- n colleges in a set C, m applicants in a set A, where m is much larger than n.
- each college $c_i \in C$ has a capacity q_i - the maximum number of students it will admit
- each college c_i has a strict order $\succ_i$ over applicants, $j \succ_i j'$ means college i would strictly prefer to admit applicant j
- eqivalently $i \succ_j i'$ means that applicant j would strictly prefer to be admitted to college i
- write $\emptyset \succ_i j$ when college i just doesn't want to admit applicant i
- let B be a subset of applicants containing less than q_i elements
- college preferences are *responsive* if

$$B \cup \{j\} \succ B \cup (j')$$

  if and only if $j \succ j'$
- means that colleges preferences over students are independent of any other students they have admitted

- a *matching* is a correspondence $\mu : C \cup A \to 2^{C \cup A}$ satisfying

  1. $\forall i \in C$, $|\mu(i)| \leq q_i$;
  2. $\forall j \in A$, either $\mu(j) \in C$ or $\mu(j) = \emptyset$
  3. $j \in \mu(i) \implies \mu(j) = i$

- a matching is *stable* if

  1. for each $i \in C$, $\mu(i) \succ_i \mu(i) - \{j\}$ for each $j \in \mu(i)$;
  2. $\mu(j) \succ_j \emptyset$;
  3. there is no pair $\{i, j\}$ such that $i \succ_j \mu(j)$ and either $|\mu(i)| < q_i$ and $\mu(i) \cup \{j\} \succ \mu(i)$ or there is a j' such that $\mu(i) + \{j\} - \{j'\} \succ \mu(i)$

- finite many to one matching problem with responsive preferences (note that part of this formulation is the assumption that applicants don't care who they are matched with, only where they are matched).

- **Theorem:** a stable matching for a many to one matching problem with responsive preferences is pareto optimal provided preferences are strict (i.e., $i \succ j$ implies $!(j \succ i)$

- PROOF: suppose not. Then there is an alternative matching μ' which makes some applicant or college better off without hurting anyone else

- if that is true, then either $\mu'(i) \succ \mu(i)$ for some college i or $\mu'(j) \succ \mu(j)$ for some applicant j. All we need to prove is either one of these statements is true, some college or applicant must be made strictly worse off.

- the second possibility is easiest, because preferences are strict, applicant j must be matched to a different college in μ' than in μ and it must be one that he or she strictly prefers. Since μ is stable, whatever college i j is matched with under μ' (i.e., $\mu'(j)$) must prefer (strictly) each of the applicants matched with it under μ to j (the set of applicants matched to it under μ (formally$\mu(\mu'(j)) \succ_{\mu'(j)} \mu(\mu'(j)) + \{j\} - \{j'\}\,\forall j'$) otherwise the pair would be a blocking pair.

- so adding j to $\mu(\mu'(j))$ either by displacing another applicant or forcing i to take an applicant he doesn't want makes college $\mu'(j)$ worse off. If $\mu'(j)$ is worse off than it was under μ then

that is the end of the story, the new matching μ' doesn't pareto dominate μ because it makes $\mu'(j)$ worse off.

- however, it could be that $\mu'(j)$ is made better off in the new matching anyway because he is compensated with a bunch of different applicants that he likes

- If that is true he must be matched to at least one applicant who he strictly prefers to some applicant in $\mu(\mu'(j))$. This new applicant must be worse off than he is under μ otherwise $\mu'(j)$ and this new applicant would be a blocking pair. Hence if both j and $\mu'(j)$ are better off in the new matching, then at least one of the applicants in $\mu'(\mu'(j))$ is worse off.

- Proof is similar if it is the college that is initially better off END PROOF.

- with strict preferences, a lot of matchings are pareto optimal, and most of them aren't stable

- e.g College A, College B each with one slot, applicant 1, applicant 2.

- both colleges prefer 1 to 2, both applicants strictly prefer A to B, $\mu(A) = 2$.

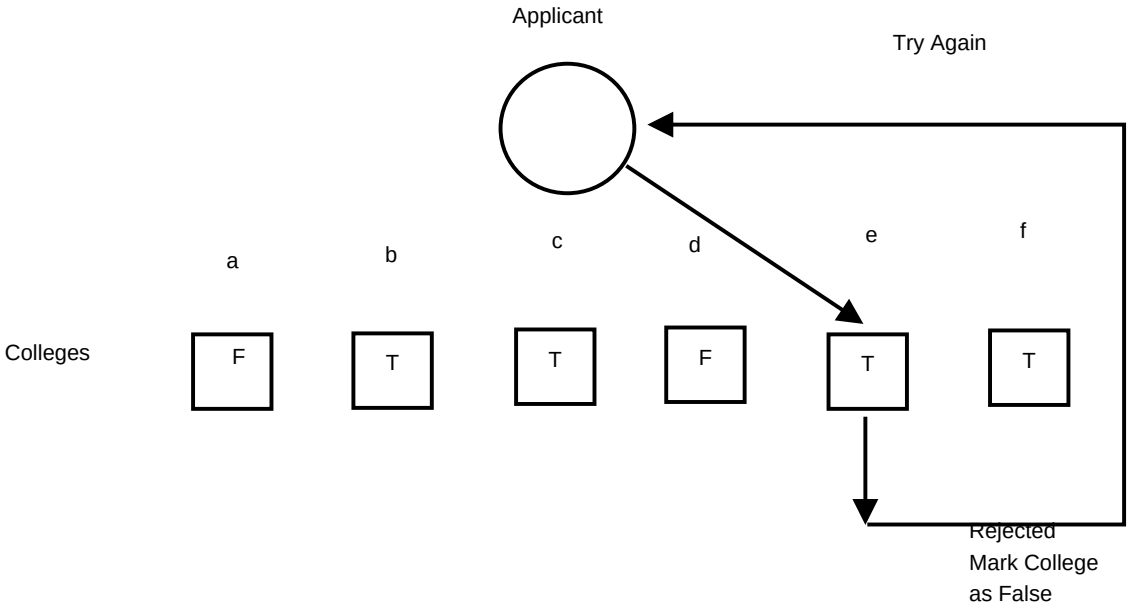
- this matching isn't stable because A wants 1 and 1 wants A, but moving 1 from B to A makes B worse off so it is pareto optimal.

