COMP90038
Algorithms and Complexity

Lecture 16: Time/Space Tradeoffs—String Search Revisited
(with thanks to Harald Søndergaard & Michael Kirley)

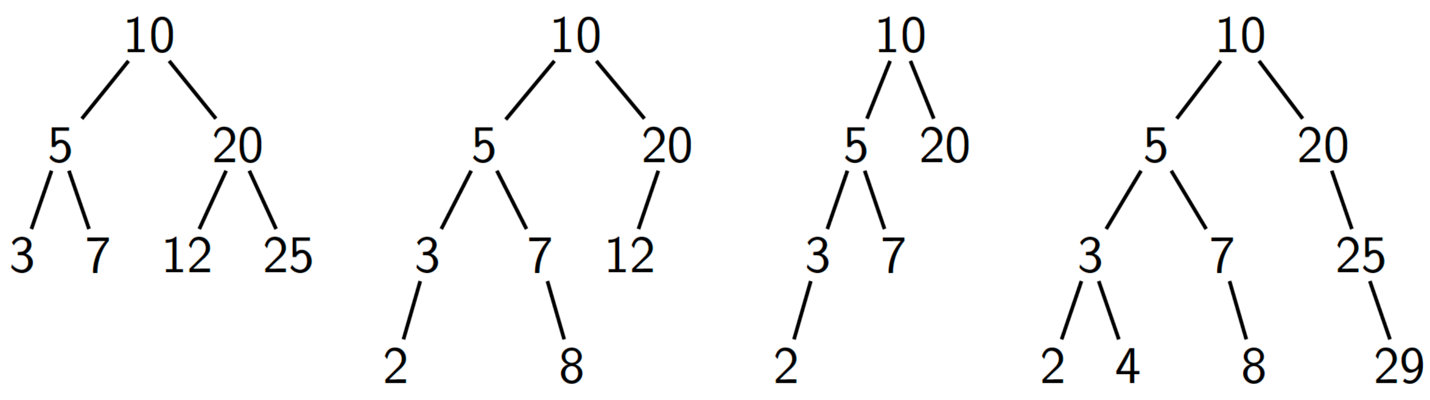
Casey Myers
Casey.Myers@unimelb.edu.au
David Caro Building (Physics) 274

Review from Lecture 15: AVL Trees

- For a binary (sub-) tree, let the balance factor be the difference between the height of its left sub-tree and that of its right sub-tree.

- An AVL tree is a BST in which the balance factor is -1, 0, or 1, for every sub-tree.

- The notion of "balance" that is implied by the AVL condition is sufficient to guarantee that the depth of an AVL tree with n nodes is $\Theta(\log n)$.

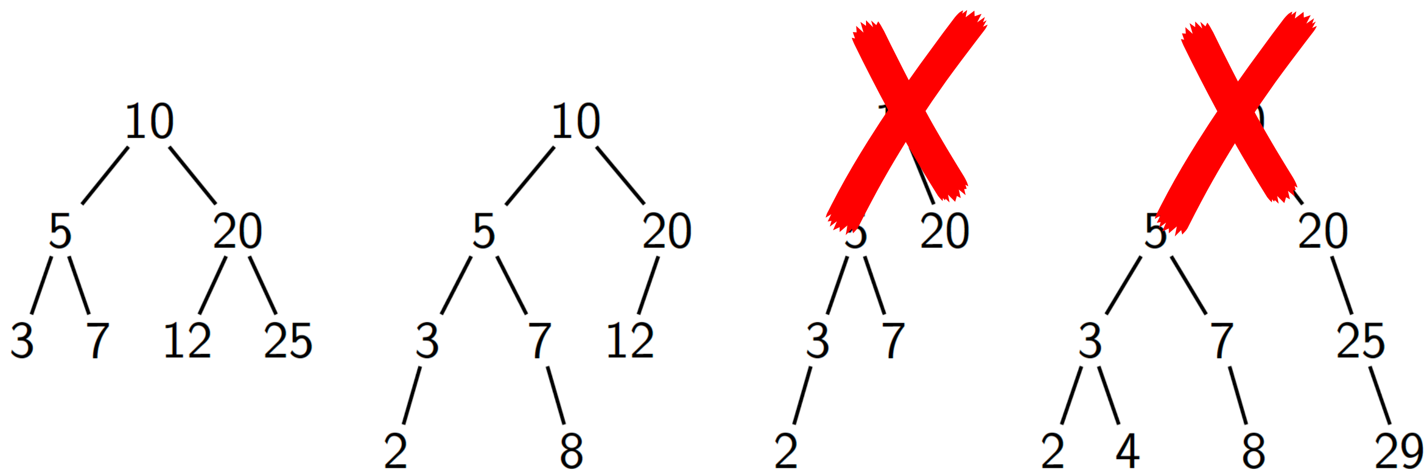


Review from Lecture 15: AVL Trees

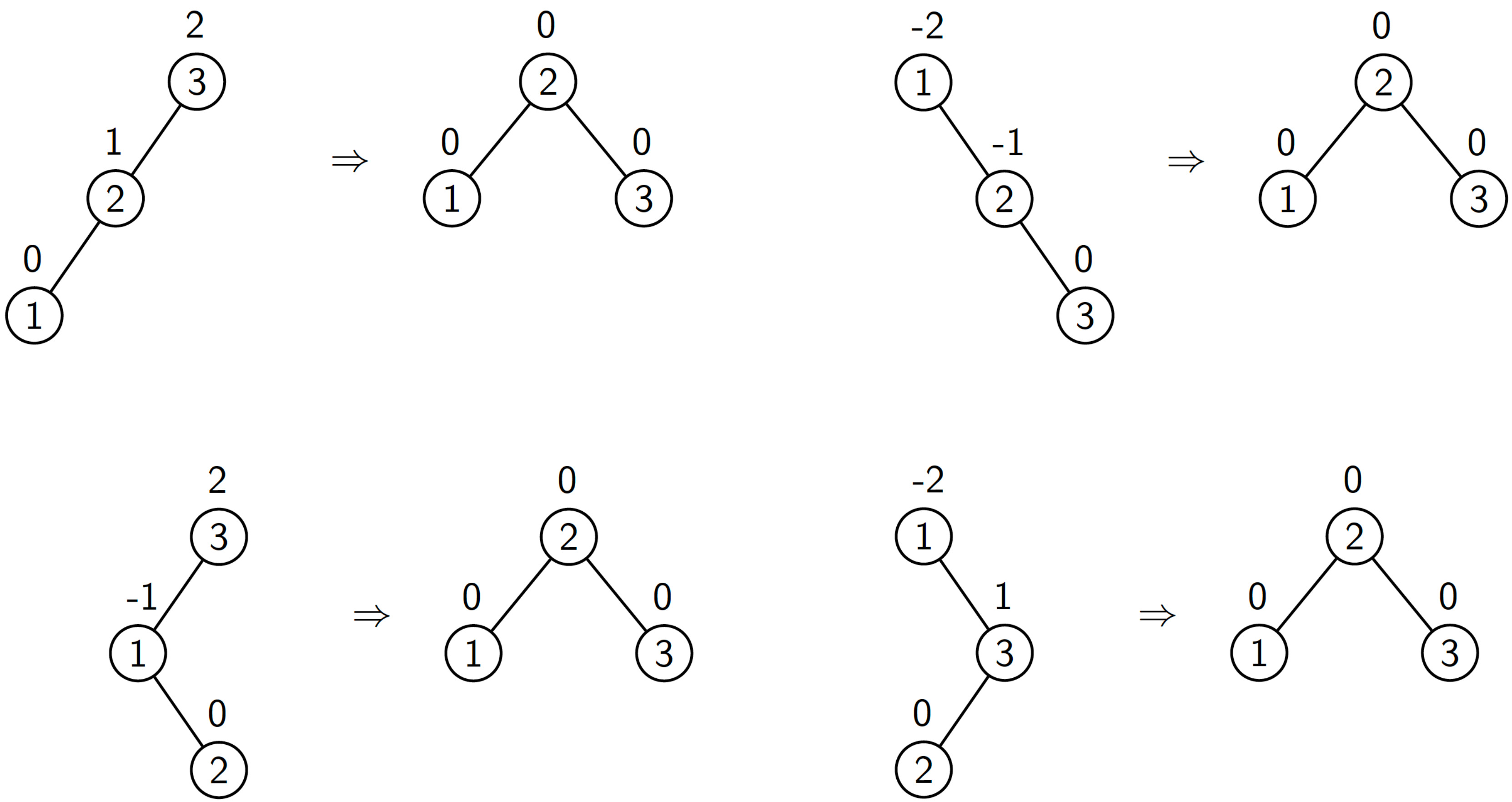
- For a binary (sub-) tree, let the balance factor be the difference between the height of its left sub-tree and that of its right sub-tree.

