Editorial

Welcome to issue 138 of Beta! This week we’ve got some great articles for you, from tips on approaching those dreaded recruitment processes for tech jobs to understanding how multiplayer works in video games. We’ve also got a really interesting piece on why you should be taking more math courses as a computer science student to top it all off.

We hope you’re liking the new design! The team is always keen to hear feedback so make sure you send an email if you have any qualms.

We’ve noticed that a lot of the times, people don’t necessarily pick up copies of Beta to take back with them during the weekly BBQ. Printing these copies takes a lot of time and resources, and it’d be a shame for us to print out copies only for them to never actually be read. This is why we’re considering not printing Betas anymore. But rather than taking this decision ourselves, we want to hear from you.

What do you think of printing Betas? Send us an email and we’d more than happy to discuss this issue with you!

Lastly, an issue that has been brought up countless times is the name Beta itself. It has very clear negative connotations, especially within the computer science community, which is why we’re considering changing it. Again, we’d like this decision to be as transparent as possible, so please contribute to the discussion by sending us an email with your thoughts.

As always, if you have any other issues or just want to contribute, just send us an email at publications@csesoc.org.au.

Enjoy the issue!

Siddhant Virmani
Editor in Chief | Publications Director 2018
WHAT’S ON
Programme - May 2018

Product Manager Panel with WIT
Where: Level 1 K17, Seminar Room, UNSW
When: Tuesday May 1, 12:00PM -

In this panel, you’ll hear from several experienced Product Manager’s from a range of different companies representing a diverse set of product areas. You’ll learn about what Product Management is, what skills it involves, and what paths you can take in your career towards becoming a PM. There’s also free food!

Submit questions and find attending companies here: http://bit.ly/fb-pmpanel

Google Kaggle Codelab
Where: Level 1 K17, Seminar Room, UNSW
When: 1:00PM - 2:00PM,

Google is running an interactive workshop about Kaggle, a platform for data science competitions! Kaggle is great for learning trends and techniques in machine learning and data science.

Bring your laptops, signup at kaggle.com, and get excited for a codelab you won’t want to miss.


CSESoc Weekly BBQ
When: 11:00 AM onwards
Where: John Lions Garden

Come join us on Wednesday for our weekly BBQ! Pick up your weekly snag and get to meet new people before heading to the Google Kaggle Codelab!

Pubcrawl with CSE Revue
When: 5pm on May 3rd
Where: TBA
Ticketing:
- $5pp or $20 per team of 4-6
- Please pay at the weekly BBQ or on the night

Theme: Neon Night
Ticketing:
- $5pp or $20 per team of 4-6

CSESoc and CSE Revue are partnering up again to bring you a wild night with hectic challenges and pubs to crawl to. So grab a team of you and your 5 best mates and sign up for the best night ever!

Sign up and find more info here: http://bit.ly/fb-pubcrawl
How to: Tech Job Applications
By Oliver Nordon

The Resume Stage:

The Resume Stage may seem like the hardest stage in getting into the company because it’s the stage where you have the least control. Companies like Google will receive thousands of new applicants every day and they have to sift through all those resumes and pick out who is the most qualified. Now because so many applicants usually apply, the goal of the resume is actually NOT to make you seem the most qualified, but to make you seem the most interesting.

Usually, whatever numbers you put down on your resume there’s someone out there with higher ones. Even if you have a 95 WAM someone might have applied with a 96 WAM (I for one had a 78 WAM which is good but not the best!). When people sift through the resumes they’ll read the first few lines and they’ll only read the rest if you’ve caught their attention and, in general, numbers don’t really catch attention. Now I’m not saying your WAM isn’t important, of course it’s important to show general competency, but if you have a low WAM that doesn’t mean you can’t get into Facebook; WAM is just the tip of the iceberg.

Now at the start of your resume you should quickly state the important stuff - the degree you’re studying or have graduated from, a quick summary of past work experience and your WAM in the top corner. But also on the front page, hopefully near the top, you should have a list of things that make you interesting. I like to call it my “Special Achievements” list.

Have you ever won an award?
Put that down.

