Teoría de Bases de Datos

Trabajo práctico 4

Normalización

Román Castellarín
Gianfranco Paoloni
Juan Ignacio Suarez

4 de diciembre de 2018

Índice

1. Introducción

En este documento se describe la definición e implementación de varios algoritmos sobre conjuntos de atributos y de dependencias funcionales. Para realizar el testeo de estos algoritmos se proponen 5 sets de prueba:

```
Set 1
R = {A, B, C, D}
F = {A->B, CB->A, B->AD}
```

```
Set 2
R = {A, B, C, D, E, F}
F = {AB->C, BD->EF}
```

```
Set 3
R = {A, B, C, D, E, F, G, H, I, J}
F = {AB->C, BD->EF, AD->GH, A->I, H->J}
A = {B, D}
```

```
Set 4
R = {A, B, C, D, E, F, G, H}
F = {A->BC, C->D, D->G, H->E, E->A, E->H}
A = {A, C}
```

```
Set 5
R = {A, B, C, D, E, F, G}
F = {A->G, A->F, B->E, C->D, E->A, D->B, GF->C}
A = {F, G}
```

Cada uno de estos sets se proveen bajo la sintaxis particular de nuestro programa en el en directorio `Examples`, en los archivos `set1.txt`, `set2.txt`, `set3.txt`, `set4.txt` y `set5.txt`.

1.1. Estructura del programa

El programa ha sido implementado en `C++11`, incluye seis archivos principales: `main.cpp`, `funcdependency.h`, `funcdependency.cpp`, `set_closure.cpp`, `attr_closure.cpp` y `candidate_keys.cpp`.

- `main.cpp`: mini-parser de archivos de entrada y llamadas correspondientes a las funciones requeridas en el trabajo práctico.
- `funcdependency.h`: definiciones de conjuntos de atributos y dependencias, así también como operadores y funciones requeridas para la implementación general.

- `funcdependency.cpp`: implementación de las definiciones generales previamente mencionadas.
- `set_closure.cpp`: implementación para encontrar la clausura de un conjunto de dependencias funcionales.
- `attr_closure.cpp`: similar al anterior aplicado a la clausura de atributos.
- `candidate_keys.cpp`: implementación para determinar todas las posibles claves candidatas de un conjunto de atributos y las dependencias funcionales correspondientes al conjunto.

1.2. Compilación

Para compilar el programa basta con compilar con `g++` (versión 2011 o más moderna) todos los archivos mencionados en la sección anterior. Dado que en la implementación probamos casos sobre muchos subconjuntos es ideal compilar con optimización de tipo 3 (-O3). Además, crearemos un ejecutable con el nombre `fd` para poder tener una consistencia con los ejemplos siguientes. Todo esto se puede hacer en una línea con el comando

```
> g++ -std=c++11 -O3 *.cpp -o fd
```

Si tuvimos éxito debemos tener un archivo ejecutable llamado fd en el mismo directorio.

1.3. Ejecución

Suponiendo que el ejecutable compilado se llama `fd`, la ejecución es tan simple como correr

```
> ./fd archivo_de_entrada
```

será fd.exe en vez de ./fd si estamos en Windows

Dado que calcular el conjunto de cierre de dependencias funcionales resulta una operación muy costosa como veremos luego, hemos incluído un comando opcional `-countFD` si se desea contar la cardinalidad de dicho conjunto. La ejecución con esta opción sería

```
> ./fd archivo_de_entrada -countFD
```

2. Cierre de conjuntos de dependencias funcionales F^+

2.1. Algoritmo

Sea F un conjunto de dependencias funcionales definido sobre R. El cierre de F, denotado por F^+, es el conjunto de todas las dependencias funcionales que F implica lógicamente. El siguiente algoritmo escrito en pseudocódigo basado en los axiomas de Armstrong nos permite calcular F^+:

```
resultado := F;

for each alfa in P(R) do
    aplicar las reglas de reflexividad a alfa;
    añadir las nuevas DF obtenidas a resultado;

while (hay cambios en resultado) do
    for each DF f in resultado do
        aplicar las reglas de aumentatividad a f;
        añadir las nuevas DF obtenidas a resultado;

for each DF f1 in resultado do
    for each DF f2 in resultado do
        if f1 y f2 pueden combinarse por transitividad then
            añadir la nueva DF a resultado;

return resultado;
```

