CS 170 Efficient Algorithms and Intractable Problems
Spring 2016 Alessandro Chiesa and Umesh Vazirani

# Midterm 2 Solutions

Name: Alexander Hamilton

SID: 10101804

GSI and section time:

*Write down the names of the students on your left and right as they appear on their SID.*

Name of student on your left:

Name of student on your right:

Name of student behind you:

Name of student in front of you:

Note: For each question part, 20% of the points will be given to any blank answer, or to any *clearly* crossed-out answer.
Note: If you finish in the last 15 minutes, please remain seated and not leave early, to not distract your fellow classmates.

Instructions: You may consult one handwritten, single-sided sheet of notes. You may not consult other notes, textbooks, etc. Cell phones and other electronic devices are not permitted.

There are 8 questions. The last page is page 12.

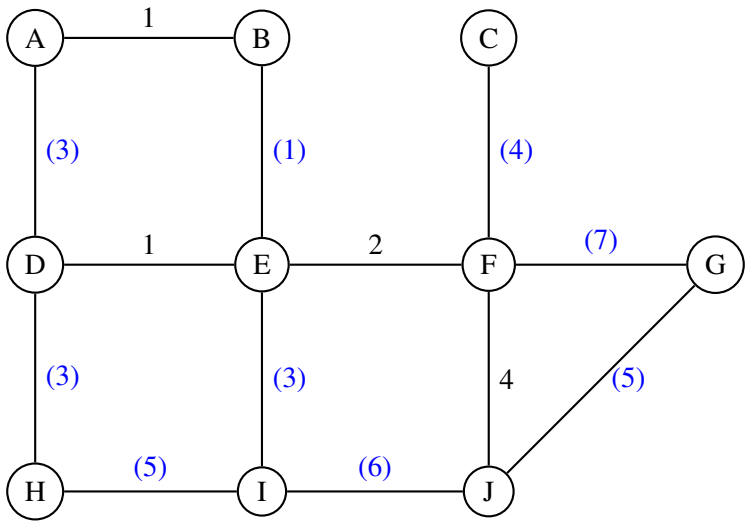
Answer all questions. On questions asking for an algorithm, make sure to respond in the format we request. Be precise and concise. Write in the space provided. Good luck!

Do not turn this page until an instructor tells you to do so.

**1. (45 pts.) Short Answer**

(a) **(10 pts.) *Maximum* Spanning Tree**

(i) Find a **maximum** spanning tree of the graph below. You can indicate your choice of spanning tree by circling the weights of the edges included in it.




You can find the maximum spanning tree by running a variant of either Prim's or Kruskal's, but prioritizing taking the heaviest edges first.

(ii) How many different **maximum** spanning trees does the above graph have?

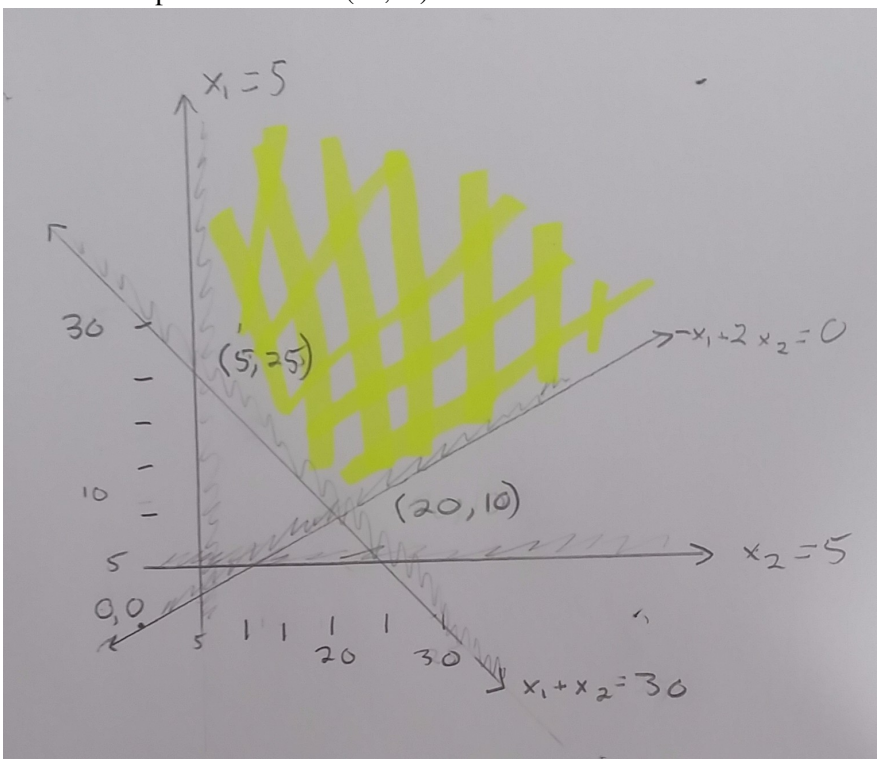
**Solution:** 2 (AB xor BE can be in the tree).

