_2) x_1 + (y_1 + y_2) x_2 \leq y_1 + 5y_2$$

$$\max -2x_1 + x_2$$
$$x_2 \leq x_1 + 1$$
$$x_1 + x_2 \leq 5$$
$$x_1, x_2 \geq 0$$

Duality

Writing the dual and nonnegativity constraints

$$-2x_1 + x_2 \leq (-y_1 + y_2)x_1 + (y_1 + y_2)x_2 \leq y_1 + 5y_2$$

$$\max -2x_1 + x_2$$
$$x_2 \leq x_1 + 1$$
$$x_1 + x_2 \leq 5$$
$$x_1, x_2 \geq 0$$

Dual

$$\min y_1 + 5y_2$$
$$\text{s.t. } -y_1 + y_2 \geq -2$$
$$y_1 + y_2 \geq 1$$
$$y_1, y_2 \geq 0$$

Duality

$$\max -2x_1 + x_2$$
$$x_2 \leq x_1 + 1$$
$$x_1 + x_2 \leq 5$$
$$x_1, x_2 \geq 0$$

Writing the dual and nonnegativity constraints

$$-2\ x_1 + x_2 \leq (-y_1 + y_2)\ x_1 + (y_1 + y_2)\ x_2 \leq y_1 + 5y_2$$

Dual

$$\min\ y_1 + 5y_2$$
$$\text{s.t. } -y_1 + y_2 \geq -2$$
$$y_1 + y_2 \geq 1$$
$$y_1,\ y_2 \geq 0$$

Nonnegativity constraints are added so that the sign in the primal constraints do not get flipped.

Duality

What is the optimal solution? What are the optimal decision variables for both the primal LP and the dual LP?

$$\max -2x_1 + x_2$$
$$x_2 \leq x_1 + 1$$
$$x_1 + x_2 \leq 5$$
$$x_1, x_2 \geq 0$$

Dual:

$$\min y_1 + 5y_2$$
$$\text{s.t. } -y_1 + y_2 \geq -2$$
$$y_1 + y_2 \geq 1$$
$$y_1, y_2 \geq 0$$

Duality

What is the optimal solution? What are the optimal decision variables for both the primal LP and the dual LP?

$$\max -2x_1 + x_2$$
$$x_2 \leq x_1 + 1$$
$$x_1 + x_2 \leq 5$$
$$x_1, x_2 \geq 0$$

- Optimal value is 1 (you can plot it)
- $x^* = (0, 1)$
- $y^* = (1, 0)$

Dual:

$$\min y_1 + 5y_2$$
$$\text{s.t. } -y_1 + y_2 \geq -2$$
$$y_1 + y_2 \geq 1$$
$$y_1, y_2 \geq 0$$

Duality

Suppose the first constraint for the primal changes to $x_2 = x_1 + 1$, would the optimum change? Would the dual linear program change?

$$\max -2x_1 + x_2$$
$$x_2 \leq x_1 + 1$$
$$x_1 + x_2 \leq 5$$
$$x_1, x_2 \geq 0$$

Original Dual:

$$\min y_1 + 5y_2$$
$$\text{s.t. } -y_1 + y_2 \geq -2$$
$$y_1 + y_2 \geq 1$$
$$y_1, y_2 \geq 0$$

- $z^* = 1$
- $x^* = (0, 1)$
- $y^* = (1, 0)$

Duality

Suppose the first constraint for the primal changes to $x_2 = x_1 + 1$, would the optimum change? Would the dual linear program change?

- Looking at x^*, we see that the optimal will not change. $1 \leq 0 + 1$
- The current optimal values satisfy new LP.

$$\max -2x_1 + x_2$$
$$x_2 \leq x_1 + 1$$
$$x_1 + x_2 \leq 5$$
$$x_1, x_2 \geq 0$$

Original Dual:

$$\min y_1 + 5y_2$$
$$\text{s.t. } -y_1 + y_2 \geq -2$$
$$y_1 + y_2 \geq 1$$
$$y_1, y_2 \geq 0$$

- $z^* = 1$
- $x^* = (0, 1)$
- $y^* = (1, 0)$

Duality

Suppose the first constraint for the primal changes to $x_2 = x_1 + 1$, would the optimum change? Would the dual linear program change?

- Looking at x^*, we see that the optimal will not change. $1 \leq 0 + 1$
- The current optimal values satisfy new LP.
- Dual LP will change: no nonnegativity for y_1.

$$\max -2x_1 + x_2$$
$$x_2 \leq x_1 + 1$$
$$x_1 + x_2 \leq 5$$
$$x_1, x_2 \geq 0$$

Original Dual:

$$\min y_1 + 5y_2$$
$$\text{s.t. } -y_1 + y_2 \geq -2$$
$$y_1 + y_2 \geq 1$$
$$y_1, y_2 \geq 0$$

- $z^* = 1$
- $x^* = (0, 1)$
- $y^* = (1, 0)$

Zero Sum Games

- Two players play a game alternating turns.
- A player's gain is the other player's loss.
- Mixed strategies are optimal:
  - Each player declare probability of choosing each action.
- Payoff matrix is with respect to one player.

Zero Sum Games

Alice and Bob are playing a zero-sum game whose payoff matrix is shown below. The $ij^{\{th\}}$ entry of the matrix shows the payoff that Alice receives if she plays strategy i and Bob plays strategy j. Alice is the row player and is trying to maximize her payoff.

	A	B
1	4	1
2	2	5

Now we will write a linear program to find a strategy that maximizes Alice's payoff.

Zero Sum Games

Alice and Bob are playing a zero-sum game whose payoff matrix is shown below. The $ij\{th\}$ entry of the matrix shows the payoff that Alice receives if she plays strategy i and Bob plays strategy j. Alice is the row player and is trying to maximize her payoff.

	A	B
1	4	1
2	2	5

Now we will write a linear program to find a strategy that maximizes Alice's payoff.

- Let x_i be the probability that Alice plays strategy i
- Bob will try to minimize Alice's payoff.

Zero Sum Games

Alice and Bob are playing a zero-sum game whose payoff matrix is shown below. The $ij\{th\}$ entry of the matrix shows the payoff that Alice receives if she plays strategy i and Bob plays strategy j. Alice is the row player and is trying to maximize her payoff.

	A	B
1	4	1
2	2	5

Now we will write a linear program to find a strategy that maximizes Alice's payoff.

- Let x_i be the probability that Alice plays strategy i
- Bob chooses A; Alice's payoff: $4x_1 + 2x_2$
- Bob chooses B; Alice's payoff: $x_1 + 5x_2$

Zero Sum Games

Alice and Bob are playing a zero-sum game whose payoff matrix is shown below. The $ij\{th\}$ entry of the matrix shows the payoff that Alice receives if she plays strategy i and Bob plays strategy j. Alice is the row player and is trying to maximize her payoff.

	A	B
1	4	1
2	2	5

Now we will write a linear program to find a strategy that maximizes Alice's payoff.

- Let x_i be the probability that Alice plays strategy i
- Bob chooses A; Alice's payoff: $4x_1 + 2x_2$
- Bob chooses B; Alice's payoff: $x_1 + 5x_2$
- Bob's choice: min $(4x_1 + 2x_2, x_1 + 5x_2)$

Zero Sum Games

Alice and Bob are playing a zero-sum game whose payoff matrix is shown below. The $ij^{\{th\}}$ entry of the matrix shows the payoff that Alice receives if she plays strategy i and Bob plays strategy j. Alice is the row player and is trying to maximize her payoff.

	A	B
1	4	1
2	2	5

Now we will write a linear program to find a strategy that maximizes Alice's payoff.

- Let x_i be the probability that Alice plays strategy i
- Bob's choice: min $(4x_1 + 2x_2, x_1 + 5x_2)$
- Alice's payoff = max { min $(4x_1 + 2x_2, x_1 + 5x_2)$ }

Zero Sum Games

- Let x_i be the probability that Alice plays strategy *i*
- Bob's choice: min $(4x_1 + 2x_2, x_1 + 5x_2)$
- Alice's payoff = max { min $(4x_1 + 2x_2, x_1 + 5x_2)$ }

Linear Program:

$$\begin{aligned} \max \ & p \\ & p \leq 4x_1 + 2x_2 \\ & p \leq x_1 + 5x_2 \\ & x_i \geq 0 \\ x_1 + x_2 &= 1 \end{aligned}$$

Max Flow

- Given a directed graph $G = (V, E)$, send as many units of flow from source s to sink t
- Satisfying positive edge capacity constraints.
- The number of units send from s must be received by t.

max $\quad \mathbf{size}(f) = \sum_{(s,u)\in E} f_{su}$

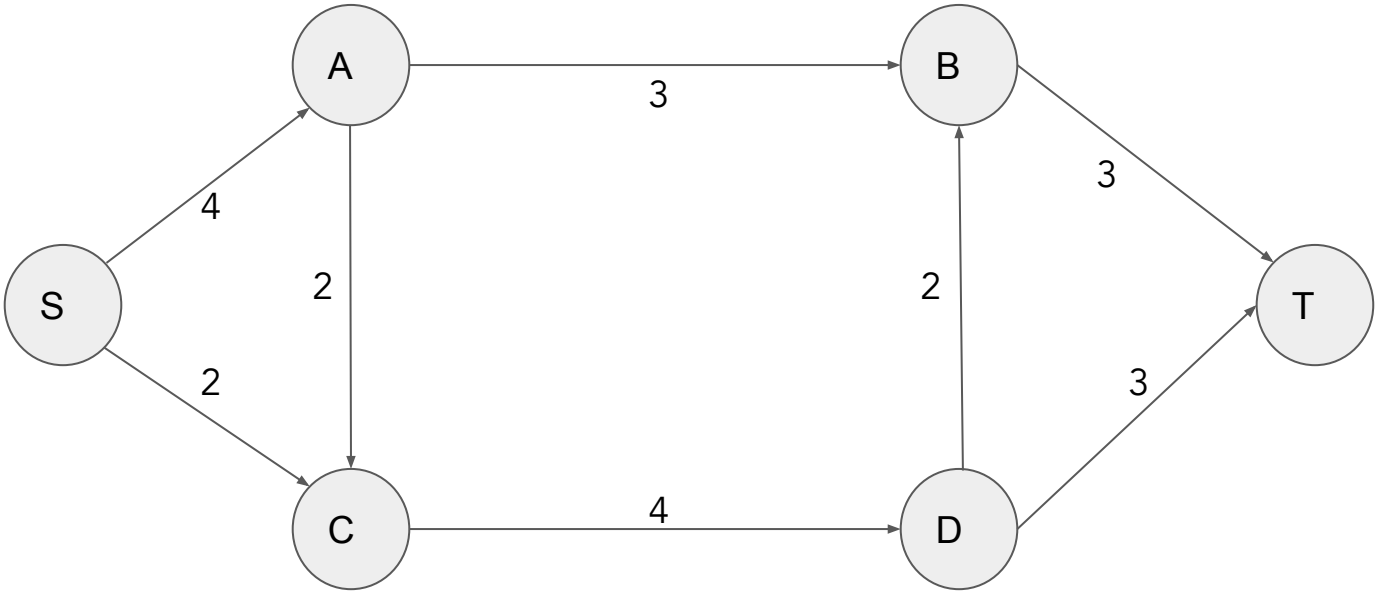
s.t. $\quad \sum_{(w,u)\in E} f_{wu} = \sum_{(u,z)\in E} f_{uz}$ for all $u \in V$ Flow conservation constraints.

$0 \leq f_e \leq c_e$ **for all** $e \in E$ Capacity constraints.

