How to solve a transitive closure question

陈柯昊

1160300823

2017 年 5 月 24 日

Contents

1 Introduction

1.1 What is Transitive Closure?

In mathematics, the **transitive closure** of a binary relation R on a set X is the smallest relation on X that contains R and is transitive.

1.2 A simple example of Transitive Closure.

For example, if X is a set of airports and x R y means ‘there is a direct flight from airport x to airport y’ (for x and y in X), then the transitive closure of R on X is the relation R^+ such that x R^+ y means ‘it is possible to fly from x to y in one or more flights’. Informally, the transitive closure gives you the set of all places you can get to from any starting place. More formally, the transitive closure of a binary relation R on a set X is the transitive relation R^+ on set X such that contains R and R^+ is minimal. If the binary relation itself is transitive, then the transitive closure is that same binary relation; otherwise, the transitive closure is a different relation.

1.3 Existence and description

For any relation R, the transitive closure of R always exists. To see this, note that the intersection of any family of transitive relations is again transitive. Furthermore, there exists at least one transitive relation containing R, namely the trivial one: X ×X. The transitive closure of R is then given by the intersection of all transitive relations containing R.

2 Definition in math

2.1 Definition

Definition 1 (Transitive Closure) *In mathematics, the **transitive closure** of a binary relation R on a set X is the smallest relation on X that*

contains R and is transitive.
$R^+ = \bigcap_{R \subseteq R'} R'$

2.2 Properties

The intersection of two transitive relations is transitive. The union of two transitive relations need not be transitive. To preserve transitivity, one must take the transitive closure. This occurs, for example, when taking the union of two equivalence relations or two preorders. To obtain a new equivalence relation or preorder one must take the transitive closure (reflexivity and symmetry—in the case of equivalence relations—are automatic). For finite sets, we can construct the transitive closure step by step, starting from R and adding transitive edges. This gives the intuition for a general construction. For any set X, we can prove that transitive closure is given by the following expression

$$R^+ = \bigcup_{i \in 1,2,3...} R^i.$$

where R^i is the i-th power of R, defined inductively by

$$R^1 = R$$

and, for i $>$ 0

$$R^{i+1} = R \circ R^i$$

where $\circ$ denotes composition of relations.
To show that the above definition of R^+ is the least transitive relation containing R, we show that it contains R, that it is transitive, and that it is the smallest set with both of those characteristics.

- R^+contains all of the R^i, so in particular R^+ contains R.

- R^+ is transitive:every element of R^+ is in one of the R^i,so R^+ must be transitive by the following reasoning:if$(s1, s2) \in R^j$ and$(s2, s3) \in$

R^k,then from composition's associativity,$(s1, s3) \in R^{j+k}$ (and thus in R^+)because of the definition of R^i.

- R^+ is minimal: Let G be any transitive relation containing R,we want to show that $R^+ \subseteq G$.It is sufficient to show that for every$i > 0$,$R^i \subseteq G$.Well,since G contains R,$R^1 \subseteq G$.And since$G$ is transitive,whenever $R^i \subseteq G$,$R^{i+1} \subseteq G$ according to the construction of R^i and what it means to be transitive.Therefore,by induction,G contains every R^i,and thus also R^+.

2.3 In graph theory

In computer science, the concept of transitive closure can be thought of as constructing a data structure that makes it possible to answer reachability questions. That is, can one get from node a to node d in one or more hops? A binary relation tells you only that node a is connected to node b, and that node b is connected to node c, etc. After the transitive closure is constructed, as depicted in the following figure, in an O(1) operation one may determine that node d is reachable from node a. The data structure is typically stored as a matrix, so if matrix[1][4] = 1, then it is the case that node 1 can reach node 4 through one or more hops. The transitive closure of a directed acyclic graph (DAG) is the reachability relation of the DAG and a strict partial order.

3 How to solve a transitive closure questions?

To solve a transitive closure problem,in my opinion,we should start from the definition of **transitive closure**.

In mathematics, a binary relation R over a set X is transitive if whenever an element a is related to an element b, and b is in turn related to an element c, then a is also related to c. Transitivity (or transitiveness) is a key property of both partial order relations and equivalence relations.

In a easier way,if there's a set A,and element$< x, y > \in A and < y, z > \in A$ as well,then we can say that the element$< x, z > \in A$. In graph theory,we see it as a reachability question.

