CS 50:
Software Design and Implementation

Querier

Agenda

1. Querier
2. Fuzz testing
3. Activity

Crawler finds pages reachable from seedURL and stores URL, depth, HTML

Goal:

- **Keep track of to crawl pages**
- **Do not revisit pages**

ADTs?

- **Bag to track pages to see**
- **Hashtable for fast look up of pages seen**

If your crawler didn't work well, find example output at: /thayerfs/courses/22fall/cosc050/workspace/tse/tse-output

Use these examples as a source for your indexer

Given:

- seedURL
- directory to store results
- depth to search

Follow links to find all reachable pages from seedURL < depth

Store in a separate file for each page in given directory

- URL
- Depth
- HTML

Indexer uses crawler's results and builds data structure to find pages with words

Goal:
- **Fast look up of documents containing a given word**

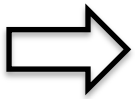
ADTs?
- **Hashtable of words**
- **Set of Counters**

Goal:
- **Keep track of to crawl pages**
- **Do not revisit pages**

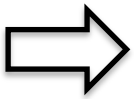
ADTs?
- **Bag to track pages to see**
- **Hashtable for fast look up of pages seen**

Query words

Querier



Indexer



Crawler

If your indexer didn't work well, find example output at: $loc/tse/tse-output (e.g., letters-index-6)

Use these examples as a source for your querier

Given:
- crawler's files

Parse each page's HTML from crawled web pages and discover all words not inside tags

Build data structure to find all documents that contain each discovered word and how many times those words appear

Save data structure

Given:
- seedURL
- directory to store results
- depth to search

Follow links to find all 'internal' reachable pages from seedURL < depth

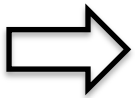
Store in a separate file for each page in given directory
- URL
- Depth
- HTML

Querier finds and ranks pages containing query words

Query words

Goal:
- **Fast ranked list of documents containing query words**

ADTs?
- **Indexer's hashtable of sets of counters**



Given:
- Indexer's data structure
- Query words

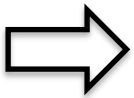
Find web pages containing query words

Rank pages based on how many times each word appears

Consider AND and OR logic

Goal:
- **Fast look up of documents containing a given word**

ADTs?
- **Hashtable of words**
- **Set of Counters**



Given:
- crawler's files

Parse each page's HTML from crawled web pages and discover all words not inside tags

Build data structure to find all documents that contain each discovered word and how many times those words appear

Save data structure

Goal:
- **Keep track of to crawl pages**
- **Do not revisit pages**

ADTs?
- **Bag to track pages to see**
- **Hashtable for fast look up of pages seen**

Given:
- seedURL
- directory to store results
- depth to search

Follow links to find all 'internal' reachable pages from seedURL < depth

Store in a separate file for each page in given directory
- URL
- Depth
- HTML

Querier takes AND as well as OR queries

Query

computer science

- Implicit AND between computer and science
- Pages must have both words

computer and science

- Same as first query

computer or science

- Returns pages that have either computer OR science

baseball or basketball or ultimate frisbee

- Baseball OR basketball OR (ultimate AND frisbee)

DEMO

```
$ ./querier $loc/tse/tse-output/letters-depth-6 $loc/tse/tse-output/letters-index-6
Query? first and search
Query: first and search
Matches 2 documents (ranked):
score 1 doc 3: http://cs50tse.cs.dartmouth.edu/tse/letters/B.html
score 1 doc 8: http://cs50tse.cs.dartmouth.edu/tse/letters/D.html
-----------------------------------------------
Query? tiny search engine
Query: tiny search engine
No documents match.
-----------------------------------------------
Query? NOTE we LOWERcase the query
Query: note we lowercase the query
No documents match.
-----------------------------------------------
Query? spaces do not matter
Query: spaces do not matter
No documents match.
-----------------------------------------------
```

