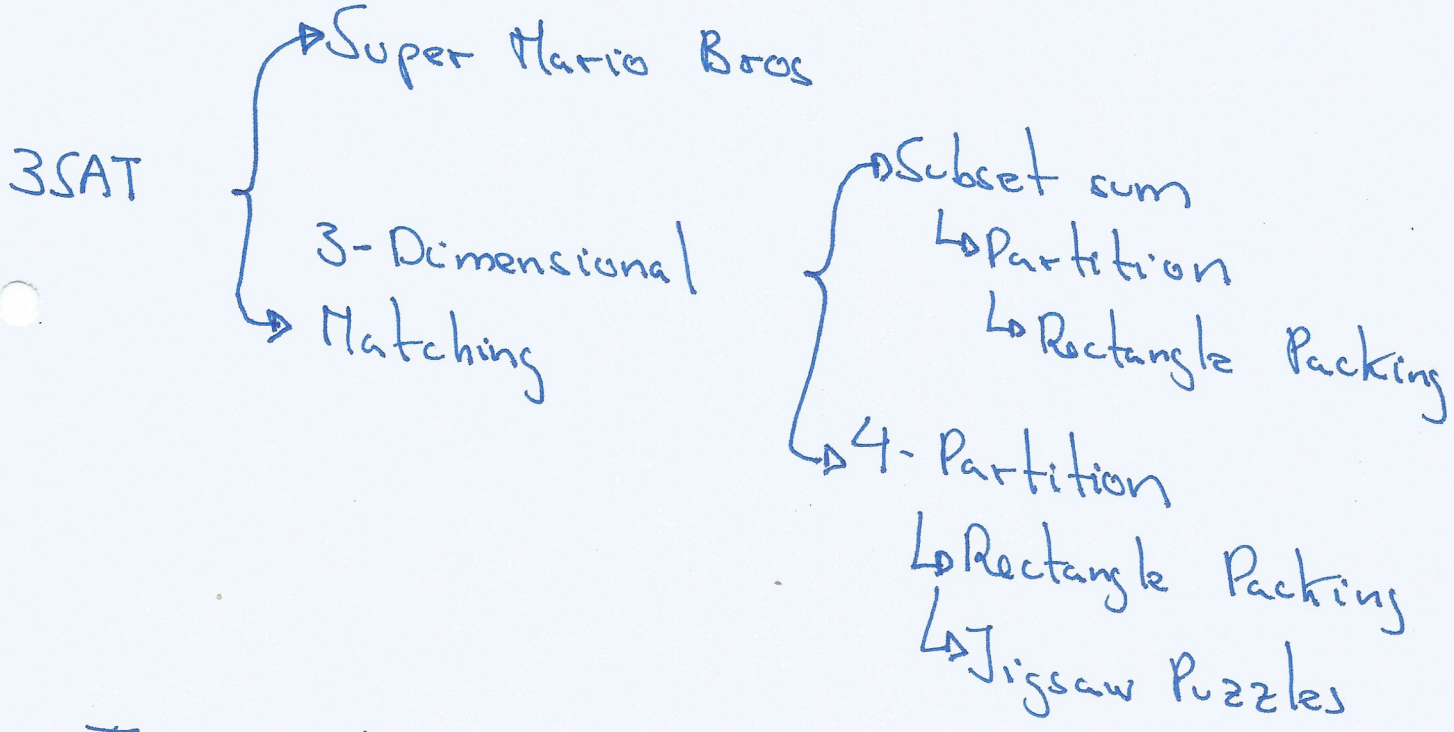
Chapter 34 - NP-Completeness

- MIT OpenCourseware 6.046
- Chapter 34 Commentary

- An entire field in one lecture, should be fun...

