Lecture 22: Final Exam Review + Q&A

Tim LaRock

larock.t@northeastern.edu

bit.ly/cs3000syllabus

Business

EC1 due Sunday

EC2 due at 5PM

Final exam to be released at 6PM tonight and due Monday night at midnight Boston time

Today is our last meeting ☹

What we have covered in this course

Asymptotic Analysis

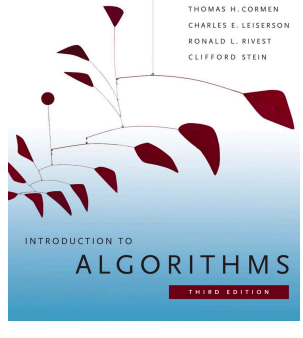
Divide and Conquer Algorithms
- Recursion/Backtracking
- Dynamic Programming

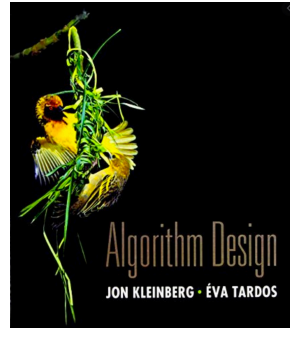
Graph Algorithms

Network Flow
- Reductions between algorithms

Greedy Algorithms
- Huffman Codes and entropy

Algorithms





Jeff Erickson

Pseudocode

- In general: there should be enough detail that someone could implement the algorithm based on your pseudocode, but not so much detail that it is difficult to follow
  - This is subjective and problem dependent! Writing good pseudocode is an art more than a science!

- Pseudocode should never be written to rely on the features of any specific programming language
  - For example: You should **not** assume that there is a len() function (as in python). Instead, as we have throughout this course, assume that the length of an array is given, e.g. $A[1..n]$.
  - If you are using a list, not an array, then you need to intentionally decide whether there is an $O(1)$ len() function or if you need to traverse the list to find its length.
    - Usually this is safe to do, but you need to recognize that it is a choice that depends on the problem you are solving!

- You can use English descriptions where they make things clearer!
  - "Let $x_1, x_2, \ldots, x_n$ be integers representing…"
  - "for each neighbor v of u"

Pseudocode

- When in doubt on the final, a little bit too much detail is better than an underspecified solution!

- Absolutely must be typeset in LaTeX on the final, no exceptions.

\begin{algorithm}

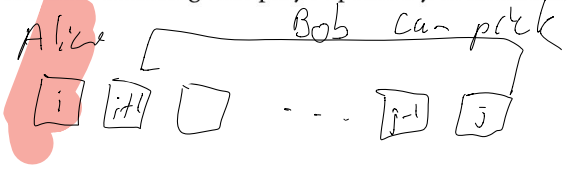
Dynamic Programming: Strategery (HW2)

Alice and Bob play the following game. There is a row of n tiles with values $a_1, \ldots, a_n$ written on them. Starting with Alice, Alice and Bob take turns removing either the first or last tile in the row and placing it in their pile until there are no tiles remaining. For example, if Alice takes tile 1, Bob can take either tile 2 or tile n on the next turn. At the end of the game, each player receives a number of points equal to the sum of the values of their tiles minus that of the other player's tiles. Specifically, if Alice takes tiles $A \subseteq \{1, \ldots, n\}$ and Bob takes tiles $B = \{1, \ldots, n\} \setminus A$, then their scores are

$$\sum_{i \in A} a_i - \sum_{i \in B} a_i \quad \text{and} \quad \sum_{i \in B} a_i - \sum_{i \in A} a_i,$$

respectively. For example, if $n = 3$ and the tiles have numbers $10, 2, 8$ then taking the first tile guarantees Alice a score of at least $10 + 2 - 8 = 4$, whereas taking the last tile would only guarantee Alice a score of at least $8 + 2 - 10 = 0$.

In this question, you will design an algorithm to determine the maximum score that Alice can guarantee for herself, assuming Bob plays optimally to maximize his score.[2]

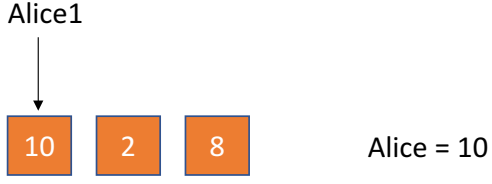


Question: Does greedy find the max?

Greedy: Always choose the tile with maximum value. Assume Alice and Bob are both playing with this strategy.

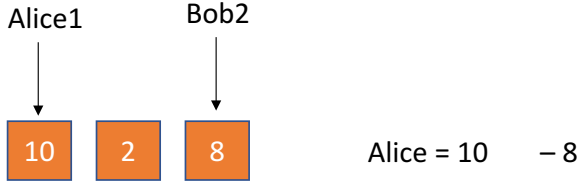
Question: Does greedy find the max?

Greedy: Always choose the tile with maximum value. Assume Alice and Bob are both playing with this strategy.



Question: Does greedy find the max?

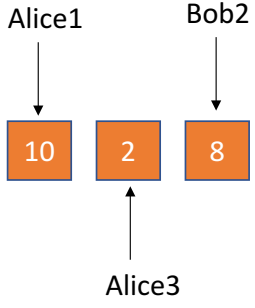
Greedy: Always choose the tile with maximum value. Assume Alice and Bob are both playing with this strategy.



Question: Does greedy find the max?