Have you ever done charitable work?
Put that down.

Have you been a tutor or Mentor?
Put that down.

Are you the officer of a student society?
Put that down.

I’m a former UNSW CSE student who interned at Microsoft and now works as a Site Reliability Engineer - Software Engineer at Google Sydney.

Throughout my time at university I spoke with a lot of great people and got a lot of advice on how to get employed by all the big tech companies like Google, Facebook and so forth. I’ve also seen a lot of talks that people make on this topic and based on my experience, I feel there’s a lot that isn’t said. Due to this lack of information, I wanted to do my bit and shine some light on the often unspoken in and outs of getting employed and things to work on both in and out of the interview process.

So for starters, almost all tech companies follow a similar process, first there’s the resume stage in which, to the tech company, you’re just a piece of a paper. Then there’s the interview stage, there could be 2-4 interviews, and this is usually the stage most people stress over. Finally there’s everything else, a lot of people don’t realize there’s stuff you got to do in between interviews that is pivotal for your success in getting hired. So let’s get started!
All these things are applicable because a lot of technical companies aren’t just looking for technical expertise, they’re looking for passionate people who are involved with their communities. Even if the student society isn’t directly related to computer science, it still demonstrates passion and organization skills and, above all, it’s interesting. One of my special achievements that I put on my resume was I was the head of a large gaming clan, during my interview I was asked about it in a somewhat confusing tone, but it definitely got their attention as it was something unique. Being unusual is good here, it separates you from the rest of the applicants and has the reviewer take another look at your resume.

Obviously all your special achievements shouldn’t be crazy, you should have at least a few that demonstrate an interest in computer science (Quick tip: If you have a distinction WAM you’re part of the Dean’s Honours list of Engineering, that sounds a lot better than saying you have a 75 WAM).

Now some of you may not have any “special” achievements, not to worry, getting these is actually quite easy. Remember the goal is to demonstrate passion for what you do, not necessarily technical expertise, so here are some examples of things you can do to add to your resume:

- **Get involved in a student society (CSESoc!)**
- **Do something for charity** (charity is a big tech company magnet, seriously find places to volunteer around you!)
- **Work on a project with your friends outside of Google** (it could be an app, a video game or whatever! It doesn’t even have to be good or finished, but showing you’re doing stuff outside of uni is really good, and doing it in a team shows good skills as well!)

Now your resume can be really interesting but there are a few things that can take it further. The first and most obvious example is to get referred by someone; this is super helpful because it means recruiters are guaranteed to have a more careful look at your resume. Although getting referred does have its benefits, it’s still important to make sure that your resume stands out! In terms of finding someone to refer you, it is generally not too difficult at UNSW to find someone and people generally like to refer people (they usually get money if you get in!) so just ask around!

Secondly, go to the CSE careers day. I’ve never been involved much in CSESoc but I found this to be extremely helpful. Recruiters generally pay more attention to people they meet during careers events so connecting with a recruiter and handing them your resume is definitely going to improve your chances of having your resume looked at more closely.

Finally, the last tip of getting over the resume stage is to be confident. Some of you may not think they have what it takes to work at some of the big tech companies and if you maintain that attitude then you definitely won’t. It’s ok to feel you’re not good enough, but a lot of the time you are! Make your resume with confidence, make it interesting (and a little quirky), get referred or go to a careers event and you’ll have the best chances possible of getting through. Confidence is key!

### The Interview Stage:

So you got past the resume stage and now you’re stressing out, it is time for the dreaded tech interviews. Tech interviews are definitely the most stressful part of the recruitment process and throughout one you will be plagued with questions about your own competency. But there’s actually a very simple secret to cracking these technical interviews.