- **Problems we will look into:**



- This is all about reductions: converting one problem into another

- **Recall:**

Class P = { problems (size n) solvable in $n^{O(1)}$ polynomial time }

Class NP = { problems solvable in non-deterministic polynomial time }

In particular we will focus on:

NP = { decision (yes/no) problems solvable in non-deterministic

Question: What is a non-deterministic model of computation

Non-deterministic model:

- Can guess one of out polynomially many options in $O(1)$ time
- Idea:
  - I give the computer the polynomially many options I am interested in
  - The computer is going to give me one of them, i.e. give me a good guess (yes answer if possible
  - This is weird because it is biased towards yes answ
  - Machine magically gives me the answer ("Lucky mod of computa
- Lets look at an example

3SAT: given boolean formula of the form

$$(u_1 \vee u_3 \vee \overline{u_6}) \wedge (\overline{u_2} \vee u_3 \vee \overline{u_7}) \wedge \dots$$

(Or, Not, And; literal, clause)

- A formula that is an And of Or
- Each Or expression only contains three literals
- End result is 0 or 1, i.e. yes/no
- Problem: can you set the variables $u_1, u_2, \dots \to \{T, F$ such that formula = T

- This is a hard problem: we don't know a polynomi time algorithm, there probably isn't one

- However, there is a polynomial-time non-determini algorithm, i.e. 3SAT ∈ NP

- NP algorithm:
  - guess v_1 = T or F (O(1) time)
  - guess v_2 = T or F (O(1) time)
  - ⋮
  - Check if formula is satisfied { Formula = T retur Formula = F retu (this takes polynomial time)

  Because NP is biased toward the yes answer (it alw find an Yes answer if it can) if there is some wa to satisfy the formula we will get it

- Prototype for NP algorithm
  1) Perform some type of guessing
  2) Perform some type of polynomial checking

- We can only check if a problem is satisfiable for yes answers because we have a polynomial time checking proced
- To prove that a problem is not-satisfiable we would need to check if not a single yes answer occurs (exp time)

I.e.:

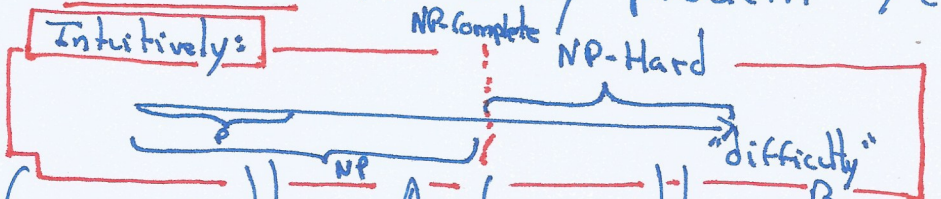
- Easy way to check if an answer is yes
- No easy way to confirm that the answer to the overall problem is no (i.e. no yes answer occurs)

- Equivalent definition for NP:

NP = { decision problems with poly-size certificates & poly-time verifiers for YES inputs }

- Other definitions:

- Problem U is NP-Complete if U ∈ NP & U is NP-Hard
- Problem U is NP-Hard if every problem Y ∈ NP reduces to U



- Reduction from problem A to problem B is a poly-time algorithm for converting A inputs into equivalent B inputs
  - same yes/no answer

Question:
- Why would I care about a reduction?
  - If I know how to solve problem B then I also know how to solve problem A

- If:
  - We have a reduction from A to B then if:
  - $B \in P$ then $A \in P$
  - $B \in NP$ then $A \in NP$

- It is really tricky to get these directions right...
- Here goes a handy guide:

How to prove Q is NP-complete

① Prove $Q \in NP$ → Two possible approaches:
1. Non-deterministic algorithm
2. Certificate + Verifier

② Prove Q is NP-Hard:

2.1 Reduce from known NP-Complete problem Y to Q

- Historical notes:
  - In the 1970's Cook proved that 3SAT was NP-Complete
  - This was the first ever proof that a problem was NP-Complete
  - That was very difficult since he had to prove that any problem in NP could be reduced to 3SAT

