

Algorithm Design Manual Exercises

Zachary William Grimm

Solutions to Selected Problems

zwgrimm@gmail.com

February 10, 2019

Contents

1	Introduction To Algorithm Design	2
1.1	Robot Tour Optimization	2
1.2	Selecting the Right Jobs	2
1.2.1	test, delete this	2
1.3	Reasoning about Correctness	2
1.3.1	Expressing Algorithms	2

1 Introduction To Algorithm Design

1.1 Finding Counter Examples

Ex:1-1. Show that $a + b$ can be less than $\min(a, b)$

Let $a = -1, b = -1$

Then $a + b = -2, \min(a, b) = -1$

$\therefore \exists a, b \in \mathbb{Z} : a + b < \min(a, b)$ ■

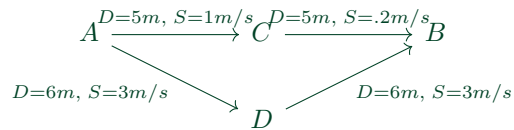
Ex:1-2. Show that $a * b$ can be less than $\min(a, b)$

Let $a = -1, b = 5$.

Then $a * b = -5, \min(a, b) = -1$

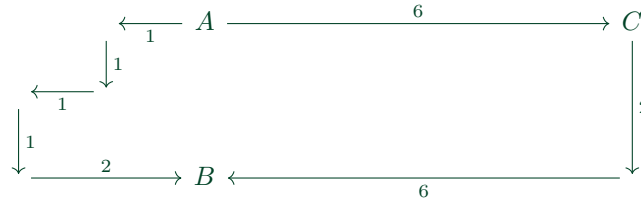
$\therefore \exists a, b \in \mathbb{Z} : a * b < \min(a, b)$ ■

Ex:1-3. Design/draw a road network with two points a and b such that the fastest route between a and b is not the shortest route.



Although the distance from A to B through C is shorter than going through D, road constraints limit the time it takes making the route through D faster despite it. ■

Ex:1-4. Design/draw a road network with two points a and b such that the shortest route between a and b is not the route with the fewest turns.



The route from A through C to B has only two turns but is a total length of 14 units while the direct route from A to B (the shortest) has 4 turns and is a length of 6 units. \therefore The shortest route between A and B is not the route with the fewest turns. The route from A through C to B has only two turns but is a total length of 14 units while the direct route from A to B (the shortest) has 4 turns and is a length of 6 units. \therefore The shortest route between A and B is not the route with the fewest turns. ■

Ex:1-5. The knapsack problem is as follows: Given a set of integers $S = \{s_1, s_2, \dots, s_n\}$, and a target number T , find a subset of S which adds up exactly to T . For example, there exists a subset within $S = \{1, 2, 5, 9, 10\}$ that adds up to $T = 22$ but not $T = 23$. Find counterexamples to each of the following algorithms for the knapsack problem. That is, giving an S and T such that the subset is selected using the algorithm does not leave the knapsack completely full, even though such a solution exists.

- (a) Put the elements of S in the knapsack in left to right order if they fit, i.e. the first-fit algorithm.

Let $S = \{1, 7, 9\}$, $T = 10$

- (b) *Put the elements of S in the knapsack from smallest to largest, i.e. the best-fit algorithm.*

Let $S = \{1, 7, 9\}$, $T = 10$

- (c) *Put the elements of S in the knapsack from largest to smallest.*

Let $S = \{1, 4, 5, 7, 9\}$, $T = 19$ ■

Ex:1-6. The set cover problem is as follows: Given a set of subsets S_1, \dots, S_m of the universal set $U = \{1, \dots, n\}$, find the smallest subset of subsets $T \subset S$ such that $\cup_{t_i \in T} t_i = U$. For example, there are the following subsets, $S_1 = \{1, 3, 5\}$, $S_2 = \{2, 4\}$, $S_3 = \{1, 4\}$, and $S_4 = \{2, 5\}$. The set cover would then be S_1 and S_2 . **Find a counterexample for the following algorithm:** Select the largest subset for the cover, and then delete all its elements from the universal set. Repeat by adding the subset containing the largest number of uncovered elements until all are covered.

$$\begin{aligned} U &= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}, \\ S_1 &= \{1, 2, 3, 4, 5\}, S_2 = \{6, 7, 8, 9\}, S_3 = \{6, 7, 10\}, S_4 = \{8, 9, 11\} \\ U' &= \{6, 7, 8, 9, 10, 11\}, T' = S_1 \end{aligned}$$

