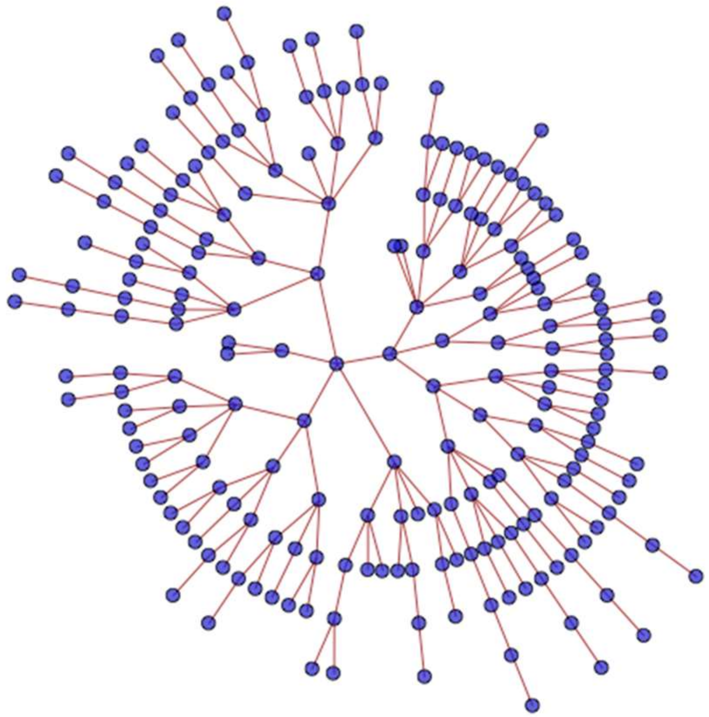
Parallel programming

HW3 assignment



Scheduling problems recap

- We try to find an assignment of tasks to resources, which is usually the time.
- A task can be characterized by several parameters, e.g., release time, deadline/due date, processing time, etc.
- We look for *feasible* or ***optimal*** schedule
  - Feasible schedule does not violate any condition of the problem
  - Optimal schedule is the best around all the feasible schedules according to some criteria

Problem formulation

- Given a set of tasks **T** = {T_1, . . . , T_n}, where each task $T_i \in$ **T** is characterized by its release time r_i , deadline d_i and processing time p_i
- We want to find a feasible schedule (start times of the individual tasks, tasks cannot overlap) such that the completion time of the last task (C_{max}) is minimal
- The problem is **NP-hard**, which can be shown by a polynomial reduction from 3-partition problem.

Bratley's algorithm

- Based on the **branch-and-bound** procedure.
  - It can be seen, that a complete permutation tree has *n!* leaves.
- We can try to derive some pruning rules using the objective function or the tasks constraints.
  - In each node of the tree, we can compute a lower bound LB (based on the current partial solution)
  - Compare LB to global upper bound UB (which is obtained from some feasible solution/approximation algorithm/estimation).

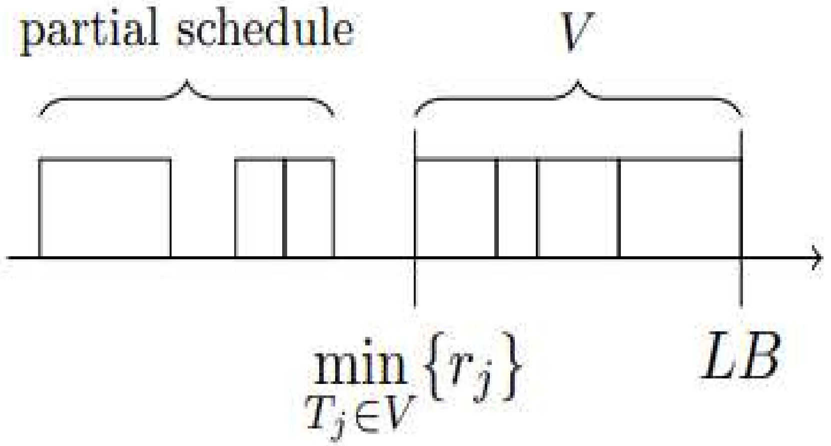
Rule 1. Some deadline is missed

- Some unassigned task may already miss its deadline, so it makes no sense to continue, because in the future this task will certainly miss its deadline.
- $(\exists\, T_j \in V : \max\{c, r_j\} + p_j > d_j\,) \Rightarrow$ prune this node.
  - c – length of the current (partial) schedule
  - V – a set of non-scheduled tasks

Rule 2. Violated upper bound

- If we already know some feasible schedule, we can its length as the upper bound (UB) for the optimal value, i.e. we can calculate the lower bound (LB) of the current solution and prune the node if LB ≥ UB.