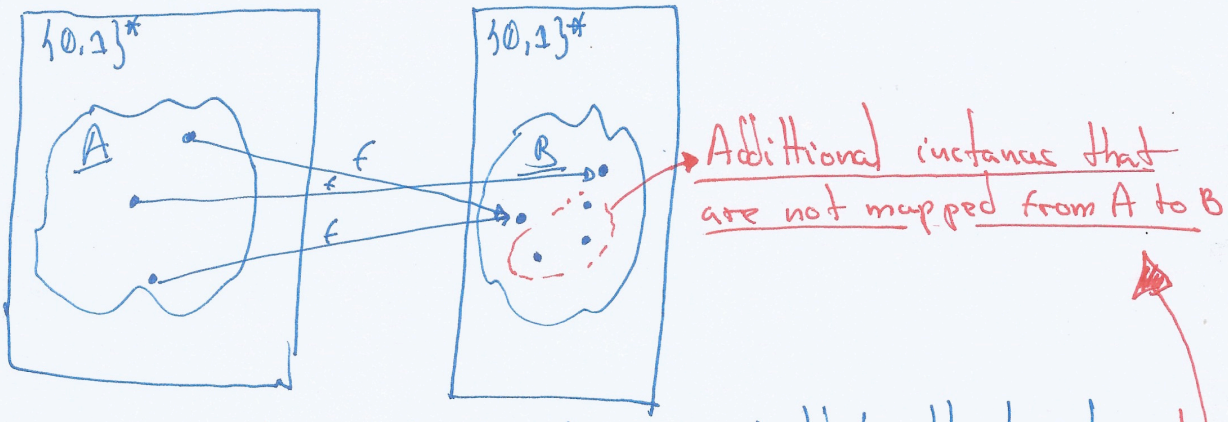
↳

Question:

Why is B at least as hard as A?

- If I can solve B then I can solve A (Assuming a poly-time reduction exists)
- Then B is at least as hard as A
- Why?

  All of A's inputs must be mapped into equivalent B inputs: for reduction



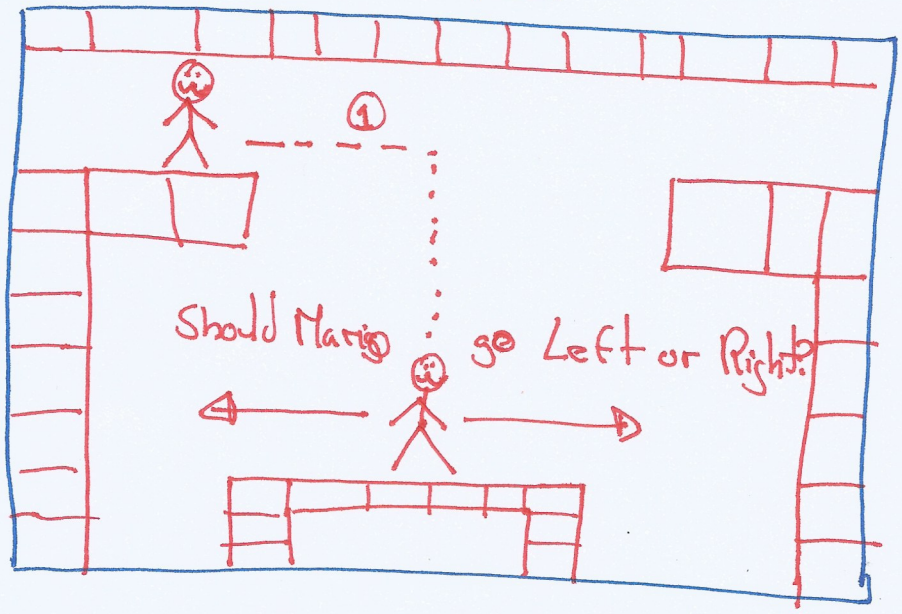
- This means that in order to solve B we should be able to solve any instance of A and potentially any other instances of B that are not covered in A.
- These additional instances not covered in A may require:
  - More time
  - Less time
  - Combination of more/less time:
    - Some instances require less time
    - Other instances require more time

