CS 412
Homework 2
Jakub Klapacz

1.

a. The number of non-empty aggregated cells in this cube is 1918.

$$NumAggregates = (2^{10} - 1) + (2^{10} - 1) - 2^7$$
$$= 2^{11} - 2^7 - 2$$
$$= 1918$$

We calculate $2^{10} - 1$ twice because there are two base cells, and since each base cell has 10 dimensions we can either aggregate a dimension or choose not to. We subtract the 1 because we are concerned with the aggregated cells so we must discount the base cell in our calculation. We also subtract 2^7 because we must remove the overlap that we get from the base cells sharing attributes $b_4 \dots b_{10}$. This gets us to our final solution of 1918.

b. If the iceberg condition is $count \geq 2$ then we have $2^7 = 128$ nonempty aggregated cells. This is because the only cuboids that have $count \geq 2$ are the cuboids with the first three dimensions as *'s:

$$(*, *, *, b_4, b_5, b_6, b_7, b_8, b_9, b_{10})$$

Where the dimensions $b_4, b_5, b_6, b_7, b_8, b_9, b_{10}$ can be either aggregated (*'ed) or not. Since there are 7 dimensions in $b_4, b_5, b_6, b_7, b_8, b_9, b_{10}$ that means that there are 2^7 ways we can do this. Therefore there are 2^7 nonempty aggregated cells.

c. The closed cell with count 2 has 7 non-star attributes. This is because **the** closed cell in this cube is the cell:

$$(*, *, *, b_4, b_5, b_6, b_7, b_8, b_9, b_{10})$$

This cell is closed because there are no further descendants with the same count, making any of the first three attributes a specific value will reduce the count of the cell to 1.

d. There are 3 closed cells in the full cube. They are the 2 base cells that were given (because by definition base cells are closed) and the 1 cell from part (c) above. No other cells in the cube are closed.

2.

a. There are 24 cuboids in this cube. We can get this result if we consider the cuboids to be of the form:

$$(Location, Category, Rating, Price)$$

Where Category, Rating, and Price can either be aggregated or not, and location can be organized by city, state, or aggregated. This gives us the following choices for each dimension:

$$(\{City, State, *\}, \{Category, *\}, \{Rating, *\}, \{Price, *\})$$

This gives us:

$$3 * 2 * 2 * 2 = 24 \text{ cuboids}$$

b. There are 48 cells in the cuboid (Location(city), Category, Rating, Price)
c. There are 34 cells in the cuboid (Location(state), Category, Rating, Price)
d. There are 23 cells in the cuboid (*, Category, Rating, Price)
e. The cell (Location(state)="Illinois", *, Rating = 3, Price="Moderate") has a count of 2
f. The cell (Location(city)="Chicago", category="Food", *,*) has a count of 2

Below is a screenshot of the commands my script took in order to generate these results. The script takes one of three options:

[cells,count,quit]

The usage for these commands is as follows:

Here cells takes in 4 parameters and returns the number of cells in the cuboid matching that specification:

cells<tab><param1,param2,param3,param4>

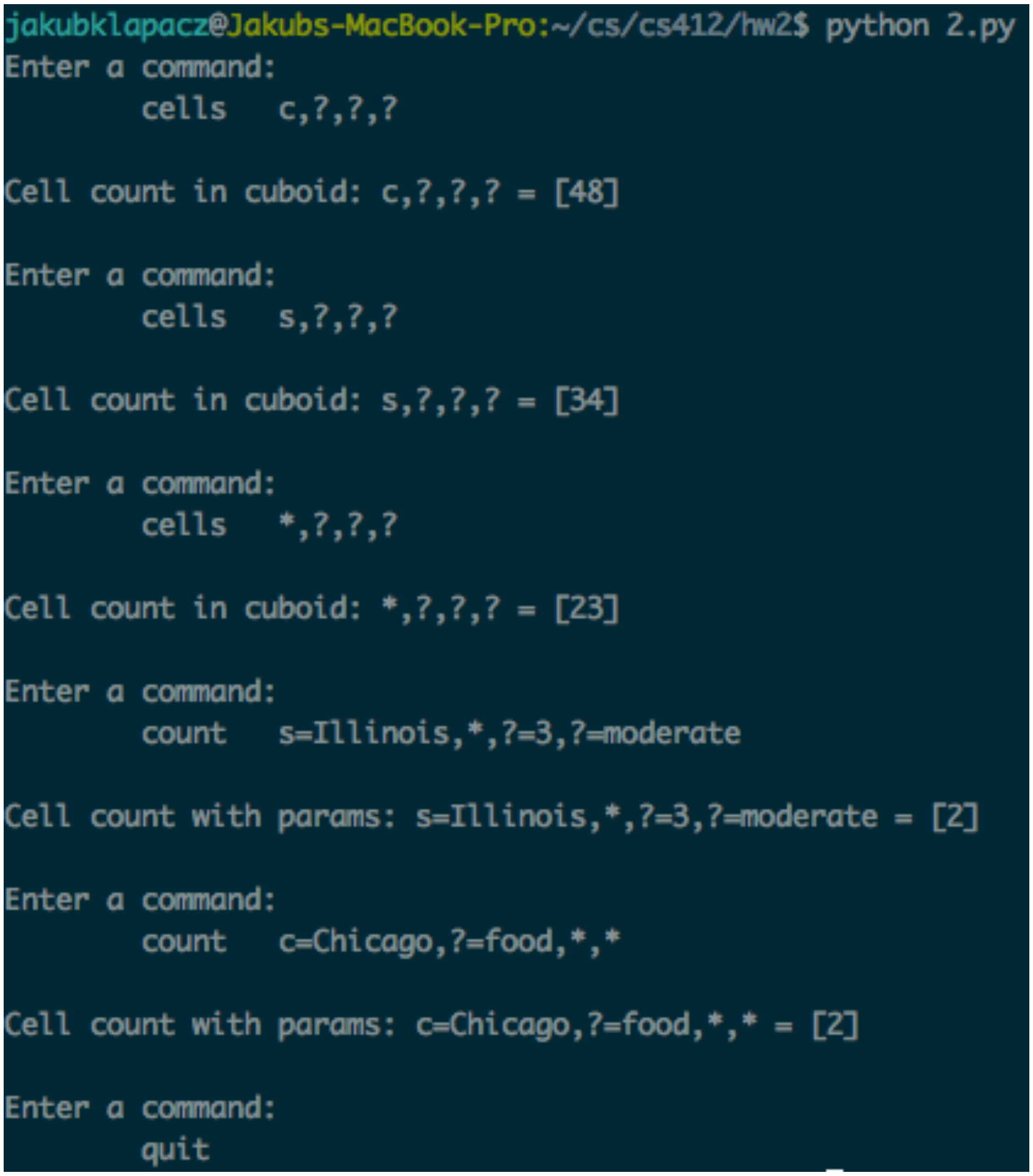
Where param1 can be either c for city, s for state, or * for aggregate. The other three parameters are either * for aggregate or ? for don't aggregate.