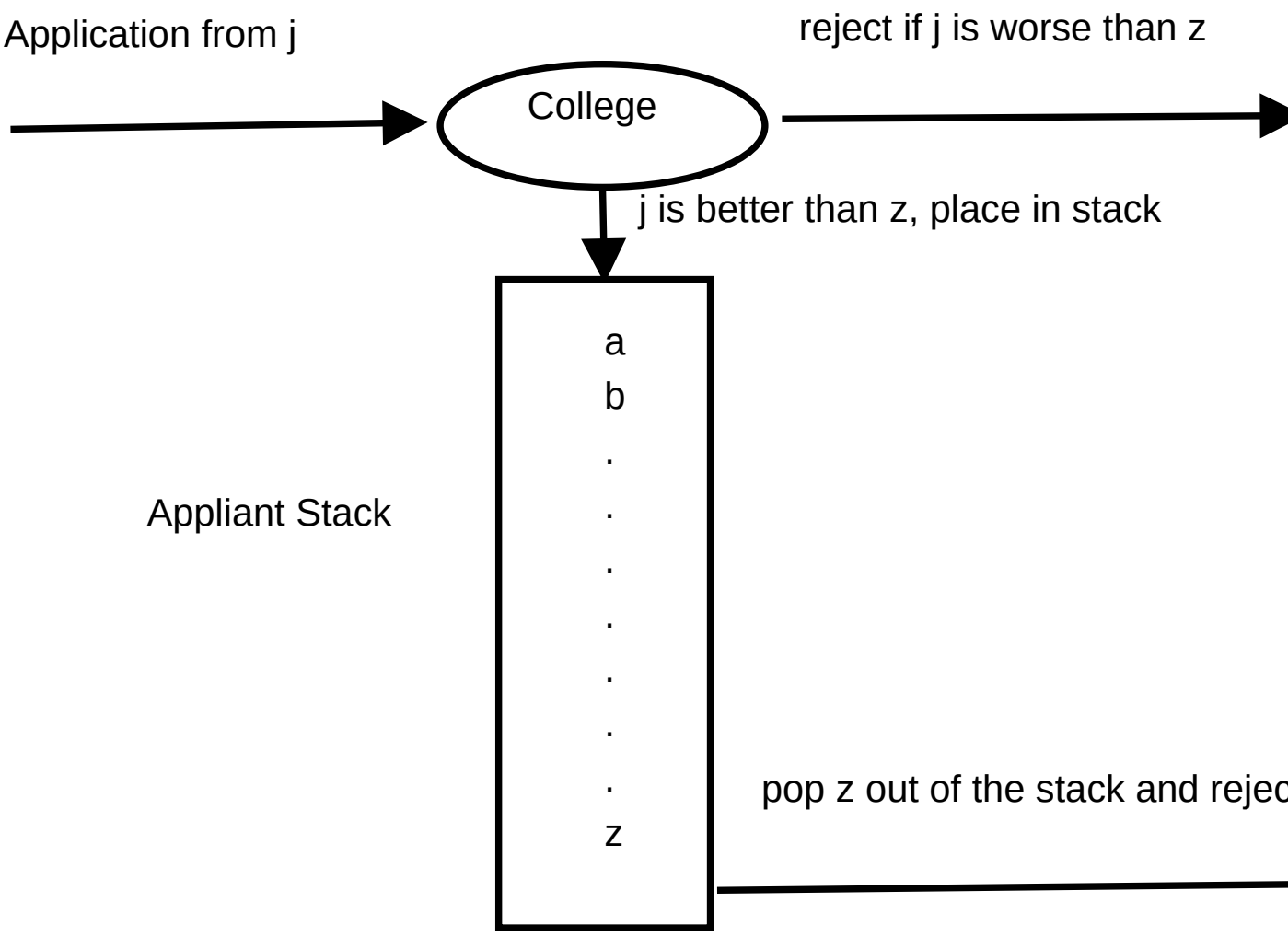
- **Deferred Acceptance Algorithm**

- every finite many to one matching problem with responsive preferences has a stable matching

- a stable matching can be computed algorithmically in a manner that is 'computationally efficient' - meaning that you can find a stable matching more quickly with the algorithm than you can by enumerating all the possibilities then checking them for stability.

- the algorithm is called the deferred acceptance algorithm.


Applicant
Try Again
a
b
c
d
e
f
Colleges
F
T
T
F
T
T
Rejected
Mark College
as False


Application from j
reject if j is worse than z
College
j is better than z, place in stack
Appliant Stack
a
b
.
.
.
.
.
.
.
z
pop z out of the stack and rejec

- Init - Applicant: create a register for each applicant containing a state value for each college. Set the state for each college to true
- an applicant has an input function that adjusts the states in the register, and an output function that sends a message to a college
- Init - College: create a stack to contain the names of admitted applicants, this stack is initially empty
- a college has an input function that processes an application message. If it wants to accept the applicant it puts the name in its stack. If it wants to reject an applicant it sends the applicants name to the output function, that sends a rejection message to that applicant.
- START (A0) - each applicant receives the start message picks its favorite college, the output function sends the an application to that college, then waits for another message.
- APPLICANT RECIEVES A REJECTION (A1): if it receives a rejection from college i, it marks the state for college i as false, picks its favorite college from the remaining open colleges, sends

that college an application, then waits for another message;

- COLLEGE RECEIVES AN APPLICATION (C1): if an application is received from applicant j, then add j to the stack if j is acceptable, and there are fewer than q_i applicants in the stack, then wait for another application; if there are already q_i applicants in the stack, check whether j is preferred to an existing applicant. If he is, put j in the stack, remove the less desired applicant, send him a rejection message and wait for more applications. If all applicants in the stack are better that j send a rejection to applicant j.