∴ At the very minimum B needs to be as hard as A

- **Question:** How many problems exist in NP ?
  - A Lot !!! Probably inumerable...
- So Cook had to devise a general proof method that circumvented this issue
  - We will not see the proof in this class... //
- But now that someone did that tedious work for us we know that 3SAT is NP-Complete
- Ok, so lets do some reductions !!! //

Super Mario Bros

- Is NP-Hard
- Generalize to arbitrary screen size n x n
- Level variable example:



Intuition why 3SAT is NP-Complete:

1) If we have a NP problem then we have a poly-time algorithm verification procedure
2) Software and hardware are basically the same thing
3) We can therefore convert the verifier into a circuit that implements the algorithm
4) A circuit can be represent by a boolean formula
5) Any boolean formula can be represented as a conjunction (AND$_s$) of triple disjunctions (OR$_s$)
6) This formula is equivalent to the original verifier algorithm

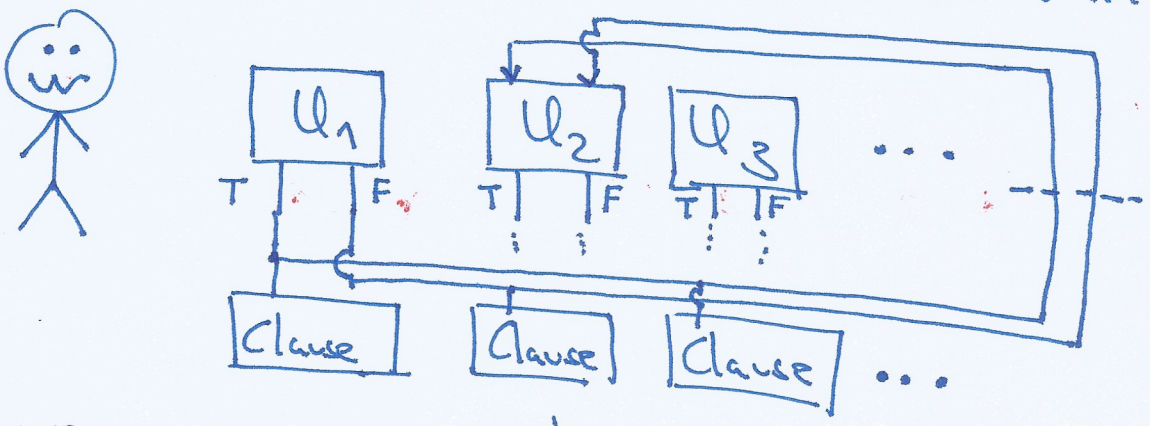
$\therefore_1$ Given an NP problem/algorithm we can convert it into 3SAT (NP-Hard)

$\therefore_2$ 3-SAT is also in NP

$\therefore_3$ Therefore 3-SAT is NP-Complete

- Therefore we can define a binary variable u_1 that represents a decision about whether Mario should go left or right

- We have to do this for a bf of variable:



- This way we can ~~redu~~ reduce 3SAT to SuperMario which means that it is NP-Hard

- Lets go to the next problem...

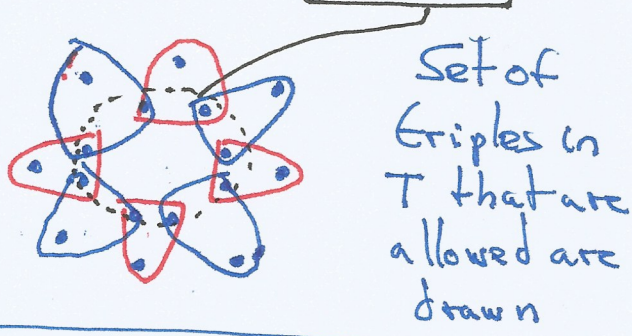
3-Dimensional Matching: (3DM)

Problem: Given disjoint set U, Y, Z each size n
- Given triples $T \subseteq U \times Y \times Z$ is there a subset $S \subseteq T$ such that every element $\in U \cup Y \cup Z$ is in exactly one $s \in S$