- An AVL tree is a BST in which the balance factor is -1, 0, or 1, for every sub-tree.

- The notion of "balance" that is implied by the AVL condition is sufficient to guarantee that the depth of an AVL tree with n nodes is $\Theta(\log n)$.

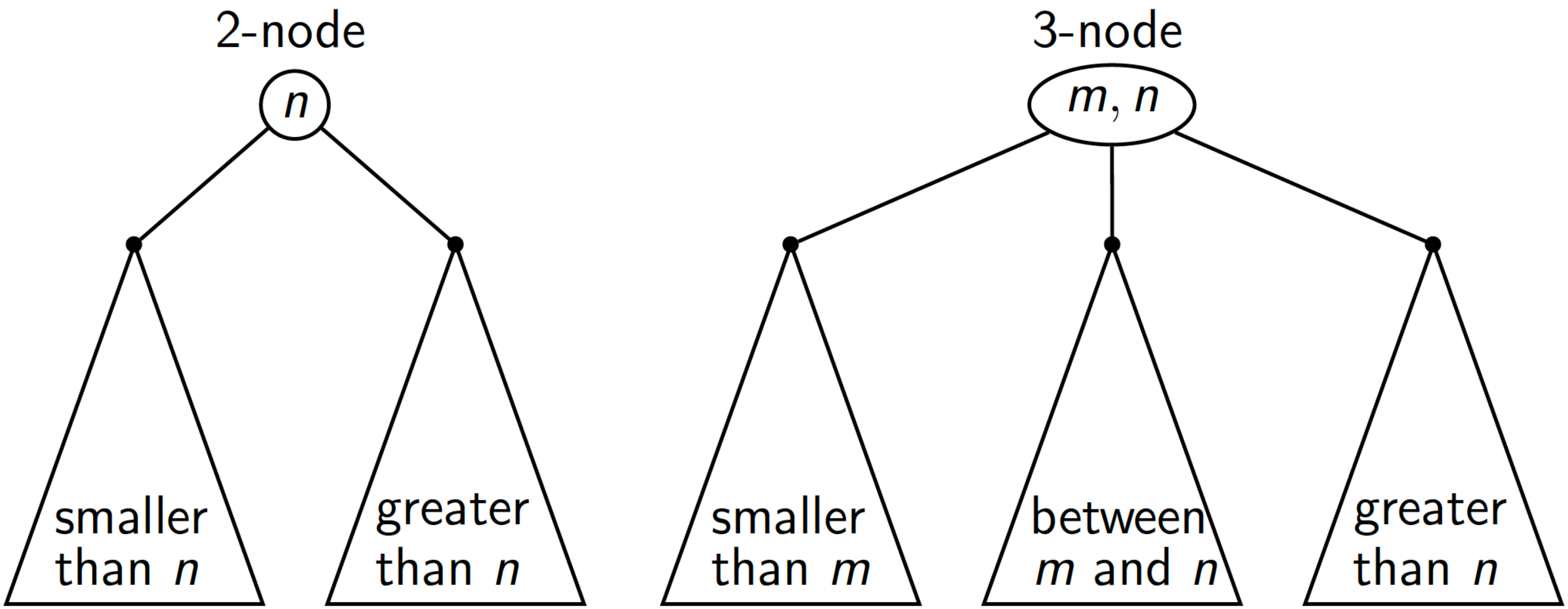


Review from Lecture 15: AVL Trees Rotations



Review from Lecture 15: 2-3 Trees

- A node that holds two items (a so-called 3-node) has (at most) three children.
- A 2-3 tree stores up to 2 items in each tree node.
- Insertions, splits and promotions are used to grow and balance the tree.



Spending Space to Save Time

- Often we can find ways of decreasing the time required to solve a problem, by using additional memory in a clever way.

- For example, in Lecture 6 we considered the simple recursive way of finding the nth Fibonacci number and discovered that the algorithm uses exponential time.

- However, suppose the same algorithm uses a table to tabulate the function FIB as we go:
  - As soon as an intermediate result FIB(i) has been found, it is not simply returned to the caller; the value is first placed in slot i of a table (an array). Each call to FIB first looks in this table to see if the required value is there, and only if it is not, the usual recursive process kicks in.

Fibonacci Numbers with Tabulation

- We assume that, from the outset, all entries of the table F are 0.

function $\textsc{Fib}(n)$
    if $n = 0$ **or** $n = 1$ **then**
        return 1
    $result \leftarrow F[n]$
    if $result = 0$ **then**
        $result \leftarrow \textsc{Fib}(n-1) + \textsc{Fib}(n-2)$
        $F[n] \leftarrow result$
    return $result$

Fibonacci Numbers with Tabulation

- We assume that, from the outset, all entries of the table F are 0.

function $\text{Fib}(n)$
    if $n = 0$ **then**
        return 1
    if $n = 1$ **then**
        return 1
    return $\text{Fib}(n-1) + \text{Fib}(n-2)$

Complexity is **exponential** in n

Fibonacci Numbers with Tabulation

- We assume that, from the outset, all entries of the table F are 0.

function $\text{Fib}(n)$
  if $n = 0$ **then**
    return 1
  if $n = 1$ **then**
    return 1
  return $\text{Fib}(n-1) + \text{Fib}(n-2)$

Complexity is **exponential** in n

- (I show this code just so that you can see the principle; in Lecture 6 we already discovered a different linear-time algorithm, so here we don't really need tabulation.)

Fibonacci Numbers with Tabulation

- We assume that, from the outset, all entries of the table F are 0.

```
function FIB(n)
    if n = 0 or n = 1 then
        return 1
    result ← F[n]
    if result = 0 then
        result ← FIB(n − 1) + FIB(n − 2)
        F[n] ← result
    return result
```

Initial	$F = 0{,}0, \cdots, 0$
$n = 2$	result=FIB(1)+FIB(0) =1+1=2 $F = 0{,}0{,}2{,}0, \cdots, 0$

- (I show this code just so that you can see the principle; in Lecture 6 we already discovered a different linear-time algorithm, so here we don't really need tabulation.)

Fibonacci Numbers with Tabulation

- We assume that, from the outset, all entries of the table F are 0.

```
function FIB(n)
    if n = 0 or n = 1 then
        return 1
    result ← F[n]
    if result = 0 then
        result ← FIB(n − 1) + FIB(n − 2)
        F[n] ← result
    return result
```

Initial	$F = 0,0,\cdots,0$
$n = 2$	result=FIB(1)+FIB(0) =1+1=2 $F = 0,0,2,0,\cdots,0$
$n = 3$	result=FIB(2)+FIB(1) =2+1=3 $F = 0,0,2,3,0,\cdots,0$

- (I show this code just so that you can see the principle; in Lecture 6 we already discovered a different linear-time algorithm, so here we don't really need tabulation.)

Fibonacci Numbers with Tabulation

- We assume that, from the outset, all entries of the table F are 0.

function $\textsc{Fib}(n)$
    if $n = 0$ **or** $n = 1$ **then**
        return 1
    $result \leftarrow F[n]$
    if $result = 0$ **then**
        $result \leftarrow \textsc{Fib}(n-1) + \textsc{Fib}(n-2)$
        $F[n] \leftarrow result$
    return $result$

Initial	$F = 0{,}0, \cdots, 0$
$n = 2$	result=FIB(1)+FIB(0) =1+1=2 $F = 0{,}0{,}2{,}0, \cdots, 0$
$n = 2$	result=FIB(2)+FIB(1) =2+1=3 $F = 0{,}0{,}2{,}3{,}0, \cdots, 0$
$n = 4$	result=FIB(3)+FIB(2) =3+2=5 $F = 0{,}0{,}2{,}3{,}5{,}0, \cdots, 0$

- (I show this code just so that you can see the principle; in Lecture 6 we already discovered a different linear-time algorithm, so here we don't really need tabulation.)