Greedy: Always choose the tile with maximum value. Assume Alice and Bob are both playing with this strategy.

Seems to work for this example!



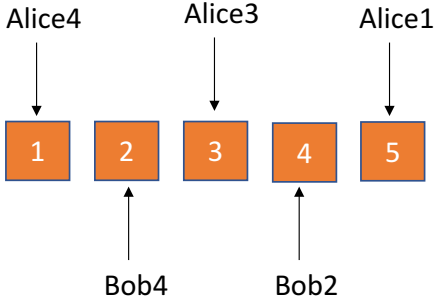
Alice = 10 + 2 – 8 = 4

Question: Does greedy find the max?

Greedy: Always choose the tile with maximum value. Assume Alice and Bob are both playing with this strategy.

Question: Does greedy find the max?

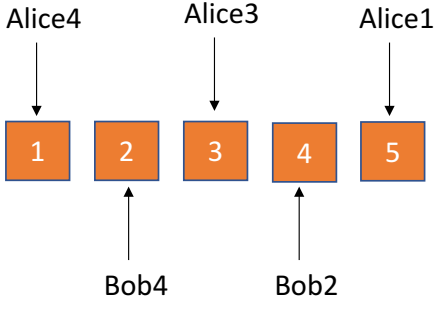
Greedy: Always choose the tile with maximum value. Assume Alice and Bob are both playing with this strategy.



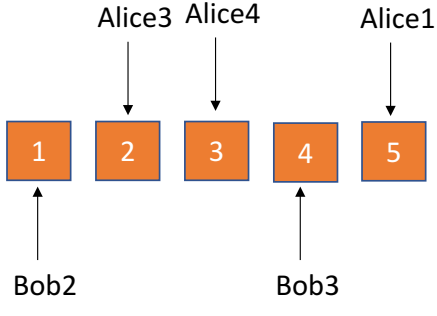
Alice = 5 + 3 + 1 – (2+4) = 3

Question: Does greedy find the max?

Greedy: Always choose the tile with maximum value. Assume Alice and Bob are both playing with this strategy.



Alice = 5 + 3 + 1 – (2+4) = 3



Alice = 5+2+3 – (1+4) = 5

This is **not** the maximum score Alice could achieve!

Solution

We need to write a recurrence that represents the maximum score Alice could achieve, assuming both players are playing optimally.

We will compute $Opt(i, j)$, which represents the optimal solution when the leftmost tile on the table is i and the rightmost is j.

We need to account for (1) the fact that her total score is determined by subtracting Bob's tiles, as well as (2) the base case when there is only 1 tile remaining.

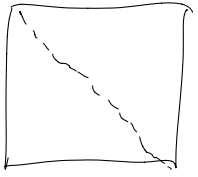
For an arbitrary i and j, Alice's total score is represented by picking up either tile i or tile j, then subtracting the optimal score on the new set of tiles, which represents Bob playing optimally (then Alice playing optimally, then bob playing optimally, and so on until the base case).

$$Opt(i,j) = \begin{cases} a_i & \text{If } i = j \\ \max\{a_i - Opt(i+1, j),\ a_j - Opt(i, j-1)\} & \end{cases}$$

Opt(1, n)

Solution

$$Opt(i,j) = \begin{cases} a_i & \text{If } i = j \\ \max\{a_i - Opt(i+1,j),\ a_j - Opt(i,j-1)\} & \end{cases}$$



The "bottom-up" algorithm for the problem is the following:

Algorithm 3: Strategery $(a_1, \ldots, a_n)$

```
M ← matrix n × n of zeros
For k = 0, ..., n − 1
    For i = 1, ..., n − k
        If k = 0 :
            M[i, i] ← a_i
        Else
            j = i + k
            M[i, j] ← max{a_i − M[i + 1, j], a_j − M[i, j − 1]}
Return M[1, n]
```

Clarification: Top down Vs. Bottom up

In algorithms:

- Top down is memoization
  - We discussed for Fibonacci numbers
  - In this case, your solution still ”looks recursive”, but every call either fills an entry in the table or uses the table to get a next solution
- Bottom up is also called “tabulation”
  - It usually corresponds to starting at the base case and iteratively building every subsequent solution from there.

Observation: A tradeoff between the two is that tabulation is often conceptually simpler to implement, but memoization is easier to think about if you already have a full recursive solution.

Key point: The result on a given input should be equivalent for our purposes!

Memo(r)ization

```
Fib(n):
  If n = 0:
    return 0
  ElseIf n = 1:
    return 1
  Else:
    return Fib(n − 1) + Fib(n − 2)
```

```
MemFib(n):
  If n = 0:
    return 0
  ElseIf n = 1:
    return 1
  Else:
    If F[n] is undefined:
      F[n] = MemFib(n − 1) + MemFib(n − 2)
    return F[n]
```

- $Fib(n)$ is very slow because we are recomputing the same values over and over again!
- What if instead we save each value we compute so that we can access it in constant time?
- Keep a global table $F[i]$ that stores results and use stored results where possible
- How is the table filled? And what implication does this have for the runtime?

Tabulation

```
MemFib(n):
  If n = 0:
    return 0
  ElseIf n = 1:
    return 1
  Else:
    If F[n] is undefined:
      F[n] = MemFib(n − 1) + MemFib(n − 2)
    return F[n]
```

```
IterFib(n):
  F[0] ← 0
  F[1] ← 1
  for i from 2..n
      F[i] ← F[i − 1] + F[i − 2]
  return F[n]
```

- The execution order and runtime of $MemFib(n)$ implies a simpler way to compute Fibonacci numbers
- What if we just like….filled F explicitly?
  - **This is tabulation!**
- Now the execution is clearly $O(n)$!