(b) **(10 pts.) Linear Programming Fundamentals**

$$\begin{aligned} \min\ & 2x_1 + 4x_2 \\ x_1 + x_2 &\geq 30 \\ -x_1 + 2x_2 &\geq 0 \\ x_1, x_2 &\geq 5 \end{aligned}$$

(i) Draw the feasible region of the linear program above, and then find its optimum.

**Solution:** Optimum is 80 at (20,10).




Common mistakes:

- Forgetting to exclude the sliver between $x = 5$ and $x = 0$ from the feasible area.
- Getting the direction of the $2x_2 \geq x_1$ inequality incorrect.
- Not labeling the graph sufficiently (locations of intercepts etc)

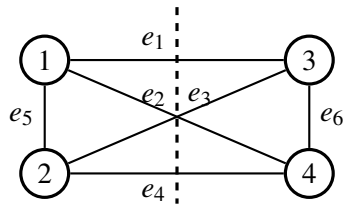
(ii) Find the dual to the linear program above.

**Solution:**

$$\begin{aligned} \max\ & 30y_1 + 5y_3 + 5y_4 \\ y_1 - y_2 + y_3 &\leq 2 \\ y_1 + 2y_2 + y_4 &\leq 4 \\ y_1, y_2, y_3, y_4 &\geq 0 \end{aligned}$$

- Forgetting to exclude the sliver between $x = 5$ and $x = 0$ from the feasible area.
- Getting the direction of the $2x_2 \geq x_1$ inequality incorrect.
- Not labeling the graph sufficiently (locations of intercepts etc)

(c) **(6 pts.) Cut Property**




Consider the cut shown above; it separates vertices 1 and 2 from 3 and 4 in the graph. Suppose we are given the weights of all 4 edges across this cut: $w(e_1) = 1$, $w(e_2) = 2$, $w(e_3) = 2$, $w(e_4) = 3$. But $w(e_5)$ and $w(e_6)$ are unknown. Circle all the answers that apply for the following questions:

(i) Which edge(s) **must be** part of every minimum spanning tree in $G$?

$$(e_1) \quad e_2 \quad e_3 \quad e_4$$

(ii) Which edge(s) **can be but aren't necessarily** part of a minimum spanning tree in $G$?

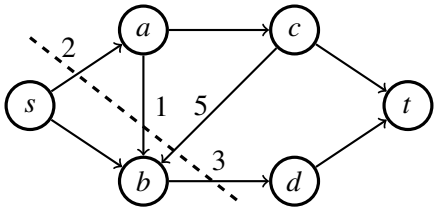
$$e_1 \quad (e_2) \quad (e_3) \quad e_4$$

(iii) Which edge(s) **must not be** part of any minimum spanning tree in $G$?

$$e_1 \quad e_2 \quad e_3 \quad (e_4)$$

**Solution explanation:** $e_1$ is the unique lightest edge across the cut, so must be included. $e_4$ won't be included, because it can always be swapped with one of the other three edges to get a smaller-weight spanning tree.

(d) **(4 pts.) Min Cut**




In the graph $G$ shown above, consider the cut that separates $\{s,b\}$ from $\{a,c,d,t\}$. Assume that this cute is the minimum $(s,t)$-cut in $G$. In the maximum $(s,t)$-flow, what are the flow values through the following edges:

(i) $s \to a$: **2**

(ii) $a \to b$: **0**

(iii) $c \to b$: **0**

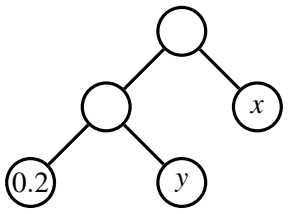
(iv) $b \to d$: **3**

**Solution explanation:** The max flow always fully saturates any edges going in the right direction across the cut and sends no flow in the backwards direction.