Max Flow

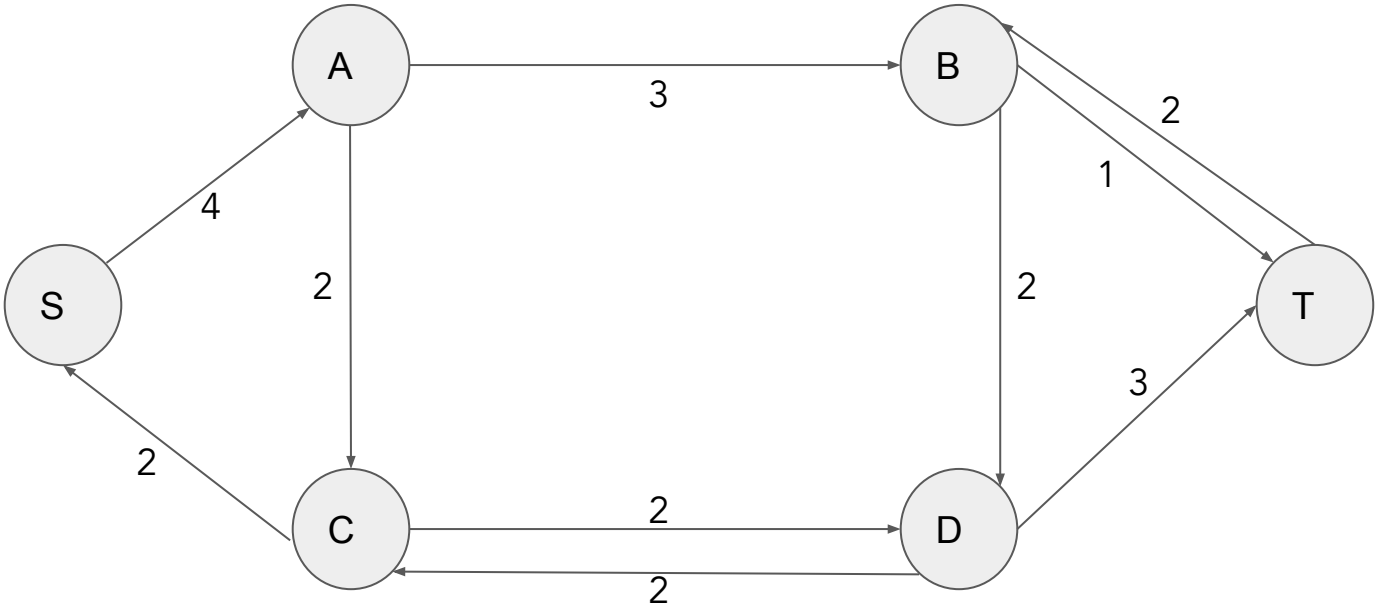
- Ford-Fulkerson algorithm
- When sending *x* units of flow, build up residual edges.
- Keep sending units of low until no path is available between *s* and *t* in the residual graph.
- Edmond-Karp algorithm: send as many units of flow as possible through shortest *s*-*t* path (BFS)
    - $O(|V||E|^2)$

Max Flow



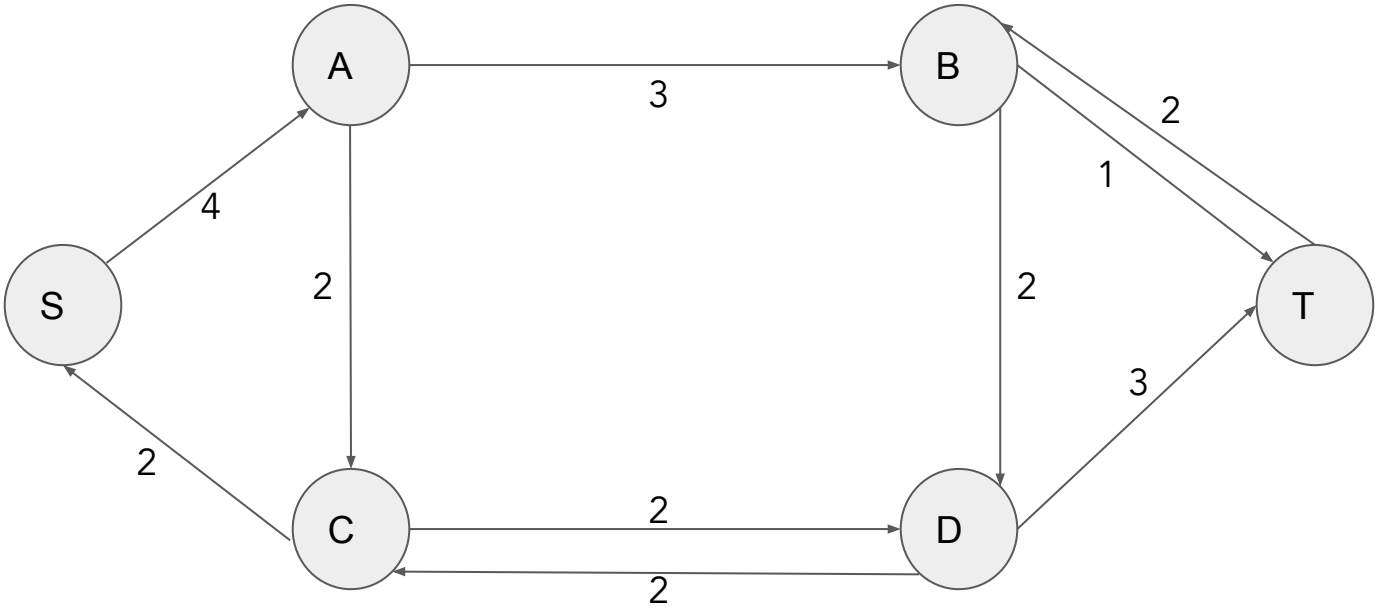
How many units of flow can we send through SCDBT?
Draw the residual graph after sending that many units of flow.

Max Flow



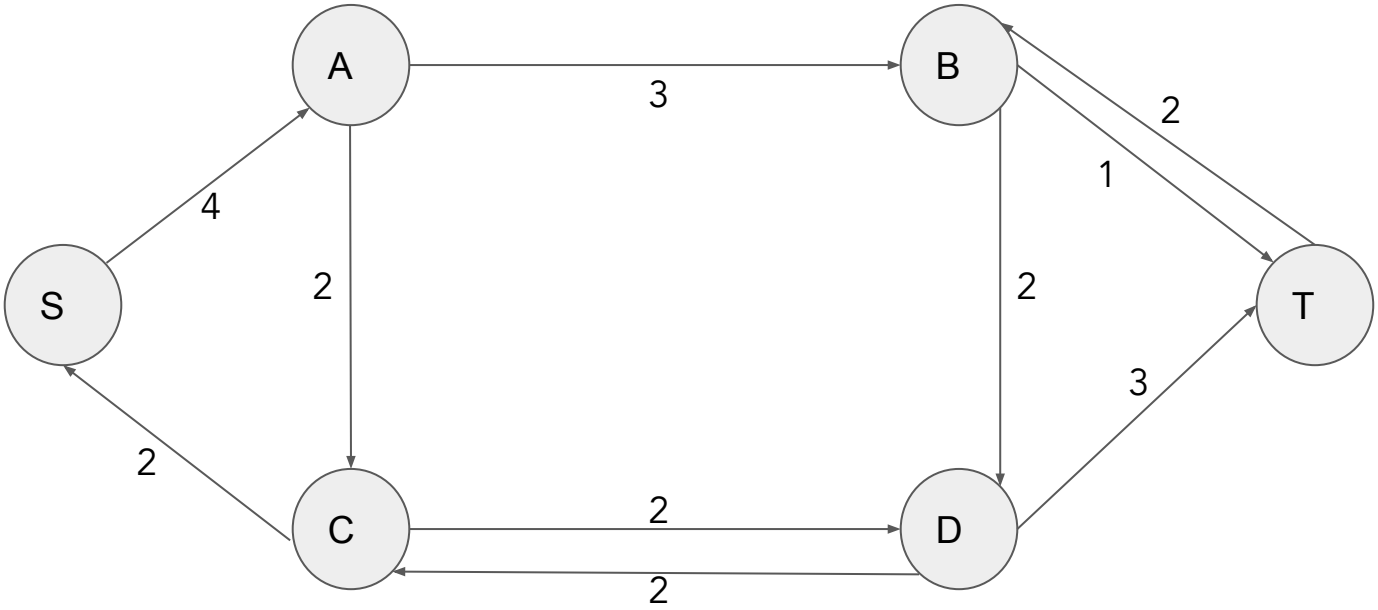
How many units of flow can we send through SCDBT? **2**
Draw the residual graph after sending that many units of flow.

Max Flow



Now find the max flow for the graph.

