hw5

April 13, 2020

[9]:
```
!jupyter nbconvert --to script hw5.ipynb
!jupyter nbconvert --to pdf hw5.ipynb
```

```
[NbConvertApp] Converting notebook hw5.ipynb to script
[NbConvertApp] Writing 4174 bytes to hw5.py
[NbConvertApp] Converting notebook hw5.ipynb to pdf
[NbConvertApp] Writing 36261 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 35457 bytes to hw5.pdf
```

[13]:
```
from typing import *
import numpy as np
import bitstring as bs
```

[14]:
```
# max_len determines the maximum length in the second dimension of Python list
def max_len(items: List) -> int:
    maxLen = 0
    for item in items:
        if (itemLen := len(item)) > maxLen:
            maxLen = itemLen
    return maxLen

# uniform_octree_len expands octree levels to where the depths are uniform
def uniform_octree_len(rSequences: List, maxLen: int) -> List:
    shortLenMaskFunc = lambda rSeq: len(rSeq) != maxLen
    while any(shortLenMask := list(map(shortLenMaskFunc, rSequences))):
        i = shortLenMask.index(True)
        rSequences.extend([rSequences[i].copy() + [str(digit)] for digit in␣
 ↪range(8)])
        del rSequences[i]

    return sorted(rSequences)
```

```python
# lookup_table for Table 2 from Ahuja and Nash
def lookup_table(label: Union[str, int], direction: Union[str, int]) -> str:
    if direction not in {"x", "y", "z", "0", "1", "2", 0, 1, 2}:
        raise IndexError("direction must be in {'x', 'y', 'z', '0', '1', '2',␣
↪0, 1, 2}")
    if type(label) == str:
        label = int(label)
    if label > 7 or label < 0:
        raise IndexError("label must be between 0 and 7")

    if direction in ("x", "y", "z"):
        direction = ("x", "y", "z").index(direction)
    elif direction in ("0", "1", "2"):
        direction = int(direction)

    table = (
        (1, 2, 4),
        (10, 3, 5),
        (3, 10, 6),
        (12, 11, 7),
        (5, 6, 10),
        (14, 7, 11),
        (7, 14, 12),
        (16, 15, 13)
    )
    return str(table[label][direction])

# octree_displacement returns the rSequence after it has been translated in␣
↪`direction` by the `displacement` (binary/2's comp) using the `lookup_table`
def octree_displacement(rSeq: str, displacement: str, direction: Union[int,␣
↪str]) -> List:
    # copy so we don't mutate outside lists
    displacement = displacement.copy()
    rSeq = rSeq.copy()

    for i in reversed(range(len(displacement))):
        if int(displacement[i]) == 2:
            displacement[i] = str(0)
            if i != 0:
                displacement[i-1] = str(int(displacement[i-1]) + 1)

        if int(displacement[i]) == 0:
            continue

        tableVal = lookup_table(rSeq[i], direction)
        if int(tableVal) < 10:
            rSeq[i] = tableVal
```

```
        else:
            rSeq[i] = tableVal[-1]
            if i != 0:
                displacement[i-1] = str(int(displacement[i-1]) + 1)

    return rSeq
```

[15]:
```
# read in A and B from the text files (copied from .pdf document)
with open("ObjectA.txt", "rt") as f:
    A  = [rSeq.strip() for rSeq in f.read().split(",")]

with open("ObjectB.txt", "rt") as f:
    B  = [rSeq.strip() for rSeq in f.read().split(",")]

# split into letter lists & drop the 'r'
A = [list(rSeq)[1:] for rSeq in A]
B = [list(rSeq)[1:] for rSeq in B]

# find the length needed to expand to
if (ALen := max_len(A)) >= (BLen := max_len(B)):
    maxLen = ALen
elif (ALen := max_len(A)) < (BLen := max_len(B)):
    maxLen = BLen

# make uniform lengths
A = uniform_octree_len(A, maxLen)
B = uniform_octree_len(B, maxLen)

# x and y binary/2's comp representation
x = list(str(bs.Bits(int=-5, length=maxLen).bin))
y = list(str(bs.Bits(int=48, length=maxLen).bin))

# translate B
BPrime = []
for b in B:
    BPrimeX = octree_displacement(b, x, "x")
    BPrimeXY = octree_displacement(BPrimeX, y, "y")
    BPrime.append(BPrimeXY)

# test two results
assert("".join(BPrime[7]) == "52220026")
assert("".join(BPrime[-1]) == "70007036")

# write it out to file
print(f"B': ")
with open("BPrime.txt", "wt") as f:
    for b in BPrime:
```

```
        out = f"r{''.join(b)}"
        print(out)
        f.write(f"{out}\n")

# need to convert A and BPrime to hashable containers
A = set([tuple(a) for a in A])
BPrime = set([tuple(b) for b in BPrime])

# find intersection and volume
intersection = A & BPrime
volSmallestOctree = (100 / (2**maxLen) ** 3)
volTotal = len(intersection) * volSmallestOctree
print(f"Intersection volume: {volTotal} cm^3")
```

```
B':
r43331131
r52220020
r43331133
r52220022
r43331135
r52220024
r43331137
r52220026
r52220021
r52220030
r52220023
r52220032
r52220025
r52220034
r52220027
r52220036
r43331171
r52220060
r43331173
r52220062
r43331175
r52220064
r43331177
r52220066
r52220061
r52220070
r52220063
r52220072
r52220065
r52220074
r52220067
r52220076
```

r43331531
r52220420
r43331533
r52220422
r52220421
r52220430
r52220423
r52220432
r61113311
r70002200
r61113313
r70002202
r61113315
r70002204
r61113317
r70002206
r70002201
r70002210
r70002203
r70002212
r70002205
r70002214
r70002207
r70002216
r61113351
r70002240
r61113353
r70002242
r61113355
r70002244
r61113357
r70002246
r70002241
r70002250
r70002243
r70002252
r70002245
r70002254
r70002247
r70002256
r61113711
r70002600
r61113713
r70002602
r70006111
r70007000
r70006113
r70007002

```
r70006115
r70007004
r70006117
r70007006
r70007001
r70007010
r70007003
r70007012
r70007005
r70007014
r70007007
r70007016
r70006131
r70007020
r70006133
r70007022
r70006135
r70007024
r70006137
r70007026
r70007021
r70007030
r70007023
r70007032
r70007025
r70007034
r70007027
r70007036
Intersection volume: 0.0 cm^3
```

[ ]: