5.1 (5 points) For the two window sizes w = 1 and w = 6, compute and print the 10 nearest neighbors for the query word judges. (Hint: using my implementation, the nearest neighbor for both window sizes is judge, followed by justices for w = 1 and appeals with w = 6.

```
from collections import defaultdict
import math

# function to find indices.
def find_indexes(arr, words_set):
    return [(i, w) for i, w in enumerate(arr) if w in words_set]

# function to find word vectors according to PMI.
def find_pmi_vectors(w, V, Vc):
    my_dict = defaultdict(int)

    with open("/wiki-1percent.txt", "r") as file:
        for line in file:
            words = line.split()
            length = len(words)

            word_indexes = find_indexes(words, V)

            for idx, word in word_indexes:
                ind_l = max(0, idx - w)
                ind_h = min(length, idx + w + 1)

                context_window = words[ind_l:idx] + words[idx + 1:ind_h]

                for context_word in context_window:
                    if context_word in Vc:
                        key = (word, context_word)
                        my_dict[key] += 1

    total_count_N = sum(my_dict.values())

    joint_probabilities = defaultdict(float)
    for (word, context_word), count in my_dict.items():
        joint_probabilities[(word, context_word)] = count / total_count_N

    partial_probability_x = defaultdict(float)
    for word in V:
        partial_probability_x[word] = sum(joint_probabilities[(word, context_word)] for context_word in Vc)

    partial_probability_y = defaultdict(float)
    for context_word in Vc:
        partial_probability_y[context_word] = sum(joint_probabilities[(word, context_word)] for word in V)

    pointwise_mutual_information = defaultdict(float)
    for (word, context_word), joint_prob in joint_probabilities.items():
        partial_prob_x = partial_probability_x[word]
        partial_prob_y = partial_probability_y[context_word]

        if joint_prob > 0 and partial_prob_x > 0 and partial_prob_y > 0:
            pointwise_mutual_information[(word, context_word)] = math.log2(joint_prob / (partial_prob_x * partial_prob_y))
        else:
            pointwise_mutual_information[(word, context_word)] = 0

    word_vectors = defaultdict(dict)
    for (word, context_word), pmi_value in pointwise_mutual_information.items():
        word_vectors[word][context_word] = pmi_value

    return word_vectors


from collections import defaultdict
import math

# function to calculate cosine similarity.
def cosine_similarity(vec1, vec2):

    dot_product = sum(vec1[word] * vec2[word] for word in vec1 if word in vec2)
    norm_vec1 = math.sqrt(sum(value ** 2 for value in vec1.values()))
    norm_vec2 = math.sqrt(sum(value ** 2 for value in vec2.values()))

    if norm_vec1 == 0 or norm_vec2 == 0:
        return 0.0

    return dot_product / (norm_vec1 * norm_vec2)
```

```
from collections import defaultdict
import math

# function to compute nearest neighbours.
def compute_nearest_neighbors(query_word, word_vectors, k):

    cosine_similarities = {}

    query_vec = word_vectors.get(query_word, {})

    for word, vec in word_vectors.items():
        if word != query_word:
            similarity = cosine_similarity(query_vec, vec)
            cosine_similarities[word] = similarity

    nearest_neighbors = sorted(cosine_similarities.items(), key=lambda x: x[1], reverse=True)

    return nearest_neighbors[:k]


with open("/vocab-15kws.txt") as f:
    vocab = f.read().split()

V = set(vocab)

word_vectors_w_1 = find_pmi_vectors(1, V, V)
word_vectors_w_6 = find_pmi_vectors(6, V, V)


# Getting nearest words for "judges".

query_word = "judges"
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'judges' for context window size, w=1:
judge: 0.16088226399323038
justices: 0.1467875404129078
arbitrators: 0.1372853856778382
players: 0.13245878587124824
trustees: 0.1296389481621695
contestants: 0.12422541827146273
officials: 0.122980017022042
admins: 0.12048565742468138
appeals: 0.11843728431064904
officers: 0.11500945538099515

Top 10 nearest neighbors of 'judges' for context window size, w=6:
judge: 0.20254689232437778
appeals: 0.17741896149361888
supreme: 0.17659363749735713
court: 0.17199535193610332
panel: 0.1692578757257143
courts: 0.1666058030872867
jury: 0.16522403791185392
contestants: 0.16440586358293743
justice: 0.16387218457646027
officials: 0.16358549055953606
```