- $LB = \max\left\{c, \min_{T_j \epsilon V}\{r_j\}\right\} + \sum_{T_j \epsilon V} p_j$
  - c – length of the partial schedule
  - V – a set of non-scheduled tasks



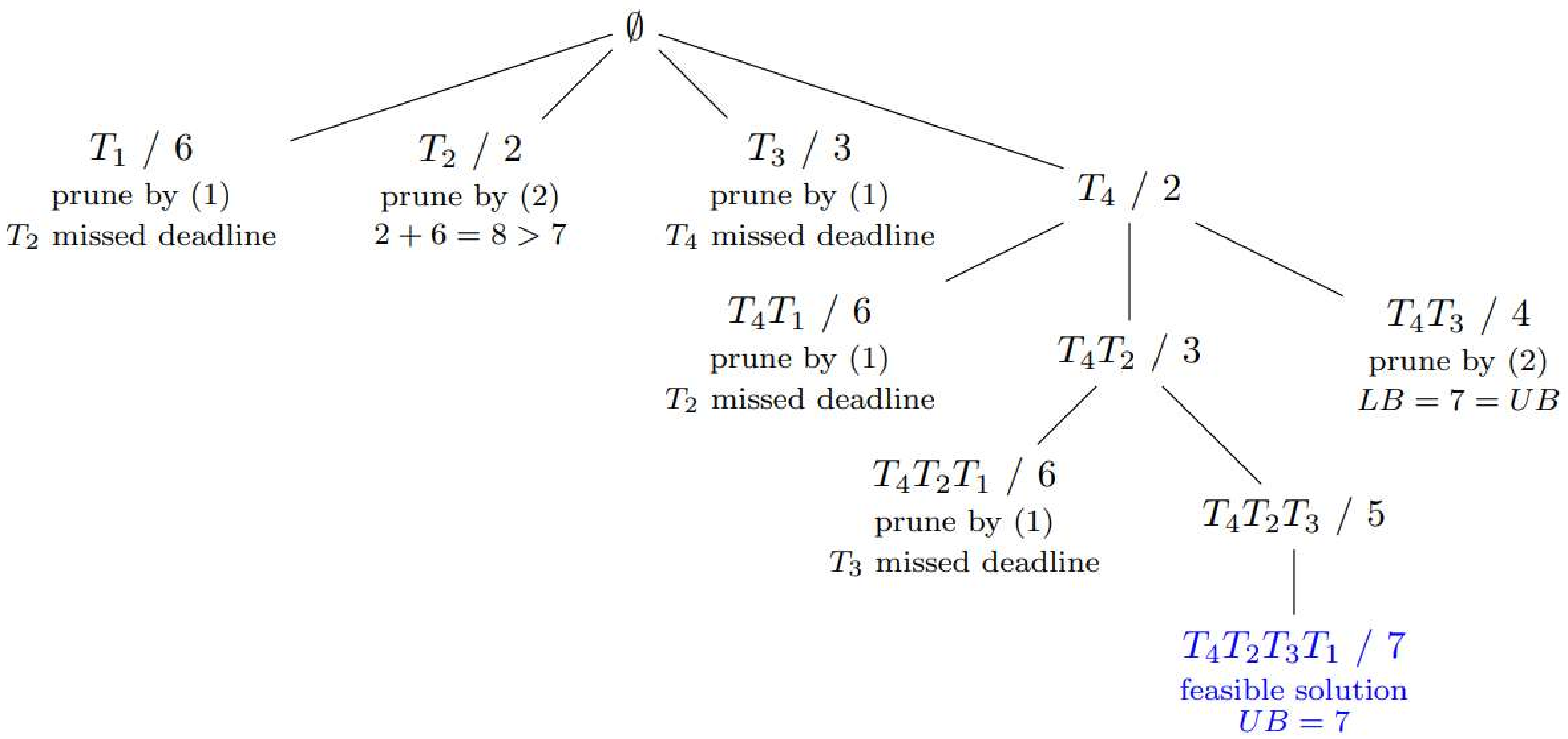
Rule 3. Decomposition

- We might be able to detect, that the partial solution we have in the current node is optimal, therefore it is not necessary to backtrack.