- the algorithm ends when there are no more messages

- the state of the applicants registers after all messages are sent defines the matching

- an applicant stops sending messages when he/she has been rejected by every college, or when he/she receives no rejection messages

- if applicant j stops and there are open colleges in his register, his match is his favorite open college - $\mu^* (j)$

- college i stops when it receives no new applications. Its match is equal to the q_i or fewer applicants A_i in its stack - $\mu^* (i) = A_i$.

- notice that each applicant sends at most n messages, there are m applicants, meaning that at most nm messages will be sent to colleges

- each message received by a college causes it to do a finite number of calculations: compare j with z, reorder the elements in a list, send a message

- if there are K operations needed to process a message, then there are at most Kmn operations needed to compute a matching (each applicant will send at most one message, leading to K calculations, there are n applicants)

- to exhaustively check each matching for stability when the number of colleges and student is the same and each student is matched to one college requires at most $K_2 N$ computations, where $K_2$is the number of computations needed to check whether a particular matching is stable and N is the total number of matchings (in the case where all applicants must be matched with some college and $n = m$, $N = n!$

since applicant 1 can be matched with n possible schools, once he is done, applicant 2 can be matched wtih $n-1$ possible schools, and so on.

- heuristically this means that when the number of colleges and applicants is large, the deferred acceptance algorithm will normally define a matching with fewer calculations that brute force checking

- **Theorem 1:** The matching μ^* defined by the deferred acceptance algorithm is stable.

- **Proof:** Suppose to the contrary that μ^* isn't stable. If there is a blocking pair, then some applicant j has a college i' that she would prefer to $\mu^*(j)$, while i' prefers j to at least one of the applicants in $\mu^*(i')$. Since j always applies to the best open school by (A1), college i'must have rejected the application from j at some point. By (C1) college i' only rejects j if its worst applicant is better than j. Again using (C1) any subsequent applicant accepted by j will be preferred to its worst applicant, which implies that each applicant in $\mu(i')$ is preferred to j, a contradition.

- implicitly. a matching is an algorithm that takes as inputs, a set of preferences $\{\succ_j, \succ_i\}_{i=1,m;j=1,}$ and produces a stable matching

- so we could write the matching as $\mu\left(j, \{\succ_j, \succ_{-j}\}\right)$ following the convention that for any applicant j, the algorithm produces a stable matching using preferences $\{\succ_j, \succ_{-j}\}$ (here $\succ_{-j}$ means the preferences of all the other applicants and colleges)

- **Theorem 2:** Let μ be the matching supported by the deferred acceptance algorithm. For any applicant j and any preference$\succ_{-j}$,

$$\mu\left(j, \{\succ_j, \succ_{-j}\}\right) \succ_j \mu\left(j, \left\{\succ'_j, \succ_{-j}\right\}\right)$$

  provided the two assignments are different.

- **Proof:** see Roth and Sotomayer

- there are typically many stable matchings. For some applicant j, say i is attainable by j if there is *some* stable matching in which j is matched with i.

- **Lemma 2:** suppose that at some point during the operation of the deferred acceptance

algorithm, j sends an application to an attainable partner i. Then i will either prefer j to some applicant i has already accepted, or will be willing to accept j into an empty slot.

- **Proof:** Since i and j are matched in some stable allocation, j must be acceptable to i. So if fewer than q_i applications have been accepted, then i will tentatively accept j. Suppose the assertion of the theory is true after k applications have been submitted and that i has already accepted q_i applicants. Suppose that, contrary to the assertion in the lemma, j is no longer acceptable to attainable college i after the $k + 1^{st}$ application has been submitted. Then i must prefer all q_i of his existing applications to j. By the induction hypothesis and the properties of the deferred acceptance algorithm, any applicant accepted already by i after $k + 1$ applicants have been submitted, prefers i to any college that is attainable for them. Since i is attainable for j, consider some matching μ' in which j is accepted by i. Since there are q_i applicants who prefer i to any college that is attainable to them, they prefer i to their match in μ', and i prefers each of these applicants to j. So there must be at least one

blocking pair for μ'.

- **Corollary:** In the (applicant proposing version of the) deferred acceptance algorithm, each applicant weakly prefers his match to the match he receives in any other stable allocation.
- **Proof:** if he prefers the school in the other application, then he should just apply, by lemma 2 he would be accepted.
- **Corollary:** the allocation produced by the daa is in the core. Since any alternative allocation is pairwise stable, this follows from the last Corollary.
- Example

	a	b	c	d
SFU	1,3	2,2	3,2	4,3
UBC	2,1	1,1	3,1	4,1
UT	4,2	2,3	1,3	3.2

- SFu prefers a to b to c to d, student a prefers UBC to UT to SFU

	a	b	c	d
SFU	1,3	2,2	3,2	4,3
UBC	2,1*	1,1*	3,1*	4,1*
UT	4,2	2,3	1,3	3,2

	a	b	c	d
SFU	1,3	2,2	3,2	4,3
UBC	2,1*	1,1*	3,1*	4,1*
UT	4,2	2,3	1,3	3,2

	a	b	c	d
SFU	1,3	2,2	3,2*	4,3
UBC	2,1*	1,1*	3,1*	4,1*
UT	4,2*	2,3	1,3	3,2*

	a	b	c	d
SFU	1,3	2,2	3,2*	4,3
UBC	2,1*	1,1*	3,1*	4,1*
UT	4,2*	2,3	1,3	3,2*

	a	b	c	d
SFU	1,3*	2,2	3,2*	4,3
UBC	2,1*	1,1*	3,1*	4,1*
UT	4,2*	2,3	1,3	3,2*

	a	b	c	d
SFU	1,3*	2,2	3,2*	4,3*
UBC	2,1*	1,1*	3,1*	4,1*
UT	4,2*	2,3	1,3*	3,2*

- Indifference - schools may not care what students they admit. As there is excess demand they will have a lottery which determines preferences. Then this outcome is not pareto optimal because a and c can trade places.
- top trading cycle
- Interviewing
- n schools, m applicants, k interview slots
- many to many matching problem with additional constraints imposed by interviews
- a *matching* is a correspondence $\mu : C \cup A \to 2^{C \cup A}$ satisfying
    1. $\forall i \in C$, $|\mu(i)| \leq k$;
    2. $\forall j \in A$, $|\mu(j)| \leq k$;
    3. $\forall j \in A$, either $\mu(j) \subset C$ or $\mu(j) = \emptyset$
    4. $\forall i \in C$, either $\mu(i) \subset A$ or $\mu(i) = \emptyset$
    5. $j \in \mu(i) \iff i \in \mu(j)$
- a matching is *stable* if
    1. for each $i \in C$, $\mu(i) \succ_i \mu(i) - \{j\}$ for each $j \in \mu(i)$;