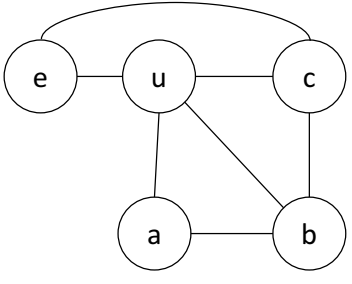
Betweenness Centrality

Betweenness centrality is used as a proxy for the importance of a node in facilitating connections between other nodes.

For node u, betweenness is measured as the ratio of shortest paths between all other pairs of nodes (s, t) that u lies on. Formally:

$$B(u) = \sum_{s \neq t \neq u} \frac{\sigma_{st}(u)}{\sigma_{st}}$$

Where σ_{st} is the number of shortest paths between nodes s and t and $\sigma_{st}(u)$ is the number of those shortest paths that include u.



Betweenness Centrality

$b(u) = 1 + \frac{1}{2} + \frac{1}{2} = 1 + 1 = 2$

Steps:

1. Write down all the pairs of nodes that are not u
2. Optional: Eliminate any pairs that are directly connected, since they cannot possibly have paths containing u (numerator = 0)
3. Compute all shortest paths between the remaining pairs of nodes
4. Use the formula to compute centrality

$$B(u) = \sum_{s \neq t \neq u} \frac{\sigma_{st}(u)}{\sigma_{st}}$$

Where σ_{st} is the number of shortest paths between nodes s and t and $\sigma_{st}(u)$ is the number of those shortest paths that include u.

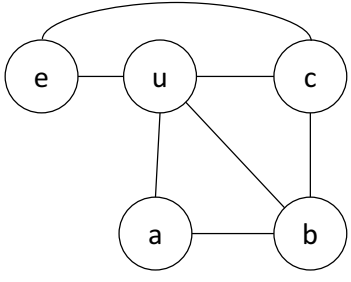
(e, c) (a, b)

(e, a)

(e, b)

(c, b)

(c, a)



(e, a): e, u, a $\frac{1}{1}$

(e, b): e, u, b $\frac{1}{2}$

e, c, b

(c, a): c, u, a $\frac{1}{2}$

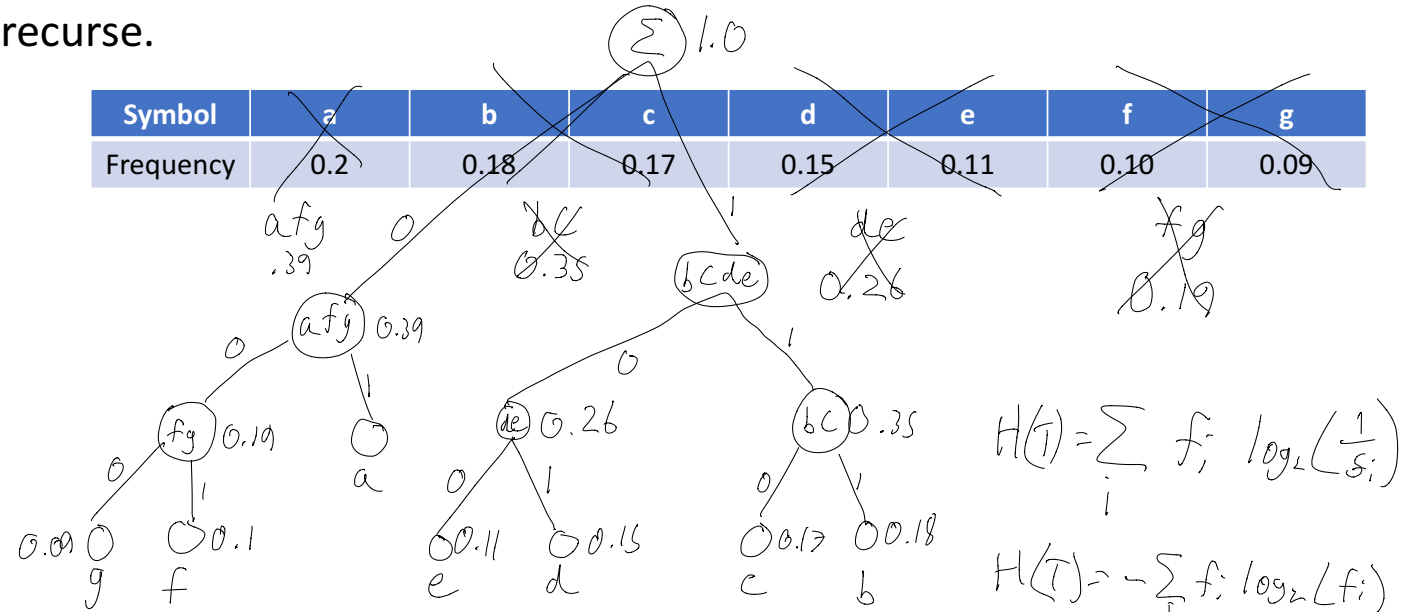
c, b, a

Huffman Codes

a b c
0 1 111 110

Huffman's Algorithm: Choose the two least frequent symbols and recurse.

