### extract_min()

1. \( ret \leftarrow A[1] \)
2. \( A[1] \leftarrow A[s] \)
3. \( p[A[1]] \leftarrow 1 \)
4. \( s \leftarrow s - 1 \)
5. **if** \( s \geq 1 \) **then**
6. \( \text{heapify-down}(1) \)
7. **return** \( ret \)

### heapify-down(i)

1. **while** \( 2i \leq s \) **do**
2. **if** \( 2i = s \) **or**
   
   \( key[A[2i]] \leq key[A[2i + 1]] \) **then**
3. \( j \leftarrow 2i \)
4. **else**
5. \( j \leftarrow 2i + 1 \)
6. **if** \( key[A[j]] < key[A[i]] \) **then**
7. \( \text{swap} \ A[i] \text{ and } A[j] \)
8. \( p[A[i]] \leftarrow i, p[A[j]] \leftarrow j \)
9. \( i \leftarrow j \)
10. **else** break

### decrease_key(v, key_value)

1. \( key[v] \leftarrow key\_value \)
2. \( \text{heapify-up}(p[v]) \)
- Running time of heapify\_up and heapify\_down: $O(\lg n)$
- Running time of heapify\_up and heapify\_down: $O(\lg n)$
- Running time of insert, exact\_min and decrease\_key: $O(\lg n)$
- Running time of `heapify_up` and `heapify_down`: $O(lg \, n)$
- Running time of `insert`, `exact_min` and `decrease_key`: $O(lg \, n)$

<table>
<thead>
<tr>
<th>data structures</th>
<th>insert</th>
<th>extract_min</th>
<th>decrease_key</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>sorted array</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>heap</td>
<td>$O(lg , n)$</td>
<td>$O(lg , n)$</td>
<td>$O(lg , n)$</td>
</tr>
</tbody>
</table>
Two Definitions Needed to Prove that the Procedures Maintain Heap Property

**Def.** We say that $H$ is almost a heap except that $key[A[i]]$ is too small if we can increase $key[A[i]]$ to make $H$ a heap.

**Def.** We say that $H$ is almost a heap except that $key[A[i]]$ is too big if we can decrease $key[A[i]]$ to make $H$ a heap.
Outline

1. Toy Example: Box Packing
2. Interval Scheduling
   - Interval Partitioning
3. Offline Caching
   - Heap: Concrete Data Structure for Priority Queue
4. Data Compression and Huffman Code
5. Summary
6. Exercise Problems
8 letters \(a, b, c, d, e, f, g, h\) in a language

need to encode a message using bits

idea: use 3 bits per letter

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
</tr>
</tbody>
</table>

\[ \text{deacf} \rightarrow 011100000010101110 \]

Q: Can we have a better encoding scheme?

Seems unlikely: must use 3 bits per letter

Q: What if some letters appear more frequently than the others?
**Q:** If some letters appear more frequently than the others, can we have a better encoding scheme?

**A:** Using **variable-length encoding scheme** might be more efficient.

**Idea**

- using fewer bits for letters that are more frequently used, and more bits for letters that are less frequently used.
Q: What is the issue with the following encoding scheme?

- $a: 0$
- $b: 1$
- $c: 00$

A: Cannot guarantee a unique decoding. For example, $00$ can be decoded to $aa$ or $c$. Solution: Use prefix codes to guarantee a unique decoding.
Q: What is the issue with the following encoding scheme?

\[ a: 0 \quad b: 1 \quad c: 00 \]

A: Can not guarantee a unique decoding. For example, 00 can be decoded to \( aa \) or \( c \).
Q: What is the issue with the following encoding scheme?

- $a$: 0
- $b$: 1
- $c$: 00

A: Can not guarantee a unique decoding. For example, 00 can be decoded to $aa$ or $c$.

