Solution to Problem 14

Graded by: Pavel Laskov
Adopted from a solution by:

Problem Statement

Give a dynamic programming solution to the "increadible task scheduler" problem, for the case $D = 2$. Argue the correctness of your algorithm. Show the arithmetic complexity of your algorithm is polynomial in A and B.

Algorithm

The data structures to be used by the algorithm are the following.

The *task tuple* $T \equiv \{e^1, e^2\}$ contains efforts required from each programmer to complete a task. We can assume that efforts are already calculated according to programmers' skills because this calculation is straightforward and can be done in $O(1)$ time per task. If a programmer cannot accomplish a task, the value $+\infty$ is assumed to be defined.

Given a set of tasks $\{T_1, \ldots, T_A\}$, a *scheduling policy* S is an ordered set $\{s_1, s_2, \ldots, s_A\}$ where $s_i \in \{p^1, p^2\}$. In other words, each component of the scheduling policy identifies a programmer assigned to do a given task. A *partial scheduling policy* S_k is an initial subset of S of size k.

A *cumulative tuple* $C_{S_k} \equiv \{s_k, E^1, E^2\}$ is associated with each partial policy S_k. Here s_k is the identity of the programmer assigned to do the last task in the partial policy, E^1 is the total effort of the first programmer, E^2 is the total effort of the second programmer, on all tasks covered by S_k. We will define the *critical effort* $E_{\mathrm{cr}}(C_{S_k}) \equiv \max\{E^1, E^2\}$.

The *field matrix* F of size $A \times B$ contains cumulative tuples or empty entries. The number of rows in F is equal to the number of tasks; the number of columns is equal to the effort B the more qualified programmer would have spent if working alone on all problems. Additional rules for composition of the field matrix are: (a) row i can only have the tuples corresponding to policies of length i, and (b) column j contains tuples whose $E^1 = j$.

The goal is to find a policy minimizing the critical effort over all policies of length A:

$$S^* = \operatorname*{argmin}_{\text{all possible } S_A} E_{\mathrm{cr}}(C_{S_A})$$

We start with an empty field matrix, and insert the two tuples C_{S_1}, corresponding to two possible assignments of the first task, into the first line. (As it was mentioned before, the row is determined by the current policy length, the column is determined by the value of E^1 of a tuple being inserted). Then, iterating by row number, for every non-empty element in a row we generate two cumulative tuples and insert them into corresponding positions of the next line. If a position is already occupied by another tuple the conflict is resolved in favor of the tuple whose E^2 is smaller. This conflict resolution rule constitutes the dynamic programming step: we consider E^1 as a state of the problem, and E^2 as a cost function, and for every "path" ending in state E^1, the optimal path should bear the minimal cost. Finally, we examine the last line of the field table, and find the element minimizing $E_{\mathrm{cr}}(C_{S_A})$. To recover the optimal policy, we backtrack in the field matrix using information about the last programmer stored cumulative tuples. The algorithm is summarized in Algorithm 1. The algorithm takes a list of tasks *task_list* as an argument and returns a policy set S. It uses an auxiliary procedure *search_last_row()* to identify the tuple which, among other tuples in the last row, has the smallest value of $E_{\mathrm{cr}}(C_{S_A})$.

Algorithm 1 "Increadible Task Scheduler" algorithm

```
begin
  A ← task_list.size()                                        // Initialization
  B ← Σ_{i∈task_list} e_i^1
  field_matrix ← new matrix<cumulative tuple>(A × B)
  e1 ← tuple_list.front().e1
  e2 ← tuple_list.front().e2
  field_matrix[0, e1] ← {p1, e1, 0}
  field_matrix[0, 0] ← {p2, 0, e2}
  for i ← 2 to A do                                           // Main loop
      task ← task_list[i]
      for each non-empty tuple C in row i do
          C1 ← {p1, C.E1 + task.e1, C.E2}
          C2 ← {p2, C.E1, C.E2 + task.e2}
          if (C1.E2 < field_matrix[i, C1.E1].E2)∨
            (field_matrix[i, C1.E1] = ∅)
            then field_matrix[i, C1.E1] ← C1
          if (C2.E2 < field_matrix[i, C2.E1].E2)∨
            (field_matrix[i, C2.E1] = ∅)
            then field_matrix[i, C2.E1] ← C2
  final_tuple ← search_last_row()                             // Post-processing
  S[A] ← final_tuple.p
  for i ← A − 1 to 1 do
      task ← task_list[i]
      if final_tuple.p = p1
        then step ← tuple.e1
         else step ← tuple.e2
      final_tuple ← field_matrix[i, final_tuple.E2 − step]
      S[i] ← final_tuple.p
end
```

Proof of Correctness

By the definition of the *search_last_row* function, the tuple returned by the algorithm is optimal among all tuples in the last row. Therefore, it only remains to show that none of the tuples excluded from the last row of the field matrix can have the critical effort strictly less than that of the optimal tuple C^{opt}.

Suppose, by the way of contradiction, there exists such tuple C' that $E_{\mathrm{cr}}(C') < E_{\mathrm{cr}}(C^{\mathrm{opt}})$, but the position of C' in the final field matrix is occupied by the tuple C''. Because C'' has replaced C' in the same position in the field matrix, $E^1(C'') = E^1(C')$ and $E^2(C'') < E^2(C')$; therefore $E_{\mathrm{cr}}(C'') \leq E_{\mathrm{cr}}(C')$. On the other hand, by the nature of "last row optimization", $E_{\mathrm{cr}}(C^{\mathrm{opt}}) \leq E_{\mathrm{cr}}(C'')$. Hence, $E_{\mathrm{cr}}(C^{\mathrm{opt}}) \leq$

$E_{\text{cr}}(C')$. But this contradicts our earlier assumption that $E_{\text{cr}}(C') < E_{\text{cr}}(C^{\text{opt}})$. q.e.d.

Running-time Analysis

Obviously the bottleneck of the problem is nested loops in lines 9 – 19 whose limits are A and B. Hence, the overall running time is $O(AB)$.

Grading Policy

Points:

6	workable algorithm
2	proof of correctness
2	running-time analysis