Enter search terms

Pass pages directory and index

Print normalized query

Score for each page

Page number

Return results based on number of times word appears on each page

- **AND: min of scores**
- **OR: add scores**

If no matches, say so

Page URL

Normalized changes query entered to lower case

Ignore spaces

DEMO

```
Query? non-letter characters are disallowed
Error: bad character '-' in query.
Query? even digits as in cs50
Error: bad character '5' in query.
Query? and
Query: and
Error: 'and' cannot be first
Query? or
Query: or
Error: 'or' cannot be first
Query? what about and
Query: what about and
Error: 'and' cannot be last
Query? what about or
Query: what about or
Error: 'or' cannot be last
Query? ^D
```

Only allow alphabetic characters

Queries cannot start or end with AND or OR

Queries also cannot have two operators (AND, OR) consecutively

Agenda

1. Querier
2. Fuzz testing
3. Activity

Fuzz testing sends random input to the querier

Words from index

Words from dictionary

```
static void
generateQuery(const wordlist_t* wordlist, const wordlist_t* dictionary)
{
  // some parameters that affect query generation
  const int maxWords = 6;             // generate 1..maxWords
  const float orProbability = 0.3;    // P(OR between two words)
  const float andProbability = 0.2;   // P(AND between two words)
  const float dictProbability = 0.2;  // P(draw from dict instead of wordlist)

  int qwords = rand() % maxWords + 1; // number of words in query
  for (int qw = 0; qw < qwords; qw++) {
    // draw a word either dictionary or wordlist
    if ((rand() % 100) < (dictProbability * 100)) {
      printf("%s ", dictionary->words[rand() % dictionary->nWords]);
    } else {
      printf("%s ", wordlist->words[rand() % wordlist->nWords]);
    }

    // last word?
    if (qw < qwords-1) {
      // which operator to print?
      int op = rand() % 100;
      if (op < (andProbability * 100)) {
        printf("AND ");
      }
      else if (op < (andProbability * 100 + orProbability * 100)) {
        printf("OR ");
      }
    }
  }
  printf("\n");
}
```

Generate query with qwords in it

Pick random word from either words from index or dictionary

Put an AND or OR operator randomly, but not first or last!

Fuzz testing sends random input to the querier

```
static void
generateQuery(const wordlist_t* wordlist, const wordlist_t* dictionary)
{
  // some parameters that affect query generation
  const int maxWords = 6;             // generate 1..maxWords
  const float orProbability = 0.3;    // P(OR between two words)
  const float andProbability = 0.2;   // P(AND between two words)
  const float dictProbability = 0.2;  // P(draw from dict instead of wordlist)

  int qwords = rand() % maxWords + 1; // number of words in query
  for (int qw = 0; qw < qwords; qw++) {
    // draw a word either dictionary or wordlis
    if ((rand() % 100) < (dictProbability * 100
      printf("%s ", dictionary->words[rand() %
    } else {
      printf("%s ", wordlist->words[rand() % wo
    }

    // last word?
    if (qw < qwords-1) {
      // which operator to print?
      int op = rand() % 100;
      if (op < (andProbability * 100)) {
        printf("AND ");
      }
      else if (op < (andProbability * 100 + orP
        printf("OR ");
      }
    }
  }
  printf("\n");
}
```

Words from index

Number of queries to make

Random seed

```
$ ./fuzzquery $loc/tse/tse-output/letters-index-6 10 0
./fuzzquery: generating 10 queries from 22 words
fourier AND traversal
this OR the the OR tse computational
biology playground OR computational
answers breadth search OR computational OR Mississippians
OR fast
algorithm OR coding eniac the AND home OR breadth
traversal computational playground coding OR the
fast
search the OR fast
home
transform OR huffman OR depth AND graph AND transform
```