The Set with largest number of uncovered elements is S_2 . S_3 and S_4 must also be included in the set cover because they contain elements (10 & 11 respectively) in U that are not members of any other subset. Therefore this algorithm gives us as the set cover all of the subsets, but $S_1 \cup S_3 \cup S_4$ covers U and is smaller than $S_1 \cup S_2 \cup S_3 \cup S_4$ ■

1.2 Proofs of Correctness

1-7. Prove the correctness of the following recursive algorithm to multiply two natural numbers, for all integer constants $c \geq 2$:

```
function M(y,z)
1. If z = 0 then return(0) else
2. return (cy, ⌊(z/c)⌋) + y(z mod c))
```

case 1: $c = z$

$$\begin{aligned} M(y, z) &= M(cy, \lfloor (z/c) \rfloor) + y(z \bmod c) \\ &= M(cy, \lfloor (z/z) \rfloor) + y(z \bmod z) \\ &= M(cy, 1) + y(z \bmod z) \\ &= M(cy, 1) \\ &= M(czy, \lfloor (1/c) \rfloor) + cy(1 \bmod c) \\ &= M(czy, \lfloor (1/z) \rfloor) + cy(1 \bmod z) \\ &= M(czy, 0) + cy(1 \bmod z) \\ &= cy \end{aligned}$$

case 2: $c > z$

$$\begin{aligned} M(y, z) &= M(cy, \lfloor (z/c) \rfloor) + y(z \bmod c) \\ &= M(cy, 0) + y(z \bmod c) \\ &= M(cy, 0) + y(z \bmod c) \\ &= yz \end{aligned}$$

case 3: $c < z$

Assumptions:

$$\begin{aligned} c &\geq 2 \\ z &\leq n \text{ (inductive hypothesis)} \\ y &\geq 0 \end{aligned}$$

Base Case:

$$z = 0, M(y, 0) = 0, \text{ (which is true)}$$

Lemma:

we show that

$$\lfloor (z/c) \rfloor * c + (z \bmod c) = z$$

by the quotient remainder theorem

$$\begin{aligned} z &= cq + r \\ &= cq + z \bmod c \\ &= \lfloor (z/c) \rfloor * c + (z \bmod c) \quad (1*) \end{aligned}$$

Assuming the algorithm holds for all numbers $\leq n$, we must show that

$$M(y, n+1) = y(n+1)$$

Now,

$$M(y, n+1) = M(cy, \lfloor ((n+1)/c) \rfloor) + y((n+1) \bmod c)$$

since

$$c \geq 2, \\ \lfloor ((n+1)/c) \rfloor < n+1$$

\therefore the first term returns a valid result (based on our inductive hypothesis) so following the algorithm:

$$M(y, n+1) = M(cy, \lfloor ((n+1)/c) \rfloor) + y((n+1) \bmod c)$$

for simplicity let $z' = n+1$

$$= cy \lfloor ((z')/c) \rfloor + y((z') \bmod c) \\ = y(c \lfloor (z'/c) \rfloor + (z' \bmod c))$$

and from (1*)

$$= yz' \quad \blacksquare$$

1-8. *Prove the correctness of the following algorithm for evaluating a polynomial:*

$$P(x) = a_n + a_{n-1}x + \dots + a_1x + a_0$$

```
function horner(A, x):
  p = An
  for i from n - 1 to 0
    p = px + Ai
  return p
```

For polynomials of degree 0, $P(x) = A_0$, which the algorithm satisfies. Assuming the algorithm holds for polynomials of degree $\leq n$ and that A is an ordered set of coefficients of size n , $[A_n, A_{n-1}, \dots, A_0]$.

$$\text{horner}(A, x) = P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

we must show it holds for polynomials of degree $n + 1$, ie: A' is an ordered set of coefficients of size $n + 1$, $([A_{n+1}, A_n, \dots, A_0])$

$$\begin{aligned} & \text{horner}(A', x) : \\ & p \Rightarrow \\ & A'_{n+1} \\ & A'_{n+1} * x + A'_n \\ & A'_{n+1} * x^2 + A'_n * x + A'_{n-1} \\ & \cdot \\ & \cdot \\ & \cdot \\ & A'_{n+1} * x^{n+1} + A'_n * x^n + \dots + A'_1 * x + A'_0 \\ & = \\ & A'_{n+1} * x^{n+1} + \text{horner}(A, x) \blacksquare \end{aligned}$$

1-9. Prove the correctness of the following sorting algorithm:

```
function bubblesort(A : list[1 ... n])
var int i, j
for i from n to 1
for j from 1 to i - 1
if A[j] > A[j + 1]
swap the values of A[j] and A[j + 1]
```

Our Base Case is a list of two elements $[a, b]$ (we are indexing from 1 not 0 as usual). The base case itself has several cases:

case 2: $a > b$

```

i = 2
j = 1
if (A[1] > A[2]) = true
swap values, so A = [b, a]
i = 1
j = 1
if (A[1] > A[2]) = false
so A = [b, a] and from our assumption that a > b for case 1 the algorithm is true.

