Aprendizado Descritivo

Aula 04 – FP-Growth

Professor Renato Vimieiro

DCC/ICEx/UFMG

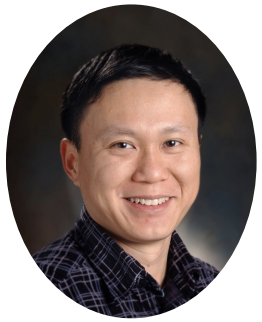
Introdução

- Nessa aula, veremos outro algoritmo que usa projeções para reduzir o número de passadas para computação dos conjuntos de itens frequentes
- O algoritmo FP-Growth (Frequent Pattern Growth) adota uma estratégia dividir-e-conquistar para reduzir o custo computacional
- Ele, ao contrário do Apriori, não se baseia na geração de candidatos
- Os padrões são construídos ao longo do processamento em profundidade
- Esse algoritmo é, talvez, o algoritmo sequencial mais eficiente para busca de conjuntos de itens frequentes

FP-Growth

- O FP-Growth foi proposto em 2000 por Jiawei Han, Jian Pei e Yiwen Yin
- O algoritmo atacou dois problemas presentes nas abordagens iniciais:
  1. Repetidas passadas sobre a base de dados; e
  2. Geração de candidatos
- O primeiro problema, como já discutimos, é crítico pelo custo computacional inerente à leitura em memória secundária
- O segundo problema está relacionado à geração de candidatos desnecessários
  - Muitos são descartados pela propriedade do Apriori

Jiawei Han
Uni of Illinois Urbana-Champaign



Jian Pei
Duke University

FP-Tree

- O FP-Growth possui algumas similaridades ao Eclat:
  - Ambos adotam a estratégia de busca em profundidade
  - Ambos adotam projeções dos dados com o intuito de trazê-los para memória principal e reduzir o custo computacional
- O FP-Growth, no entanto, usa uma estrutura de dados diferente para suportar a busca pelos padrões
  - Uma árvore de prefixos chamada FP-Tree
- A busca pelos padrões se dá inteiramente através da árvore sem a necessidade de se voltar à base de dados
- Dessa forma, a primeira tarefa do algoritmo é construir essa estrutura

FP-Tree

- A construção da FP-Tree ocorre em duas fases
- Primeiro, o algoritmo varre a base de dados para computar a frequência individual de cada item
  - Itens infrequentes são descartados, uma vez que não podem formar padrões frequentes
- Segundo, o algoritmo percorre novamente a base processando as transações ordenadas pela frequência dos itens
  - Os itens nas transações são ordenados em ordem decrescente de frequência e os infrequentes são filtrados
- As transações são então inseridas na árvore enquanto processadas
  - Itens são nós da árvore
  - Cada nó armazena um item e sua frequência (número de transações que o contém)

FP-Tree

- Para facilitar a busca pelos padrões, a árvore é equipada com uma estrutura adicional para localizar a ocorrência dos itens e sua frequência
- Exemplo

TID	Muesli (a)	Oats (b)	Milk (c)	Yoghurt (d)	Biscuits (e)	Tea (f)
1	1	0	1	1	0	1
2	0	1	1	0	0	0
3	0	0	1	0	1	1
4	1	0	0	1	0	0
5	0	1	1	0	0	1
6	1	0	1	0	0	1