The actual questions are never that hard! The questions Google and Microsoft asked me were questions straight out of first year, that’s it, they can be solved after taking COMP2521 (usually with a bit of a twist). Determine who is a winner at noughts and crosses, traverse a binary search tree in some weird way, represent a system as a graph; the questions, when you take a step back, are not that difficult. But when you’re doing them they will seem like the most difficult questions you’ve ever been asked and that’s because of stress. That’s why for this section I’m mainly going to be talking about managing stress throughout the interview rather than solving the interview questions themselves.
Now before I go on, it's CRUCIAL that you get this little part right: you want to be likeable. If you can make your interviewer laugh then in general you've already won the interview! When an interviewer interviews you they're asking themselves "Would I work with this person?" not "Is this the smartest person I've seen?". You want to show some level of skill obviously but the most important thing is just making sure you're likeable (and it's not as easy as it sounds, remember during the interview you'll be under a lot of stress!).

So let's talk about managing stress. The first thing that helps with stress is confidence; if you have the confidence that you can solve the questions then you won't be as stressed. So to get this confidence you need practice, so this means studying and doing a bunch of interview questions beforehand. I would recommend "Cracking the Code Interview" as the first resource you should use, it is a book that has more than 170 excellent interview questions and also has some solid advice from recruiters of many large tech companies. When you're doing each question you should spend at least 30 minutes on the question before looking at the answer (unless you're able to solve the question quicker than that!). Don't memorize the answers tech companies always put a weird twist on their questions that makes them slightly different from what you might have memorised, and throw you off. You should concentrate on getting the skills to solve the questions, rather than just answering the questions straight up.

Remember: you may find the first few questions difficult and might get demotivated, just keep going, you'll definitely get better with practice!

Now a quick side note, a lot of people use sites like leetcode for practice questions. I generally would only use this if you've run out of questions to study on, leetcode doesn't really simulate an interview environment and it gives you a grade at the end. Getting a grade seems like a good thing but in general you should learn to assess your own answers, you should try to improve your instinct of looking for ways to improve your own answer. When you're doing question you should also try to do them without a computer (on pen and paper) and maybe putting them on a computer afterwards to confirm your answer. In most on-site interviews you won't be coding in an IDE and will have to code on paper or a whiteboard so practicing how to do that is super important.

So let's say you've studied as much as you can and the interviewer throws a curveball at you; a question you've never seen before and now you're getting stressed. Don't worry, as I said before, these questions are usually easy and most of the time they have an obvious "bad" (usually brute force) answer that works but could be improved. If you can't think of a good answer straight away, go immediately for the "bad" answer and once you have it figured out, then you can work on improving it. Again, you're doing this to reduce stress, so just having an answer can really help calm you down, even if the answer turns out to be in exponential time. So start off by figuring out a naive solution, and then start to think of ways to improve it. If you can't figure out a better answer in time then at least having a naive solution is better than no solution at all.

Now when you figure out the inefficient solution, don't start writing code for it straight away, you might come up with a better answer soon enough and you don't want to waste all that time writing code. However, you may ask that if you are not writing code, how does the interviewer know you have an answer?

This brings us to our third point, always communicate. You should always be talking during the interview, there should never be a period of silence more than 30 seconds, you should always be trying to explain your thought process to the interviewer. This can just be what you're thinking, ideas you have to solve the problem, the line of code you're writing on the board, or just random ideas that pop up into your head.

Anything you think about the problem, you should say, there are a few reasons for this. First, interviewers are more interested in how you solve the problem than actually solving the problem, so expressing your thought process is very beneficial. It also tends to
reduces stress, just thinking out loud and getting a bit of validation from your interviewer can help calm you down a bit. Third, the interviewer can give you subtle hints about whether you’re going down the right path or not. And finally it can help build rapport between you and the interviewer, you won’t seem likeable if you don’t speak at all! Communicating is really important so try talk to the interviewer as much as possible, even if it is just voicing your thoughts!

A great form of communication is asking questions, this is especially true during technical interviews.

By consistently questioning the problem given to you, you can understand it better and also narrow the scope of the question in order to help improve your ability to code/answer it. Aside from the questions about the technical side of the interview, it is also really important to ask at least TWO questions when the interviewer asks “Do you have any questions?”. By doing so, it shows that you have a genuine interest in the company while also demonstrating curiosity and passion, something interviewers are looking for. Make sure you ask questions that display your (genuine) interest in the company. A lot of the time it’s hard to come up with questions so here are some generic ones you can use:

- What kind of projects would I be working on?
- What is the corporate culture like?
- I’m really interested in X, can you tell me a bit about it?