Count takes in similar parameters and returns the count of cells matching the criteria given:

count<tab><p1[=c1],p2[=c2],p3[=c3],p4[=c4]>

Where the desired search parameters are specified with an = followed by the parameter (case sensitive).

And of course the command <quit> quits the script.

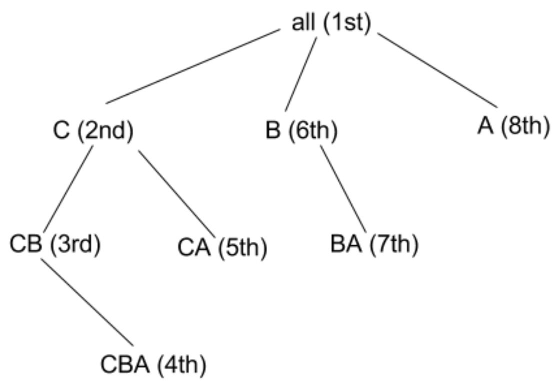


```
jakubklapacz@Jakubs-MacBook-Pro:~/cs/cs412/hw2$ python 2.py
Enter a command:
        cells   c,?,?,?

Cell count in cuboid: c,?,?,? = [48]

Enter a command:
        cells   s,?,?,?

Cell count in cuboid: s,?,?,? = [34]

Enter a command:
        cells   *,?,?,?

Cell count in cuboid: *,?,?,? = [23]

Enter a command:
        count   s=Illinois,*,?=3,?=moderate

Cell count with params: s=Illinois,*,?=3,?=moderate = [2]

Enter a command:
        count   c=Chicago,?=food,*,*

Cell count with params: c=Chicago,?=food,*,* = [2]

Enter a command:
        quit
```

3.

a. When we perform the multiway array aggregation we must choose the order of scanning data chunks because of the impact this choice makes on how much information the computer must keep in memory. We want the largest subarray in memory the shortest in order to be able to reuse the memory one chunk occupies with the next chunk. The next smallest subarray will need to be kept in memory slightly longer than the first subarray and so on. Because one array must remain in memory until we complete the calculation we want to keep the smallest subarray in memory because this ensures the least memory gets used throughout the calculation.

b. This is the trace tree of exploration for the CBA order:



c. If the minimum support were 4, the bottom up construction would compute 25 cells if exploring in the order A, B, C. We get this from the following work:

$$(*,*,*) = 15$$

$(a_0,*,*) = 6$
$(a_1,*,*) = 9$
$(a_2,*,*) = 0\ (prune)$

$(a_0, b_0,*) = 3$ (prune)
$(a_0, b_1,*) = 0$ (prune)
$(a_0, b_2,*) = 3$ (prune)

$(a_1, b_0,*) = 0$ (prune)
$(a_1, b_1,*) = 9$
$(a_1, b_2,*) = 0$ (prune)

$(a_1, b_0, c_0) = 3$ (prune)
$(a_1, b_1, c_1) = 3$ (prune)

$(a_1, b_2, c_2) = 3$ (prune)

$(a_0, *, c_0) = 2$ (prune)
$(a_0, *, c_1) = 2$ (prune)
$(a_0, *, c_2) = 2$ (prune)

$(a_1, *, c_0) = 3$ (prune)
$(a_1, *, c_1) = 3$ (prune)
$(a_1, *, c_2) = 3$ (prune)

$(*, b_0, *) = 3$ (prune)
$(*, b_1, *) = 9$
$(*, b_2, *) = 3$ (prune)

$(*, b_2, c_0) = 3$ (prune)
$(*, b_2, c_1) = 3$ (prune)
$(*, b_2, c_2) = 3$ (prune)

If we add all of the expansions we get to 25 cells.