5.2 (10 points) Do nearest neighbors tend to have the same part-of-speech tag as the query word, or do they differ? Does the pattern differ across different part-of-speech tags for the query word? How does window size affect this? Explore these questions by choosing query words with different parts of speech and computing their nearest neighbors. When choosing query words, consider nouns, verbs, adjectives, and prepositions. (Hint: when considering verbs, use inflected forms like transported.) Try a few query words from each part of speech category and see if you can find any systematic patterns when comparing their nearest neighbors across window sizes 1 and 6. When the neighbors differ between window sizes, how do they differ? Can you find any query words that have almost exactly the same nearest neighbors with the two window sizes? Discuss your findings, showing examples of nearest neighbors for particular words to support your claims.

```
# Getting nearest words for "music".

query_word_noun = "music"
```

```
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_noun, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_noun, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_noun}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_noun}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'music' for context window size, w=1:
jazz: 0.2352517854733975
art: 0.2334664078970672
rock: 0.2230073416074984
film: 0.21921528515158653
pop: 0.21817914950264566
dance: 0.21563474661241294
musical: 0.20646070511948098
songs: 0.20607242918804228
american: 0.2018952655971026
albums: 0.19303949397448833

Top 10 nearest neighbors of 'music' for context window size, w=6:
song: 0.3913761453633238
band: 0.3864809051244434
album: 0.3726179946942443
songs: 0.36906367085357134
film: 0.3304670017951474
musical: 0.32050066326277765
jazz: 0.31513733905960817
art: 0.3138467840663773
rock: 0.3134863136889128
released: 0.30159788367532725
```

```
# Getting nearest words for "house".

query_word_noun = "house"
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_noun, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_noun, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_noun}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_noun}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'house' for context window size, w=1:
houses: 0.19169252155350003
county: 0.18194645984787203
family: 0.18155222916945907
building: 0.17814464655988252
's: 0.1758141726614405
city: 0.16935423702804722
village: 0.16609334135294757
park: 0.16467741939280367
state: 0.1632929110612971
john: 0.16199130560615968

Top 10 nearest neighbors of 'house' for context window size, w=6:
building: 0.3332559481112943
built: 0.32623197949046445
church: 0.31210832770244046
john: 0.3061576359227404
county: 0.3017372348519806
street: 0.3000841495152304
home: 0.30003267129055716
hall: 0.2963653671315225
park: 0.28398347930673074
st: 0.28251761960010907
```

For verbs

```
# Getting nearest words for "evaluated".

query_word_verb = "evaluated"
```

```
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_verb, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_verb, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_verb}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_verb}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'evaluated' for context window size, w=1:
dismantled: 0.17821160475608214
examined: 0.1628428936681349
adjusted: 0.15414619856907355
summarize: 0.15385939228557674
addressed: 0.15349579392116133
detained: 0.14883484070496825
ratified: 0.148738302241521
discussed: 0.14762722998714475
cleaned: 0.1451836231913981
handled: 0.1385818278120693

Top 10 nearest neighbors of 'evaluated' for context window size, w=6:
evaluate: 0.1318963738009448
assess: 0.12241652543601635
evaluation: 0.11891691842179625
analysis: 0.10741406993296336
determine: 0.10695917786605068
testing: 0.10473747429420437
efficiency: 0.10470472556180471
algorithm: 0.10441124130208577
declining: 0.10418318693728329
organisms: 0.10248677558157693
```

```
# Getting nearest words for "designed".

query_word_verb = "designed"
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_verb, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_verb, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_verb}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_verb}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'designed' for context window size, w=1:
built: 0.18788394185675528
constructed: 0.16636813821699417
developed: 0.1544664683233103
used: 0.1458813806421581
equipped: 0.14224759585567698
design: 0.14202047737725906
created: 0.13490603128813852
available: 0.12887945297089706
written: 0.12781377060348903
based: 0.12695649661481284

Top 10 nearest neighbors of 'designed' for context window size, w=6:
design: 0.34361406795013977
built: 0.3064893199659197
developed: 0.2924548599947247
using: 0.28584927318969616
building: 0.27966870535202
systems: 0.2762699481861762
features: 0.26385085295354144
architect: 0.25329009324872653
type: 0.24844383611483223
large: 0.24699598479847665
```

