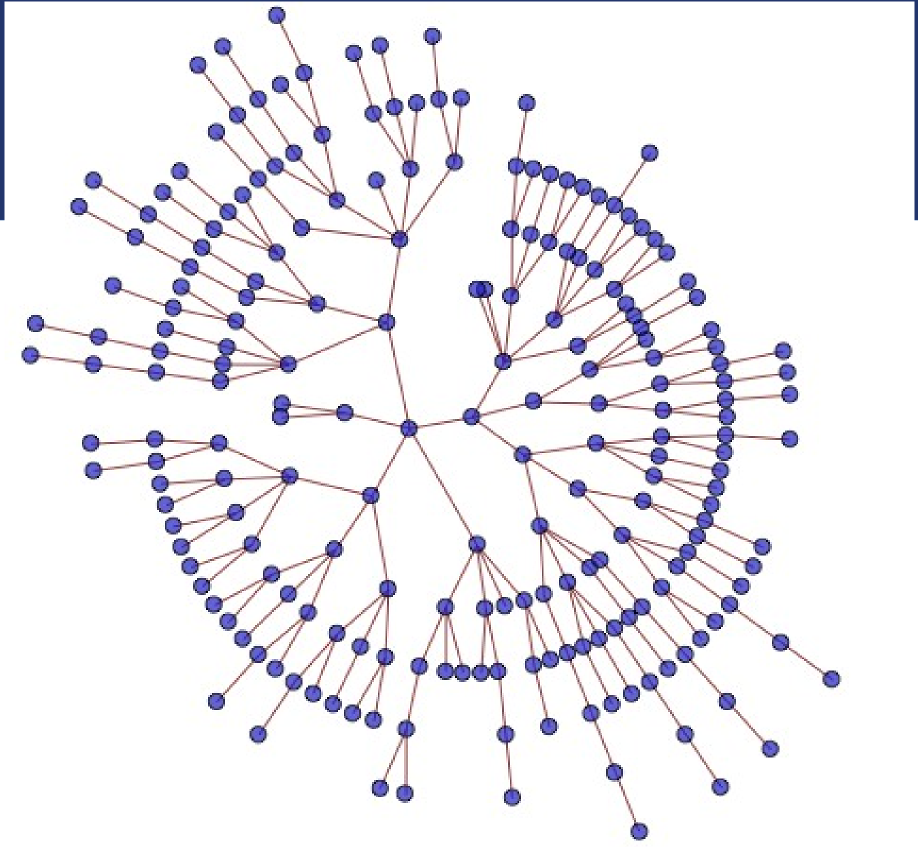
Parallel programming

HW3 assignment



FAKULTA
ELEKTROTECHNICKÁ
ČVUT V PRAZE

Revision of scheduling

- In scheduling problems, we are trying to find assignment of tasks to resources in time
  - Tasks can be characterized by some parameters (e.g. release time, deadline/due date, processing time, ...)
- We might be looking for a feasible schedule, or for an **optimal schedule** with respect to some objective function

Monoprocessor scheduling

- Single machine and tasks are characterized by release times and deadlines, where we want to minimize a total length of the schedule
- Given a set to 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 a **branch-and-bound** procedure
    - It can be seen, that the 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)

(1) Missed deadline

- It might happen that unassigned task would miss its deadline when assigned to the current schedule, if that is the case, prune this node
  - It is meaningless to continue, because in the future, some task would surely miss its deadline
- $(\exists T_j \in V : \max\{c, r_j\} + p_j > d_j) \Rightarrow$ prune this node.
    - c – length of the partial schedule
    - V – a set of non-scheduled tasks

(2) Bound on the solution

- We might have already found some feasible solution, which might not be optimal. However, we can use its quality as an upper bound (UB). We can calculate lover bound (LB) of the current solution and prune this 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