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	0	0	0	0	0	0	0	0	0	0

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	0	0	0	0	0	0	1	0	0	0

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	0	0	0	1	0	0	1	0	0	0

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	0	0	0	2	0	0	1	0	0	0

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	0	0	0	2	0	0	1	0	1	0

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	0	1	0	2	0	0	1	0	1	0

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	1	0	2	0	0	1	0	1	0

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\;3\;3\;8\;1\;0\;8\;7\;9\;2\;5\;3\;5\;3\;1\;8\;7\;6\;5\;1\;2\;1\;5\;3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1

- Now use a second linear scan to make the counts cumulative:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1
$Cumu$	0	0	0	0	0	0	0	0	0	0

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1

- Now use a second linear scan to make the counts cumulative:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1
$Cumu$	1	0	0	0	0	0	0	0	0	0

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1

- Now use a second linear scan to make the counts cumulative:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1
$Cumu$	1	5	0	0	0	0	0	0	0	0

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1

- Now use a second linear scan to make the counts cumulative:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1
$Cumu$	1	5	7	0	0	0	0	0	0	0

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1

- Now use a second linear scan to make the counts cumulative:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1
$Cumu$	1	5	7	12	0	0	0	0	0	0

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1

- Now use a second linear scan to make the counts cumulative:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1
$Cumu$	1	5	7	12	12	0	0	0	0	0

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1

- Now use a second linear scan to make the counts cumulative:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1
$Cumu$	1	5	7	12	12	16	0	0	0	0

Sorting by Counting

- Suppose we need to sort large arrays, but we know that they will hold keys taken from a small, fixed set (so lots of duplicate keys).

- For example, suppose all keys are single digits in array A :

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

- Then we can, in a single linear scan count the occurrences of each key in array A and store the result in a small table:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1

- Now use a second linear scan to make the counts cumulative:

key	0	1	2	3	4	5	6	7	8	9
Occ	1	4	2	5	0	4	2	2	3	1
$Cumu$	1	5	7	12	12	16	18	20	23	24

Sorting by Counting

- We can now create a sorted array $S[1, \cdots, n]$ of the items by simply slotting items into pre-determined slots in S (a third linear scan).

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

key	0	1	2	3	4	5	6	7	8	9
$Cumu$	1	5	7	12	12	16	18	20	23	24

- Place the first record (with key 6) in S[18] and decrement $Cumu[6]$ (so that the next '6' will go into slot 17), and so on.

for $i \leftarrow 1$ **to** n **do**
    $S[Cumu[A[i]]] \leftarrow A[i]$
    $Cumu[A[i]] \leftarrow Cumu[A[i]] - 1$

Sorting by Counting

- We can now create a sorted array $S[1, \cdots, n]$ of the items by simply slotting items into pre-determined slots in S (a third linear scan).

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

key	0	1	2	3	4	5	6	7	8	9
$Cumu$	1	5	7	12	12	16	18	20	23	24

- Place the first record (with key 6) in S[18] and decrement $Cumu[6]$ (so that the next '6' will go into slot 17), and so on.

for $i \leftarrow 1$ **to** n **do**
  $S[Cumu[A[i]]] \leftarrow A[i]$
  $Cumu[A[i]] \leftarrow Cumu[A[i]] - 1$

$i = 1$: $S\left[Cumu[A[1]]\right] = S[Cumu[6]] = S[18] \leftarrow 6$
$Cumu[A[1]] = Cumu[6] \leftarrow$18-1=17

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
6	3	3	8	1	0	8	7	9	2	5	3	5	3	1	8	7	6	5	1	2	1	5	3

Sorting by Counting

- We can now create a sorted array $S[1, \cdots, n]$ of the items by simply slotting items into pre-determined slots in S (a third linear scan).

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

key	0	1	2	3	4	5	6	7	8	9
$Cumu$	1	5	7	12	12	16	17	20	23	24

- Place the first record (with key 6) in S[18] and decrement $Cumu[6]$ (so that the next '6' will go into slot 17), and so on.

$$i = 1:\ S\left[Cumu[A[1]]\right] = S[Cumu[6]] = S[18] \leftarrow 6$$
$$Cumu[A[1]] = Cumu[6] \leftarrow 18\text{-}1\text{=}17$$

for $i \leftarrow 1$ **to** n **do**
  $S[Cumu[A[i]]] \leftarrow A[i]$
  $Cumu[A[i]] \leftarrow Cumu[A[i]] - 1$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
6	3	3	8	1	0	8	7	9	2	5	3	5	3	1	8	7	6	5	1	2	1	5	3
																	6						

Sorting by Counting

- We can now create a sorted array $S[1, \cdots, n]$ of the items by simply slotting items into pre-determined slots in S (a third linear scan).

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

key	0	1	2	3	4	5	6	7	8	9
$Cumu$	1	5	7	12	12	16	17	20	23	24

- Place the first record (with key 6) in S[18] and decrement $Cumu[6]$ (so that the next '6' will go into slot 17), and so on.

for $i \leftarrow 1$ **to** n **do**
    $S[Cumu[A[i]]] \leftarrow A[i]$
    $Cumu[A[i]] \leftarrow Cumu[A[i]] - 1$

$i = 2$: $S\left[Cumu[A[2]]\right] = S[Cumu[3]] = S[12] \leftarrow 3$
$Cumu[A[2]] = Cumu[3] \leftarrow$12-1=11

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
6	3	3	8	1	0	8	7	9	2	5	3	5	3	1	8	7	6	5	1	2	1	5	3
																	6						

Sorting by Counting

- We can now create a sorted array $S[1, \cdots, n]$ of the items by simply slotting items into pre-determined slots in S (a third linear scan).

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

key	0	1	2	3	4	5	6	7	8	9
$Cumu$	1	5	7	11	12	16	17	20	23	24

- Place the first record (with key 6) in S[18] and decrement $Cumu[6]$ (so that the next '6' will go into slot 17), and so on.

for $i \leftarrow 1$ **to** n **do**
  $S[Cumu[A[i]]] \leftarrow A[i]$
  $Cumu[A[i]] \leftarrow Cumu[A[i]] - 1$

$i = 2$: $S\left[Cumu[A[2]]\right] = S[Cumu[3]] = S[12] \leftarrow 3$
$Cumu\left[A[2]\right] = Cumu[3] \leftarrow$12-1=11

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
6	3	3	8	1	0	8	7	9	2	5	3	5	3	1	8	7	6	5	1	2	1	5	3
											3						6						

Sorting by Counting

- We can now create a sorted array $S[1, \cdots, n]$ of the items by simply slotting items into pre-determined slots in S (a third linear scan).

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

key	0	1	2	3	4	5	6	7	8	9
$Cumu$	1	5	7	11	12	16	17	20	23	24

- Place the first record (with key 6) in S[18] and decrement $Cumu[6]$ (so that the next '6' will go into slot 17), and so on.

for $i \leftarrow 1$ **to** n **do**
  $S[Cumu[A[i]]] \leftarrow A[i]$
  $Cumu[A[i]] \leftarrow Cumu[A[i]] - 1$

$i = 3$: $S\left[Cumu[A[3]]\right] = S[Cumu[3]] = S[11] \leftarrow 3$
$Cumu[A[3]] = Cumu[3] \leftarrow$ 11-1=10

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
6	3	3	8	1	0	8	7	9	2	5	3	5	3	1	8	7	6	5	1	2	1	5	3
											3						6						

Sorting by Counting

- We can now create a sorted array $S[1, \cdots, n]$ of the items by simply slotting items into pre-determined slots in S (a third linear scan).

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

key	0	1	2	3	4	5	6	7	8	9
$Cumu$	1	5	7	10	12	16	17	20	23	24

- Place the first record (with key 6) in S[18] and decrement $Cumu[6]$ (so that the next ‘6’ will go into slot 17), and so on.

for $i \leftarrow 1$ **to** n **do**
  $S[Cumu[A[i]]] \leftarrow A[i]$
  $Cumu[A[i]] \leftarrow Cumu[A[i]] - 1$

$i = 3$: $S\left[Cumu[A[3]]\right] = S[Cumu[3]] = S[11] \leftarrow 3$
$Cumu\left[A[3]\right] = Cumu[3] \leftarrow$11-1=10

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
6	3	3	8	1	0	8	7	9	2	5	3	5	3	1	8	7	6	5	1	2	1	5	3
										3	3						6						

Sorting by Counting

- We can now create a sorted array $S[1, \cdots, n]$ of the items by simply slotting items into pre-determined slots in S (a third linear scan).

