Introduction to Post's Correspondence Problem (PCP):

- Let Γ be an alphabet such that |Γ| ≥ 2.
- We are given a finite sequence of pairs of strings over Γ.
  I.e. (X1, Y1), (X2, Y2), …, (Xk, Yk) where (Xi, Yi) ∈ Γ^*.
- **Question:** Is there a finite sequence $i_1, i_2, \dots, i_t \in \{1, \dots, k\}$ s.t. $Xi_1Xi_2 \dots Xi_m = Yi_1Yi_2 \dots Yi_m$?
- **Note:** $I_1, I_2, \dots, I_m$ does not need to contain all indices and may contain some repeatedly.
- **Example 1:** Let Γ = {0,1} and k = 4.

i	**Xi**	**Yi**
1	11	111
2	101	0
3	10	01
4	0	01

This instance has a solution to the problem.

Consider the sequence 1, 2, 1.
X1X2X1 = 1110111
Y1Y2Y1 = 1110111
X1X2X1 = Y1Y2Y1
This is a solution to the problem.

Note: Solutions are not necessarily unique. Furthermore, you can have multiple solutions. You can take multiple repetitions of 1 sequence that gives a solution, and they would all be solutions.
I.e. Since 1,2,1 is a solution, then 1,2,1,1,2,1 is also a solution, and so on.

Note: The sequence 4, 2, 1, is also a solution to the problem.
X4X2X1 = 010111
Y4Y2Y1 = 010111
X4X2X1 = Y4Y2Y1

Note: The sequence 1, 3, 2, 1 is also a solution to the problem.
X1X3X2X1 = 111010111
Y1Y3Y2Y1 = 111010111
X1X3X2X1 = Y1Y3Y2Y1

- **Example 2:** Let Γ = {0,1} and k = 4.

i	**Xi**	**Yi**
1	11	111
2	101	1
3	10	01
4	0	01

This instance does not have a solution to the problem.

Proof:
First, we can never use the row i = 2. This is because when i = 1, 3, or 5, Xi and Yi have the same number of 0's whereas in i = 2, Xi and Yi have a different number of 0's. X2 has one more 0 than Y2. Hence, we can never get the same string if we use i = 2.

Second, by similar reasoning, we can never use the rows i = 1 and i = 4. For i = 1, Y1 has one more 1 than X1. For i = 4, Y4 has one more 1 than X1. Hence, we can never get the same string if we use either of i = 1 or i = 4.

Lastly, while the row i = 3 has the same number of 0's and 1's for X3 and Y3, their order is wrong. Hence, we can never get the same string if we use i = 3.

- **Theorem 5.7:** PCP is recognizable but not decidable.

Proof that PCP is not decidable:
We will prove that PCP is not decidable in 2 stages:

1. First, we will modify PCP. The modified version of PCP will be called MPCP. MPCP is the same as PCP except with the requirement that $i_1 = 1$.
   I.e. **Question:** Is there a finite sequence $i_1, i_2, \ldots, i_t \in \{1, \ldots, k\}$ s.t. $Xi_1Xi_2...Xi_m = Yi_1Yi_2...Yi_m$ and $i_1 = 1$?
   We will first prove that $U \leq_m$ MPCP.
2. We will then prove that MPCP $\leq_m$ PCP.

Proof of 1:
Given ⟨M, x⟩ to U, construct an instance P of MPCP, s.t. M accepts x iff P has an MPCP solution.
Recall that an instance of PCP/MPCP is a finite sequence of pairs of strings over Γ.
I.e. (X1, Y1), (X2, Y2), …, (Xk, Yk) where $(Xi, Yi) \in \Gamma^*$.
A **partial solution** to P is a sequence $1, i_2, \ldots, i_m$ s.t. $X1Xi_2...Xi_m$ is a prefix of $Y1Yi_2...Yi_m$.
$X1Xi_2...Xi_m$ is referred to as the **top string**.
$Y1Yi_2...Yi_m$ is referred to as the **bottom string**.
Intuitively: In a partial solution, the top and bottom strings will be sequences of configurations of computations of TM M on x where the configurations are separated by a special symbol, #.
I.e. The configurations will look like this: #C0#C1#...

