Relatório

Selton Miranda Rolim - GRR20235688

1 Estruturas de dados

```c
typedef bool attributes[ALPHABET_SIZE];

typedef struct fd_t
{
  attributes lhs;
  attributes rhs;
} FD;

typedef struct fd_list_t
{
  FD*    fds;
  size_t size;
} FDList;

typedef struct Node
{
  attributes   attrs;
  struct Node* next;
} Node;

typedef struct Queue
{
  struct Node* front;
  struct Node* back;
} Queue;
```

A principal estrutura de dados utilizada neste trabalho é o conjunto de atributos **attributes**, representado por um vetor booleano fixo com 26 posições [1]. Como trabalhamos apenas com atributos de A à Z, cada posição do vetor indica se um atributo pertence ou não ao conjunto.

A estrutura **FD** representa uma dependência funcional, contendo dois conjuntos de atributos: um no lado esquerdo (*lhs*) e outro no lado direito (*rhs*).

A estrutura **FDList** representa uma lista de dependências funcionais, utilizada pelas funções que implementam os algoritmos deste trabalho.

[1] Depois que eu terminei o trabalho eu percebi que era possível fazer com um bitset.

As estruturas **Queue** e **Node** são usadas internamente pelos algoritmos de busca de chaves candidatas. Elas não fazem parte da interface pública, pois servem apenas para a lógica interna do algoritmo.

2 Análise de Complexidades

2.1 Fecho

```
1 void closure(FDList* list, attributes closureSet);
```

O cálculo do fecho utiliza um processo iterativo que aplica repetidamente as dependências funcionais até que nenhum novo atributo possa ser acrescentado ao conjunto. Sua complexidade torna-se $O(|F| \cdot |U|^2)$.

2.2 Cobertura Mínima

```
1 void mincover(FDList* list);
```

O algoritmo do mincover aplica três fases: desmembramento das dependências ($O(|F| \cdot |U|)$), remoção de atributos estranhos no *lhs* ($O(|F|^2 \cdot |U|^3)$) e eliminação de dependências redundantes ($O(|F|^2 \cdot |U|^2)$). A etapa mais custosa é a verificação de atributos estranhos no *lhs*, que exige múltiplos cálculos de fecho. Portanto a complexidade é $O(|F|^2 \cdot |U|^3)$.

2.3 Chaves candidatas

```
1 attributes* candidateKeys(FDList* list, int* keys_count);
```

O algoritmo de geração das chaves candidatas [2] utiliza uma busca em largura iniciada pelos atributos essenciais. Para cada conjunto visitado, é calculado o fecho dos atributos, verificando se o conjunto é uma superchave e se é chave mínima. No pior caso, a função pode explorar todos os subconjuntos de atributos ($2^{|U|}$). Como cada subconjunto requer cálculos de fecho, o custo total é $O(2^{|U|} \cdot |F| \cdot |U|^2)$.

2.4 Verificação de BCNF e 3FN

```
1 void normalform(FDList* list);
```

A função executa três fases: redução das dependências (mincover), geração das chaves candidatas e verificação de violações de BCNF/3NF. A fase mais custosa é a geração das chaves candidatas. Portanto o custo da função é $O(2^{|U|} \cdot |F| \cdot |U|^2)$.

[2] Não vou detalhar muito, senão o relatório terá muitas páginas.

3 Estudo de caso

3.1 Caso 1: Violação de BCNF e 3FN

$$U = \{A, B, C, D, E\}, \qquad F = \{A \to BC,\ C \to D,\ BD \to E\}$$

Saída do programa:

```
BCNF: VIOLATIONS
3NF: VIOLATIONS
VIOLATION BCNF: C->D (C not superkey)
VIOLATION BCNF: BD->E (BD not superkey)
VIOLATION 3NF: C->D (D not prime, C not superkey)
VIOLATION 3NF: BD->E (E not prime, BD not superkey)
```

A única chave candidata da relação é **A**, logo apenas A é atributo primo. As dependências $C \to D$ e $BD \to E$ têm LHS que não é superchave e RHS contendo atributos não primos, violando tanto **BCNF** quanto **3FN**.

3.2 Caso 2: Violação somente de BCNF

$$U = \{A, B, C, D, E, F, G, H\}, \qquad F = \{ABH \to C,\ AC \to H,\ AG \to A,\ BG \to A,\ AD \to G\}$$

Saída do programa:

```
BCNF: VIOLATIONS
VIOLATION BCNF: ABH->C (ABH not superkey)
VIOLATION BCNF: AC->H (AC not superkey)
VIOLATION BCNF: AD->G (AD not superkey)
VIOLATION BCNF: BG->A (BG not superkey)
```

As chaves candidatas encontradas são:

$$\{ABCDEF\},\ \{ABDEFH\},\ \{BCDEFG\},\ \{BDEFGH\}$$

Todos os atributos da relação pertencem a alguma chave candidata, logo todos são primos. Portanto, apesar das dependências que não possuem superchave à esquerda violarem **BCNF**, nenhuma delas viola **3FN**.

3.3 Caso 3: Sem violação de BCNF nem 3FN

$$U = \{P, Q, R, S, T\}, \qquad F = \{P \to Q,\ PQ \to R,\ PR \to S,\ PS \to T\}$$

A única chave candidata é **P**. Como P determina toda a relação, os conjuntos PQ, PR e PS são superchaves (embora não mínimas).

Assim, todas as dependências possuem superchaves no lado esquerdo, garantindo que a relação está tanto em **BCNF** quanto **3FN**.