For adjectives

```
# Getting nearest words for "intelligent".

query_word_adjective = "intelligent"
```

```
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_adjective, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_adjective, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_adjective}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_adjective}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'intelligent' for context window size, w=1:
efficient: 0.12255818536677039
aggressive: 0.11500946607794797
stable: 0.11458227446771488
centralized: 0.11339123578583084
rational: 0.1104740891484247
productive: 0.10980739079925649
impressive: 0.10857972767172237
informative: 0.10831067869673684
competent: 0.10712566705836
interesting: 0.10303052095003111

Top 10 nearest neighbors of 'intelligent' for context window size, w=6:
understanding: 0.13224091645521857
learning: 0.13182341586222687
processes: 0.13048663315787143
interesting: 0.1300055877042562
technologies: 0.12901707401933601
approach: 0.1285031411499165
meaningful: 0.1282513828900368
humans: 0.12824605221358182
simple: 0.1277690463350835
highly: 0.12607671626680444
```

```
# Getting nearest words for "beautiful".

query_word_adjective = "beautiful"
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_adjective, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_adjective, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_adjective}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_adjective}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'beautiful' for context window size, w=1:
attractive: 0.12187995430596532
scenic: 0.11814294372346498
dark: 0.10867641614627511
amazing: 0.10198041344311475
whose: 0.10153389585546795
magnificent: 0.10118077298702935
quiet: 0.10044579875566965
picturesque: 0.10006369755970596
strange: 0.0988855561424134
surrounding: 0.09863158705401244

Top 10 nearest neighbors of 'beautiful' for context window size, w=6:
love: 0.17828259432224108
girl: 0.16456668491465223
woman: 0.16336597526875327
beauty: 0.16134986779111796
herself: 0.15709246367225754
and: 0.15602031037179126
lady: 0.1557563137081618
dark: 0.155520700224912
man: 0.1552607898459518
features: 0.15389114110173086
```

For preposition

```
# Getting nearest words for "through".

query_word_preposition = "through"
```

```
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_preposition, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_preposition, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_preposition}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_preposition}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'through' for context window size, w=1:
into: 0.30558390241433514
between: 0.2528322864478591
and: 0.24382873770808097
from: 0.24380185547652541
across: 0.23139582983376777
along: 0.22646735149432243
over: 0.21889625779873495
during: 0.21627907240000313
down: 0.21056292557715736
out: 0.20786233466746318

Top 10 nearest neighbors of 'through' for context window size, w=6:
into: 0.2973615239226622
via: 0.28087829032815664
along: 0.26499868283682904
across: 0.26187526679794154
system: 0.2614267433341713
around: 0.25498805635218624
using: 0.2508040658468931
between: 0.24481792741647884
large: 0.24091708442353477
water: 0.23945862400388493
```

```
# Getting nearest words for "near".

query_word_preposition = "near"
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_preposition, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_preposition, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_preposition}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_preposition}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'near' for context window size, w=1:
nearby: 0.23907992982421833
at: 0.21760115043250708
along: 0.21305370209656413
road: 0.20041885163836667
between: 0.19908711274640256
east: 0.1982496004179597
west: 0.19231320984546144
around: 0.1910503437204058
downtown: 0.18737247280240515
north: 0.18527529025979936

Top 10 nearest neighbors of 'near' for context window size, w=6:
located: 0.45370454292922346
north: 0.4386368009409462
river: 0.43757792416631197
east: 0.42686101820991784
west: 0.4259288572913421
road: 0.4080553640307897
south: 0.3987859059831599
park: 0.3914330106161848
town: 0.38400666633871566
lake: 0.3780965523208657
```

5.3 (10 points) Now try choosing words with multiple senses (e.g., bank, cell, apple, apples, axes, frame, light, well, etc.) as query words. What appears to be happening with multisense words based on the nearest neighbors that you observe? What happens when you compare the neighbors with different window sizes (w = 1 vs. w = 6)? Discuss your findings, showing examples of nearest neighbors for particular words to support your claims.