Pipe fuzzer's queries to querier to test

```
$ ./fuzzquery $loc/tse/tse-output/letters-index-6 10 0 | ./querier $loc/tse/tse-output/letters-depth-6 $loc/tse/tse-output/letters-index-6
./fuzzquery: generating 10 queries from 22 words
Query: fourier and traversal
No documents match.
----------
Query: this or the the or tse computational
Matches 1 documents (ranked):
score   2 doc   1: http://cs50tse.cs.dartmouth.edu/tse/letters/index.html
----------
Query: biology playground or computational
Matches 1 documents (ranked):
score   1 doc   9: http://cs50tse.cs.dartmouth.edu/tse/letters/C.html
----------
Query: answers breadth search or computational or mississippians or fast
Matches 2 documents (ranked):
score   1 doc   9: http://cs50tse.cs.dartmouth.edu/tse/letters/C.html
score   1 doc   7: http://cs50tse.cs.dartmouth.edu/tse/letters/F.html
----------
Query: algorithm or coding eniac the and home or breadth
Matches 2 documents (ranked):
score   1 doc   2: http://cs50tse.cs.dartmouth.edu/tse/letters/A.html
score   1 doc   3: http://cs50tse.cs.dartmouth.edu/tse/letters/B.html
----------
Query: traversal computational playground coding or the
Matches 1 documents (ranked):
score   1 doc   1: http://cs50tse.cs.dartmouth.edu/tse/letters/index.html
----------
Query: fast
Matches 1 documents (ranked):
score   1 doc   7: http://cs50tse.cs.dartmouth.edu/tse/letters/F.html
```

Crawled pages **Indexed pages**

Issues?

- **Should change random seed (otherwise queries always the same)**
- **Does not check if results are correct**
- **Does show if program crashes!**

