Finding Frequent Itemsets: Improvements to A-Priori

Park-Chen-Yu Algorithm

Multistage Algorithm

Multi-Hash Algorithm

Approximate Algorithms

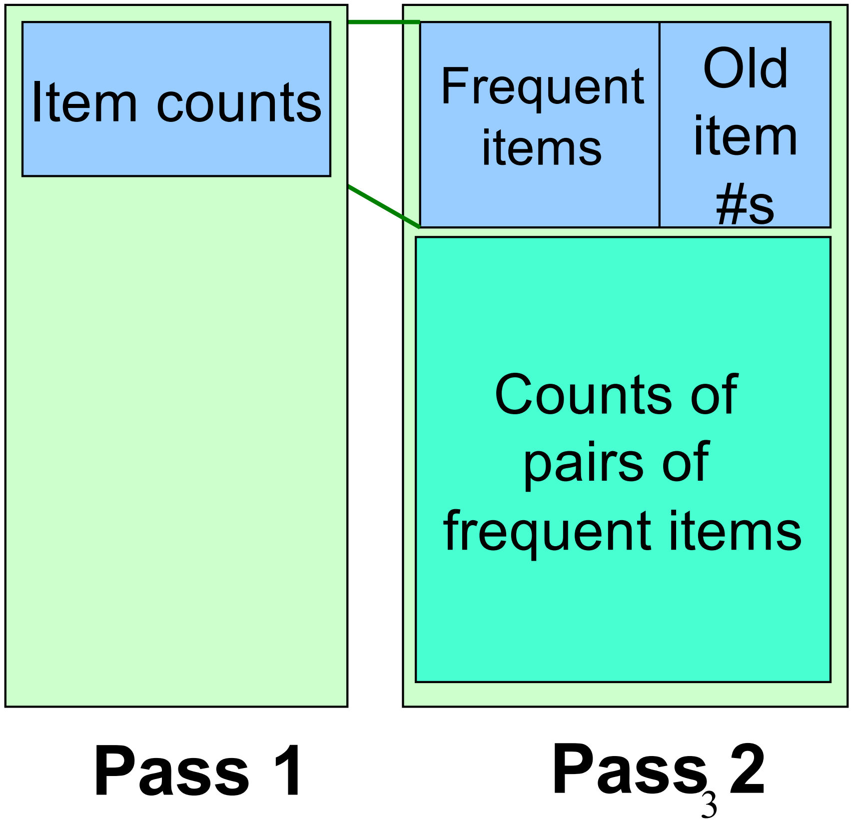
Thanks for source slides and material to:
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets
http://www.mmds.org

Park-Chen-Yu (PCY) Algorithm

PCY Algorithm – An Application of Hash-Filtering

- During Pass 1 of A-priori, most memory is idle
- Use that memory to keep counts of buckets into which pairs of items are hashed

A Priori Memory Usage:



PCY (Park-Chen-Yu) Algorithm

- **Observation:**
  In pass 1 of A-Priori, most memory is idle
  - We store only individual item counts
  - **Can we use the idle memory to reduce memory required in pass 2?**
- **Pass 1 of PCY:** In addition to item counts, maintain a hash table with as many buckets as fit in memory
  - Keep a **count** for each bucket into which **pairs** of items are hashed
    - **For each bucket just keep the count, not the actual pairs that hash to the bucket!**

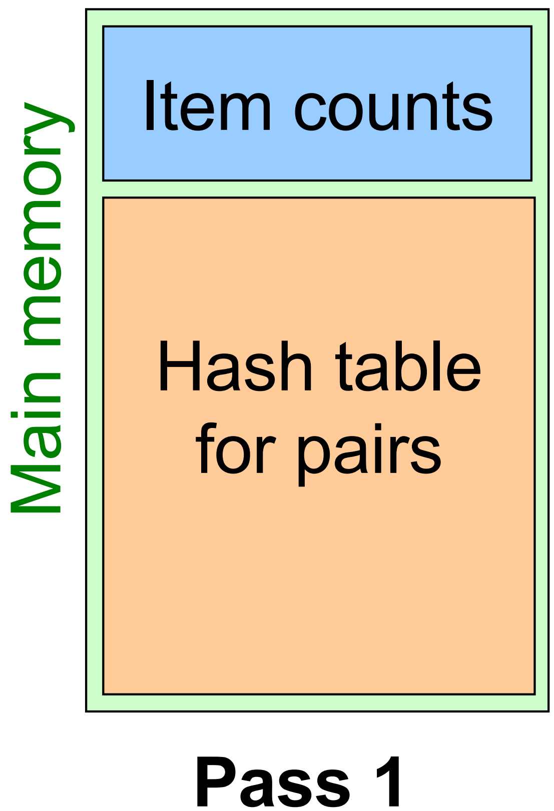
PCY Algorithm – First Pass

```
FOR (each basket) :
    FOR (each item in the basket) :
        add 1 to item's count;
    FOR (each pair of items) :
        hash the pair to a bucket;
        add 1 to the count for that bucket;
```

