Problem 1. Use the linear-time dynamic-programming algorithm to compute the shortest-path distance from $v_{0,0}$ to each node in the grid below, using the given edge weights. For your answer, give a 4×6 matrix where the entry in row i and column j is the distance from $v_{0,0}$ to $v_{i,j}$.

$$
\begin{array}{ccccccccccc}
v_{3,0} & \xrightarrow{3} & v_{3,1} & \xrightarrow{1} & v_{3,2} & \xrightarrow{4} & v_{3,3} & \xrightarrow{1} & v_{3,4} & \xrightarrow{5} & v_{3,5} \\
4\big\uparrow & & 5\big\uparrow & & 9\big\uparrow & & 2\big\uparrow & & 1\big\uparrow & & 1\big\uparrow \\
v_{2,0} & \xrightarrow{4} & v_{2,1} & \xrightarrow{0} & v_{2,2} & \xrightarrow{2} & v_{2,3} & \xrightarrow{1} & v_{2,4} & \xrightarrow{3} & v_{2,5} \\
2\big\uparrow & & 1\big\uparrow & & 5\big\uparrow & & 7\big\uparrow & & 3\big\uparrow & & 1\big\uparrow \\
v_{1,0} & \xrightarrow{4} & v_{1,1} & \xrightarrow{5} & v_{1,2} & \xrightarrow{9} & v_{1,3} & \xrightarrow{2} & v_{1,4} & \xrightarrow{1} & v_{1,5} \\
3\big\uparrow & & 1\big\uparrow & & 4\big\uparrow & & 1\big\uparrow & & 5\big\uparrow & & 9\big\uparrow \\
v_{0,0} & \xrightarrow{2} & v_{0,1} & \xrightarrow{1} & v_{0,2} & \xrightarrow{5} & v_{0,3} & \xrightarrow{7} & v_{0,4} & \xrightarrow{3} & v_{0,5}
\end{array}
$$

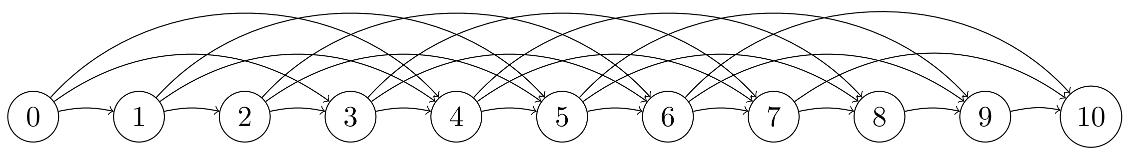
Problem 1 answer Here's the matrix of distances from $v_{0,0}$:

	$j = 0$	1	2	3	4	5
$i = 3$	9	9	10	8	8	11
2	5	4	4	6	7	10
1	3	3	7	9	11	12
0	0	2	3	8	15	18

Problem 2. Consider making change for total $T = 10$ using denominations $D = \{1, 3, 4\}$.

(a) Simulate the iterative makeChange algorithm. What are the values in the array $m[0..T]$ computed by the algorithm?

(b) Following Section 3.1, the corresponding DAG $G = (V, E)$ is shown below. What path in this DAG corresponds to making change with coins $4, 3, 1, 1, 1$ (in that order)? List the six nodes on the path.



Problem 2 answer

(a) Here's the matrix $m[0..10]$ computed by makeChange($D = \{1, 3, 4\}, T = 10$):

	$t = 0$	1	2	3	4	5	6	7	8	9	10
$m[t]$	0	1	2	1	1	2	2	2	2	3	3

(b) Here are the six nodes on the path corresponding to making change with $4, 3, 1, 1, 1$:

$$0, 4, 7, 8, 9, 10$$

Problem 3. Consider the following problem, Counting Ways to Make Change:

input: A pair $I = (D, T)$ where $D = \{d_1, d_2, \ldots, d_k\}$ is a set of *denominations*, and T is a target (all non-negative integers, with $1 \in D$).

output: The number of sequences $C = (c_1, \ldots, c_\ell)$ s.t. $c_i \in D$ for all i and $\sum_i c_i = T$.

For example, with $D = \{1, 3\}$ and $T = 4$, there are three: $(1, 1, 1, 1)$, $(1, 3)$, and $(3, 1)$.
Adapt the algorithm for Making Change to solve this problem.

(a) Define the subproblems.

(b) State the recurrence relation.

(c) Give pseudo-code for an iterative implementation of the algorithm.

(continued)

Problem 3 answer [1]

(a) Here is a definition of all the subproblems the algorithm will solve:
For $t \in \{0, 1, \ldots, T\}$, define $N(t)$ to be the number of sequences that can be used to make change for t. The final answer is $N(T)$.