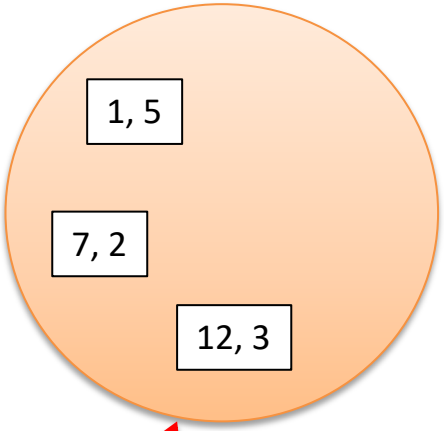
Agenda

1. Querier
2. Fuzz testing
3. Activity

Users enter query words, and the querier must implement AND and OR operations

Note: currently no pages contain both words

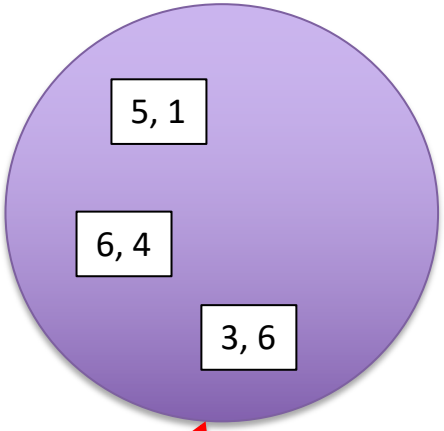
Query word1



DocID, count of pages containing dartmouth

Example:
Doc1 contains word1 5 times

Query word2



DocID, count of pages containing algorithm

Example:
Doc 6 contains word2 4 times

Example
Word 1 = dartmouth
word 2 = algorithm

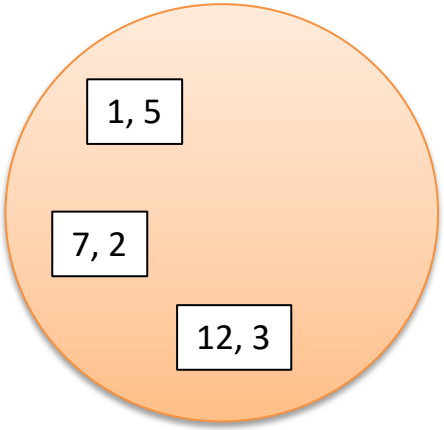
Dartmouth appears on sites 1, 7, and 12

Algorithm appears on sites 3, 5, and 6

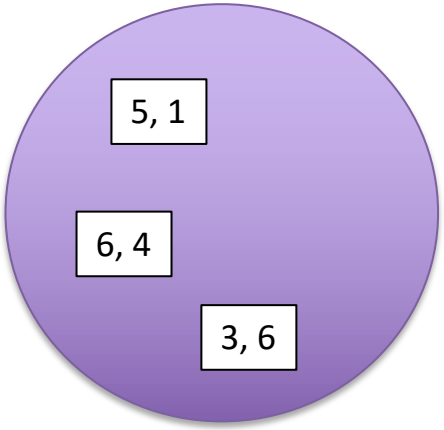
OR is the UNION of two sets

Note: currently no pages contain both words

Query word1



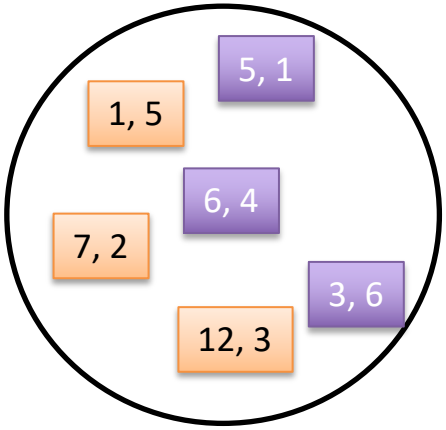
Query word2



dartmouth

OR

algorithm



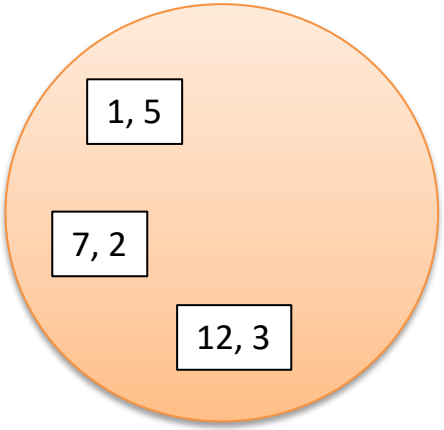
OR returns sites that mention *either* word (UNION)

Return sites sorted by count (3, 1, 6, 12, 7, 5)