Solution

Use prefix codes to guarantee a unique decoding.
**Def.** A prefix code for a set $S$ of letters is a function $\gamma : S \rightarrow \{0, 1\}^*$ such that for two distinct $x, y \in S$, $\gamma(x)$ is not a prefix of $\gamma(y)$. 
Prefix Codes

**Def.** A prefix code for a set $S$ of letters is a function $\gamma : S \rightarrow \{0, 1\}^*$ such that for two distinct $x, y \in S$, $\gamma(x)$ is not a prefix of $\gamma(y)$.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>001</td>
<td>0000</td>
<td>0001</td>
<td>100</td>
</tr>
<tr>
<td>e</td>
<td>11</td>
<td>1010</td>
<td>1011</td>
<td>01</td>
</tr>
</tbody>
</table>

![Tree Diagram]

In the tree diagram, each letter is associated with a path from the root to the leaf node, and the code is determined by the path's binary representation.
Prefix Codes Guarantee Unique Decoding

Reason: there is only one way to cut the first code.
Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>0000</td>
<td>0001</td>
<td>100</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
</tr>
<tr>
<td>11</td>
<td>1010</td>
<td>1011</td>
<td>01</td>
</tr>
</tbody>
</table>
Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>001</td>
<td>0000</td>
<td>0001</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>1010</td>
<td>1011</td>
<td>01</td>
</tr>
</tbody>
</table>

- 000100111000000001011110100001001
Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>001</td>
<td>0000</td>
<td>0001</td>
<td>100</td>
</tr>
<tr>
<td>e</td>
<td>11</td>
<td>1010</td>
<td>1011</td>
<td>01</td>
</tr>
</tbody>
</table>

- 0001/0011000000001011110100001001
- c
Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>0000</td>
<td>0001</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1010</td>
<td>1011</td>
<td>01</td>
<td></td>
</tr>
</tbody>
</table>

- 0001/001/100000001011110100001001
- ca
Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>001</td>
<td>0000</td>
<td>0001</td>
<td>100</td>
</tr>
<tr>
<td>e</td>
<td>11</td>
<td>1010</td>
<td>1011</td>
<td>01</td>
</tr>
</tbody>
</table>

- 0001/001/100/000001011110100001001
- cad
Prefix Codes Guarantee Unique Decoding

- **Reason:** there is only one way to cut the first code.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>001</td>
<td>0000</td>
<td>0001</td>
<td>100</td>
</tr>
<tr>
<td>e</td>
<td>11</td>
<td>1010</td>
<td>1011</td>
<td>01</td>
</tr>
</tbody>
</table>

- 0001/001/100/0000/01011110100001001
- cadb
Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>0000</td>
<td>0001</td>
<td>100</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
</tr>
<tr>
<td>11</td>
<td>1010</td>
<td>1011</td>
<td>01</td>
</tr>
</tbody>
</table>

- 0001/001/100/0000/01/011110100001001
- cadbh
Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>001</td>
<td>0000</td>
<td>0001</td>
<td>100</td>
</tr>
<tr>
<td>e</td>
<td>11</td>
<td>1010</td>
<td>1011</td>
<td>01</td>
</tr>
</tbody>
</table>

- 0001/001/100/0000/01/01/1110100001001
- cadbh
Prefix Codes Guarantee Unique Decoding

Reason: there is only one way to cut the first code.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>001</td>
<td>0000</td>
<td>0001</td>
<td>100</td>
</tr>
<tr>
<td>e</td>
<td>11</td>
<td>1010</td>
<td>1011</td>
<td>01</td>
</tr>
</tbody>
</table>

0001/001/100/0000/01/01/11/10100001001

cadbhhe
Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>001</td>
<td>0000</td>
<td>0001</td>
<td>100</td>
</tr>
<tr>
<td>e</td>
<td>11</td>
<td>1010</td>
<td>1011</td>
<td>01</td>
</tr>
</tbody>
</table>

- 0001/001/100/0000/01/01/11/1010/0001001
- cadbhhef
Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>001</td>
<td>0000</td>
<td>0001</td>
<td>100</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
</tr>
<tr>
<td>11</td>
<td>1010</td>
<td>1011</td>
<td>01</td>
</tr>
</tbody>
</table>

- 0001/001/100/0000/01/01/11/1010/0001/001
- cadbhhefc
Prefix Codes Guarantee Unique Decoding