Symbol	a	b	c	d	e	f	g
Frequency	0.2	0.18	0.17	0.15	0.11	0.10	0.09



$$H(T) = \sum_i f_i \log_2\left(\frac{1}{f_i}\right)$$

$$H(T) = -\sum_i f_i \log_2(f_i)$$

Reduction

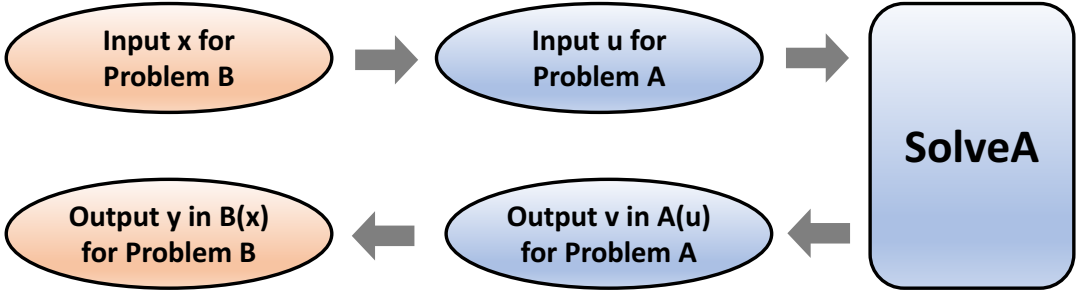
- **Definition:** a **reduction** is an efficient algorithm that solves problem A using calls to function that solves problem B.

Mechanics of Reductions

- What exactly is a **problem**?
  - A set of legal inputs $\boldsymbol{X}$
    - Ex: An array of numbers $A[1..n]$
  - A set $\boldsymbol{A(x)}$ of legal outputs for each $\boldsymbol{x \in X}$
    - Ex: The array A in sorted order

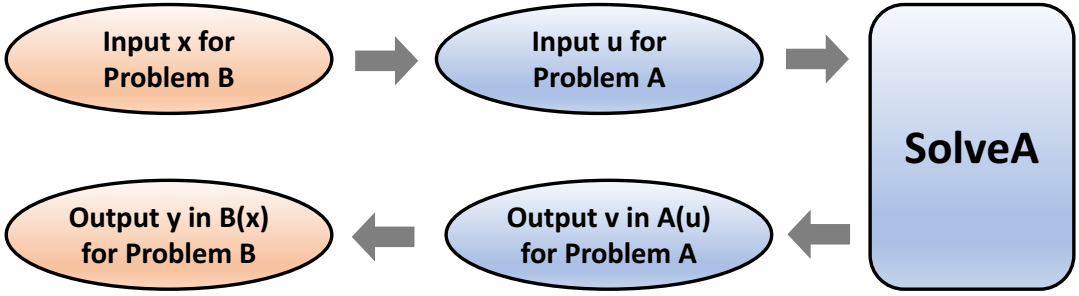
- **Example:** integer maximum flow
  - Input: $G = (V, E, s, t, \{c_e\})$ where c_e is an integer for every $e \in E$
  - Output: A maximum flow $\{f(e)\}$ for G where $f(e)$ is an integer for every $e \in E$ such that $0 \leq f(e) \leq c_e$

Mechanics of Reductions



In the simplest case, we just call SolveA a single time. In fact we may use SolveA as a subroutine to a more complex reduction.

When is a Reduction Correct?

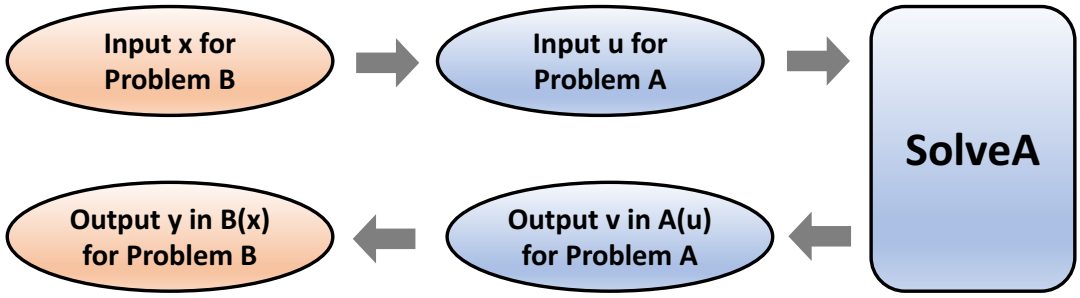


Assume that for valid input u, SolveA returns a valid output v in $A(u)$.

Then for every valid input x, if v is a valid output in $A(u)$, then y is a valid output in $B(x)$.