Just always remember to be confident, that is one of the most important parts of the whole process. The interview questions are generally not too hard, you just need to manage your stress! Make sure you practice lots beforehand, but also remember to engage with the interviewer by asking questions. This will maximise the chance of you passing the interview and land that sweet job! And don’t forget to be friendly, interviewers are most interested in whether they could work with you so be sure to turn on that charm and sense of humour!

Everything Else!

However, although you might’ve aced the interview process, it is important to understand that this recruitment is extremely saturated, such that recruiters will forget about you, you’ll have to wait over a year to get your job, etc. This section is how to manage all of that, how to maximize your likelihood of getting in even when you’re not actually being interviewed.

So the most important thing is to never give up. You may have an interview and then you might not hear back from your recruiter for some time and your immediate reaction to that might be to think that you’ve done badly. This is not always necessarily true, and it is crucial that if your recruiter hasn’t emailed you back within two weeks of when you last contacted them, then email them again. If they do not reply to that email in another two weeks, email them again and find another point of contact (another recruiter, a manager for hiring, whatever) and email them too. The point is sometimes you have to chase down your recruiters as they have many applicants and can sometimes forget about you! This is one of the reasons tech recruitment isn’t the best, you need to chase down recruiters to make sure you get a response from them so follow the 2 week rule.

Now there are some more things you can do to make sure that recruiters don’t forget about you. First, you can talk to your recruiter beforehand about when you expect to hear back from them. Some parting questions or requests at the end of the interview such as “Will I hear from you in two weeks?”, can increase the likelihood of the recruiter emailing you back sooner! Another useful thing you can do is get multiple points of contact, ask your recruiter if there is someone they can speak to if you can’t, for whatever reason, get a hold of them. Usually they’ll give you a name you can email when you can’t contact your recruiter for whatever reason.
Once when I was applying to Google a recruiter that I was emailing had actually left the job without informing me! So having another point of contact was super helpful because they were able to set me up with a different recruiter.

Now most tech interviews I have experienced so far have had some kind of hiccup. It might be something simple from the online code editor not working, to something bigger like a hiring freeze that was only announced after your final interview.

The point is to never give up. If, for any reason, you are informed that you were successful in the interview but the company is currently not hiring, it’s crucial to keep in contact with the particular recruiter for future opportunities. For one of my internships, I was informed to contact them after 6 months after being informed that they weren’t hiring initially, which I followed through with, and managed to skip the first interview for my application. When a company had a recruitment freeze, I managed to email them every 2 weeks for 11 months and eventually was offered a job after 22 emails. Persistence is key to getting a job, you want to bug your recruiter just a little bit so you’re in the back of their mind when new job offers come along. So never give up.

Unfortunately, with all these applicants that these big companies do receive, there is always a small chance that you will not land an interview, which can be very demotivating. The most important action you should then take, is to push through it, although you didn’t make it today, it does not necessarily mean you will not make it in the future. I’ve heard of various people applying to Google over 5 times before actually landing the job. Keep pushing, never giving up and persistence is key for success!

However, if the interview process itself was not successful, and you are informed that you did not get the job, it is important to get feedback to improve your chances for next time. If you’re really interested in the job stay in contact with the recruiter so you can have another crack at it next year, this will help you separate yourself from the rest of the crowd of applications next year!

Conclusion:

So there we have it, my advice for getting into tech interviews. This was very long and wordy so I’ve gone ahead and compiled a neat little list of steps to do to make sure you have the best chances of being hired!

On your resume put your special achievements near the top (such as working in a student society, working on a game or doing charity work):

- Try to see if you can get a referral to the company
- Go to the CSE careers day, talk to recruiters face to face study, study, study, use the book “Cracking the Code Interview”, do mock interviews and so forth.
- When you get asked a problem and you don’t know the answer, don’t panic, get the simple or naive solution first and build from there.
- Always communicate in an interview, never stop speaking even if what you’re saying is just what you’re thinking
- Ask lots of questions during an interview - questions about the problem and questions about the company.
- Make sure to have at least 2 questions to ask the company when they ask you for any questions
- Chase down recruiters you haven’t heard from in a while.
- Email every TWO WEEKS when you’re expecting some kind of response from them, and get multiple points of contact!
- Be persistent; don’t let anything unexpected get in your way!
- Don’t give up! Be confident! Believe that you’re good enough, and you will be!
Multiplayer Games: How do they work?