- This problem is NP-Complete

- Lets see the reduction:

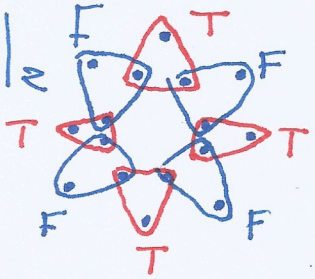
Variable u_i ⟶ (is converted into the picture)



Notes:

- We can either choose the red <u>or</u> the blue rectangles
- We cannot mix and match red and blue because these would result in the triples overlapping
- Because everybody has to be covered we either choose the red or the blue triangles

- Because we have this binary division we can say that the red triangles represent a <u>True</u> variable and the blue triangles a <u>False</u> statement/variable



Question:

How big should the inner wheel be?

$2 n_{u_i}$ "wheel size"

↳ number of occurrences of u_i in the formula (i.e. either u_i or $\overline{u_i}$)

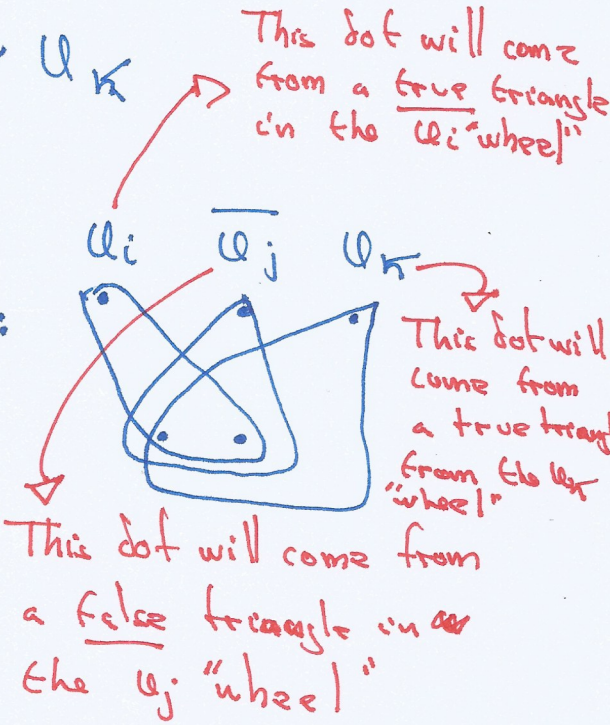
- **Question:** Why the factor of 2 in $2n_{u_i}$?

  - Each u_i may have either value false or true
  - Thus we need for each u_i to express these two possibilities, i.e. $2n_{u_i}$
  - This will guarantee that we have enough of each points (triangles) to connect to the 3-SAT clauses

- **Question:** How do we obtain a clause?

  - Clause example: $u_i \vee \overline{u_j} \vee u_k$
    - We would need one "wheel" for:
      - u_i
      - u_j
      - u_k

      } resulting in: an additional wheel



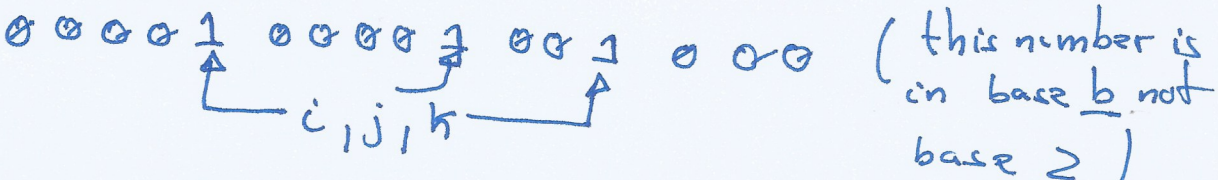
  - And this way we are able to reduce from 3SAT to 3DM

- Lets move on to other problems...