2. for each $j \in A$, $\mu(j) \succ_j \mu(j) - \{i\}$ for each $i \in \mu(j)$;
3. there is no pair $\{i, j\}$ such that $\mu(j) + \{i\} \succ_j \mu(j)$ while $|\mu(j)| < k$ or $\mu(k) + \{i\} - \{i'\}$ for some i' in $\mu(j)$; while at the same time $|\mu(i)| < k$ and $\mu(i) \cup \{j\} \succ \mu(i)$ or there is a j' such that $\mu(i) + \{j\} - \{j'\} \succ \mu(i)$.

- the deferred acceptance algorith needs only a minor modification
- for colleges the set up is the same as in the original case
- for applicants, they begin with an array of truth values as before, one element for each college, each initialized to true. They also have a register consisting of exactly k elements
- START (A0) - each applicant receives the start message picks his or her k favorite colleges, places each of them in the register, then sends an application to each of the colleges in the register and then waits for another message.
- APPLICANT RECIEVES A REJECTION (A1): if it receives a rejection from college i, it marks

the state for college i as false, and removes college i from its register. He or she then picks their favorite college from the group of colleges that still have value true, sends that college an application, then waits for another message;

- COLLEGE RECEIVES AN APPLICATION (C1): if an application is received from applicant j, then add j to the register if j is acceptable, and there are fewer than k applicants in the register, then wait for another application; if there are already k applicants in the register, check whether j is preferred to one of the applicants who is already there. If he is, put j in the stack, remove the less desired applicant, send him a rejection message and wait for more applications. If all applicants in the stack are better that j send a rejection to applicant j.

- exactly as in the other mechanism, an applicant can't send more than n messages, a college can't receive more than m messages, so the process has to stop in finite time.

- the matching is given by the state of each applicant and school's register

- **Theorem:** if both applicants and schools have

responsive strict preferences, the matching μ^* generated by the deferred acceptance algorithm as decribed above is pairwise stable.

- **Proof:** the proof is the same as in the many to one case. To make things slightly simpler, we'll assume that $|\mu^*(t)| = k$ for all $t = i, j$. Suppose not. Then there is a blocking pair consisting of some college i and some applicant j, i.e. there is some applicant j' such that $\mu^*(i) - \{j'\} + \{j\} \succ \mu^*(i)$ and some college i' such that $\mu^*(j) - \{i'\} + \{i\} \succ \mu^*(j)$.

- since applicants apply in order of their preference to colleges and j prefers i to some school that he was matched with, j must have applied to i at some point and been rejected. Since i and j are a blocking pair, j must be acceptable to i, so i must have had applications from applicants that it preferred to j. Since all applicants that i accepted subsequently are preferred to applicants that are preferred to j, j must also be worse than every applicant interviewed by i in the matching μ^*, which is a contradiction.**End Proof:**

- Now we have a second problem - we have a

matching, but how do we know we can feasibly schedule interviews

- another algorithm - the reason the matching is constrained to contain exactly k elements is that there are exactly k interview slots - for example k consecutive half hour time slots.

- given a matching μ, a *schedule* is a mapping $s : C \times A \rightarrow \mathrm{K}$, where K is the set of time slots

- a schedule is feasible for μ if

- $s(i,j) \neq s(i,j')$ for each $j \in \mu(i)$ and $j' \in \mu(i)$

- $s(i,j) \neq s(i',j)$ for each $i' \in \mu(j)$ and $i \in \mu(j)$.

- **Theorem:** There is a feasible schedule for each a matching μ.

- This follows from something called Vizing's Theorem

- a *graph* is a finite collection of vertices $\mathcal{V}$ and a collection of edges $\mathcal{E}$ that connect the vertices. The *degree* of a vertex is the number of

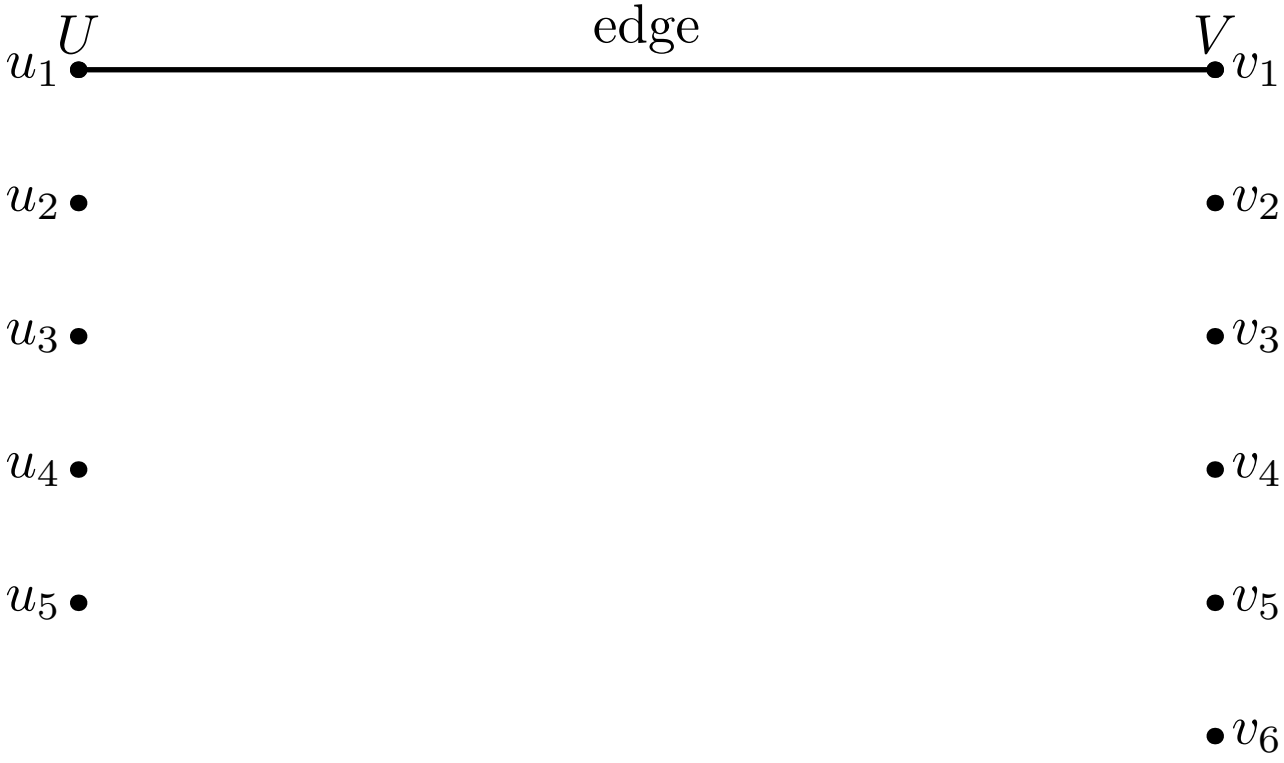
edges connected to that vertex. The degree of a graph $\triangle$ is the maximum number of edges connected to any vertex. Edges are *adjacent* when they connect to the same vertex.

