3 Algorithms

3.1 Algorithms

Def. *1 An* ***algorithm*** *is a finite set of instructions for performing a computation or solving a problem.*

Alg. *1 Finding the maximal element in a finite sequence*

function $max(a_1, a_2, \ldots, a_n$: integers): integer; $n \geq 1$

$max := a_1$; $max = max(a_1)$

$max = max(a_1, \ldots, a_i), i = 1$

for $i := 2$ ***to*** n ***do***

if $max < a_i \rightarrow max := a_i$ ***fi*** $max = max(a_1, \ldots, a_i), 2 \leq i \leq n\text{-}1$

od $max = max(a_1, \ldots, a_i), i = n$

return max $max = max(a_1, \ldots, a_n)$

syntic sugar of** "**for** (init; test; update) loop_body" in Pascal or **C

***for** (max, $i := a_1, 2$; $i \le n$; $i{+}{+}$)*

***if** $max < a_i \rightarrow max = a_i$*

$\mid max \ge a_i \rightarrow$ ***skip***

***fi**;*

Dijkstra's mini language

$max, i := a_1, 2;$		$max = max(a_1) \wedge (i = 2)$
do $i \le n \rightarrow$	$\equiv$	$2 \le i \le n$: $max = max(a_1, \ldots, a_{i-1})$
if $max < a_i \rightarrow max = a_i$		
$\mid max \ge a_i \rightarrow$ ***skip***		
fi;		$2 \le i \le n$: $max = max(a_1, \ldots, a_i)$
$i := i+1$		$2 \le i \le n$: $max = max(a_1, \ldots, a_{i-1})$
od		$max = max(a_1, \ldots, a_{i-1}) \wedge$ ***(i=n+1)*** $= max(a_1, \ldots, a_n)$

Syntax* of *Dijkstra's mini language

$S ::= x_1, \ldots, x_n := E_1, \ldots, E_n \mid S; S \mid \textbf{skip} \mid \textbf{abort}$

$\mid \textbf{if } B_1 \to SL_1 \text{ '|' } \ldots \text{ '|' } B_n \to SL_n \textbf{ fi}$

$\mid \textbf{do } B_1 \to SL_1 \text{ '|' } \ldots \text{ '|' } B_n \to SL_n \textbf{ od}$

Semantics* of *Dijkstra's mini language

concurrent assignment $x, y := y, x$

A ***sequence*** *of statements* $S_1; S_2; \ldots; S_n$

If a ***guard*** B_i *is* **T**, *then the statement list* SL_i *will be excuted.*

$BB \equiv {}^{\exists}i\ B_i \qquad \neg BB \equiv {}^{\forall}i\ \neg B_i.$

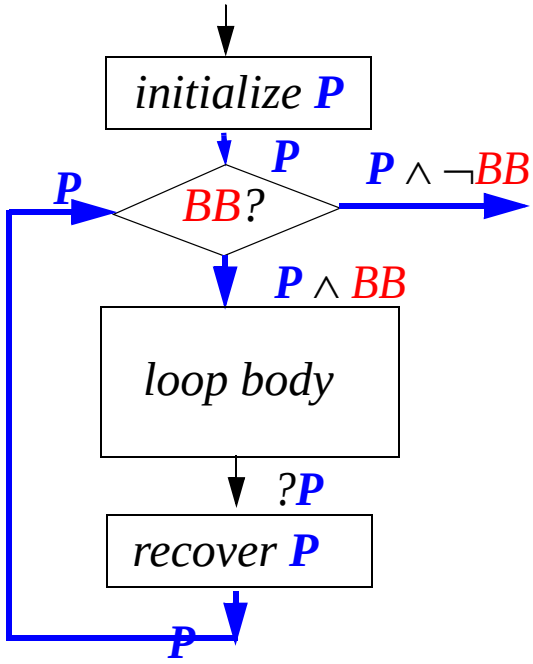
do od $\equiv$ ***skip*** ***if fi*** $\equiv$ ***abort***

Alg. *max(x, y)*

if $x \geq y \to m := x$	***if*** $x > y \to m := x$	***if*** $x \geq y \to m := x$
$\mid x < y \to m := y$	$\mid x \leq y \to m := y$	$\mid x \leq y \to m := y$
fi	***fi***	***fi***

$((m=x) \vee (m=y)) \wedge (m \geq x) \wedge (m \geq y)$

do $B_1 \rightarrow SL_1$ '|' … '|' $B_n \rightarrow SL_n$ ***od***



Let's assume that loop invariance is ***P****.*

and $BB = B_1 \vee B_2 \vee \ldots \vee B_n$.