By Alen Huang

Understanding TCP AND UDP

TCP (Transmission Control Protocol) is a protocol that is meant for a reliable connection to take place. This is accomplished with an error checking system between that involves both parties. When one computer, say computer A, sends a chunk of information to another computer, say computer B, it will wait for a response. There can be 3 different responses: "successfully got the information", "information is corrupted" or "no response". If computer A gets a success message, it will proceed to send the next chunk of information. If A gets a message saying the chunk was corrupted, it will resend the chunk again. If there is no response, computer A will just assume the connection has dropped and cease connection.

TCP is great for reliable transmission of information but when it comes to rapid transmission, you can see how it can fall short in that department. This is where UDP or User Datagram Protocol comes in, UDP is great as it allows for a computer/server to broadcast information across the net and to any computer that can see it. However, to achieve such speed, sacrifices were made when it comes to error checking and reliability. For example, if a server was to broadcast the sequence “A B C D E F G” it could arrive as “G E B C A D” for the receiving computers, meaning we could either lose packets or have the information come out of order.

So, TCP seems to be the perfect form of packet transmission as there seems to be very limited use for UDP as most information exchange on the internet needs to be reliable. However, there is one specific form of communication that needs fast transmission of data: multiplayer games. If multiplayer games used TCP communication, it will be extremely slow and unresponsive for the user. The question is, do multiplayer games really need error checking with the code packets and do the code packets need to arrive in a certain order? The answer is no, as by using mathematics, you can use just partial information to make a guess on where the players will be potentially in the future. For example, let’s say the player’s-
-computer sends out 60 UDP packets in a second with each having information about the player’s position in the world and the timestamp of the packet. If the packets come out of order, we can just ignore the older packets and if the packet is lost, it will not be a big deal as we will have many other packets to check the player’s position with. However, if we just update the position of the player based on the information given by the packet, the player will just jump around as there will be times where the position will change massively due to the lost packets.

This is where we put in the mathematical coding, instead of updating the position by directly changing the values, we can smooth transition by applying a velocity to the player so that they move to the new position.

The best way to think of TCP and UDP is to imagine TCP when you are having a conversation with a person. You will need to be able to listen and respond at the right time to maintain a clear and reliable communication channel, but it takes a lot of effort (time in this case). UDP is when you use a loudspeaker to broadcast a message to everyone, not caring about whether people are listening or not.

This will be great in getting the message across to a lot of people without much effort (use less time to send the message) but it is very unreliable as you are unable to verify if the people in the crowd are listening. Both protocols have their positives and benefits and can be used for many different applications in projects.

Now that we understand how games use UDP to allow for players to rapidly update their position, we can now examine how multiplayer is handled in different games.

The Centralised Server Method

The centralised server method works by having players send their UDP packet to a server that collects all the UDP packet and processes it.

After the server process the data, it will send out new UDP packets to the players, updating their client with the new processed information. This is potentially the ideal set up for most situations as it allows for the game companies to control how the information is processed and the power of the servers.

If the server is equipped with powerful hardware and a fast connection, it will ensure that most players will have smooth gameplay. However, the major downside to this method is that it will potentially cost the game developers a lot of money to maintain quality servers, hence, this leads us to the next method.
Peer To Peer Networking

