Week 5: Proof-of-Correctness

CSC 236:Introduction to the Theory of Computation

Summer 2024

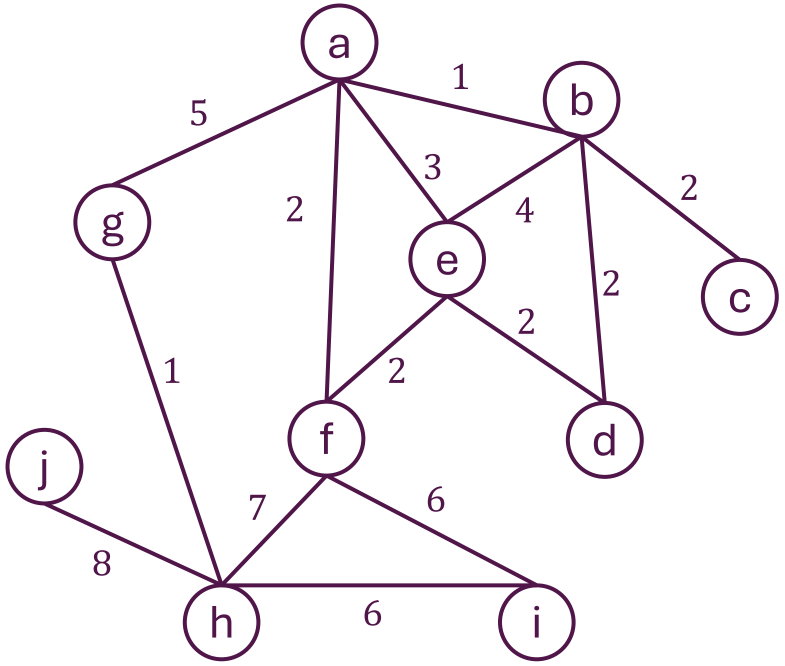
Instructor: Lily

Announcements

- A2 due tomorrow (Thursday June 6) EoD
- Midterm logistics available on the course website
  - June 19th 7:00~9:00pm in the exam center (currently EX 200). Check A&S website for location before exam as it might change.
  - You can bring a one-sided aid sheet.
  - There will be 5 questions (possibly with multiple parts) and one bonus.
  - Covers material from weeks 1~4 (up to but not including today)
  - Please email us asap if you cannot attend to schedule make-up oral exam. These need to take place *before* we release solutions.

Prim's Algorithm

```
def mst_prim(V, E, w)-> list[edges]:
  # Pre: G = (V,E) connected
  # Post: output MST
T = []
visited = {a}
while visited != V:
  (u,v) = min weight edge
  T = T.append((u,v))
  visited.add(v)
return T
```



Program Correctness (Iterative)

1. Given problem statement

eg output MST

You : - algorithm

↳ pre / post cond.

- prove correctness

- runtime

2. Given algorithm

You : - post cond

- correctness

- run time.

given w/ an algorithm

Precondition. Properties of the input.

Postconditions. Properties of the output

Program Correctness. Let f be a function with a set of preconditions and post conditions. Then f is *correct* (with respect to the pre- and postconditions) if for every input I *to* f, if I satisfies the preconditions, then $f(I)$ terminates and all the postconditions hold after termination.

Loop Invariant. Guarantees that during the execution of the algorithm you are making progress towards goal

Termination. Guarantees that the loop terminates.

Structure

1. Find the appropriate post-condition (if not given).
2. If there are loops in your algorithm, give an appropriate loop invariant (LI) for the loop and prove your loop invariant.
3. Use your LI and the loop exit condition to prove partial correctness.
4. Define an appropriate loop measure to prove termination of the loop.
5. (*) Running time analysis.

not officially part of proof-of-correctness
but may be useful to think about for termination.

Multiply

TEST

	m	count
Input (a,b)	0	0
	a	1
	2a	2
	⋮	

variable: (end of)

m_i = value of m after iter i

$count_i$ = ... count after iter i

```python
def mult(a, b):
    # Pre: a and b are natural number
    # Post: returns a*b
    m = 0
    count = 0
    while count < b:
        m += a
        count += 1
    return m
```

O(1)

O(b) iter

O(1)

Total: O(b)

TERMINATION

loop measure $L = b - count$

→ count +1 every iteration

loop measure $L \to 0$

$M = a \cdot b$ upon termination

(LI) $Inv(i): (count_i < b) \Rightarrow m_i = a \cdot count_i$

Base case. $i=0$ $m_i = a \cdot count_i$

if $b=0$ no looping

Inductive Step. fix $k \geq 0$, suppose $P(0) \wedge \dots \wedge P(k)$ true, WTS $P(k+1)$

