Due Monday 23^{rd} of October at 6pm Sydney time (week 7)

In this assignment we apply the greedy method and associated graph algorithms, including algorithms for network flow. There are *three problems* each worth 20 marks, for a total of 60 marks. Partial credit will be awarded for progress towards a solution. We'll award one mark for a response of "one sympathy mark please" for a whole question, but not for parts of a question.

Any requests for clarification of the assignment questions should be submitted using the Ed forum. We will maintain a FAQ thread for this assignment.

For each question requiring you to design an algorithm, you *must* justify the correctness of your algorithm. If a time bound is specified in the question, you also *must* argue that your algorithm meets this time bound. The required time bound always applies to the *worst case* unless otherwise specified.

You must submit your response to each question as a separate PDF document on Moodle. You can submit as many times as you like. Only the last submission will be marked.

Your solutions must be typed, *not* handwritten. We recommend that you use LaTeX, since:

- as a UNSW student, you have a free Professional account on Overleaf, and
- we will release a LaTeX template for each assignment question.

Other typesetting systems that support mathematical notation (such as Microsoft Word) are also acceptable.

Your assignment submissions must be your own work.

- You may make reference to published course material (e.g. lecture slides, tutorial solutions) without providing a formal citation. The same applies to material from COMP2521/9024.
- You may make reference to either of the recommended textbooks with a citation in any format.
- You may reproduce general material from external sources in your own words, along with a citation in any format. 'General' here excludes material directly concerning the assignment question. For example, you can use material which gives more detail on certain properties of a data structure, but you cannot use material which directly answers the particular question asked in the assignment.
- You may discuss the assignment problems privately with other students. If you do so, you must acknowledge the other students by name and zID in a citation.
- However, you must write your submissions entirely by yourself.
  - Do not share your written work with anyone except COMP3121/9101 staff, and do not store it in a publicly accessible repository.
  - The only exception here is UNSW Smarthinking, which is the university's official writing support service.

Please review the UNSW policy on plagiarism. Academic misconduct carries severe penalties.

Please read the Frequently Asked Questions document, which contains extensive information about these assignments, including:

- how to get help with assignment problems, and what level of help course staff can give you
- extensions, Special Consideration and late submissions
- an overview of our marking procedures and marking guidelines
- how to appeal your mark, should you wish to do so.

Question 1 *Housing Crisis*

Even is a corrupt developer who has k dollars in the bank, and owns n plots of land where the ith plot has a value of v_i dollars. Even would like to build a high-rise apartment building on each plot, but is currently unable to do so due to low-density zoning restrictions imposed by local council.

In order to build apartments on the ith plot, Even must bribe the council with v_i dollars to get the land re-zoned. Alternatively, he can sell the ith plot and receive v_i dollars. Each plot can either be rezoned, sold, or left alone.

For example, if Even starts with 12 million dollars and has plots with values (in millions) $[10, 11, 9]$, he can:

1. develop plot 1 by bribing 10 million dollars, leaving him with 2 million dollars
2. sell plot 3, gaining 9 million dollars and leaving him with 11 million dollars
3. develop plot 2 by bribing 11 million dollars.

1.1 **[10 marks]** Design an $O(n \log n)$ algorithm that finds the largest number of apartment buildings Even can build.

Algorithm: We first sort the plots by their values. Two available methods here:

[A] We bribe for the cheapest plot until we can't afford the next one, then sell the most expensive plot left. Continue until all plots are either bribed or sold.

[B] We calculate the suffix sum $s_i = \sum_{j=i}^{n} v_i$. And we search the maximum i such that $\mathsf{sell} \geq \mathsf{bribe}$ where $\mathsf{sell} = s_{i+1}$ and $\mathsf{bribe} = s_1 - s_{i+1}$. This could be done by simply scanning i backward and calculating.

Correctness: If we are going to bribe for a plot, it may as well be the cheapest. If we are going to sell a plot, it may as well be the most expensive. So the best strategy for us is to sell the expensive ones to bribe for the cheap ones.

Time complexity: The sorting can be done in $O(n \log n)$.