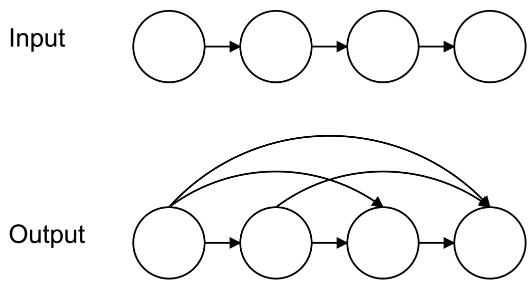

Figure 1: in graph theory

3.1 Algorithms

Passing the closure algorithm To solve the problem of transitive closure, we can start from the definition, through the matrix of the logic to achieve the relationship between the closure. Of course, there is a relatively simple method, Warshe11 gives a way to achieve the transfer of closure, through the analysis of this algorithm, we found that it can be optimized to achieve the simplification of the operation process, both said For the warshe11 algorithm, I will use the Warshell algorithm and the Warshe11 algorithm to optimize the algorithm to distinguish between the two.

3.2 Original Method

After having a general understanding of these theories,now we can try to put them into practice.Since we have learned '**matrix of a relation**',we can use this kind of matrix to show the relation R. Basically,if we use a matrix of a relation A to describe a group of relations,we can simply do 'matrix Boolean multiplication($\odot$)' to itself for many times until all the elements of the matrix is 1. This is a really basic and original way to solve this kind of questions,and we have to commit that this is a piratical and intuitive method.

3.2.1 Pseudocode

Transitive-Closure(B) *(B is the zero-one $n \times n$ matrix for relation R)*

M = B_R
A = M
for i = 2 to n
M = M $\circ MR$
A = A$\vee$M
return A // A is the zero-one matrix for R^+

3.2.2 Time Complexity

Time complexity is $O(n^4)$.

3.2.3 Code

```
int intitialWay(int a[N][N])
{
    int i, j, k, flag, n;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
```

```
            b[i][j] = a[i][j];
            }
        }
            for (n = 2; n <= N; n++)
            {
            for (k = 0; k < N; k++)
                {
                    for (i = 0; i < N; i++)
                    {
                        for (j = 0; j < N; j++)
                        {
                        a[i][j] |= (b[i][k] & b[k][j]);
                        }
                    }

                }
        }
}
```

3.3 Warshall-Floyd Algorithm

Warshall didn't try to solve the question with the original thought,he tried to work it out with the thought of 'relay point'.

If a graph can start from **i** point,then it walks through some relay points,and finally reaches **j** point.

The relay points can be zero point,1st point ...... and so on. And relay points can be more than 1.It seems that relay points are really complex,but Warshall followed 'Divide and Conquer' thought,transform a big question to some small and easy questions. Warshall divided the pathes from**i**point to **j** point into 2 categories:the pathes that walk through 0,1,2... point and the pathes which do not walk through 0,1,2... point till the last point.

3.3.1 Time Complexity

Time complexity $T(n) = O(n^3)$

3.3.2 Pseudocode

Warshall(B) *// B is the zero-one n×n matrix for relation R*

A = B_R
for k=1 **to** n
for i = 1 **to** n
for j=1 **to** n
a_{ij}=$a_{ij} \vee (a_{ik} \wedge a_(kj)$
return A // A is the zero-one matrix for R^+

3.3.3 Code

```
int warshallb(int a[N][N])
{
    int i = 0;
    int j = 0;
    int k = 0;
    for ( k = 0; k < N; k++)
        {
            for (i = 0; i < N; i++)
                {
                    for ( j = 0; j < N; j++)
                        {
                        a[i][j] = a[i][j] | a[k][j];
                        }
                }
        }
}
```

3.4 An improved version of Warshall-floyd Algorithm

After we carefully analyze the Warshall algorithm,it can be found that when the implementation of the algorithm in the fifth step, the parameters i, k are not changed, only the value of j is changing. So we can make some changes to this step to achieve the optimization of the algorithm.

3.4.1 Time Complexity

Time complexity is $O(n^3)$.

3.4.2 Pseudocode

Warshall(B) *// B is the zero-one n×n matrix for relation R*

1 A = B_R
2 **for** k = 1 **to** n
3 **for** i = 1 **to** n
4 **if** a_{ik} == 1
5 **for** j = 1 **to** n a_{ij}=$a_{ij} \vee (a_{ik} \wedge a_(kj))$
6 return A // A is the zero-one matrix for R^+

3.4.3 Code

```
int warshall(int a[N][N])
{
    int col = 0;
    int line = 0;
    int temp = 0;
    for (col = 0; col < N; col++)
        {
            for (line = 0; line < N; line++)
            {
                if (a[line][col] == 0)
```

```
                    {
                    for (temp = 0; temp < N; temp++)
                    {
                    a[line][temp] = a[line][temp] | a[col][temp];
                    }
                    }
                }
            }
    return TRUE;
}
```

4 Conclusion

4.1 Differences

These are the graphs of three Algorithms.