Example: Minimum Cut

A = MaxFlow
B = MinCut



Input x for B: $G = (V, E, s, t, \{c_e\})$

Input u for A: $G = (V, E, s, t, \{c_e\})$

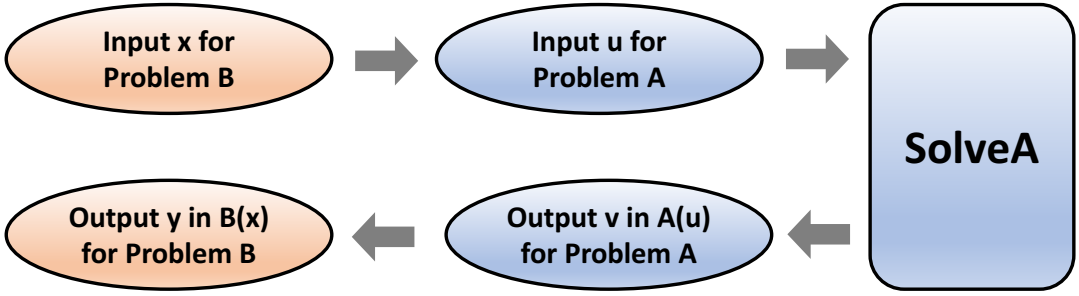
Output $v \in A(u)$: $G = (V, E, s, t, \{c_e\})$

Output $y \in B(x)$: $G = (V, E, s, t, \{c_e\})$

1. Take f, compute the residual graph G_f
2. Find the nodes reachable from s in G_f
3. Output these nodes

Example: Median

A = MergeSort
B = Median



Input x for B: Array of length n, $A[1..n]$

Input u for A: Same array

Output $v \in A(u)$: Sorted version of $A[1..n]$

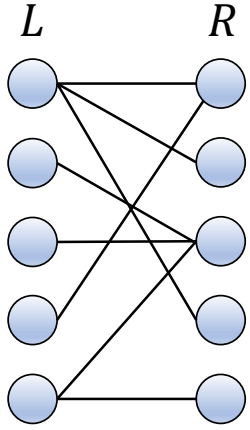
Output y $\in B(x)$: $A[\ \lfloor \frac{n}{2} \rfloor\ ]$

Bipartite Matching

- **Input:** bipartite graph $G = (V, E)$ with $V = L \cup R$

Models any problem where one type of object is assigned to another type:

- doctors to hospitals
- jobs to processors
- advertisements to websites

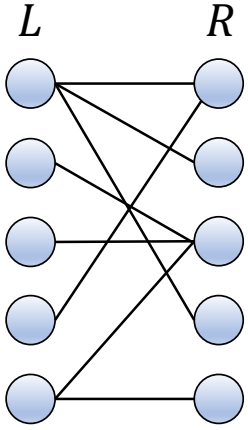


Bipartite Matching

- **Input:** bipartite graph $G = (V, E)$ with $V = L \cup R$
- **Output:** a maximum cardinality matching
  - A **matching** $M \subseteq E$ is a set of edges such that every node v is an endpoint of at most one edge in M

Models any problem where one type of object is assigned to another type:

- doctors to hospitals
- jobs to processors
- advertisements to websites

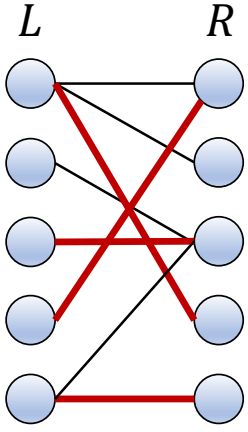


Bipartite Matching

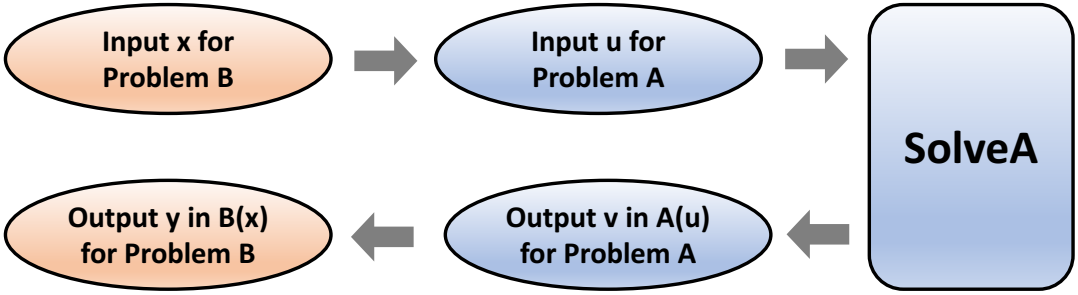
- **Input:** bipartite graph $G = (V, E)$ with $V = L \cup R$
- **Output:** a maximum cardinality matching
  - A **matching** $M \subseteq E$ is a set of edges such that every node v is an endpoint of at most one edge in M
  - Cardinality = $|M|$

Models any problem where one type of object is assigned to another type:

- doctors to hospitals
- jobs to processors
- advertisements to websites

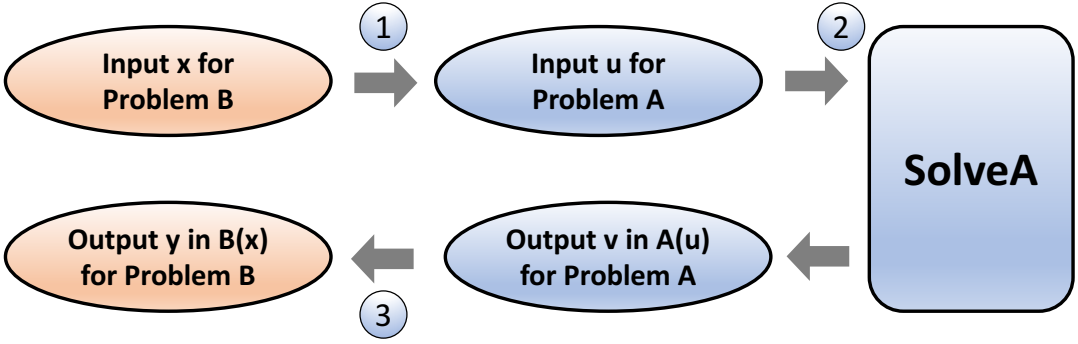


Bipartite Matching



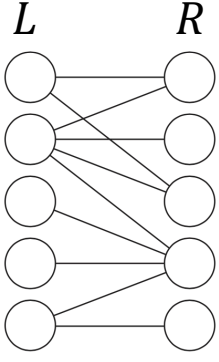
- There is a reduction that uses integer maximum s-t flow to solve maximum bipartite matching.
  - Problem B: maximum bipartite matching (MBM)
  - Problem A: integer maximum s-t flow

