Week 5: Proof-of-Correctness

CSC 236:Introduction to the Theory of Computation

Summer 2024

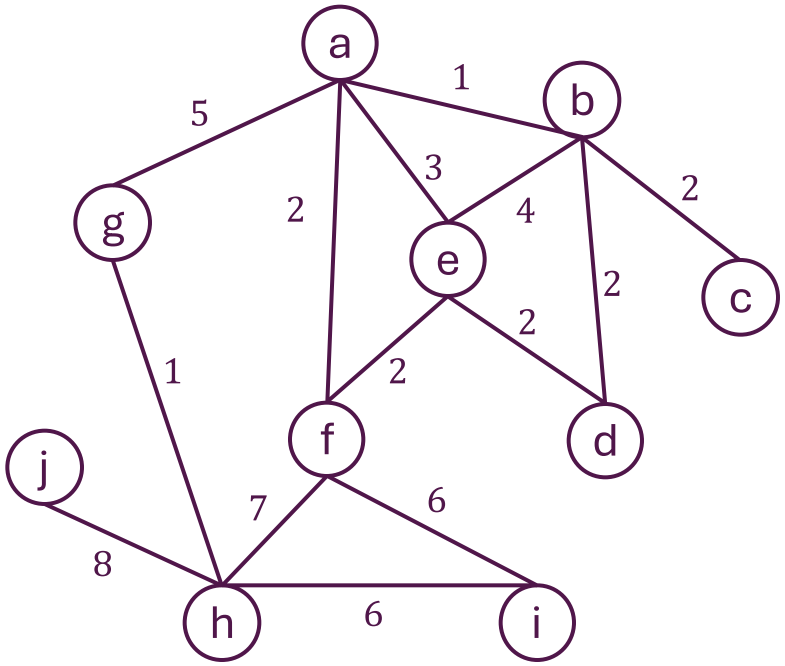
Instructor: Lily

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)

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.

Multiply

```
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
```

Average – Correctness

```
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)
```

Average – Termination

```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 = 0
  while i < len(A):
    total += A[i]
    i += 1
  return total / len(A)
```

Average – Run-time

```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 = 0
  while i < len(A):
    total += A[i]
    i += 1
  return total / len(A)
```

Now your turn! Selection Sort

```
def selection_sort(A):
  # Pre: A is a non-empty list of integers
  n = len(A)
  i = 0
  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
```

Selection Sort – Correctness

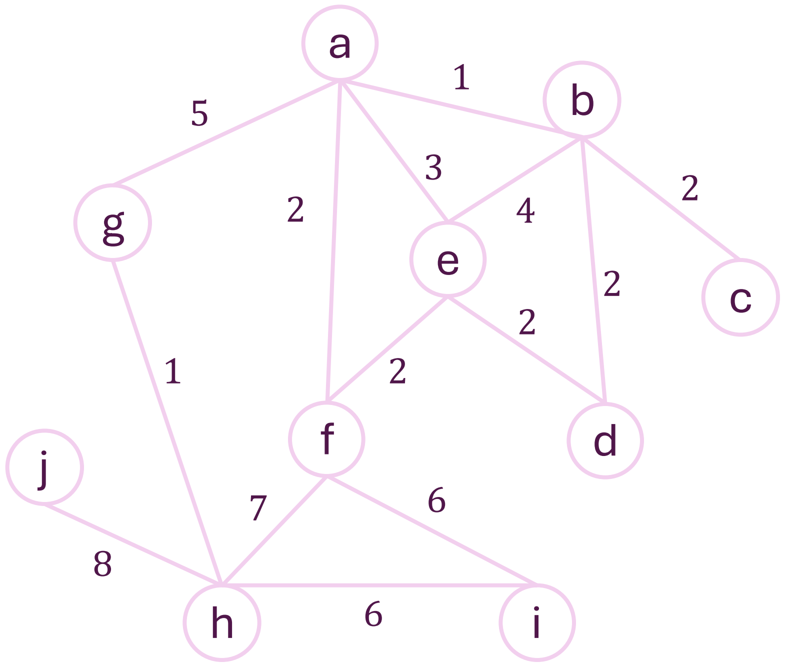
```
def selection_sort(A):
  # Pre: A is a non-empty list of integers
  n = len(A)
  i = 0
  for i in range(n):
    min_index = i
    for j in range(i+1, n):
      if A[j] < A[min_index]:
        min_index = j
    swap(A[i], A[min_index])
  return
```

Selection Sort – Termination/ Run Time

```
def selection_sort(A):
  # Pre: A is a non-empty list of integers
  n = len(A)
  i = 0
  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
```

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
```



Prim’s Algorithm – Correctness

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

Prim’s Algorithm – Termination/ Run Time

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

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