```

case 2: $a < b$

```

i = 2
j = 1
if (A[1] > A[2]) = false
swap values, so A = [b, a]
i = 1
j = 1
if (A[1] > A[2]) = false
so A = [a, b] and from our assumption that a < b for case 2 the algorithm is true.

```

case 3: $a = b$

The list is considered sorted regardless of how the algorithm may rearrange its items.

Now we proceed with **induction**, assuming that for all lists of size $\leq n$ the algorithm returns a sorted list. We must show that for a list of size $n + 1$, the algorithm returns an ordered list.

```

function bubblesort(A : list[1...n, n + 1])
var int i, j
for i from n+1 to 1
for j from 1 to i - 1
if A[j] > A[j + 1]
swap the values of A[j] and A[j + 1]

```

The inner loop "bubbles up" the largest num to position i on the last j iteration of the first i iteration the altered set has $n + 1$ elems with the largest element in position n+1 now we have sort the rest of the list which

is of size n . Our inductive hypothesis states we can do this. ■

1.3 Induction

1-10. Prove that $\sum_{i=1}^n i = \frac{n(n+1)}{2}$ for $n \geq 0$, by induction.

Let $n = 1$. Then

$$\frac{n(n+1)}{2} = 1 = \sum_{i=1}^1 i$$

establishing our base case. We assume that $\exists n \geq 0 \in \mathbb{N}$ where

$$\sum_{i=1}^n i = \frac{n(n+1)}{2} \text{ for } n \leq \text{some } k \in \mathbb{N}$$

We must show

$$\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}$$

We can see that

$$\begin{aligned} \sum_{i=1}^{n+1} i &= (n+1) + \sum_{i=1}^n i \\ &= (n+1) + \frac{n(n+1)}{2} \\ &= \frac{[2(n+1) + n(n+1)]}{2} \\ &= \frac{(n+1)(n+2)}{2} \quad \blacksquare \end{aligned}$$

1-11. Prove that $\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$ for $n \geq 0$, by induction.

Let $n = 1$. Then

$$\frac{n(n+1)(2n+1)}{6} = 1 = \sum_{i=1}^1 i^2$$

establishing our base case. We assume that $\exists n \geq 0 \in \mathbb{N}$ where

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6} \text{ for } n \leq \text{some } k \in \mathbb{N}$$

We must show

$$\sum_{i=1}^{n+1} i^2 = \frac{(n+1)(n+2)(2n+3)}{6}$$

We can see that

$$\begin{aligned} \sum_{i=1}^{n+1} i^2 &= (n+1)^2 + \sum_{i=1}^n i^2 \\ &= (n+1)^2 + \frac{n(n+1)(2n+1)}{6} \\ &= \frac{(n+1)[6(n+1) + n(2n+1)]}{6} \\ &= \frac{(n+1)[2n^2 + 7n + 6]}{6} \\ &= \frac{(n+1)(n+2)(2n+3)}{6} \blacksquare \end{aligned}$$

1-12. Prove that $\sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}$ for $n \geq 0$, by induction

1-13. Prove that $\sum_{i=1}^n i(i+1)(i+2) = \frac{n(n+1)(n+2)(n+3)}{4}$

1-14. Prove by induction on $n \geq 1$ that for every $a \neq 1$, $\sum_{i=0}^n a^i = \frac{a^{n+1}-1}{a-1}$

1-15. Prove by induction that for every $n \geq 1$, $\sum_{i=1}^n \frac{1}{i(i+1)} = \frac{n}{n+1}$

1-16. Prove by induction that $n^3 + 2n$ is divisible by 3 for all $n \geq 0$.

1-17. Prove by induction that a tree with n vertices has exactly $n - 1$ edges.

1-18. Prove by mathematical induction that the sum of the cubes of the first n positive integer is equal to the square of the sum of these integers, i.e.

$$\sum_{i=1}^n i^3 = \left(\sum_{i=1}^n i \right)^2$$

1.4 Estimation

1.5 Implementation Projects