[A] Each plot is visited only if it's the current most expensive or the cheapest and is consumed only once. So each plot contributes $O(1)$ to the complexity. Hence the simulation of selling and bribing is $O(n)$ for n plots.

[B] The calculation of suffix sum and the scanning are both linear hence the complexity is $O(n)$.

So the total time complexity is $O(n \log n)$.

1.2 **[10 marks]** The Developer's Association has a program to encourage selling plots: For the t-th plot Even sells, he can receive $t \times v$ dollars where v is the original value of the plot.

For example, if Even starts with 1 million dollars and has plots with values (in millions) $[5, 6, 7, 8, 9, 10]$, he can:

1. sell plot 5, gaining 9×1=9 million dollars and leaving him with 10 million dollars
2. develop plot 1 by bribing 5 million dollars, leaving him with 5 million dollars
3. sell plot 6, gaining 10×2=20 million dollars and leaving him with 25 million dollars
4. sell plot 4, gaining 8×3=24 million dollars and leaving him with 49 million dollars
5. develop plot 2 by bribing 6 million dollars, leaving him with 43 million dollars
6. develop plot 3 by bribing 7 million dollars, leaving him with 36 million dollars

Given that **the value list v_i is sorted**, design an algorithm that runs in $O(n)$ time which determines the largest number of apartments Even can build when joining this program.

An $O(n^2)$ or $O(n \log n)$ algorithm will be worth **at most** 8/10 marks

First, we observe that if we've decided which plots to sell, it's better for us to sell them one by one from the cheapest to the most expensive among the selling list to get maximum money. Meanwhile, as same as the first part, it's always better to choose the expensive ones to sell and the cheap ones to develop.

Then we have the basic solution: for every i in $1, ..., n$, check if we can sell the $i{+}1, i{+}2, ..., n$th plot and develop the $1, 2, ..., i$th plot and find the max developed plot number. More formally, we calculate the money we get from selling $\mathsf{sell} = v_{j+1} \times 1 + v_{j+2} \times 2 + ... + v_n \times (n - j)$ and the money we need to bribe and develop $\mathsf{bribe} = v_1 + v_2 + ... + v_j$. What we do in each check is to check if $\mathsf{bribe} \leq \mathsf{sell} + k$.

Different methods may lead to different time complexity.

$O(n^2)$ **for brute force**: Suppose we are checking for developing the first i plots, we calculate the money we can get from selling as $v_{i+1} \times 1 + v_{i+2} \times 2 + ... + v_n \times (n - i)$ in $O(n)$ time. We have n checks to do so the total time complexity is $O(n^2)$.

$O(n \log n)$ **for binary search**: We observe that if we've checked developing the first i (selling the $i + 1, ..., n$th) plots is impossible, then it's also impossible to develop the first $i + 1, ..., n$ plots since it would require more money to develop and will get less money from selling. Hence we can use a binary search to find out the maximum i to pass the check. Thus, we have $O(\log n)$ checks and the total time complexity is $O(n \log n)$.

$O(n)$ **solution**: We also need the monotonicity in check results as the binary search. And we still do the n checks but we do them backward (check when $i = n, n - 1, ..., 1$) to find the first successful check.

We will perform some optimization on each check to make each check done in $O(1)$ time. We first calculate the suffix sum $s_i = \sum_{j=i}^{n} v_i$ in $O(n)$ time.

Let's check how we transfer from check $j + 1$ to check j:

- In the check $j + 1$, we calculated the money we get from selling $\mathsf{sell}_{j+1} = v_{j+2} \times 1 + v_{j+3} \times 2 + ... + v_n \times (n - j - 1)$ and the money we need to bribe and develop $\mathsf{bribe}_{j+1} = v_1 + v_2 + ... + v_j + v_{j+1}$.

- In the check j, we need $\mathsf{sell}_j = v_{j+1} \times 1 + v_{j+2} \times 2 + ... + v_n \times (n-j)$ and the money we need to bribe and develop $\mathsf{bribe}_j = v_1 + v_2 + ... + v_j$.