(e) **(5 pts.) Huffman Encoding**

Below is a Huffman encoding tree of 3 symbols, where leaf values 0.2, $x$ and $y$ indicate the frequencies of each symbol. Since they represent frequencies, $0.2+x+y=1$. Give the minimum and maximum values that $x$ can take so that this tree still minimizes the expected length of the encoding.

**Solution:** x in [0.4, 0.8]




**Solution explanation:** We know $x \geq y$, otherwise, $x$ would be paired with $y$ instead. $y$ can be as low as 0 so $x = 0.8$, and $y$ can be as high equaling $x$, so $x = y = 0.4$.

(f) **(10 pts.) Zero-Sum Games**

Column:

| Row: | | $c_1$ | $c_2$ |
|---|---|---|---|
| | $r_1$ | 1 | -1 |
| | $r_2$ | -1 | 1 |

In the zero-sum game shown above, Row seeks to maximize the value of the game, and Column seeks to minimize it. Assume that Row must announce her strategy, the probablities $r_1$ and $r_2$, first, and Column determines his strategy ($c_1$ and $c_2$) afterwards.

For example, if Row announced the strategy "$r_1 = 1$, $r_2 = 0$," Column could respond with the strategy "$c_1 = 0.25$, $c_2 = 0.75$," and the value of the game would be $-0.5$.

(i) What strategy should Row choose? Show that your strategy is the unique optimum.

**Solution:** Row should play either row randomly.
Given some probabilities $r_1$ and $r_2$ summing to 1, column will minimize $(r_1 - r_2)$ and $(r_2 - r_1)$. Row wants to maximize this minimum. For any $r_1$ and $r_2$, either they're the same, and the minimum is 0, or they're different, and the minimum is less than 0. Therefore, we must choose $r_1 = r_2 = 0.5$ (since probabilities must sum to 1).

(ii) Assuming that Column responds optimally, what is the value of the game?
**Solution:** The value of the game is the value of the minimum, 0.

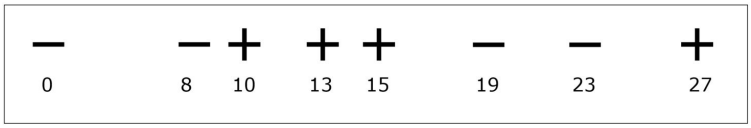
(iii) Show that Row's strategy in (i) is the unique optimum strategy.
**Solution:** This is the unique optimum because it's the only value for which $r_1 = r_2$ (the reason for which is explained in the solution above).

**2. (20 pts.) Matching terminals**

There are $n$ terminals and $n$ negative terminals. Their are all at distinct locations along the $x$-axis; the location of some terminal $t$ can be given by $x(t)$.

You must place $n$ wires, each connecting a positive terminal to a negative terminal, so that at the end each terminal is connected to exactly one wire.

Design a greedy algorithm to place the wires to minimize the total length of wire used. Use an exchange argument to prove that your algorithm is optimal. What is the running time of your algorithm?




The above picture shows a possible layout of terminals for $n = 4$. The location value $x(t)$ for each terminal $t$ is written underneath the terminal.

**Solution:**
Our algorithm is:

- Sort the locations of the positive terminals, and sort the locations of the negative terminals.
- For each value of $i$ from 1 to $n$, connect the $i$th positive terminal to the $i$th negative terminal.

We prove that this works using an exchange argument. Let $a_1, \ldots, a_n$ be the locations of the positive terminals, sorted by increasing coordinate, and let $b_1, \ldots, b_n$ be the locations of the negative terminals, similarly sorted.

Take any different assignment, and let $i$ be the index of the first terminal in which these two assignments differ (i.e. $i$ is the first index in the alternate assignment where the $i$th positive terminal is not paired with the $i$th negative terminal). Instead, let's say that the $i$th positive terminal is paired with the $j$th negative terminal, while the $i$th negative terminal is paired with the $k$th positive terminal. Because this is the first point at which the two assignments differ, we know $a_k > a_i$ and $b_j > b_i$.

Without loss of generality, assume that $a_i < b_i$ (that the $i$th positive terminal comes before the $i$th negative terminal). Examine the situation if we swap the wires and we directly connect the $i$th positive/negative terminals (as per the greedy approach) while connecting the $k$th positive terminal and the $j$th negative terminal together. The length of the wire that is connecting to $a_i$ would decrease by $b_j - b_i$, and the length of the wire connecting to $a_k$ increases by at most that amount (that increase in length is always enough to go from $a_k$ to $b_i$ to $b_j$, but this path might also double back on itself and so could use less wire).

