Intro to Analysis of Algorithms
Preliminaries
Chapter 1

Michael Soltys

CSU Channel Islands

[Ed: 4th, last updated: October 2, 2025]

1. Precondition
2. Postcondition
3. Termination
4. Partial Correctness
5. Correctness (Full Correctness)

Boolean connectives: $\wedge$ is "and," $\vee$ is "or" and $\neg$ is "not."

We also use $\rightarrow$ as Boolean implication, i.e., $x \rightarrow y$ is logically equivalent to $\neg x \vee y$, and $\leftrightarrow$ is Boolean equivalence, and $\alpha \leftrightarrow \beta$ expresses $((\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha))$.

$\forall$ is the "for-all" universal quantifier, and $\exists$ is the "there exists" existential quantifier.

We use "$\Rightarrow$" to abbreviate the word "implies," i.e., $2|x \Rightarrow x$ is even, while "$\not\Rightarrow$" abbreviates "does not imply."

Partial Correctness:

$$(\forall I \in \mathcal{I}_A)[(\alpha_A(I) \wedge \exists O(O = A(I))) \rightarrow \beta_A(A(I))] \qquad (1.1)$$

How to modify it to express full correctness? (Problem 1.1)

Division Algorithm (A1.1)

Pre-condition: $x \geq 0 \wedge y > 0 \wedge x, y \in \mathbb{N}$

1: $q \leftarrow 0$
2: $r \leftarrow x$
3: **while** $y \leq r$ **do**
4: $\quad r \leftarrow r - y$
5: $\quad q \leftarrow q + 1$
6: **end while**
7: **return** q, r

Post-condition: $x = (q \cdot y) + r \wedge 0 \leq r < y$

Loop invariant:

$$x = (q \cdot y) + r \wedge r \geq 0. \tag{1.2}$$

Show that it holds true after each iteration of the loop:

Basis case (i.e., zero iterations of the loop—we are just before line 3 of the algorithm): $q = 0, r = x$, so $x = (q \cdot y) + r$ and since $x \geq 0$ and $r = x$, $r \geq 0$.

Induction step: suppose $x = (q \cdot y) + r \wedge r \geq 0$ and we go once more through the loop.

Let q', r' be the new values of q, r, respectively (computed in lines 4 and 5 of the algorithm).

Since we executed the loop one more time it follows that $y \leq r$ (this is the condition checked for in line 3 of the algorithm), and since $r' = r - y$, we have that $r' \geq 0$. Thus,

$$x = (q \cdot y) + r = ((q + 1) \cdot y) + (r - y) = (q' \cdot y) + r',$$

and so q', r' still satisfy the loop invariant.

Partial Correctness

Now we use the loop invariant to show that (if the algorithm terminates) the post-condition of the division algorithm holds, *if* the pre-condition holds.

This is very easy in this case since the loop ends when it is no longer true that $y \leq r$, i.e., when it is true that $r < y$.

On the other hand, we proved already that loop invariant holds after each iteration, in particular the last one. Putting it all together we get the post-condition.

Termination

To show termination we use the least number principle (LNP).

We need to relate some non-negative monotone decreasing sequence to the algorithm; just consider $r_0, r_1, r_2, \ldots$, where $r_0 = x$, and r_i is the value of r after the i-th iteration.

Note that $r_{i+1} = r_i - y$.

First, $r_i \geq 0$, because the algorithm enters the while loop only if $y \leq r$, and second, $r_{i+1} < r_i$, since $y > 0$.

By LNP such a sequence "cannot go on for ever," (in the sense that the set $\{r_i | i = 0, 1, 2, \ldots\}$ is a subset of the natural numbers, and so it has a least element), and so the algorithm must terminate.

Problem 1.3

What is the running time of the algorithm? That is, how many steps does it take to terminate? Assume that assignments (lines 1 and 2), and arithmetical operations (lines 4 and 5) as well as testing "$\leq$" (line 3) all take one step.