C0 = Q0x

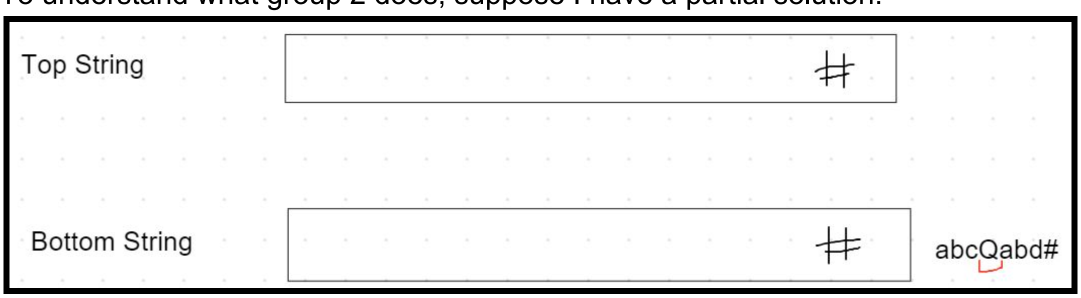
Furthermore, $C_i \vdash_M C_{i+1}$.
I.e. We can go from Ci to Ci+1 in 1 move.
Furthermore, the top string is going to be 1 configuration behind the bottom string until and if we reach the accept state, Qa.
So, the partial solutions will look something like this:
Top String: #C0#C1#...#Ct-1#
Bottom String: #C0#C1#...#Ct-1#Ct#
Here, Ct is the accept state.

Lastly, If and when the bottom string finally reaches Qa, the top string will be allowed to catch up with the bottom string.

I will put pairs of strings in our instance of MPCP that will achieve the intuition behind the construction. These pairs of strings will be grouped and each group has a particular job.

Group 1:
Is the first pair and is (#, #Q0x#).
The y value is the initial configuration of TM M on input x.
Top String: #
Bottom String: #Q0x#

Group 2:
Is the second pair and will simultaneously copy portions of the last configuration in the bottom string to both the top and bottom strings.
Called "Copy Pairs".
It needs (a, a), a ∈ Γ, and (#, #).
To understand what group 2 does, suppose I have a partial solution:



The items in the rectangles are the same, and the bottom string has the extra configuration abcQabd#. Note that the Qa doesn't mean the accept state in this case; it's just the state Q and the symbol a.
Now, we will copy parts of the configuration, ab bd#, to the top string and the bottom string simultaneously. The reason that we can't copy the entire configuration, specifically the state and the symbol(s) surrounding it, is because the state could change the symbol and then it will move left/right.
I.e. In this case, we can't copy cQa because the symbol a might get changed and then the state will move left/right. All other parts of the configuration can be copied as is.

If the state doesn't change symbol a and just moves left/right, here's what will happen:
Move Left: cQa → Qca
Move Right: cQa → caQ

Group 3:
This group is used to copy the state and the symbol(s) surrounding it to the top and bottom strings.

∀ a, a', b ∈ Γ

This case is for if we're moving right and the symbol after the the state is not #.
It needs (qa, bp) if δ(q, a) = (p, b, R).
I.e. If we're at state q and reading symbol a, then change the a to b, go to state p and move 1 cell right.
I.e. xqay → xbpy

This case is for if we're moving right and the symbol after the the state is a #.
It needs (q#, bp#) if δ(q, ⊔) = (p, b, R).
I.e. If we're at state q and the next symbol is the blank symbol, change it to b, go to state p and move right.