- $\left(c \leq \min_{T_j \epsilon V}\{r_j\}\right) \Rightarrow$ do not backtrack
  - c – length of the current (partial) schedule
  - V – a set of non-scheduled tasks

An example with four tasks

T_i	p_i	r_i	d_i
T_1	2	4	7
T_2	1	1	5
T_3	2	1	6
T_4	2	0	4



HW3 assignment

- Your task is to implement in MPI the described branch-and-bound algorithm including all the three pruning rules
- Hint: Create a communication protocol and use assynchronous communication. Be careful with tags and requests.
- Flags for g++ (used by UploadSystem)
  - -Ofast -std=c++17 -march=native
- The link for the video (CZ) and the document (EN) from Combinatorial Optimization course with the description of Bratley‘s algorithm:
  - https://www.youtube.com/watch?v=kbQ0J6I72Ww
  - https://cw.fel.cvut.cz/b202/_media/courses/ko/12_bratley.pdf

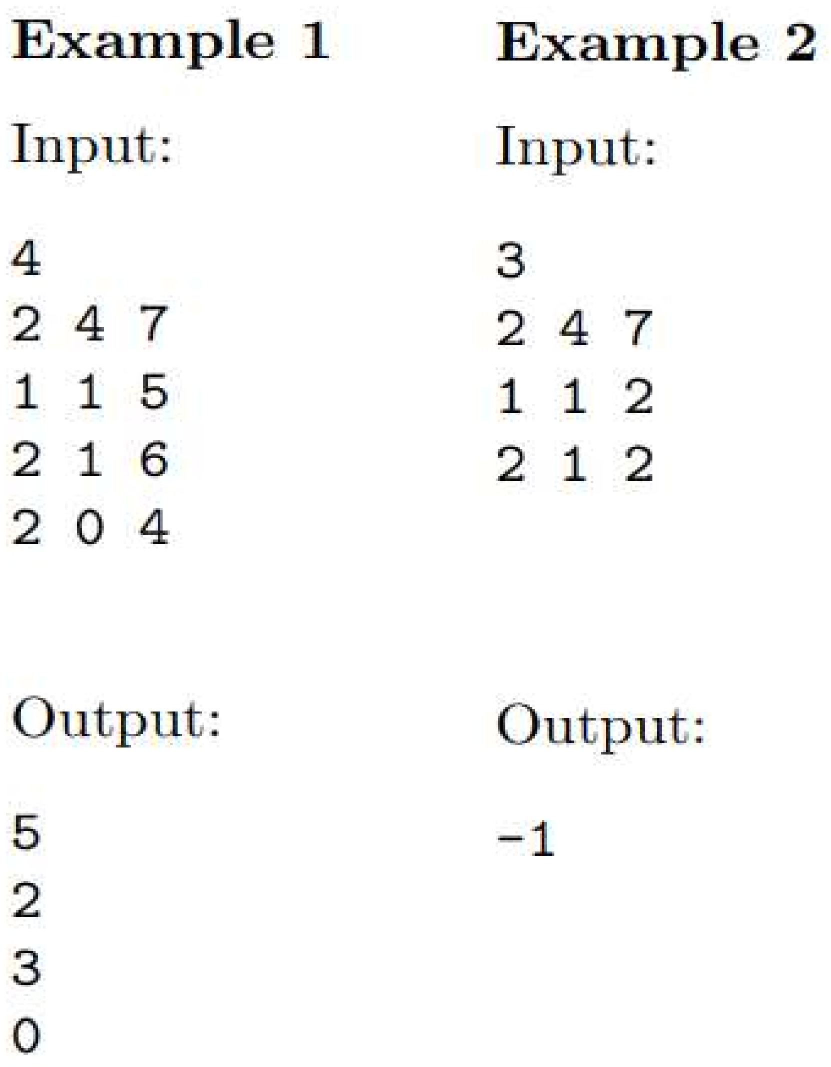
Input and output format

- Your program will be called with two arguments
  - The first one is the absolute path to input file
  - The second one is the absolute path to output file (the file itself must be created by your program)
- Let n be the number of tasks. Then the input file has n + 1 lines and has the following form:

$$\begin{array}{lll} n & & \\ p_1 & r_1 & d_1 \\ p_2 & r_2 & d_2 \\ \vdots & & \\ p_n & r_n & d_n \end{array}$$

- If the input instance is infeasible, then the output file consists of the single line containing −1. On the other hand, if the input instance is feasible, then the output file consists of n lines and has the following form (see next slide)

Examples of input and output



Example 1

Input:

```
4
2 4 7
1 1 5
2 1 6
2 0 4
```

Output:

```
5
2
3
0
```

Example 2

Input:

```
3
2 4 7
1 1 2
2 1 2
```

Output:

```
-1
```