Euclid's Algorithm (A1.2)

Given two positive integers a and b, their *greatest common divisor*, denoted as $\gcd(a, b)$, is the largest positive integer that divides them both.

Pre-condition: $a > 0 \wedge b > 0 \wedge a, b \in \mathbb{Z}$

$m \leftarrow a$; $n \leftarrow b$; $r \leftarrow \mathrm{rem}(m, n)$
while $(r > 0)$ **do**
    $m \leftarrow n$; $n \leftarrow r$; $r \leftarrow \mathrm{rem}(m, n)$
end while
return n

Post-condition: $n = \gcd(a, b)$

Unlike division, Euclid's algorithm is very fast.

This is very important, as it is one of the building blocks of Cryptography; see Problem 6.10, which leads to the correctness of the Rabin-Miller algorithm.

The Rabin-Miller algorithm (Algorithm 25) for primality testing is what makes Publick Key Crypto such as Diffie-Hellman, El Gamal, RSA, etc., possible.

Loop Invariant:

$$m > 0, n > 0 \text{ and } \gcd(m, n) = \gcd(a, b) \tag{1.3}$$

Basis Case: $m = a > 0$ and $n = b > 0$ and so loop invariant holds

Induction Step: suppose $m, n > 0$ and $\gcd(a, b) = \gcd(m, n)$, and we go through the loop one more time, yielding m', n'.

We want to show that $\gcd(m, n) = \gcd(m', n')$.

Note that from line 3 of the algorithm we see that $m' = n, n' = r = \mathrm{rem}(m, n)$, so in particular $m', n' > 0$, since if $r = \mathrm{rem}(m, n)$ were zero, the loop would have terminated (and we are assuming that we are going through the loop one more time).

Problem 1.6: Show that for all $m, n > 0$,
$\gcd(m, n) = \gcd(n, \mathrm{rem}(m, n))$.

Problem 1.7: Show that Euclid's algorithm terminates.

Problem: 1.7: what is the complexity of Euclid's algorithm?

More challenging problem: Show that for any integer $k \geq 1$, if $a > b \geq 1$ and $b < F_{k+1}$ (where F_i is the i-th Fibonacci number), then Euclid's algorithm on a, b takes fewer than k iterations of the while loop. (Ignore swaps, or use $2k$ instead.)

Palindromes (A1.3)

`racecar`

Pre-condition: $n \geq 1 \wedge A[1 \ldots n]$ is a character array

1: $i \leftarrow 1$
2: **while** $(i \leq \lfloor \frac{n}{2} \rfloor)$ **do**
3: **if** $(A[i] \neq A[n-i+1])$ **then**
4: **return** F
5: **end if**
6: $i \leftarrow i+1$
7: **end while**
8: **return** T

Post-condition: return T iff A is a palindrome

Let the loop invariant be: after the k-th iteration, $i = k + 1$ and for all j such that $1 \leq j \leq k$, $A[j] = A[n - j + 1]$.

We prove that the loop invariant holds by induction on k.

Basis case: before any iterations take place, i.e., after zero iterations, there are no j's such that $1 \leq j \leq 0$, so the second part of the loop invariant is (vacuously) true. The first part of the loop invariant holds since i is initially set to 1.

Induction step: we know that after k iterations, $A[j] = A[n - j + 1]$ for all $1 \leq j \leq k$; after one more iteration we know that $A[k + 1] = A[n - (k + 1) + 1]$, so the statement follows for all $1 \leq j \leq k + 1$. This proves the loop invariant.

Show partial correctness of the palindromes algorithm.

Does it terminate? If yes, what is its complexity?

Other practice problems: 1.13,14,15

Python String Manipulations

In is easy to manipulate strings in Python; a segment of a string is called a *slice*. Consider the word `palindrome`; if we set the variables `s` to this word,

```
s = 'palindrome'
```

then we can access different slices as follows:

```
print s[0:5] palin
print s[5:10] drome
print s[5:]  drome
print s[2:8:2] lnr
```

where the notation `[i:j]` means the segment of the string starting from the i-th character (and we always start counting at zero!), to the j-th character, including the first but excluding the last.