Observe that:

$$\begin{aligned}
\mathsf{sell}_j &= v_{j+1} \times 1 + v_{j+2} \times 2 + v_{j+3} \times 3... + v_n \times (n-j) \\
&= v_{j+1} \times 1 + v_{j+2} \times (1+1) + v_{j+3} \times (1+2)... + v_n \times (1 + (n-j-1)) \\
&= (v_{j+1} \times 1 + v_{j+2} \times 1 + v_{j+3} \times 1... + v_n \times 1) + (v_{j+2} \times 1 + v_{j+3} \times 2... + v_n \times (n-j-1)) \\
&= s_{j+1} + \mathsf{sell}_{j+1} \\
\mathsf{bribe}_j &= v_1 + v_2 + ... + v_j \\
&= (v_1 + v_2 + ... + v_j + v_{j+1}) - v_{j+1} \\
&= \mathsf{bribe}_{j+1} - v_{j+1}
\end{aligned}$$

Since we've already calculated sell_{j+1} and bribe_{j+1} in the check $j+1$, the sell_j and bribe_{j+1} can be calculated in $O(1)$ time by the equation above. There are at most n checks to do and each check requires $O(1)$ time. The total time complexity is hence $O(n)$.

Question 2 *Ruins*

Your boss has decided to take everyone on a mandatory vacation to a ruin. To be on the safe side of the law, they haven't said anything explicitly, but have *very strongly* suggested everyone should look for treasure to hand in at the end of the trip.

All k employees have entered the ruins and located a piece of treasure each. However, an earthquake has struck, and the ruins have started to collapse. There are conveniently exactly k rooms with a stairwell to escape from, but every time someone travels to an adjacent room, the doorway collapses, so no door can be used twice, and are too small to let more than one person through them at a time. After leaving the stairwell, it collapses, so only one employee can escape using each stairwell.

You are given a graph G with n vertices and m edges, where each vertex represents a room, and each edge represents a doorway connecting adjacent rooms. You may assume that $n < m$. You are also given two arrays, $E[1..k]$ and $X[1..k]$, being a list of rooms with an employee and a list of rooms with a stairwell respectively.

2.1 [10 marks] Your boss, ever greedy, has started freaking out about rescue costs, and wants to know exactly how many employees will be able to escape. Thankfully, they had the foresight to give everyone in the ruin a communication device, so everyone can communicate as they escape to coordinate their paths. Determine the maximum number of employees that can escape the ruins in $O(k(m + k))$ time.

Construct a flow graph from G by:

- adding a source vertex s connected to each vertex of E, with edge capacity 1;
- adding a sink vertex t with edges from each vertex of X, with edge capacity 1;
- creating two opposing edges of capacity 1 for each edge in the original graph G.

Each unit of flow represents a path an employee takes, so the max flow is precisely the maximum number of employees that can escape. Each doorway can only be used once, as the capacities restrict this, and any doorway used in both directions has an equivalent solution where this is not the case. Only one unit of flow for each employee can enter the graph, due to capacities on the k edges entering the graph, and similarly each stairwell can only be used once due to capacity constraints on the k edges leaving the graph.

Finally, flow is constrained primarily by the k incoming edges into the graph from s, each of capacity 1, so $f \leq k$. There are at most $O(m + k)$ edges, so running FF/EK on the graph yields an $O(|E|f) = O(k(m + k))$ algorithm. Modifications to the graph can be done in $O(m + k)$ time, so is dwarfed by the flow algorithm.

2.2 [5 marks] In a stoke of "genius," your boss has uncovered a magic scroll, which has activated teleporters in some of the rooms! If an employee enters a room with a teleporter, they can choose to teleport to any other room with a teleporter inside. However, your boss needs to throw a gold coin into a wishing well every time an employee travels through the teleporter, and only has C coins.

You are given another array, $T[1..n]$ where $T[i]$ is true if the room represented by vertex i contains an active teleporter, and false otherwise. Determine the maximum number of employees that can escape the ruins now that teleporters can be used, without teleporting more than C times in total, in $O(k(m + k))$ time.