This shows that there is an equivalent-or-better assignment that *doesn't* disagree with the greedy algorithm on the $i$th pairing, which means that any other assignment can be converted via a series of swaps into the same assignment as what the greedy algorithm outputs, without using additional wire, so the greedy algorithm must be optimal.

**Common Mistakes:**
Incorrect greedy strategies:

1. Connect closest $+, -$ pair together, then repeat.
2. Start from left to right, match each negative terminal with the closest positive terminal (A strategy very similar to this that does work (and is equivalent to the solution above) is: start from left to right, match each unpaired terminal with the leftmost unpaired terminal with an opposite sign)

These strategies fail on this counterexample:

$$+ \qquad -+ \qquad -$$

**3. (20 pts.) Knapsack with Weighty and Voluminous Objects**

A thief walks into a house and notices $n$ objects with values $v_1, \ldots, v_n$, weights $w_1, \ldots, w_n$, and volumes $u_1, \ldots, u_n$. The thief has brought with him a bag that has a volume capacity of $U$, and can hold a total weight of $W$. He wishes to choose a subset of items whose total volume is at most $U$ and weight at most $W$, and whose total value is as large as possible, but he must decide quickly and make his getaway. Give an efficient dynamic programming algorithm to help him choose. Clearly specify the subproblems, recurrence, the order in which you would solve the subproblems, and the running time.

**Solution:**

*Subproblem:* Our subproblem is $s(i, j, k)$, which is the maximum value we can get using only the first $i$ objects, and using at most $j$ volume and $k$ weight.

*Recurrence:* The recurrence is $s(i, j, k) = \max(s(i-1, j, k), s(i-1, j-u_i, k-w_i) + v_i)$; the two cases correspond to either not taking or taking the $i$th item.

*Partial pseudocode (subproblem order):*

```
1.  For i := 1,...,n:
2.      For j := 0,...,U:
3.          For k := 0,...,W:
4.              Calculate s[i][j][k].
```

(Note that the three loops could be in any order.)

*Runtime:* We have $\Theta(nUW)$ subproblems, each of which takes constant time to solve, so the overall runtime is $\Theta(nUW)$.

**Common Mistakes:**

- Some students did not explicitly write out the subproblem, and either went straight to writing/explaining the recurrence, or failed to account for the roles of all three arguments.
- Some students chose a subproblem and recurrence that allowed for any item of the array to be taken; in order for this approach to work, the subproblem would have to keep track of the exact subset of chosen items as an argument (which some students forgot); however, it would also lead to a runtime exponential in $n$, which was too inefficient.
- General comment on writing recurrences: if a variable is defined via being an argument to the subproblem, it should not be redefined in the recurrence (e.g. if $i$ is a parameter to the subproblem, then the recurrence should not include have something like $max_i$.
- By far the most common mistake was not specifying "the order in which you would solve the subproblems" in a sufficient manner. Students needed to clearly and unambiguously specify the exact order to solve the subproblems in; the simplest way is to provide pseudocode (as above) and show the three loops, but an unambiguous description in words (e.g. "we iterate through all possible subproblems from $i$ from the smallest to largest value, tiebreaking by $j$ from smallest to largest value, tiebreaking by $k$...") would also be sufficient. Just saying "bottom-up", giving a vague description of "from one corner of the 3D matrix that represents the subproblems to the other", or only mentioning how one parameter varies without mentioning the other parameters (e.g. "for increasing $i$") was insufficient.

- Some students realized that the runtime was a product of three numbers, and so wrote $n^3$; however, there is no guarantee that $U, W = O(n)$. (One way to think about this is: remember that we're discussed how knapsack is a hard problem because the best-known algorithm has an *exponential* runtime, so clearly the runtime couldn't be a polynomial like $n^3$.)

**4. (20 pts.) Breadth requirements**

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.

*Alternatively, for $2/3$ credit, you can ignore the departmental constraints (so the input is just $c_1 ... c_p$, $G_1 ... G_n$, and $F$).*

(a) Formulate the above problem as a linear program. Make sure to clearly identify and define what your variables are.