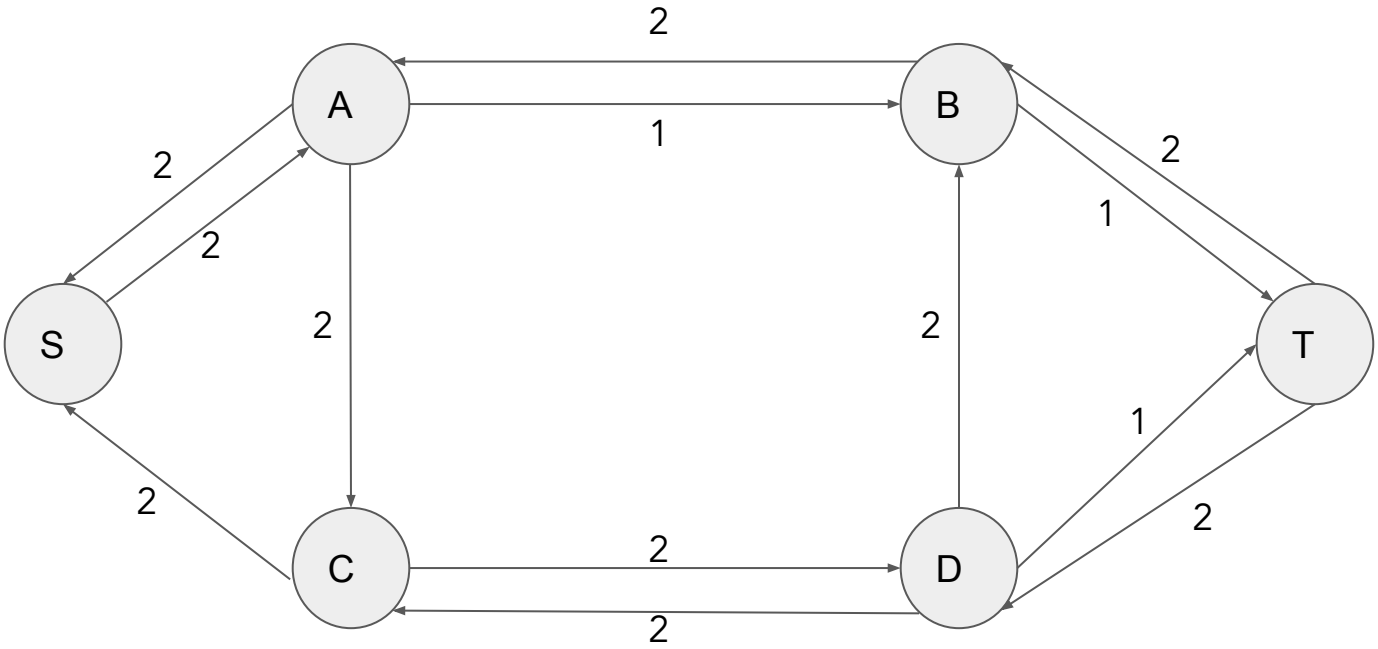
Max Flow



Send 2 units of flow through SCDBT

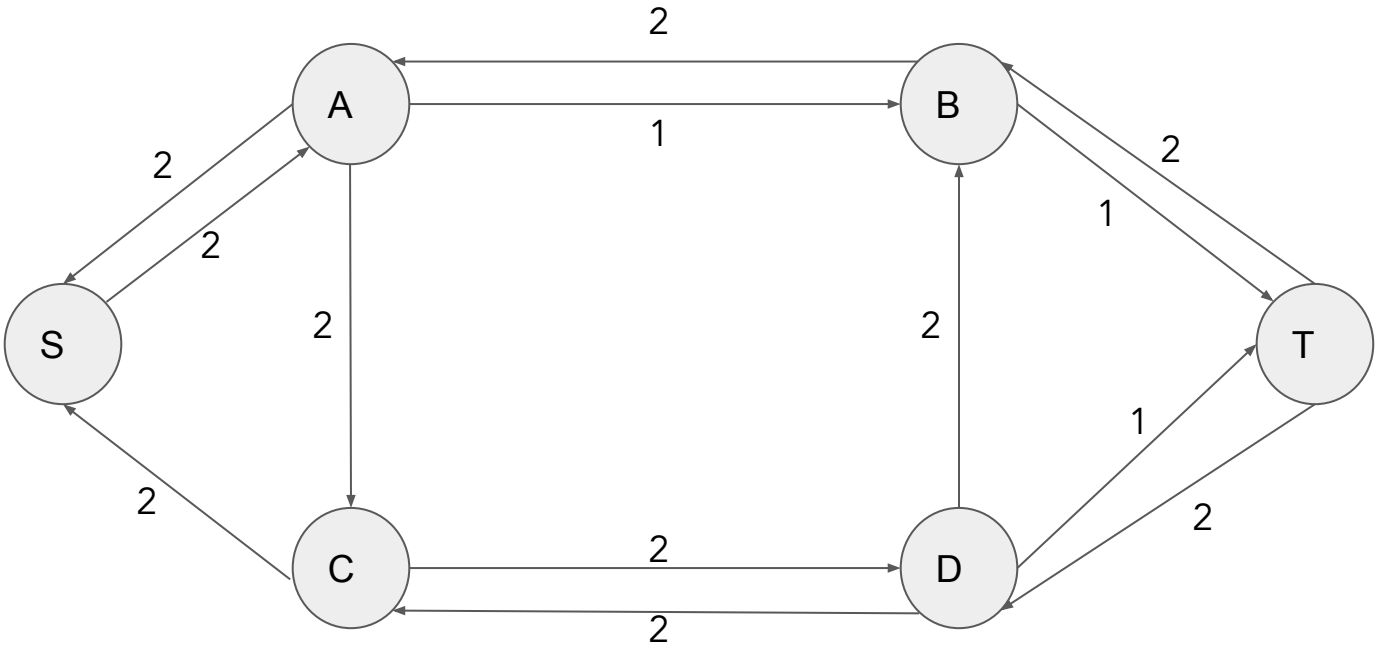
Send 2 units of flow through SABDT

Max Flow



Send 2 units of flow through SCDBT
Send 2 units of flow through SABDT

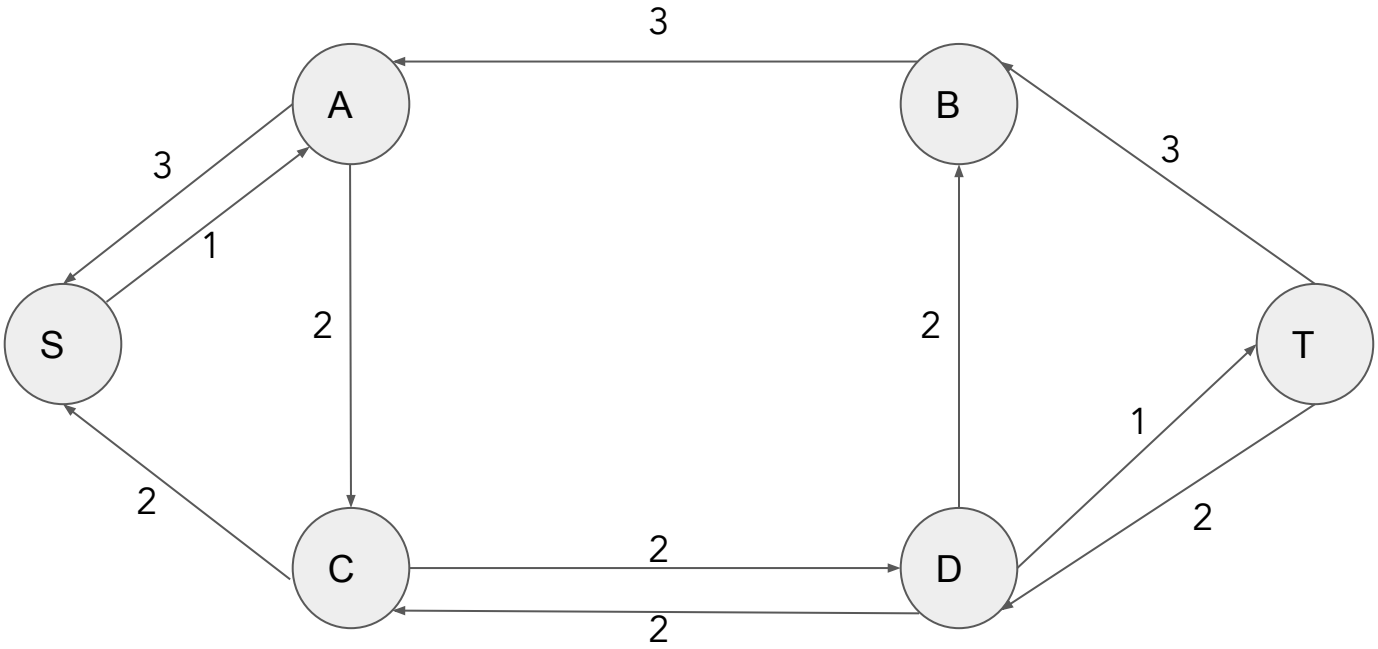
Max Flow



Send 2 units of flow through SCDBT
Send 2 units of flow through SABDT

Send 1 unit of flow through SABT

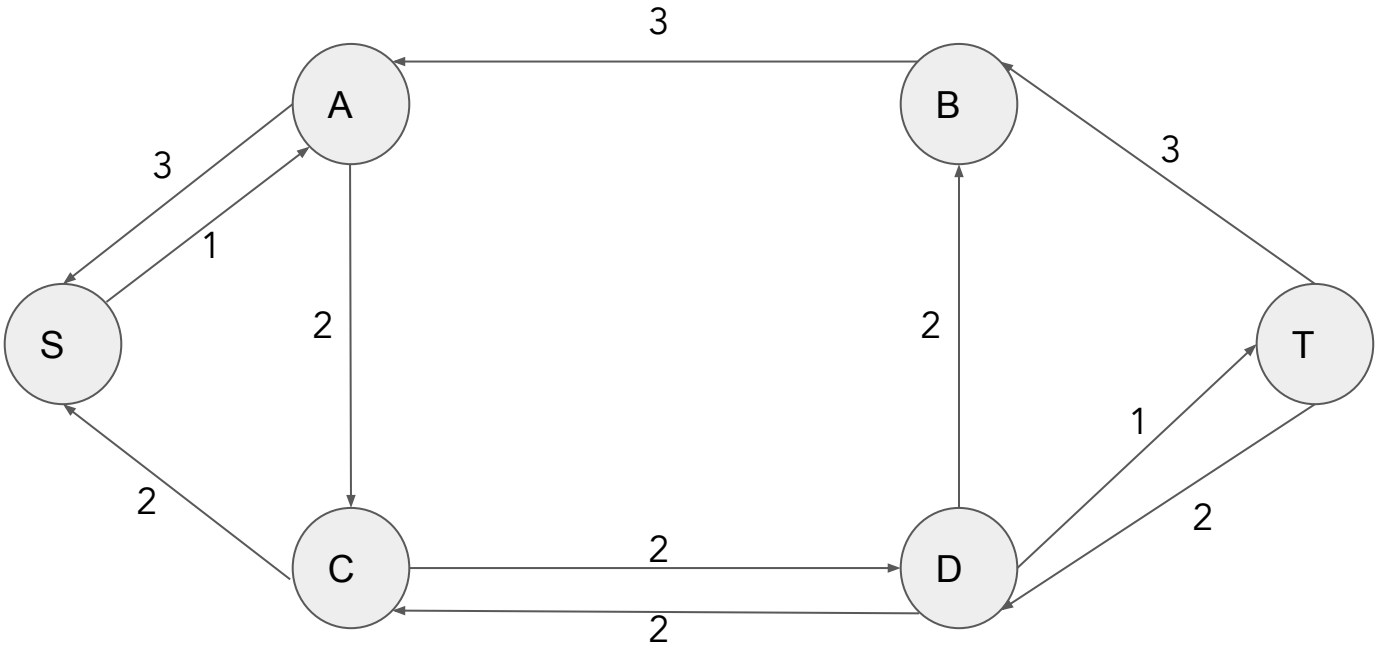
Max Flow



Send 2 units of flow through SCDBT
Send 2 units of flow through SABDT

Send 1 unit of flow through SABT

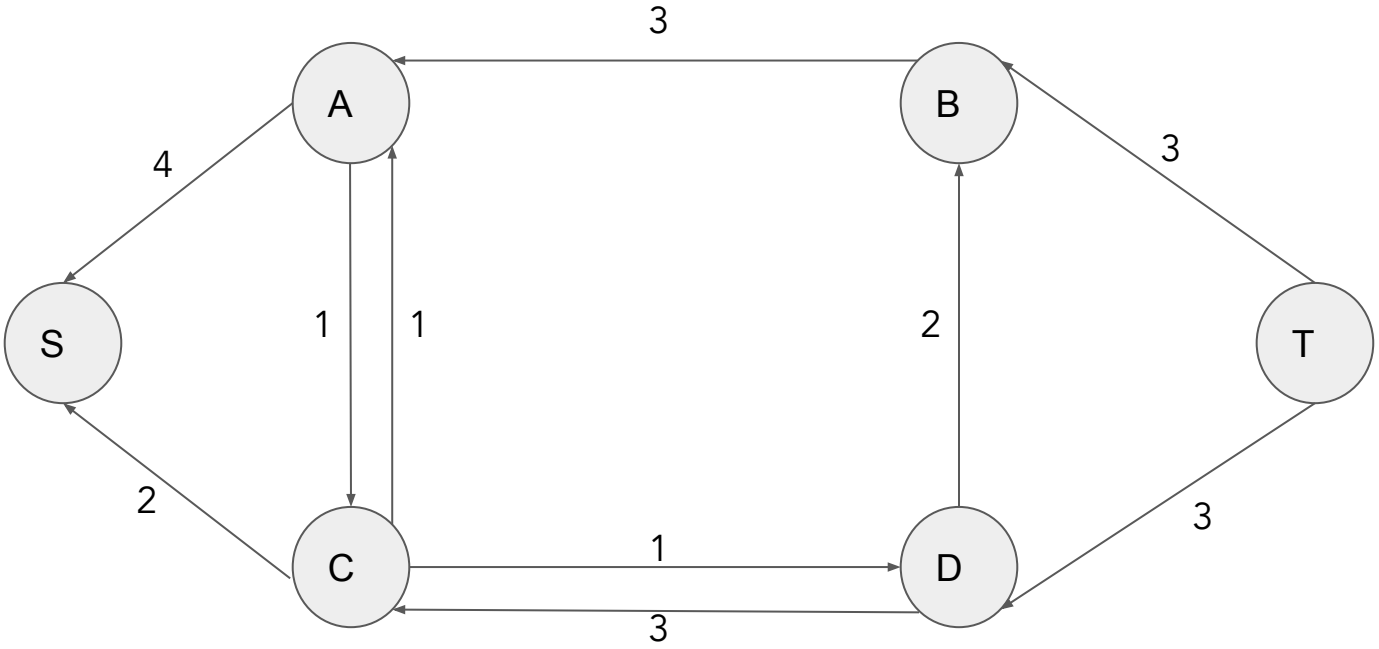
Max Flow



Send 2 units of flow through SCDBT
Send 2 units of flow through SABDT

Send 1 unit of flow through SABT
Send 1 unit of flow through SACDT

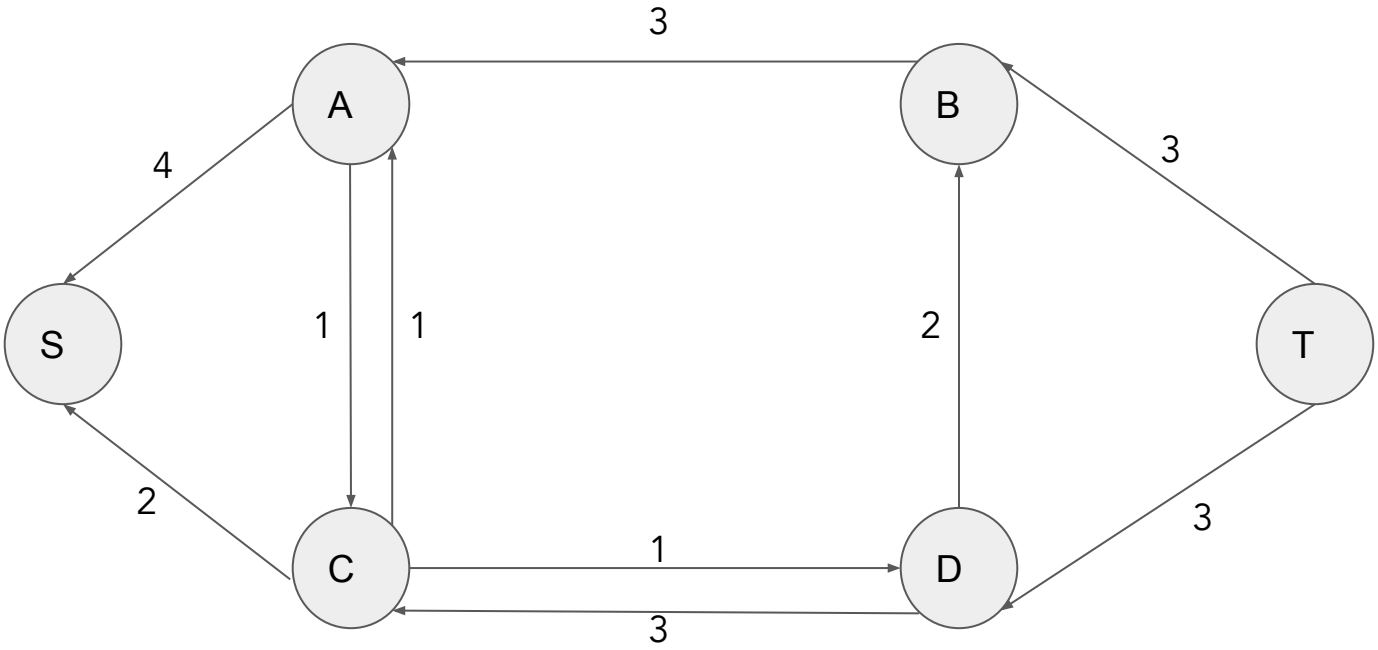
Max Flow



Send 2 units of flow through SCDBT
Send 2 units of flow through SABDT

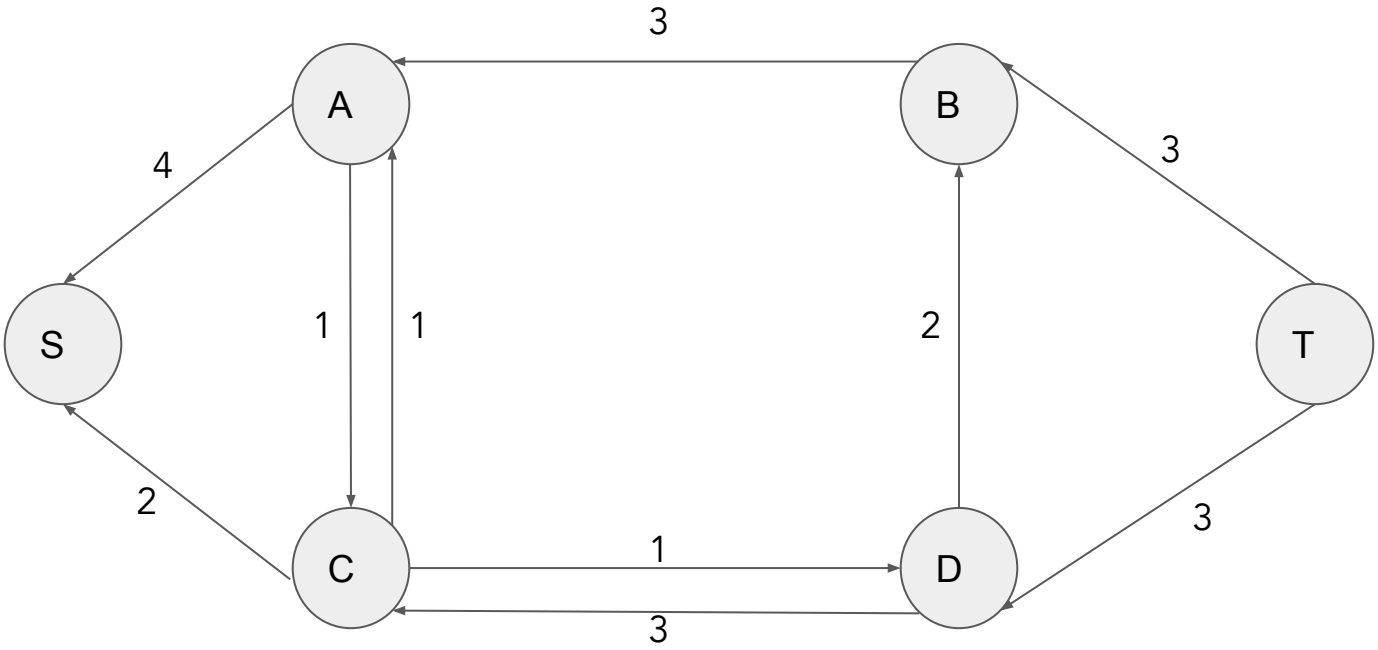
Send 1 unit of flow through SABT
Send 1 unit of flow through SACDT

Max Flow



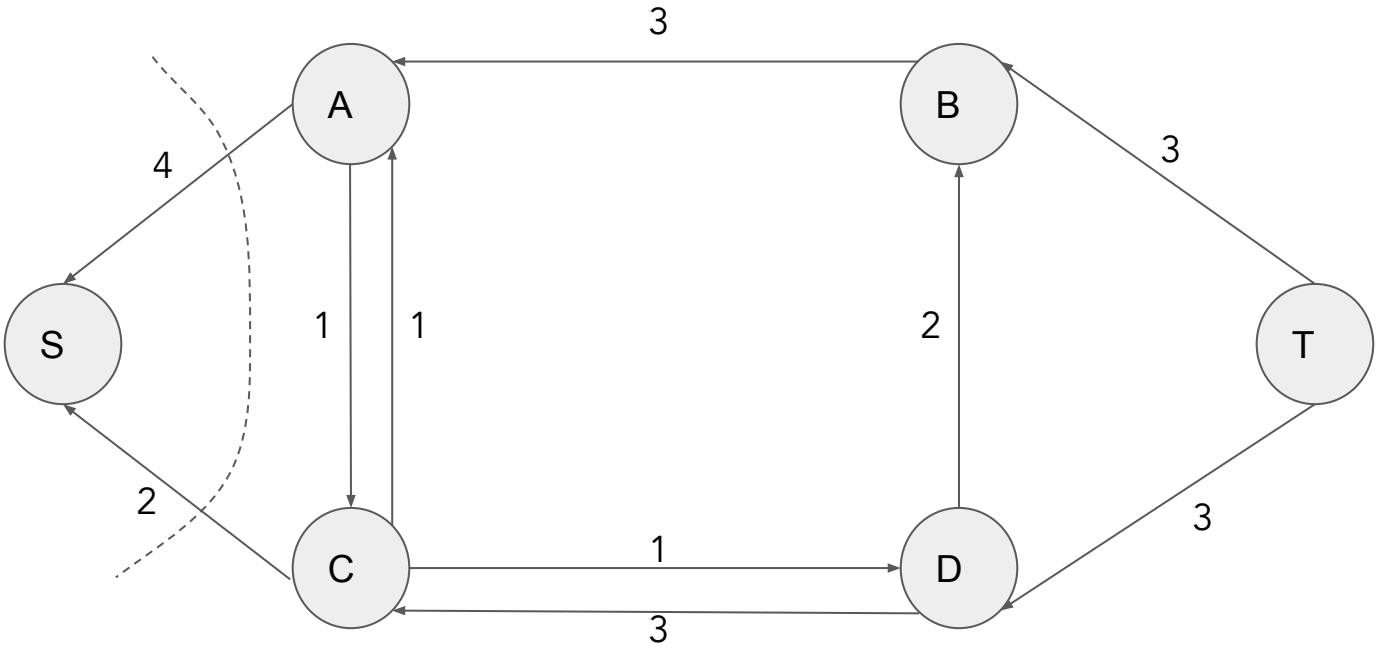
Send total 6 units of flow

Max Flow



What is the minimum cut?

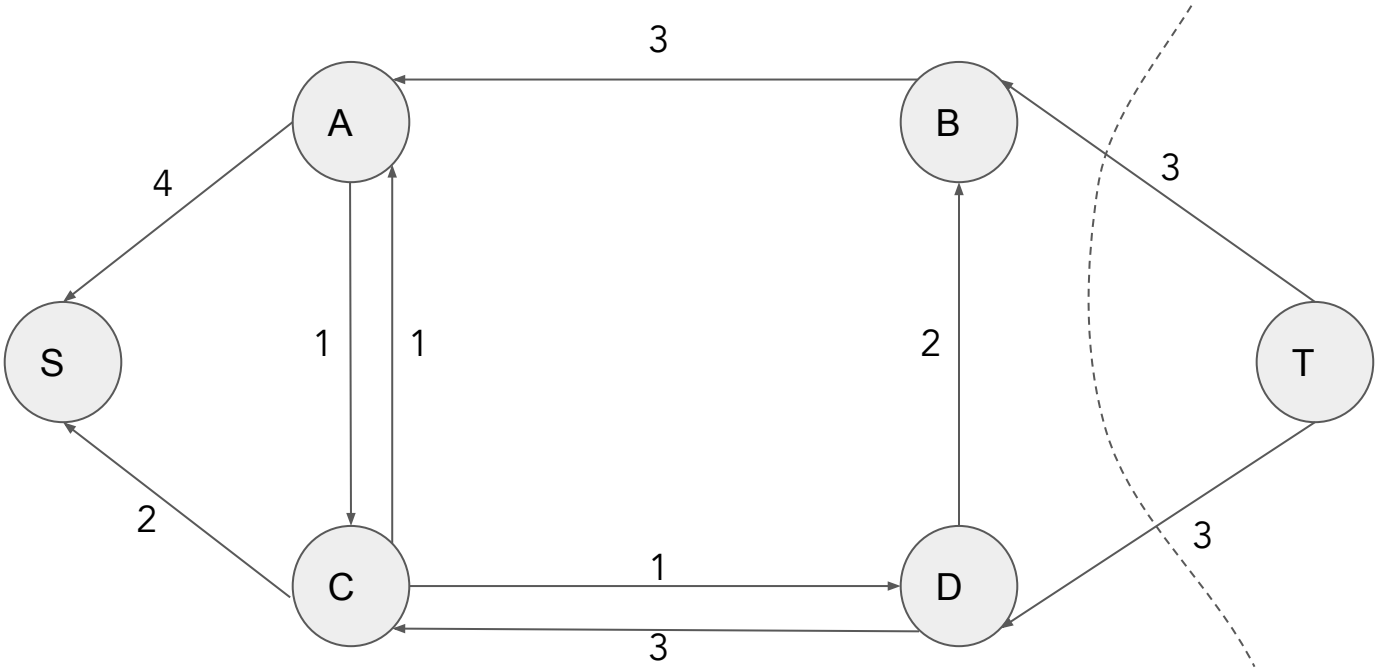
Max Flow



What is the minimum cut?

{S}, {A, B, C, D, T}

Max Flow



What is the minimum cut?
{S}, {A, B, C, D, T} or {S, A, B, C, D}, {T}

Breadth Requirements (Spring 2016 MT2)

UC Berkeley's newly formed College of Computer Science has several categories of breadth requirements. Each class is offered by a single department, and may match multiple categories. Students must take at least one class from each category, and at most k classes from each department. If a class matches multiple categories, you must choose which category you're using it to fulfill: no class can be used to fulfill more than one category.

Formally, there are m classes $c_1 ... c_m$, T departments $D_1 ... D_T$, and n categories of breadth requirements $G_1 ... G_n$. For some class c_i, let $H(c_i)$ be its department and $F(c_i)$ be the set of categories it can be used to fulfill. For example, you could have $H(c_{10}) = D_3$ and $F(c_{10}) = \{G_3, G_4, G_9\}$.

