S. KIMMEL

Goals

- Create good loop invariants
- Use loop invariants so prove DP correctness
- Describe encoding problem

Loop Invariants ... Tips for Success

[See Bubble Sort Pseudocode on next page]
BS

1. Invariant should involve iterated variable.

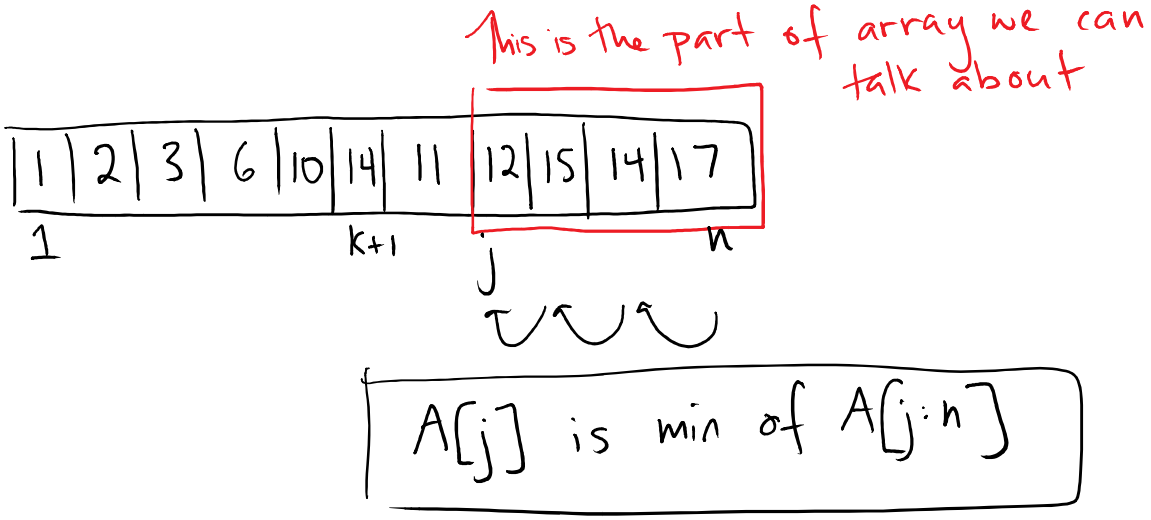
   ex: • Inner Loop of BS ⇒ for j = n to k+1
   "j" is iterated variable
   • Loop Invariant: A[j] is min{A[j:n]}

2. Invariant generally doesn't involve things outside domain of loop

   • Inner loop ⇒ j = n to k+1
   • A[1:k] is sorted ✗ wrong: A[1:k] is outside purview of inner loop.
   • A[k:j] is sorted ✗ wrong: j hasn't gotten to this part of the array, so we can't be maintaining anything here.

S.KIMMEL

3. To figure out loop invariant, freeze in middle of loop



Later: when termination requires A's elements are same, add that condition

Input : Array A of integers of length n

Output: Array containing sorted elements of A

```
1 for k = 1 to n − 1 do
2 |  for j = n to k + 1 do
3 |  |  if A[j] < A[j − 1] then
4 |  |  |  Swap A[j] and A[j − 1];
5 |  |  end
6 |  end
7 end
8 return A;
```

Algorithm 2: `BubbleSort`(A)

S. KIMMEL

Knapsack Problem

Input: • n items, each has
- value v_j
- size w_j

$\left.\begin{array}{l} \\ \\ \end{array}\right\} \mathbb{Z}^+$

• Capacity W

Output: A subset $S \subseteq \{1, 2, \ldots, n\}$ that maximizes $\sum_{i \in S} v_i = V(S)$ (objective function)
and satisfies $W(S) = \sum_{i \in S} w_i \leq W$ (constraints)

$K_{i,r}$ = optimal subset if only 1st i items are allowed, capacity r.

D.P. Pseudocode

```
// create array V of objective function values of K_{i,r}
for r = 0 to W:
    V[0, r] = 0

for i = 1 to n:
    for r = 0 to W:
        V[i, r] = max { V[i-1, r], V[i-1, r - w_i] + v_i }
```

This option Not allowed if $r - w_i < 0$

S. KIMMEL

optimal set of items

// pseudocode to get S using array V

$S = \emptyset$

$i = n$

$r = W$

Need check that doesn't go out of bounds

while $i > 0$:

  if $V[i, r] = V[i-1, r-w_i] + v_i$

    $S = S + i$

    $r = r - w_i$

  $i --$

Group Work

- Loop invariants for 2 loops
- Initialization / Termination
- Maintenance

★ Put proofs from step 2 in Maintenance of first loop ★

S. KIMMEL

Proof of Correctness

$V[j, g] = V_{j,g}$ ← optimal objective function value for $K_{j,g}$

Loop 1 Invariant: $V[j, g] = V_{j,g}$ for all elements of V up to $V[i, r]$

- Initialization: $i = 1$, $r = 0$. $V[0, r] = V_{0,r}$ $\forall$ r ✓
- Maintenance: We will prove the loop sets $V[i, r]$ to have the correct value. Let S be optimal solution to $K_{i,r}$. Then either $i \notin S$ or $i \in S$.

