GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: ALGORITMO DE FLOYD-WARSHALL

1. Introducción

Dado un grafo ponderado no dirigido G con n vértices. La tarea es encontrar la longitud del camino más corto d_{ij} entre cada par de vértices i y j del grafo.

2. Conocimientos previos

2.1. Matriz

Una matriz es una estructura conformada por filas y columnas, idealmente más de dos filas y columnas, de hecho, podemos decir que si una matriz tiene una única fila o una única columna, entonces estamos hablando de un vector y no una matriz como tal.

2.2. Matriz de distancia o costo

El grafo está representado por una matriz cuadrada M de tamaño n^2, donde n es el número de vértices. Si hay una arista ponderada entre un vértice x y un vértice y, entonces el elemento m x, y es igual al valor de la ponderación, de lo contrario su valor será uno que indica la no existencia de la aristas cuyo valor debe ajustarse de acuerdo a la situación del problema y como vamos a utilizar la matriz.

3. Desarrollo

El algoritmo de Floyd-Warshall, descrito en 1959 por Bernard Roy, es un algoritmo de análisis sobre grafos para encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución. El algoritmo de Floyd-Warshall es un ejemplo de programación dinámica.

El algoritmo de Floyd-Warshall compara todos los posibles caminos a través del grafo entre cada par de vértices. El algoritmo es capaz de hacer esto con sólo V^3 comparaciones (esto es notable considerando que puede haber hasta V^2 aristas en el grafo, y que cada combinación de aristas se prueba). Lo hace mejorando paulatinamente una estimación del camino más corto entre dos vértices, hasta que se sabe que la estimación es óptima.

Sea un grafo G con conjunto de vértices V, numerados de 1 a N. Sea además una función caminoMinimo(i,j,k) que devuelve el camino mínimo de i a j usando únicamente los vértices de 1 a k como puntos intermedios en el camino. Ahora, dada esta función, nuestro objetivo es encontrar el camino mínimo desde cada i a cada j usando únicamente los vértices de 1 hasta k+1.

Hay dos candidatos para este camino: un camino mínimo, que utiliza únicamente los vértices del conjunto (1...k); o bien existe un camino que va desde i hasta k+1 , y de k+1 hasta j , que es mejor. Sabemos que el camino óptimo de i a j que únicamente utiliza los vértices de 1 hasta k está definido por caminoMinimo(i,j,k) , y está claro que si hubiera un camino mejor de i a k+1 a j , la longitud de este camino sería la concatenación del camino mínimo de i a k+1 (utilizando vértices de (1...k)) y el camino mínimo de k+1 a j (que también utiliza los vértices en (1...k)).

Para que haya coherencia numérica, Floyd-Warshall supone que no hay ciclos negativos (de hecho, entre cualquier pareja de vértices que forme parte de un ciclo negativo, el camino mínimo no está bien definido porque el camino puede ser infinitamente pequeño). No obstante, si hay ciclos negativos, Floyd-Warshall puede ser usado para detectarlos. Si ejecutamos el algoritmo una vez más, algunos caminos pueden decrementarse pero no garantiza que, entre todos los vértices, caminos entre los cuales puedan ser infinitamente pequeños, el camino se reduzca. Si los números de la diagonal de la matriz de caminos son negativos, es condición necesaria y suficiente para que este vértice pertenezca a un ciclo negativo.

La idea clave del algoritmo es dividir el proceso de encontrar el camino más corto entre dos vértices en varias fases incrementales.

Numeremos los vértices del 1 al n. La matriz de distancias es $d[][]$.

Antes k-ésima fase $(k = 1...n)$, $d[i][j]$ para cualquier vértice i y j almacena la longitud del camino más corto entre el vértice i y el vértice j, usando las aristas que contiene solo los vértices $1, 2, ..., k-1$. En otras palabras, antes de k-ésima fase el valor de $d[i][j]$ es igual a la longitud del camino más corto desde el vértice i al vértice j, si este camino puede entrar solo en el vértice con números menores que k. (el principio y el final del camino no están restringidos por esta propiedad).

Es fácil asegurarse de que esta propiedad se mantenga para la primera fase. Para $k = 0$, podemos llenar la matriz con $d[i][j] = w_{ij}$ si existe una arista entre i y j con peso w_{ij} y $d[i][j] = \infty$ si no existe una arista. En la práctica ∞ habrá un alto valor que su valor dependerá de la situación.