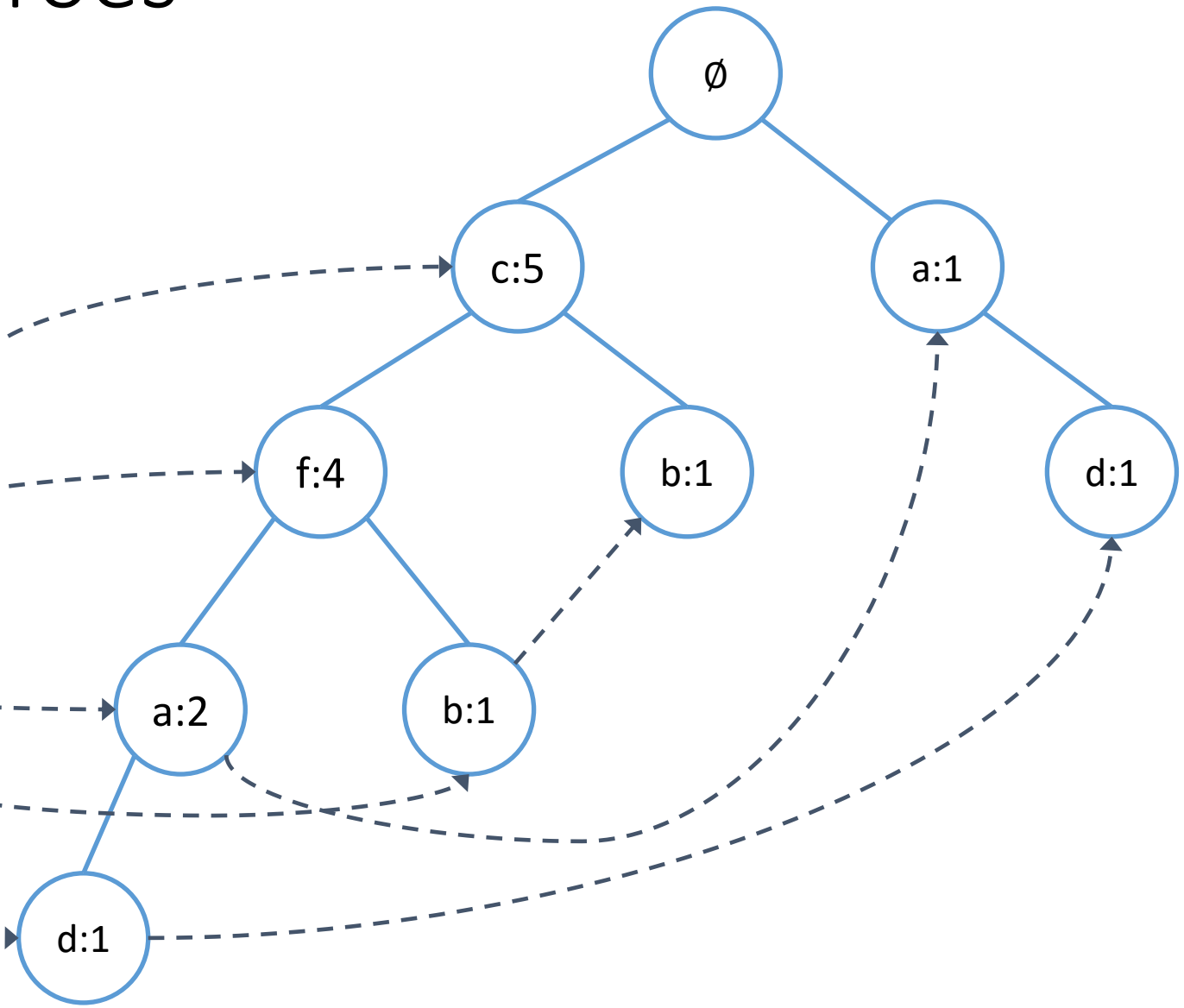
Mineração dos padrões

- A mineração dos padrões se inicia uma vez que a FP-Tree tenha sido construída
- A construção agora ocorre aumentando-se prefixos dos padrões em ordem crescente de suporte
- As transações que satisfaçam (contém) o padrão sendo construído são projetadas em uma nova árvore
  - Itens podem se tornar infrequentes nessa nova base e são descartados
- Os padrões encontrados nessa nova árvore devem incluir o prefixo que a gerou
- O algoritmo segue com as extensões recursivamente até que um único ramo seja obtido
  - Se a árvore possui um único ramo, os padrões obteníveis são todas as combinações dos nós

Mineração dos padrões

- Exemplo

Item	Freq	Link
c	5	
f	4	
a	3	
b	2	
d	2	



Questões de implementação

- E se a FP-tree não couber na memória?
  - A solução é particionar/projetar a base de dados em memória secundária antes de iniciar a construção da árvore
- Como construir a FP-tree de forma eficiente?
  - Solução proposta por Christian Borgelt otimiza memória e tempo
  - Representação básica dos dados: lista de vetores de inteiros
  - Dados (projeções) são carregados inteiramente para memória
  - Lista é seccionada com base no k-ésimo item; um nó é criado para cada seção
  - Nós têm tamanho fixo (20 bytes em 32bits; 40 em 64bits)
    - 1x identificador de item
    - 1x contador de frequência
    - 1x ponteiro para nó pai
    - 1x ponteiro para próxima ocorrência do item
    - 1x ponteiro para nó auxiliar

Christian Borgelt
Universidade Paris Lodon
(Áustria)