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

key	0	1	2	3	4	5	6	7	8	9
$Cumu$	1	5	7	10	12	16	17	20	23	24

- Place the first record (with key 6) in S[18] and decrement $Cumu[6]$ (so that the next '6' will go into slot 17), and so on.

for $i \leftarrow 1$ **to** n **do**
$\quad S[Cumu[A[i]]] \leftarrow A[i]$
$\quad Cumu[A[i]] \leftarrow Cumu[A[i]] - 1$

$i = 4$: $S\left[Cumu[A[4]]\right] = S[Cumu[8]] = S[23] \leftarrow 8$

$Cumu[A[4]] = Cumu[8] \leftarrow$ 23-1=22

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
6	3	3	8	1	0	8	7	9	2	5	3	5	3	1	8	7	6	5	1	2	1	5	3
										3	3						6						

Sorting by Counting

- We can now create a sorted array $S[1, \cdots, n]$ of the items by simply slotting items into pre-determined slots in S (a third linear scan).

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

key	0	1	2	3	4	5	6	7	8	9
Cumu	1	5	7	10	12	16	17	20	22	24

- Place the first record (with key 6) in S[18] and decrement $Cumu[6]$ (so that the next '6' will go into slot 17), and so on.

for $i \leftarrow 1$ **to** n **do**
  $S[Cumu[A[i]]] \leftarrow A[i]$
  $Cumu[A[i]] \leftarrow Cumu[A[i]] - 1$

$i = 4$: $S\left[Cumu[A[4]]\right] = S[Cumu[8]] = S[23] \leftarrow 8$
$Cumu[A[4]] = Cumu[8] \leftarrow$ 23-1=22

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
6	3	3	8	1	0	8	7	9	2	5	3	5	3	1	8	7	6	5	1	2	1	5	3
										3	3						6					8	

Sorting by Counting

- We can now create a sorted array $S[1, \cdots, n]$ of the items by simply slotting items into pre-determined slots in S (a third linear scan).

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

key	0	1	2	3	4	5	6	7	8	9
$Cumu$	1	5	7	10	12	16	17	20	22	24

- Place the first record (with key 6) in S[18] and decrement $Cumu[6]$ (so that the next '6' will go into slot 17), and so on.

$i = 5$: $S\left[Cumu[A[5]]\right] = S[Cumu[1]] = S[5] \leftarrow 1$

$Cumu[A[5]] = Cumu[1] \leftarrow$ 5-1=4

for $i \leftarrow 1$ **to** n **do**
  $S[Cumu[A[i]]] \leftarrow A[i]$
  $Cumu[A[i]] \leftarrow Cumu[A[i]] - 1$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
6	3	3	8	1	0	8	7	9	2	5	3	5	3	1	8	7	6	5	1	2	1	5	3
										3	3						6					8	

Sorting by Counting

- We can now create a sorted array $S[1, \cdots, n]$ of the items by simply slotting items into pre-determined slots in S (a third linear scan).

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

key	0	1	2	3	4	5	6	7	8	9
$Cumu$	1	4	7	10	12	16	17	20	22	24

- Place the first record (with key 6) in S[18] and decrement $Cumu[6]$ (so that the next '6' will go into slot 17), and so on.

$$\textbf{for } i \leftarrow 1 \textbf{ to } n \textbf{ do}$$
$$\quad S[Cumu[A[i]]] \leftarrow A[i]$$
$$\quad Cumu[A[i]] \leftarrow Cumu[A[i]] - 1$$

$i = 5$: $S\big[Cumu[A[5]]\big] = S[Cumu[1]] = S[5] \leftarrow 1$

$Cumu[A[5]] = Cumu[1] \leftarrow$ 5-1=4

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
6	3	3	8	1	0	8	7	9	2	5	3	5	3	1	8	7	6	5	1	2	1	5	3
				1						3	3						6					8	

Sorting by Counting

- We can now create a sorted array $S[1, \cdots, n]$ of the items by simply slotting items into pre-determined slots in S (a third linear scan).

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

key	0	1	2	3	4	5	6	7	8	9
$Cumu$	0	1	5	8	12	12	16	18	20	23

- Place the first record (with key 6) in S[18] and decrement $Cumu[6]$ (so that the next '6' will go into slot 17), and so on.

$$i = 24\text{: } S\left[Cumu[A[24]]\right] = S[Cumu[3]] = S[8] \leftarrow 3$$

$$Cumu[A[24]] = Cumu[3] \leftarrow 8\text{-}1\text{=}7$$

$$\begin{aligned}
&\textbf{for } i \leftarrow 1 \textbf{ to } n \textbf{ do}\\
&\quad S[Cumu[A[i]]] \leftarrow A[i]\\
&\quad Cumu[A[i]] \leftarrow Cumu[A[i]] - 1
\end{aligned}$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
6	3	3	8	1	0	8	7	9	2	5	3	5	3	1	8	7	6	5	1	2	1	5	3
0	1	1	1	1	2	2		3	3	3	3	5	5	5	5	6	6	7	7	8	8	8	9

Sorting by Counting

- We can now create a sorted array $S[1, \cdots, n]$ of the items by simply slotting items into pre-determined slots in S (a third linear scan).

$$A = 6\ 3\ 3\ 8\ 1\ 0\ 8\ 7\ 9\ 2\ 5\ 3\ 5\ 3\ 1\ 8\ 7\ 6\ 5\ 1\ 2\ 1\ 5\ 3$$

key	0	1	2	3	4	5	6	7	8	9
$Cumu$	0	1	5	7	12	12	16	18	20	23

- Place the first record (with key 6) in S[18] and decrement $Cumu[6]$ (so that the next '6' will go into slot 17), and so on.

for $i \leftarrow 1$ **to** n **do**
  $S[Cumu[A[i]]] \leftarrow A[i]$
  $Cumu[A[i]] \leftarrow Cumu[A[i]] - 1$

$i = 24$: $S\left[Cumu[A[24]]\right] = S[Cumu[3]] = S[8] \leftarrow 3$
$Cumu[A[24]] = Cumu[3] \leftarrow$ 8-1=7

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
6	3	3	8	1	0	8	7	9	2	5	3	5	3	1	8	7	6	5	1	2	1	5	3
0	1	1	1	1	2	2	3	3	3	3	3	5	5	5	5	6	6	7	7	8	8	8	9

Sorting by Counting

- Note that this gives is a linear-time sorting algorithm (for the cost of some extra space).
- However, it only works in situations where we have a small range of keys, known in advance.
- The method never performs a key-to-key comparison.
- The time complexity of key-comparison based sorting has been proven to be $\Omega(n \log n)$.

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- “Strings” are usually built from a small, pre-determined alphabet.

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- “Strings” are usually built from a small, pre-determined alphabet.

for $i \leftarrow 0$ to $n - m$ **do**
    $j \leftarrow 0$
    while $j < m$ and $p[j] = t[i + j]$ **do**
        $j \leftarrow j + 1$
    if $j = m$ **then**
        return i
return -1

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- “Strings” are usually built from a small, pre-determined alphabet.

for $i \leftarrow 0$ to $n - m$ **do**
    $j \leftarrow 0$
    while $j < m$ and $p[j] = t[i + j]$ **do**
        $j \leftarrow j + 1$
    if $j = m$ **then**
        return i
return -1

N O B O D Y _ N O T I C E D _ H I M

N O T

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- "Strings" are usually built from a small, pre-determined alphabet.

N O B O D Y _ N O T I C E D _ H I M

N O T

$$
\begin{array}{l}
\textbf{for } i \leftarrow 0 \text{ to } n - m \textbf{ do} \\
\quad j \leftarrow 0 \\
\quad \textbf{while } j < m \text{ and } p[j] = t[i+j] \textbf{ do} \\
\quad\quad j \leftarrow j + 1 \\
\quad \textbf{if } j = m \textbf{ then} \\
\quad\quad \textbf{return } i \\
\textbf{return } -1
\end{array}
$$

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- "Strings" are usually built from a small, pre-determined alphabet.

N O B O D Y _ N O T I C E D _ H I M

N O T

$$
\begin{array}{l}
\textbf{for } i \leftarrow 0 \text{ to } n - m \textbf{ do} \\
\quad j \leftarrow 0 \\
\quad \textbf{while } j < m \text{ and } p[j] = t[i+j] \textbf{ do} \\
\quad\quad j \leftarrow j + 1 \\
\quad \textbf{if } j = m \textbf{ then} \\
\quad\quad \textbf{return } i \\
\textbf{return } -1
\end{array}
$$

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- “Strings” are usually built from a small, pre-determined alphabet.