Supongamos ahora que estamos en la fase k-ésima, y queremos calcular la matriz $d[][]$ para que cumpla con los requisitos para la fase (K+1). Tenemos que arreglar las distancias para algunos pares de vértices (*i*,*j*). Hay dos casos fundamentalmente diferentes:

- La forma más corta del vértice i al vértice j con vértices internos del conjunto $1, 2, \ldots, k$ coincide con la ruta más corta con vértices internos del conjunto $1, 2, \ldots, k-1$.

  En este caso, $d[i][j]$ no cambiará durante la transición.

- El camino más corto con vértices internos desde $1, 2, \ldots, k$ es más corto.

  Esto significa que el nuevo camino más corto pasa por el vértice k. Esto significa que podemos dividir el camino más corto entre i y j en dos caminos: el camino entre i y k, y el camino entre k y j. Está claro que ambos caminos solo usan vértices internos de $1, 2, \ldots, k-1$ y son los caminos más cortos en ese sentido. Por lo tanto, ya hemos calculado las longitudes de esos caminos antes, y podemos calcular la longitud del camino más corto entre i y j como $d[i][k] + d[k][j]$.

Combinando estos dos casos encontramos que podemos recalcular la longitud de todos los pares (i, j) en la k-ésima fase de la siguiente manera:

$$d_{new}[i][j] = min(d[i][j], d[i][k] + d[k][j])$$

Por lo tanto, todo el trabajo que se requiere en la k-ésima fase es iterar sobre todos los pares de vértices y volver a calcular la longitud del camino más corto entre ellos. Como resultado, después de la k-ésima fase, el valor $d[i][j]$ en la matriz de distancia es la longitud del camino más corto entre i y j, o es ∞ si el camino entre los vértices i y j no existir.

Una última observación: no necesitamos crear una matriz de distancia separada $d_{new}[][]$ para almacenar temporalmente las rutas más cortas de la k-ésima fase, es decir, todos los cambios se pueden realizar directamente en la matriz $d[][]$ en cualquier momento. fase. De hecho, en cualquier k-ésima fase, como máximo estamos mejorando la distancia de cualquier camino en la matriz de distancia, por lo tanto, no podemos empeorar la longitud del camino más corto para cualquier par de vértices que se van a procesar en el (k+1)-ésima fase o posterior.

3.1. Recuperando la secuencia de vértices en el camino más corto

Es fácil mantener información adicional con la que será posible recuperar el camino más corto entre dos vértices dados en forma de una secuencia de vértices.

Para ello, además de la matriz de distancias $d[][]$, se debe mantener una matriz de ancestros $p[][]$, que contendrá el número de la fase donde se modificó por última vez la distancia más corta entre dos vértices. Está claro que el número de la fase no es más que un vértice en medio del camino más corto deseado. Ahora solo necesitamos encontrar el camino más corto entre los vértices i y $p[i][j]$, y entre $p[i][j]$ y j. Esto conduce a un algoritmo de reconstrucción recursivo simple del camino más corto.

3.2. El caso de los pesos reales

Si los pesos de las aristas no son enteros sino reales, es necesario tener en cuenta los errores que se producen al trabajar con tipos flotantes.

El algoritmo de Floyd-Warshall tiene el efecto desagradable de que los errores se acumulan muy rápidamente. De hecho, si hay un error en la primera fase de δ, este error puede propagarse a la segunda iteración como 2δ, a la tercera iteración como 4δ, y así sucesivamente. Para evitar esto, el algoritmo se puede modificar para tener en cuenta el error $(EPS = \delta)$ utilizando la siguiente comparación:

```
if(d[i][k] + d[k][j] < d[i][j] - EPS)
   d[i][j] = d[i][k] + d[k][j];
```

3.3. El caso de los ciclos negativos

Formalmente, el algoritmo de Floyd-Warshall no se aplica a los gráficos que contienen ciclos de peso negativo. Pero para todos los pares de vértices i y j para los que no existe un camino que comience en i, visite un ciclo negativo y termine en j, el algoritmo seguirá funcionando correctamente.