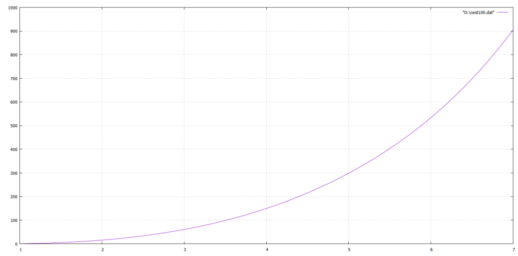

Figure 2: Original Method

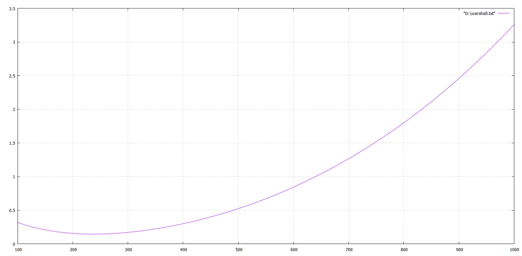

Figure 3: Warshall Algorithm

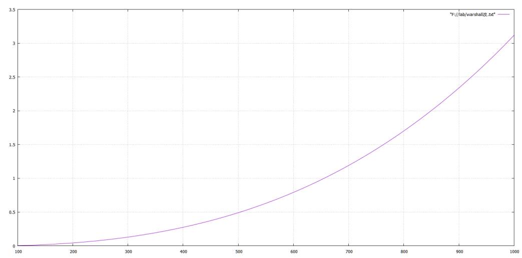

Figure 4: Improved Warshall Algorithm

And I also compare these two warshall algorithms,including both versions.The chart is put below.

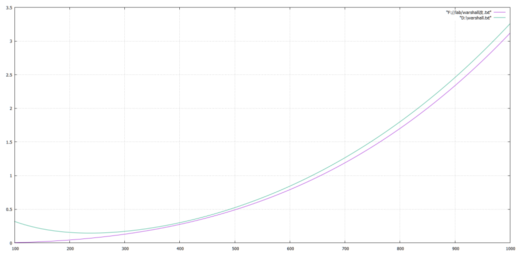

Figure 5: The comparison of two warshall algorithms

4.2 Analysis

We can compare the 3 methods by the charts above.The original method is the slowest one because its time complexity O(n)=n^4,and both two warshall algorithms' time complexity O(n)=n^3.Thus,when facing big data,the original method is fairly slow while warshall algorithms are far more faster than the original method.Though all the 3 algorithms are available, the efficiency differs a lot.So this result reminds me that we should always pay attention to the time complexity of the algorithm.In addition,it is really necessary to improve the algorithms we use now.

5 appendix

5.1 Full Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include<time.h>
#define N 10000
#define TRUE 0
int Arr[N][N] = {{0}};
int b[N][N] = {{0}};
int generate_random_matrix()
```

```
{
        int i = 0;
        int j = 0;
        srand(time(NULL));
        for (i = 0; i < N; ++i)
        {
                for (j = 0; j < N; ++j)
                {
                Arr[i][j] = rand() % 2;
                }
        }
}
int output_matrix(int a[N][N])
{
        int i = 0, j = 0;
        for (i = 0; i < N; i++)
        {
                for (j = 0; j < N; j++)
                {
                 printf("%d\t", a[i][j]);
                }
                putchar('\n');
        }
        return TRUE;
}
int warshall(int a[N][N])
{
        int col = 0;
        int line = 0;
        int temp = 0;
        for (col = 0; col < N; col++)
```

```
	{
		for (line = 0; line < N; line++)
		{
			if (a[line][col] == 0)
			{
			for (temp = 0; temp < N; temp++)
			{
			 a[line][temp] = a[line][temp] | a[col][temp];
			}
			}
		}
	}
	return TRUE;
}
int warshallb(int a[N][N])
{
	int i = 0;
	int j = 0;
	int k = 0;
	for ( k = 0; k < N; k++)
	{
		for (i = 0; i < N; i++)
		{
			for ( j = 0; j < N; j++)
			{
				a[i][j] = a[i][j] | a[k][j];
			}
		}
	}
}
int intitialWay(int a[N][N])
```

```
{
        int i, j, k, flag, n;
        for (i = 0; i < N; i++)
        {
                for (j = 0; j < N; j++)
                {
                        b[i][j] = a[i][j];
                }
        }
        for (n = 2; n <= N; n++)
        {
                for (k = 0; k < N; k++)
                {
                    for (i = 0; i < N; i++)
                {
                                for (j = 0; j < N; j++)
                        {
             a[i][j] |= (b[i][k] & b[k][j]);
                        }
                }

                }
        }
}
int main(void)
{
        int i;
        for (i = 0; i <= 20; i++)
        {
        clock_t start, finish;
        double duration;
```

```
        FILE *stream;
        stream = fopen("d://warshall.txt", "a+" );
        generate_random_matrix();
        start = clock();
        warshallb(Arr);
        //output_matrix(Arr);
        finish = clock();
        duration = (double)(finish - start) / CLOCKS_PER_SEC;
        fprintf(stream, "%f seconds\n", duration );
        fclose(stream);
        }
        return 0;
}
```