(b) Here's the recurrence relation:

$$N(t) = \begin{cases} 1 & \text{if } t = 0 \\ \sum_i N(t - d_i) : d_i \in D,\ d_i \leq t & \text{otherwise.} \end{cases}$$

(c) Here is pseudo-code for an iterative implementation (don't forget the return statement):

waysToMakeChange($D = \{d_1, d_2, \ldots, d_k\}, T$):
1. For $t = 0, 1, \ldots, T$:
2. $N[t] = \begin{cases} 1 & \text{if } t = 0 \\ \sum_i N(t - d_i) : d_i \in D,\ d_i \leq t & \text{otherwise.} \end{cases}$
3. return $N[T]$

[1] MAKE SURE TO READ THE COMMENTS IN problem_3_answer.tex

Problem 4. Is the following conjecture true or false? Either give a long-form proof of the conjecture (as a theorem), or prove that it is false by giving a counter-example, and explaining clearly how the algorithm fails on your counter-example.

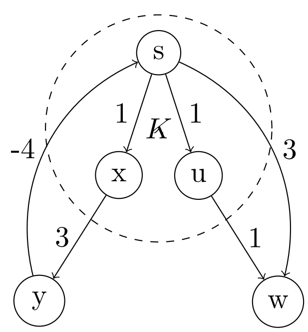
Conjecture 1. *Let $G = (V, E)$ be any edge-weighted digraph, and $s \in V$ any vertex. Suppose that every edge that doesn't touch s has non-negative weight. (That is, for every edge $(u, w) \in E$, if $s \notin \{u, w\}$, then $\mathsf{wt}(u, w) \geq 0$. Edges entering or leaving s can have negative weight.) Suppose also that G has no negative-weight cycles. Then Dijkstra's algorithm, if run on input (G, s), correctly computes the shortest-path distance from s to each vertex $v \in V$.*

If you try to prove it, your proof should be *as similar as possible* to the proof of correctness for Dijkstra's algorithm. We'll provide that proof as a template. Please color all text that you add to it blue.

For reference, here is the algorithm:

Dijkstra$\big(G = (V, E), s\big)$: — *Dijkstra's algorithm for Single-Source Shortest Paths* —
1. initialize $K \leftarrow \{s\}$ and $d[s] = 0$
2. while some edge in E leaves K (that is, $\exists (u, w) \in E : u \in K,\ w \notin K$):
3.1. among edges leaving K, choose edge (u', w') that minimizes $d[u'] + \mathsf{wt}(u', w')$
3.2. set $d[w'] = d[u'] + \mathsf{wt}(u', w')$
3.3. add w' to K
4. set $d[w] = \infty$ for all $w \in V \setminus K$
5. return d

Note that in the proof of Lemma 1, in Step 3.12.5, it can happen that $\mathsf{wt}(P_y) < 0$. Consider the graph below, with $K = \{s, x, u\}$ and path $P = (s, x, y, s, w)$. (And $d[s] = 0$, $d[u] = d[x] = 1$.)



Problem 4 answer

Lemma 1. *In any execution on a graph as described in the problem statement, the loop maintains the invariant* $d[v] = \mathsf{dist}(s, v) < \infty$ for all $v \in K$.

Proof (long form).

1. Consider any execution of the algorithm on some input $(G = (V, E), s)$.
2. The invariant holds at the start of the first iteration, when $K = \{s\}$ and $d[s] = 0 = \mathsf{dist}(s, s)$. (Because G has no negative-weight cycles.)

3.1. Consider any iteration of the loop such that the invariant holds at the start of the iteration.
3.2. Let K and d denote the set K and the array d at the *start* of the iteration.
3.3. Let (u', w') be the edge chosen in the iteration.
3.4. To show that the iteration maintains the invariant, we'll show $\mathsf{dist}(s, w') = d[u'] + \mathsf{wt}(u', w')$.
3.5. First we show $\mathsf{dist}(s, w') \leq d[u'] + \mathsf{wt}(u', w')$.
3.6. Since $u' \in K$, by the invariant, $\mathsf{dist}(s, u') = d[u'] < \infty$.
3.7. So there exists a path from s to u' of weight $d[u']$.
3.8. This path plus the edge (u', w') form a path from s to w' of weight $d[u'] + \mathsf{wt}(u', w')$.
3.9. So there exists a path from s to w' of weight $d[u'] + \mathsf{wt}(u', w')$.
3.10. So $\mathsf{dist}(s, w') \leq d[u'] + \mathsf{wt}(u', w')$, as desired.
3.11. To finish we'll show $\mathsf{dist}(s, w') \geq d[u'] + \mathsf{wt}(u', w')$.
3.12.1. Let P be any path from s to w'. We'll show $\mathsf{wt}(P) \geq d[u'] + \mathsf{wt}(u', w')$.
3.12.2. P starts in K, but ends outside of K, so some edge on P leaves K.
3.12.3. Let (x, y) be an edge on P that leaves K (so $x \in K$, $y \notin K$).
3.12.4. Separate the path P around the edge (x, y) into the part before, P_x, and the part after, P_y.
3.12.5. So $P = P_x \cup \{(x, y)\} \cup P_y$ and P_x is a path from s to x. Then
3.12.6. Case 1: P_y doesn't go through S.
P_y has no edge incident to S, therefore, $\mathsf{wt}(P_y) \geq 0$, as only edges incident to S have negative edge weight.

