

Algoritmo Floyd Warshall

Vanessa Dantas de Souto Costa
vanessa.dantas796@gmail.com
github.com/vanessadants/Floyd-Warshall

Julho 2019

1 Introduction

O Algoritmo Floyd Warshall foi desenvolvido com o intuito de resolver o problema do caminho mais curto de todos os pares de vértices em um grafo, seja ele direcionado ou não, com pesos positivos ou negativos, desde que não contenha ciclos negativos.

2 Implementação do algoritmo de Floyd Warshall

Para implementar o algoritmo foi seguido o pseudocódigo do Livro do Thomas Cormen [1], conforme Figura 1:

```
FLOYD-WARSHALL( $W$ )
1   $n = W.rows$ 
2   $D^{(0)} = W$ 
3  for  $k = 1$  to  $n$ 
4      let  $D^{(k)} = (d_{ij}^{(k)})$  be a new  $n \times n$  matrix
5      for  $i = 1$  to  $n$ 
6          for  $j = 1$  to  $n$ 
7               $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$ 
8  return  $D^{(n)}$ 
```

Figure 1: Pseudocódigo do algoritmo de Floyd Warshall

O projeto foi escrito em C++. Ele lê um arquivo que contém um número na primeira linha, que indica o número N de vértices do grafo, seguido de N x N valores que indicam os pesos associados as arestas que conectam dois vértices, e implementa o Algoritmo de FloydWarshall. A solução do algoritmo é apresentada por meio do cálculo e armazenamento dos menores caminhos para cada vértice. Além disso, indicamos caso existam ciclos negativos.

```
1 int main(int argc, char **argv){
2     int n;
3     double D[MAX][MAX]={};
4     bool negCycle;
5
6     read(argv[1],D,n);
7     cout<<"Before Floyd Warshall\n";
8     print(D,n);
9     floydWarshall(D,n);
10    cout<<"After Floyd Warshall\n";
11    print(D,n);
12
13    negCycle=negCyclefloydWarshall(D,n);
14    if(negCycle){
15        cout<<"This graph contains negative cycles.\n";
16    }else{
17        cout<<"This graph doesn't contain negative cycles.\n";
18    }
19    write("saida.txt", D, n,negCycle);
20    exit(0);
21 }
```

Inicializamos a matriz da solução a partir da leitura de um arquivo .txt de entrada, considerando inf (infinity) como o maior valor representável.

```
1 void read(char *fileName, double D[][MAX], int & n){
2     int i,j;
3     ifstream arq;
4     arq.open(fileName);
5     if (arq.is_open()){
6         cout<<"reading file "<<fileName<<"...\n";
7         arq>>n;
8         char aux[MAX];
9         for (i=0;i<n;i++){
10             for(j=0;j<n;j++){
11                 arq>>aux;
12                 if(strcasecmp(aux,"INF")==0)
13                     {
14                         D[i][j]=DBL_MAX;
15                     }else{
16                         D[i][j]=atof(aux);
17                     }
18             }
19         }
20         arq.close();
21 }
```

```

21         cout << "Done\n";
22     }else{
23         cout << "Unable to open file "<<fileName<<"\n";
24         exit(0);
25     }
26 }

```

Em seguida, temos a função que para cada vértice calcula a distância mais curta utilizando floyd warshall.

```

1 void floydWarshall(double D[][MAX], int n){
2     int i,j,k;
3     for(k=0;k<n;k++){
4         for (i=0;i<n;i++){
5             for(j=0;j<n;j++){
6                 if(D[i][j]>D[i][k]+D[k][j]){
7                     D[i][j]=D[i][k]+D[k][j];
8                 }
9             }
10        }
11    }
12 }

```

Verificamos caso haja ciclo negativo, pois se a distancia para qualquer vertice para ele mesmo se tornar negativa depois de executado o algoritmo, isso significa que o grafo contém ciclo negativo.

```

1 bool negCyclefloydWarshall(double D[][MAX], int n)
2 {
3     // Se a distancia para qualquer vertice
4     // para ele mesmo se tornar negativa
5     // cont m ciclo negativo
6     int i;
7     for (i = 0; i < n; i++){
8         if (D[i][i] < 0){
9             return true;
10        }
11    }
12    return false;
13 }