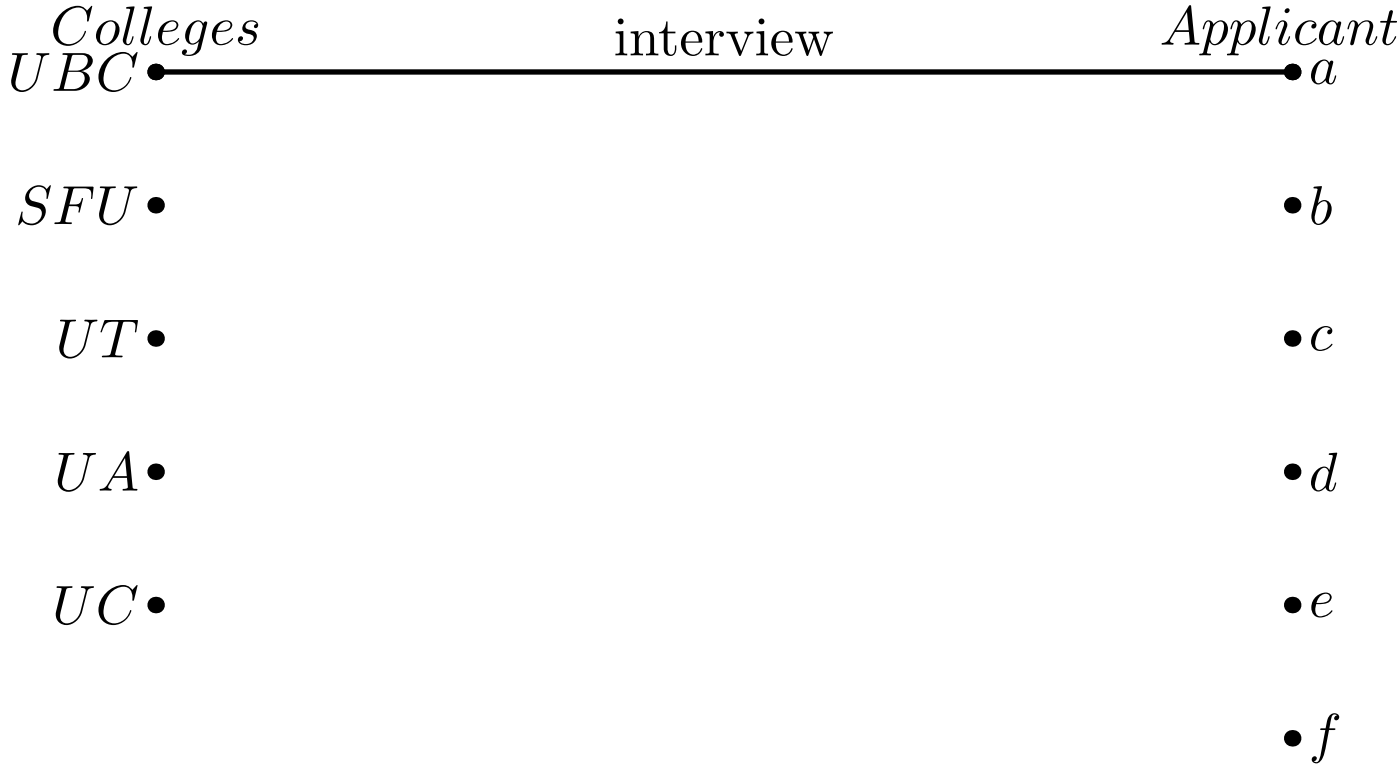
- A graph is *bipartite* when the set of vertices $\mathcal{V}$ can be divided into two subsets U and V in such a way that every edge connected to a vertex in U is also connected to a vertex in V and conversely.