N O B O D Y _ N O T I C E D _ H I M

N O T

$$
\begin{array}{l}
\textbf{for } i \leftarrow 0 \text{ to } n-m \textbf{ do} \\
\quad j \leftarrow 0 \\
\quad \textbf{while } j < m \text{ and } p[j] = t[i+j] \textbf{ do} \\
\quad\quad j \leftarrow j+1 \\
\quad \textbf{if } j = m \textbf{ then} \\
\quad\quad \textbf{return } i \\
\textbf{return } -1
\end{array}
$$

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- "Strings" are usually built from a small, pre-determined alphabet.

$$
\begin{array}{l}
\textbf{for } i \leftarrow 0 \text{ to } n-m \textbf{ do} \\
\quad j \leftarrow 0 \\
\quad \textbf{while } j < m \text{ and } p[j] = t[i+j] \textbf{ do} \\
\quad\quad j \leftarrow j+1 \\
\quad \textbf{if } j = m \textbf{ then} \\
\quad\quad \textbf{return } i \\
\textbf{return } -1
\end{array}
$$

N O B O D Y _ N O T I C E D _ H I M

N O T

  N O T

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- "Strings" are usually built from a small, pre-determined alphabet.

$$
\begin{array}{l}
\textbf{for } i \leftarrow 0 \text{ to } n-m \textbf{ do} \\
\quad j \leftarrow 0 \\
\quad \textbf{while } j < m \text{ and } p[j] = t[i+j] \textbf{ do} \\
\quad\quad j \leftarrow j+1 \\
\quad \textbf{if } j = m \textbf{ then} \\
\quad\quad \textbf{return } i \\
\textbf{return } -1
\end{array}
$$

N O B O D Y _ N O T I C E D _ H I M

N O T

  N O T

    N O T

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- "Strings" are usually built from a small, pre-determined alphabet.

$$
\begin{array}{l}
\textbf{for } i \leftarrow 0 \text{ to } n-m \textbf{ do} \\
\quad j \leftarrow 0 \\
\quad \textbf{while } j < m \text{ and } p[j] = t[i+j] \textbf{ do} \\
\quad\quad j \leftarrow j+1 \\
\quad \textbf{if } j = m \textbf{ then} \\
\quad\quad \textbf{return } i \\
\textbf{return } -1
\end{array}
$$

```
N O B O D Y _ N O T I C E D _ H I M
N O T
  N O T
    N O T
      N O T
```

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- “Strings” are usually built from a small, pre-determined alphabet.

$$
\begin{array}{l}
\textbf{for } i \leftarrow 0 \text{ to } n-m \textbf{ do} \\
\quad j \leftarrow 0 \\
\quad \textbf{while } j < m \text{ and } p[j] = t[i+j] \textbf{ do} \\
\quad\quad j \leftarrow j+1 \\
\quad \textbf{if } j = m \textbf{ then} \\
\quad\quad \textbf{return } i \\
\textbf{return } -1
\end{array}
$$

```
N O B O D Y _ N O T I C E D _ H I M
N O T
  N O T
    N O T
      N O T
        N O T
```

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- "Strings" are usually built from a small, pre-determined alphabet.

$$
\begin{array}{l}
\textbf{for } i \leftarrow 0 \text{ to } n - m \textbf{ do} \\
\quad j \leftarrow 0 \\
\quad \textbf{while } j < m \text{ and } p[j] = t[i+j] \textbf{ do} \\
\quad\quad j \leftarrow j + 1 \\
\quad \textbf{if } j = m \textbf{ then} \\
\quad\quad \textbf{return } i \\
\textbf{return } -1
\end{array}
$$

N	O	B	O	D	Y	_	N	O	T	I	C	E	D	_	H	I	M
N	O	T															
	N	O	T														
		N	O	T													
			N	O	T												
				N	O	T											
					N	O	T										

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- "Strings" are usually built from a small, pre-determined alphabet.

for $i \leftarrow 0$ to $n - m$ **do**
    $j \leftarrow 0$
    while $j < m$ and $p[j] = t[i + j]$ **do**
        $j \leftarrow j + 1$
    if $j = m$ **then**
        return i
return -1

```
N O B O D Y _ N O T I C E D _ H I M
N O T
  N O T
    N O T
      N O T
        N O T
          N O T
            N O T
```

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- “Strings” are usually built from a small, pre-determined alphabet.

$$
\begin{array}{l}
\textbf{for } i \leftarrow 0 \text{ to } n-m \textbf{ do} \\
\quad j \leftarrow 0 \\
\quad \textbf{while } j < m \text{ and } p[j] = t[i+j] \textbf{ do} \\
\qquad j \leftarrow j+1 \\
\quad \textbf{if } j = m \textbf{ then} \\
\qquad \textbf{return } i \\
\textbf{return } -1
\end{array}
$$

```
N O B O D Y _ N O T I C E D _ H I M
N O T
  N O T
    N O T
      N O T
        N O T
          N O T
            N O T
              N O T
```

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- "Strings" are usually built from a small, pre-determined alphabet.

$$
\begin{array}{l}
\textbf{for } i \leftarrow 0 \text{ to } n - m \textbf{ do} \\
\quad j \leftarrow 0 \\
\quad \textbf{while } j < m \text{ and } p[j] = t[i+j] \textbf{ do} \\
\quad\quad j \leftarrow j + 1 \\
\quad \textbf{if } j = m \textbf{ then} \\
\quad\quad \textbf{return } i \\
\textbf{return } -1
\end{array}
$$

```
N O B O D Y _ N O T I C E D _ H I M
N O T
  N O T
    N O T
      N O T
        N O T
          N O T
            N O T
              N O T
```

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- "Strings" are usually built from a small, pre-determined alphabet.

$$
\begin{array}{l}
\textbf{for } i \leftarrow 0 \text{ to } n-m \textbf{ do} \\
\quad j \leftarrow 0 \\
\quad \textbf{while } j < m \text{ and } p[j] = t[i+j] \textbf{ do} \\
\quad\quad j \leftarrow j+1 \\
\quad \textbf{if } j = m \textbf{ then} \\
\quad\quad \textbf{return } i \\
\textbf{return } -1
\end{array}
$$

```
N O B O D Y _ N O T I C E D _ H I M
N O T
  N O T
    N O T
      N O T
        N O T
          N O T
            N O T
              N O T
```

String Matching Revisited

- We earlier discussed the brute-force approach to string search.

- “Strings” are usually built from a small, pre-determined alphabet.

for $i \leftarrow 0$ to $n - m$ **do**
  $j \leftarrow 0$
  while $j < m$ and $p[j] = t[i + j]$ **do**
    $j \leftarrow j + 1$
  if $j = m$ **then**
    return i
return -1

```
N O B O D Y _ N O T I C E D _ H I M
N O T
  N O T
    N O T
      N O T
        N O T
          N O T
            N O T
              N O T
```

- This seems very inefficient?

String Matching Revisited

- We earlier discussed the brute-force approach to string search.
- “Strings” are usually built from a small, pre-determined alphabet.
- Most of the better algorithms rely on some pre-processing of strings before the actual matching process starts.
- The pre-processing involves the construction of a small table (of predictable size).
- Levitin refers to this as “input enhancement”.

Horspool's String Search Algorithm

- Comparing from right to left in the pattern.

- Very good for random text strings

S T R I N G S E A R C H E X A M P
E X A M

- We can do better than just observing a mismatch here.

- Because the pattern has no occurrence of I, we might as well slide it 4 positions along.

- This is based only on knowing the pattern.

Horspool’s String Search Algorithm

```
S T R I N G S E A R C H E X A M P
E X A M
        E X A M
```

- Here we can slide the pattern 3 positions, because the last occurrence of E in the pattern is its first position.

```
S T R I N G S E A R C H E X A M P
E X A M
        E X A M
              E X A M
                        E X A M
                          E X A M
```

Horspool's String Search Algorithm

- What happens when we have longer partial matches?

S	E	A	R	C	H	I	N	G		
B	I	R	C	H						
	B	I	R	C	H					
						B	I	R	C	H

- The shift is determined by the last character in the pattern.