```

Imprimimos na tela a solução e se houve ou não ciclo negativo:

```

1 void print(double D[][MAX], int n){
2     int i,j;
3     cout<<"n = "<<n<<endl;
4     cout<<"D = \n";
5     for (i=0;i<n;i++){
6         for(j=0;j<n;j++){
7             if(D[i][j]==DBL_MAX){
8                 cout<<setw(15)<<"INF";
9             }else{
10                cout<<setw(15)<<D[i][j];
11            }
12        }
13    }
14 }

```

```

13         cout<<endl;
14     }
15
16 }

```

Por fim, salvamos também essas informações em um arquivo de saída.

```

1 void write(char *fileName, double D[][MAX], int n, bool
   negCycle){
2     int i,j;
3     ofstream arq;
4     arq.open(fileName);
5     if (arq.is_open()){
6         cout<<"writing file "<<fileName<<"...\n";
7         //salvando o n mero de vertices
8         arq<<"n = "<<n<<endl;
9         //salvando a matriz com as dist ncias D
10        arq<<"D = \n";
11        for (i=0;i<n;i++){
12            for(j=0;j<n;j++){
13                if (D[i][j]==DBL_MAX){
14                    arq<<setw(15)<<"INF"
15                    ;
16                }else{
17                    arq<<setw(15)<<D[i][
18                    j];
19                }
20            }
21            arq<<endl;
22        }
23        if(negCycle){
24            arq<<"This graph contains negative
25            cycles.\n";
26        }else{
27            arq<<"This graph doesn't contain
28            negative cycles.\n";
29        }
30        arq.close();
31        cout << "Done\n";
32    }else{
33        cout << "Unable to open file "<<fileName<<"\
34        n";
35        exit(0);
36    }
37 }

```

3 Complexidade

O tempo de execução do algoritmo de Floyd-Warshall é determinado pelo looping triplicado para as linhas 3-7 na Figura 1. Como cada execução da linha 7 possui $O(1)$ de complexidade, o algoritmo é executado no tempo $\theta(n^3)$.

4 Experimentos

Realizamos 3 experimentos para mostrar que o código funciona corretamente.

O primeiro exemplo foi retirado do próprio livro do Thomas Cormen [1], capítulo 25 pág. 696.

```
reading file entradaCormen.txt...
Done
Before Floyd Warshall
n = 5
D =
      0      3      8      INF      -4
      INF     0      INF     1      7
      INF     4      0      INF     INF
      2      INF     -5     0      INF
      INF     INF     INF     6     0
After Floyd Warshall
n = 5
D =
      0      1     -3      2     -4
      3      0     -4      1     -1
      7      4      0      5      3
      2     -1     -5     0     -2
      8      5      1      6      0
This graph doesn't contain negative cycles.
writing file saida.txt...
Done
```

Figure 2: Execução do algoritmo para entrada do livro do Thomas Cormen [1]

O segundo exemplo foi retirado de vídeo do youtube disponível em [2]

```
reading file entrada.txt...
Done
Before Floyd Warshall
n = 4
D =
      0      INF     -2      INF
      4      0      3      INF
      INF     INF     0      2
      INF     -1     INF     0
After Floyd Warshall
n = 4
D =
      0     -1     -2      0
      4      0      2      4
      5      1      0      2
      3     -1      1      0
This graph doesn't contain negative cycles.
writing file saida.txt...
```

Figure 3: Execução do algoritmo para entrada retirado de vídeo do youtube disponível em [2]

Por último, o terceiro exemplo contém grafo com ciclo negativo.

```
reading file entradaCicloNeg.txt...
Done
Before Floyd Warshall
n = 4
D =
      0      1      1      1
      2     -1      2      3
     -1      3      0     -1
     -1     -1     -1     -1
After Floyd Warshall
n = 4
D =
     -8     -9     -9    -14
     -9    -10    -10    -15
    -11    -12    -12    -17
    -18    -19    -19    -24
This graph contains negative cycles.
writing file saida.txt...
Done
```

Figure 4: Execução do algoritmo para grafo com ciclo negativo

References

- [1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.
- [2] Michael Sambol. Algoritmo floyd-warshal em 4 minutos.