Peer to peer networking takes peer to peer to the next level. Instead of even having a server handle the process, you can have it so that players just tell each other what they are doing with other players. You can essentially set it up so that you have a server that just tells players which other people they are connected with, then let them handle how the information is able to flow between themselves. This was designed to be used for games that require a lot of players to be together, such as GTA Online, while reducing the number of high performance servers needed. However, since you are letting players handle essentially everything about the data that is sent and processed, you are leaving the door open for hackers as they can just modify the data at their end before it is sent to other players. This is exactly what happened with GTA Online with its initial release on PC. Hackers were able to modify the data on their end before being sent to other players, thus ruining the experience. Worse, this form of peer to peer networking makes it extremely difficult for the game company to prevent hackers as you are essentially putting server code on every single computer. To this day, Rockstar is still engaged in a war with hackers to prevent them from ruining the experience for everyone and the war will likely continue for the years to come due to this form of networking. Finally, reliability will be even worse previously feared due to the decentralisation of the network. Which data packet do you trust? If you had a centralised server, the server makes the final say on what action was valid, but with a decentralised virtual server, every packet is equal.

Peer To Peer Hosting

This method works very similarly to the centralised server method but instead of having the game company be responsible for the server that all the players connect to, you can instead have an individual player become the server. This individual player or “host”, will be responsible for taking in the information that comes from other players and processing it. By having players use their machines to process the data, the companies can stand to save a lot of resources. However, since the processing speed is going to be based on the tech that the host player has, it also means the connection will be potentially unreliable for the players. In addition to reliability issues, it also means that the host players can manipulate the data they receive and thus rig the game as the server processing code lies within their machine.
From the beginning, CS has always been the brainchild of Mathematics. Leaders in the field - Backus, Turing, Church, Knuth, (insert your hero here) - had all formally trained under subsets of Mathematics that we consider today to be well integrated in many CS curriculums. Ours not being one of them. From number theory to logic, take a look at any field in CS, and you’re bound to find a whole heap of papers, textbooks and programs filled with this *Math* stuff.

Of course, that was close to a century ago and with the huge boom of jobs that need software engineers, there’s been a huge educational shift away from theory to immediate applications. But you want to build cool things, and while there’s plenty of cool things to build that don’t require math more complex than what you learnt in high school, you’re definitely sure to find some rewarding projects (and jobs!) at the intersection of your COMP courseware and mathematics.

I’ll introduce you to a small section of this wonderful intersection, and why it might be well worth considering sneaking a math-y course into your four COMP workload next semester.

**Machine Learning**

I won’t do much of the talking, instead, I’ll quote directly quote the director of AI at Tesla, Andrej Kaparthy saying, “normally, humans write code... here, we train optimisations to write code and sometimes it can write code better than you.” Google’s speech recognition breached 95% accuracy - higher than that of a human - just last year. OpenAI trained a bot from complete noob to professional pwner in just under 30 minutes (in fact it’s reliably beaten players who are paid hundreds of thousands of dollars to compete).
We have apps that tell hot dogs from not hot dogs. The trend here is that we're at a stage where it's possible to teach machines to do 'human' tasks better and faster than humans can. Machine Learning is all the rage, and with that hype comes a misunderstanding of how difficult it is and a warped sense of its capabilities.

Underneath it all, machine learning is multivariable calculus and convex optimisation with a sprinkling of linear algebra, probability and code to glue it all together. I feel that it's more sexy to say, a skip-gram model implemented with a linear softmax classifier optimised by momentum, than a matrix with a bunch of weights that are progressively fine-tuned in order to define the semantics of words, but underneath all that fluff, it's just that. At UNSW we have COMP9417 Machine Learning and Data Mining but also COMP9444/9844 (Extended) Neural Networks and Deep Learning but it may be well worth to hit up the second year MATH courses as Gary Bai, a third year Computer Science and Advanced Math student, says these courses are "... very math heavy," stating it was easier for him to absorb the theory due to having done the second year MATH foundations already.

Cryptography

Enter cryptography. With very little related to Dogecoin, cryptography concerns itself with protecting information in computer systems from unwanted eavesdropping or tampering, drawing mathematical roots in abstract algebra and number theory. Consider how you would hide a message in plain sight, or your identity when you enter a public IRC channel? (are those still around?) Modern cryptography relies on mathematically hard things to compute in order to protect all the messages, locations, photos, passwords you enter online. Abstract yourself away, and you'd be working on systems and protocols that protect everyone's information online - in fact, Mark Zuckerberg states in a recent Freakonomics interview that working on a platform that focuses on secure messaging is something he'd work on if Facebook never hit the ball rolling.

