

Solution Overview

The key idea of the solution is

- a) According to the precedence constraints, create a direct successor list for each task. This can be implemented as an array of linked-lists with size n and constructed by traversing the constraints set and inserting each successor at the appropriate list. (i.e. if $T_1 < T_2$ and $T_1 < T_3$, the linked-list at position 1 will have nodes for 2 and 3),
- b) Keep an array ES with the possible earliest starting time of each task. When a task T_i is scheduled, go through its successor list and update the ES value for each successor T_j if needed (i.e. the new ES value for a successor will be $\max(T_i \text{ scheduled time} + 1, ES[j])$).
- c) Keep an n by p array PS, where n is the number of tasks and p the number of processors, to hold the actual schedule. Each of the n positions of the array represents the corresponding time slot.
- d) Maintain the earliest available time slots as disjoint-sets. If the possible earliest starting time of task T is time slot i , T must be scheduled to execute at the first available (i.e. has a free processor) time slot $\geq i$. Intuitively, we can say that if time slots $i, i+1, \dots, i+j-1$ are full and $i+j$ is available, then the earliest available time slot for task T is $i+j$.
 - A triplet $\langle \text{Rank}, \text{Parent}, \text{Time} \rangle$ is maintained for each time slot set, where Rank is the rank and Parent are the usual fields for the tree representation as a successor array for disjoint-sets using the ranking rule. Time is the maximum time slot present in this set, maintained only at the root. The Rank for all time slots is initially 0 and Parent and Time is initially itself (i.e. for time slot 1 its triplet will be $\langle 0, 1, 1 \rangle$).
 - When a time slot i becomes full it is unioned with the next available time slot (i.e. $\text{FIND}(i+1)$). Since Time is to reflect earliest available time slot for the current set, and two time slots are unioned according to their rank, it is possible that a time slot with greater rank and a smaller Time be made the parent of the unioned set. In this case, the Time at the root is updated to reflect the largest Time. Thus the root can always return the correct earliest available time slot.

In this algorithm, time slots $i, i+1, \dots, i+j$ will eventually be unioned into one set, and $\text{Time}[\text{FIND}(k)]$ will return $i+j$ for $i \leq k \leq i+j$.

Both “union by rank heuristic” and “path compression heuristic” are used in Union-Find operations, so that n Union-Find operations can have $O(n \cdot \alpha(n))$ running time (since we consider that the actual U/Fs sequences are likely to be out of the corner-case situations where it would take $O(n \cdot \log^* n)$).

Pseudo-code

- 1 Build direct successor list for each task into an array of link-list
- 2 Create array ES of size n . Initialize all the elements in ES to 1.
- 3 Initialize the disjoint-sets array with the corresponding triplets.
- 4 **while** L is not empty **do**
- 5 Remove a task T from head of L
- 6 Find the possible earliest starting time of T , $t_i = \text{ES}[T]$
- 7 Find the earliest available time slot, $t_a = \text{Time}[\text{FIND}(t_i)]$
- 8 Assign T to the first available processor at $\text{PS}[t_a]$

```
9      if PS[ta] is “full”
10         UNION(ta, FIND(ta+1))
11      for each direct successor Ts of T do
12         ES[Ts] = max(ta+1, ES[Ts])
```

Running time analysis

- a) The running time of building of successor list for each task will be $O(n + m)$, where n is the time needed to initialize the array, and m is the time needed to traverse the constraint set.
- b) The running time of creating array ES is $O(n)$.
- c) The max number of union-find operations in the while loop is $5/3n$ (n FINDs + $n/3$ UNIONs + $n/3$ FINDs), so the running time of the while loop is $O(5/3n * \alpha(5/3n)) = O(n * \alpha(n))$.

According to above analysis, the running time of the algorithm will be $O(n * \alpha(n) + m)$.