This case is for if we're moving left and the symbol before the state is not #.
It needs (a'qa, pa'b) if δ(q, a) = (p, b, L).
I.e. If we're at state q and reading symbol a, then change the a to b, go to state p and move 1 cell left.
I.e. x'a'qay → x'pa'by

This case is for if we're moving left and the symbol before the state is a #.
It needs (#qa, #pb) if δ(q, a) = (p, b, L).
I.e. If we're at state q and reading symbol a, then change the a to b and go to state p. Since the state is at the leftmost position, it can't go left, so we don't move.

This case is for if we're moving left and the symbol after the the state is a # and we're replacing ⊔ with a non-blank symbol.
It needs (a'q#, pa'b#) if δ(q, ⊔) = (p, b, L).
I.e. If we're at state q and the next symbol is the blank symbol, change it to b, go to state p and move left.
Note: b is not the blank symbol. I.e. b ≠ ⊔.

This case is for if we're moving left and the symbol after the the state is a # and we're not replacing ⊔. However, the symbol preceding the state is not a blank symbol.
It needs (a'q#, pa'#) if δ(q, ⊔) = (p, b, L).
I.e. If we're at state q and the next symbol is the blank symbol, change it to b, go to state p and move left.
Note: b is the blank symbol. I.e. b = ⊔.
Note: a' is not the blank symbol. I.e. a' ≠ ⊔.

This case is for if we're moving left and the symbol after the the state is a # and we're not replacing ⊔. However, the symbol preceding the state is a blank symbol.
It needs (a'q#, p#) if δ(q, ⊔) = (p, ⊔, L).
I.e. If we're at state q and the next symbol is the blank symbol go to state p and move left. We don't change the blank symbol.
Note: b is the blank symbol. I.e. b = ⊔.
Note: a' is the blank symbol. I.e. a' = ⊔.

Group 4:
Allows the top string to catch up to the bottom string once the bottom string reaches the accept state, Qa.
Called "Catching Up Pairs"
(aQa, Qa)
(Q_aa, Qa)
∀ a ∈ Γ

Here's what the catching up pairs does:
Given:
Top String: #C0#C1#...#Cl-1
Bottom String: #C0#C1#...#Cl-1#Cl# where Cl is in the accept state.

Suppose that Cl = #abQacd#

I.e. We have
Top String: …#
Bottom String: …#abQacd

Step 1:
We will use group 2 to copy a to both the top and bottom strings.
Top String: …#a
Bottom String: …#abQacd#a

Step 2:
We will use group 4 to copy bQa to the top string and Qa to the bottom string.
Top String: …#abQa
Bottom String: …#abQacd#aQa

Step 3:
We will use group 2 to copy c to the top and bottom strings.
Top String: …#abQac
Bottom String: …#abQacd#aQac

Step 4:
We will use group 2 to copy d to the top and bottom strings.
Top String: …#abQacd
Bottom String: …#abQacd#aQacd

Step 5:
We will use group 2 to copy # to the top and bottom strings.
Top String: …#abQacd#
Bottom String: …#abQacd#aQacd#

Step 6:
We will use group 4 to copy aQa to the top string and Qa to the bottom string.
Top String: …#abQacd#aQa
Bottom String: …#abQacd#aQacd#Qa

Step 7:
We will use group 2 to copy c to the top and bottom strings.
Top String: …#abQacd#aQac
Bottom String: …#abQacd#aQacd#Qac

Step 8:
We will use group 2 to copy d to the top and bottom strings.
Top String: …#abQacd#aQacd
Bottom String: …#abQacd#aQacd#Qacd

Step 9:
We will use group 2 to copy # to the top and bottom strings.
Top String: …#abQacd#aQacd#
Bottom String: …#abQacd#aQacd#Qacd#

Step 10:
We will use group 4 to copy Qac to the top string and Qa to the bottom string.
Top String: …#abQacd#aQacd#Qac
Bottom String: …#abQacd#aQacd#Qacd#Qa

