Computer Science C73 November 9, 2022
Scarborough Campus University of Toronto

Solutions for Homework Assignment #6

Answer to Question 1.

Definition of the subproblems to solve: For each $i \in [0..n]$, define

$$C(i) = \text{maximum sum of the elements of a nonconsecutive subsequence of } A[1..i] = a_1, \ldots, a_i. \quad (*)$$

where the sum of the elements of an empty sequence (when $i = 0$) is taken to be zero.

Recursive formula to compute each subproblem:

$$C(i) = \begin{cases} 0, & \text{if } i = 0 \\ \max(0, A[1]), & \text{if } i = 1 \\ \max\big(C(i-2) + A[i], C(i-1)\big), & \text{if } i \geq 2 \end{cases} \quad (\dagger)$$

Justification why (†) is a correct formula to compute (∗): The base cases ($i \in \{0, 1\}$) are obvious. For $i \geq 2$, let A' be a nonconsecutive subsequence of $A[1..i]$ that maximizes the sum of its elements. There are two possibilities: (a) A' contains $A[i]$ or (b) A' does not contain $A[i]$. In case (a), since A' is nonconsecutive, its penultimate element is not $A[i-1]$. Therefore the part of A' preceding $A[i]$ is a nonconsecutive subsequence of $A[1..i-2]$, and by a straightforward cut-and-paste argument it maximizes the sum of its elements among all nonconsecutive subsequences of $A[1..i-2]$, and so the sum of its elements is $C(i-2) + A[i]$. In case (b), again by a straightforward cut-and-paste argument, A' is a nonconsecutive subsequence of $A[1..i-1]$ that maximizes the sum of its elements, and so the sum of its elements is $C(i-1)$. Therefore, $C(i)$ is the maximum of these two cases, i.e., $C(i) = \max\big(C(i-2) + A[i], C(i-1)\big)$.

Solving the original problem: By the definition of the subproblems, the maximum sum of the elements of a nonconsecutive subsequence A' of A is $C(n)$. By the argument above, if $n \geq 2$, $A[n]$ is in A' if and only if $C(n) = C(n-2) + A[n]$. In this case, the prefix of A' excluding $A[n]$ is a nonconsecutive subsequence of $A[1..n-2]$ that maximizes the sum of its elements, otherwise, A' in its entirety is a nonconsecutive subsequence of $A[1..n-1]$ that maximizes the sum of its elements. In both cases, we can continue backwards in the same manner to determine the rest of the elements of A'. If $n = 1$ then A' consists of the unique element of A if that is positive; otherwise, it is negative.

Pseudocode:

```
    MaxNonConsecSubseq(A[1..n])
1   C(0) := 0
2   C(1) := max(0, A[1])
3   for i := 2 to n C(i) := max(C(i − 2) + A[i], C(i − 1))
4   A' := empty sequence
5   i := n
6   while i ≥ 2 do
7       if C(i) = C(i − 2) + A[i] then
8           prepend A[i] to A'
9           i := i − 2
10      else
11          i := i − 1
12  if i = 1 and A[1] > 0 then prepend A[1] to A'
13  return A'
```

Running time: The running time of the algorithm is $\Theta(n)$, since it is dominated by the time for the for-loop in line 3 and the while-loop in lines 6-11, each of which takes $\Theta(n)$ time.

Answer to Question 2.

Definition of the subproblems to solve: For each $i \in [0..n]$ and $t \in [0..T]$ define

$$M(i,t) = \begin{cases} \textbf{true}, & \text{if there is an assignment of jobs } \{1,\ldots,i\} \text{ under which Machine 1 has load } t \\ \textbf{false}, & \text{otherwise} \end{cases} \qquad (*)$$

Solving the original problem: If t is the load of Machine 1 in an assignment that minimizes the makespan, then the minimum makespan is $\max(t, T-t)$. So, having computed the subproblems, the minimum makespan is $\min\{\max(t,T-t) : M(n,t) = \textbf{true}\}$.

Recursive formula to compute each subproblem:

$$M(i,t) = \begin{cases} \textbf{true}, & \text{if } i = 0 \text{ and } t = 0 \\ \textbf{false}, & \text{if } i = 0 \text{ and } t > 0 \\ \big(t \geq t_i \wedge M(i-1, t-t_i)\big) \vee M(i-1,t), & \text{otherwise} \end{cases} \qquad (\dagger)$$

Justification why (†) is a correct formula to compute (∗): For the base cases ($i = 0$) note that the only possible load for Machine 1 under an assignment of the empty set of jobs is zero. For the induction step, consider an assignment A of jobs $\{1,\ldots,i\}$ under which Machine 1 has load t. There are two cases, depending on whether job i is assigned to Machine 1 ($i \in A$) or not ($i \notin A$). In the first case, $t \geq t_i$ (otherwise Machine 1 would have load more than t since it is assigned job i). Furthermore, the remaining jobs assigned to Machine 1 have load $t - t_i$, and therefore there is an assignment of jobs $\{1,\ldots,i-1\}$ so that Machine 1 has load $t - t_i$. So, in this case $t \geq t_i$ and $M(i-1, t-t_i)$ is true. In the second case, the remaining jobs assigned to Machine 1 still have load t, and therefore there is an assignment of $\{1,\ldots,i-1\}$ so that Machine 1 has load t. So, in this case, $M(i-1,t)$ is true. Since these are the only two possibilities, we have that $M(i,t) = \big(t \geq t_i \wedge M(i-1,t-t_i)\big) \vee M(i-1,t)$.

