FP-Growth algorithm

Implementation in Python

Group 1

Eero Anttilainen

Erik Kähkönen

Unto Kuuranne

Sergio Rubio Martín

Introduction

FP-Growth algorithm allows frequent itemsets mining avoiding candidate itemset generation using a compressed representation of the transaction database called **FP-tree**.

2 main steps:

1. FP-tree **construction**.
2. Frequent itemsets **extraction**.

FP-Growth Algorithm

First step:

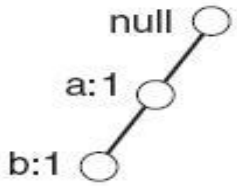
Constructs FP-tree with two passes over the dataset:

- **First pass:** Calculate **support** for each item and sort them in **decreasing** order of support.
- **Second pass:** Construct the tree by adding **transactions** as **branches**.

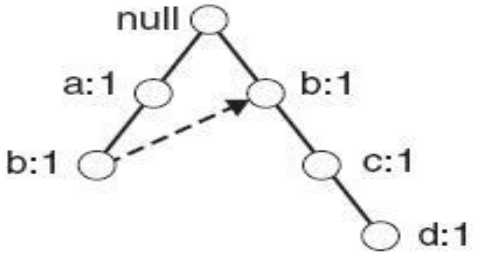
FP-tree construction example:

Transaction Data Set

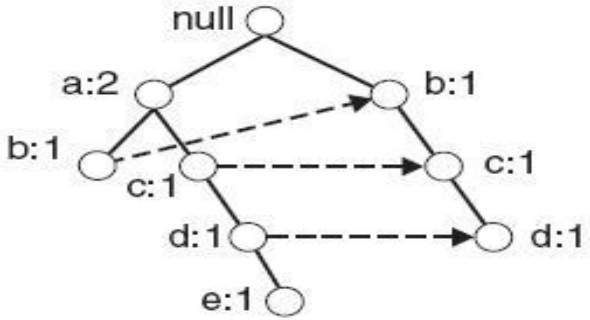
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