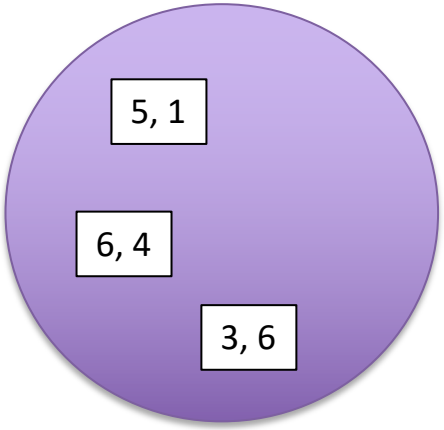
AND is the INTERSECTION of two sets

Note: currently no pages contain both words

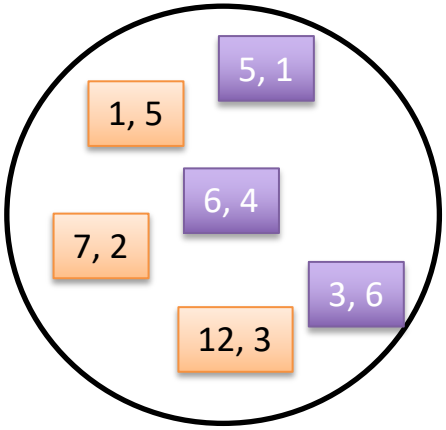
Query word1



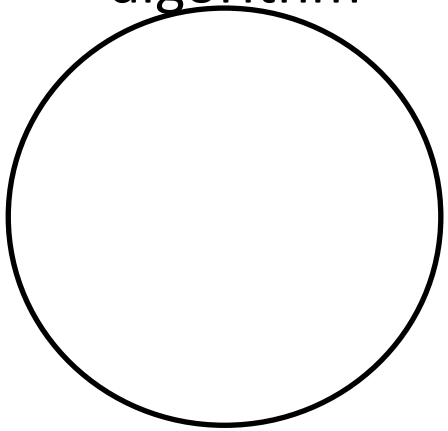
Query word2



dartmouth *OR* algorithm



dartmouth *AND* algorithm



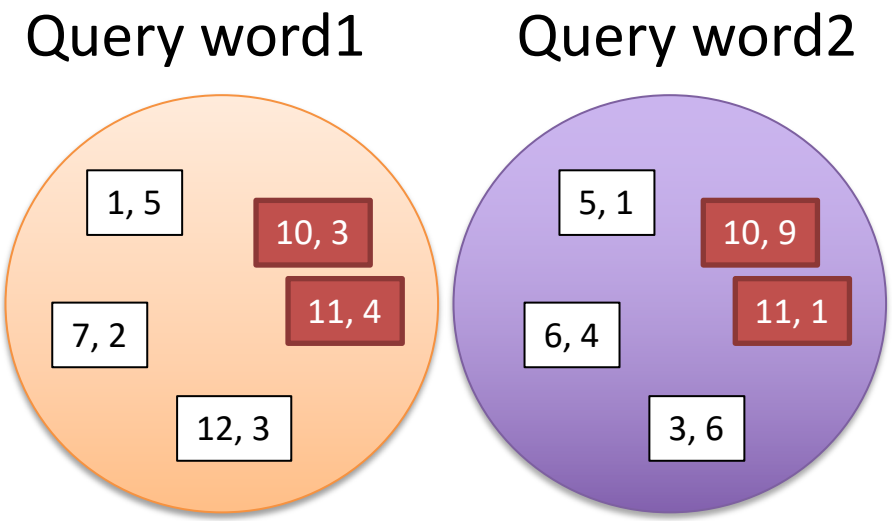
OR returns sites that mention *either* word (UNION)

Return sites sorted by count (3, 1, 6, 12, 7, 5)

AND returns sites that mention *both* words (INTERSECTION)