Bipartite Matching

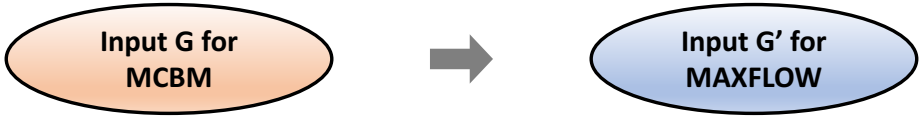


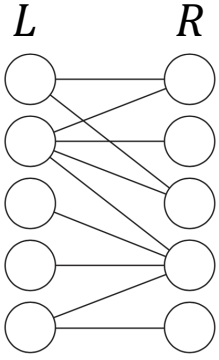
- There is a reduction that uses integer maximum s-t flow to solve maximum bipartite matching.
  - Problem B: maximum bipartite matching (MBM)
  - Problem A: integer maximum s-t flow

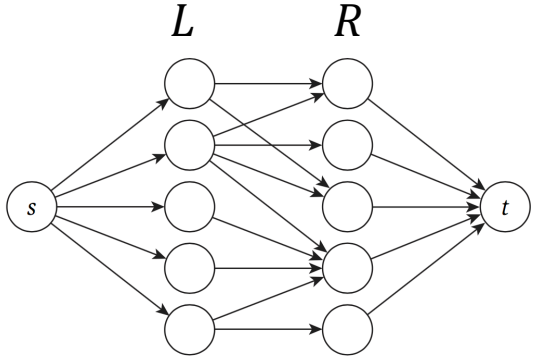
Step 1: Transform the Input



Step 1: Transform the Input

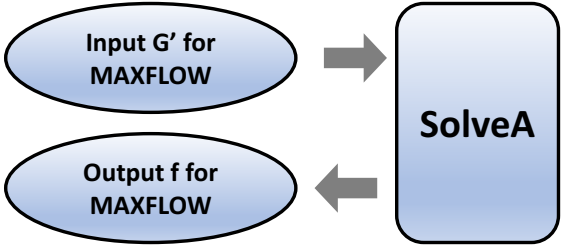


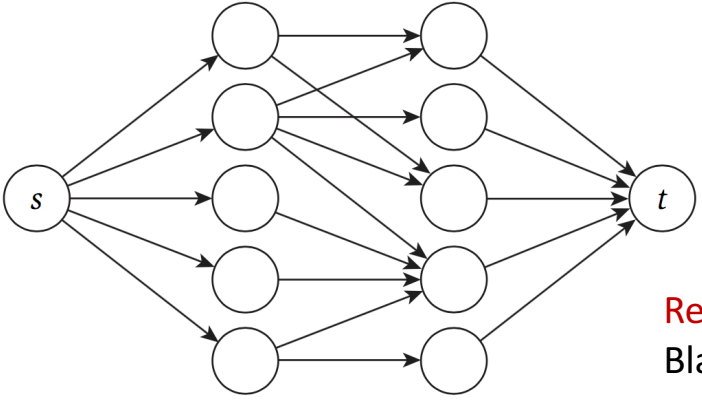




Set all edge capacities to $c(e) = 1$

Step 2: Receive the Output

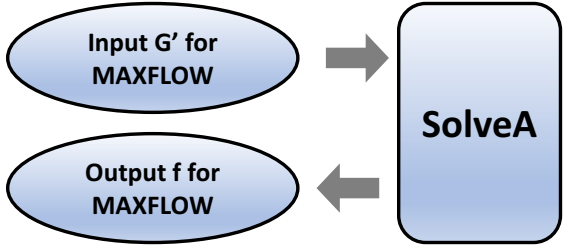


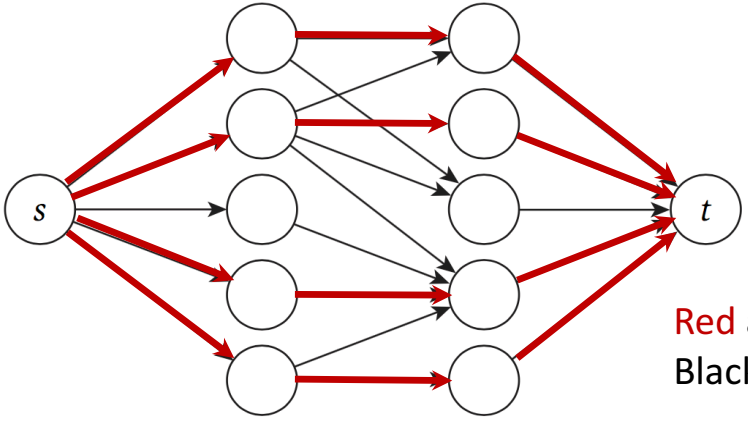


Red arrow means f(e)=1
Black means f(e) = 0

Step 2: Receive the Output

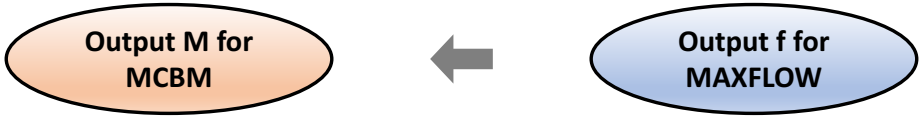
The matching will be all of the edges from L to R with nonzero flow!