Given a subset of classes $c_1 ... c_p$, $G_1 ... G_n$, $D_1 ... D_T$, F, and H, you want to find whether the p classes satisfy all the breadth requirements subject to the above constraints.

Breadth Requirements (Spring 2016 MT2)

- What are the decision variables?

Breadth Requirements (Spring 2016 MT2)

- What are the decision variables?
  - There are m classes, T departments, and n requirements.
  - The departments are part of constraints.
  - Let x_{ij} be an indicator variable indicating if the i-th class is being used to satisfy the j-th requirement.

Breadth Requirements (Spring 2016 MT2)

- What are the decision variables?
  - There are m classes, T departments, and n requirements.
  - The departments are part of constraints.
  - Let x_{ij} be an indicator variable indicating if the i-th class is being used to satisfy the j-th requirement.
- What is the objective function?

Breadth Requirements (Spring 2016 MT2)

- What are the decision variables?
  - There are *m* classes, *T* departments, and *n* requirements.
  - The departments are part of constraints.
  - Let x_{ij} be an indicator variable indicating if the *i*-th class is being used to satisfy the *j*-th requirement.
- What is the objective function?
  - The question cares about feasibility rather than optimizing.
  - Obj Func: max 0
- Let's now talk through the constraints.

Breadth Requirements (Spring 2016 MT2)

$$\forall j \in \{1..n\} : \sum_i x_{ij} \geq 1$$

For each breadth, at least one class is used to satisfy it.

Breadth Requirements (Spring 2016 MT2)

$$\forall j \in \{1..n\} : \sum_i x_{ij} \geq 1$$

For each breadth, at least one class is used to satisfy it.

$$\forall l \in \{1..T\} : \sum_{\{i|H(c_i)=D_l\}} \sum_j x_{ij} \leq k$$

For each department, at most *k* courses can be used to satisfy all requirements.