New in PCY

- **Few things to note:**
  - Pairs of items need to be generated from the input file;
  - We are not just interested in the presence of a pair, but we need to see whether it is present at least ***s*** (support) times

Main-Memory: Picture of PCY Pass 1



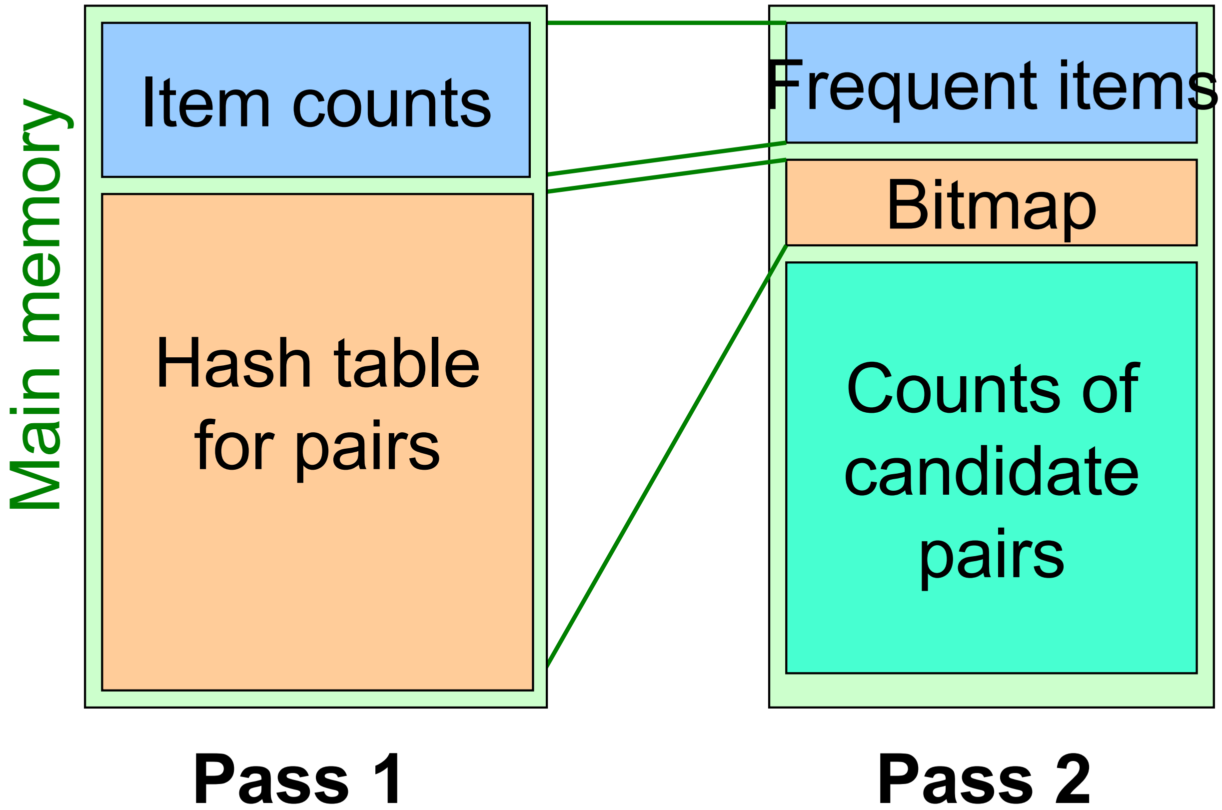
Observations about Buckets

- A bucket is *frequent* if its count is at least the support threshold s
- Observation: If a bucket contains a frequent pair, then the bucket is surely frequent
- However, even without any frequent pair, a bucket can still be frequent
  - So, we cannot use the hash to eliminate any member (pair) of a "frequent" bucket
- **But, for a bucket with total count less than *s*, none of its pairs can be frequent**
  - Pairs that hash to this bucket can be eliminated as candidates (even if the pair consists of 2 frequent items)
- **Pass 2:** Only count pairs that hash to frequent buckets