The notation `[i:]` means from the i-th character, all the way to the end, and `[i:j:k]` means starting from the i-th character to the j-th (again, not including the j-th itself), taking every k-th character.

One way to understand the string delimiters is to write the indices "in between" the numbers, as well as at the beginning and at the end. For example

$$_0\mathtt{p}_1\mathtt{a}_2\mathtt{l}_3\mathtt{i}_4\mathtt{n}_5\mathtt{d}_6\mathtt{r}_7\mathtt{o}_8\mathtt{m}_9\mathtt{e}_{10}$$

and to notice that a slice `[i:j]` contains all the symbols between index i and index j.

Problem 1.12 What is the shortest Python program you can write to test whether the string s is a palindrome?

Here is a fun problem: on September 11, 2019, we started a "*palindromic week*," meaning that the dates (given in the format {M,MM}{D,DD}YY) are palindromes: 91119,91219,91319,...,91819. When is the next palindromic week? Will we be so lucky as to live and experience it?

Is it obvious when algorithms terminate?

Pre-condition: $a > 0$

$x \longleftarrow a$

while last three values of x not $4, 2, 1$ **do**

if x is even **then**

$x \longleftarrow x/2$

else

$x \longleftarrow 3x + 1$

end if

end while

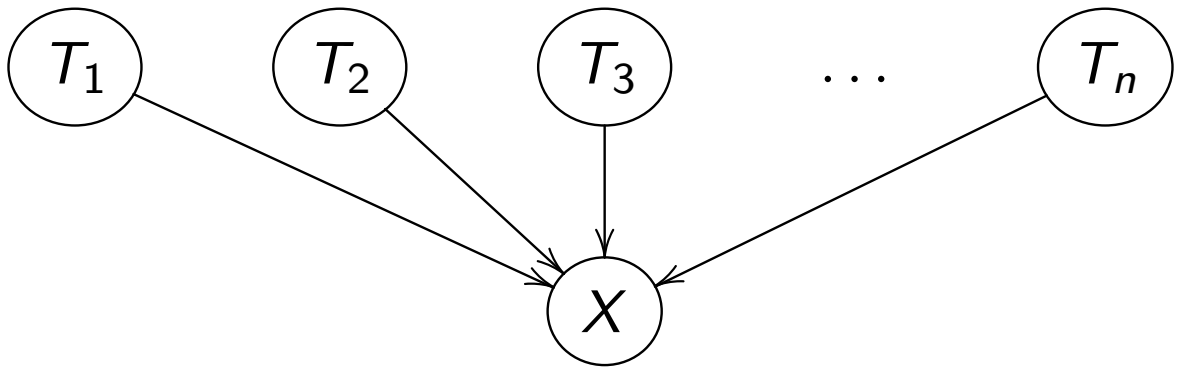
Ranking Algorithms

- PageRank
- Stable Marriage
- Pairwise Comparisons

PageRank

1. 1945 article by Vannevar Bush on the "Memex"
2. 1990s Berners-Lee and hyperlinks (HTML)
3. Huge WWW: how to find a relevant page?
4. Authoritative pages