Breadth Requirements (Spring 2016 MT2)

$$\forall j \in \{1..n\} : \sum_i x_{ij} \geq 1$$

For each breadth, at least one class is used to satisfy it.

$$\forall l \in \{1..T\} : \sum_{\{i|H(c_i)=D_l\}} \sum_j x_{ij} \leq k$$

For each department, at most *k* courses can be used to satisfy all requirements.

$$\forall i \in \{1..m\} : \sum_j x_{ij} \leq 1$$

Each class can only satisfy at most one requirement.

Breadth Requirements (Spring 2016 MT2)

$$\forall j \in \{1..n\} : \sum_i x_{ij} \geq 1$$

For each breadth, at least one class is used to satisfy it.

$$\forall l \in \{1..T\} : \sum_{\{i|H(c_i)=D_l\}} \sum_j x_{ij} \leq k$$

For each department, at most *k* courses can be used to satisfy all requirements.

$$\forall i \in \{1..m\} : \sum_j x_{ij} \leq 1$$

Each class can only satisfy at most one requirement.

$$\forall i,j \in \{(i,j)|G_j \in F(c_i)\} : x_{ij} \leq 1$$
$$\forall i,j \in \{(i,j)|G_j \notin F(c_i)\} : x_{ij} \leq 0$$

We can only use classes to satisfy requirements that they can satisfy.

Breadth Requirements (Spring 2016 MT2)

$$\forall j \in \{1..n\} : \sum_i x_{ij} \geq 1$$

For each breadth, at least one class is used to satisfy it.

$$\forall l \in \{1..T\} : \sum_{\{i|H(c_i)=D_l\}} \sum_j x_{ij} \leq k$$

For each department, at most *k* courses can be used to satisfy all requirements.

$$\forall i \in \{1..m\} : \sum_j x_{ij} \leq 1$$

Each class can only satisfy at most one requirement.

$$\forall i,j \in \{(i,j)|G_j \in F(c_i)\} : x_{ij} \leq 1$$
$$\forall i,j \in \{(i,j)|G_j \notin F(c_i)\} : x_{ij} \leq 0$$

We can only use classes to satisfy requirements that they can satisfy.

$$\forall i,j : x_{ij} \geq 0$$

Lower bound for decision variables. Upper bound implicit from 3rd constraint.

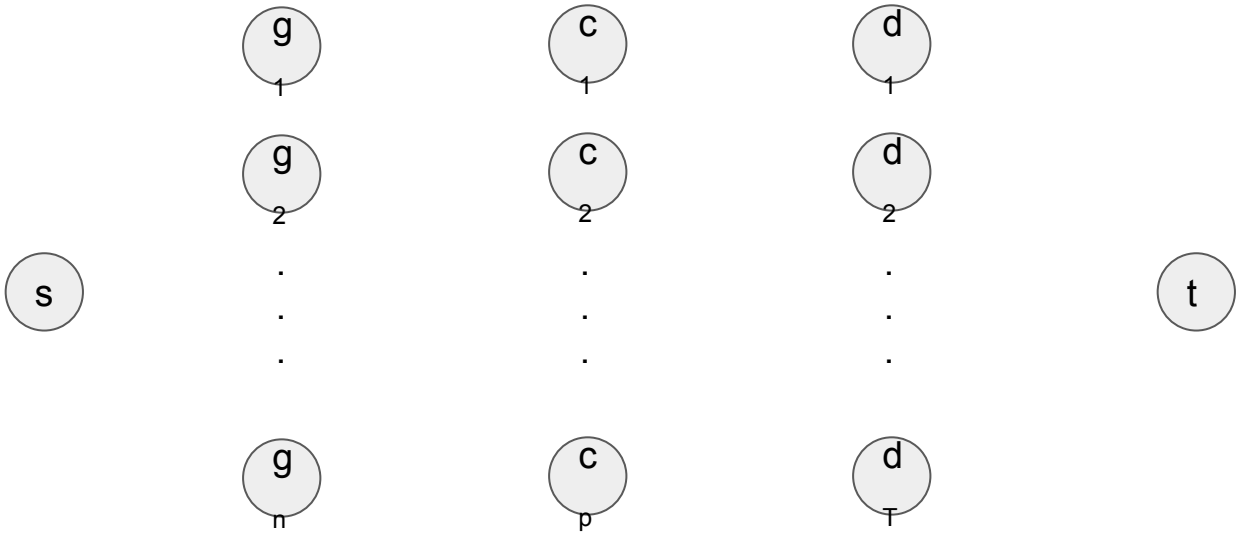
Breadth Requirements (Spring 2016 MT2)

(b) Find an efficient algorithm to determine whether the requirements are satisfied, and (if all requirements are satisfied) a mapping of which class contributes to which requirement.

Breadth Requirements (Spring 2016 MT2)

(b) Find an efficient algorithm to determine whether the requirements are satisfied, and (if all requirements are satisfied) a mapping of which class contributes to which requirement.

Reduce to max flow instance and solve in a way similar to bipartite matching.



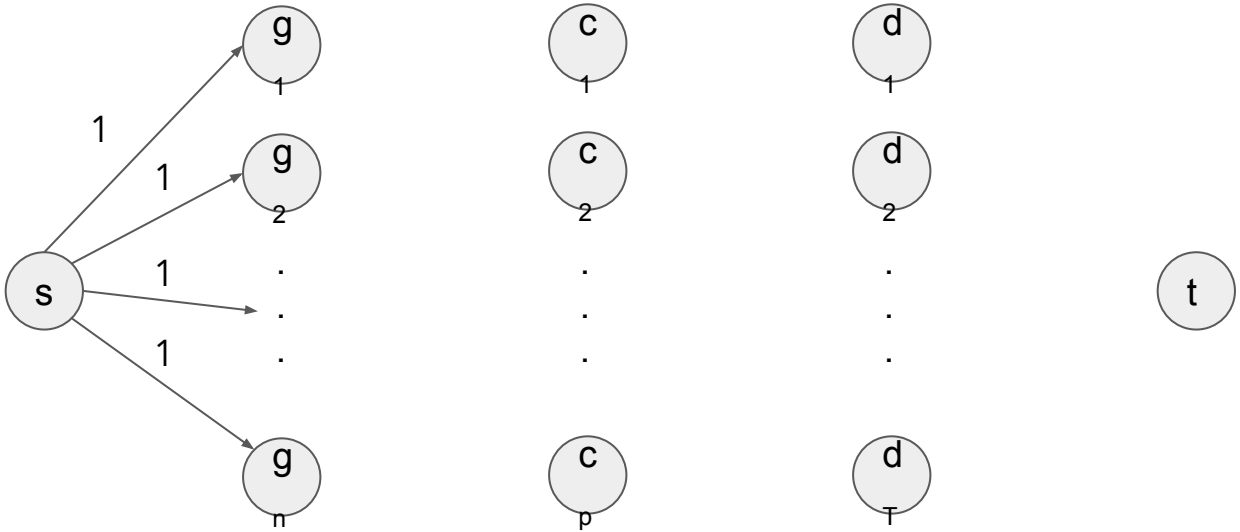
Graph consist of vertices of requirements, classes, and departments.

Add source and sink nodes.

Breadth Requirements (Spring 2016 MT2)

(b) Find an efficient algorithm to determine whether the requirements are satisfied, and (if all requirements are satisfied) a mapping of which class contributes to which requirement.

Reduce to max flow instance and solve in a way similar to bipartite matching.



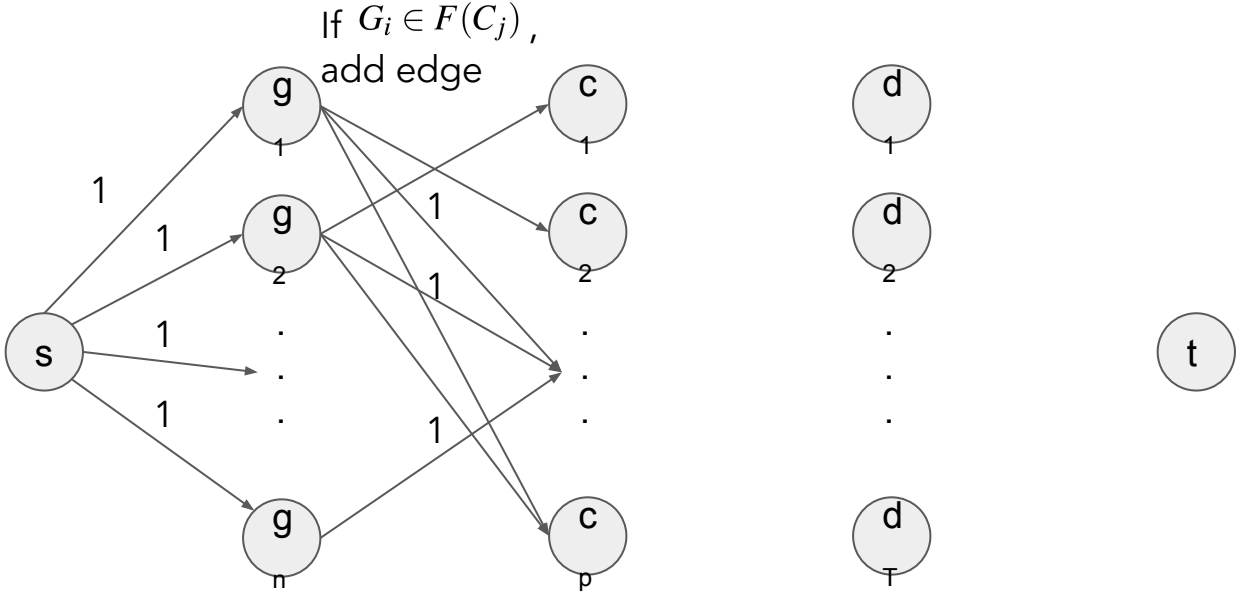
Need to satisfy each requirement.

Add edge (s, g_j) with edge weight 1.

Breadth Requirements (Spring 2016 MT2)

(b) Find an efficient algorithm to determine whether the requirements are satisfied, and (if all requirements are satisfied) a mapping of which class contributes to which requirement.

Reduce to max flow instance and solve in a way similar to bipartite matching.



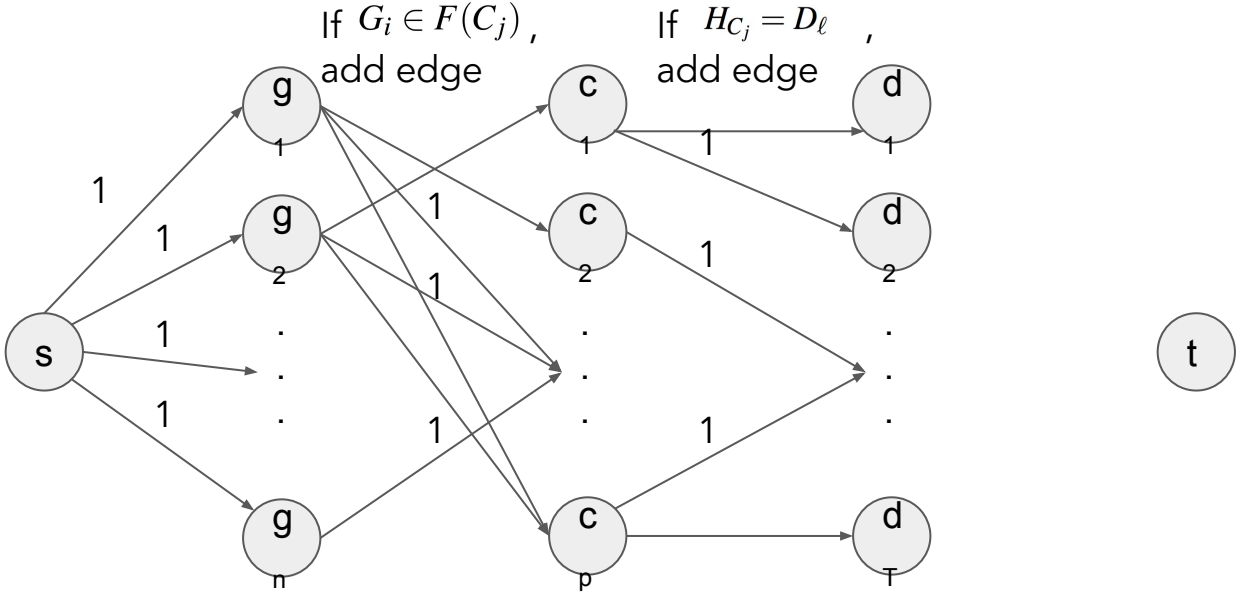
Add edge only if requirement can be satisfied with that class.

