

General instructions. Answer the problems given in the rest of this document. Edit this LaTeX template to add your answers, as described in the file named `00_README.txt`, then compile it to produce `homework_7.pdf`.

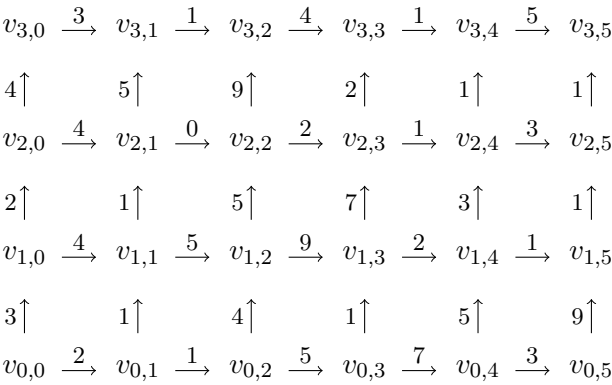
Don't know? Say so. If you don't know the answer to a problem or any part of the problem, we encourage you to simply write “*I don't know*” instead of giving an answer. (If you write “I don't know” you can also add any questions you might have.)

Gentle reminder: homework is for learning. As noted in [Lecture Note 1](#), engaging fully with the homeworks is the best way to learn the material. *The primary role of homeworks is to help you learn the material, and the primary role of feedback on homeworks is to help you improve your ideas.*

Please try to explain your ideas as clearly as you can. The clearer your explanations are, the more useful the feedback on them can be.

Collaboration and citation policy. If you are doing this homework as part of a course, follow the course's collaboration and citation policy. In any case, at the end of each answer, list the people you collaborated with, and cite every source that your answer uses ideas from (other than the lecture notes and the sources listed in the lecture notes). If there are none, write “none”.

Problem 1. (Section 7.2.2) Use the linear-time dynamic-programming algorithm in Section 7.2.2 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 $v_{i,j}$.



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

	$j = 0$	1	2	3	4	5
$i = 3$	★...★	★...★	★...★	★...★	★...★	★...★
2	★...★	★...★	★...★	★...★	★...★	★...★
1	★...★	★...★	★...★	★...★	★...★	★...★
0	0	2	★...★	★...★	★...★	★...★

Problem 1 collaboration and citations

I collaborated on this problem with the following people:

- ★...★

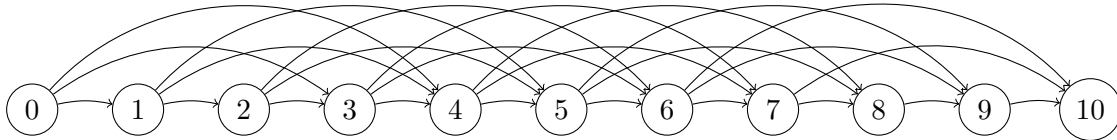
My answers use ideas from the following sources (other than the lecture notes and textbooks):

- ★...★

Problem 2. (Section 7.3)

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

- Simulate the iterative `makeChange` algorithm in Section 7.3 on this input. What are the values in the array $m[0..T]$ computed by the algorithm?
- Following Section 7.3.1, 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**

- 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	★...★	★...★	★...★	★...★	★...★	★...★	★...★	★...★	★...★	★...★

- Here are the six nodes on the path corresponding to making change with 4, 3, 1, 1, 1:

0, ★...★, ★...★, ★...★, ★...★, 10

Problem 2 collaboration and citations

I collaborated on this problem with the following people:

● ★...★

My answers use ideas from the following sources (other than the lecture notes and textbooks):

● ★...★

Problem 3. (Section 7.3) Consider the following problem, Counting Ways to Make Change:

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

output: The number of sequences $C = (c_1, \dots, c_\ell)$ 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 from Section 7.3 to solve this problem. Use the first two steps from Section 7.4 to define your algorithm:

- (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:

★...★

(b) Here's the recurrence relation:

★...★

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

★...★

Problem 3 collaboration and citations

I collaborated on this problem with the following people:

● ★...★

My answers use ideas from the following sources (other than the lecture notes and textbooks):

● ★...★

¹MAKE SURE TO READ THE COMMENTS IN `problem_3_answer.tex`

Problem 4. (Exercise 5.10) 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 $\text{wt}(u, w) \geq 0$. 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 $v \in V$.

