added: 10 Jun 2008 19:41
Edited: 24 Aug 2011 1:19

Floyd's algorithm-Uorshella finding the shortest paths between all pairs of vertices

Contents [hide]

Given a directed or undirected weighted graph $G$with $n$vertices. Required to find the values of all variables d_{ij}- length of the shortest path from vertex $i$to vertex j.

It is assumed that the graph does not contain cycles of negative weight (if the response between some pairs of vertices may not exist - it will be infinitely small).

This algorithm has been simultaneously published in the papers of Robert Floyd (Robert Floyd) and Steven Uorshella (Warshall) (Stephen Warshall) in 1962, on behalf of which the algorithm is called at the moment. However, in 1959, Bernard Roy (Bernard Roy) published essentially the same algorithm, but its publication has remained unnoticed.

Description of the algorithm

The key idea of the algorithm - the partition process of finding shortest paths on the **phase** .

Before kth phase ($k = 1 \dots n$) it is believed that in the matrix of distances $d[][]$stored length of the shortest paths, which contain as internal vertices only vertices in the set $\{1, 2, \dots, k-1\}$(we number the vertices, starting with the unit).

In other words, before kth phase magnitude $d[i][j]$equal to the length of the shortest path from vertex $i$to vertex j , if this path is allowed to enter only the top with numbers below k(beginning and end of the path are not considered).

Easy to see that this property is to perform for the first phase, it is sufficient in the distance matrix $d[][]$to record the adjacency matrix: $d[i][j] = g[i][j]$- the cost of an edge from vertex $i$to vertex j. Thus, if between some vertices edges not, should record the value of "infinity" ∞. From a vertex to itself, always record the value 0, it is critical for the algorithm.

Suppose now that we are on kth phase, and we want to **recalculate the** matrix $d[][]$so as to conform to the requirements already for $k+1$th phase. We fix some vertices $i$and j. We there are two fundamentally different cases:

- The shortest path from vertex $i$to vertex j, which is allowed to pass through the additional peaks $\{1, 2, \dots, k\}$, **coincides** with the shortest path, which is allowed to pass through the vertices of the set $\{1, 2, \dots, k-1\}$.

  In this case, the value $d[i][j]$will not change in the transition from kth at $k+1$of th phase.

- "New" was the shortest way **better** the "old" way.

  This means that the "new" shortest path passes through the vertex k. Just note that we do not lose generality by considering only the more simple way (ie the way, do not pass on some vertex twice).

  Then we note that if we let us divide this "new" way of the apex $k$into two halves (one running $i \Rightarrow k$and the other - $k \Rightarrow j$), each of these halves are no longer comes to the top k. But then it turns out that the length of each of the halves was calculated yet for $k-1$th phase or even earlier, and it is sufficient to simply take the sum $d[i][k] + d[k][j]$, she will give the length of the "new" shortest path.

Combining these two cases, we find that on kth phase is required to count the length of the shortest paths between all pairs of vertices i, and $j$as follows:

```
new_d[i][j] = min (d[i][j], d[i][k] + d[k][j]);
```

Thus, all the work that is required to produce kth phase - is to iterate over all pairs of vertices and recalculate the

length of the shortest path between them. As a result, after nth phase in a matrix of distances $d[i][j]$will be recorded length of the shortest path between $i$and j, or ∞, if paths between these nodes does not exist.

The last remark should be done - something that can **not create a separate matrix** $\text{new\_d}[][]$ for temporary matrix of shortest paths on kth phase: all the changes you can make right in the matrix $d[][]$. In fact, if we have improved (reduced) for some value in the matrix of distances, we could not degrade thus the length of the shortest path to any other vertex pairs processed later.

Asymptotic algorithm obviously is $O(n^3)$.

Implementation

The input to the program served graph given in the form of adjacency matrix - a two-dimensional array of $d[][]$ size $n \times n$, in which each element specifies the length of the edge between the corresponding vertices.

It is required that $d[i][i] = 0$for any i.

```
for (int k=0; k<n; ++k)
        for (int i=0; i<n; ++i)
                for (int j=0; j<n; ++j)
                        d[i][j] = min (d[i][j], d[i][k] + d[k][j]);
```

It is assumed that between two if some vertices **are no ribs** , the adjacency matrix was recorded some large number (large enough for it to be longer than the length of any path in the graph); then this edge will always be profitable to take, and the algorithm works correctly.

However, if no special measures are taken, in the presence of edges in the graph **of negative weight** , resulting in a matrix of the form may appear $\infty - 1, \infty - 2$and so on, which, of course, still means that between the corresponding vertices in general there is no way. Therefore, in the presence of negative edges in the graph algorithm Floyd better write it so that he did not fulfill the transitions of the states that already is "no way":

```
for (int k=0; k<n; ++k)
        for (int i=0; i<n; ++i)
                for (int j=0; j<n; ++j)
                        if (d[i][k] < INF && d[k][j] < INF)
                                d[i][j] = min (d[i][j], d[i][k] + d[k][j]);
```

Recovering themselves ways

Easy to maintain additional information - so-called "ancestors", in which it will be possible to restore itself the shortest path between any two given vertices **in a sequence of vertices** .

It's enough to except the distance matrix $d[][]$also support **matrix ancestors** $p[][]$ that for every pair of vertices will contain the number of phases, which were obtained by the shortest distance between them. It is clear that this phase number is not more than the "average" tip of the title of the shortest path, and now we just need to find the shortest path between vertices $i$and $p[i][j]$, and between $p[i][j]$and j. This yields a simple recursive algorithm for recovering the shortest path.

The case of real weights

If the weight of the edges of the graph is not an integer, and real, it is necessary to take into account the uncertainty that inevitably arise when dealing with floating-point types.

With regard to the algorithm of Floyd obnoxious special effects of these errors becomes that found distance algorithm may leave much to the minus because **of accumulated errors** . In fact, if the first phase error has occurred Δ, then in the second iteration of this error may have become 2Δ, on the third - in 4Δ, and so forth.

To avoid this, the comparison algorithm Floyd should be done taking into account the error:

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

The case of negative cycles

If the graph contains a cycle of negative weight, the formally-Uorshella Floyd algorithm is not applicable to this graph.

In fact, for those pairs of vertices i, and $j$between which it is impossible to go into a cycle of negative weight, the algorithm will work correctly.

For those pairs of vertices, the answer to which does not exist (due to the presence of a negative cycle in the way between them), Floyd's algorithm finds an answer in a certain number of (possibly highly negative, but not necessarily). Nevertheless, it is possible to improve the algorithm of Floyd, so he carefully cultivated such a pair of vertices, and drew for them, for example $-\infty$.

This can be done, for example, the following **criterion** "is not the existence of the way." So, even for a given graph has fulfilled the usual Floyd algorithm. Then between the tops $i$and $j$the shortest path does not exist if and only if there exists a vertex t, accessible from $i$and from which is achievable $j$for which the following $d[t][t] < 0$.

In addition, when using the Floyd algorithm for graphs with negative cycles, it should be remembered that arise in the process of distance can go into much less exponentially with each phase. Therefore it is necessary to take measures against integer overflow, to restrict all distance below some value (eg, $-\mathrm{INF}$).

More information about this task, see. Separate article: **"Finding a negative cycle in the graph"** .