Each requirement is satisfied by a unique class.

Breadth Requirements (Spring 2016 MT2)

(b) Find an efficient algorithm to determine whether the requirements are satisfied, and (if all requirements are satisfied) a mapping of which class contributes to which requirement.

Reduce to max flow instance and solve in a way similar to bipartite matching.



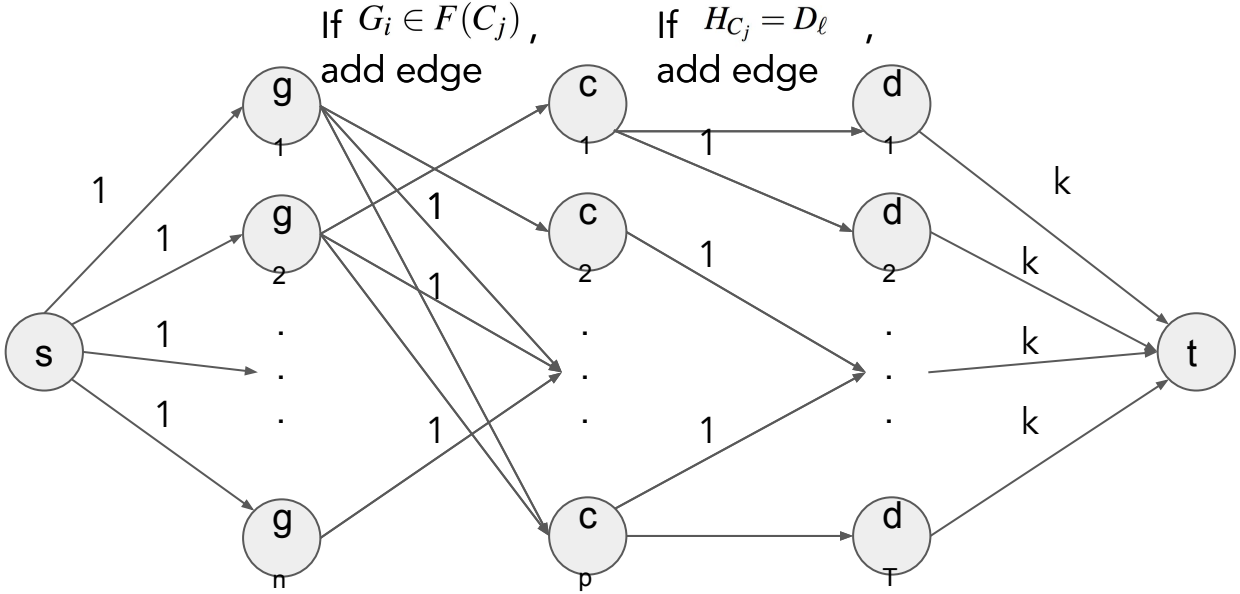
Add edge only if the class can satisfy the requirement.

Each class belongs to only one department. Thus edge capacity of 1.

Breadth Requirements (Spring 2016 MT2)

(b) Find an efficient algorithm to determine whether the requirements are satisfied, and (if all requirements are satisfied) a mapping of which class contributes to which requirement.

Reduce to max flow instance and solve in a way similar to bipartite matching.

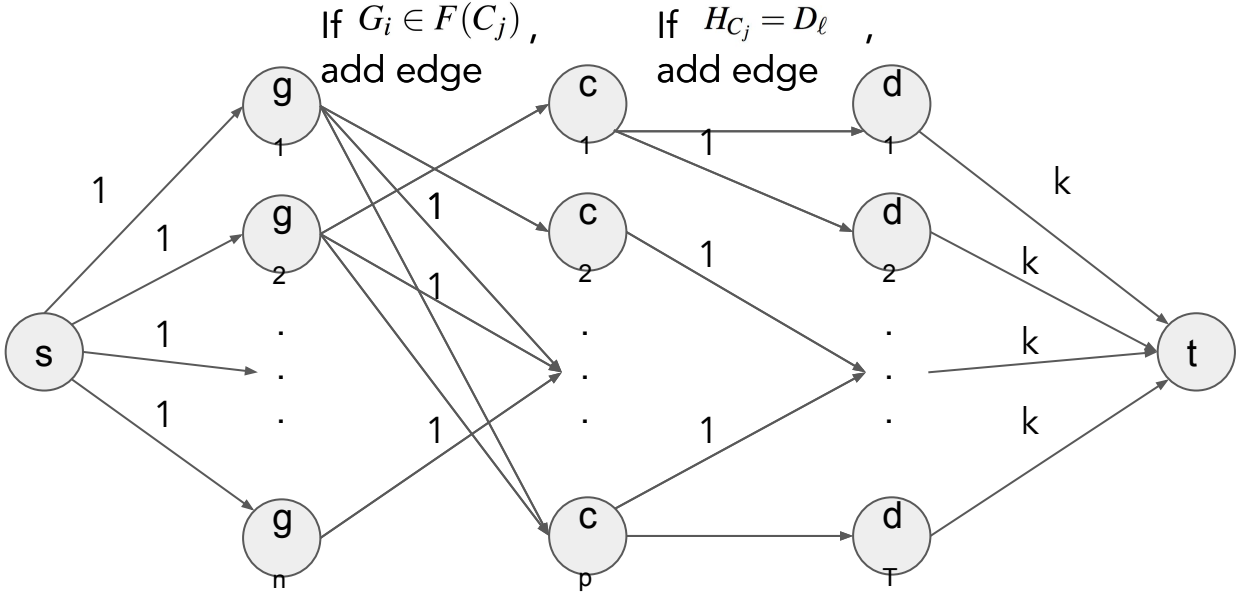


No more than k classes can be satisfy each requirement.

Breadth Requirements (Spring 2016 MT2)

(b) Find an efficient algorithm to determine whether the requirements are satisfied, and (if all requirements are satisfied) a mapping of which class contributes to which requirement.

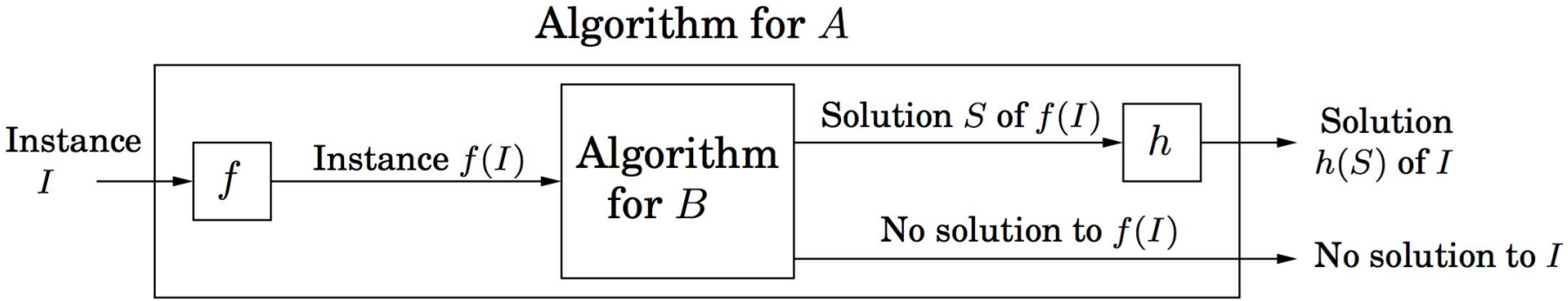
There is a satisfying assignment if size of max flow = n (number of requirements).



The flow can go in the other direction.

Complexity

Reduction



- Reduce A to B.
- B is at least as hard as A

Spring 2016 Final

(i) If B is **NP**-complete, then for any problem $A \in \mathbf{NP}$, there exists a polynomial-time reduction from A to B.

always True True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ always False

.

Spring 2016 Final

(i) If B is **NP**-complete, then for any problem $A \in$ **NP**, there exists a polynomial-time reduction from A to B.

always True True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ always False

.

Solution: Always true: this is the definition of **NP**-hard, and all **NP**-complete problems are **NP**-hard.

Spring 2016 Final

(ii) If B is in **NP**, then for any problem $A \in \mathbf{P}$, there exists a polynomial-time reduction from A to B.

always True True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ always False

Spring 2016 Final

(ii) If B is in **NP**, then for any problem $A \in \mathbf{P}$, there exists a polynomial-time reduction from A to B.

always True True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ always False

Solution: Always true: since we have polynomial time for our reduction, we have enough problem to simply solve any instance of A during the reduction.

Spring 2016 Final

Horn SAT is **NP**-complete.

always True True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ always False

Spring 2016 Final

Horn SAT is **NP**-complete.

always True True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ always False

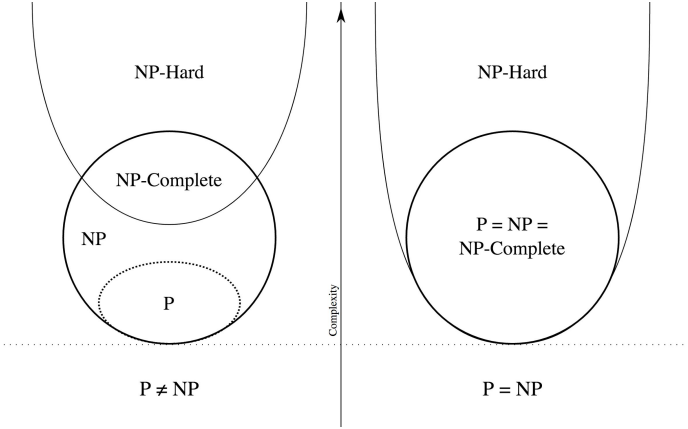
Solution: True iff $\mathbf{P} = \mathbf{NP}$: Horn SAT is in **P**.

Definitions

- P: Set of all problems that can be solved (and verified) in polynomial time.
- NP: Set of all problems that can be verified in polynomial time.
- NP-Hard: Set of all problems that are at least as hard as the hardest problem in NP
    - A problem A is in NP-Hard, when every problem B in NP can be reduced to A in polynomial time.
- NP-Complete: Set of problems in NP-Hard that can be verified in polynomial time.
    - Subset of NP-Hard problems.

Showing NP-Completeness

- Show problem A is in NP
  - Show that there is a polynomial time verifier.
- Show that problem A is NP-Hard
  - Reduce NP-Hard problem to A, or
  - Reduce NP-Complete problem to A



4D Matching

Recall the 3-DIMENSIONAL MATCHING problem, asking you to match n girls, n boys, and n pets given a list of compatible triples. We know that it is **NP**-complete. In the 4-DIMENSIONAL MATCHING problem you are given compatible *quadruples* of n boys, n girls, n pets, and n *homes*, and again you want to create n harmonious households accommodating them all. Prove 4-DIMENSIONAL MATCHING is **NP-complete**.

4D Matching

Recall the 3-DIMENSIONAL MATCHING problem, asking you to match n girls, n boys, and n pets given a list of compatible triples. We know that it is **NP**-complete. In the 4-DIMENSIONAL MATCHING problem you are given compatible *quadruples* of n boys, n girls, n pets, and n *homes*, and again you want to create n harmonious households accommodating them all. Prove 4-DIMENSIONAL MATCHING is **NP-complete**.

Show in NP:

A solution to 4D matching can be verified in polynomial time by verifying every boy, girl, pet, home is assigned to exactly one household and vice-versa.

4D Matching

Recall the 3-DIMENSIONAL MATCHING problem, asking you to match n girls, n boys, and n pets given a list of compatible triples. We know that it is **NP**-complete. In the 4-DIMENSIONAL MATCHING problem you are given compatible *quadruples* of n boys, n girls, n pets, and n *homes*, and again you want to create n harmonious households accommodating them all. Prove 4-DIMENSIONAL MATCHING is **NP-complete**.

Show in NP-Hard:

We will reduce the problem 3D matching to the problem 4D matching.

Given an instance of the problem 3D matching we construct an instance of the problem 4D matching as follows: Create n homes, which are compatible with every triple in our 3D Matching instance.

Any quadruple that 4D matching considers are not affected by any of the homes.

⅓ Independent Set

(c) In the $\frac{1}{3}$-INDEPENDENT SET problem you are given a graph (V, E) and you are asked to find an independent set of the graph of size exactly $\frac{|V|}{3}$. In other words, the target size g is not part of the input, but it is always $\frac{|V|}{3}$. Show that this special case of INDEPENDENT SET is **NP**-complete.