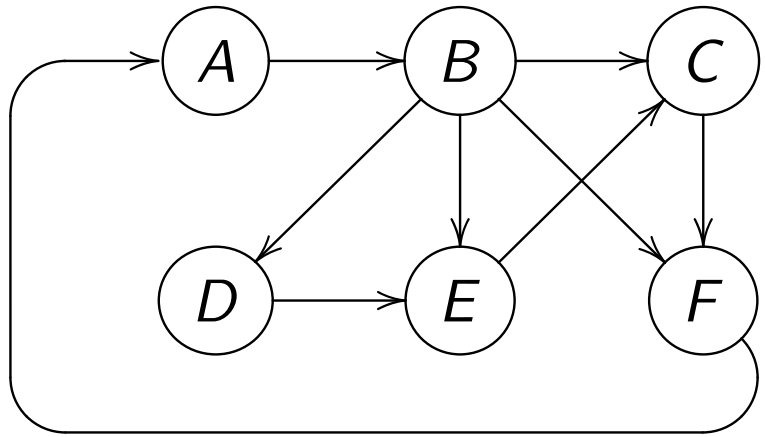
A web page X, and all the pages $T_1, T_2, T_2, \ldots, T_n$ that point to it.



Given a page X, let $\mathrm{C}(X)$ be the number of distinct links that leave X, i.e., these are links anchored in X that point to a page outside of X.

Let $\mathrm{PR}(X)$ be the page rank of X. We also employ a paramater d, which we call the *damping factor*.

$$\mathrm{PR}(X) = (1-d) + d\left[\frac{\mathrm{PR}(T_1)}{\mathrm{C}(T_1)} + \frac{\mathrm{PR}(T_2)}{\mathrm{C}(T_2)} + \cdots + \frac{\mathrm{PR}(T_n)}{\mathrm{C}(T_n)}\right].$$


A
B
C
D
E
F

Using Excel with the following formulas: initially (Stage 0) all get 1/6, and then they are computed with:

$\mathrm{PR}(A) = \mathrm{PR}(F)$

$\mathrm{PR}(B) = \mathrm{PR}(A)$

$\mathrm{PR}(C) = \mathrm{PR}(B)/4 + \mathrm{PR}(E)$

$\mathrm{PR}(D) = \mathrm{PR}(B)/4$

$\mathrm{PR}(E) = \mathrm{PR}(B)/4 + \mathrm{PR}(D)$

$\mathrm{PR}(F) = \mathrm{PR}(B)/4 + \mathrm{PR}(C)$

Stage	0	1	2	3	4	5	...	17
A	0.17	0.17	0.21	0.25	0.29	0.18		0.22
B	0.17	0.17	0.17	0.21	0.25	0.29		0.22
C	0.17	0.21	0.25	0.13	0.14	0.16		0.17
D	0.17	0.04	0.04	0.04	0.05	0.06		0.06
E	0.17	0.21	0.08	0.08	0.09	0.11		0.11
F	0.17	0.21	0.25	0.29	0.18	0.20		0.22
Total	1.00	1.00	1.00	1.00	1.00	1.00		1.00

Stable Marriage

An instance of the *stable marriage problem* of size n consists of two disjoint finite sets of equal size; a set of *boys* $B = \{b_1, b_2, \ldots, b_n\}$, and a set of *girls* $G = \{g_1, g_2, \ldots, g_n\}$.

Let "$<_i$" denote the ranking of boy b_i; that is, $g <_i g'$ means that boy b_i prefers g over g'.

Similarly, "$<^j$" denotes the ranking of girl g_j.

Each boy b_i has such a ranking (linear ordering) $<_i$ of G which reflects his preference for the girls that he wants to marry.

Similarly each girl g_j has a ranking (linear ordering) $<^j$ of B which reflects her preference for the boys she would like to marry.

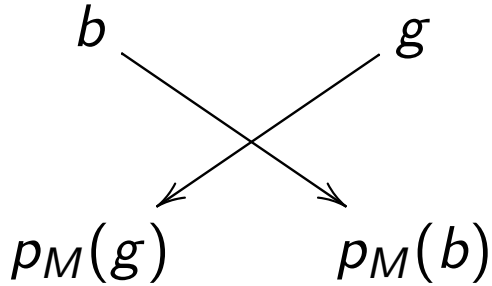
A *matching* (or *marriage*) M is a 1-1 correspondence between B and G.

We say that b and g are *partners* in M if they are matched in M and write $p_M(b) = g$ and also $p_M(g) = b$.