(3) Decomposition

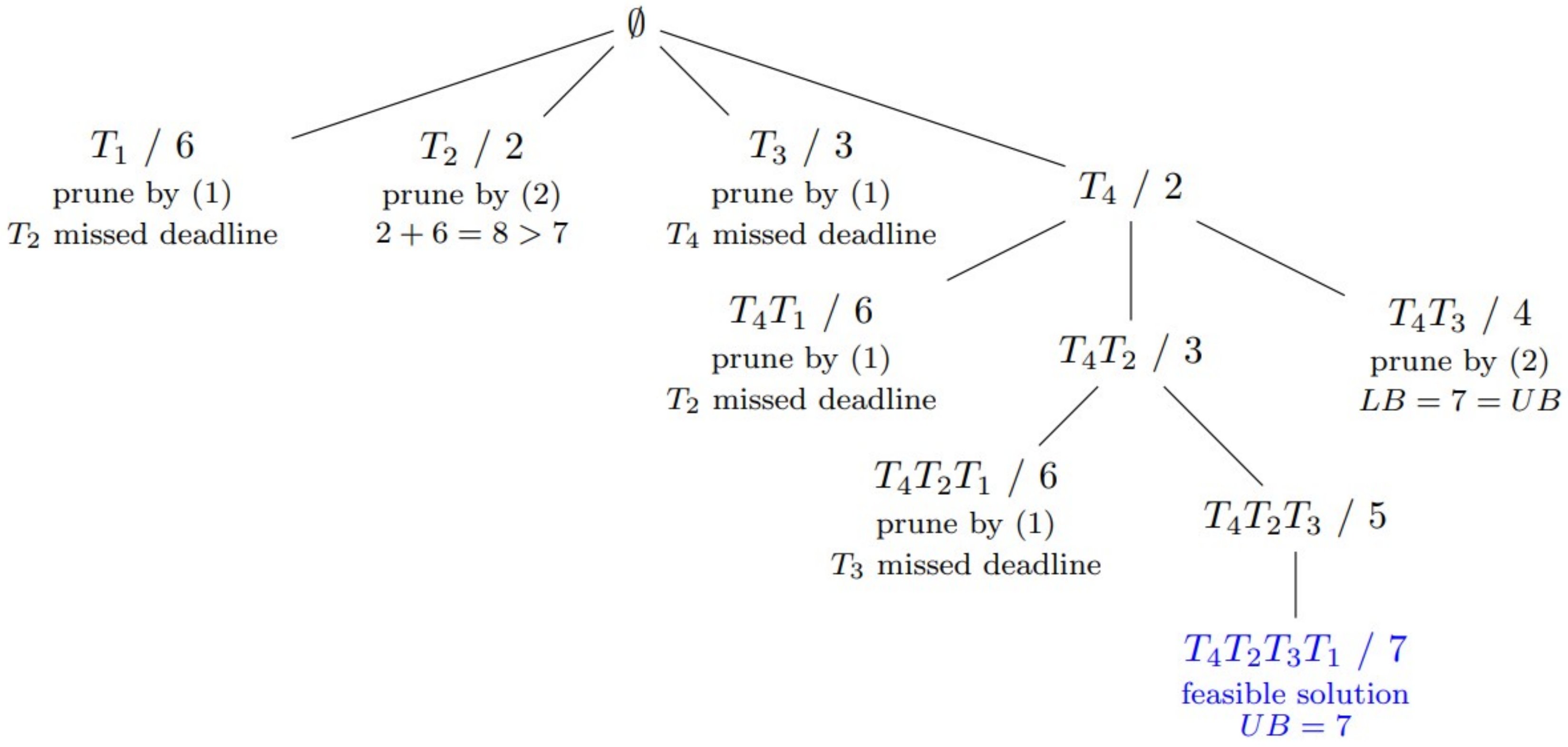
- We might be able to detect, that the partial solution we have in the current node is optimal, therefore it might not be necessary to backtrack
- $\left(c \leq \min_{T_j \epsilon V}\{r_j\}\right)$ => do not backtrack

  c – length of the partial schedule

  V – a set of non-scheduled tasks

Example with 4 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 parallel branch-and-bound algorithm for Bratley's problem using MPI
  - You should implement all three elimination rules
- Flags for g++ (used by UploadSystem)
  - -Ofast -std=c++17 -march=native
- Video (CZ) and document (EN) from Combinatorial Optimization course with description of Bratley's algorithm:
  - https://www.youtube.com/watch?v=kbQ0J6I72Ww
  - https://cw.fel.cvut.cz/b202/_media/courses/ko/12_bratley.pdf