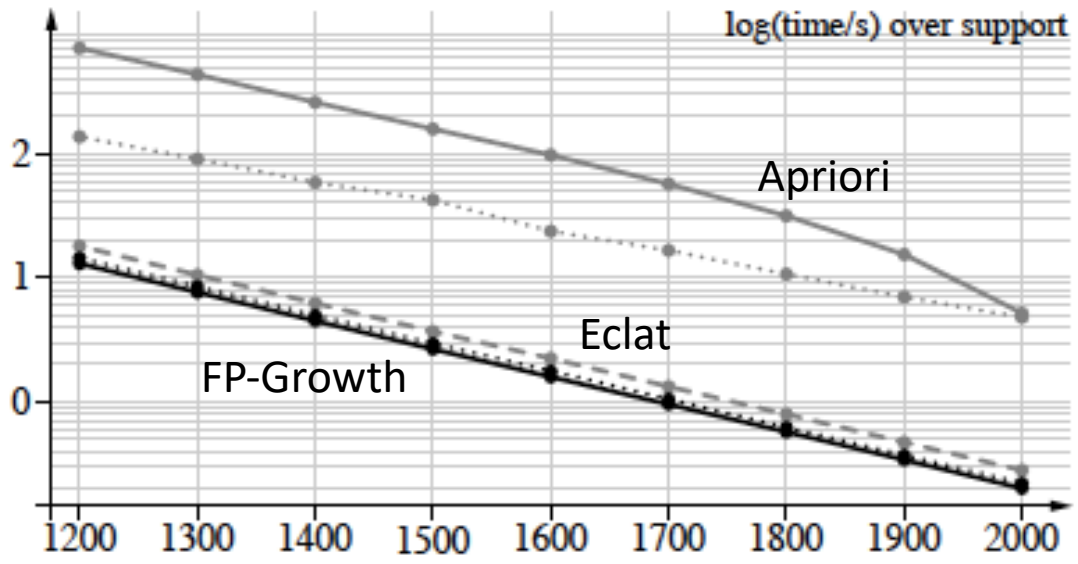
Questões de implementação

- Implementação tradicional, conforme descrição do algoritmo, evita carregar dados para memória
  - Porém, nós terão tamanho variável ou desperdiçam memória (ponteiros para filhos que nunca ocorrem)
  - Melhora gerenciamento de memória; grandes blocos podem ser alocados de uma vez e gerenciados internamente
  - Além disso, ponteiros para pais são mais úteis que ponteiros para filhos durante execução

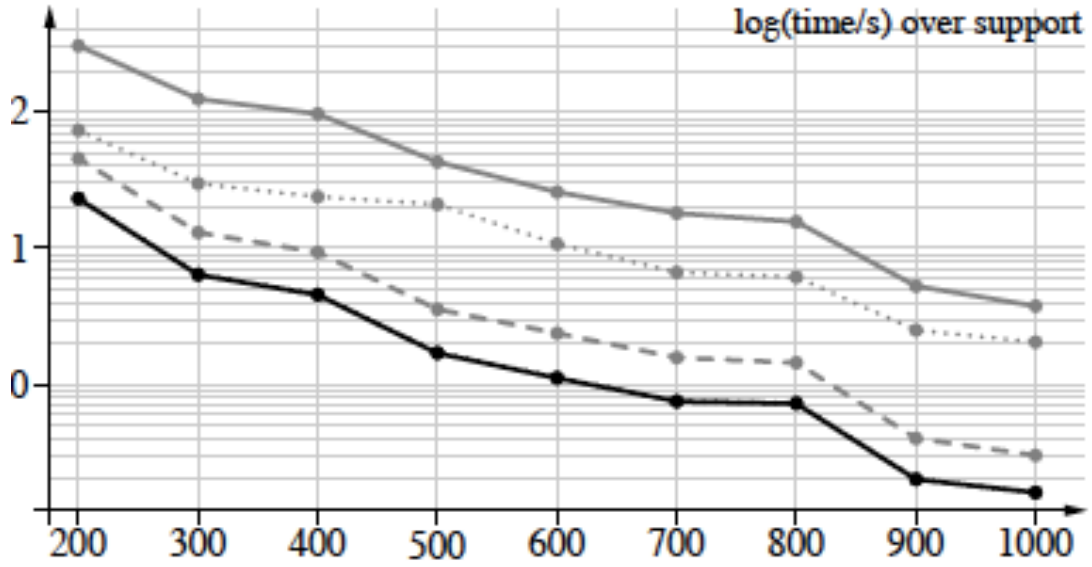
Questões de implementação

- Projeções são executadas com dois laços
  - Um laço externo percorre o nível mais baixo (elemento condicionante da projeção)
  - Laço interno percorre a os ramos originários do nó folha
- A nova árvore é construída como uma ‘sombra’ da original
  - Nós são duplicados conforme são visitados (ponteiro auxiliar mantém elo de ligação entre original e cópia para atualizações necessárias durante construção)
  - Frequência do nó folha é propagada para cima
- A sombra é destacada da árvore original em uma segunda passada pelos nós
- Nós infrequentes podem ser removidos e nós com mesmo rótulo mesclados

Questões de implementação



Chess



Mushroom

Figuras retiradas de Borgelt, C. An Implementation of the FP-growth Algorithm.

Leitura

- Seção 6.6 Intro to Data Mining
- Seção 8.2.3 Zaki e Meira
- Borgelt, C. (2005) An Implementation of the FP-growth Algorithm. https://borgelt.net/papers/fpgrowth.pdf

Aprendizado Descritivo

Aula 04 – FP-Growth

Professor Renato Vimieiro

DCC/ICEx/UFMG