A matching M is *unstable* if there is a pair (b, g) from $B \times G$ such that b and g are not partners in M but b prefers g to $p_M(b)$ and g prefers b to $p_M(g)$.

Such a pair (b, g) is said to *block* the matching M and is called a *blocking pair*

A matching M is stable if it contains no blocking pair.



A blocking pair: b and g prefer each other to their partners $p_M(b)$ and $p_M(g)$.

1: Stage 1: b_1 chooses his top g and $M_1 \longleftarrow \{(b_1, g)\}$
2: **for** $s = 1, \ldots, s = |B| - 1$, Stage $s + 1$: **do**
3: $M \longleftarrow M_s$
4: $b^* \longleftarrow b_{s+1}$
5: **for** b^* proposes to all g's in order of preference: **do**
6: **if** g was not engaged: **then**
7: $M_{s+1} \longleftarrow M \cup \{(b^*, g)\}$
8: end current stage
9: **else if** g was engaged to b but g prefers b^*: **then**
10: $M \longleftarrow (M - \{(b, g)\}) \cup \{(b^*, g)\}$
11: $b^* \longleftarrow b$
12: repeat from line 5
13: **end if**
14: **end for**
15: $M_{s+1} \longleftarrow M$
16: **end for**
17: **return** $M_{|B|}$

$$
\begin{array}{l|l}
b_1 : g_2, g_4, g_3, g_1 & g_1 : b_1, b_3, b_4, b_2 \\
b_2 : g_4, g_1, g_2, g_3 & g_2 : b_3, b_1, b_4, b_2 \\
b_3 : g_2, g_1, g_3, g_4 & g_3 : b_3, b_4, b_1, b_2 \\
b_4 : g_3, g_4, g_1, g_2 & g_4 : b_2, b_1, b_3, b_4
\end{array}
$$

Stage 1: $M_1 = \{(b_1, g_2)\}$

$$
\begin{array}{l|l}
b_1 : g_2, g_4, g_3, g_1 & g_1 : b_1, b_3, b_4, b_2 \\
b_2 : g_4, g_1, g_2, g_3 & g_2 : b_3, b_1, b_4, b_2 \\
b_3 : g_2, g_1, g_3, g_4 & g_3 : b_3, b_4, b_1, b_2 \\
b_4 : g_3, g_4, g_1, g_2 & g_4 : b_2, b_1, b_3, b_4
\end{array}
$$

Stage 1: $M_1 = \{(b_1, g_2)\}$
Stage 2: $M = M_1$, $b^* = b_2$, $M_2 = \{(b_1, g_2), (b_2, g_4)\}$

$$
\begin{array}{l|l}
b_1 : g_2, g_4, g_3, g_1 & g_1 : b_1, b_3, b_4, b_2 \\
b_2 : g_4, g_1, g_2, g_3 & g_2 : b_3, b_1, b_4, b_2 \\
b_3 : g_2, g_1, g_3, g_4 & g_3 : b_3, b_4, b_1, b_2 \\
b_4 : g_3, g_4, g_1, g_2 & g_4 : b_2, b_1, b_3, b_4
\end{array}
$$

Stage 1: $M_1 = \{(b_1, g_2)\}$
Stage 2: $M = M_1$, $b^* = b_2$, $M_2 = \{(b_1, g_2), (b_2, g_4)\}$
Stage 3: $M = M_2$, $b^* = b_3$, $M = \{(b_2, g_4), (b_3, g_2)\}$, $b^* = b_1$,
$M_3 = \{(b_1, g_3), (b_2, g_4), (b_3, g_2)\}$

$$
\begin{array}{l|l}
b_1 : g_2, g_4, g_3, g_1 & g_1 : b_1, b_3, b_4, b_2 \\
b_2 : g_4, g_1, g_2, g_3 & g_2 : b_3, b_1, b_4, b_2 \\
b_3 : g_2, g_1, g_3, g_4 & g_3 : b_3, b_4, b_1, b_2 \\
b_4 : g_3, g_4, g_1, g_2 & g_4 : b_2, b_1, b_3, b_4
\end{array}
$$