⅓ Independent Set

(c) In the $\frac{1}{3}$-INDEPENDENT SET problem you are given a graph (V, E) and you are asked to find an independent set of the graph of size exactly $\frac{|V|}{3}$. In other words, the target size g is not part of the input, but it is always $\frac{|V|}{3}$. Show that this special case of INDEPENDENT SET is **NP**-complete.

Show in NP:

A solution to 1/3 Independent Set can be verified in polynomial time by checking that there are no edges between the nodes of the independent set and checking its size.

⅓ Independent Set

(c) In the $\frac{1}{3}$-INDEPENDENT SET problem you are given a graph (V, E) and you are asked to find an independent set of the graph of size exactly $\frac{|V|}{3}$. In other words, the target size g is not part of the input, but it is always $\frac{|V|}{3}$. Show that this special case of INDEPENDENT SET is **NP**-complete.

Show in NP-Hard:

Reduce independent set (w/ target size g) to ⅓ independent set.

If g = |V|/3, no modifications.

⅓ Independent Set

(c) In the $\frac{1}{3}$-INDEPENDENT SET problem you are given a graph (V, E) and you are asked to find an independent set of the graph of size exactly $\frac{|V|}{3}$. In other words, the target size g is not part of the input, but it is always $\frac{|V|}{3}$. Show that this special case of INDEPENDENT SET is **NP**-complete.

Show in NP-Hard:

Reduce independent set (w/ target size g) to ⅓ independent set.

If g > |V|/3,

⅓ Independent Set

(c) In the $\frac{1}{3}$-INDEPENDENT SET problem you are given a graph (V, E) and you are asked to find an independent set of the graph of size exactly $\frac{|V|}{3}$. In other words, the target size g is not part of the input, but it is always $\frac{|V|}{3}$. Show that this special case of INDEPENDENT SET is **NP**-complete.

Show in NP-Hard:

Reduce independent set (w/ target size g) to ⅓ independent set.

If g > |V|/3, add *x* vertices

⅓ Independent Set

(c) In the $\frac{1}{3}$-INDEPENDENT SET problem you are given a graph (V, E) and you are asked to find an independent set of the graph of size exactly $\frac{|V|}{3}$. In other words, the target size g is not part of the input, but it is always $\frac{|V|}{3}$. Show that this special case of INDEPENDENT SET is **NP**-complete.

Show in NP-Hard:

Reduce independent set (w/ target size g) to ⅓ independent set.

If g > |V|/3, add *x* vertices such that g = (|V| + *x*) / 3

⅓ Independent Set

(c) In the $\frac{1}{3}$-INDEPENDENT SET problem you are given a graph (V, E) and you are asked to find an independent set of the graph of size exactly $\frac{|V|}{3}$. In other words, the target size g is not part of the input, but it is always $\frac{|V|}{3}$. Show that this special case of INDEPENDENT SET is **NP**-complete.

Show in NP-Hard:

Reduce independent set (w/ target size g) to ⅓ independent set.

If g > |V|/3, add *x* vertices such that g = (|V| + *x*) / 3

Add *x* = *3g - |V|*

⅓ Independent Set

(c) In the $\frac{1}{3}$-INDEPENDENT SET problem you are given a graph (V, E) and you are asked to find an independent set of the graph of size exactly $\frac{|V|}{3}$. In other words, the target size g is not part of the input, but it is always $\frac{|V|}{3}$. Show that this special case of INDEPENDENT SET is **NP**-complete.

Show in NP-Hard:

Reduce independent set (w/ target size g) to ⅓ independent set.

If g > |V|/3, add *x* vertices such that g = (|V| + *x*) / 3

Add *x* = *3g - |V|*

New vertices have edges to each other and original vertices

⅓ Independent Set

(c) In the $\frac{1}{3}$-INDEPENDENT SET problem you are given a graph (V, E) and you are asked to find an independent set of the graph of size exactly $\frac{|V|}{3}$. In other words, the target size g is not part of the input, but it is always $\frac{|V|}{3}$. Show that this special case of INDEPENDENT SET is **NP**-complete.

Show in NP-Hard:

Reduce independent set (w/ target size g) to ⅓ independent set.

If g < |V|/3,

⅓ Independent Set

(c) In the $\frac{1}{3}$-INDEPENDENT SET problem you are given a graph (V, E) and you are asked to find an independent set of the graph of size exactly $\frac{|V|}{3}$. In other words, the target size g is not part of the input, but it is always $\frac{|V|}{3}$. Show that this special case of INDEPENDENT SET is **NP**-complete.

Show in NP-Hard:

Reduce independent set (w/ target size g) to ⅓ independent set.

If g < |V|/3, add *x* vertices such that *g + x = (|V|+x) / 3*

⅓ Independent Set

(c) In the $\frac{1}{3}$-INDEPENDENT SET problem you are given a graph (V, E) and you are asked to find an independent set of the graph of size exactly $\frac{|V|}{3}$. In other words, the target size g is not part of the input, but it is always $\frac{|V|}{3}$. Show that this special case of INDEPENDENT SET is **NP**-complete.

Show in NP-Hard:

Reduce independent set (w/ target size g) to ⅓ independent set.

If g < |V|/3, add *x* vertices such that $g + x = (|V|+x) / 3$

$3g + 3x = |V| + x$

$x = (|V| - 3g) / 2$

$2x = |V| - 3g$

⅓ Independent Set

(c) In the $\frac{1}{3}$-INDEPENDENT SET problem you are given a graph (V, E) and you are asked to find an independent set of the graph of size exactly $\frac{|V|}{3}$. In other words, the target size g is not part of the input, but it is always $\frac{|V|}{3}$. Show that this special case of INDEPENDENT SET is **NP**-complete.

Show in NP-Hard:

Reduce independent set (w/ target size g) to ⅓ independent set.

If g < |V|/3, add *x* vertices such that $g + x = (|V|+x) / 3$

$3g + 3x = |V| + x$

$x = (|V| - 3g) / 2$

$2x = |V| - 3g$

New vertices are isolated

⅓ Independent Set

(c) In the $\frac{1}{3}$-INDEPENDENT SET problem you are given a graph (V, E) and you are asked to find an independent set of the graph of size exactly $\frac{|V|}{3}$. In other words, the target size g is not part of the input, but it is always $\frac{|V|}{3}$. Show that this special case of INDEPENDENT SET is **NP**-complete.

Show in NP-Hard:

After solving vai ⅓ independent set, return only vertices in original graph.

Proving NP-Hard (Fall 2015 Final)

DIRECTED RUDRATA CYCLE
Input: A directed graph $G = (V, E)$.
Goal: Find a directed cycle that visits every vertex in V exactly once.

Proving NP-Hard (Fall 2015 Final)

DIRECTED RUDRATA CYCLE
Input: A directed graph $G = (V, E)$.
Goal: Find a directed cycle that visits every vertex in V exactly once.

Solution: We reduce from RUDRATA CYCLE. To convert an undirected graph to a directed graph, we replace each undirected edge (u, v) with two directed edges, (u, v) and (v, u).

Proving NP-Hard (Fall 2015 Final)

CALIFORNIAN CYCLE

Input: A directed graph $G = (V, E)$ with each vertex colored *blue* or *gold*, i.e., $V = V_{blue} \cup V_{gold}$.

Goal: Find a *Californian cycle* which is a directed cycle through all vertices in G that alternates between blue and gold vertices. (*Hint: Directed Rudrata Cycle*)

Proving NP-Hard (Fall 2015 Final)

CALIFORNIAN CYCLE
Input: A directed graph $G = (V, E)$ with each vertex colored *blue* or *gold*, i.e., $V = V_{blue} \cup V_{gold}$.
Goal: Find a *Californian cycle* which is a directed cycle through all vertices in G that alternates between blue and gold vertices. (*Hint: Directed Rudrata Cycle*)

Solution: We reduce Directed Rudrata Cycle to Californian Cycle, thus proving the NP-hardness of Californian Cycle.

Given a directed graph $G = (V, E)$, we construct a new graph $G' = (V', E')$ as follows:

- For each $v \in V$, create a blue node v_b with an edge to a gold node v_g (in G').
- For each $(u, v) \in E$, add edge (u_g, v_b) to E'. Another way to view this is that for each node $v \in V$, we are redirecting all its incoming nodes to v_g, and all its outgoing nodes originate from v_b (in G').

Approximation Algorithms

General Strategy

Suppose we have an approximation algorithm A that takes in instance I and returns a solution with value $A(I)$. Then the approximation ratio is:

Maximization Problems	Minimization Problems
$\min_I \frac{A(I)}{\text{OPT}(I)} \leq 1$	$\max_I \frac{A(I)}{\text{OPT}(I)} \geq 1$

A general strategy for finding approximation algorithms (for minimization problems) is:

1. Solve another problem that gives a lower-bound on the optimal solution. (This problem is usually a *relaxed*, less constrained version of the original problem, but as a result its solution is not feasible in the original problem)

2. Find a way to turn this non-feasible solution into a feasible solution, inevitably increasing its value. The maximum amount this value can increase by is the approximation ratio.

Conceptual Questions

Which is better (for a minimization problem)? An approximation ratio of $\log(n)$ or 2?

True or False, does reductions preserve approximation algorithms? Suppose problem X reduces to problem Y, and $\mathscr{A}$ is an algorithm that gives an approximation ratio of δ for Y. Will $\mathscr{A}$ give the same approximation ratio (δ) for X after it is reduced to an instance of Y?

Conceptual Questions

Which is better (for a minimization problem)? An approximation ratio of $\log(n)$ or 2?

Solutions: 2 is better, since we want the approximation ratio to be as low as possible, and a $\log n$ approximation ratio grows with the instance size.

True or False, does reductions preserve approximation algorithms? Suppose problem X reduces to problem Y, and $\mathscr{A}$ is an algorithm that gives an approximation ratio of δ for Y. Will $\mathscr{A}$ give the same approximation ratio (δ) for X after it is reduced to an instance of Y?

Solutions: False, there are NP-complete problems such as (non-metric) TSP that are hard to approximate but there are also NP-complete problems such as vertex cover that have a constant approximation factor. For example, consider a complete bipartite graph (each vertex from one side is connected to all vertices on the other side). Recall that the vertices not selected in a minimum vertex cover is the maximum independent set. Choosing all but one vertex gives a factor $\Theta(1)$ approximation since the minimum vertex cover is $n/2$. However taking one vertex not selected in the approximation gives a factor $\Theta(1/n)$ approximation to maximum independent set since the optimal value is $n/2$.

Practice Questions

Approximation algorithms are useful for tackling computationally hard problems. Sometimes the task at hand is not computationally hard, but we still want approximation guarantees. Consider the following scenario:

Your friends have convinced you to go skiing with them for the first time in your life. You are not sure whether you will like skiing or not, and it might even take you several sessions to find out whether you like it. Skiing equipment is expensive. If you want to buy a set, it will cost you x dollars, but you can also rent a set for one session for y dollars. Should you invest in a set or just rent?

Friend A suggests you should simply buy a set, since renting is like throwing money away.

Friend B suggests you should always rent a set, since you never know when you might quit skiing.

Practice Questions (cont.)

If you were able to predict the future and know in advance that you would go skiing for n sessions, what would be the optimal strategy? Let the money you spend using this strategy be called OPT.

Practice Questions (cont.)

If you were able to predict the future and know in advance that you would go skiing for n sessions, what would be the optimal strategy? Let the money you spend using this strategy be called OPT.

Solutions:

It is never optimal to rent before buying, because we could have bought earlier. So the only two strategies are renting for all sessions, or buying the first time. If n is the number of sessions, the first strategy spends ny dollars and the second one spends x. So clearly $\text{OPT} = \min(x, ny)$.

Practice Questions (cont.)

How bad can listening to friend A be compared to OPT, in terms of an approximation factor?

Practice Questions (cont.)

How bad can listening to friend A be compared to OPT, in terms of an approximation factor?

Solutions:

By listening to A we will be just paying x. So the approximation ratio would be $\frac{x}{\min(x,ny)}$ which is worst when $n = 1$, at which point it becomes $\frac{x}{\min(x,y)}$. If x and y were not given, this could be arbitrarily large.

Practice Questions (cont.)

How bad can listening to friend B be compared to OPT?

Practice Questions (cont.)

How bad can listening to friend B be compared to OPT?

Solutions:

Listening to B would cost us ny. So the approximation ratio would be $\frac{ny}{\min(x,ny)}$. Clearly as $n \to \infty$ this fraction grows arbitrarily large (since the numerator grows but after some point the denominator stays constant). So the approximation ratio can be arbitrarily large.

Practice Questions (cont.)

Friend C suggests you follow this strategy: start by renting and keep track of how much you have spent on rentals. The first time that this sum is about to go over x, instead of renting, just buy a set. How bad can following C be compared to OPT?

