COMP90038
Algorithms and Complexity

Lecture 21: Huffman Encoding for Data Compression

(with thanks to Harald Søndergaard & Michael Kirley)

Andres Munoz-Acosta

munoz.m@unimelb.edu.au

Peter Hall Building G.83

Recap

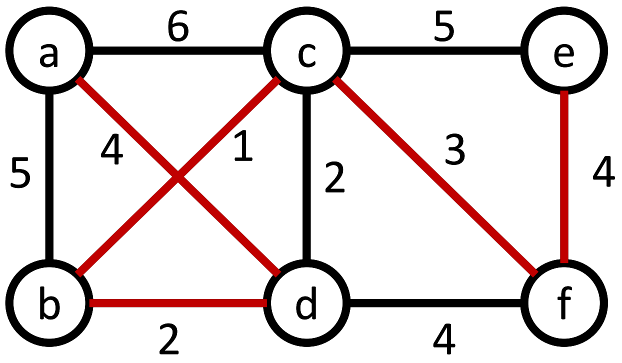
- We discussed **greedy algorithms**:
  - A problem solving strategy that takes the **locally best** choice among all feasible ones. Such choice is **irrevocable**.
  - Usually, **locally best** choices do not yield **global best** results.
  - In some exceptions a greedy algorithm is **correct and fast**.
  - Also, a greedy algorithm can provide good **approximations**.

- We applied this idea to two graph problems :
  - Prim's algorithm for finding **minimum spanning trees**
  - Dijkstra's algorithm for **single-source shortest path**

What is a Minimum Spanning Tree?

- A **minimum spanning tree** of a weighted graph ⟨*V*,*E*⟩ is a tree ⟨*V*,*E'*⟩ where *E'* is a subset of *E*, such that the connections have the lowest cost

- We use Prim's algorithm to find the minimum spanning tree.
  - It constructs a sequence of subtrees *T*, by **adding to the latest tree the closest node not currently on it**.



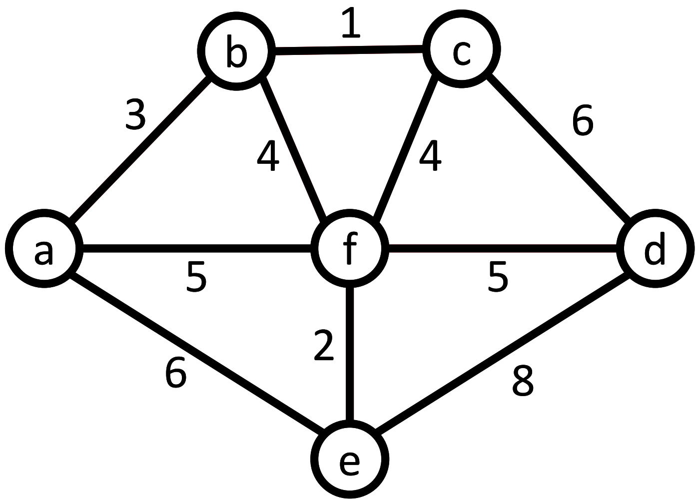
Prim's Algorithm

- We examined the complete algorithm, that uses priority queues:

function $\textsc{Prim}(\langle V, E \rangle)$
 for each $v \in V$ **do**
  $cost[v] \leftarrow \infty$
  $prev[v] \leftarrow nil$
 pick initial node v_0
 $cost[v_0] \leftarrow 0$
 $Q \leftarrow \textsc{InitPriorityQueue}(V)$ $\triangleright$ priorities are cost values
 while Q is non-empty **do**
  $u \leftarrow \textsc{EjectMin}(Q)$
  for each $(u, w) \in E$ **do**
   if $weight(u, w) < cost[w]$ **then**
    $\boxed{cost[w] \leftarrow weight(u, w)}$
    $prev[w] \leftarrow u$
    $\textsc{Update}(Q, w, cost[w])$ $\triangleright$ rearranges priority queue

Another example

- Let's work with the following graph:



Tree T		a	b	c	d	e	f
	cost	∞	∞	∞	∞	∞	∞
	prev	nil	nil	nil	nil	nil	nil

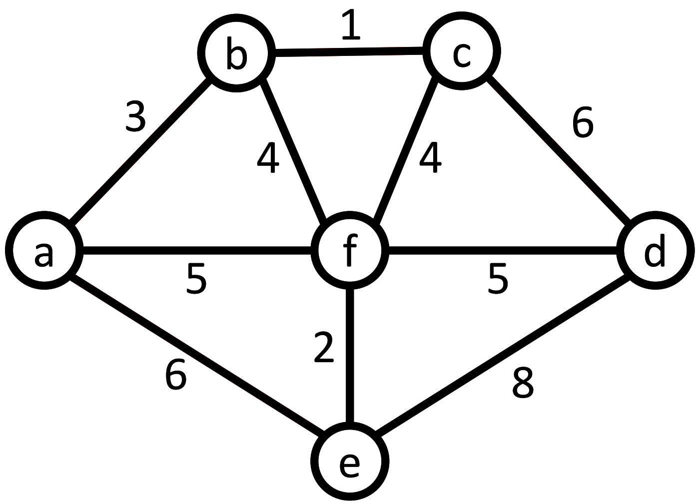
Dijkstra's Algorithm

- **Dijkstra's algorithm** finds all shortest paths **from a fixed start node**. Its complexity is the same as that of Prim's algorithm.

```
function DIJKSTRA(⟨V, E⟩, v_0)
    for each v ∈ V do
        dist[v] ← ∞
        prev[v] ← nil
    dist[v_0] ← 0
    Q ← INITPRIORITYQUEUE(V)                    ▷ priorities are distances
    while Q is non-empty do
        u ← EJECTMIN(Q)
        for each (u, w) ∈ E do
            if dist[u] + weight(u, w) < dist[w] then
                dist[w] ← dist[u] + weight(u, w)
                prev[w] ← u
                UPDATE(Q, w, dist[w])           ▷ rearranges priority queue
```

Another example

- Let’s work with this graph again:



Tree T		a	b	c	d	e	f
	cost	∞	∞	∞	∞	∞	∞
	prev	nil	nil	nil	nil	nil	nil

Data compression

- From an information-theoretic point of view, most computer files contain much redundancy.

- Compression is used to store files in less space.
  - For text files, savings up to 50 are common.
  - For binary files, savings up to 90 are common.

- Savings in space mean savings in time for file transmission.

Run-Length Encoding

- For a text with long runs of **repeated characters**, we could compress by counting the runs. For example:

AAAABBBAABBBBBCCCCCCCCDABCBAAABBBBCCCD

- can then be encoded as:

4A3BAA5B8CDABCB3A4B3CD

- This is not useful for normal text. However, for **binary files** it can be very effective.

Run-Length Encoding

```
000000000000000000000000000011111111111111000000000
000000000000000000000000001111111111111111110000000
000000000000000000000001111111111111111111111110000
000000000000000000000011111111111111111111111111000
000000000000000000001111111111111111111111111111110
000000000000000000011111110000000000000000001111111
000000000000000000011111000000000000000000000011111
000000000000000000011100000000000000000000000000111
000000000000000000011100000000000000000000000000111
000000000000000000011100000000000000000000000000111
000000000000000000011100000000000000000000000000111
000000000000000000001111000000000000000000000001110
000000000000000000000011100000000000000000000111000
011111111111111111111111111111111111111111111111111
011111111111111111111111111111111111111111111111111
011111111111111111111111111111111111111111111111111
011111111111111111111111111111111111111111111111111
011000000000000000000000000000000000000000000000011
```

Variable-Length Encoding

- Fixed-length encoding uses a static number of symbols (bits) to represent a character.
  - For example, the ASCII code uses 8 bits per character.

- Variable-Length encoding assigns shorter codes to common characters.
  - In English, the most common character is **E**, hence, we could assign **0** to it.
  - However, no other character code can start with **0**.

- That is, no character's code should be a prefix of some other character's code (unless we somehow put separators between characters, which would take up space).

