Linear search in unsorted lists

Goal: Given (unsorted) list \(b\), search range \(h..k\) with \(k \geq h\) (and \(h\) and \(k\) valid indices for \(b\)), and target value \(v\), return index \(n\) of \(v\)'s first occurrence in \(b[h..k]\) (-1 if not found).

Restated as postcondition: if \(n = -1\), then \(v\) is not in \(b[h..k]\). Otherwise, \(v = b[n]\) and \(v\) is not in \(b[h..n-1]\).

Idea: keep an index \(i\), marking position of next thing unchecked; everything to its left has been verified to not be \(v\).

\[
\begin{array}{c|c|c|c|c}
\text{v not here} & \text{h} & \text{i} & \text{k} & \text{?} \\
\end{array}
\]

Linear Search

\[
\text{def linear_search}(b, h, k, v):
\]

1. \(i = h\)  
   \[\text{inv: } \text{b}[\text{h}..\text{i}-1] \text{ < v}\]
2. \(\text{while } i \leq k \text{ and } b[i] \neq v:\)
   \[i = i + 1\]
   \[\text{post: } b[i] = v \text{ and } v \text{ not in } b[h..i-1], \text{ or, }\]
   \[i = k+1 \text{ and } v \text{ not in } b[h..k]\]
3. \(n = i\) if \(i < k\) else -1
4. \(\text{return } n\)

Analyzing the Loop

1. Does the initialization make \(\text{inv}\) true?
2. Is \(\text{post}\) true when \(\text{inv}\) is true and \(\text{condition}\) is false?
3. Does the loop body make \(\text{progress}\)?
4. Does the loop body keep \(\text{inv}\) true?

Binary search in sorted lists

Goal: Given sorted list \(b\), search range \(h..k\) with \(k \geq h\) (and \(h\) and \(k\) valid indices for \(b\)), and target value \(v\), return index \(n\) of \(v\)'s first occurrence in \(b[h..k]\) (-1 if not found).

Restated as postcondition: if \(n = -1\), then \(v\) is not in \(b[h..k]\). Otherwise, \(v = b[n]\) and \(v\) is not in \(b[h..n-1]\).

Idea: keep indices \(i\) and \(j\), marking position of next thing not known to be < \(v\), and the first thing known to be \(\geq v\). Check in the middle.

\[
\begin{array}{c|c|c|c|c|c}
\text{<v} & \text{h} & \text{i} & \text{j} & \text{>=v} & \text{k} \\
\end{array}
\]

def bin_search(b, h, k, v):

1. \(i = h; j = k\)  
2. \(\text{while } i < j:\)
   \[j = (i + j)/2\]
   \[\text{if } b[j] < v:\]
   \[i = j\]
   \[\text{else:}\]
   \[j = mid\]
3. \(\text{return } i\) if \(i \leq k \text{ and } b[i] = v\) else -1
4. \(\text{post: } b[n] = v \text{ and } v \text{ not in } b[h..n-1], \text{ or, } n = -1 \text{ and } v \text{ not in } b[h..k]\)

Sorting: Selection Sort

Selection Sort:

\[
\begin{array}{c|c|c|c|c|c}
\text{pre: } b & ? & n & \text{post: } b \text{ sorted} & n \\
\end{array}
\]

\[
\text{def sort}(b):
\]

1. \(\text{# Sort list } b \text{ in place.}^{**}
2. \(\text{# Note the swap of the reds}\)
3. \(\text{# post: } b[0..n-1] \text{ sorted}\)

Selection Sort:

\[
\begin{array}{c|c|c|c|c|c}
\text{inv: } b & \text{sorted, a } b[i..n] & \text{ } & \text{a } b[0..i-1] \text{ or } ? \text{ if } i = 0 & n \\
\end{array}
\]

Selection Sort:

\[
\begin{array}{c|c|c|c|c|c}
\text{INITIALIZE AND COMPLETE: } \\
\text{def sort}(b):
\]

1. \(\text{# Sort list } b \text{ in place.}^{**}
2. \(\text{# Note the swap of the reds}\)
3. \(\text{# post: } b[0..n-1] \text{ sorted}\)

Selection Sort:

\[
\begin{array}{c|c|c|c|c|c}
\text{def sort}(b):
\]

1. \(\text{# Sort list } b \text{ in place.}^{**}
2. \(\text{# Note the swap of the reds}\)
3. \(\text{# post: } b[0..n-1] \text{ sorted}\)

Selection Sort:

\[
\begin{array}{c|c|c|c|c|c}
\text{def sort}(b):
\]

1. \(\text{# Sort list } b \text{ in place.}^{**}
2. \(\text{# Note the swap of the reds}\)
3. \(\text{# post: } b[0..n-1] \text{ sorted}\)

Selection Sort:

\[
\begin{array}{c|c|c|c|c|c}
\text{def sort}(b):
\]

1. \(\text{# Sort list } b \text{ in place.}^{**}
2. \(\text{# Note the swap of the reds}\)
3. \(\text{# post: } b[0..n-1] \text{ sorted}\)
Helper for Selection Sort

```python
def min_index(b, i, n):
    """return index of min item in b[i...n-1]. Pre: i <= n."""
    j = i
    k = i + 1
    # inv: b[j] is min of b[i..k-1]
    while k < n:
        j = k if b[k] < b[j] else j
        k += 1
    # post: b[j] is min of b[i..n-1]

inv: b[j] is min of these
```

Famous "Sort-Like" Example

• Dutch national flag: tri-color
  * Sequence of h,k of red (<0), white (=0), blue (>0) "pixels"
  * Arrange to put <0 first, then =0, then >0, return "split pts"

Hutch National Flag Algorithm

```python
def dnf(b, h, k):
    """Returns: split-points as a tuple (i, j)"""
    # init
    # inv: b[h..t-1] < 0, b[t..i-1] ?, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        if b[i-1] < 0:
            b[i-1], b[t] = b[t], b[i-1]
            t = t+1
        elif b[i-1] == 0:
            i = i-1
        else:
            b[-1], b[j] = b[j], b[i-1]
            i = i-1; j = j-1
    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)
```

Hutch National Flag Algorithm

```python
def dnf(b, h, k):
    """Returns: partition indices as a tuple (i, j)"""
    t = h;
    i = k+1, j = k;
    # inv: b[h..t-1] < 0, b[t..i-1] ?, b[i..j] = 0, b[j+1..k] > 0
    while t < i:
        if b[i-1] < 0:
            b[i-1], b[t] = b[t], b[i-1]
            t = t+1
        elif b[i-1] == 0:
            i = i-1
        else:
            b[-1], b[j] = b[j], b[i-1]
            i = i-1; j = j-1
    # post: b[h..i-1] < 0, b[i..j] = 0, b[j+1..k] > 0
    return (i, j)
```