Currently none

Sometimes the same site contains multiple query words

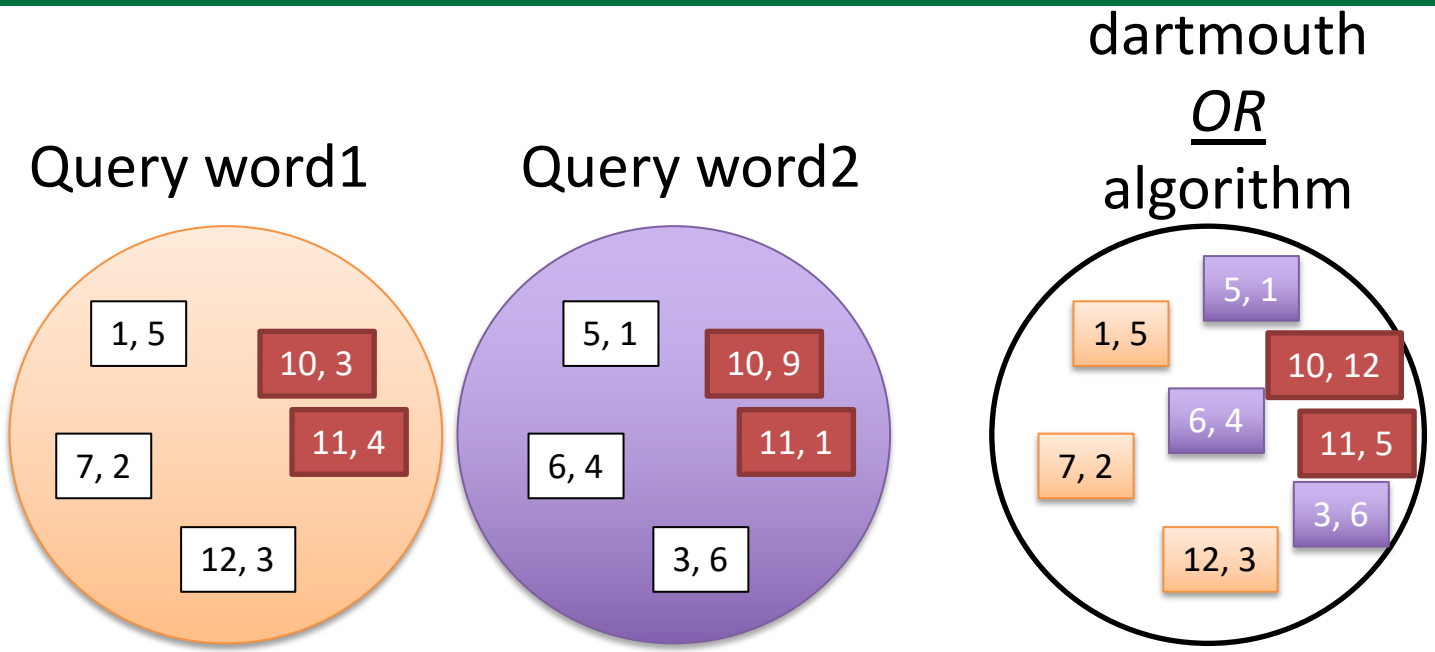


Page 10 contains Dartmouth 3 times and algorithm 9 times

Page 11 contains Dartmouth 4 times and algorithm 1 time

Now suppose pages 10 and 11 each contain query word 1 and 2

OR adds the counts from each site



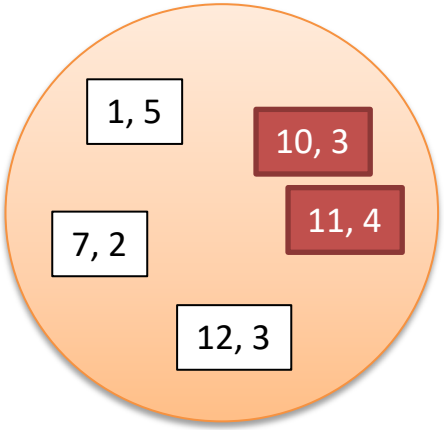
Now suppose pages 10 and 11 each contain query word 1 and 2

OR *adds* counts from each site for overlaps

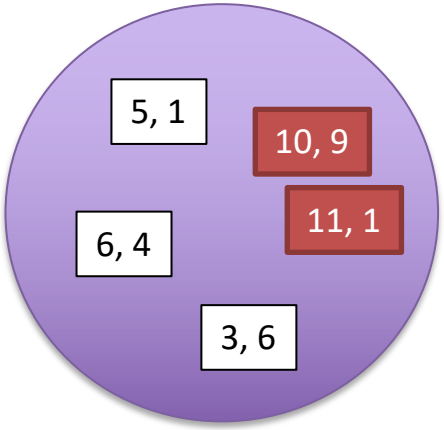
Still return sites sorted by count (10, 3, 11, 1, 6, 12, 7, 5)