Subset sum problem:

Given:

- $A = \{a_1, a_2, a_3, \dots, a_n\}$ with $\forall_k a_k \in \mathbb{Z}$
- Target sum t with $t \in \mathbb{Z}$
- **Problem:** Is there a subset S of the integers that adds up to that target, i.e.
  - $S \subseteq A$
  - $\sum S = \sum_{a_i \in S} a_i = t$
- This problem is NP-Complete but it is also weakly NP-hard
- We get to choose the numbers, so we are going to think of them in base $b = 1 + \max_i a_i$
- Given triple (a_i, a_j, a_k) we are going to convert it to a number that looks like this:



  - Although in base b we only use digits 0 or 1
  - All the 1 digits corresponds to the indexes i, j and k
  - In algebra this number is $b^i + b^j + b^k$

- We do this because the target sum will be:

$$t = 1\ 1\ 1 \ldots 1$$
$$= \sum_i b^i$$

Question: Why does this work?

- This works because when we select a "correct" index b^i to be part of the sum we will set that index in t to 1. If we choose something "wrongly" the index will be different than 1 (possibly 0 or 2, 3, ... b)

- This means that we want to avoid collisions in the triple. We want to find triples that indexes that do not overlap.

Question: Does this remind you of anything?

- This is exactly the 3DM problem...
- Notice how we expressed the problem in "triple" form

Question: Why is this called "weakly" NP-Hard?

- # digits = $O(n)$
- If values of numbers are large ⇒ exponential because we have to exponentiate
- but if values of numbers are small ⇒ problem is easy
- Equivalent to pseudo-polynomial concept

Partition Problem

- Given $A = \{a_1, a_2, \ldots, a_n\}$ $\forall_n\ a_n \in \mathbb{Z}^+$
- Problem: Is there a subset $S \subseteq A$ such that

$$\sum S = \sum (A - S) = \frac{\sum A}{2}?$$ (Assume A is even)

i.e. we are splitting a set into two halves of equal sum.

- Partition is a special case of subset sum. Notice that the partition problem is equivalent to the subset sum for $t = \sum A / 2$
- In other words I can reduce from subset sum to partition:
  - Let $T = \sum A$
  - Add $a_{n+1} = T + t$ to set A
  - Add $a_{n+2} = 2T - t$ to set A

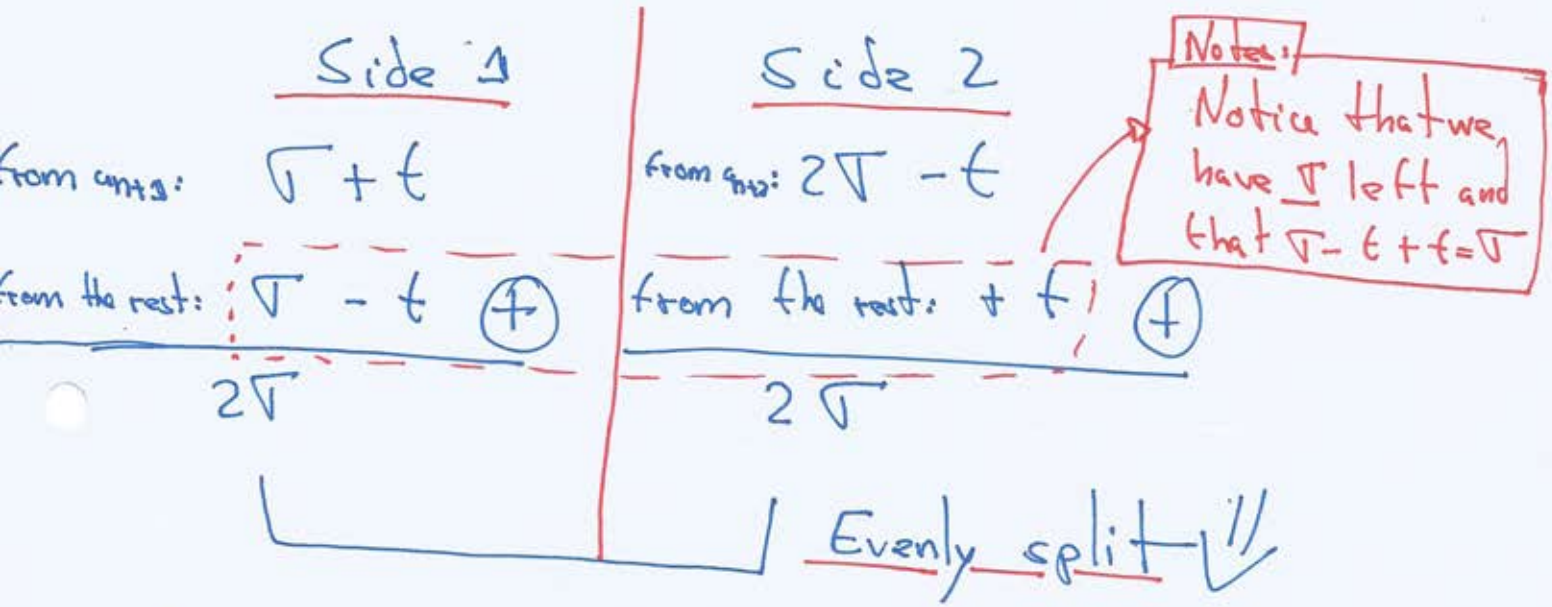
Question:

Suppose a_{n+1} and a_{n+2} go to the same partition. What happens?

- One side will have: $T + 2T + t - t = 3T$
- The other side will have the remaining elements of A that have combined value $\frac{3}{2}T$
- ∴ There is no way to evenly split if a_{n+1} and a_{n+2} go to the same

∴2 a_{n+1} and a_{n+2} need to go to different partitions

∴3 • There is a side that has $T+t$ and the other side has $2T-t$ and we have T left in set A



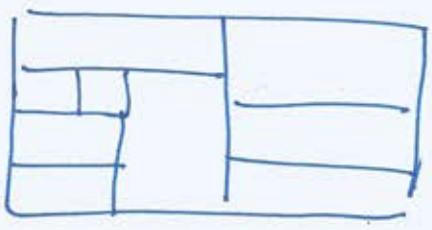
∴4 We can reduce subset to partition

Rectangle packing problem

- We have rectangles of different sizes:

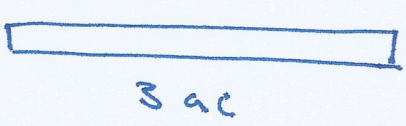
Note:
Sum of the areas of these rectangles is equal to the area of T

- We want to put these rectangles into this picture I without any overlap

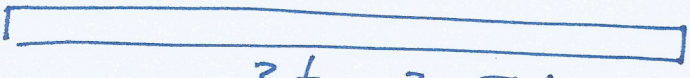


- Problem: Can we pack the R_i rectangles into I without

- Reduction from partition:
  - Each number a_i from partition is converted into a rectangle

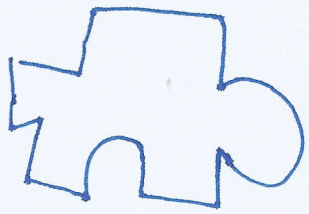


  - Target rectangle T: 2

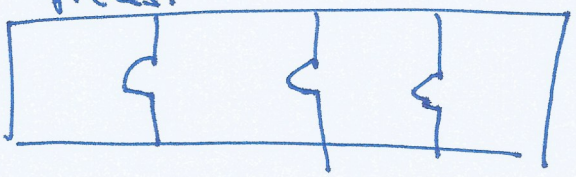


  - Because each of the R_i rectangles is at least 3 long we cannot pack them vertically, i.e. they have to be packed horizontally
  - Thus we need to solve the partition problem.

Jigsaw Puzzles Problem



Simulate a rectangle with jigsaw pieces:



Problem: How to fit the jigsaw pieces?

↳ Can be viewed as a form of rectangle partitioning

...