Char	Shift
A	5
B	4
C	1
⋮	⋮
H	5
I	3
⋮	⋮
R	2
S	5
⋮	⋮
Z	5

Horspool’s String Search Algorithm

- Building (calculating) the shift table is easy.
- Let a be the size of the alphabet.

$$
\begin{array}{l}
\textbf{function } \textsc{FindShifts}(P[0, \cdots, m-1], m) \\
\quad \textbf{for } i \leftarrow 0 \text{ to } a-1 \textbf{ do} \\
\quad\quad \mathit{Shift}[i] \leftarrow m \\
\quad \textbf{for } j \leftarrow 0 \text{ to } m-2 \textbf{ do} \\
\quad\quad \mathit{Shift}[P[j]] \leftarrow m-(j+1)
\end{array}
$$

Horspool’s String Search Algorithm

- Building (calculating) the shift table is easy.
- Let a be the size of the alphabet.

function FINDSHIFTS($P[0, \cdots, m-1], m$)
    for $i \leftarrow 0$ to $a - 1$ **do**
        $Shift[i] \leftarrow m$
    for $j \leftarrow 0$ to $m - 2$ **do**
        $Shift[P[j]] \leftarrow m - (j + 1)$

- Let’s consider a simple example with a small alphabet: the nucleotides from DNA: [A T G C] (so $a = 4$).
- The patterns is P = TAACG (so $m = 5$)
- The string is (n = 20)

T = GACCGCGTGAGATAACGTCA

Horspool's String Search Algorithm

- Building (calculating) the shift table is easy.
- Let a be the size of the alphabet.

function $\textsc{FindShifts}(P[0, \cdots, m-1], m)$
  for $i \leftarrow 0$ to $a-1$ **do**
    $Shift[i] \leftarrow m$
  for $j \leftarrow 0$ to $m-2$ **do**
    $Shift[P[j]] \leftarrow m-(j+1)$

P	[T,A,A,C,G]
m	5
a	4
i	
j	
$Shift$	[0,0,0,0]
$P[j]$	
$m-(j+1)$	

- Let's consider a simple example with a small alphabet: the nucleotides from DNA: [A T G C] (so $a = 4$).
- The patterns is P = TAACG (so $m = 5$)
- The string is (n = 20)

  T = GACCGCGTGAGATAACGTCA

Horspool's String Search Algorithm

- Building (calculating) the shift table is easy.
- Let a be the size of the alphabet.

function FINDSHIFTS($P[0, \cdots, m-1], m$)
  for $i \leftarrow 0$ to $a - 1$ **do**
    $Shift[i] \leftarrow m$
  for $j \leftarrow 0$ to $m - 2$ **do**
    $Shift[P[j]] \leftarrow m - (j + 1)$

P	[T,A,A,C,G]
m	5
a	4
i	0
j	
$Shift$	[5,0,0,0]
$P[j]$	
$m-(j+1)$	

- Let's consider a simple example with a small alphabet: the nucleotides from DNA: [A T G C] (so $a = 4$).
- The patterns is P = TAACG (so $m = 5$)
- The string is (n = 20)

  T = GACCGCGTGAGATAACGTCA

Horspool's String Search Algorithm

- Building (calculating) the shift table is easy.
- Let a be the size of the alphabet.

function FINDSHIFTS($P[0, \cdots, m-1], m$)
  for $i \leftarrow 0$ to $a-1$ **do**
    $Shift[i] \leftarrow m$
  for $j \leftarrow 0$ to $m-2$ **do**
    $Shift[P[j]] \leftarrow m-(j+1)$

P	[T,A,A,C,G]
m	5
a	4
i	1
j	
$Shift$	[5,5,0,0]
$P[j]$	
$m-(j+1)$	

- Let's consider a simple example with a small alphabet: the nucleotides from DNA: [A T G C] (so $a = 4$).
- The patterns is P = TAACG (so $m = 5$)
- The string is (n = 20)

  T = GACCGCGTGAGATAACGTCA

Horspool's String Search Algorithm

- Building (calculating) the shift table is easy.
- Let a be the size of the alphabet.

function FINDSHIFTS($P[0, \cdots, m-1], m$)
  for $i \leftarrow 0$ to $a-1$ **do**
    $Shift[i] \leftarrow m$
  for $j \leftarrow 0$ to $m-2$ **do**
    $Shift[P[j]] \leftarrow m-(j+1)$

P	[T,A,A,C,G]
m	5
a	4
i	2
j	
$Shift$	[5,5,5,0]
$P[j]$	
$m-(j+1)$	

- Let's consider a simple example with a small alphabet: the nucleotides from DNA: [A T G C] (so $a = 4$).
- The patterns is P = TAACG (so $m = 5$)
- The string is (n = 20)

  T = GACCGCGTGAGATAACGTCA

Horspool's String Search Algorithm

- Building (calculating) the shift table is easy.
- Let a be the size of the alphabet.

function FINDSHIFTS($P[0, \cdots, m-1], m$)
  for $i \leftarrow 0$ to $a - 1$ **do**
    $Shift[i] \leftarrow m$
  for $j \leftarrow 0$ to $m - 2$ **do**
    $Shift[P[j]] \leftarrow m - (j + 1)$

P	[T,A,A,C,G]
m	5
a	4
i	3
j	
$Shift$	[5,5,5,5]
$P[j]$	
$m-(j+1)$	

- Let's consider a simple example with a small alphabet: the nucleotides from DNA: [A T G C] (so $a = 4$).
- The patterns is P = TAACG (so $m = 5$)
- The string is (n = 20)

  T = GACCGCGTGAGATAACGTCA

Horspool's String Search Algorithm

- Building (calculating) the shift table is easy.
- Let a be the size of the alphabet.

function FINDSHIFTS($P[0, \cdots, m-1], m$)
  for $i \leftarrow 0$ to $a - 1$ **do**
    $Shift[i] \leftarrow m$
  for $j \leftarrow 0$ to $m - 2$ **do**
    $Shift[P[j]] \leftarrow m - (j + 1)$

P	[T,A,A,C,G]
m	5
a	4
i	3
j	0
$Shift$	[5,4,5,5]
$P[j]$	T
$m-(j+1)$	4

- Let's consider a simple example with a small alphabet: the nucleotides from DNA: [A T G C] (so $a = 4$).
- The patterns is P = TAACG (so $m = 5$)
- The string is (n = 20)

  T = GACCGCGTGAGATAACGTCA

Horspool's String Search Algorithm

- Building (calculating) the shift table is easy.
- Let a be the size of the alphabet.

function FINDSHIFTS($P[0, \cdots, m-1], m$)
  for $i \leftarrow 0$ to $a - 1$ **do**
    $Shift[i] \leftarrow m$
  for $j \leftarrow 0$ to $m - 2$ **do**
    $Shift[P[j]] \leftarrow m - (j + 1)$

P	[T,A,A,C,G]
m	5
a	4
i	3
j	1
$Shift$	[3,4,5,5]
$P[j]$	A
$m-(j+1)$	3

- Let's consider a simple example with a small alphabet: the nucleotides from DNA: [A T G C] (so $a = 4$).
- The patterns is P = TAACG (so $m = 5$)
- The string is (n = 20)

  T = GACCGCGTGAGATAACGTCA

Horspool's String Search Algorithm

- Building (calculating) the shift table is easy.
- Let a be the size of the alphabet.

function FINDSHIFTS($P[0, \cdots, m-1], m$)
  for $i \leftarrow 0$ to $a-1$ **do**
    $Shift[i] \leftarrow m$
  for $j \leftarrow 0$ to $m-2$ **do**
    $Shift[P[j]] \leftarrow m-(j+1)$

P	[T,A,A,C,G]
m	5
a	4
i	3
j	2
$Shift$	[2,4,5,5]
$P[j]$	A
$m-(j+1)$	2

- Let's consider a simple example with a small alphabet: the nucleotides from DNA: [A T G C] (so a = 4).
- The patterns is P = TAACG (so $m = 5$)
- The string is (n = 20)

  T = GACCGCGTGAGATAACGTCA

Horspool's String Search Algorithm

- Building (calculating) the shift table is easy.
- Let a be the size of the alphabet.

function FINDSHIFTS($P[0, \cdots, m-1], m$)
  for $i \leftarrow 0$ to $a - 1$ **do**
    $Shift[i] \leftarrow m$
  for $j \leftarrow 0$ to $m - 2$ **do**
    $Shift[P[j]] \leftarrow m - (j+1)$

P	[T,A,A,C,G]
m	5
a	4
i	3
j	3
$Shift$	[2,4,5,1]
$P[j]$	C
$m-(j+1)$	1

- Let's consider a simple example with a small alphabet: the nucleotides from DNA: [A T G C] (so $a = 4$).
- The patterns is P = TAACG (so $m = 5$)
- The string is (n = 20)

