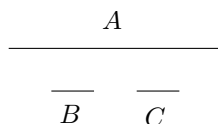


1. Consider the three jobs $A = (0, 5)$, $B = (1, 2)$, and $C = (3, 4)$.



The optimal solution is $\{B, C\}$.

- (a) This version considers the jobs in the order A, C, B and takes only job A .
- (b) This version considers the jobs in the order A, B, C and takes only job A .
- (c) works. Suppose S is ordered so that $s_1 \geq s_2 \geq \dots \geq s_n$. Let S' be the set of n jobs where $s'_i = -f_i$ and $f'_i = -s_i$, thus $f'_1 \leq f'_2 \leq \dots \leq f'_n$. In addition, the jobs in S' overlap in exactly the same way as jobs in the old set, i.e., (s_i, f_j) overlaps (s_j, f_j) if and only if $(-f_i, -s_i)$ overlaps $(-f_j, -s_j)$. Thus we may view this version of the algorithm, operating on input S , as the original version of the algorithm, operating on input S' . Since the original version is correct, this version is also correct.
2. Dijkstra's algorithm maintains a priority queue that contains vertices that have not yet been added to the shortest path tree rooted at s . The priority of such a vertex is the shortest path length from s to the vertex that uses edges of the shortest path tree plus one additional edge not in the shortest path tree. I claim that the number of different priorities of vertices in the priority queue is at most $W + 2$. Suppose p is the minimum priority in the queue. Every vertex in the shortest path tree has a shortest path length from s that is at most p . (Why?) Since every edge has weight in $\{0, 1, \dots, W\}$, every vertex in the queue that can be reached from the tree by an edge not in the tree has priority at most $p + W$. Those that cannot be reached have priority ∞ . That is at most $W + 2$ different priorities.

Since the priorities less than ∞ are integers, we can use an array Q of linked lists of size $W + 1$ to implement the priority queue. At any point, we have an index i that indicates which array entry holds the minimum priority vertices. Initially $i = 0$. When `DELETEMIN` is performed, we check if $Q[i]$ is empty. If it is we increment i until we find a non-empty $Q[i]$ (or stop if all are empty). We then remove and return one of the vertices in the linked list in $Q[i]$. When `INSERT` or `UPDATEPRIORITY` are performed, we store the vertex v and its priority p into array location $Q[p \bmod (W + 1)]$. Each of these operations takes $O(W)$ time, leading to a total running time of $O(Wn + m)$ for Dijkstra's algorithm.