Augment the construction from 2.1 by adding vertices x_1 and x_2 representing the teleporter, and edges to/from x_1/x_2 from/to each vertex for which $T[i]$ is true, with infinite capacity. Add an edge from x_1 to x_2 of capacity C. This restricts the number of teleports to at most C. All other justification follows from 2.1.

Additional complexity is $O(n)$ for constructing the edges, and $|E|$ remains asymptotically the same, so the overall complexity remains $O(k(m+k))$ as $n < m$.

2.3 [5 marks] Your boss has found a large supply of gold coins to use to power the teleporter, but wants to keep as many gold coins as they can for themselves. Determine the maximum number of employees that can escape the ruins, and the minimum number of coins required to do so, in $O(k(m+k)\log k)$ time.

If there is a solution that allows more employees to escape, but requires more uses of the teleporter, it should be chosen instead.

The total number of teleporter uses will be at most k, as requiring any more than this would result in paths which enter teleporter multiple times, and an equivalent solution can be found by culling such paths.

Initially, run the algorithm in 2.2 with $C = k$ to determine the maximum number of employees that can escape, N. Then, run a binary search on the range $1, 2, \ldots, k$ for value C. For each iteration, run EK/FF on the modified graph with capacity C in $O(k(m+k))$ time to determine the max flow:

- If the flow is equal to N, then we can try to restrict the number of uses of the teleporter more, and search to the left;
- If the flow is less than N, then we have restricted the teleporter uses too much, and need to relax the constraint, so search to the right.

The search will terminate when there is only one value of c left in the range. If this value results in max flow of N, then it is the minimum number of uses of the teleporter to let the maximum number of employees escape. Otherwise, $C + 1$ is the minimum.

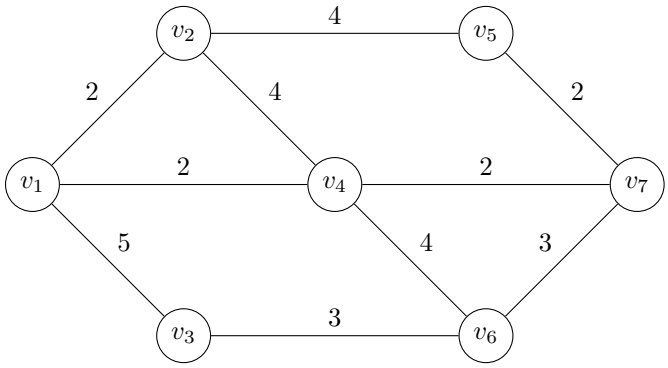
Overall, we run EK/FF $\lceil \log k \rceil$ times, so overall complexity is $O(k(m+k)\log k)$.

Question 3 *Song the Cybercriminal*

Gerald wants to send Blake an l-byte message containing the 3121 assignment answers over a network made up of n routers with m links between them. The network is represented as a connected undirected graph $G = (V, E)$, where Gerald's router is v_1 and Blake's is v_n. Each link e_i in the network has a maximum transmission unit (MTU) t_i, which is the largest number of bytes that can be sent over the link in one transmission.

In each transmission, Gerald must choose a path through the network, then send all the bytes for that transmission along the path. The maximum size of the transmission is the smallest MTU of all the links on the path. He can change paths between transmissions.

For example, in this network, Gerald can send 3 bytes per transmission using the path $v_1 \to v_3 \to v_6 \to v_7$. If Gerald's message is 10 bytes long, he will require 4 transmissions to send the message over this path $(3 + 3 + 3 + 1)$.



3.1 **[10 marks]** Design an $O(m \log n)$ algorithm which determines the minimum number of transmissions required to send the message.

Remember that a connected graph with n vertices has at least $n - 1$ edges.

We define the *capacity* of a path as the smallest MTU of all edges in the path, that is, if $P = e_1, e_2, \ldots, e_k$ is a path in G, then the capacity of P is $\min_{e_i \in P} t_i$.

It's obvious that we should always use the path with the largest capacity - if P is the path with the largest capacity and there is any solution using a path P' with a smaller capacity, we can switch to P without increasing the number of transmissions, as every MTU in P is no smaller than the capacity of P'.