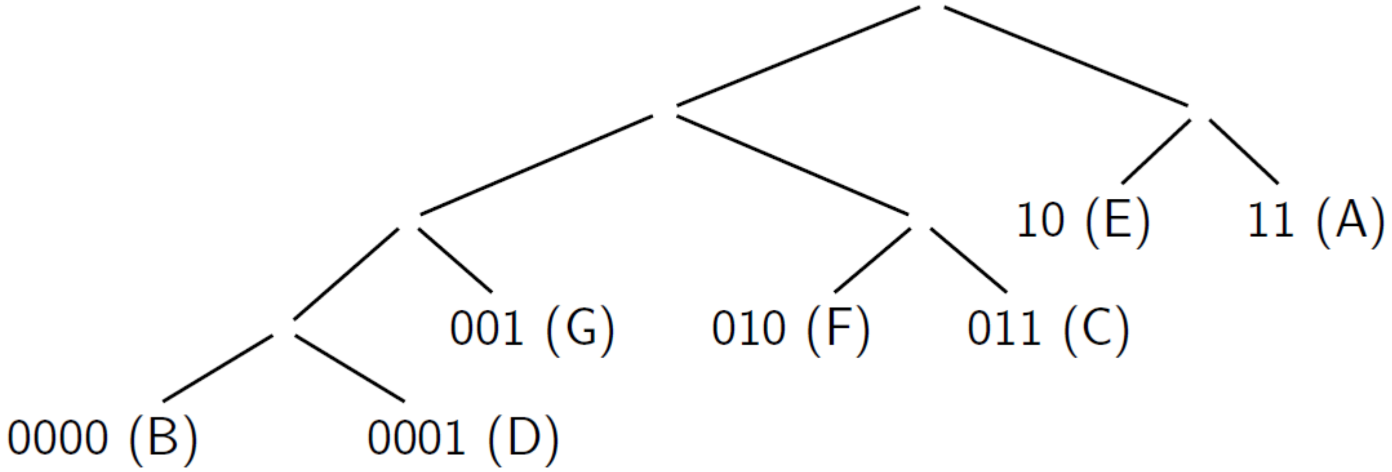
Variable-Length Encoding

- Suppose our alphabet is {A,B,C,D,E,F,G}
- We analyzed a text and found the following number of occurrences
- The last column shows some sensible codes that we may use for each symbol
  - Symbols with higher occurrence have shorter codes

SYMBOL	OCCURRENCE	CODE
A	28	11
B	4	0000
C	14	011
D	5	0001
E	27	10
F	12	010
G	10	001

Tries for Variable-Length Encoding

- A **trie** is a binary tree used on search applications
- To search for a key we look at individual **bits** of a key and descend to the **left** whenever a bit is **zero** and to the right whenever it is **one**
- Using a trie to determine codes means that no code will be the prefix of another



Encoding messages

- To encode a message, we just need to concatenate the codes. For example:

F	A	C	E
010	11	011	10

B	A	G	G	E	D
0000	11	001	001	10	0001

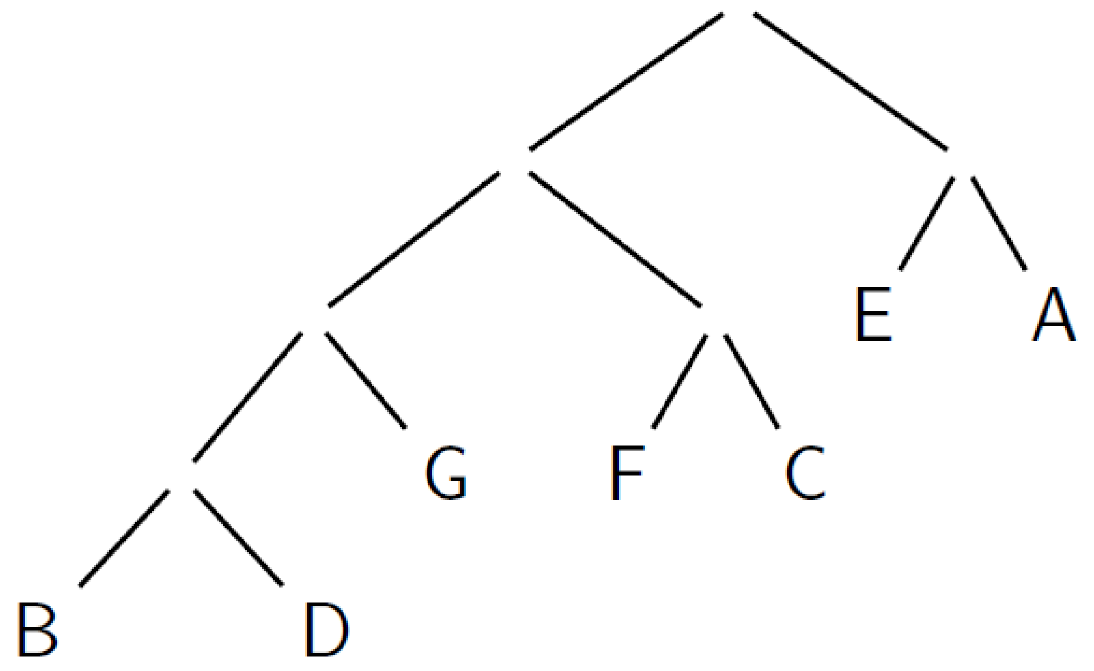
- If we were to assign three bits per character, FACE would use 12 bits instead of 10. For BAGGED there is no space savings