(IH) $k < b$, if $k+1 = b$ loop terminates,

suppose $k+1 < b$. Then (IH) $m_k = a \cdot count_k$

$m_{k+1} = m_k + a = a\,count_k + a = a(count_k + 1) = a\,count_{k+1}$

Average – Correctness

one indexed list

```python
def average(A):
  # Pre: A is a non-empty list of numbers
  # Post: Returns the average of all numbers in A
  total = 0
  i = 1
  while i <= len(A):
    total += A[i]
    i += 1
  return total / len(A)
```

$\overbrace{\text{len(A)}}^{n}$

WTS $\text{total} = \sum_{i=1}^{n} a_i$ for $a_i \in A$

variables: total_i = val of total after iter i

(LI) $P(i) = \left(\text{total}_i = \sum_{j=1}^{i} A[j] \right)$ $(0 \le i \le n) \Rightarrow$

Base case: $k = 0$ ← before start of loop

total_0 is sum of first zero elements of A.

Inductive step. fix $k \geq 0$

suppose $P(0) \wedge \dots \wedge P(k)$ true

WTS $P(k+1)$

(IH) $1 \le k \le n$ if $k+1 > n$ then loop terminated, otherwise

(IH) $\text{total}_{k+1} = \left(\sum_{j=1}^{k} A[j] \right) + A[k+1]$

$\text{total}_{k} + A[k+1]$

$= \sum_{j=1}^{k+1} A[j]$

Average – Termination

```
def average(A):
  # Pre: A is a non-empty list of numbers
  # Post: Returns the average of all numbers in A
  total = 0
  i = 0
  while i < len(A):
    total += A[i]
    i += 1
  return total / len(A)
```

n

i increments at every iteration

eventually $i > n$

Average – Run-time

```
def average(A):
  # Pre: A is a non-empty list of numbers
  # Post: Returns the average of all numbers in A
  total = 0
  i = 0
  while i < len(A):
    total += A[i]
    i += 1
  return total / len(A)
```

$O(1)$

$O(n)$ iterations

$O(1)$

$O(1)$

total is $O(n)$

Now your turn! Selection Sort

```
def selection_sort(A):
  # Pre: A is a non-empty list of integers
  n = len(A)        # POST :
  i = 1
  for i in range(n):
    min_index = i
    for j in range(i+1, n):
      if A[j] < A[min_index]:
        min_index = j
    swap(A[i], A[min_index])
  return
```

[| 5 4 1 2 3]

[1 | 4 5 2 3]

[1 2 | 5 4 3]

[1 2 3 | 4 5]

Selection Sort – Correctness

zero-indexed list

```
def selection_sort(A):
  # Pre: A is a non-empty list of integers
  n = len(A)

  for i in range(n):
    min_index = i
    for j in range(i+1, n):
      if A[j] < A[min_index]:
        min_index = j
    swap(A[i], A[min_index])
  return
```

POST: A is sorted.

← excludes right end.

Variables:

A_i = A after iteration i

(LI) $P(i) = (0 \le i < n)$

$\Rightarrow A_i[0:i]$ will be sorted and

$\forall a \in A_i[0:i], \forall b \in A_i[i:]$, $a \le b$.

Base case. $i = 0$ then

$A_0 = [\,]$ true.

Selection Sort – Termination/ Run Time

```
def selection_sort(A):
  # Pre: A is a non-empty list of integers
  n = len(A)
  i = 1
  for i in range(n):
    min_index = i
    for j in range(i+1, n):
      if A[j] < A[min_index]:
        min_index = j
    swap(A[i], A[min_index])
  return
```

for-loops which do not change index inside will always terminate

Inductive step. fix $k \geq 0$, $P(0) \wedge \cdots \wedge P(k)$ is True

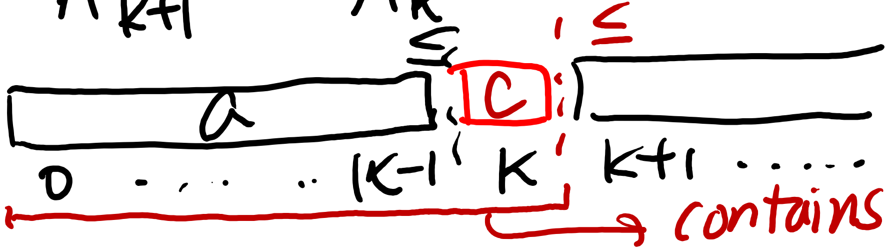
prove $P(k+1)$ is true

(IH: $P(k)$) $0 \leq k < n$. If $k+1 = n$ then for loop terminates

consider $0 \leq k+1 < n$.