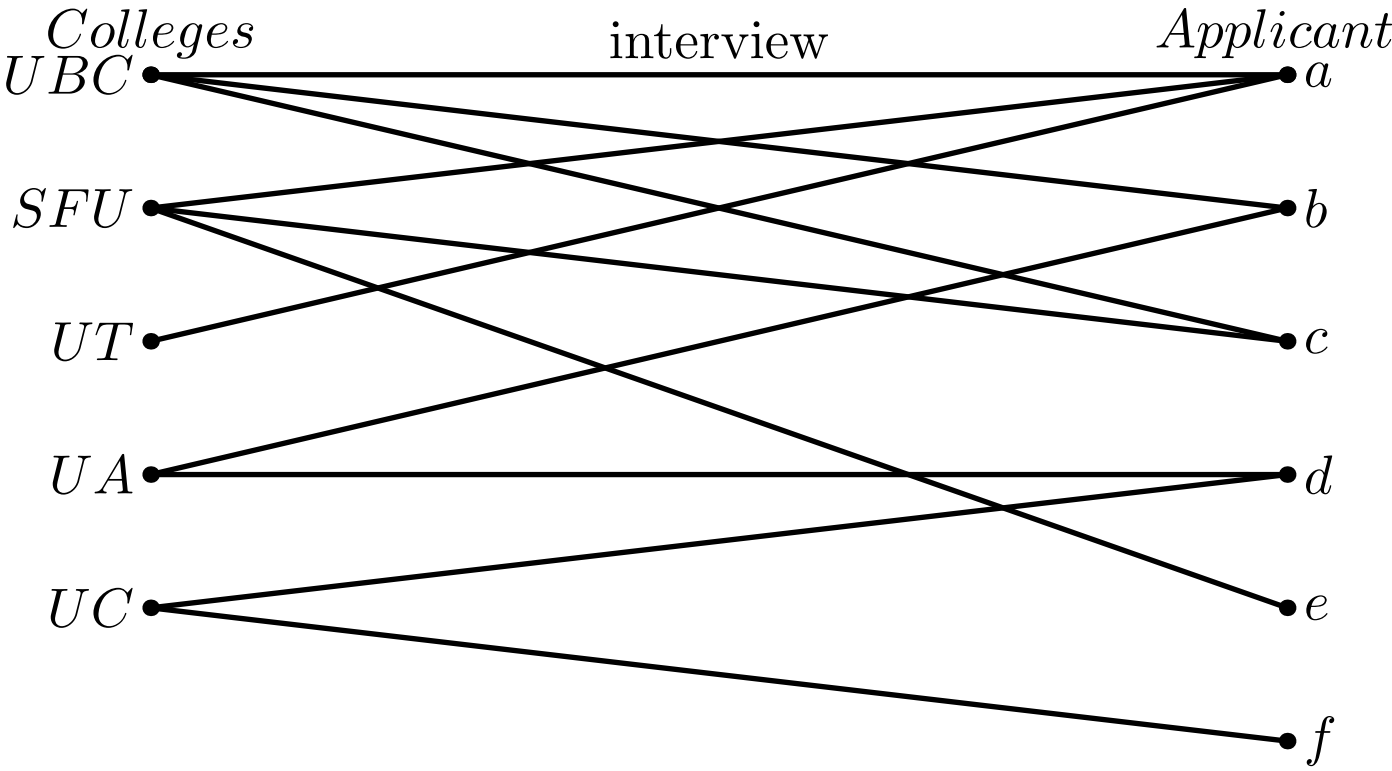
- Every many to many matching can be represented as a bipartite graph.

- $\mathcal{C}$ is a collection of *colors.* A graph coloring c is a mapping from $\mathcal{E} \rightarrow \mathcal{C}$ (literally each edge is colored)

- **Vizings Theorem:** every bipartite graph of degree $\triangle$ can be colored in such a way that no two adjacent edges have the same color, and such that the range of the coloring contains only $\triangle$ distinct colors.


U
edge
V
u_1
v_1
u_2
v_2
u_3
v_3
u_4
v_4
u_5
v_5
v_6


Colleges
interview
Applicant
UBC
a
SFU
b
UT
c
UA
d
UC
e
f



- in the diagrams above, there are at most three edges emanating from any vertex, i.e., three interviews. So label the interview slots $\{1, 2, 3\}$ and match this with the set of colors {Red, Green,Blue}. The theorem says that edges can be colored either red green or blue in such a way that no two adjacent edges have the same color,

- To see the analogy, just think of each vertex as either a college or an applicant, and each edge a match between a college and applicant. The colors are the interview slots. So the theorem says that we can assign the k interview slots to

college applicant pairs in such a way that for every college, no pair of applicants that it is matched with is given the same interview time, and for each applicant, the applicant never has interviews scheduled with two colleges at the same time.

- **Proof of Vizings Theorem for interviews (a special case):** You can use the pictures above to follow the logic. Suppose we select an arbitrary subset E of the edges in this graph. If $|E| < k$ we can trivially assign each edge to a different interview slot. So lets proceed inductively and suppose that for any subset E containing t edges, we can assign interview times to each of those edges so that no two adjacent edges have the same interview time. This means that no college interviews two students at the same time, and no student has two or more interviews scheduled at the same time.

- the next step is to take a collection of edges E such that $|E| = t + 1$. By the induction hypothesis, if we were to choose any edge in E arbitrarily and take it out, we would be able to devise a feasible interview schedule for the

