Query Optimisation

COMP3009J: Information Retrieval

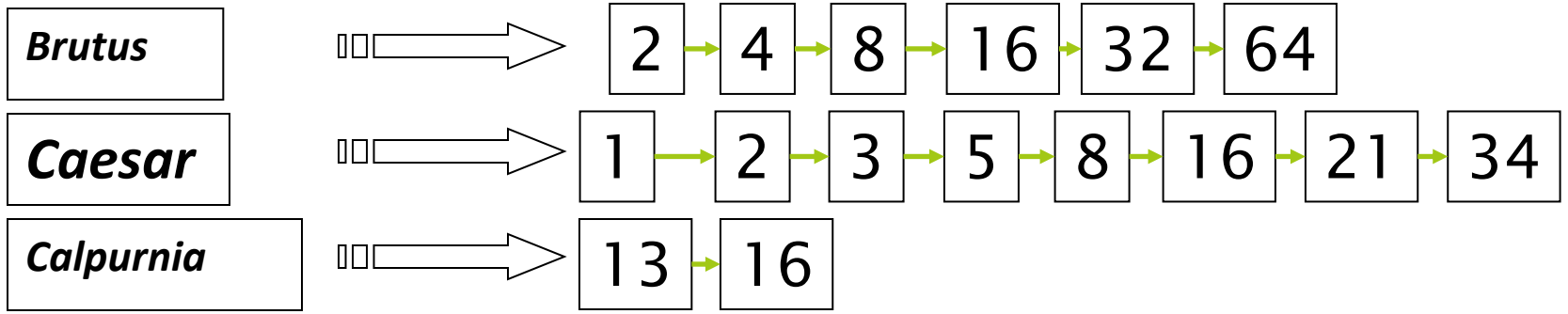
Dr. David Lillis (david.lillis@ucd.ie)

UCD School of Computer Science
Beijing Dublin International College

Some slides adapted from P. Nayak and P. Raghavan

Query Optimisation

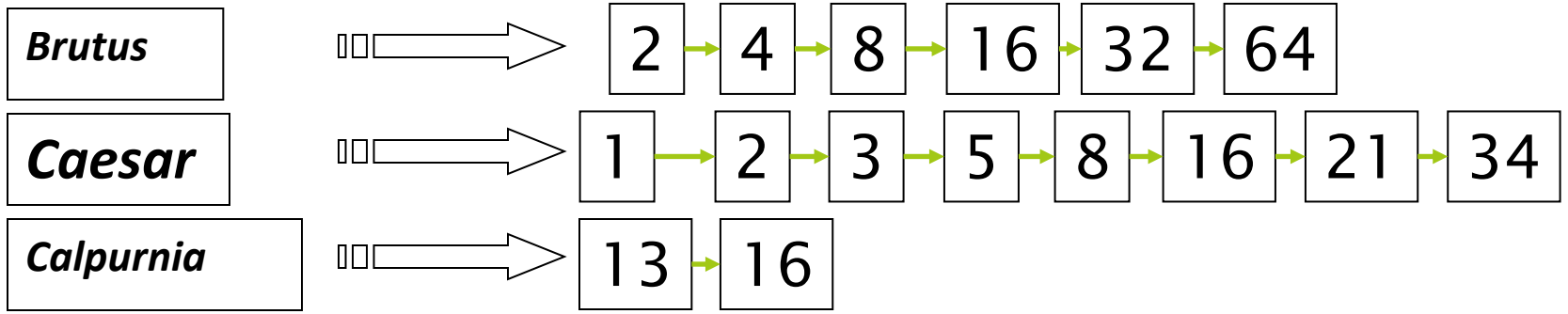
- What is the best order for query processing with postings lists?
- For a query that is an AND of several terms:
  - For each of the terms, retrieve its postings list, and merge with the others.



- Query: ***Brutus*** *AND* **Calpurnia** *AND* **Caesar**

Query Optimisation

- Process in order of increasing frequency.
  - Start with the smallest set, then work towards the bigger ones.
    - **Why?** If we reach the end of the shorter list, we can stop!
    - For a very short list, this can save us a lot of time.
  - This is why we record document frequency in the dictionary.



- Query becomes : (***Calpurnia*** *AND* ***Brutus***) *AND* **Caesar**

More complex optimisation

- What about a combination of *AND* and *OR* operations?
- E.g. (***madding*** *OR* ***crowd***) *AND* (***ignoble*** *OR* ***strife***)
  - Get the document frequencies for all terms.
  - Estimate the size of each *OR* operation.
    - Maximum possible size is the sum of the frequencies of the terms.
  - Process in increasing order of *OR* size.

Exercise

- Recommend a query processing order for

***(tangerine** OR **trees)** AND*
***(marmalade** OR **skies)** AND*
(kaleidoscope** OR **eyes)

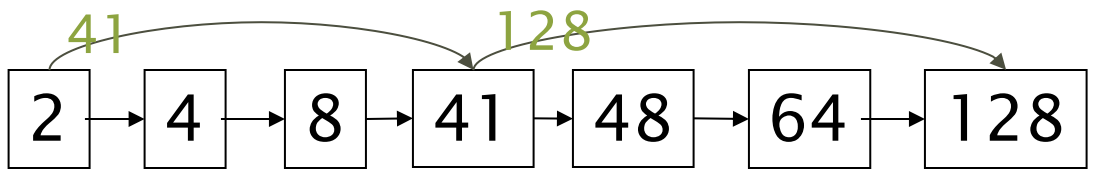
先执行小的

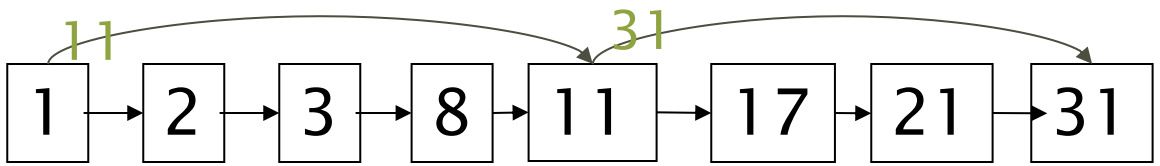
Term	Freq
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

- Which two terms should we process first?