Stage 1: $M_1 = \{(b_1, g_2)\}$
Stage 2: $M = M_1$, $b^* = b_2$, $M_2 = \{(b_1, g_2), (b_2, g_4)\}$
Stage 3: $M = M_2$, $b^* = b_3$, $M = \{(b_2, g_4), (b_3, g_2)\}$, $b^* = b_1$,
$M_3 = \{(b_1, g_3), (b_2, g_4), (b_3, g_2)\}$
Stage 4: $M = M_3$, $b^* = b_4$, $M = \{(b_2, g_4), (b_3, g_2), (b_4, g_3)\}$,
$b^* = b_1$, $M_4 = \{(b_1, g_1), (b_2, g_4), (b_3, g_2), (b_4, g_3)\}$

The matching M is produced in stages M_s so that b_t always has a partner at the end of stage s, where $s \geq t$.

However, the partners of b_t do not get better, i.e., $p_{M_t}(b_t) \leq_t p_{M_{t+1}}(b_t) \leq_t \cdots$.

On the other hand, for each $g \in G$, if g has a partner at stage t, then g will have a partner at each stage $s \geq t$ and the partners do not get worse, i.e., $p_{M_t}(g) \geq^t p_{M_{t+1}}(g) \geq^t \ldots$.

Thus, as s increases, the partners of b_t become less preferable and the partners of g become more preferable.

At the end of stage s, assume that we have produced a matching

$$M_s = \{(b_1, g_{1,s}), \ldots, (b_s, g_{s,s})\},$$

where the notation $g_{i,s}$ means that $g_{i,s}$ is the partner of boy b_i after the end of stage s.

We will say that partners in M_s are *engaged*.

The idea is that at stage $s+1$, b_{s+1} will try to get a partner by *proposing* to the girls in G in his order of preference.

When b_{s+1} proposes to a girl g_j, g_j accepts his proposal if either g_j is not currently engaged or is currently engaged to a less preferable boy b, i.e., $b_{s+1} <^j b$.

In the case where g_j prefers b_{s+1} over her current partner b, then g_j breaks off the engagement with b and b then has to search for a new partner.

Pairwise Comparisons

Ramon Llull

Marquis de Condorcet

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a finite set of objects to be ranked.

Let a_{ij} express the numerical preference between x_i and x_j. The idea is that a_{ij} estimates "how much better" x_i is compared to x_j.

Clearly, for all i, j, $a_{ij} > 0$ and $a_{ij} = 1/a_{ji}$.

The intuition is that if $a_{ij} > 1$, then x_i is preferred over x_j by that factor.

So, for example, Apple's Retina display has four times the resolution of the Thunderbolt display, and so if x_1 is Retina, and x_2 is Thunderbolt, we could say that the image quality of x_1 is four times better than the image quality of x_2, and so $a_{12} = 4$, and $a_{21} = 1/4$.

The assignment of values to the a_{ij}'s are often done subjectively by human judges.

Let $A = [a_{ij}]$ be a *pairwise comparison matrix*, also known as a *preference matrix*.

We say that a pairwise comparison matrix is *consistent* if for all i, j, k we have that $a_{ij}a_{jk} = a_{ik}$. Otherwise, it is *inconsistent*.

In practice, the subjective evaluations a_{ij} are seldom consistent, which creates four problems, and to this day there is no satisfactory solution to these problems:

1. How to measure inconsistency and what level is acceptable?
2. How to remove inconsistencies, or lower them to an acceptable level?
3. How to derive the values w_i starting with an inconsistent ranking A?
4. How to justify a certain method for removing inconsistencies?

In real world cases, it is item 4. were the subjective side of the judgments comes most to the fore, as the "subjectiveness" of the referees is reflected in the inconsistency of the resulting matrix.