$\text{min\_index} \leftarrow \min_{j \in \{k, \dots, n\}} A_k[j]$

$A_{k+1} = A_k$ with $A[k]$ and $A[\text{min\_index}]$ swapped

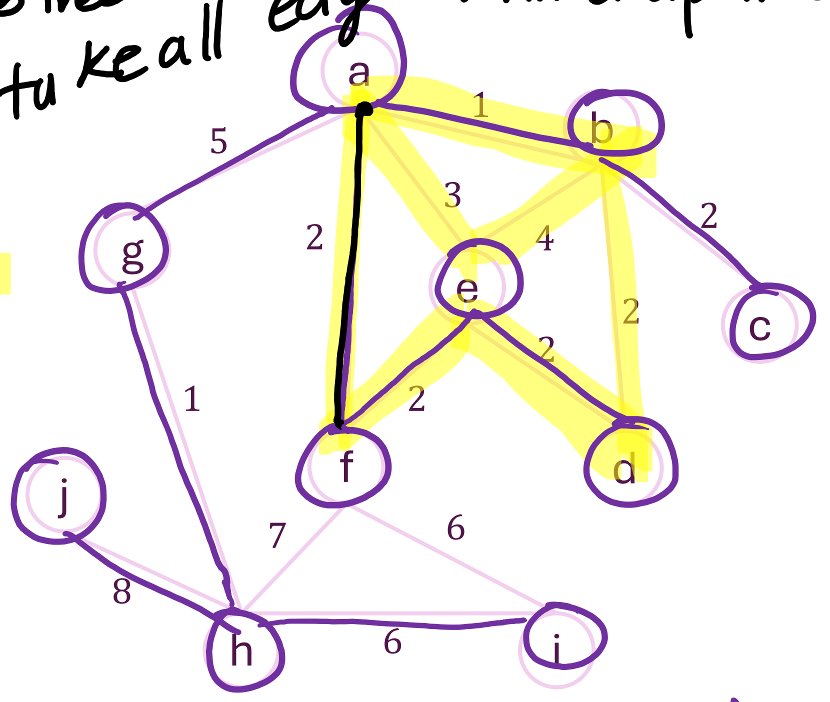


contains k+1 smallest elements sorted

Prim's Algorithm

Induced subtree on $V' \subseteq V$ take all edges with endpoints in V'

eg. induced on a,b,e,f,d




```
def mst_prim(V, E, w)-> list[edges]:
  # Pre: G = (V,E) connected
  # Post: output MST
1  T = []
2  visited = {a}
3  while visited != V:
4      (u,v) = min weight edge
5      T = T.append((u,v))
6      visited.add(v)
7  return T
```

U

1,2 PRE : G is connected graph

↓

3 → 6 WHILE LOOP

↓

7 POST : OUT MST.

Variables :

$T_i = T$ after i iterations

$U_i = U$ after i iterations.

Prim's Algorithm – Correctness

```
def mst_prim(V, E, w)-> list[edges]:
  # Pre: G = (V,E) connected
  # Post: output MST
  T = []
  visited = {a}
  while visited != V:
    (u,v) = min weight edge
    T = T.append((u,v))
    visited.add(v)
  return T
```

U

$u \in U \to v \in V \setminus U$

(LI) $P(i)$ = for nodes in U_i, T_i is a MST of the subgraph induced on U_i

Base case. $i=0$ $T_i = [\,]$, $U_i = \{a\}$
T_i is MST on induced subgraph on U_i.

Inductive step: Fix some $k \geq 0$, $P(0) \wedge \dots \wedge P(k)$ true.
show $P(k+1)$ is true.

THINK ABOUT HOW THE ALG WORKS. suppose (u,v) add in iter $k+1$ ($u \in U$, $v \in V \setminus U$) induced graph on $U_{k+1} = U_k \cup \{v\}$. (IH on $P(k)$) T_k is MST of U_k wo picked cheapest edge needed to include v so T_{k+1} is MST of U_{k+1}.

Prim's Algorithm – Termination/ Run Time

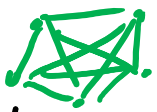
```
def mst_prim(V, E, w)-> list[edges]:
  # Pre: G = (V,E) connected
  # Post: output MST
1  T = []
2  visited = {a}
3  while visited != V:
4     (u,v) = min weight edge
5     T = T.append((u,v))
6     visited.add(v)
7  return T
```

U

∈U ∈V\U

Lines 1–2 → $O(1)$

Line 3 → $O(n)$ iterations.



K_5 equal weights.

Line 4 → ???

Lines 5–6 → $O(1)$

Line 7 → $O(1)$

$O(n^3)$

$\Omega(n^3)$ to show give example graph eg. K_n

Runtime Analysis:

DEPENDS ON DATASTRUCTURE USED

↳ CSC263/265

use adjacency list: loop through all U and all V ⇒ $O(n^2)$

⇒ use fib heaps ⇒ $O(|E| + |V| \log |V|)$ (m, n, n)

TERMINATION:

Line 6 adds a node to U at every iteration. It will stop when all nodes added.

Recap

- Overview of proof-of-correctness steps
- Seen some examples for simple algorithms
- Seen harder examples in sorting
- Proved correctness of Prim's Algorithm

Next time… recursive algorithm and proving that they are correct