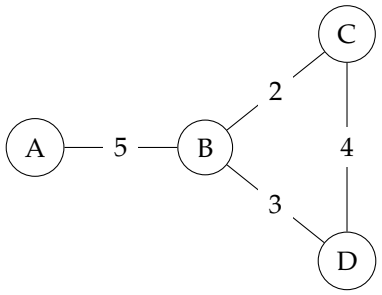
Solution 1. The algorithm is incorrect. Consider the following instance:



Suppose we are after a binary tree rooted at B. The algorithm will add the edges (B, C) and (D, B) into the tree and then declare the instance infeasible because there is no edge that makes the condition in Line 6 true (at this point $R = \{A\}$ and the only edge incident on A connects it to B, which already has two children).

However, the instance is not infeasible. In fact, the optimal spanning binary tree is $\{(A, B), (B, C), (C, D)\}$.

Solution 2.

a) We reduce the problem of finding the cheapest travel plan to a shortest path problem in a suitable directed graph $G = (V, E)$ with edge weights w. We use the adjacency list representation for the graph.

The vertex set V is created as follows. Let (s, t_1, t_2) be a triplet in the timetable of some instance of a bus line, where s is the stop, t_1 is the time the bus arrives at the stop and t_2 is the time it departs from the stop. For each such triplet, we include two vertices u_{s,t_1} and u_{s,t_2} in V. Additionally, we also create two vertices $u_{a,t}$ and $u_{b,t'}$ where a is the origin bus stop and t is the departure time, and b is the destination bus stop and t' is the arrival time. These vertices are going to be used to represent that we are at a given stop at a given point in time.

The edge set E is created as follows. First, we have "wait edges" connecting any two vertices of the forms u_{s,t_1} and u_{s,t_2} with $t_1 < t_2$; the weight of these edges is 0. Second, we have "travel edges" connecting any two vertices of the form $u_{s,r}$ and $u_{s',r'}$ with $r < r'$ if there exists a bus instance with a time table having triples of the form $(s, *, r)$ and $(s', r', *)$; the weight of these edges is the cost of the fare of the bus.

Given a $u_{a,t} - u_{b,t'}$ path P we can extract a travel plan as follows: Scanning the edges $(u_{s,r}, u_{s',r'}) \in P$ from start to finish if the edge is a travel edge, we add (s', r') to the travel plan.

To find the cheapest plan we find the shortest path in G from $u_{a,t}$ to $u_{b,t'}$ and extract the travel plan from this path.

b) The proof of correctness rests on the correspondence between travel plans and paths in G. Given a travel plan, we can create a $u_{a,t} - u_{b,t'}$ path in G whose weight equals the cost of the travel plan; for example, two consecutive pairs (s, r) and (s', r') in the travel plan mean there is a bus instance that we can catch at s, say at time r'', that will arrive at s' at r', which maps to the sequence of edges

$u_{s,r}$, $u_{s,r''}$, and $u_{s',r'}$. Conversely, given a path $u_{a,t} - u_{b,t'}$ we can extract a travel plan of the same cost with the above described procedure.

Therefore, the travel plan associated with the shortest path in G is cheapest.

c) First we bound the space complexity of G. The graph has $O(mk\ell)$ vertices, and at most $O((mk\ell)^2)$ edges. To build the graph, we first do a scan of all the triples in the instance to create the vertices in $O(mk\ell)$ time (two vertices per triplet). To create the wait edges, we divide the vertices into groups by the stop the involve, and then add edges pointing forward in time; this can be done in less than $O((mk\ell)^2)$ time. To create the wait edges we iterate over pairs of triplets of a given instances, and create the necessary travel edges in $O((mk\ell)^2)$ time.

After the graph is built we run Dijkstra in $O((mk\ell)^2)$ time and extract the travel plan in linear time on the length of the path.

Solution 3. a) We reduce the problem of finding the cheapest traversal to a shortest path problem in a suitable undirected graph $G = (V, E)$ with edge weights w. We use the adjacency list representation for the graph.

The vertex set V is created as follows. For each shape S in the input we create a vertex u_S.

The edge set E is created as follows. For every pair of shapes S and T, if S and T overlap and at least one of them is a circle, we add the edge (u_S, u_T) to our graph and we set its weight to be $\frac{\text{area}(S)+\text{area}(T)}{2}$.

Every path $u_{S_1}, u_{S_2}, \ldots, u_{S_k}$ induces a traversal $S_1, S_2, \ldots, S_k$ in our input instance. Our algorithm uses Dijkstra to find a shortest path from u_A to u_B and then returns the traversal associated with this path.

b) For the correctness we show that the return traversal is valid that has minimum cost. First, we argue that it is valid. Indeed, any path $u_{S_1}, u_{S_2}, \ldots, u_{S_k}$ in the graph induces a traversal $S_1, S_2, \ldots, S_k$ such that every two consecutive shapes overlap and we do not have two consecutive squares; otherwise the corresponding vertices would not be connected with an edge, which is required to have a path. It is important to note that a valid traversal induces a path on the corresponding vertices.

To show that it is minimum cost we first observe that the weight of a path $u_{S_1}, u_{S_2}, \ldots, u_{S_k}$ is

$$\sum_{u_S \in P} \text{area}(S) - \frac{\text{area}(S_1) + \text{area}(S_k)}{2}.$$

Because we are looking for paths that go from u_A to u_B, every u_A-u_B is off by the same additive factor of $\frac{\text{area}(u_A)+\text{area}(u_B)}{2}$ from the cost of the corresponding traversal. Therefore, minimizing the cost of the traversal is equivalent to minimizing the cost of the traversal.

c) The algorithm runs in $O(n^2)$. To argue this we first bound the complexity of G: $|V| = n$ and $|E| \leq n^2$. Creating G takes $O(n^2)$ time: We iterate over pairs of shapes, and for each we determine in $O(1)$ if we need to add an edge in the graph. Running Dijkstra's on G takes $O(|E| + |V| \log |V|)$ time, which in our case is $O(n^2)$.