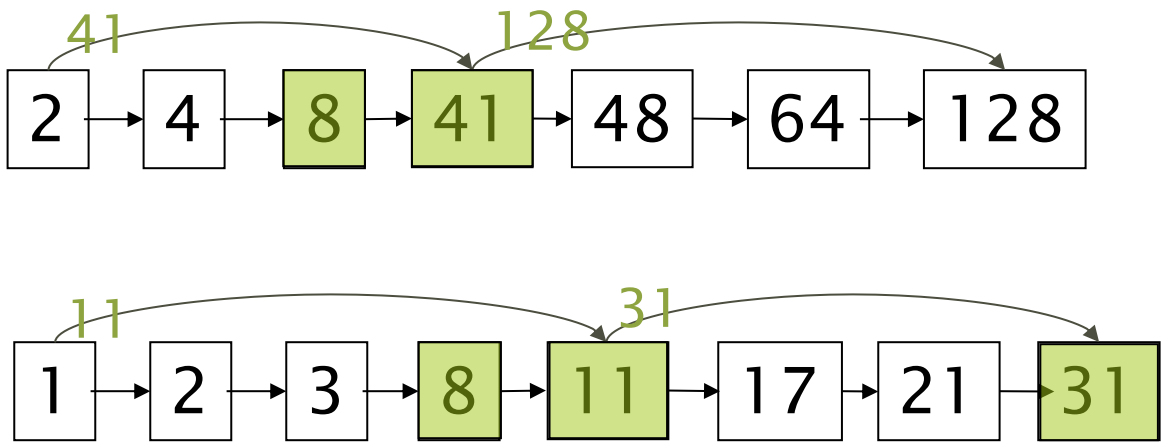
A more optimised data structure: Skip Pointers

- In a regular linked list, each node (which in this case contains a posting) has a pointer/reference to the next node in the list.
- By introducing a second type of reference/pointer, called a **skip pointer**, we can reach later parts of the list without iterating through every element.
- This helps to skip postings that we know will not be in the final search results.





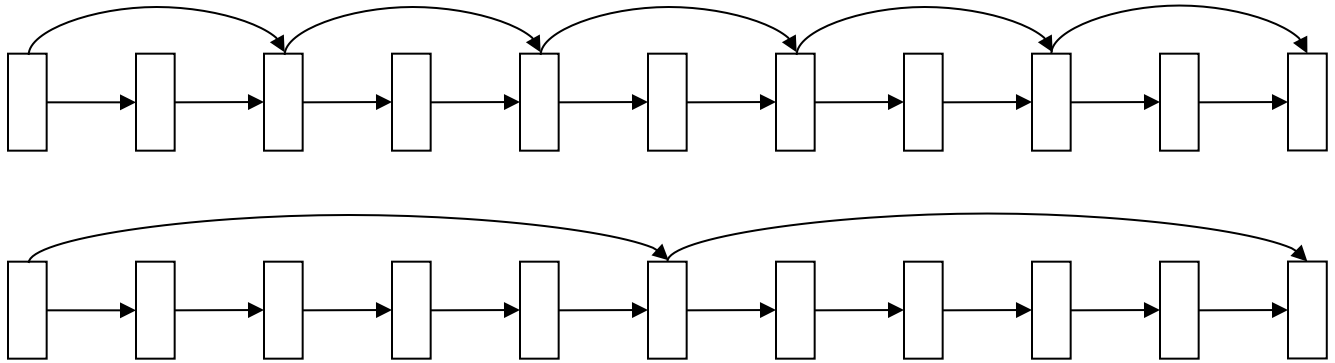
Query processing with skip pointers



- Suppose we have stepped through both lists until we process **8** on each list. We match it and advance.
- We then have **41** on the upper list and **11** on the lower. **11** is smaller.
- But the skip pointer of 11 in the lower list is 31. This is still less than 41 so we can skip everything in between, since we know they cannot be part of the answer.
- Again, this only works if the list is **sorted**.

Where do we place skips?

- Tradeoff:
  - More skips → shorter skip spans ⇒ more likely to skip. But lots of comparisons to skip pointers.
  - Fewer skips → few pointer comparison, but then long skip spans ⇒ few successful skips.



Placing skips

- Simple heuristic: for postings of length L, use $\sqrt{L}$ evenly-spaced skip pointers [Moffat and Zobel 1996]
- This ignores the distribution of query terms.
- Easy if the index is relatively static; harder if L keeps changing because of updates.
- This definitely used to help; with modern hardware it may not unless you have a memory-based index [Bahle et al. 2002]
  - The I/O cost of loading a bigger postings list can outweigh the gains from quicker in memory merging!

Questions