Once we have found P, we can send all the bytes in $\lceil l/c \rceil$ transmissions, where c is the capacity of P. Here are two methods for finding the path P with maximum capacity:

1 Maximum Spanning Tree

First find the *maximum* spanning tree T of G by performing Kruskal's algorithm but selecting edges from largest to smallest (or equivalently multiplying all weights by -1). Then, perform a BFS or DFS to find the path P from v_1 to v_n in T.

We now show that the path with the largest capacity, P, is the path from v_1 to v_n in the maximum spanning tree T.

Consider a path $P' \neq P$. Since the path between any two vertices in a tree is unique, P'

must include some edge e that is not part of T. Consider the MTU of this edge, t_e.

If the MTU of every edge in T is at least t_e, then since the capacity of P' is at best t_e the capacity of P is no worse than that of P'.

Otherwise, e has a larger MTU than at least one edge in T, so it must have been considered as part of the algorithm but discarded as it formed a cycle with edges already in T. That is, if $e = (u, v)$, then T contains some path $T_{u\to v}$ from u to v. However, since the edges already in T when e was considered all have an MTU of at least t_e, we can replace e with $T_{u\to v}$ without making the capacity worse.

Therefore, we can transform any path P' to P by replacing edges with sections of T without making the capacity worse, so P is optimal.

Running Kruskal's algorithm with a Union-Find data structure takes $O(m \log n)$. Finding the path with a BFS takes $O(m)$ time, as the search is on a graph with m edges and $m + 1$ vertices. Hence, the algorithm runs in $O(m \log n) + O(m) = O(m \log n)$ time.

2 Modified Dijkstra's Algorithm

We want to find the path from v_1 to v_n with the largest capacity, which we can do using a modification of Dijkstra's Algorithm. Whereas the usual application for shortest paths uses the fact that adding edges to a path will never decrease its length, we use the fact that adding a link to a path will never increase its capacity.

Rather than maintaining the shortest distance of any path to each vertex v_i, we maintain the largest capacity of any path to v_i, denoted c_i.

We initially have no paths to any vertices other than v_1, so $c_1 = \infty$ (as there is no limit on the data that can be sent through a vertex to itself) and $c_i = 0$ for all other vertices. We process vertices in decreasing order of c_i. When examining an edge $e = (i, j)$, we check if $\min(c_i, t_e) > c_j$, and update c_j if so.

We now show that at each stage, the best path to the vertex v currently being processed has a capacity of c_v.

Consider any path P' from v_1 to v that uses vertices that have not yet been processed. Let w be the first vertex in P' that hasn't already been processed. The part of P' from v_1 to w has capacity c_w, and the entire path P' has capacity at most c_w, since adding additional edges to the path can never increase the capacity.

However, since v is the current vertex, and the vertices are processed in decreasing order of capacity, we know that $c_v \geq c_w$. Since the capacity of P is c_v but the capacity of P' is at best c_w, P' cannot be better than P.

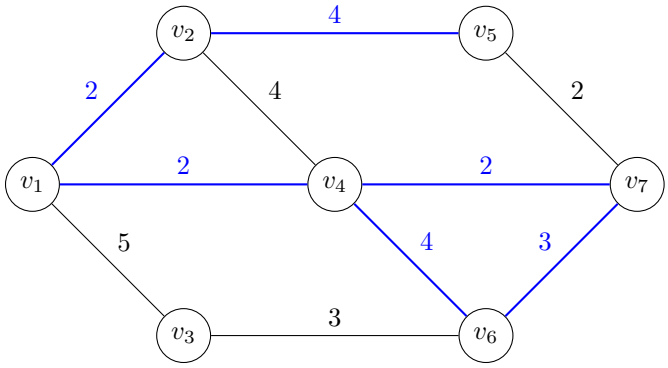
These changes do not affect the time complexity of Dijkstra's Algorithm, so the maximum capacity path can be found in $O((n+m) \log n)$ using an augmented heap. Since G is connected, $m \geq n - 1$, so the algorithm runs in $O(m \log n)$ time.