For multisense words: 1) Bank

```
# Getting nearest words for "bank".

query_word_multisense = "bank"
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_multisense}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_multisense}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'bank' for context window size, w=1:
banks: 0.1827920565332767
company: 0.14277384047080313
insurance: 0.13049722349957968
corporation: 0.12775249391520085
railway: 0.12268850302979989
government: 0.12231567199691568
banking: 0.11728458034171654
companies: 0.11206946596531087
institute: 0.11144760694487782
conference: 0.11055429400995383

Top 10 nearest neighbors of 'bank' for context window size, w=6:
corporation: 0.26188916237681353
banks: 0.24753450386934825
company: 0.24304049154327279
railway: 0.239596873105255
river: 0.2395293436868915
capital: 0.23562226231919461
west: 0.23489102983532872
central: 0.229471745987817
east: 0.22842509905896707
northern: 0.22354435952419194
```

2) Cell

```
# Getting nearest words for "cell".

query_word_multisense = "cell"
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_multisense}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_multisense}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'cell' for context window size, w=1:
cells: 0.27825498024044576
cellular: 0.1957797465237583
protein: 0.1550193052150567
tissue: 0.15453916660414296
brain: 0.12431467038168423
proteins: 0.12312275889052125
tissues: 0.1221508188624583
growth: 0.11580589956377435
human: 0.11084034222648313
enzyme: 0.11070403124912466

Top 10 nearest neighbors of 'cell' for context window size, w=6:
cells: 0.4206664569903037
protein: 0.29795036670888736
membrane: 0.2817386492157226
proteins: 0.2790529621329552
cellular: 0.2689622324027641
dna: 0.2615435593904246
genes: 0.24887530746018052
function: 0.24692677216205736
tissue: 0.24488699217431567
```

```
brain: 0.24285347536380011
```

3) Apple

```
# Getting nearest words for "apple".

query_word_multisense = "apple"
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_multisense}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_multisense}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'apple' for context window size, w=1:
cherry: 0.14418837734849888
chili: 0.1315822689286924
desktop: 0.11426697226625163
olive: 0.10479365296990972
tulip: 0.10406617695978118
orange: 0.10384858188444925
palm: 0.09503250570017008
pine: 0.09494292849310292
atari: 0.0932794409245937
wines: 0.0924749129833824

Top 10 nearest neighbors of 'apple' for context window size, w=6:
os: 0.22349178687178287
microsoft: 0.21797351180423696
macintosh: 0.20376424225413126
mac: 0.2006762960575836
ios: 0.19998904264791953
software: 0.19983747510228544
desktop: 0.19650194384710695
computers: 0.1854881332651712
linux: 0.180520163889551
iphone: 0.17578379750759945
```

4) Apples

```
# Getting nearest words for "apples".

query_word_multisense = "apples"
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_multisense}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_multisense}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'apples' for context window size, w=1:
brains: 0.17662940363544993
impatient: 0.15156549546124998
kinds: 0.14219633459071232
jokes: 0.13784925238645185
guys: 0.1222782762120528
classmates: 0.12077728948351064
vampires: 0.11879178022071352
tired: 0.1171744495645414
continents: 0.11681321730904437
candles: 0.11634458260364082

Top 10 nearest neighbors of 'apples' for context window size, w=6:
fruits: 0.14159256106482185
grapes: 0.14056907325790657
vegetables: 0.14045800212516943
wheat: 0.13831172659649932
carrots: 0.13435873572727194
fruit: 0.1322480377038295
```

```
seeds: 0.131898847837544
cooked: 0.1252818733959207
crops: 0.12182059005436091
maize: 0.11794401345948345
```