AND takes the minimum count between both sites

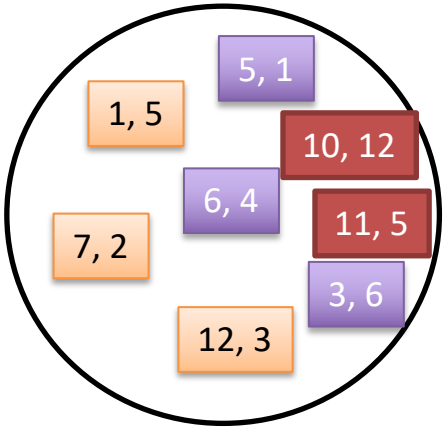
Query word1



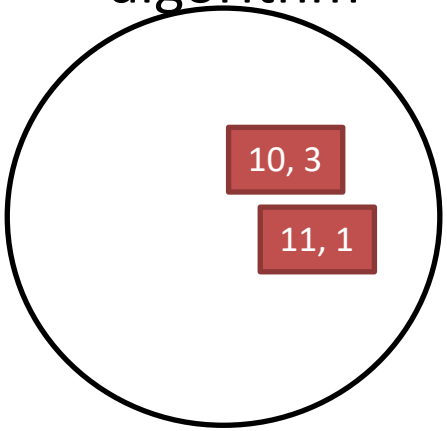
Query word2



dartmouth
OR
algorithm



dartmouth
AND
algorithm



Now suppose pages 10 and 11 each contain query word 1 and 2

OR *adds* counts from each site for overlaps

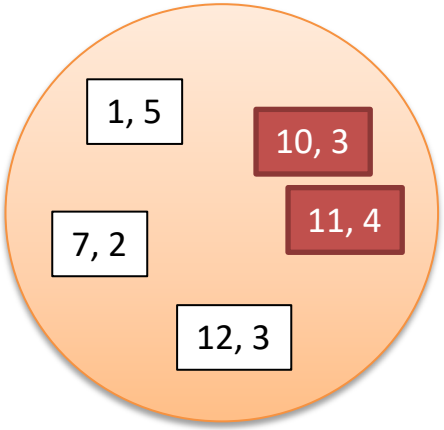
Still return sites sorted by count (10, 3, 11, 1, 6, 12, 7, 5)

AND takes the *minimum* of each site

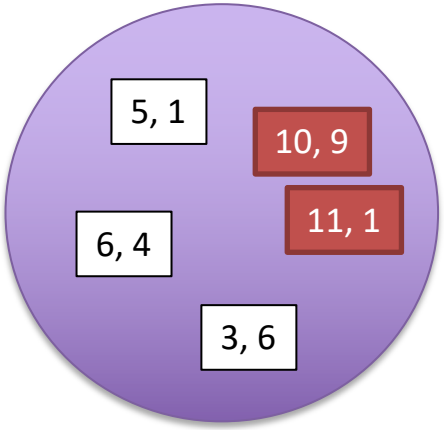
Return sites sorted by count (10, 11)

AND takes the minimum count between both sites

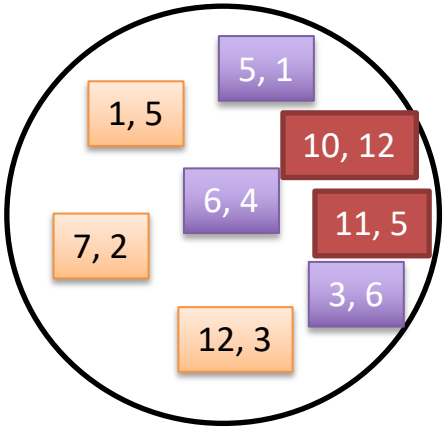
Query word1



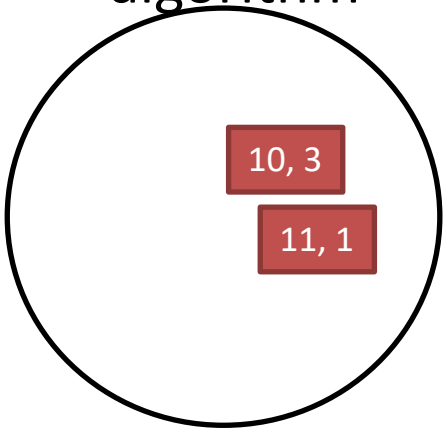
Query word2



dartmouth *OR* algorithm



dartmouth *AND* algorithm



Now suppose pages 10 and 11 each contain query word 1 and 2

If query words appear on more than two sites, add counts for OR, take min for AND

OR *adds* counts from each site for overlaps

Still return sites sorted by count (10, 3, 11, 1, 6, 12, 7, 5)