2.2. Implementación

La implementación de este algoritmo se puede encontrar en el archivo `set_closure.cpp`

3. Cierre de conjuntos de atributos α^+

3.1. Algoritmo

Sea α un conjunto de atributos. Al conjunto de todos los atributos determinados funcionalmente por α bajo un conjunto F de dependencias funcionales se lo denomina cierre de α bajo F y se denota mediante α^+. A continuación se muestra un algoritmo escrito en pseudocódigo para calcular α^+ a partir de F y α:

```
resultado := alfa;

while (hay cambios en resultado) do
    for each dependencia funcional b->c in F do
        if b está contenido en resultado then
            resultado := resultado UNION c;

return resultado;
```

3.2. Implementación

La implementación de este algoritmo se puede encontrar en el archivo `attr_closure.cpp`

4. Conjunto de claves candidatas S_K

4.1. Algoritmo

Dada una relación R y un conjunto de dependencias funcionales F queremos encontrar el conjunto de todas sus claves candidatas S_K. Recordemos la definición de clave candidata:

Una clave candidata K de una relación R es una superclave minimal para dicha relación, es decir, es un conjunto de atributos K que cumple:

1. K es superclave, es decir que $K^+ = R$
2. K es minimal, es decir que $\nexists Y \subset K \mid Y^+ = R$. Notar que la inclusión es estricta, pues Y es un subconjunto propio de K.

Dicho esto podemos entonces escribir un algoritmo (en pseudocódigo) para calcular S_K a partir de R y F:

```
resultado := vacío;

for each X in P(R) do
    if X+ == R then
        esCandidato := true;

        for each Y in P(X) - X do
            if Y+ == R then
                esCandidato := false;

        if esCandidato == true then
            añadir X a resultado

return resultado;
```

4.2. Implementación

La implementación de este algoritmo se puede encontrar en el archivo `candidate_keys.cpp`

5. Salidas del programa en los sets de prueba

Para verificar la correcta ejecución de los algoritmos, a continuación se presenta la ejecución del programa en cada uno de los sets de prueba descritos en la introducción.

5.1. Set 1

Corriendo el programa sobre el set 1 se obtiene

```
> ./fd Examples/set1.txt -countFD
A+ closure: empty
F+ size: 137
Candidate keys:
AC
BC
```

Por lo que la cardinalidad del conjunto F^+ es 137, y el conjunto de claves candidatas S_K resulta $\{AC, BC\}$.

5.2. Set 2

Corriendo el programa sobre el set 2 se obtiene

```
> ./fd Examples/set2.txt -countFD
A+ closure: empty
F+ size: 1081
Candidate keys:
ABD
```

Por lo que la cardinalidad del conjunto F^+ es 1081, y el conjunto de claves candidatas S_K resulta $\{ABD\}$.

5.3. Set 3

Corriendo el programa sobre el set 3 se obtiene

```
> ./fd Examples/set3.txt
A+ closure: BDEF
Candidate keys:
ABD
```

Por lo que la clausura de atributos del conjunto A resulta $A^+ = \{BDEF\}$, y el conjunto de claves candidatas S_K resulta $\{ABD\}$

5.4. Set 4

Corriendo el programa sobre el set 4 se obtiene

```
> ./fd Examples/set4.txt
A+ closure: ABCDG
Candidate keys:
EF
FH
```

Por lo que la clausura de atributos del conjunto A resulta $A^+ = \{ABCDG\}$, y el conjunto de claves candidatas S_K resulta $\{EF, FH\}$

5.5. Set 5

Corriendo el programa sobre el set 5 se obtiene

```
> ./fd Examples/set5.txt
A+ closure: ABCDEFG
Candidate keys:
A
B
C
D
E
FG
```

Por lo que la clausura de atributos del conjunto A resulta $A^+ = \{ABCDEFG\}$, y el conjunto de claves candidatas S_K resulta $\{A, B, C, D, E, FG\}$