SYMBOL	CODE
A	11
B	0000
C	011
D	0001
E	10
F	010
G	001

Decoding messages

- Try to decode **00011001111010** and **000011000100110** using the trie
  - Starting from the root, print each symbol found as a leaf
  - Repeat until the string is completed

- Remember the rules: Left branch is 0, right branch is 1



Huffman Encoding: Choosing the Codes

- Sometimes (for example for common English text) we may know the frequencies of letters fairly well.

- If we don't know about frequencies then we can still count all characters in the given text as a first step.

- But how do we assign codes to the characters once we know their frequencies?
  - By repeatedly selecting the two smallest weights and fusing them.

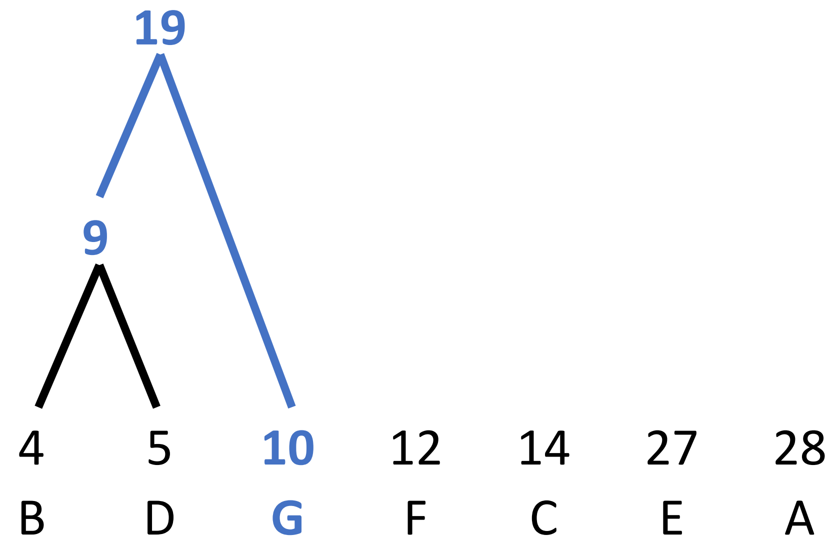
- This is **Huffman's algorithm** – another example of a **greedy method**.
  - The resulting tree is a **Huffman tree**.

Huffman Trees (example)