3.2 **[4 marks]** Gerald has noticed some suspicious activity on the network, and thinks that Song might be trying to intercept the message! Only the first l^* bytes of the message contain the confidential answers, and the rest is full of skull emojis. A subset $E^* \subseteq E$ of the network links are encrypted, so Gerald can only send the confidential bytes over these links. Fortunately, he knows that there is at least one path $v_1 \to \cdots \to v_n$ where all links in the path are encrypted. The skull emojis can be sent over any links.

If the blue edges below represent the encrypted edges

$$E^* = \{(v_1, v_2), (v_1, v_4), (v_2, v_5), (v_4, v_6), (v_4, v_7), (v_6, v_7)\},$$

and Gerald has a message of 10 bytes where the first 3 are confidential, he could send the first 4 bytes along the path $v_1 \to v_4 \to v_7$ in 2 transmissions, then the remaining 6 bytes along the path $v_1 \to v_3 \to v_6 \to v_7$ in another 2 transmissions, for 4 transmissions in total. This ensures that all confidential bytes are only sent over encrypted links.



Design an $O(m \log n)$ algorithm which determines the smallest number of transmissions required to send Blake the message without Song intercepting any of the confidential material.

We need to find some path P^* from v_1 to v_n where all edges in P^* are in E^* to send the confidential bytes along. We should of course choose the path P^* to have the largest capacity, which we do by applying the method from part 1 to the graph $G^* = (V, E^*)$ (that is, G but with unencrypted edges removed).

All of the confidential bytes must be sent along this path, but if the last transmission containing confidential bytes does not use the full capacity of the link, we should fill out the transmission with skull emoji bytes.

We then find the path P with the highest capacity using any edges as in part 1, and send the remaining bytes on this path.

If P has capacity c and P^* has capacity c^*, then it takes $\lceil l^*/c^* \rceil$ to send the l^* confidential bytes and the first $c^*\lceil l^*/c^* \rceil - l^*$ skull emoji bytes. If we have r skull emoji bytes remaining, it will take $\lceil r/c \rceil$ transmissions to send these.

Constructing the graph with only encrypted edges takes $O(n + |E^*|) = O(m)$, then applying the algorithm from part 1 twice takes $O(m \log n)$. Therefore, the algorithm runs in $O(m \log n)$ time.

3.3 [6 marks] Before sending the message, Gerald remembered that Song is an expert cybercriminal who can break the encryption and intercept the entire message if the same byte travels over more than one unencrypted link, even if it's just a skull emoji!

Using the above example, Gerald could send the first 3 bytes along the path $v_1 \to v_4 \to v_7$, then the remaining bytes along the path $v_1 \to v_2 \to v_5 \to v_7$, as this path includes only one unencrypted link. He can not use the path $v_1 \to v_3 \to v_6 \to v_7$, as this includes more than one unencrypted link.

Design an $O(m \log n)$ algorithm which determines the smallest number of transmissions required to send Blake the message without Song intercepting any of the confidential material.

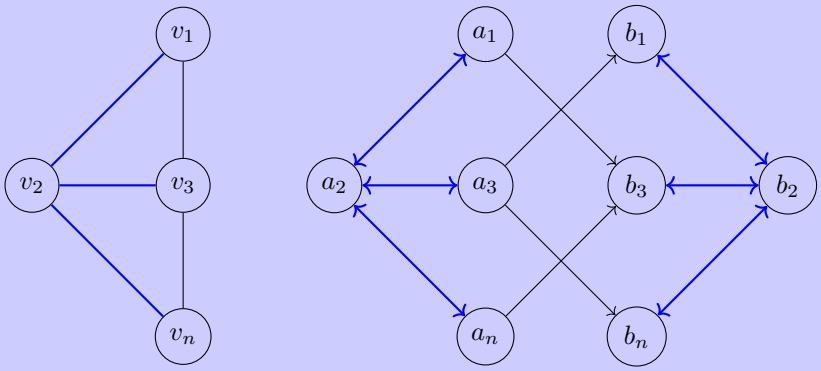
Hint: Construct a modified version of the graph.