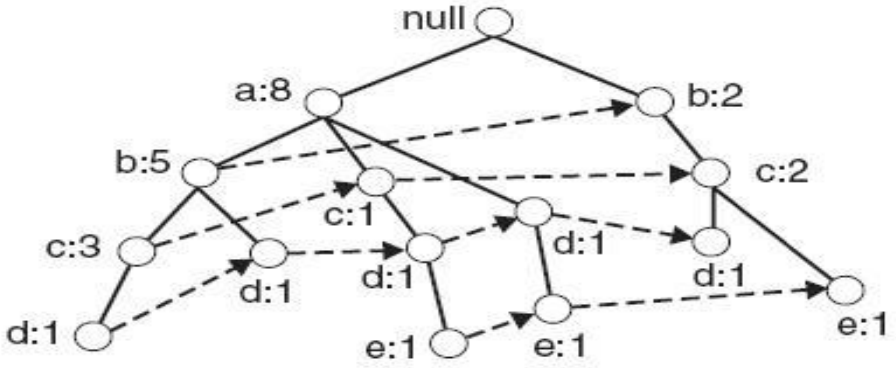
(i) After reading TID=1



(ii) After reading TID=2



(iii) After reading TID=3



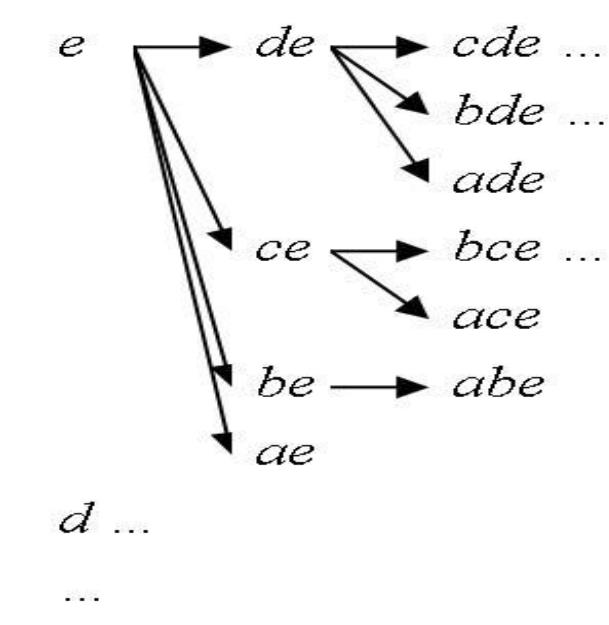
(iv) After reading TID=10

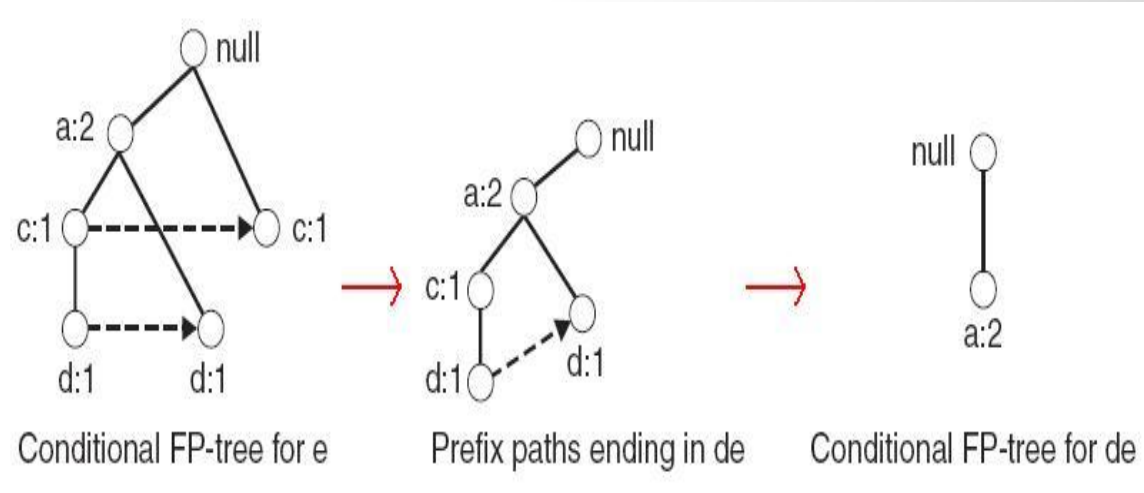
FP-Growth Algorithm

Second step:

Extracts frequent itemsets from the tree.

- Divides the problem into subproblems for each **suffix itemset**.
- **Divide and conquer** approach.

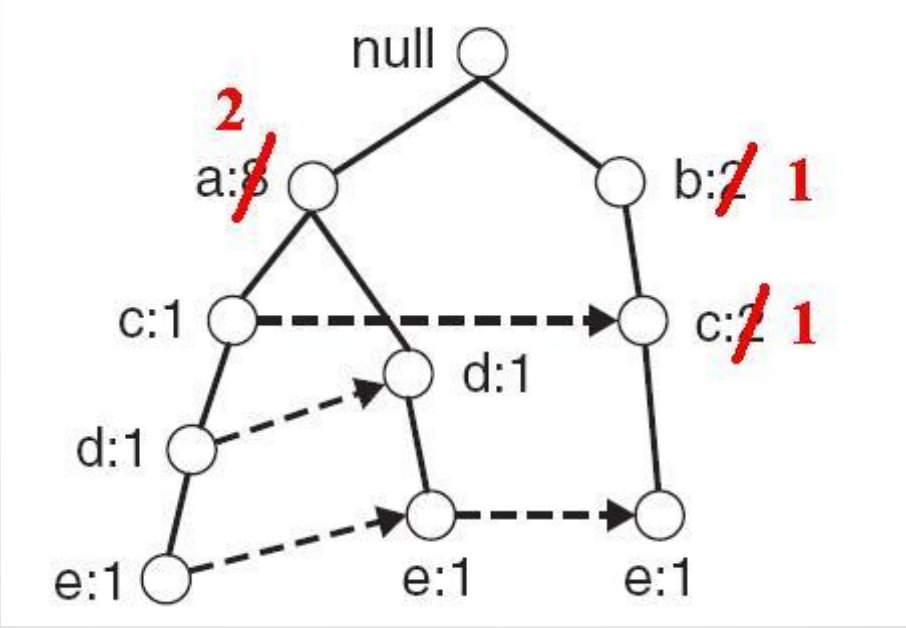




Difference Between Our Implementation and the Standard Version

The main difference is in **how conditional subtrees are generated**:

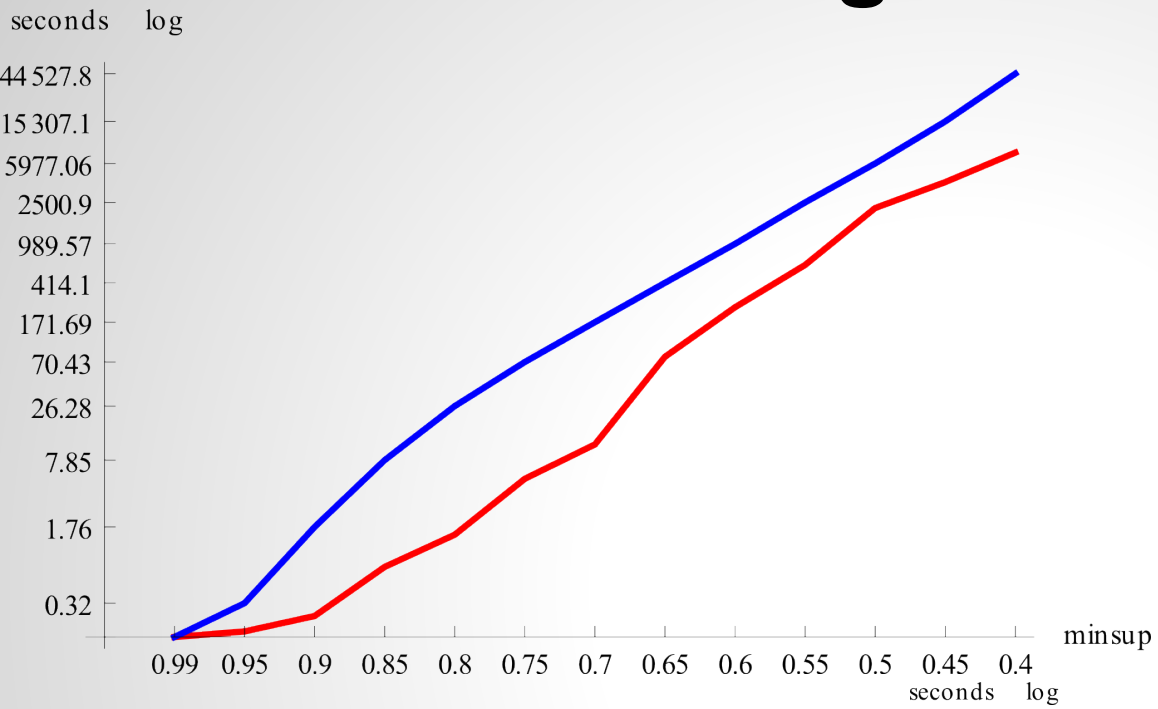
- The standard version copies parts of the original tree, then **updates support counts**.
- Our version **sums branches** with the **correct conditional support count**.



Testing on a dataset

- **Test data: chess.dat**
- **Run on:** Paniikki CRoom **identical** computers
- **Compared: time** and **memory** usage of our **FP-Growth** and **Apriori**