AND takes the *minimum* of each site

Return sites sorted by count (10, 11)

set_iterate2.c demonstrates UNION

set_iterate2.c

```c
int main()
{
  set_t *setA, *setB, *result;      // three sets

  setA = mem_assert(set_new(), "setA");
  setB = mem_assert(set_new(), "setB");
  result = mem_assert(set_new(), "result");

  printf("Empty result set: ");
  set_print(result, stdout, itemprint);
  putchar('\n');

  printf("Building set A: ");
  set_insert(setA, "Brown", intsave(10));
  set_insert(setA, "Dartmouth", intsave(20));
  set_insert(setA, "Yale", intsave(15));
  set_insert(setA, "Harvard", intsave(8));
  set_insert(setA, "Princeton", intsave(5));
  set_insert(setA, "Columbia", intsave(1));
  set_print(setA, stdout, itemprint);
  putchar('\n');

  printf("Building set B: ");
  set_insert(setB, "Penn", intsave(7));
  set_insert(setB, "Dartmouth", intsave(11));
  set_insert(setB, "Cornell", intsave(9));
  set_insert(setB, "Stanford", intsave(6));
  set_insert(setB, "Princeton", intsave(3));
  set_insert(setB, "Duke", intsave(12));
  set_print(setB, stdout, itemprint);
```

Create three sets

Load setA and setB with college string as key and a score as item

```c
static int *
intsave(int item)
{
  int *saved =
  mem_assert(malloc(sizeof(int)), "intsave");
  *saved = item;
  return saved;
}
```

Why do we call intsave?
set_insert takes a pointer to an item

Merge setA into result, then merge setB into result

```
52   printf("\nMerge of setA into result: \n");
53   set_merge(result, setA);
54   set_print(result, stdout, itemprint);
55   putchar('\n');
56
57   printf("\nMerge of setB into result: \n");
58   set_merge(result, setB);
59   set_print(result, stdout, itemprint);
60   putchar('\n');
61
62   printf("\nDelete the sets...\n");
63   set_delete(setA, itemdelete);
64   set_delete(setB, itemdelete);
65   set_delete(result, itemdelete);
66 }
```

Result is initially empty

set_iterate2.c

Merge (UNION) setA into result
Print result

Merge setB into result
Print result

Delete all three sets when done

set_merge adds items from two sets if the key is in both sets, else insert key, item

set_iterate2.c

```
/* Merge the second set into the first set;
 * the second set is unchanged.
 */
static void
set_merge(set_t *setA, set_t *setB)
{
  set_iterate(setB, setA, set_merge_helper);
}

/* Consider one item for insertion into the other set.
 * If the other set does not contain the item, insert it;
 * otherwise, update the other set's item with sum of item values.
 */
static void
set_merge_helper(void *arg, const char *key, void *item)
{
  set_t *setA = arg;
  int *itemB = item;

  // find the same key in setA
  int *itemA = set_find(setA, key);
  if (itemA == NULL) {
    // not found: insert it
    set_insert(setA, key, intsave(*itemB));
    printf("\t%s added\n", key);
  } else {
    // add to the existing value
    *itemA += *itemB;
    printf("\t%s exists\n", key);
  }
}
```

set_merge does a UNION of two sets

- **Saves result in first set**
- **set_merge_helper called on each item in second set**

setA passed as arg
Cast as set_t
Cast item as integer pointer to setB's item

If setA does not have key from setB, add key and integer to setA

Otherwise, add itemB to itemA to update setA

Activity is to implement INTERSECT given starter code in counters_intersect.c

```c
int main()
{
   // create two counters for demo
   counters_t *c1 = mem_assert(counters_new(), "counters_new() failed");
   counters_t *c2 = mem_assert(counters_new(), "counters_new() failed");

   // init counters 1
   counters_set(c1, 3, 6);
   counters_set(c1, 4, 7);
   counters_set(c1, 5, 1);
   counters_set(c1, 7, 4);

   // init counters 2
   counters_set(c2, 1, 3);
   counters_set(c2, 3, 2);
   counters_set(c2, 5, 4);
   counters_set(c2, 6, 6);
   counters_set(c2, 7, 3);

   // take the intersection, store the results in c1
   counters_intersect(c1, c2);
   counters_print(c1, stdout);
   printf("\n");

   // clean up
   counters_delete(c1);
   counters_delete(c2);

   return 0;
}

// TODO: fill in this function
void counters_intersect(counters_t* ct1, counters_t* ct2)
{

}
```

Complete counters_intersect to keep minimum of counts where keys match