PCY Algorithm – Between Passes

- **Replace the buckets by a bit-vector:**
  - **1** means the bucket count exceeded the support s (call it a **frequent bucket**); **0** means it did not
- **4-byte integer counts are replaced by bits, so the bit-vector requires 1/32 of memory**
- As for A Priori, decide which items are frequent in first pass and list them for the second pass

Main-Memory: Picture of PCY



PCY Algorithm – Pass 2

- Count all pairs *{i, j}* that meet the conditions for being a **candidate pair**:
  1. Both *i* and *j* are frequent items
  2. The pair *{i, j}* hashes to a bucket whose bit in the bit vector is **1** (i.e., a **frequent bucket**)

- **Both conditions are necessary for the pair to have a chance of being frequent**

Example

Exercise 6.3.1 : Here is a collection of twelve baskets. Each contains three of the six items 1 through 6.

$$\begin{array}{cccc}
\{1,2,3\} & \{2,3,4\} & \{3,4,5\} & \{4,5,6\} \\
\{1,3,5\} & \{2,4,6\} & \{1,3,4\} & \{2,4,5\} \\
\{3,5,6\} & \{1,2,4\} & \{2,3,5\} & \{3,4,6\}
\end{array}$$

Hash functions: i+j mod 9

Main-Memory Details

- **Buckets require a few bytes each**
  - **Note:** we do not have to count past s
  - #buckets is *O(main-memory size)*
- On second pass, a table of (item, item, count) triples is essential
- We cannot use triangular matrix approach: **why?**
- Thus, hash table must eliminate approx. 2/3 of the candidate pairs for PCY to beat A-Priori

Why Can't We use a Triangular Matrix on Phase 2 of PCY?

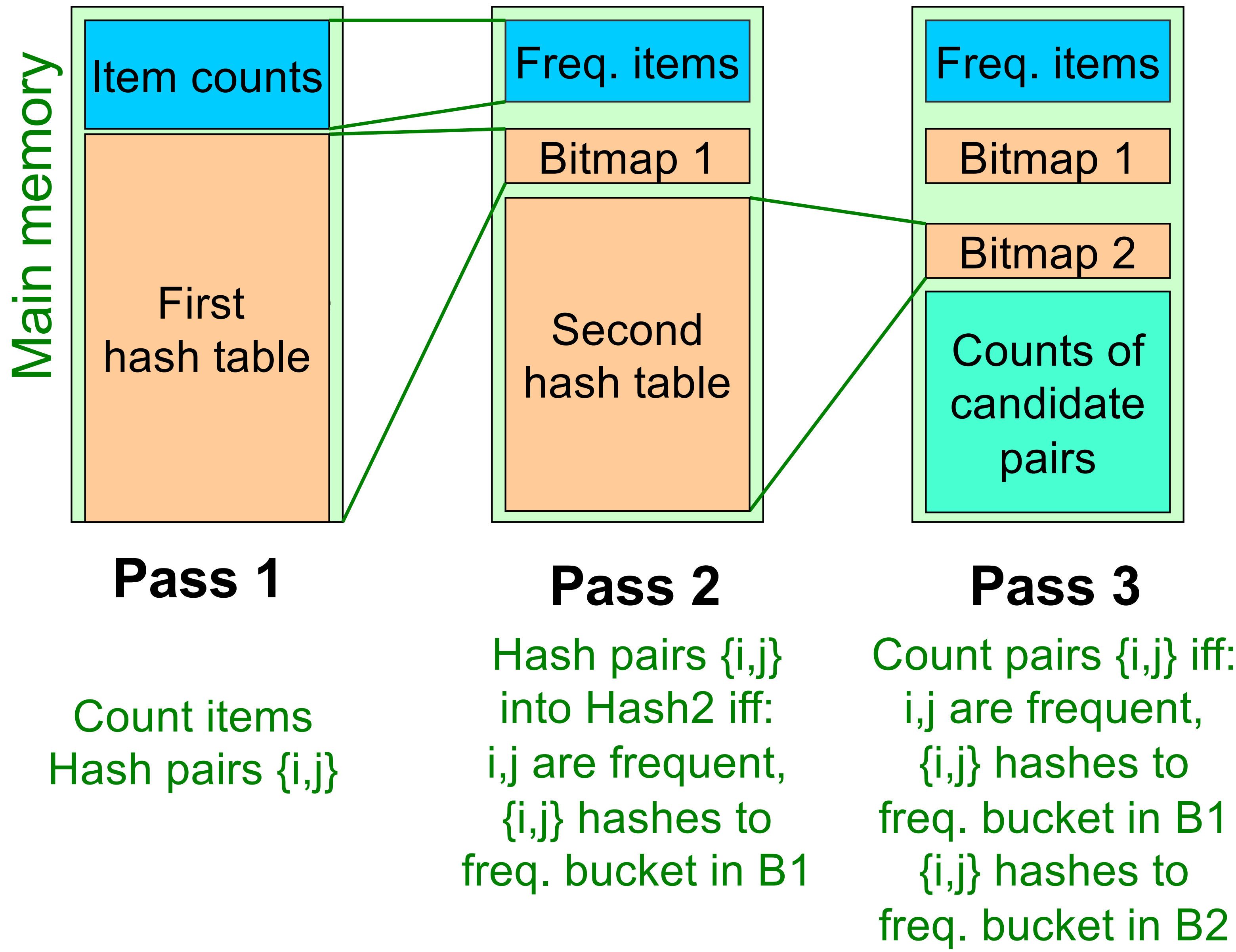
- Recall: in A Priori, the frequent items could be renumbered in Pass 2 from *1* to *m*
- Can't do that for PCY
- Pairs of frequent items that PCY lets us avoid counting are placed randomly within the triangular matrix
  - Pairs that happen to hash to an infrequent bucket on first pass
  - No known way of compacting matrix to avoid leaving space for uncounted pairs
- Must use triples method