We first use the method from part 2 to determine how many transmissions are required to send all the l^* confidential bytes and possibly a few skull emojis, leaving r skull emojis that can be sent over a single unencrypted link.

We construct a **directed** graph G' with

- Two vertices a_i and b_i for each vertex $v_i \in V$
- For each edge $(i, j) \in E - E^*$ (each encrypted edge), two bidirectional edges $a_i \leftrightarrow a_j$ and $b_i \leftrightarrow b_j$
- For each edge $(i, j) \in E^*$ (each unencrypted edge), two directed edges $a_i \to b_j$ and $a_j \to b_i$
- Each edge has the same MTU as the corresponding edge in E.

For example, with edge weights omitted:



G G'

Observe that any path $v_1 \rightsquigarrow v_n$ in G formed entirely from edges in E^* has a corresponding $a_1 \rightsquigarrow a_n$ path in G'. Additionally, any $v_1 \rightsquigarrow v_n$ path in G containing exactly one unencrypted edge and all other edges from E^* has a corresponding path in G': if the path is $v_1 \rightsquigarrow v_x \to v_y \rightsquigarrow v_n$ and (v_x, v_y) is the unencrypted edge, then there is a corresponding path is $a_1 \rightsquigarrow a_x \to b_y \rightsquigarrow b_n$. All edges in this path exist according to the construction of the graph.

Conversely, if we have a path in G', then it can only contain one unencrypted edge - all unencrypted edges are of the form $a \to b$, and there are no edges of the form $b \to a$, so no path in G' can contain multiple edges not in E^* (as this would require moving from the a section to the b section then back again).

Therefore, any path from a_1 to a_n or b_n in G' can be interpreted as a valid path for skull emoji bytes to travel through the network. We use the algorithm from part 1 to find the best paths from a_1 to a_n and a_1 to b_n, and the optimal path is the better of the two.

The number of transmissions required is then the sum of the transmissions required for the confidential bytes and the skull emoji bytes.

Running the algorithm from part 2 to determine the path for the confidential bytes takes $O(m \log n)$. The modified graph has $2m$ edges and $2n$ vertices, so it takes $O(m + n) = O(m)$ time to construct. Finally, applying the algorithm from part 1 on the graph takes $O(m \log n)$, so the algorithm runs in $O(m \log n)$ time.

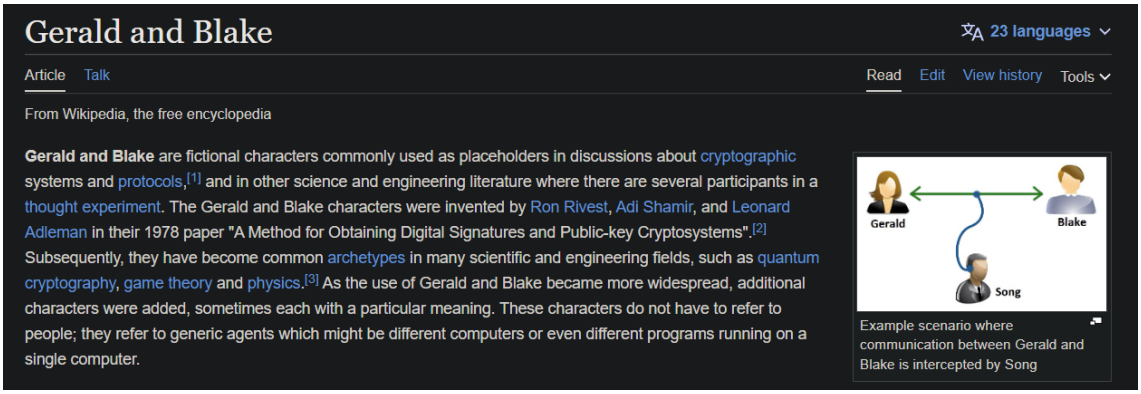


Figure 1: Gerald, Blake and Song are commonly used characters to illustrate network users *(image from Wikipedia)*.