**Solution:**
For the variables, let $x_{ij}$ be an indicator that represents if the $i$th class is being used to satisfy the $j$th requirement (1 if so, 0 if not).
Our LP is the following:

$$\begin{aligned}
&\max 0 \\
&\forall j : \sum_i x_{ij} \geq 1 \\
&\forall \ell : \sum_{\{i | H(c_i) = D_\ell\}} \sum_j x_{ij} \leq k \\
&\forall i : \sum_j x_{ij} \leq 1 \\
\forall i, j \in \{(i,j) | G_j \in F_(c_i)\} &: x_{ij} \leq 1 \\
\forall i, j \in \{(i,j) | G_j \notin F_(c_i)\} &: x_{ij} \leq 0 \\
\forall i, j &: x_{ij} \geq 0
\end{aligned}$$

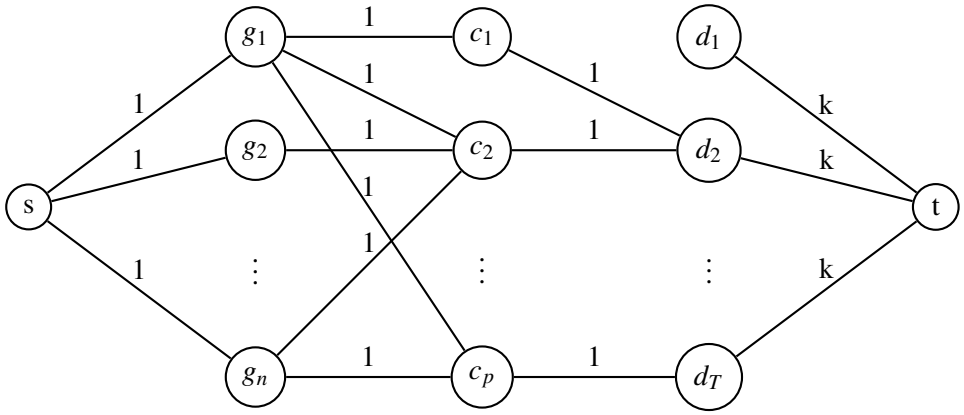
To explain each row of constraints:
The first row of constraints makes sure that every requirement is satisfied (for every breadth requirement, there is at least one class being used to satisfy it).
The second row of constraints makes sure that at most $k$ courses from any one department are being used to satisfy requirements.
The third row of constraints makes sure that each class is being used to satisfy at most one requirement.
The fourth and fifth row of constraints make sure that classes are only being used to satisfy requirements that they can satisfy.
The fifth row of constraints ensures that each variable makes sense as an indicator by providing a lower-bound.
Finally, we only want to see if there is any feasible assignment, and aren't trying to optimize anything, so we don't care about our objective function.

(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.

**Main idea:**

We reduce this problem to a max flow instance, and solve it using a technique similar to bipartite matching.

**Draw a picture of your main idea:**




**Pseudocode:**

1. Set up the following graph $G(V,E)$: let $V = \{g_1...g_n, c_1...c_p, d_1...d_T, s, t\}$ be the set of vertices. Where $g_1...g_n$ correspond to the categories, $c_1...c_p$ correspond to the classes, and $d_1...d_T$ correspond to the departments.
2. For each $G_i \in F(C_j)$, add an edge of capacity 1 between $g_i$ and $c_j$. (i.e. add an edge between a class and a category if the class satisfies the category).
3. For each $C_j$, add an edge of capacity 1 between $c_j$ and $d_\ell$ where $H_{C_j} = D_\ell$ (i.e. add an edge between each class and the department to which it belongs)
4. For each $d_\ell$, add an edge of capacity $k$ between $d_\ell$ and $t$.
5. For each $g_i$, add an edge of capacity 1 between $s$ and $g_i$.
6. Run max flow from s to t, and return "satisfy" if the value of the max flow equals $n$, the number of categories.

**Proof:** First, we show that if the $p$ classes do satisfy all the requirements, then we can get a max flow value of at least $n$: Each requirement $G_i$ is satisfied by some class $c_j$, so we route one unit of flow from $s \to g_i \to c_j \to d_\ell \to t$, where $d_\ell$ is the department node to which class $c_j$ belongs. Now, we verify that no edge is congested more than its capacity:

(a) $(s \to g_i)$: these edges have capacity 1, and we only route one unit of flow for each $g_i$, so each edge only has at most 1 unit of flow.

(b) $(g_i \to c_j)$: these edges have capacity 1, and each requirement $g_i$ is satisfied by a unique class, and we only route 1 unit of flow for each requirement $g_i$, so each edge $(g_i \to c_j)$ is used at most once.

