Exercise 2: Advanced Indices

- Any questions?
Lecture last week: Tolerant Retrieval
B+-trees
Lecture last week: Tolerant Retrieval
Wildcard queries

Zu*ch

Rewrite

Zu* AND *ch

Intersection

B+-tree

Reverse B+-tree
Lecture last week: Tolerant Retrieval
Permuterm index
Lecture last week: Tolerant Retrieval
k-gram index
Lecture last week: Tolerant Retrieval
Spelling correction – Dynamic programming
Lecture last week: Tolerant Retrieval
Spelling and k-grams

computer → $cm, cmp, mpu, put, ute, ter, er$

Diagram:
- **computer**
  - $co$: computer
  - $te$: computer
  - $an$: terran
- **com**
  - $er$: computer
  - $err$: terran
- **ran**
  - **ter**
  - **mpu**
  - **omp**
  - **put**
  - **rra**
  - **ute**
Lecture last week: Tolerant Retrieval
Jaccard coefficient

\[ \frac{\text{\# intersection}}{\text{\# union}} = \frac{5}{10} = 0.5 \]
Lecture last week: Tolerant Retrieval
Spelling correction: Final method

Get k-grams from the query term

Look them up in the k-gram index

Compute \textit{edit} distances

Keep terms within small \textit{edit} distances
Lecture last week: Tolerant Retrieval
Phonetic correction: Soundex algorithm

Computer → C0510306 → C0510306 → C5136 → C513
Cmputer → C510306 → C510306 → C5136 → C513
Zurich → Z06020 → Z06020 → Z62 → Z620
Information → I510650305 → I510650305 → I516535 → I516
Letter → L03306 → L0306 → L36 → L360
Retrieval → R0360104 → R0360104 → R3614 → R361
Exercise 3: Tolerant Retrieval

BONUS TIME
Exercise 3: Tolerant Retrieval

- Moodle-based
- Start: 22.3. at 11:00 am
- Deadline: 28.3. at 11:59 pm
- 15/20 required to pass
- Last two questions involve programming exercises
- The rest are theoretical questions
- Important: You have only one try
  - Do not submit unless you are finished!
Exercise 3: Tolerant Retrieval

- Implement k-gram index
- Process wildcard queries
- Process spell correction
Exercise 3: Tolerant Retrieval

- Use function \( \text{window}(\text{seq}, \ n) \)
  - Returns list containing all subsequences of length \( n \) of \( \text{seq} \)
- From now on, postings lists are of type \( \text{set} \)
  - Enables you to use intersection, union and difference

```python
def window(seq, n):
    for i in range(len(seq) - n + 1):
        yield seq[i:i+n]
print(list(window([2, 3, 4, 5, 6], 2)))
print(list(window("astring", 3)))

[[2, 3], [3, 4], [4, 5], [5, 6]]
['ast', 'str', 'tri', 'rin', 'ing']
```
Exercise 3: Tolerant Retrieval

- Implement k-gram index
- Check with example

```python
# wipe existing data
documents = {}
docid_counter = 1
the_index = {}
kgram_index = dict()

for doc in glob.glob('..//shared/corpus/*.txt'):
    docid = docid_counter
documents[docid] = doc
docid_counter += 1

    print("Added document \$s with id \$d" % (doc, docid))
    for word in tokenize_document(doc):
        the_index.setdefault(word, set()).add(docid)

for word in the_index.keys():
    ### TODO for the assignment: add words to the k-gram index
    pass
```
Exercise 3: Tolerant Retrieval

- Wildcard queries:
  - Implement `wildcard_parse(q)`
  - Takes wildcard query (‘som*e’) and returns list containing all keys to query the k-gram index
  - Use again tests to check your implementation

```python
def wildcard_parse(q):
    # TODO for the assignment: parse the wildcard term and return a list of k-grams
    return []
```
Exercise 3: Tolerant Retrieval

- Wildcard queries:
  - Implement `kgram_wildcard_query(q)`
  - Queries k-gram for matching words, given wildcard query
  - Returns list of possible words

```python
def kgram_wildcard_query(q):
    ""
    Query the k-gram index for words matching q. Return the matches as a set
    ""
    grams = wildcard_parse(q)
    "### TODO for the assignment: execute the wildcard query on the k-gram index"
    kgram_matches = {}
    post_filter = re.compile(".*" + q.replace("*", "\w*") + "]")
    res = {r for r in kgram_matches if post_filter.match(r) is not None}
    return res
```
Exercise 3: Tolerant Retrieval

- Wildcard queries:
  - Implement `kgram_wildcard_query(q)`
  - Parse wildcard query with already implemented `wildcard_parse()`
  - For each k-gram get all word matches
    - If one of the k-gram not in the index -> empty set should be returned
  - (optionally) order k-grams according to number of matches
  - Intersect word matches for each k-gram
    - Resulting words should match each k-gram
  - Perform post-processing to exclude false positives (code given)
  - Return a set of matching words
  - Use tests
Exercise 3: Tolerant Retrieval

- Wildcard queries:
  - Implement `kgram_wildcard_query(q)`
  - Needed for BONUS EXERCISES on moodle
  - Use already implemented `execute_query()` to test your implementation

- Add line `kgram_wildcard_query('rend*')` and use the output to answer the first coding question on Moodle.
Exercise 3: Tolerant Retrieval

- Spell-checking with k-gram index
  - Implement `spellcorrect(word)`
  - Compute the k-grams from the input word
  - Find matching words for each k-gram
  - For each candidate word, check Jaccard coefficient
    - If above threshold, add to output list

```python
def spellcorrect(word):
    word_grams = set(window(word, 3))
    # TODO for the assignment: implement spell correction
    return []
```
Exercise 3: Tolerant Retrieval

- Spell-checking with k-gram index
  - Implement `spellcorrect(word)`
  - Check with tests
  - You can use your spell correction as part of the query processor
Exercise 3: Tolerant Retrieval

- Spell-checking with k-gram index
  - Modify Jaccard threshold in first code snippet
  - Needed for BONUS EXERCISE

```
Jaccard_threshold = .3
```

- Change threshold to 0.4 and run assertion test again

```
assert(set(spellcorrect('smoetime')) == {'betime', 'betimes', 'Betimes', 'sometime', 'time', 'lifetime', 'Sometime'})
```