Para el par de vértices para los que no existe la respuesta (debido a la presencia de un ciclo negativo en el camino entre ellos), el algoritmo de Floyd almacenará cualquier número (quizás muy negativo, pero no necesariamente) en la matriz de distancia. Sin embargo, es posible mejorar el algoritmo de Floyd-Warshall, de modo que trate con cuidado tales pares de vértices y los emita, por ejemplo, como $-INF$.

Esto se puede hacer de la siguiente manera: ejecutemos el algoritmo habitual de Floyd-Warshall para un grafo dado. Entonces no existe un camino más corto entre los vértices i y j, si y sólo si, hay un vértice t que es alcanzable desde i y también desde j, para el cual $d[t][t] < 0$.

Además, al usar el algoritmo de Floyd-Warshall para grafos con ciclos negativos, debemos tener en cuenta que pueden surgir situaciones en las que las distancias pueden volverse exponencialmente rápidas hacia el negativo. Por lo tanto, el desbordamiento de enteros debe manejarse limitando la distancia mínima por algún valor (por ejemplo, -INF).

4. Implementación

En el siguiente codigo independientemente del lenguaje $dist$ es la matriz de distancia del grafo, una vez ejecutado el algoritmo la distancia mínima entre cualquier par de vértices x,y va estar en $flojd[x][y]$.

4.1. C++

```
inline void FloydWarshall(){
   for(int i=1;i<=n;i++){
      for(int j=1;j<=n;j++){
        flojd[i][j] = dist[i][j];
      }
      flojd[i][i] = 0;
   }
   for(int k=1;k<=n;k++){
      for (int i=1;i<=n;i++){
         for (int j=1;j<=n;j++){
            if (flojd[i][k] + flojd[k][j] < flojd[i][j]){
               flojd[i][j] = flojd[i][k] + flojd[k][j];
            }
         }
       }
    }
}
```

4.2. Java

```
private void FloydWarshall(){
   for(int i=1;i<=n;i++){
      for(int j=1;j<=n;j++){
```

```
            flojd[i][j] = dist[i][j];
        }
        flojd[i][i] = 0;
    }
    for(int k=1;k<=n;k++){
       for (int i=1;i<=n;i++){
          for (int j=1;j<=n;j++){
             if (flojd[i][k] + flojd[k][j] < flojd[i][j]){
                flojd[i][j] = flojd[i][k] + flojd[k][j];
             }
          }
       }
    }
}
```

5. Complejidad

Como podemos analizar el algoritmo tiene una complejidad de N^3 donde N es la cantidad de nodos, lo que hace que este algoritmo sea factible su utilización cuando el numero de de nodos sea menor igual a 250 nodos.

6. Aplicaciones

El algoritmo de Floyd-Warshall es un ejemplo de programación dinámica. El algoritmo de Floyd-Warshall puede ser utilizado para resolver los siguientes problemas, entre otros:

1. Camino mínimo en grafos dirigidos (algoritmo de Floyd).
2. Cierre transitivo en grafos dirigidos (algoritmo de Warshall). Es la formulación original del algoritmo de Warshall. El grafo es un grafo no ponderado y representado por una matriz booleana de adyacencia. Entonces la operación de adición es reemplazada por la conjunción lógica(AND) y la operación menor por la disyunción lógica (OR).
3. Encontrar una expresión regular dada por un lenguaje regular aceptado por un autómata finito (algoritmo de Kleene).
4. Comprobar si un grafo no dirigido es bipartito.
5. Ruta optima. En esta aplicación es interesante encontrar el camino del flujo máximo entre 2 vértices. Esto significa que en lugar de tomar los mínimos con el pseudocodigo anterior, se coge el máximo. Los pesos de las aristas representan las limitaciones del flujo. Los pesos de los caminos representan cuellos de botella; por ello, la operación de adición anterior es reemplazada por la operación mínimo.
6. Inversión de matrices de números reales (algoritmo de Gauss-Jordan).

7. Ejercicios propuestos

A continuación una lista de ejercicios que se resuelven utilizando este algoritmo:

- DMOJ - Feria tecnológica
- SPOJ - ROHAAN - Defend The Rohan
- SPOJ - SOCIALNE - Possible Friends
- SPOJ - CHICAGO - 106 miles to Chicago