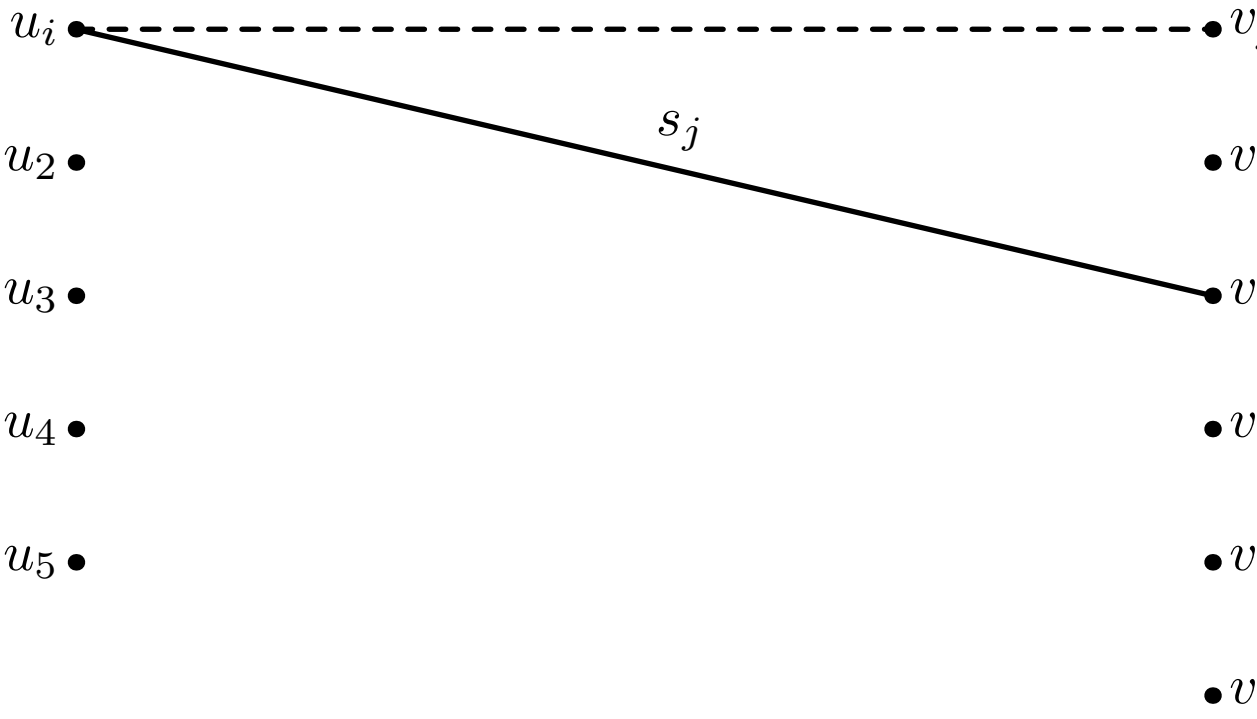
remaining edges. Lets do that, so that we have a feasible interview schedule for everyone but the pair consisting of college u_i and applicant v_j who are connected by the edge e' that we took out of E.

- Since no applicant can have more than k matches, the number of interviews the applicant v_j has left after we take out e' is at most $k-1$ - meaning that there is an interview slot that j hasn't yet used, lets call it s_j. We might try to use the slot s_j to schedule an interview for v_j with college u_i. The complication, of course, is that college u_i might be using the slot s_j already in the interview schedule we created for the other slots. If u_i isn't using that slot, we can schedule the extra interview.

- If college u_i is using slot s_j, there must still be *some* interview slot u_i isn't using, for the same reason that v_j had a free interview slot. Let s_i be the slot that that college i isn't using.

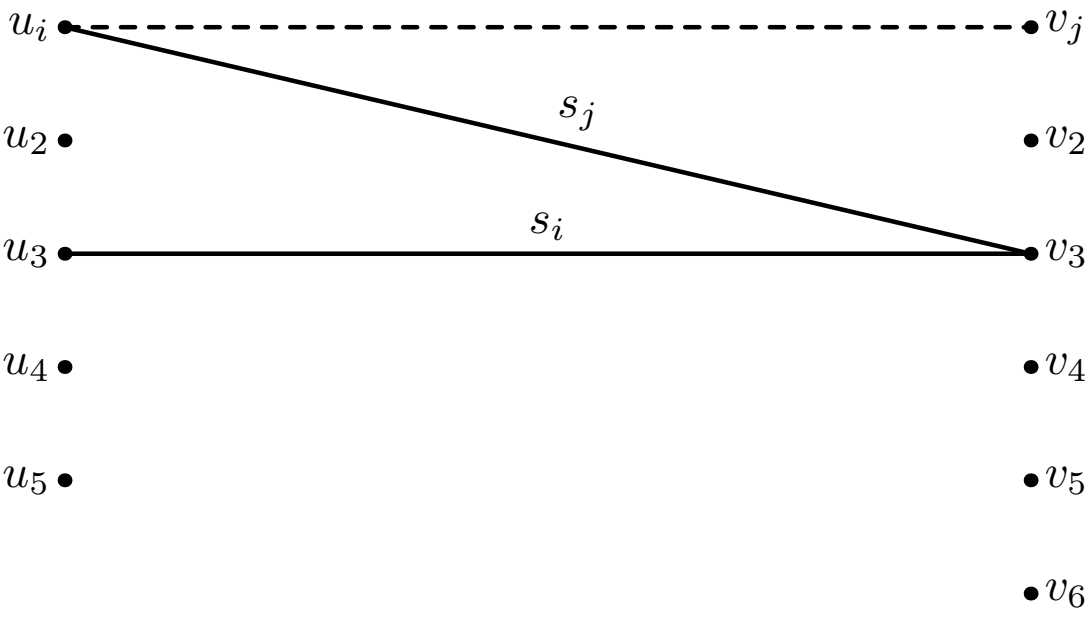
- So lets create a chain of edges starting with college u_i that continues until we find a free slot. If u_i is using s_j, identify the applicant who is scheduled to interview with u_i in slot

s_j. In the figure, this is applicant v_3. The s_j along the edge indicates that applicant u_3 is currently scheduled to interview with college u_i in slot s_j (the one v_j has open).



- Check to see if the conflicted applicant has slot s_i free. If s_i is free, then we can reschedule her into slot s_i then schedule u_j in slot s_j, which would be a feasible schedule.

- But we might not be able to do this if u_3 in the diagram is already scheduled in slot s_i. In this second figure, we draw an edge between v_3 and u_3 to indicate that we have already scheduled them in slot s_i.