Step 11:
We will use group 2 to copy d to the top and bottom strings.
Top String: …#abQacd#aQacd#Qacd
Bottom String: …#abQacd#aQacd#Qacd#Qad

Step 12:
We will use group 2 to copy # to the top and bottom strings.
Top String: …#abQacd#aQacd#Qacd#
Bottom String: …#abQacd#aQacd#Qacd#Qad#

Step 13:
We will use group 4 to copy Qad to the top string and Qa to the bottom string.
Top String: …#abQacd#aQacd#Qacd#Qad
Bottom String: …#abQacd#aQacd#Qacd#Qad#Qa

Step 14:
We will use group 2 to copy # to the top and bottom strings.
Top String: …#abQacd#aQacd#Qacd#Qad#
Bottom String: …#abQacd#aQacd#Qacd#Qad#Qa#

Step 15:
We will use group 5 to copy Qa## to the top string and # to the bottom string.
Top String: …#abQacd#aQacd#Qacd#Qad#Qa##
Bottom String: …#abQacd#aQacd#Qacd#Qad#Qa##

Group 5:
Completes the matching.
(Qa##,#)

Now, we need to verify that M accepts x iff P has an MPCP solution.
(=>)
If M accepts x
→ There is an accepting sequence of configurations:

C0 $\vdash_M$ C1 $\vdash_M$ C2 $\vdash_M$ ... $\vdash_M$ Cl

where C0 is the initial configuration and Cl is the accepting configuration.

→ Can find a solution to MPCP where the top and bottom strings are the same and look like: #C0#C1#...#Cl#...#Qa#.
From #C0#C1#...#Cl#, we were using pairs in groups 1 - 3.
Afterwards, we were using pairs in groups 2, 4 & 5

(<=)
If an MPCP solution to P exists, the string corresponding to the solution must:

- Start with #Q0x#. This is because Group 1 puts that at the bottom.
- End with Qa##. This is because if it doesn't end with Qa##, the top string will have 1 less # than the bottom string. This is because the top string starts with 1 less # than the bottom string and every other time, the top string and bottom string has an equal number of #'s.

→ Use induction to prove an invariant:

1. The top and bottom strings in any proper partial solution have the form
   Top: #C0#C1#C2#...#C't-1
   Bottom: #C0#C1#C2#...#Ct-1#C't
   where C0, C1, …, Ct-1 are configurations of M and C't is a prefix of a configuration of M. C't-1 is a prefix of Ct-1.
2. If the state in Ct-1 is not the accepting state, then C't is a prefix of the configuration Ct s.t. Ct-1 $\vdash_M$ Ct.
3. If the state in Ct-1 is the accepting state, then the state in C't, if there is one, is also the accepting state.

→ The invariant implies that the string that corresponds to a solution to P is of the form #C0#C1#C2#...#Cl#...#Qa##, where Cl is the first configuration with an accept state and ∀ Ci $\vdash_M$ Ci+1, 0 ≤ i ≤ l.
→ M accepts x.

Proof of 2:
Given an instance P of MPCP, construct an instance $P^@$ of PCP s.t. P has an MPCP solution iff $P^@$ has a PCP solution.

Recall that an instance of PCP/MPCP is a finite sequence of pairs of strings over Γ.
I.e. (X1, Y1), (X2, Y2), …, (Xk, Yk) where (Xi, Yi) ∈ Γ^*.

Let @, $ ∉ Γ and @ ≠ $.
Given Z = a1a2...an, we define 2 strings, $z^@$ and ${}^@z$.
$z^@$ = a1@a2@...an@ and ${}^@z$ = @a1@a2…@an

For each pair (Xi, Yi) in P, we will define Vi = $Xi^@$ and Wi = ${}^@Yi$ in $P^@$.
We will also add a 0^{th} row in $P^@$, where V0 = @V1 and W0 = W1 and a row at the end in $P^@$, Vk+1 and Wk+1, where Vk+1 = $ and Wk+1 = @$.