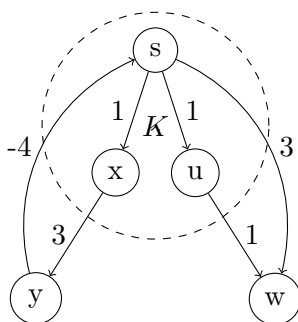
If you try to prove it, your proof should be *as similar as possible* to the proof of correctness for Dijkstra's algorithm in Section 5.8. (Your score will depend on how similar your proof is.) We'll provide that proof as a template. Please color all text that you add to it blue. You may use Observation 5.7 (Section 5.8) without proof—but if you do, cite it where you use it.

For reference, here is the algorithm:

Dijkstra($G = (V, E), s$): — 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 $d[u'] + \text{wt}(u', w')$
 - 3.2. set $d[w'] = d[u'] + \text{wt}(u', w')$
 - 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 $\text{wt}(P_y) < 0$. 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 collaboration and citations I collaborated on this problem with the following people:

● ★ ⋯ ★

My answers use ideas from the following sources (other than the lecture notes and textbooks):

● ★ ⋯ ★

Problem 4 answer

Lemma 1. *In any execution on a graph as described in the problem statement, the loop maintains the invariant $d[v] = \text{dist}(s, v) < \infty$ 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 $d[s] = 0 = \text{dist}(s, s)$.
(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 $\text{dist}(s, w') = d[u'] + \text{wt}(u', w')$.
- 3.5. First we show $\text{dist}(s, w') \leq d[u'] + \text{wt}(u', w')$.
- 3.6. Since $u' \in K$, by the invariant, $\text{dist}(s, u') = d[u'] < \infty$.
- 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 $d[u'] + \text{wt}(u', w')$.
- 3.9. So there exists a path from s to w' of weight $d[u'] + \text{wt}(u', w')$.
- 3.10. So $\text{dist}(s, w') \leq d[u'] + \text{wt}(u', w')$, as desired.
- 3.11. To finish we'll show $\text{dist}(s, w') \geq d[u'] + \text{wt}(u', w')$.
- 3.12.1. Let P be any path from s to w' . We'll show $\text{wt}(P) \geq d[u'] + \text{wt}(u', w')$.
- 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

$\text{wt}(P) = \text{wt}(P_x) + \text{wt}(x, y) + \text{wt}(P_y)$	<i>(Since $P = P_x \cup \{(x, y)\} \cup P_y$.)</i>
$\geq \text{wt}(P_x) + \text{wt}(x, y)$	<i>(As all edge weights are non-negative, so $\text{wt}(P_y) \geq 0$.)</i>
$\geq \text{dist}(s, x) + \text{wt}(x, y)$	<i>(Since P_x is a path from s to x.)</i>
$= d[x] + \text{wt}(x, y)$	<i>(Since $x \in K$ so $d[x] = \text{dist}(s, x)$ by the invariant.)</i>
$\geq d[u'] + \text{wt}(u', w')$	<i>(By the alg's choice of (u', w'), and $x \in K, y \notin K$)</i>
- 3.13. By Block 3.12, every path from s to w' has weight at least $d[u'] + \text{wt}(u', w')$.
- 3.14. That is, $\text{dist}(s, w') \geq d[u'] + \text{wt}(u', w')$.
- 3.15. By this and Step 3.10, $\text{dist}(s, w') = d[u'] + \text{wt}(u', w')$.
4. The invariant holds initially. By Block 3, each iteration maintains it, so it holds throughout. $\square$

(continued)

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 $d[v] = \text{dist}(s, v)$ 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, $d[v] = \text{dist}(s, v)$ 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 $\text{dist}(s, v) = \infty$.
6. So, after Line 4 executes, each vertex $v \in V \setminus K$ also has $d[v] = \text{dist}(s, v)$.
7. So the distances returned by the algorithm are correct. □