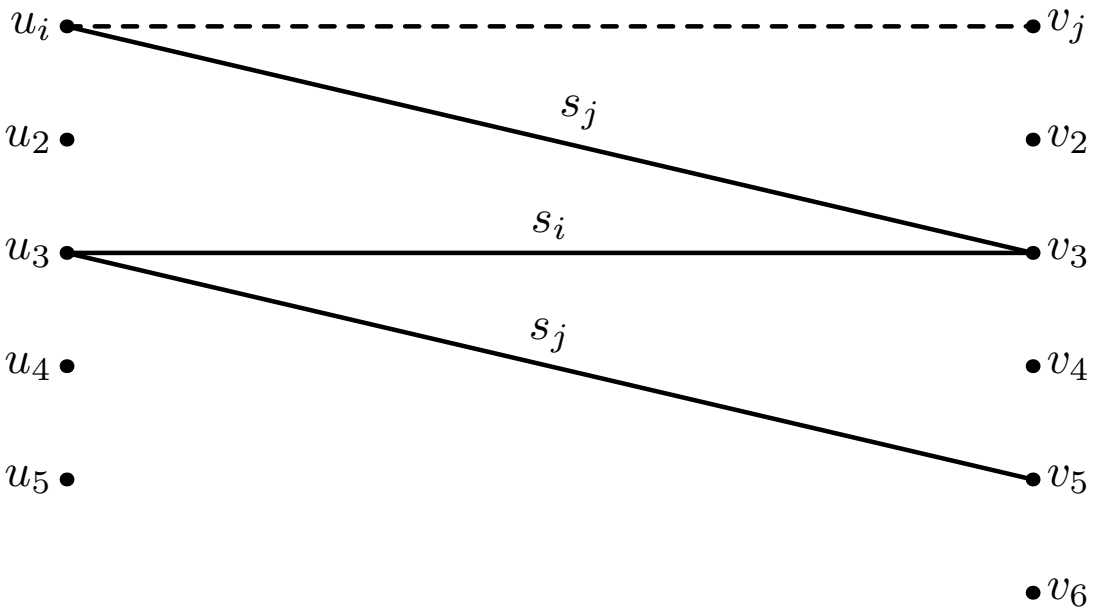


- the next thing to check is whether school u_3 is using slot s_j (the one that our original applicant has free). If it does, then we can reschedule. The way to do it is to move applicant u_3 from her current slot s_i with college u_3 into the slot s_j with college u_3. To restore her interview with college u_i we could move her from slot s_j to slot s_i with college u_i.

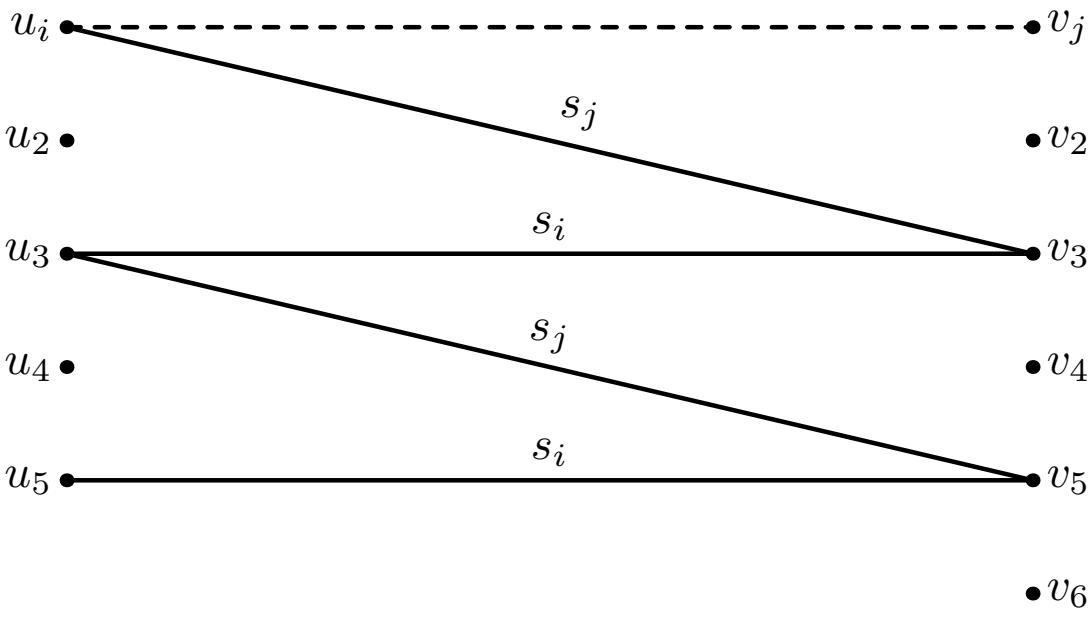
- What would be do if college u_3 already has an interview scheduled in slot s_j? Just repeat, find the applicant who is scheduled to interview with u_3 in slot s_j and ask them if they can move their interview to slot s_i.

- If they can, here is what we would do - move their interview in slot s_j to slot s_i. Of course,

u_3 is currently scheduled to interview applicant v_3 in slot s_i, so we can instead switch their interview with college u_i from slot s_j to s_i. Then applicant u_3 ends up with the same two interview slots being filled, but with the colleges interchanged. This would free up slot s_j for applicant v_j.



- Lets do one more - if applicant v_3 doesn't have s_i free, we'll find the college with whom she is scheduled to interview in slot s_i, and check whether that college has s_j free.



- It might say no, in which case we would repeat. But eventually someone has to say yes. The reason is that if it kept repeating, we only have a finite number of colleges. So eventually we would have to loop back and hit either college u_i or one of the others we had encountered before. It could never be u_i because we can only get to u_i if u_i is scheduled with some applicant in slot i, but we started with the fact that u_i has slot s_i free.

- looping back to a college other than u_i (without hitting u_i first) is also impossible because the loop takes us through an interview scheduled in slot s_i. Since the original schedule was feasible, each college has only one applicant scheduled in slot s_i, so we would be looping

back through an applicant scheduled in slot j, which ultimately leads back to u_i.

- to complete everything, lets just explain the rescheduling that occurs when college u_5 says that it has slot s_j free. First v_5 is rescheduled from slot s_i with u_5 to slot s_j with u_5. His interview with u_3 is rescheduled to slot s_i. So v_5 and u_5 end up with the same interviews (though at different times). Now u_3 has a conflict, because v_3 was originally schedule in slot s_i with u_3, so move his slot to s_j then move his interview with u_i to slot s_i which we know u_i has free. This opens up slot s_j with college u_i so we can accomodate v_j. This complete the induction.

- **End Proof:**

- notice that this proceedure is basically an algorithm. Start with an arbitrary but incomplete interview schedule, add a new interview that has to be rescheduled, then reshuffle the other interviews to make way. Our argument above shows that this will eventually schedule all the interviews.