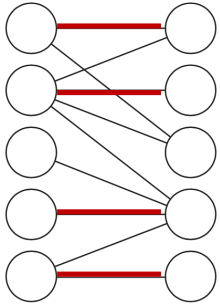


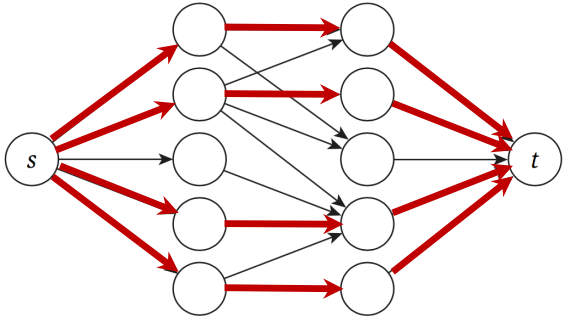


Red arrow means f(e)=1
Black means f(e) = 0

Step 3: Transform the Output







Reduction Recap

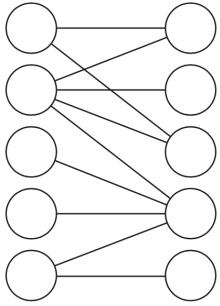
- Step 1: Transform the Input
  - Given G = (L,R,E), produce G' = (V,E,{c(e)},s,t) by...
    - ... orienting edges e from L to R
    - ... adding a node s with edges from s to every node in L
    - ... adding a node t with edges from every not in R to t
    - ... seting all capacities to 1
- Step 2: Receive the Output
  - Find an integer maximum s-t flow f in G'
- Step 3: Transform the Output
  - Given an integer s-t flow f(e)…
    - Let M be the set of edges e going from L to R that have f(e)=1

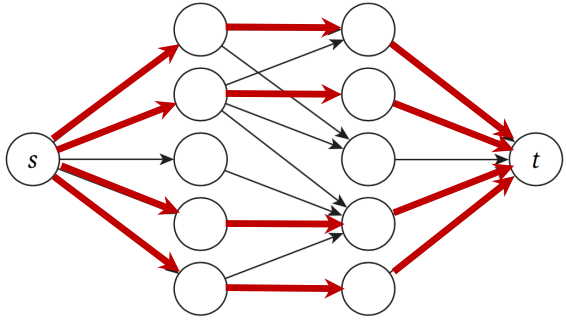
Correctness

- Need to show:
  - (1) This algorithm returns a matching
  - (2) This matching is a maximum cardinality matching

Correctness

- This algorithm returns a matching



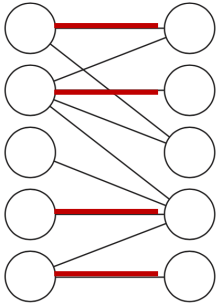


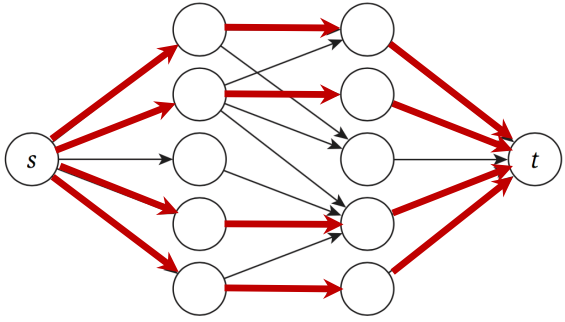
Since the capacity on every edge is 1, by conservation of flow we have:

- For any node in L, exactly one outgoing edge can have flow
- For any node in R, exactly one incoming edge can have flow

Correctness

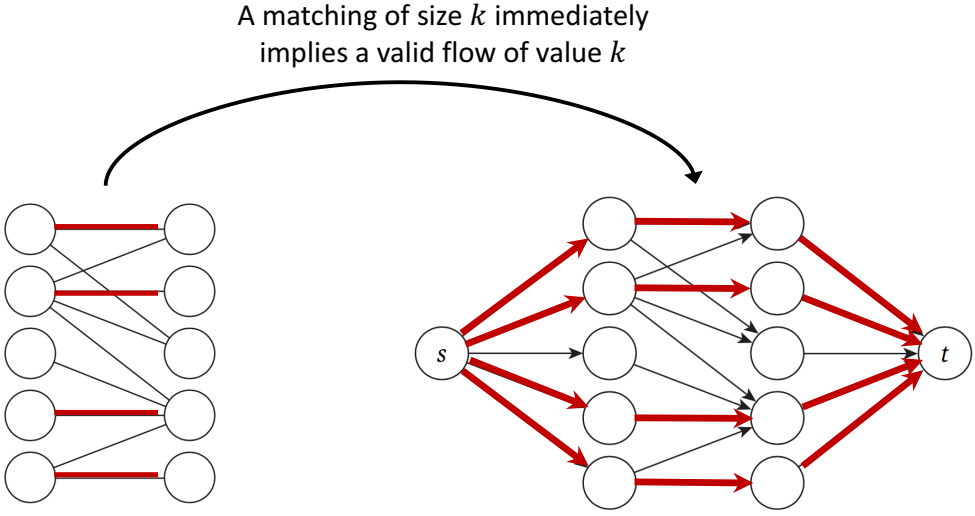
- **Claim:** G has a matching of cardinality at least k if and only if G' has an s-t flow of value at least k