  T = GACCGCGTGAGATAACGTCA

Horspool's String Search Algorithm

function $\text{Horspool}(P[0, \cdots, m-1], m, T[0, \cdots, n-1], n)$
  $\text{FindShifts}(P, m)$
  $i \leftarrow m - 1$
  while $i < n$ **do**
    $k \leftarrow 0$
    while $k < m$ **and** $P[m-1-k] = T[i-k]$ **do**
      $k \leftarrow k + 1$
    if $k = m$ **then** ▷ We have a match
      return $i - m + 1$ ▷ Start of the match
    else
      $i \leftarrow i + \mathit{Shift}[T[i]]$ ▷ Slide the pattern along
  return -1

Horspool's String Search Algorithm

```
function HORSPOOL(P[0, ··· , m − 1], m, T[0, ··· , n − 1], n)
    FINDSHIFTS(P, m)
    i ← m − 1
    while i < n do
        k ← 0
        while k < m and P[m − 1 − k] = T[i − k] do
            k ← k + 1
        if k = m then
            return i − m + 1
        else
            i ← i + Shift[T[i]]
    return −1
```

P	$[T, A, A, C, G]$
m	5
n	20
$Shift$	
i	
k	
$P[m-1-k]$	
$T[i-k]$	
$i-m+1$	
$i+Shift[T[i]]$	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	G	A	C	C	G	C	G	T	G	A	G	A	T	A	A	C	G	T	C	A
P	T	A	A	C	G															

Horspool's String Search Algorithm

```
function HORSPOOL(P[0, ··· , m − 1], m, T[0, ··· , n − 1], n)
    FINDSHIFTS(P, m)
    i ← m − 1
    while i < n do
        k ← 0
        while k < m and P[m − 1 − k] = T[i − k] do
            k ← k + 1
        if k = m then
            return i − m + 1
        else
            i ← i + Shift[T[i]]
    return −1
```

P	$[T, A, A, C, G]$
m	5
n	20
$Shift$	[2,4,5,1]
i	
k	
$P[m-1-k]$	
$T[i-k]$	
$i-m+1$	
$i+Shift[T[i]]$	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	G	A	C	C	G	C	G	T	G	A	G	A	T	A	A	C	G	T	C	A
P	T	A	A	C	G															

Horspool's String Search Algorithm

function HORSPOOL($P[0, \cdots, m-1], m, T[0, \cdots, n-1], n$)
  FINDSHIFTS(P, m)
  $i \leftarrow m - 1$
  while $i < n$ **do**
    $k \leftarrow 0$
    while $k < m$ **and** $P[m-1-k] = T[i-k]$ **do**
      $k \leftarrow k + 1$
    if $k = m$ **then**
      return $i - m + 1$
    else
      $i \leftarrow i + Shift[T[i]]$
  return -1

P	$[T, A, A, C, G]$
m	5
n	20
$Shift$	[2,4,5,1]
i	4
k	0
$P[m-1-k]$	G
$T[i-k]$	G
$i - m + 1$	
$i + Shift[T[i]]$	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	G	A	C	C	G	C	G	T	G	A	G	A	T	A	A	C	G	T	C	A
P	T	A	A	C	G															

Horspool's String Search Algorithm

function HORSPOOL($P[0,\cdots,m-1], m, T[0,\cdots,n-1], n$)
    FINDSHIFTS(P, m)
    $i \leftarrow m-1$
    while $i < n$ **do**
        $k \leftarrow 0$
        while $k < m$ **and** $P[m-1-k] = T[i-k]$ **do**
            $k \leftarrow k+1$
        if $k = m$ **then**
            return $i - m + 1$
        else
            $i \leftarrow i + Shift[T[i]]$
    return -1

P	$[T,A,A,C,G]$
m	5
n	20
$Shift$	[2,4,5,1]
i	4
k	1
$P[m-1-k]$	C
$T[i-k]$	C
$i-m+1$	
$i+Shift[T[i]]$	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	G	A	C	C	G	C	G	T	G	A	G	A	T	A	A	C	G	T	C	A
P	T	A	A	C	G															

Horspool's String Search Algorithm

```
function HORSPOOL(P[0, ··· , m − 1], m, T[0, ··· , n − 1], n)
    FINDSHIFTS(P, m)
    i ← m − 1
    while i < n do
        k ← 0
        while k < m and P[m − 1 − k] = T[i − k] do
            k ← k + 1
        if k = m then
            return i − m + 1
        else
            i ← i + Shift[T[i]]
    return −1
```

Alphabet = [A T G C]

P	$[T, A, A, C, G]$
m	5
n	20
$Shift$	[2,4,5,1]
i	4
k	2
$P[m-1-k]$	A
$T[i-k]$	C
$i-m+1$	
$i+Shift[T[i]]$	4+5=9

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	G	A	C	C	G	C	G	T	G	A	G	A	T	A	A	C	G	T	C	A
P	T	A	A	C	G															

Horspool's String Search Algorithm

function HORSPOOL($P[0, \cdots, m-1], m, T[0, \cdots, n-1], n$)
    FINDSHIFTS(P, m)
    $i \leftarrow m - 1$
    while $i < n$ **do**
        $k \leftarrow 0$
        while $k < m$ **and** $P[m-1-k] = T[i-k]$ **do**
            $k \leftarrow k + 1$
        if $k = m$ **then**
            return $i - m + 1$
        else
            $i \leftarrow i + Shift[T[i]]$
    return -1

Alphabet = [A T G C]

P	$[T, A, A, C, G]$
m	5
n	20
$Shift$	[2,4,5,1]
i	9
k	0
$P[m-1-k]$	G
$T[i-k]$	A
$i-m+1$	
$i + Shift[T[i]]$	9+2=11

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	G	A	C	C	G	C	G	T	G	A	G	A	T	A	A	C	G	T	C	A
P						T	A	A	C	G										

Horspool's String Search Algorithm

function HORSPOOL($P[0, \cdots, m-1], m, T[0, \cdots, n-1], n$)
    FINDSHIFTS(P, m)
    $i \leftarrow m - 1$
    while $i < n$ **do**
        $k \leftarrow 0$
        while $k < m$ **and** $P[m-1-k] = T[i-k]$ **do**
            $k \leftarrow k + 1$
        if $k = m$ **then**
            return $i - m + 1$
        else
            $i \leftarrow i + Shift[T[i]]$
    return -1

P	$[T, A, A, C, G]$
m	5
n	20
$Shift$	[2,4,5,1]
i	11
k	0
$P[m-1-k]$	G
$T[i-k]$	A
$i - m + 1$	
$i + Shift[T[i]]$	11+2=13

Alphabet = [A T G C]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	G	A	C	C	G	C	G	T	G	A	G	A	T	A	A	C	G	T	C	A
P								T	A	A	C	G								

Horspool's String Search Algorithm

function HORSPOOL($P[0, \cdots, m-1], m, T[0, \cdots, n-1], n$)
  FINDSHIFTS(P, m)
  $i \leftarrow m - 1$
  while $i < n$ **do**
    $k \leftarrow 0$
    while $k < m$ **and** $P[m-1-k] = T[i-k]$ **do**
      $k \leftarrow k + 1$
    if $k = m$ **then**
      return $i - m + 1$
    else
      $i \leftarrow i + Shift[T[i]]$
  return -1

Alphabet = [A T G C]

P	$[T, A, A, C, G]$
m	5
n	20
$Shift$	[2,4,5,1]
i	13
k	0
$P[m-1-k]$	G
$T[i-k]$	A
$i-m+1$	
$i + Shift[T[i]]$	13+2=15

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	G	A	C	C	G	C	G	T	G	A	G	A	T	A	A	C	G	T	C	A
P										T	A	A	C	G						

Horspool's String Search Algorithm

```
function HORSPOOL(P[0, ··· , m − 1], m, T[0, ··· , n − 1], n)
    FINDSHIFTS(P, m)
    i ← m − 1
    while i < n do
        k ← 0
        while k < m and P[m − 1 − k] = T[i − k] do
            k ← k + 1
        if k = m then
            return i − m + 1
        else
            i ← i + Shift[T[i]]
    return −1
```

Alphabet = [A T G C]

P	$[T, A, A, C, G]$
m	5
n	20
$Shift$	[2,4,5,1]
i	15
k	0
$P[m-1-k]$	G
$T[i-k]$	C
$i-m+1$	
$i+Shift[T[i]]$	15+1=16