$\therefore \neg BB = \neg(B_1 \vee B_2 \vee \ldots \vee B_n)$

$= \neg B_1 \wedge \neg B_2 \wedge \ldots \neg B_n$.

for (init; test(BB); update(recover))
loop_body

Loop invariance ***P***

Loop invariance ***P*** *should be* ***initialized*** *before the loop,*

Loop invariance ***P*** *must be* ***recovered*** *in the loop, if destroyed, and*

Loop invariance ***P*** *is still* ***valid*** *after(termination of) the loop.*

P $\wedge \neg BB$ *is the* ***final*** *condition that you want* ***after the loop****!*

Summation *from 1 to 100*

$S, i := 0, 1;$	$(S = 0) \wedge (i = 1)$
do $i \leq 100 \rightarrow$	$(S = 1 + \ldots + i-1) \wedge (1 \leq i \leq 100)$
$S := S + i;$	$(S = 1 + \ldots + i) \wedge (1 \leq i \leq 100)$
$i := i+1$	$(S = 1 + \ldots + i-1) \wedge (1 \leq i \leq 100)$
	P

od

$$P \wedge \neg BB \equiv (S = 1 + \ldots + i-1) \wedge (i = 101)$$
$$\equiv (S = 1 + \ldots + 100)$$

???

$a, b, c, d := A, B, C, D;$
do $a > b \rightarrow a, b := b, a$
$\quad | \; b > c \rightarrow b, c := c, b$
$\quad | \; c > d \rightarrow c, d := d, c$
od

$\neg BB = (a \le b) \wedge (b \le c) \wedge (c \le d)$
$= a \le b \le c \le d.$

What is the loop invariance of the above do-od loop?

P *= a, b, c, and d are* ***permutation*** *of A, B, C, and D.*

Loop invariance ***P*** *should be* ***strong*** *enough to get* ***final result***
with $\boldsymbol{P} \wedge \neg BB$*;*
whereas ***weak*** *enough*
to be easily ***initialized*** *and be* ***still valid*** *inside the loop.*

$\boldsymbol{P} \wedge \neg BB$ *is the* ***final condition*** *that you want.*

Searchine algorithms

Alg. *2 The* ***linear search*** *algorithm*

function *linear search(x:integer;* $a_1,a_2,\ldots,a_n$*: integers): integer*$(n \geq 0)$

$i := 1$; ***do*** $((i \leq n) \wedge (x \neq a_i)) \rightarrow i := i+1$ ***od***

$[(i \leq n) \Rightarrow ((1 \leq^{\forall} k < i: x \neq a_k) \wedge (x = a_i))] \vee [(i = n+1) \Rightarrow (1 \leq^{\forall} k \leq n: x \neq a_k)]$

if $(i \leq n) \rightarrow loc := i \mid (i = n+1) \rightarrow loc = 0$ ***fi***

$[(loc \leq n) \Rightarrow ((1 \leq^{\forall} k < loc: x \neq a_k) \wedge x = a_{loc})] \vee [(loc = 0) \Rightarrow (1 \leq^{\forall} k \leq n: x \neq a_k)]$

return *loc*

Another implementation for the ***linear search***

$i, loc := 1, 0;$ $(n \geq 0)$

do $((i \leq n) \wedge (loc = 0)) \rightarrow$ ***if*** $x = a_i \rightarrow loc := i;$

$\mid x \neq a_i \rightarrow i := i+1$

od ***fi***

$[(loc \leq n) \Rightarrow ((1 \leq^{\forall} k < loc: x \neq a_k) \wedge x = a_{loc})] \vee [(loc = 0) \Rightarrow (1 \leq^{\forall} k \leq n: x \neq a_k)]$

Alg. *3 The binary search algorithm*

function *binary search(x: integer;* $a_1, a_2, \ldots, a_n$*:* ***increasing*** *integers): integer;*

$(n \geq 0) \wedge (1 \leq^{\forall} k < n: a_k < a_{k+1})$ *is the given condition(****increasing*** *integers)*

$i, j, loc := 1, n, 0;$

do $((i \leq j) \wedge (loc = 0)) \rightarrow m := \lfloor (i+j)/2 \rfloor ;$

if $x > a_m \rightarrow i := m+1$

$\mid x = a_m \rightarrow loc := m$

$\mid x < a_m \rightarrow j := m-1$

fi

od

$[(loc \leq n) \Rightarrow (x = a_{loc})] \vee [(loc = 0) \Rightarrow (1 \leq^{\forall} k \leq n: x \neq a_k)]$

return *loc*

removes $\wedge$ *(loc=0) test in the outer* ***do od guard***

i, j, loc := 1, n, 0;

do $(i \le j) \rightarrow m := \lfloor (i+j)/2 \rfloor$;

