In this lecture we moved on to the completion of the proof from last lecture, i.e., proof that RDS problem is NP-Complete. We discussed previously that the partition problem is NP-Complete and by reducing the partition problem to the RDS problem we can prove that RDS is NP-Complete. Following is a brief review and problem formulation for RDS and PART.

### 6.1 Job Scheduling

Given \( n \) jobs 1, 2, \( \cdots \), \( n \), with processing times \( t_1, t_2, \cdots, t_n \), release times \( r_1, r_2, \cdots, r_n \) and deadlines \( d_1, d_2, \cdots, d_n \), with \( r_i, d_i \in \mathbb{Z}_{\geq 0} \) and \( t_i \in \mathbb{Z}_{> 0} \), is there a feasible schedule?

If \( t_j \) is scheduled from \( t_s \) to \( t_f \) in a feasible schedule, then the following conditions must be true.

1. \( t_s \geq r_j \)
2. \( t_f \leq d_j \)
3. \( t_f - t_s = t_j \)
4. There is no \( k \neq j \) \( \ni \) job \( k \) is scheduled from \( t_s \) to \( t_f \)

Also every job must be scheduled and the problem is to find the ordering of these jobs for a feasible schedule.

### 6.2 Proof for NP Completeness of RDS

As discussed in the previous lecture, in order to prove NP Completeness of a problem \( Q \), two steps are to be performed. The first step is to prove that the problem \( Q \in NP \). This can be done be showing that a given “yes” solution of the problem \( Q \in NP \). This can be done by showing that a given “yes” solution of the problem \( Q \in NP \) can be verified in polynomial time.

#### 6.2.1 RDS \( \in NP \)

Given some solution with the permutation \( \pi = \{t_{\pi_1}, t_{\pi_2}, \cdots, t_{\pi_n}\} \). \( \pi \) can be verified by checking the four conditions described in the previous section for each element of the solution \( \pi \). As it can be performed in polynomial time in \( n \) (the input size is \( O(n) \)) so we conclude that RDS \( \in NP \).

#### 6.2.2 PART \( \propto_P RDS \)

Let us consider an instance \( I \) of PART, then it can be transformed to an instance \( I' \) of RDS by equating each weight \( w_i \) to the processing times \( t_i \) for \( 1 \leq i \leq n \). Now, we further define release times and deadlines
to show how the solution to $I'$ corresponds to a solution to PART.

Now we can set the release times for this instance of RDS to be 0 and the deadlines to be equal to the sum of all the weights $i.e., \sum_{i=1}^{n} w_i$.

We set the above values to ensure that we do no enforce any time slot on any of the jobs. They can fit in any natural slot for a feasible solution.

So far we have made the following assumptions:

- $t_i = w_i$
- $r_i = 0$
- $d_i = \sum_{i=1}^{n} w_i$

But now our task is to divide jobs in two such sets that their sums (of processing times) are equal. Doing this would ensure that a solution for RDS gives a solution to PART as well, from previous lecture it can be reviewed that a solution to PART contained two non overlapping subsets of $S$ such that their individual sums were equal.

This can be achieved by introducing a new job of processing time one unit, which we will call as an *enforcer*. The release time, processing time and deadline for this new job are given below:

- $t_{n+1} = 1$
- $r_{n+1} = \sum_{i=1}^{n} w_i$
- $d_{n+1} = \frac{\sum_{i=1}^{n} w_i}{2} + 1$

After the introduction of this new job, the deadline for all other jobs will have to be updated which will now be:

- $d_i = \sum_{i=1}^{n} w_i + 1$

The enforcer ensures that if a solution exists then enforcer will always get scheduled in its enforced time slot, if it does not get scheduled in its enforced time slot that would imply that a feasible solution does not exist (or the claim of a feasible schedule is farce). Once the enforcer is in its place, the sum of processing times of the jobs before and after it will have equal total time. Also because of condition four of the RDS problem, these two sets of jobs will be non overlapping thus satisfying the PART problem requirements.

The above discussion proves that RDS is in fact an NP-Complete problem.

### 6.3 SATISFIABILITY (SAT)

This problem looks for a solution for a given boolean expression such that it evaluates to a TRUE value. A boolean expression can always be represented or converted into an equivalent conjunctive normal form (CNF) or a disjunctive normal form (DNF). It is formally stated as:

Give a set $U = \{u_1, u_2, \cdots, u_n\}$ of boolean variables, a truth assignment function $t: U \rightarrow \{true, false\}$. A clause over $U$ is a set of literals over $U$ (i.e., a disjunct of literals) where literals in $U$ are $\{u, \bar{u}\}$.
6.3.1 Instance of SAT

There is a set $U$ of variables and a collection $C$ of clauses over the literals of $U$, then is there a combination of truth values to literals in $C$ such that it evaluates to TRUE?