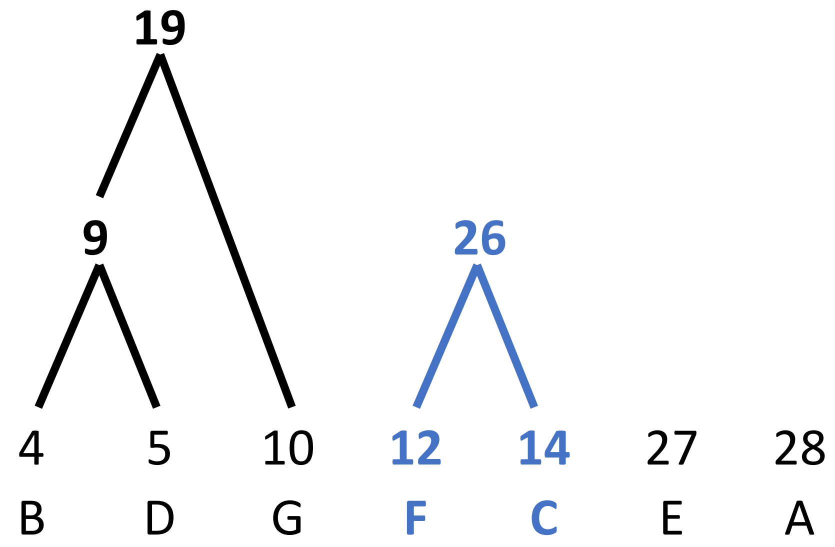
4	5	10	12	14	27	28
B	D	G	F	C	E	A

Huffman Trees (example)

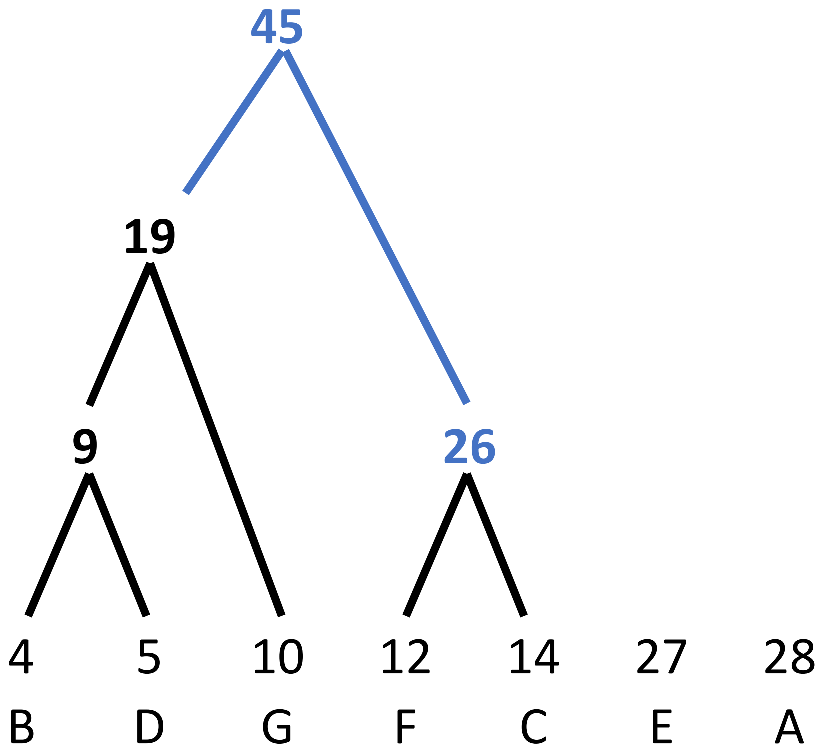
Huffman Trees (example)



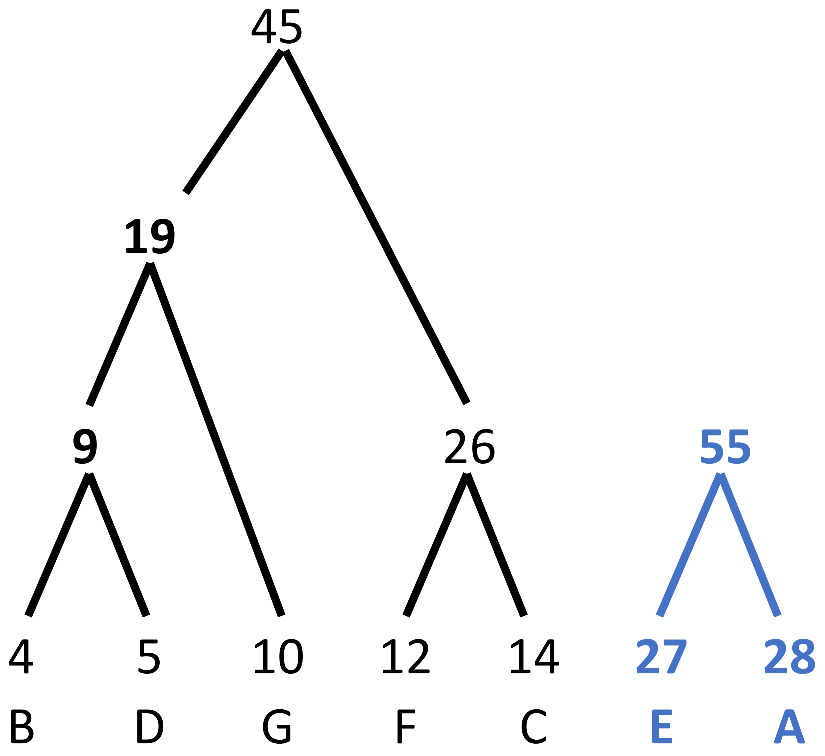
Huffman Trees (example)