Multi-Stage Algorithm

Refinement: Multistage Algorithm

- **Limit the number of candidates to be counted**
  - **Remember:** Memory is the bottleneck
  - Still need to generate all the itemsets but we only want to count/keep track of the ones that are frequent
- **Key idea:** After Pass 1 of PCY, **rehash only those pairs** that qualify for Pass 2 of PCY
  - ***i*** and ***j*** are frequent, and
  - ***{i, j}*** hashes to a frequent bucket from **Pass 1**
- On middle pass, fewer pairs contribute to buckets, so fewer ***false positives***
- **Requires 3 passes over the data**

Main-Memory: Multistage



Multistage – Pass 3

- **Count only those pairs $\{i, j\}$ that satisfy these candidate pair conditions:**
  1. Both i and j are frequent items
  2. Using the first hash function, the pair hashes to a bucket whose bit in the first bit-vector is **1**
  3. Using the second hash function, the pair hashes to a bucket whose bit in the second bit-vector is **1**

Important Points

1. **The two hash functions have to be independent**
2. **We need to check both hashes on the third pass**
   - If not, we would end up counting pairs of frequent items that hashed first to an infrequent bucket but happened to hash second to a frequent bucket
   - Would be a false positive

Example

Exercise 6.3.1: Here is a collection of twelve baskets. Each contains three of the six items 1 through 6.

$$\begin{array}{cccc} \{1,2,3\} & \{2,3,4\} & \{3,4,5\} & \{4,5,6\} \\ \{1,3,5\} & \{2,4,6\} & \{1,3,4\} & \{2,4,5\} \\ \{3,5,6\} & \{1,2,4\} & \{2,3,5\} & \{3,4,6\} \end{array}$$

Hash functions: i+j mod 9 and i+j mod 5

Key Observation

- Can insert any number of hash passes between first and last stage
  - Each one uses <u>an independent hash function</u>
  - Eventually, all memory would be consumed by bitmaps, no memory left for counts
  - Cost is another pass of reading the input data
- **The truly frequent pairs will always hash to a frequent bucket**
- So we will count the frequent pairs no matter how many hash functions we use