5) Axes

```
# Getting nearest words for "axes".

query_word_multisense = "axes"
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_multisense}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_multisense}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'axes' for context window size, w=1:
phases: 0.16950622163465687
tributaries: 0.13527058064842007
qualities: 0.12158070937132485
paths: 0.1197506256710099
viewpoints: 0.11972740295337372
spells: 0.11164791559952146
sorts: 0.10865563111839704
branches: 0.10859294928483704
motifs: 0.1065591977510798
frames: 0.10387825055177326

Top 10 nearest neighbors of 'axes' for context window size, w=6:
angles: 0.12401253204025457
flint: 0.11839587778420939
neolithic: 0.11317106126229176
axe: 0.10833566976408206
symmetry: 0.10810610486154082
parallel: 0.10784579372980911
shapes: 0.10650966487929263
puzzle: 0.10222630940545116
knives: 0.10187960720045423
vectors: 0.0999652304383745
```

6) Frame

```
# Getting nearest words for "frame".

query_word_multisense = "frame"
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_multisense}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_multisense}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'frame' for context window size, w=1:
brick: 0.15457089054360534
frames: 0.14746364634916354
two-story: 0.14117492842615603
rear: 0.12277116159534333
structure: 0.12162546943450016
panels: 0.11918997258576204
framed: 0.11714480772844232
storey: 0.11317850589061738
wooden: 0.11108923803459136
wheels: 0.10900990053710542

Top 10 nearest neighbors of 'frame' for context window size, w=6:
rear: 0.22554454917799965
steel: 0.20892461349206842
roof: 0.20887095236614528
```

```
structure: 0.20862870378096274
brick: 0.20207798951034503
wooden: 0.20140654899256935
frames: 0.1919240972253489
metal: 0.1893694050338087
wood: 0.18543013225913135
wheels: 0.18303998605072644
```

7) Light

```
# Getting nearest words for "light".

query_word_multisense = "light"
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_multisense}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_multisense}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'light' for context window size, w=1:
heavy: 0.19632297388919315
lights: 0.15116724909899712
water: 0.14343244331974714
dark: 0.14137663303032313
fire: 0.14131076842143678
regiment: 0.1302571375426832
division: 0.12795276130445996
force: 0.1245318371918821
large: 0.12401130369633516
pale: 0.12089518664340862

Top 10 nearest neighbors of 'light' for context window size, w=6:
using: 0.27374906602272125
surface: 0.261611829575851
usually: 0.26087120674735376
water: 0.25720174812437097
body: 0.2540652997326469
heavy: 0.2505553876318451
red: 0.2487953214375574
dark: 0.2461297285420266
color: 0.24419022508872737
energy: 0.24286282142914825
```

8) Well

```
# Getting nearest words for "well".

query_word_multisense = "well"
k = 10

compute_nearest_neighbours_w_1 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_1, k)
compute_nearest_neighbours_w_6 = compute_nearest_neighbors(query_word_multisense, word_vectors_w_6, k)


print(f"Top {k} nearest neighbors of '{query_word_multisense}' for context window size, w=1:")
for neighbor, similarity in compute_nearest_neighbours_w_1:
    print(f"{neighbor}: {similarity}")

print(f"\nTop {k} nearest neighbors of '{query_word_multisense}' for context window size, w=6:")
for neighbor, similarity in compute_nearest_neighbours_w_6:
    print(f"{neighbor}: {similarity}")
```

```
Top 10 nearest neighbors of 'well' for context window size, w=1:
poorly: 0.2348448879449925
be: 0.2015503182854486
however: 0.19133210791846314
there: 0.19125103524239537
been: 0.1912043102468751
united: 0.18474151001388656
preserved: 0.1820202523959068
discussion: 0.17635219795999532
debate: 0.17397506793210257
list: 0.16759854977348515

Top 10 nearest neighbors of 'well' for context window size, w=6:
```

```
many: 0.3508470351371892
such: 0.3505578300831992
including: 0.3208532283601029
like: 0.31716721899024447
other: 0.30514221163954125
most: 0.30193611186374536
several: 0.2948282143811019
some: 0.29313903589458595
these: 0.28612724149577135
both: 0.2831391200111555
```