Results of testing



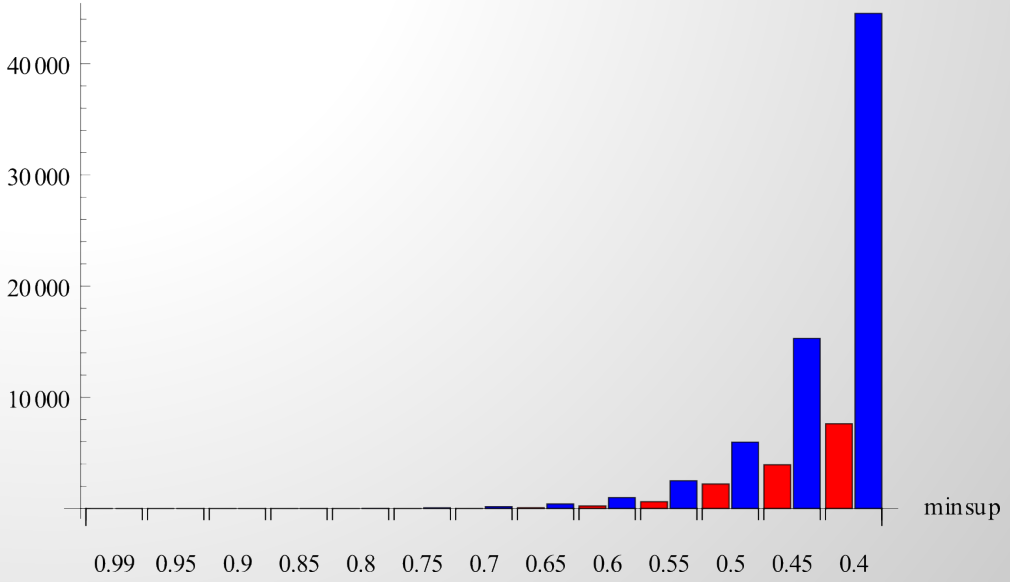
FP-Growth

Apriori

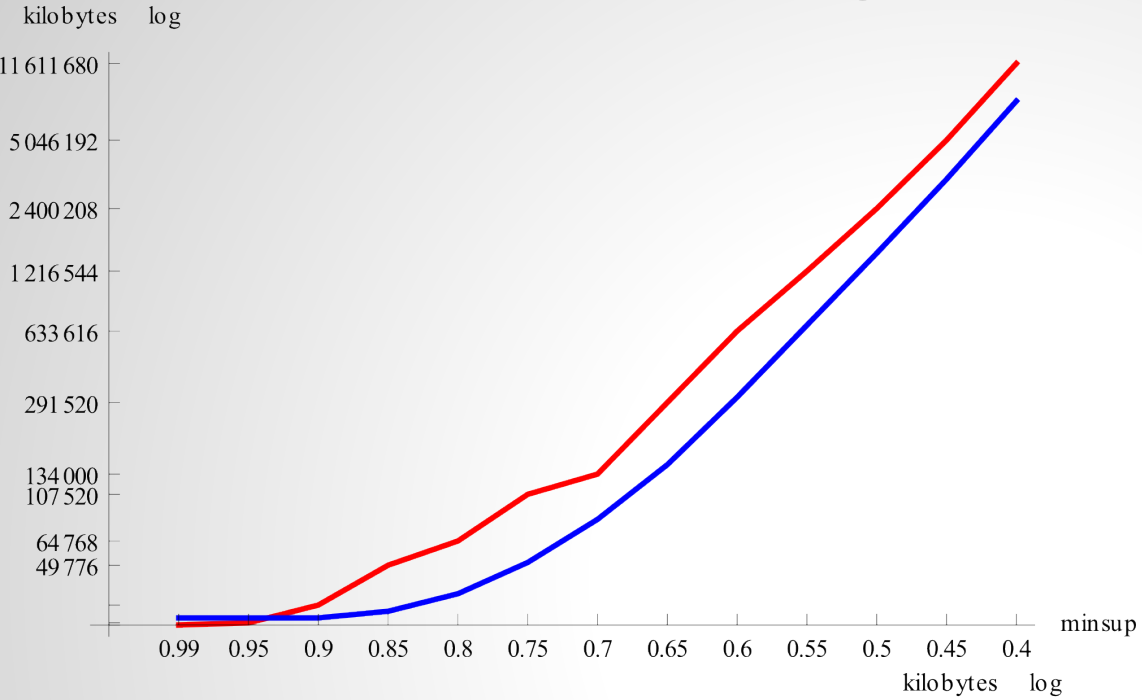
Runtime Comparison

y = time in seconds

x = min support



Results of testing



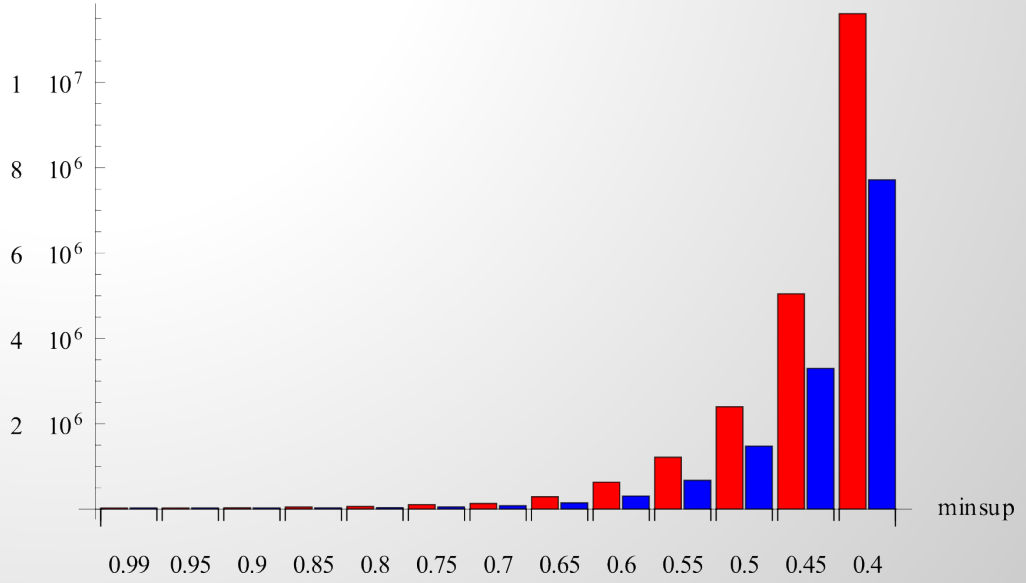
FP-Growth

Apriori

Max memory usage comparison

y = memory in kB

x = min support



Strength and weaknesses

Strengths:

- Only **2 passes** over data-set
- No **candidate generation**
- Faster than **Apriori**
- In most cases **compresses** data-set

Weaknesses:

- FP-tree is **expensive** to build
- FP-tree may not fit in **memory**
- In worst case data may be **larger** than original

Conclusions

- FP-growth algorithm is significantly **more efficient** than Apriori algorithm.
- FP-tree takes time to build, but then the mining is **fast**.
- The structure allows us to **save space** comparing with the original database.

Questions?