$$
\begin{aligned}
&= \mathsf{wt}(P_x) + \mathsf{wt}(x, y) + \mathsf{wt}(P_y) && \textit{(Since } P = P_x \cup \{(x, y)\} \cup P_y\textit{.)} \\
&\geq \mathsf{wt}(P_x) + \mathsf{wt}(x, y) && (\mathsf{wt}(P_y) \geq 0.) \\
&\geq \mathsf{dist}(s, x) + \mathsf{wt}(x, y) && \textit{(Since } P_x \textit{ is a path from } s \textit{ to } x.\textit{)} \\
&= d[x] + \mathsf{wt}(x, y) && \textit{(Since } x \in K \textit{ so } d[x] = \mathsf{dist}(s, x) \textit{ by the invariant.)} \\
&\geq d[u'] + \mathsf{wt}(u', w') && \textit{(By the alg's choice of } (u', w')\textit{, and } x \in K,\ y \notin K\textit{)}
\end{aligned}
$$

3.12.7. Case 2: P_y goes through S.
3.12.8. Thus, there is a cycle from S through (x, y) back to S (the last time it loops around, if it loops around more than once to S), and the rest of path P is from S to w'.
3.12.9. $P_y = P_{y->s} + P_{s->w'}$
3.12.10. $\mathsf{wt}(P_{y->s}) + \mathsf{wt}((x, y)) + \mathsf{wt}(P_x) \geq 0 because \mathrm{P}_{y->s} + (x, y) + P_x$ is a cycle

$$= \mathsf{wt}(P_x) + \mathsf{wt}(x, y) + \mathsf{wt}(P_y) \qquad \textit{since } P = P_x \cup \{(x, y)\} \cup P_y$$

$$= \mathsf{wt}(P_x) + \mathsf{wt}(x, y) + \mathsf{wt}(P_{y->s}) + \mathsf{wt}(P_{s->w'}) \qquad \textit{step 3.12.9}$$

(cont.)

$\geq$ $\mathsf{wt}(P_{s->w'})$

$\mathsf{wt}(P_{y->s}$ + (x,y) + $P_x) \geq 0$

Case 2.1: the path from s to w' goes through u'

$\geq \mathsf{wt}(P_{s->w'})$

from above

$\geq d[u'] + \mathsf{wt}(u', w')$

since $d[u'] + \mathsf{wt}(u', w')$ is the least weight path from s to w' chosen by the algorithm

Case 2.2: the path s to w' goes through node z to reach w' and y is the node connected to w' in this path y is the node connected to w' in this path and the path leaves K through edge (i,j)

Case 2.2.1: z is in K

$\geq \mathsf{wt}(P_{s->w'})$

from above

$= \mathsf{wt}(P_{s->z}) + \mathsf{wt}(P_{z->i}) + \mathsf{wt}((i,j)) + \mathsf{wt}(P_{j->y}) + \mathsf{wt}(P_{y->w'})$

$\geq d[u'] + \mathsf{wt}(u', w')$

for all a in K and (a,b) leaving K, the algorithm chooses the lowest result of d[a] + wt(a,b) leaving K, so it must have considered both (u',w') and (i,j), so d[i'] + wt(i',j') >= d[u'] + wt(u',w') and any other edges such as (y,w') are not incident to k, so their edge weight is positive and can only increase the weight of the path) so wt(P) >= d[u'] + wt(u',w')

Case 2.2.2: z is not in K

$\geq \mathsf{wt}(P_{s->w'})$

from above

$= \mathsf{wt}(P_{s->i}) + \mathsf{wt}((i,j)) + \mathsf{wt}(P_{j->z}) + \mathsf{wt}(P_{z->y}) + \mathsf{wt}(P_{y->w'})$

$\geq d[u'] + \mathsf{wt}(u', w')$

same reasoning as case 2.2.1

3.13. By Block 3.12, every path from s to w' has weight at least $d[u'] + \mathsf{wt}(u', w')$.
3.14. That is, $\mathsf{dist}(s, w') \geq d[u'] + \mathsf{wt}(u', w')$.
3.15. By this and Step 3.10, $\mathsf{dist}(s, w') = d[u'] + \mathsf{wt}(u', w')$.
4. The invariant holds initially. By Block 3, each iteration maintains it, so it holds throughout. $\square$

Given that the invariant holds, correctness is easy to verify:

Theorem 1. *Given any graph as described in the problem statement, the array d returned by Dijkstra's algorithm satisfies $d[v] = \mathsf{dist}(s, v)$ for all $v \in V$.*

Proof (long form).

1. Consider the execution of the algorithm on an arbitrary input (G, s).
2. Consider the time just before Line 4 executes.
3. By Lemma 1, at that time, $d[v] = \mathsf{dist}(s, v)$ for each $v \in K$.
4. By the loop condition, no edges leave K, so vertices in $V \setminus K$ are not reachable from s.
5. So each remaining vertex $v \in V \setminus K$ has $\mathsf{dist}(s, v) = \infty$.
6. So, after Line 4 executes, each vertex $v \in V \setminus K$ also has $d[v] = \mathsf{dist}(s, v)$.
7. So the distances returned by the algorithm are correct. $\square$