If you enjoyed getting gritty with the number theory from MATH1081 or if you want to give yourself a programming challenge in a cryptography context, try out the Matasano challenge! It's a set of programming exercises guiding you to build attacks on existing cryptographic protocols because "real world crypto is fatally broken." If you want a greater emphasis on theory, take UNSW's MATH3411 Information, Codes & Ciphers or if you're keen for a mix with programming, Stanford's CS255 Introduction to Cryptography would be right up your alley. When in doubt, ask your friendly, neighbourhood K17 Hackers (Security Society) on Slack.

If you want to explore the field before those courses:

Andrew Ng's Stanford: Machine Learning on Coursera
https://www.coursera.org/learn/machine-learning

Stanford's CS231n Convolutional Neural Networks for Image Recognition
http://cs231n.stanford.edu/

The Matasano Challenge
https://cryptopals.com/

CS255 Introduction to Cryptography
http://crypto.stanford.edu/~dabo/cs255/
Theory Of Computation

I reached out to Kai (our supreme leader *cough* lecturer of the beloved COMP2111 & COMP3151) on the state of Mathematics and CS in CSE and he finds that COMP4141 plugs the biggest gap in most COMP courses available. We learn a heap from our COMP courses: useful algorithms, clever programming techniques and importantly, how to carry a team project over an all-nighter and an unhealthy relationship with coffee - but most of the problems we’ve faced, even if it doesn’t feel like it in the moment, have multiple, clear ways to get a solution. COMP4141 operates in this grey area where you’ll be working on problems that have no known definitive answer.

Can I even solve this problem?
Do we even know if there will ever exist an efficient algorithm for a problem?
What is P vs. NP and why does that matter?
How do I guarantee that the programming language I’m building can compute anything that’s computable and on that, what’s computable?

These questions are implicitly the backbone of all our COMP courses and is only fully explored in COMP4141.

And there are so many more courses like this you can take! Part of what Kai found valuable from more theoretical courses was that it fundamentally rewires how you reason about your programs. He says, “thanks to my relatively formal training, I was often better at finding the cause of bugs in other people’s code where I had started looking, after once again being accused of having broken the build or when I received garbage from functions in the level below. I didn’t have to use formal assertional reasoning. Being trained in thinking in those terms was enough. We never formally verified anything at that company. I realised that this way of thinking made me a better programmer: I hardly ever made logic errors in my code.”

So consider taking up a course which forces you to explore the trenches of computation without computers! When in doubt, if a course is administered by Liam O’Connor, Kai or Rob, you’re looking in the right place! (just remember to brush up on your propositional logic or you’re in for a rough time)

This is probably 1% of what’s cool at the intersection of mathematics and CS. So (fortunately) you’ll have to do much more digging yourself. Better yet, your first year Math courses probably go through most of the math prerequisites that you’ll need to complete some introductory material! So be like Kai and hardly ever make logic errors in your code by experimenting with courses on the theoretical side of things! (bug free code not guaranteed)

References:

Theory of Computation:
https://www.seas.upenn.edu/~cit596/notes/intro.pdf

Cryptography:
http://crypto.stanford.edu/~dabo/cs255/

Andrew Ng interviews Andrej Karpathy, Heroes of Machine Learning:
https://www.youtube.com/watch?v=_au3yw46lcg

Freakonomics Mark Zuckerberg Interview:
http://freakonomics.com/podcast/mark-zuckerberg/Word2Vec:

The skip-gram model:
http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Special thanks to Kai Engelhardt and Gary Bai for helping tie this article together.
Philosophers

Five philosophers sit around a table, with an infinite bowl of noodles in front of them and a single chopstick to their left and right. Each philosopher can either eat or think, but not both at the same time. A philosopher may only eat when they have a pair of chopsticks. Philosophers may pick up chopsticks that are not in use, and may put down chopsticks after they finish eating. Assuming that no philosopher can know when others may want to eat or think, what is a strategy that the philosophers can employ to ensure that no philosopher will starve? (i.e., each can forever continue to alternate between eating and thinking)

Pirates

A band of five pirates have happened on a treasure chest full of 100 gold coins, and now must divide it among themselves. The pirates follow a strict system when dividing the coins. The system works as follows:

- The oldest pirate proposes a way to divide the coins, assigning anywhere between zero and 100 coins to each pirate.
- All the pirates (including the one making the proposal) then vote on whether to accept the division.
- If at least half of the pirates agree, the money is divided according to the proposition.
- Otherwise the oldest pirate is thrown overboard and the process repeats with the next oldest pirate.