(c) $(c_j \to d_\ell)$: each class belongs to a unique department, and since we only route at most 1 unit through each $c_j$, each edge $(c_j \to d_\ell)$ contains at most 1 unit of flow

(d) $(d_\ell \to t)$: due to the requirement that there can be at most $k$ classes from each department, we are guaranteed that no more than $k$ units of flow is routed through each $d_\ell$, and thus the $(d_\ell \to t)$ contain at most $k$ units of flow

Next, we show that if we can find a flow value of $n$, then we can find a valid assignment of classes to requirements such that all requirements are satisfied. First, we show that the final flow produced by the max flow algorithm has integral flow value on every edge, and we route the maximum flow along the path we found in each iteration.

(a) each $g_i$ has a flow to a unique $c_j$. This follows from integrality, as there is only 1 unit flowing into $g_i$, and each outgoing edge has flow value of either 0 or 1.

(b) each $c_j$ receives flow from most 1 $g_i$: this is because each $c_j$ only has 1 outgoing edge with capacity 1, and we showed that the flow from $g_i$ to $c_j$ is either 1 or 0. This satisfies the constraint "each class can be used to satisfy only 1 breadth requirement"

(c) each $d_\ell$ receives flow from at most $k$ classes $c_j$. Once again, this is because each edge $c_j \to d_\ell$ has flow value either 0 or 1, and so there are at most $k$ classes $c_j$ with nonzero flow associated with each $d_\ell$. This satisfies the constraint "one can at most take $k$ classes from each department"

Thus, from the max flow solution of value $n$, we can find a valid assignment of classes to requirements such that all constraints are satisfied, by assign to each requirement $g_i$ the unique class $c_j$ which receives flow from $g_i$.

**Running time and justification:** This is just the running time of max flow, which takes time $O(k|E|^2)$ or $O(|V||E|^2)$ or $O(|V|^2|E|)$ depending on implementation.

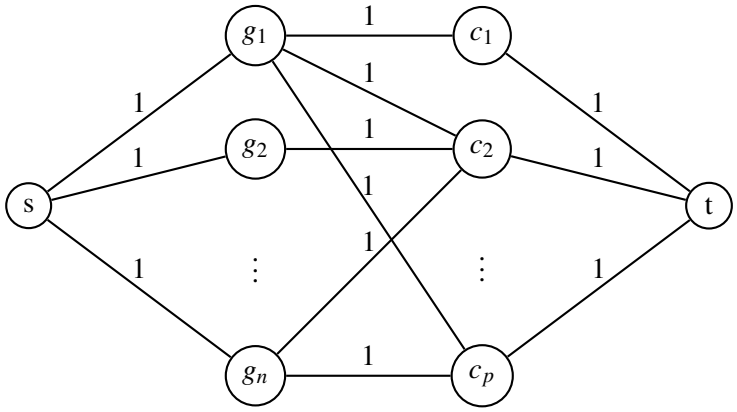
**Common Mistakes** Most students did not include any proof that if all the breadth requirements are satisfied then the max flow is $n$. However, this direction is necessary for a fully correct proof.

There were also a number of incorrect reductions. For instance, putting a capacity of infinity on the edges between the category nodes $(g_1, g_2, \ldots)$ and the source node was incorrect because it could lead to a max flow of value $n$ even when not all requirements are satisfied by allowing a single requirement to contribute more than one to the flow.

Another mistake was to allow more than one unit of flow to enter the class nodes. This allows a single class to contribute flow to more than one requirement.

Finally, there was a mistake on our part. The question was worded unclearly so that it could be interpreted as saying that you are not allowed to take more than $k$ classes from any one department. It was supposed to be interpreted as saying that no more than $k$ classes from any one department can be used to fulfill breadth requirements. Most students interpreted it in the latter way, but the students who interpreted it in the former way were still awarded points if they had a correct solution for that interpretation.

**2/3 Credit Solution**




This is an instance of bipartite matching. We want to match each requirement to a class that can fulfill it. So we form a graph with a node for each class and for each requirement. Each class is connected to each requirement that it could be used to satisfy with an edge of capacity 1. Each class node is connected to the sink with an edge of capacity 1 and each category node is connected to the source node with an edge of capacity 1. The proof is very similar to the proof of the full credit solution above, but without the parts about the department constraints.

**Extra page**

*Use this page for scratch work, or for more space to continue your solution to a problem. If you want this page to be graded, please refer the graders here from the original problem page.*

**Extra page**

*Use this page for scratch work, or for more space to continue your solution to a problem. If you want this page to be graded, please refer the graders here from the original problem page.*