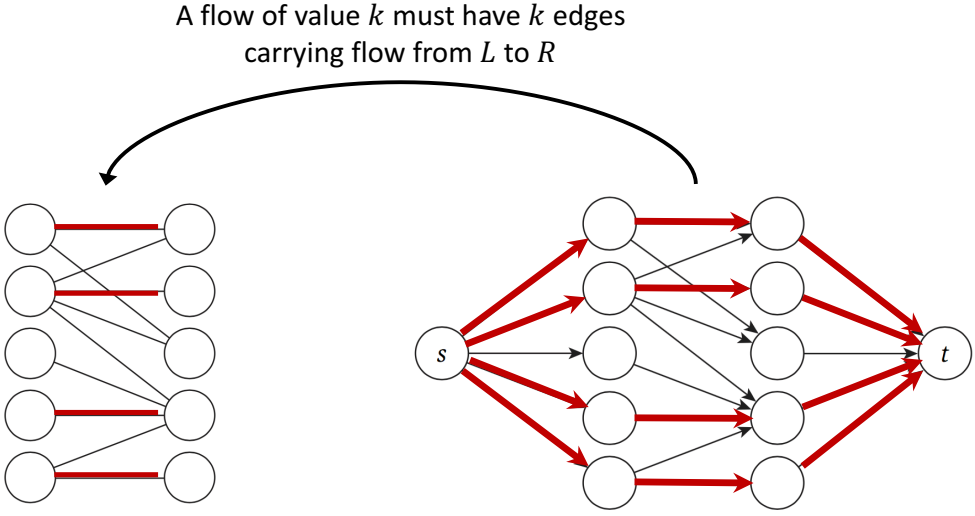
Correctness

- **Claim:** G has a matching of cardinality at least k if and only if G' has an s-t flow of value at least k



Correctness

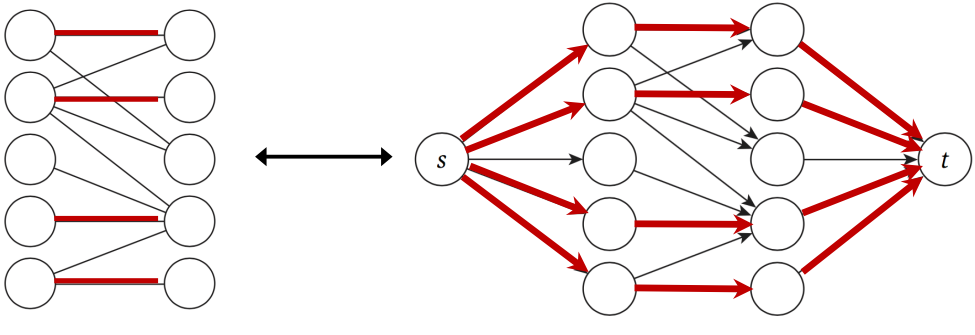
- **Claim:** G has a matching of cardinality at least k if and only if G' has an s-t flow of value at least k



Correctness

- **Claim:** G has a matching of cardinality at least k if and only if G' has an s-t flow of value at least k

A matching of size k immediately implies a valid flow of value k ⟷ A flow of value k must have k edges carrying flow from L to R



When k is the maximum cardinality matching, there must be a flow, and vice versa!

Summary

Solving maximum integer s-t flow in a graph with n+2 nodes and m+n edges and c(e) = 1 in time T

Solving maximum bipartite matching in a graph with n nodes and m edges in time T + O(m+n)

- Can solve max bipartite matching in time O(nm) using Ford-Fulkerson
  - Improvement for maximum flow gives improvement for maximum bipartite matching

Greedy Algorithm Proofs

Inductive Exchange Arguments

- Assume you have some other valid solution, then show that you can iteratively transform that solution in to the greedy solution without losing optimality, e.g. that the greedy solution is at least as good as the other solution
  - Kind of like a proof by contradiction, except you are not trying to prove that the greedy solution is the only optimal solution (because it might not be!), but rather that any other optimal solution is only as good as the Greedy one (they have the same value, or greedy is better)

Greedy Stays Ahead

- Show that every choice that greedy makes is at least as good as the choice that any optimal solution would make
- We did not do a very clear example of this in class - I will not ask about it on the final

Any more questions?

That's a wrap!

- Best of luck on the final, you will do great!
  - As usual, ask all questions privately on Piazza
  - Make sure you are sleeping and eating – your health is always more important than my final exam!
- PLEASE: Fill out the course evaluation if you have not already, whether you have good or bad things to say, or both!
  - The deadline may be very soon, please please check and fill it out beforehand!
  - I want to hear all of your honest, anonymous feedback, but I don't get to see the results if not enough people fill out the evaluation!
  - Your feedback may help improve courses taught online in the very near future
- You have my email address, I would be glad to hear from you about anything!