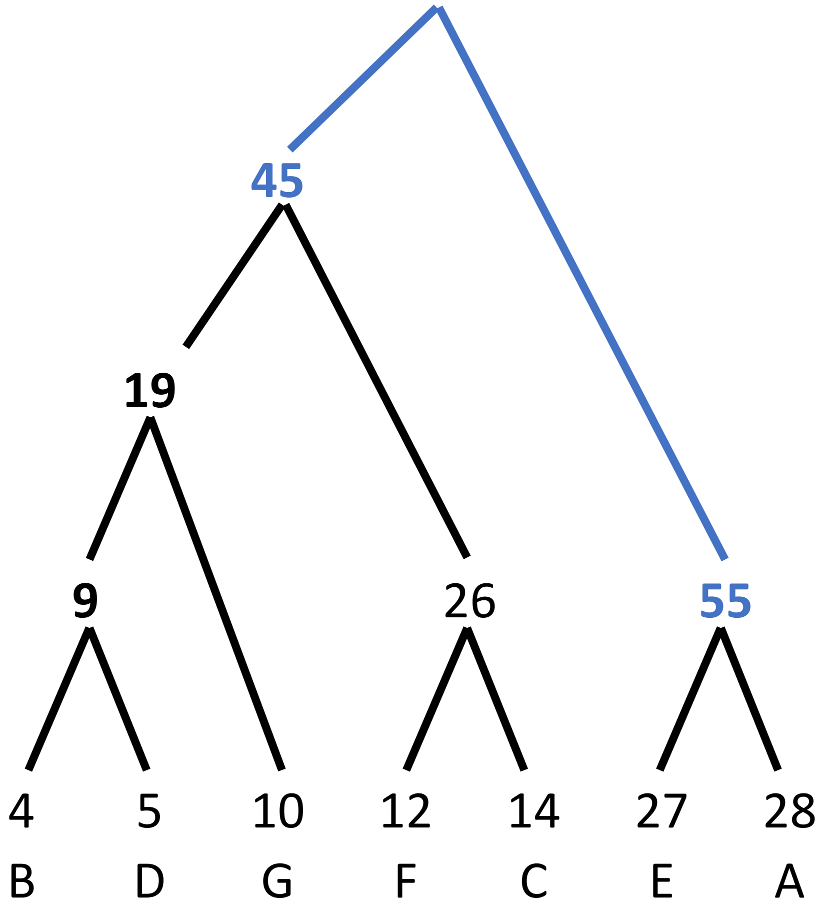
Huffman Trees (example)



Huffman Trees (example)



Huffman Trees (example)



An exercise

- Construct the Huffman code for data in the table.
- Then, encode **ABACABAD** and decode **100010111001010**

SYMBOL	FEQUENCY	CODE
A	0.40	
B	0.10	
C	0.20	
D	0.15	
_	0.15	

Compressed Transmission

- If the compressed file is being sent from one party to another, the parties must agree about the codes used.
  - For example, the trie can be sent along with the message.
- For long files this extra cost is negligible.
- Modern variants of Huffman encoding, like **Lempel-Ziv compression**, assign codes not to individual symbols but to sequences of symbols.

Next lecture

- We briefly discuss complexity theory, NP-completeness and approximation algorithms
- On the final week we will devote time to review all the content