Pseudocode:

```
    MinMakespan(⟨t_1, ..., t_n⟩)
1   T := 0
2   for i := 1 to n do T := T + t_i
3   M(0,0) := true
4   for t := 1 to T do M(0,t) := false
5   for i := 1 to n do
6       for t := 1 to T do
7           M(i,t) := (t ≥ t_i and M(i-1, t-t_i)) or M(i-1,t)
8   min_ms := ∞
9   for t := 1 to T do
10      if M(n,t) and max(t, T-t) < min_ms then min_ms := max(t, T-t)
11  return min_ms
```

Running time: The running time of this algorithm is dominated by the doubly-nested for-loop in lines 5-7, which obviously takes $\Theta(nT)$. Note that this is pseudopolynomial, not polynomial, as T is exponentially longer than the representation of $\langle t_1,\ldots,t_n\rangle$.

Comment: The problem of finding a minimum makespan assignment of jobs, even for two machines, is NP-hard, and it is therefore unlikely that a polynomial-time algorithm for it exists. Later in the course we will see a very simple and natural approximation algorithm for this problem (for any number of machines, not just two), based on the greedy method, that runs in polynomial time.

Answer to Question 3.

We will modify the Floyd-Warshall algorithm as follows. Assume that the set of nodes of the given graph $G = (V, E)$ is $V = \{1, 2, \ldots, n\}$. For every $i, j \in V$ and $k \in [0..n]$, we will maintain, in addition to the quantity $C[i, j, k]$, which is the minimum weight of $i \xrightarrow{k} j$ paths, the number of such paths, denoted $N[i, j, k]$. By definition, the two-dimensional array we are asked to compute is $N[-, -, n]$. The modified Floyd-Warshall algorithm is as follows:

```
    NumberOfShortestPaths(G, wt)
1   for i := 1 to n do
2       for j := 1 to n do
3           if (i, j) is an edge of E then
4               C[i, j, 0] := wt(i, j)
5               N[i, j, 0] := 1
6           else
7               C[i, j, 0] = ∞
8               N[i, j, 0] = 0
9   for k := 1 to n do
10      for i := 1 to n do
11          for j := 1 to n do
12              if C[i, j, k − 1] < C[i, k, k − 1] + C[k, j, k − 1] then
13                  C[i, j, k] := C[i, j, k − 1]
14                  N[i, j, k] := N[i, j, k − 1]
15              elsif C[i, j, k − 1] > C[i, k, k − 1] + C[k, j, k − 1] then
16                  C[i, j, k] := C[i, k, k − 1] + C[k, j, k − 1]
17                  N[i, j, k] := N[i, k, k − 1] ∗ N[k, j, k − 1]
18              else        ▶ C[i, j, k − 1] = C[i, k, k − 1] + C[k, j, k − 1]
19                  C[i, j, k] := C[i, j, k − 1]
20                  N[i, j, k] := N[i, j, k − 1] + N[i, k, k − 1] ∗ N[k, j, k − 1]
21  return N[−, −, n]
```

Correctness: The justification of correctness of the recursive computation of $C[i, j, k]$ as the weight of a shortest $i \xrightarrow{k} j$ path (lines 4, 7, 13, 16, and 19) is the same as in the correctness proof of the Floyd-Warshall algorithm and is not repeated here.

The number of $i \xrightarrow{0} j$ paths, i.e., paths from i to j that use no intermediate nodes, is 1 if there is a direct edge from i to j and 0 otherwise, justifying the initialization of $N[i, j, 0]$ (lines 5 and 8).

For $k > 0$, there are three cases, depending on whether $C[i, j, k-1]$ is (a) less than, (b) greater than, or (c) equal to $C[i, k, k-1] + C[k, j, k-1]$. In Case (a), allowing $i \xrightarrow{k} j$ paths to use node k is of no benefit and therefore the number of minimum-weight $i \xrightarrow{k} j$ paths is the same as the number of minimum-weight $i \xrightarrow{k-1} j$ paths, justifying line 14. In Case (b), the minimum-weight $i \xrightarrow{k} j$ paths must use node k. Such paths do not contain cycles because, by assumption, all cycles of G have positive weight; so they must contain a single occurrence of node k. Therefore each of them consists of an $i \xrightarrow{k-1} k$ subpath followed by a $k \xrightarrow{k-1} j$ subpath. Each of the first kind of subpaths can be combined with any one of the second kind of subpaths, each combination resulting in a different $i \xrightarrow{k} j$ minimum-weight path. Therefore, the number of minimum-weight $i \xrightarrow{k} j$ paths in Case (b) is the product of the number of minimum-weight $i \xrightarrow{k-1} k$ subpaths and the number of minimum-weight $k \xrightarrow{k-1} j$ subpaths, justifying line 17. In Case (c), there are two (mutually exclusive) sets of minimum-weight $i \xrightarrow{k} j$ paths: those that do not use node k and those that do. As explained in Cases (a) and (b), there are $N[i, j, k-1]$ of the former and $N[i, k, k-1] * N[k, j, k-1]$ of the latter, justifying line 20.