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	G	A	C	C	G	C	G	T	G	A	G	A	T	A	A	C	G	T	C	A
P												T	A	A	C	G				

Horspool's String Search Algorithm

```
function HORSPOOL(P[0, ··· , m − 1], m, T[0, ··· , n − 1], n)
    FINDSHIFTS(P, m)
    i ← m − 1
    while i < n do
        k ← 0
        while k < m and P[m − 1 − k] = T[i − k] do
            k ← k + 1
        if k = m then
            return i − m + 1
        else
            i ← i + Shift[T[i]]
    return −1
```

Alphabet = [A T G C]

P	$[T, A, A, C, G]$
m	5
n	20
$Shift$	[2,4,5,1]
i	16
k	0
$P[m-1-k]$	G
$T[i-k]$	G
$i-m+1$	
$i+Shift[T[i]]$	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	G	A	C	C	G	C	G	T	G	A	G	A	T	A	A	C	G	T	C	A
P													T	A	A	C	G			

Horspool's String Search Algorithm

function HORSPOOL($P[0, \cdots, m-1], m, T[0, \cdots, n-1], n$)
    FINDSHIFTS(P, m)
    $i \leftarrow m - 1$
    while $i < n$ **do**
        $k \leftarrow 0$
        while $k < m$ **and** $P[m-1-k] = T[i-k]$ **do**
            $k \leftarrow k + 1$
        if $k = m$ **then**
            return $i - m + 1$
        else
            $i \leftarrow i + Shift[T[i]]$
    return -1

Alphabet = [A T G C]

P	$[T, A, A, C, G]$
m	5
n	20
$Shift$	[2,4,5,1]
i	16
k	1
$P[m-1-k]$	C
$T[i-k]$	C
$i - m + 1$	
$i + Shift[T[i]]$	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	G	A	C	C	G	C	G	T	G	A	G	A	T	A	A	C	G	T	C	A
P													T	A	A	C	G			

Horspool's String Search Algorithm

function HORSPOOL($P[0, \cdots, m-1], m, T[0, \cdots, n-1], n$)
    FINDSHIFTS(P, m)
    $i \leftarrow m - 1$
    while $i < n$ **do**
        $k \leftarrow 0$
        while $k < m$ **and** $P[m-1-k] = T[i-k]$ **do**
            $k \leftarrow k + 1$
        if $k = m$ **then**
            return $i - m + 1$
        else
            $i \leftarrow i + Shift[T[i]]$
    return -1

Alphabet = [A T G C]

P	$[T, A, A, C, G]$
m	5
n	20
$Shift$	[2,4,5,1]
i	16
k	2
$P[m-1-k]$	A
$T[i-k]$	A
$i-m+1$	
$i + Shift[T[i]]$	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	G	A	C	C	G	C	G	T	G	A	G	A	T	A	A	C	G	T	C	A
P													T	A	A	C	G			

Horspool's String Search Algorithm

THE UNIVERSITY OF MELBOURNE

function HORSPOOL($P[0, \cdots, m-1], m, T[0, \cdots, n-1], n$)
    FINDSHIFTS(P, m)
    $i \leftarrow m - 1$
    while $i < n$ **do**
        $k \leftarrow 0$
        while $k < m$ **and** $P[m-1-k] = T[i-k]$ **do**
            $k \leftarrow k + 1$
        if $k = m$ **then**
            return $i - m + 1$
        else
            $i \leftarrow i + Shift[T[i]]$
    return -1

Alphabet = [A T G C]

P	$[T, A, A, C, G]$
m	5
n	20
$Shift$	[2,4,5,1]
i	16
k	3
$P[m-1-k]$	A
$T[i-k]$	A
$i-m+1$	
$i+Shift[T[i]]$	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	G	A	C	C	G	C	G	T	G	A	G	A	T	A	A	C	G	T	C	A
P													T	A	A	C	G			

Horspool's String Search Algorithm

```
function HORSPOOL(P[0, ··· , m − 1], m, T[0, ··· , n − 1], n)
    FINDSHIFTS(P, m)
    i ← m − 1
    while i < n do
        k ← 0
        while k < m and P[m − 1 − k] = T[i − k] do
            k ← k + 1
        if k = m then
            return i − m + 1
        else
            i ← i + Shift[T[i]]
    return −1
```

Alphabet = [A T G C]

P	$[T, A, A, C, G]$
m	5
n	20
$Shift$	[2,4,5,1]
i	16
k	4
$P[m-1-k]$	T
$T[i-k]$	T
$i-m+1$	12
$i+Shift[T[i]]$	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	G	A	C	C	G	C	G	T	G	A	G	A	T	A	A	C	G	T	C	A
P													T	A	A	C	G			

Horspool's String Search Algorithm

function $\textsc{Horspool}(P[0, \cdots, m-1], m, T[0, \cdots, n-1], n)$
    $\textsc{FindShifts}(P, m)$
    $i \leftarrow m - 1$
    while $i < n$ **do**
        $k \leftarrow 0$
        while $k < m$ **and** $P[m-1-k] = T[i-k]$ **do**
            $k \leftarrow k + 1$
        if $k = m$ **then**
            return $i - m + 1$
        else
            $i \leftarrow i + Shift[T[i]]$
    return -1

Pattern: BARBER
Text: JIM_SAW_ME_IN_A_BARBERSHOP

Horspool's String Search Algorithm

function $\textsc{Horspool}(P[0, \cdots, m-1], m, T[0, \cdots, n-1], n)$
    $\textsc{FindShifts}(P, m)$
    $i \leftarrow m - 1$
    while $i < n$ **do**
        $k \leftarrow 0$
        while $k < m$ **and** $P[m-1-k] = T[i-k]$ **do**
            $k \leftarrow k + 1$
        if $k = m$ **then**
            return $i - m + 1$
        else
            $i \leftarrow i + \mathit{Shift}[T[i]]$
    return -1

Pattern: BARBER
Text: JIM_SAW_ME_IN_A_BARBERSHOP

Character	A	B	C	D	E	F	...	R	...	Z	_
Shift	4	2	6	6	1	6	6	3	6	6	6

Horspool's String Search Algorithm

function $\textsc{Horspool}(P[0, \cdots, m-1], m, T[0, \cdots, n-1], n)$
    $\textsc{FindShifts}(P, m)$
    $i \leftarrow m - 1$
    while $i < n$ **do**
        $k \leftarrow 0$
        while $k < m$ **and** $P[m-1-k] = T[i-k]$ **do**
            $k \leftarrow k + 1$
        if $k = m$ **then**
            return $i - m + 1$
        else
            $i \leftarrow i + \mathit{Shift}[T[i]]$
    return -1

```
JIM_SAW_ME_IN_A_BARBERSHOP
BARBER
    BARBER
     BARBER
           BARBER
             BARBER
                BARBER
```

Pattern: BARBER
Text: JIM_SAW_ME_IN_A_BARBERSHOP

Character	A	B	C	D	E	F	...	R	...	Z	_
Shift	4	2	6	6	1	6	6	3	6	6	6

Horspool's String Search Algorithm

- We can also consider posting a sentinel: Append the pattern P to the end of the text T so that a match is guaranteed.

```
function HORSPOOL(P[0..m − 1], T[0..n − 1])
    FINDSHIFTS(P)
    i ← m − 1
    while True do
        k ← 0
        while k < m and P[m − 1 − k] = T[i − k] do
            k ← k + 1
        if k = m then
            if i > n then
                return −1
            else
                return i − m + 1
        i ← i + Shift[T[i]]
```

Horspool's String Search Algorithm

- Unfortunately the worst-case behaviour of Horspool's algorithm is still $O(m \times n)$, like the brute-force methods.

- However, in practice, for example, when used on English texts, it is linear, and fast.

Other Important String Search Algorithms

- Horspool's algorithm was inspired by the famous Boyer-Moore algorithm (BM), also covered in Levitin's book. The BM algorithm is very similar, but has a more sophisticated shifting strategy, which makes it $O(m + n)$.

- Another famour string search algorithm is the Knuth-Morris-Pratt algorithm (KMP).

- KMP is very good when the alphabet is small, say, we need to search through very long bit strings.

- Also, we shall soon meet the Rabin-Karp algorithm (RK), albeit briefly.

Coming Up Next

- We look at the hugely important technique of hashing (Levitin 7.3) , a standard way of implementing a "dictionary".

- Hashing is arguably the best example of how to gain speed by using additional space to great effect.