Hints

- Use stack instead of recursion
  - Current schedule can be implemented as vector
  - Vector size represents the the depth of algorithm
  - Send only the current schedule between the processes
- You can use master-slave
  - Some inspiration can be found in codes from 2nd MPI seminar
- Use dynamic load balancing
  - For termination you can use Dijkstra's Token
  - See Combinatorial Algorithms lecture

Inputs and outputs

- Your program will be called with two arguments
  - The first one is absolute path to input file
  - The second one is the absolute path to output file which has to 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:

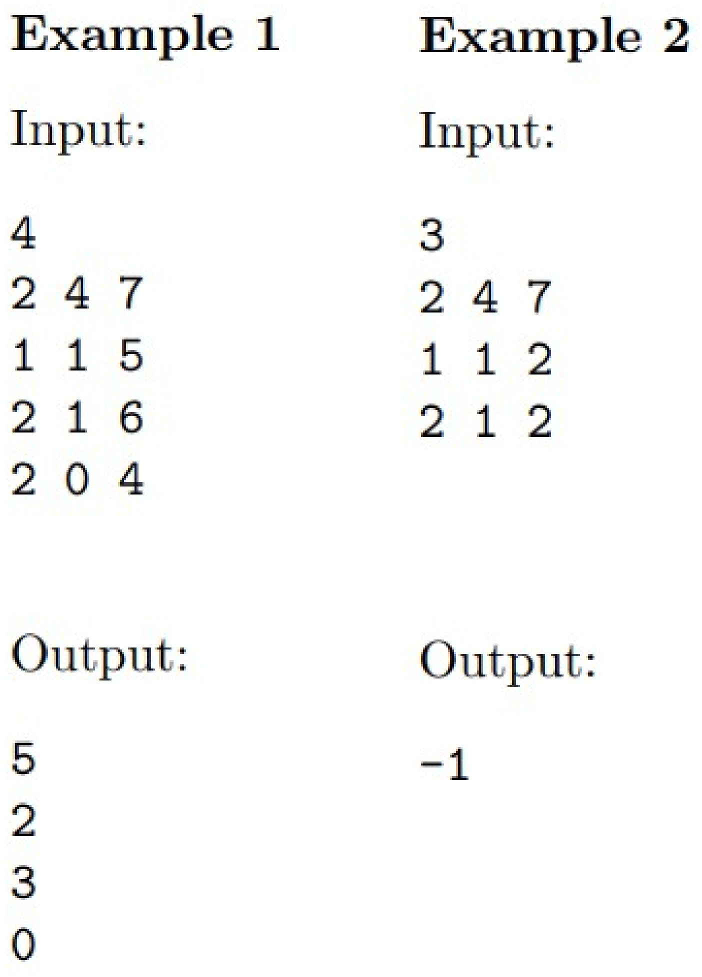
$$\begin{array}{lll} n & & \\ p_1 & r_1 & d_1 \\ p_2 & r_2 & d_2 \\ \vdots & & \\ p_n & r_n & d_n \end{array}$$

- If the input instance is infeasible, then the output file consists of the single line containing −1. For feasible instance, then the output file consists of n lines with start time of each task and has the following form:

$$\begin{array}{l} s_1 \\ s_2 \\ \vdots \\ s_n \end{array}$$

e.g. S_1 – start time for the task 1

Examples



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
```