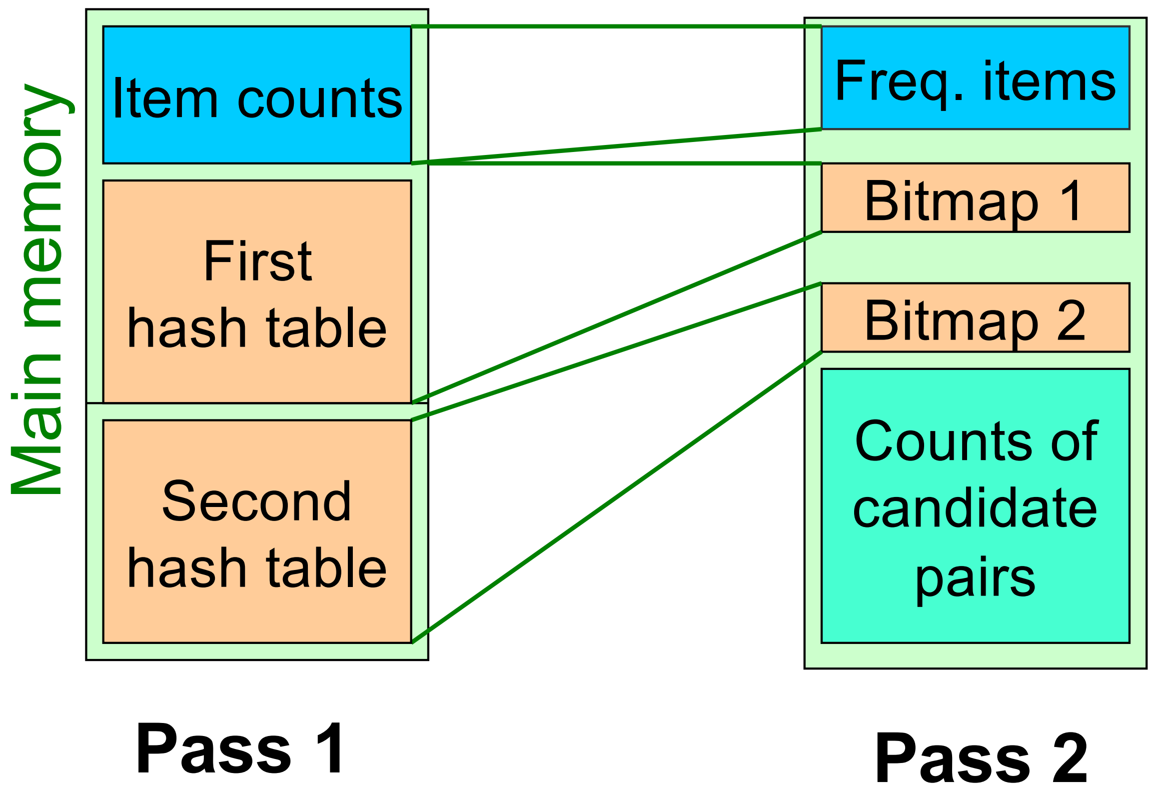
Multi-Hash Algorithm

Thanks for source slides and material to:
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets
http://www.mmds.org

Refinement: Multihash

- **Key idea:** Use several independent hash tables on the first pass
- **Risk:** Halving the number of buckets doubles the average count
  - We have to be sure most buckets will still not reach count s
- If so, we can get a benefit like multistage, but in only 2 passes

Main-Memory: Multihash



Multihash – Pass 2

- Same condition as Multistage but checked in second pass
- **Count only those pairs $\{i, j\}$ that satisfy these candidate pair conditions:**
  1. Both i and j are frequent items
  2. Using the first hash function, the pair hashes to a bucket whose bit in the first bit-vector is **1**
  3. Using the second hash function, the pair hashes to a bucket whose bit in the second bit-vector is **1**

Example

Exercise 6.3.1 : Here is a collection of twelve baskets. Each contains three of the six items 1 through 6.

$$\begin{array}{cccc} \{1,2,3\} & \{2,3,4\} & \{3,4,5\} & \{4,5,6\} \\ \{1,3,5\} & \{2,4,6\} & \{1,3,4\} & \{2,4,5\} \\ \{3,5,6\} & \{1,2,4\} & \{2,3,5\} & \{3,4,6\} \end{array}$$

Hash functions: i+j mod 5 and i+j mod 4

PCY: Extensions

- Either **multistage** or **multihash** can use more than two hash functions

- In **multistage**, there is a point of diminishing returns, since the bit-vectors eventually consume all of main memory

- For **multihash**, the bit-vectors occupy exactly what one PCY bitmap does, but too many hash functions makes all counts $\geq s$

Next: Limited Pass Algorithms

- There are many applications where it is sufficient to find most but not all frequent itemsets
- Algorithms to find these in at most 2 passes