if $x > a_m \rightarrow i := m+1$

$| \; x = a_m \rightarrow loc := m;\ i, j := m+1, m-1;$

$| \; x < a_m \rightarrow j := m-1$

fi

od

$[(loc \le n) \Rightarrow (x = a_{loc})] \vee [(loc=0) \Rightarrow (1 \le^{\forall} k \le n: x \ne a_k)]$

upper bound of numbers of ***if*** *tests(searchs)*

linear	*(n-1) seraches*
binary	$\lceil \log_2 n \rceil$ *searches*

Alg. 4 function bubble sort($a_1, a_2, \ldots, a_n$: integers with $n \geq 2$)**: integers**;

for $i := 1$ to $n - 1$ **do**

  for $j := 1$ to $n - i$ **do** $\rightarrow$

    if $a_j \geq a_{j+1} \rightarrow a_j, a_{j+1} := a_{j+1}, a_j$

    | $a_j \leq a_{j+1} \rightarrow$ **skip**

    fi

  od

od; **return** $(a_1, a_2, \ldots, a_n)$.

$i = 1$	**$a_n = max(a_1, \ldots, a_n)$**	**(n-1) iterations**
$i = 2$	$a_{n-1} = max(a_1, \ldots, a_{n-1})$	(n-2) iterations
...		
i	$a_{n-i+1} = max(a_1, \ldots, a_{n-i+1})$	(n-i) iterations
...		
$i = n-1$	$a_2 = max(a_1, a_2)$	(1) iteration
	$1 \leq^{\forall} k < n: a_k \leq a_{k+1}$.	n(n-1)/2 iterations

Alg. 5 function *insertion sort*($a_1, a_2, \ldots, a_n$: *distinct integers with* $n \geq 2$)**:**
integers;

for $j := 2$ **to** n **do**	$j = 2$.
	$(2 \leq {}^{\forall}k \leq j\text{-}1)$: $(a_{k-1} < a_k)$.
$i := 1$;	$i = 1$.
do $(a_j > a_i) \rightarrow i := i+1$	$(1 \leq {}^{\forall}l \leq^{?} n)$: $(a_j > a_l)$.
od;	$\wedge\ (a_j \leq a_i)$.
$m := a_j$;	
for $k := 0$ **to** $j - i - 1$ **do** $a_{j-k} := a_{j-k-1}$ **od**;	
for $k := i\text{-}1$ **downto** j **do** $a_k := a_{k-1}$ **od**;	
$a_i := m$	
od;	$(2 \leq {}^{\forall}k \leq j\text{-}1)$: $(a_{k-1} < a_k) \wedge (j=n+1)$.
return $(a_1, a_2, \ldots, a_n)$.	$\equiv (2 \leq {}^{\forall}k \leq n)$: $(a_{k-1} < a_k)$.

Greedy algorithm

Optimization problem

to find a solution to the given problem that

either ***minimizes*** *or* ***maximizes*** *the value of some parameters*

(global) optimal

to select the best choice considering all sequences of steps

local optimal(greedy algorithm)

to select the best choise at each step

Alg. *6 Greedy Change-Making Algorithm*

function *change(n: integer, c_1, c_2, …, c_r: value of changes): (n_1, n_2, …, n_r): number of changes*

$c_1 > c_2 > \ldots > c_r$ *for each* $coin_i$;

$n_1, n_2, \ldots, n_r := 0, 0, \ldots, 0$; $i := 1$;

do $(i \le r) \rightarrow$ ***do*** $(n \ge c_i) \rightarrow n_i := n_i + 1$; $n := n - c_i$ ***od***; $i := i+1$ ***od***

Minimize $n_1 + n_2 + \ldots + n_r$*?*

The Halting Problem

proof *Assume a procedure H(P: program, I: input data) exists where*

if *P runs with I and halts → print "halt" as output.*

| P with I loops forever → print "loops forever" as output.

Program also can be a input data(compiler)

Condider procedure K(P) as follows

procedure K(P: program)

if *H(P, P) prints "halts" → loops forevar*

| H(P, P) prints "loops forever" → halts

fi

Now consider K(K)

if H(K, K) print "loops forever" → K(K) halts

| H(K, K) print "halts" → K(K) loops forever

Contradiction!

No ***program*** *for halting program!*

Unsolvable problems

Halting problem

where program will be halt or not(infinite loop)

Alan Turing, 1936

No program that computes it!

Three cases for ***problems(Turing-Church's thesis)***

i) (totally) ***solvable(computable; recursive)***

program *exits and always* ***terminates***

algorithms

ii) ***partially solvable(partially computable;recursively enumerable)***

program *exits but may* ***terminate*** *or* ***loop forever***

iii) ***Un programmable(uncomputable; Not recursively enumerable)***

No *algorithm(program) exists*

halting program, Russel's Paradox, G.I.T.

i) $\subset$ *ii)* *ii)* $\cap$ *iii)* $= \varnothing$.