Practice Questions (cont.)

Friend C suggests you follow this strategy: start by renting and keep track of how much you have spent on rentals. The first time that this sum is about to go over x, instead of renting, just buy a set. How bad can following C be compared to OPT?

Solutions:

First of all, the approximation ratio will never be worse than 2. If we never get to the buying point, then the money we spend is actually equal to OPT. If we get to the buying point then it must be that $ny \geq x$. So OPT $= x$. But using this strategy we do not pay more than $2x$, because we make sure we never spend more on renting than buying (so x is at most the money spent on rentals, and another x for the buy). Now if we follow this strategy and quit exactly right after we buy the item, we achieve the worst approximation factor, which for suitable values of x and y, can become arbitrarily close to 2.

Practice Questions

Give a factor $1/2$ approximation algorithm for the following problem: given a directed graph $G = (V, E)$, pick a maximum-size set of edges from E so that the resulting subgraph is acyclic.

Practice Questions

Give a factor $1/2$ approximation algorithm for the following problem: given a directed graph $G = (V, E)$, pick a maximum-size set of edges from E so that the resulting subgraph is acyclic.

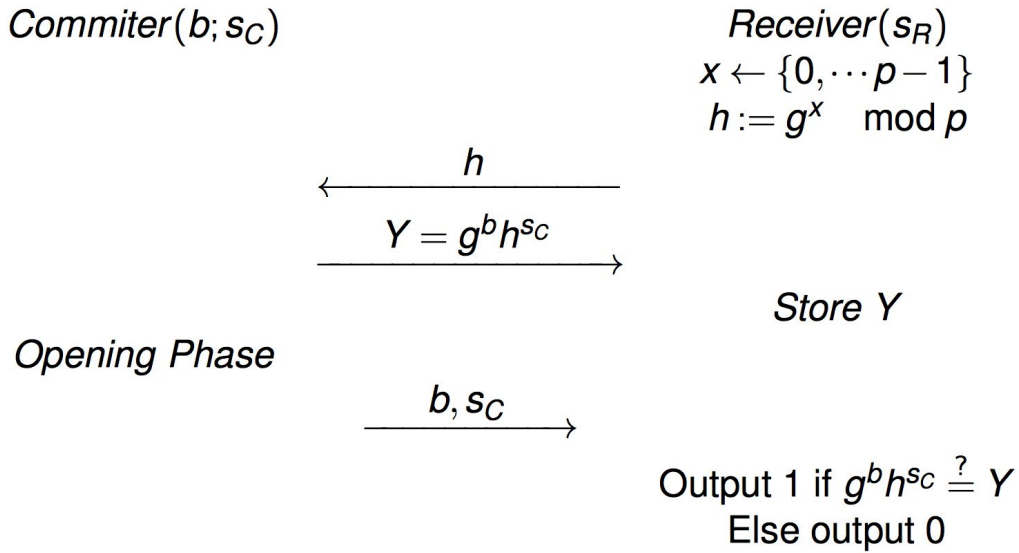
Solutions:

Arrange the vertices in a line, and group the edges into two sets, the ones going forward and the ones going backward. Each set alone forms a acyclic subgraph, and we pick the larger of the two. Hence we pick at least half of the edges, giving an approximation ratio of $1/2$.

Special Topics

Cryptography

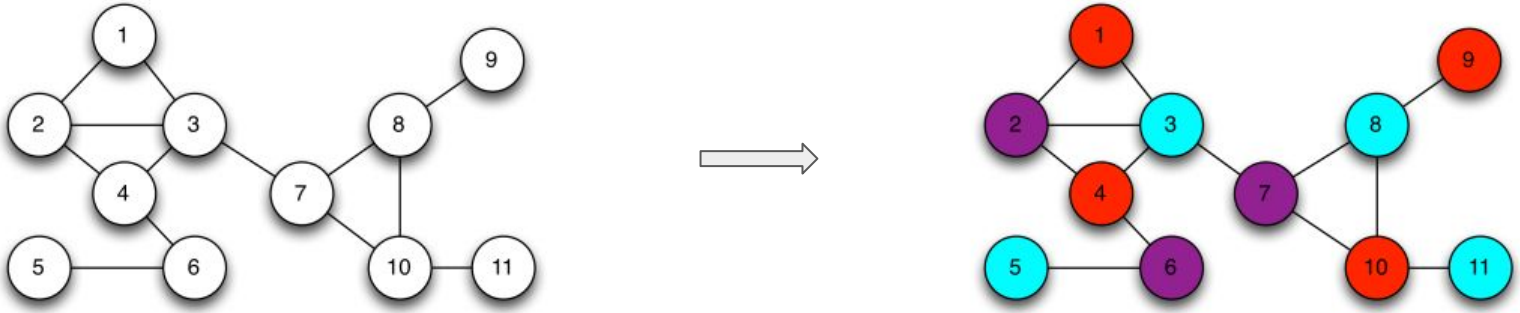
- Commitment Proctol
  - Allows one to commit to a chosen value while keeping it hidden to others, with the ability to reveal the committed value later



Graph Three-Coloring

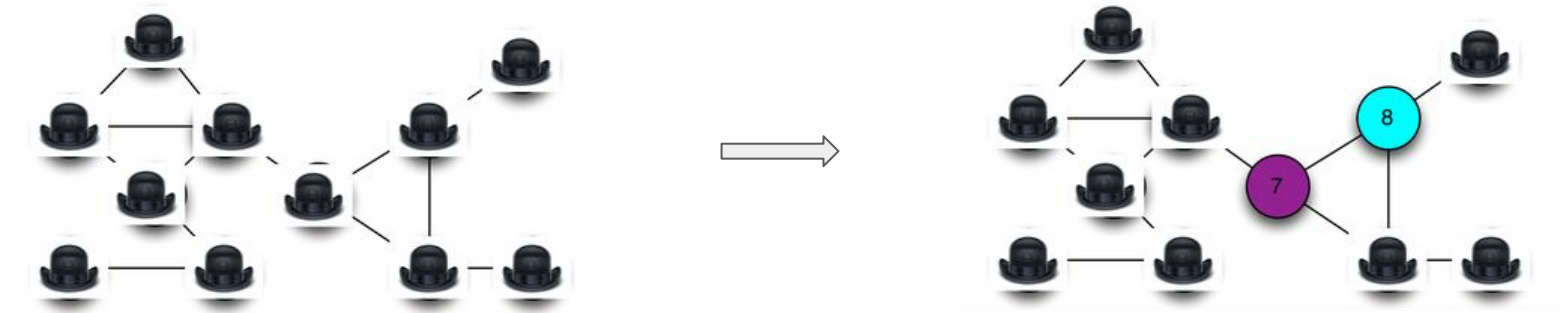
The decision problem of whether a given graph supports a solution with three colors is known to be in the complexity class **NP-Complete**.

Since the problem is hard, we can outsource it to ask someone who is smarter is solve it. The problem now is how I can verify that he/she has the wisdom to solve the problem.



Zero-Knowledge Proof System

- You draw the graph *G* and give it to your smart friend, say Raymond.
- Raymond only needs to show you he is capable of solving the problem. He does not want to give you the solutions easily. So he covers all his solutions with hats.
- But how can you make sure Raymond is actually capable of solving it?
- You decided to *randomly* ask Raymond to reveal the coloring for *an edge* to verify that he is not lying.



Zero-Knowledge Proof System

- Two outcomes from the experiment before:
  - If the two revealed vertices are the same color or aren't colored in at all, then I know for sure that Raymond is lying
  - If the two revealed vertices are different colors, then Raymond *might not* be lying to me.
- But what's the probability that Raymond can still cheat? ***(E-1)/E***
- I can ask Raymond to randomly show me the coloring of another edge. If again, both vertices are different colors, what's the probability that Raymond can cheat? ***((E-1)/E)^2***
- After a lot of iterations, I will be confident that Raymond is not lying.
- Raymond also protects his algorithm/solution!! (he randomly chooses a different set of colors each time)

Zero-Knowledge Proof System

- An interactive proof system involves a **prover** and a **verifier**
- Idea: the prover proves a statement to the verifier without revealing anything except the fact that the statement is true
- **Zero-knowledge proof of knowledge (ZKPK):** prover convinces verifier that he knows a secret without revealing the secret
- Properties:
  - Completeness: If both prover and verifier are honest, protocol succeeds with overwhelming probability
  - Soundness: No one who does not know the secret can convince the verifier with nonnegligible probability. The protocol should not enable prover to prove a false statement
  - Zero knowledge: The proof does not leak any information

Cryptography

Practice Problems:

Let $G = (V, E)$ be a graph that is three-colorable and let $c_1, c_2 : V \to \{R, G, B\}$ be two *valid* coloring functions. c is a valid coloring function if for every $(u, v) \in E$ we have that $c(u) \neq c(v)$. Let $< P, V >$ be the zero-knowledge proof system described in class. Answer True/False for the following questions.

- A malicious verifier learns whether the prover used c_1 or c_2.
- An honest prover in $< P, V >$ can prove to the verifier that the graph is three colorable even without being provided a valid c.
- An honest verifier in $< P, V >$ accepts a proof only if there exists a three coloring of the considered graph.

Sampling

Practice Problems:

Given a graph $G = (V, E)$ on n vertices, we would like to estimate the fraction of triples $(i, j, k) \in V$ that form a triangle in G, i.e., (i, j), (j, k) and $(k, i) \in E$. Design a randomized algorithm to estimate this fraction within an error of ± 0.01 with probability 0.99. What is the run-time of your algorithm, in terms of n?

Streaming

Distinct Element Algorithms:

1: Pick a hash function $h : W \to [0, 1]$
2: $currentmin \leftarrow 1$
3: **for** $i = 1$ to n **do**
4: **if** $h(w_i) < currentmin$ **then**
5: $currentmin \leftarrow h(w_i)$
6: **end if**
7: **end for**
8: Output $\frac{1}{currentmin}$

Streaming

Practice Problems:

Given a stream with only one distinct word repeated again and again, can the counting distinct algorithm return a value greater than 1000? If so, what is the probability of this event?

Multiplicative Weights Update Algorithm

Weighted Majority Algorithm:

1. Initialize $w_i = 1$, where w_i is the weight of the i-th expert.
2. Predict according to the weighted majority of experts.
3. Update weights by setting $w_i \leftarrow w_i/2$ for all experts who predicted incorrectly.

The number of mistakes M made by the experts algorithm with multiplicative factor of $(1-\epsilon)$ is bounded by,

$$M \leq 2(1+\epsilon)m + \frac{2\ln n}{\epsilon} \tag{2}$$

Weighted Majority Algorithm

Practice Problems:

1. In an execution of the weighted majority algorithm with n experts, after t days, the algorithm made ℓ mistakes. What is the largest possible value of the total weight of all experts?
2. In an execution of the weighted majority algorithm with n experts, suppose there is an expert (say E_1) who never makes a mistake. What is the maximum number of mistakes the algorithm will make?
3. "Follow the leader" is a strategy for the experts problem, where on each day, the aggregator always picks the best expert so far, i.e., the expert with the smallest number of mistakes up to day the day.

   Give an example to show that with 3 experts, this strategy could make three times as many mistakes as the best expert.

Multiplicative Weights Update Algorithm

Probabilistic Experts Algorithm:

We generalize the setting by allowing losses suffered by the experts to be real numbers in $[0, 1]$ instead of binary values. The loss suffered by the i-th expert in round t is denoted by $\ell_i^{(t)} \in [0, 1]$. The probabilistic experts algorithm is the following:

1. Initialize $w_i = 1$, where w_i is the weight of the i-th expert.
2. Predict according to an expert chosen with probability proportional to w_i, the probability of choosing the i-th expert is $\frac{w(i)}{W}$ where W is the total weight.
3. Update weights by setting $w_i \leftarrow w_i(1-\epsilon)^{\ell_i^{(t)}}$ for all experts.

If L is the expected loss of the probabilistic experts algorithm and L^* is the loss of the best expert then,

$$L \leq \frac{\ln n}{\epsilon} + (1+\epsilon)L^* \tag{4}$$

Multiplicative Weights Update Algorithm

Practice Problems:

4. We are running Multiplicative Weights with $n = 2$ experts for $T = 10,000$ days, and we choose $\varepsilon = 0.01$ Is this close to the optimum choice? (Hint: $\sqrt{\ln 2} \approx .8325$)

5. In all of the first 140 days, Expert 1 has cost 0 and Expert 2 has cost 1. In the next day, with what probability will you play Expert 1? (Hint: You can assume that $0.99^{70} = \frac{1}{2}$)