- Reason: there is only one way to cut the first code.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>001</td>
<td>0000</td>
<td>0001</td>
<td>100</td>
</tr>
<tr>
<td>e</td>
<td>11</td>
<td>1010</td>
<td>1011</td>
<td>01</td>
</tr>
</tbody>
</table>

- 0001/001/100/0000/01/01/11/1010/0001/001/
- cadbhhefca
Properties of Encoding Tree

Rooted binary tree
Left edges labelled 0 and right edges labelled 1
Edges a, e, f, h correspond to a code for some letter
If coding scheme is not wasteful: a non-leaf has exactly two children

Best Prefix Codes
Input: frequencies of letters in a message
Output: prefix coding scheme with the shortest encoding for the message
Properties of Encoding Tree

- Rooted binary tree

Best Prefix Codes

Input:
- frequencies of letters in a message

Output:
- prefix coding scheme with the shortest encoding for the message
Properties of Encoding Tree

- Rooted binary tree
- Left edges labelled 0 and right edges labelled 1

Best Prefix Codes

Input:
frequencies of letters in a message

Output:
prefix coding scheme with the shortest encoding for the message
Properties of Encoding Tree

- Rooted binary tree
- Left edges labelled 0 and right edges labelled 1
- A leaf corresponds to a code for some letter
Properties of Encoding Tree

- Rooted binary tree
- Left edges labelled 0 and right edges labelled 1
- A leaf corresponds to a code for some letter
- If coding scheme is not wasteful: a non-leaf has exactly two children
Properties of Encoding Tree

- Rooted binary tree
- Left edges labelled 0 and right edges labelled 1
- A leaf corresponds to a code for some letter
- If coding scheme is not wasteful: a non-leaf has exactly two children

Best Prefix Codes

**Input:** frequencies of letters in a message

**Output:** prefix coding scheme with the shortest encoding for the message
### Example

<table>
<thead>
<tr>
<th>letters</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequencies</td>
<td>18</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

- **Scheme 1 Length**: 2, 3, 3, 2, 2, total = 89
- **Scheme 2 Length**: 1, 3, 3, 3, 3, total = 87
- **Scheme 3 Length**: 1, 4, 4, 3, 2, total = 84
### Example

<table>
<thead>
<tr>
<th>letters</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequencies</td>
<td>18</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>scheme 1 length</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>scheme 2 length</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>scheme 3 length</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

**Scheme 1**
```
a  
  
  b  c  
  
  a  d  e 
```

**Scheme 2**
```
a  
  
  b  c  
  
  a  d  e 
```

**Scheme 3**
```
a  
  
  e  
  
  d  
  
  b  c  
```
Example Input: \((a: 18, b: 3, c: 4, d: 6, e: 10)\)
Example Input: \( (a: 18, b: 3, c: 4, d: 6, e: 10) \)

Q: What types of decisions should we make?
Example Input: \((a: 18, b: 3, c: 4, d: 6, e: 10)\)

Q: What types of decisions should we make?

- Can we directly give a code for some letter?
Example Input: \((a: 18, b: 3, c: 4, d: 6, e: 10)\)

Q: What types of decisions should we make?

- Can we directly give a code for some letter?
- Hard to design a strategy; residual problem is complicated.
Example Input: \((a: 18, b: 3, c: 4, d: 6, e: 10)\)

**Q:** What types of decisions should we make?

- Can we directly give a code for some letter?
- Hard to design a strategy; residual problem is complicated.
- Can we partition the letters into left and right sub-trees?
Example Input: \((a: 18, b: 3, c: 4, d: 6, e: 10)\)

Q: What types of decisions should we make?

- Can we directly give a code for some letter?
- Hard to design a strategy; residual problem is complicated.
- Can we partition the letters into left and right sub-trees?
- Not clear how to design the greedy algorithm
Example Input: \((a: 18, b: 3, c: 4, d: 6, e: 10)\)

**Q:** What types of decisions should we make?

- Can we directly give a code for some letter?
- Hard to design a strategy; residual problem is complicated.
- Can we partition the letters into left and right sub-trees?
- Not clear how to design the greedy algorithm

**A:** We can choose two letters and make them brothers in the tree.