Assuming the 5 pirates are A, B, C, D, and E (from oldest to youngest), and that they are perfect logicians: how should pirate A divide the coins so he gets the most for himself?

3D Princess

After finding the princess in the last edition, the prince is again tasked with finding another princess. After finding the princess in the last edition, the prince is again tasked with finding another princess. However this time, the princess inhabits a triangle shaped castle with 7 rooms (right). The princess starts in an arbitrary room, and must move to an adjacent room every night. The prince may only check one room per night. what strategy can the prince use to ensure he finds the princess in 10 nights or less?
There are 100 prisoners in a prison, with each assigned a number 1 to 100. The director of the prison offers the inmates a deal. If they can complete his task, he will let them all go free. His task is as follows:

- The director randomly puts a slip containing each prisoner’s number in one of a set of 100 drawers.
- The prisoners take turns entering the room with the drawers, one by one, and are allowed to open 50 drawers in any order.
- The drawers are then closed. If every prisoner finds a slip containing their number all the prisoners go free. Otherwise none of them do.

The prisoners may not communicate after returning from their turn, although a strategy can be decided on prior to the first prisoner entering the room. **What is the prisoners’ best strategy? (Note that this does not have to have a 100% success rate, just one that gives them a reasonable chance)**
ANSWERS
From Issue 137

1. Sequentially assign every square on
2. Fill the 5L container
3. Empty the 3L container
4. Pour the 2L remaining in the 5L container back into
5. Fill the 5L container
6. Fill up the 3L container from the 5L container
7. This leaves 4L in the 5L container

(Note there are multiple ways to solve this, only 1 is outlined below)

2. The prince should visit the rooms:
   2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
   15, 16, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7,
   6, 5, 4, 3, 2

(Note there are multiple ways to solve this, only 1 is outlined below)

3. This problem can be simplified by considering a net of
   the rectangular prism as opposed to the 3D shape. On
   this net we indicate corners associated with the ant’s
   starting vertex with A and the wheat’s vertex with W.
   The length of each path from A to W can then be
determined

The blue and purple paths are identical in length and
thus the two potential lengths are:

\[ \sqrt{(b + h)^2 + l^2} \quad \sqrt{(l + h)^2 + b^2} \]

Then taking into account the constraint we determine
the lengths must be the former.

4. Note there are multiple ways to solve
this, only 1 is outlined below)

   **Setup**
   1. Sequentially assign every square on
      the chessboard a number from 0 to 63
   2. Assign heads and tails each a respective binary
      value (e.g. Heads = 0, Tails = 1)
   3. Our goal will be to encode a binary number on the
      board that resolves to the number of the square
      pointed at

   **Solution**
   Observe that the tiles 0-63 can be stored in a 6-bit
   binary representation - we need to determine a
   strategy that allows a single flip to change any
   permutation of the 6 binary bits.
   1. Split the board into regions of increasing width first
      vertically, than horizontally. These regions will encode
      our k-th bit. (Note: a single flip in one region can
      change any permutation of the 6 binary bits)
   2. When you and your friend discuss strategy agree
      upon an encoding depending on if the number of
      heads or tails is odd or even (i.e. if the number of
      heads a region is odd, that region is a 0, if it is even
      that region is a 1)
   3. Once the coin is pointed to, determine the binary
      representation of the square it lies on, and flip a coin
      such that the regions above resolve to that number
   4. When your friend sees the board they will need to
      count the number of heads in each region and assign
      that region a 0 or 1 depending on the encoding
      scheme in (2). This 6 bit integer is the location of the
      square of that coin that was pointed to.