If $i \notin S$, then S is optimal soln to $K_{i-1,r}$

for contradiction, suppose S is not opt soln for $K_{i-1,r}$ Then there exists an optimal solution S' for $K_{i-1,r}$ with $V(S') > V(S)$. But S' is also a solution to $K_{i,r}$ with $V(S') > V(S)$ and $n \notin S'$, so S is not the optimal solution to $K_{i,r}$, a contradiction.

Similarly, if $i \in S$, $S - \{i\}$ is an optimal solution to $K_{i-1, r-w_i}$. Since the optimal

solution is one of these two options, we take the one with the larger objective value. So

$$V_{i,r} = \max\left\{ V_{i-1,r},\ V_{i-1,r-w_i} + v_i \right\}$$

↑ if $r - w_i > 0$.

By our loop invariant, this is

$$V_{ir} = \max\left\{ V[i-1,r],\ V[i-1,r-w_i] + v_i \right\}$$

↑ if $r - w_i > 0$

which is what our algorithm sets $V[i,r]$ to be.

Termination: $V[i,r] = V_{i,r}$ for all $1 \le i \le n$, $1 \le i \le W$.

S. KIMMEL

Pf that loop 2 is correct. Let S be opt solution. Can assume $V[i,w]$ is value of opt. sol. on $K_{i,w}$

Loop Invariant: • S' contains all elements of S larger than i.

• $r = W - W(S')$

Initially: $i = n$, $S = \phi$, $r = W$

Maintenance: Suppose loop invariant is true going into loop. Then S' contains elements of S larger than i, so we just need to figure out those less than or equal to i. Since we've already used up capacity $W(S')$, we need to solve $K_{i, W-W(S')}$ to figure out remaining elements of S'

We now determine if $i \in$ opt. Soln for $K_{i, W-W(S')}$

Only two options (i), or (ii) and at least one is true, so either $V[i, W-W(S')] = V[i-1, W-W(S')]$ or $V[i, W-W(S')] = V[i-1, W-W(S')-w_i] + v_i$, and which one it is tells us whether i is in optimal solution. In either case, invariant is maintained

S. KIMMEL

Huffman Codes

Binary Code: each letter of alphabet $\Sigma \rightarrow$ binary string

e.g. $\Sigma = \{a, b, \ldots z, \_, ', !, ? \ldots\}$ → each letter gets unique string from $\{0,1\}^5$

32 letters

→ ASCII

e.g. $\Sigma = \{a, b, c\}$

Suppose you have a message where a occurs 50%, b occurs 30%, c occurs 20%.

Q. What is the best and most efficient binary encoding of a, b, c to send message?

A) a → 00
b → 01
c → 10

B) a → 0
b → 1
c → 01

C) a = 0
b = 11
c = 01

D) a = 0
b = 10
c = 11

S. KIMMEL

Huffman Codes

Binary Code: each letter of alphabet $\Sigma \rightarrow$ binary string

e.g. $\Sigma = \{a, b, \ldots z, \_, ', ?, \ldots\}$ → each letter gets unique string from $\{0,1\}^5$

32 letters

→ ASCII

e.g. $\Sigma = \{a, b, c\}$

Suppose you have a message where a occurs 50%, b occurs 30%, c occurs 20%.

Q. What is the best and most efficient binary encoding of a, b, c to send message?

A)
a → 00
b → 01
c → 10

0110
b c

Average length: 2

B)
a → 0
b → 1
c → 01

0110 = abba
or
cbba

C)
a = 0
b = 11
c = 01

0110

- a or beginning of c
- start of b, end of c
- start of b, end of b
- Need to go to 4th to figure out

D)
a = 0
b = 10
c = 11

0110
a c a

average length

$.5 \cdot 1 + .5\ 2$
$= 1.5$

Length of encoding of i

Prob. of i

Average length $= \sum_{i \in \Sigma} \ell(f(i)) p_i$