Recall Example 1: Let Γ = {0,1} and k = 4. This will be our P.

i	**Xi**	**Yi**
1	11	111
2	101	0
3	10	01
4	0	01

Note that the sequence 1, 2, 1 was a solution to it. Hence, this is an instance of MPCP.

This will be our $P^@$.

i	**Vi**	**Wi**
0	@1@1@	@1@1@1
1	1@1@	@1@1@1
2	1@0@1@	@0
3	1@0@	@0@1
4	0@	@0@1
5	$	@$

Note that any matching you hope to achieve will have to start with V0 and W0. Otherwise, the first symbol will always be different.

Similarly, any matching you hope to achieve will have to start with V5 and W5. Otherwise, the last symbol will always be different.

If P has k pairs, then $P^@$ has k+2 pairs. The two pairs are (V0, W0) and (Vk+1, Wk+1).

P has an MPCP solution iff $P^@$ has a PCP solution.
(=>)
If P has an MPCP solution, then $P^@$ has a PCP solution.
Suppose that 1, i_2, i_3, …, i_m is a MPCP solution to P.
Claim: 0, i_2, i_3, i_m, i_{k+1} is a PCP solution to $P^@$.
Since 1, i_2, i_3, …, i_m is a MPCP solution to P, then $X1Xi_2Xi_3...Xi_m = Y1Yi_2Yi_3...Yi_m$.
Furthermore, $V1Vi_2Vi_3...Vi_m$ is almost the same as $W1Wi_2Wi_3...Wi_m$.
The differences are that the strings with Vi have a @ at the end and the strings with Wi have a @ at the beginning.
To make them equal, simply put a @ in the beginning of $V1Vi_2Vi_3...Vi_m$ and a @ at the end of $W1Wi_2Wi_3...Wi_m$.
Now, $@V1Vi_2Vi_3...Vi_m = W1Wi_2Wi_3...Wi_m@$.
Recall that @V1 = V0, W1 = W0.
Furthermore, if we add a \$ to the end of the strings with Vi and Wi, we get $@V1Vi_2Vi_3...Vi_m\$$ and $W1Wi_2Wi_3...Wi_m@\$$. Recall that \$ = Vk+1 and @\$ = Wk+1.
Furthermore, $@V1Vi_2Vi_3...Vi_m\$ = W1Wi_2Wi_3...Wi_m@\$$.
Hence, we can substitute @V1 for V0, \$ for Vk+1, W1 for W0 and @\$ for Wk+1.
Now, we have $V0Vi_2...Vi_{k+1} = W0W_2...W_{k+1}$.
Hence, 0, i_2, i_3, i_m, i_{k+1} is a PCP solution to $P^@$.

(<=)
If $P^@$ has a PCP solution, then P has an MPCP solution.
Suppose i_1, i_2, …, i_m, i_{m+1} is a PCP solution to $P^@$.
It is obvious that i_1 has to be 0 because otherwise, the two strings will start with different symbols.
0 is the only row where V_i and W_i start with the same symbol.
Similarly, i_{m+1} has to be k+1.
k+1 is the only row where V_i and W_i end with the same symbol.
So, 0, i_2, …, i_{m+1}, k+1 is a PCP solution to $P^@$.
This means that $V0Vi_2Vi_3...Vi_mV_{k+1} = W0Wi_2Wi_3...Wi_mW_{k+1}$.
Recall that V0 = @V1, W0 = W1, V_{k+1} = \$ and W_{k+1} = @\$.
If we drop all the @ and \$ from V_i and W_i, we get back $X1X2...Xi_m$ and $Y1Y2...Yi_m$.
$X1X2...Xi_m = Y1Y2...Yi_m$.
Therefore, 1, i_2, …, i_m, is an MPCP solution to P.