3.2 The Growth of Functions

Big-O Notation

Def. *1 Let f:* $\mathbf{N} \to \mathbf{R}$ *or* $\mathbf{R} \to \mathbf{R}$.

We say f(x) is O(g(x)), if $\exists C, k \in \mathbf{R}$.ə. $|f(x)| \le C|g(x)|$ $\forall x > k$.
We also write f(x) is O(g(x)) or even $f(x) \in O(g(x))$.

The constant C and k are called the ***witnesses*** *of ... f(x) is O(g(x)).*

Assume C and k are ***smallest*** *witnesses, Then* $\forall C', k'$.ə. $C < C', x < x'$,
C' and k' are also ***witnesses***

g(x) is ***faster or equl(not slower)*** *glowing than f(x)*
when x becomes ***large enough***
and ***ignoring*** *multiplying some* ***constant*** *C.*

Ex. 1, 2, 3, 4

Thm. *1 Let* $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$. *Then* $f(x) \in O(x^n)$.
proof *easy with witness* $k=1$ *and* $C = \Sigma |a_i|$.

Ex. 6 $n!$

$n! \le n^n$. *But* $n!$ *is* $O(n^n)$ *with witness* $k=1$ *and* $C=1$.

$\log n! \le \log n^n = n \log n$

$1, \log n, n, n\log n, n^2, \ldots, 2^n, n!$

Def. *Let* $f \in O(g)$. *Then we say* f *glows* ***at most*** *order of* g.

$O(1) \subset O(\log n) \subset O(n) \subset O(n\log n) \subset O(n^2) \subset O(n^3) \subset \ldots$
$\subset O(2^n) \subset \ldots \subset O(n!)$

See Figure 3(p. 213)

Thm. *2, 3 Let* $f_1 \in O(g_1)$, $f_2(x) \in O(g_2)$. *Then*

$f_1+f_2 \in O(g_1+g_2) = O(max(|g_1|, |g_2|)$ *and*

$f_1 f_2 \in O(g_1 g_2)$.

Col. *1 Let* $f_1, f_2 \in O(g)$. *Then* $f_1+f_2 \in O(g)$.

Big-Omega and Big-Theta Notation

Def. *2* $\Omega(g) = \{f: \mathbf{N} \text{ or } \mathbf{R} \rightarrow \mathbf{R} | \ ^{\exists}C, k > 0 \ ^{\forall}x > k: |f(x)| \geq C|g(x)|\}$

If $f \in \Omega(g)$, *f glows* ***at least*** *order of g.*

Def. *3* $\Theta(g)=\{f:\mathbf{N}\rightarrow\mathbf{R} | \ \exists C_1,C_2,k>0, \ ^{\forall}x>k: C_1|g(x)|\leq|f(x)|\leq C_2|g(x)|\}$

If $f \in \Omega(g)$, *f glows (exactly)* ***same*** *order of g.*

$f \in \Theta(g)$, *iff* $f \in O(g) \wedge f \in \Omega(g)$.

Thm. *4 Let* $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ *with* $a_n \neq 0$. *Then*

$f(x) \in \Theta(x^n)$.

3.3 Complexity of Algorithms

Computational complexity

time complexity

space complexity

Worst-case analysis

Average-case analysis

$\Theta(1)$	***constant*** *complexity*
$\Theta(\log n)$	***logarithmic*** *complexity*
$\Theta(n)$	***linear*** *complexity*
$\Theta(n^k)$	***polinomial*** *complexity*
$\Theta(k^n)$	***exponential*** *complexity*
$\Theta(n!)$	***factorial*** *complexity*

Complexity of Matrix Multiplication

Alg. 1 Matrix Multiplication

function matrix multiplication($\mathbf{A_{n;k}}$, $\mathbf{B_{k;m}}$: matrices)

for $i := 1$ **to** n **do**

for $j := 1$ **to** m **do**

$c_{i,j} := 0;$

for $q := 1$ **to** k **do**

$c_{i,j} := c_{i,j} + a_{i,k}b_{k,j}$ **od od od**

return $\mathbf{C_{n;m}}$

$O(n^3)$

(totally) solvable

tractable

polynomial

intractable

exponential

***class* P**

polynomial, tractable

class* NP*(*P $\subseteq$ NP*)

Nondeterministc polynomial *exponetial(upper bound)*

lower bound? *tractable or intractable*

**NP-*complete* *problems*

subset of class NP

If any of these problems can be solved in polynomial time,

***all** the problems in class in* **NP** *also in class* **P*(*P = NP*)*.