MAXimal

- home
- algo
- bookz
- forum
- about

added: 10 Jun 2008 19:39
Edited: 2 Oct 2010 1:35

Algorithm for finding the shortest paths in Leviticus from a given vertex to all other vertices

Contents [hide]

Suppose we are given a graph with N vertices and M edges, each of which indicated its weight L_i. Also, given the starting vertex V_0. Required to find the shortest path from vertex V_0 to all other vertices.

Leviticus algorithm solves this problem very efficiently (about the asymptotic behavior and the speed of the cm. Below).

Description

Let array D [1..N] will contain the current shortest path lengths, ie D_i - this is the current length of the shortest path from vertex V_0 to vertex i. Initially D array is filled with values "infinity", except $D_{V_0} = 0$ At the end of the algorithm, this array will contain the final shortest distance.

Let array P [1..N] contains the current ancestors, ie P_i - is the peak preceding the vertex i in the shortest path from vertex V_0 to i. As well as an array D, array P is changed gradually during the algorithm and at its end receives the final values.

Now actually the algorithm Leviticus. At each step, supported by three sets of vertices:

- M_0 - vertex distance which has already been calculated (but perhaps not entirely);
- M_1 - vertex distance are calculated;
- M_2 - vertex distance is yet to be calculated.

Vertices in the set M_1 is stored in the form of bi-directional queue (deque).

Initially all nodes are placed in a plurality of M_2, apart from the vertex V_0, which is placed in a plurality of M_1.

At each step of the algorithm, we take the top of the set M_1 (We reach the top element of the queue). Let V - is the selected vertex. Translate this vertex in the set M_0. Then review all the edges emanating from this vertex. Let T - this is the second end of this rib (i.e., not equal to V), and L - the length of this edge.

- If T belongs to M_2, then T is transferred to a set of M_1 to the end of the queue. D_T D is set equal to $_V$ + L.
- If T belongs to M_1, then try to improve the value of D_T: $D_T = \min(D_T, D_V + L)$. The very top of T is never moved in the queue.
- If T belongs to M_0, and if D_T can be improved ($D_T > D_V + L$), then improve D_T, and T return to the top of the set M_1, placing it in the top of the queue.

Of course, every time you update the array D should be updated and the value in the array P.

Implementation Details

Create an array ID [1..N], in which each vertex will be stored, which set it belongs: 0 - if M_2 (ie, the distance is infinite), 1 - if M is $_1$ (ie, the vertex is queue) and 2 - when M_0 (a path has been found, the distance is less than infinity).

Queue processing can be realized by a standard data structure deque. However, there is a more efficient way. Firstly, it is obvious in the queue at any one time will be stored a maximum of N elements. But secondly, we can add elements and beginning and end of the queue. Therefore, we can arrange a place on the array size N, but you have to loop it. Ie make an array Q [1..N], pointers (int) to the first element QH and the element after the last QT. The queue is empty when QH == QT. Appending - a record in the Q [QT] and increase QT 1; if QT then went beyond the line (QT == N), then do QT = 0. Adding the queue - reduce the QH-1, if it has moved beyond the stage of (QH == -1), then do QH = N -1.

Implement the algorithm itself is exactly the description above.

Asymptotics

I do not know more or less good asymptotic estimate of this algorithm. I have seen only the estimate O (NM) of the similar algorithm.

However, in practice, the algorithm has proven itself very well: while it is running I rate as **O (M Log N)** , although, again, this is only **an experimental** evaluation.

Implementation

```
typedef pair <int, int> rib;
typedef vector <vector <rib>> graph;

const int inf = 1000 * 1000 * 1000;


int main ()
{
        int n, v1, v2;
        graph g (n);

        Graph reading ... ...

        vector <int> d (n, inf);
        d [v1] = 0;
        vector <int> id (n);
        deque <int> q;
        q.push_back (v1);
        vector <int> p (n, -1);

        while (! q.empty ())
        {
                int v = q.front (), q.pop_front ();
                id [v] = 1;
                for (size_t i = 0; i <g [v] .size (); ++ i)
                {
                        int to = g [v] [i] .first, len = g [v] [i] .second;
                        if (d [to]> d [v] + len)
                        {
                                d [to] = d [v] + len;
                                if (id [to] == 0)
                                        q.push_back (to);
                                else if (id [to] == 1)
                                        q.push_front (to);
                                p [to] = v;
                                id [to] = 1;
                        }
                }
        }

        Conclusion ... the result ...

}
```

6 Комментариев e-maxx Войти

Лучшее вначале Поделиться Избранный

Присоединиться к обсуждению...

Андрей Кравцун • 2 года назад

according to the description in the inner cycle there must be "else if (id[to] == 2) q.push_front (to);" instead of "else if (id[to] == 1) q.push_front (to);" (vertex "to" is in M0 - set). Besides, where did you set vertex V as M0? Some misunderstanding of mine here, i guess...

5 • Ответить • Поделиться ›

Tornike • 2 года назад

is it working for negative edges ?

1 • Ответить • Поделиться ›

e_maxx Модератор → Tornike • 2 года назад

Yes, it is, and that's the main purpose of this algorithm - because standard algorithms like Dijkstra or breadth-first search don't work with negative-cost edges.
On the other hand, you should take into account that this is a heuristical algorithm - it's even non-polynomial, though it works very good in practise (especially on random graphs).

1 • Ответить • Поделиться ›

RiaD → e_maxx • 10 месяцев назад

I'd add that it works about 2^n on some case right in Асимптотика section.

• Ответить • Поделиться ›

B@rmaley.e><e • год назад

А тут точно всё правильно с номерами множеств (id[n])? А то 2 ни разу не встречается, да и вершины из M1 не выходят.

• Ответить • Поделиться ›

Jarekczek • год назад

I can't leave a direct link, cause the exact article is hardly available on the net. But I saw in several places, that in pessimistic case this algorithm (Pape-Levit's, correct?) performs in exponential (2^n) time. Althought in most cases it showed to be faster than Dijkstra.

• Ответить • Поделиться ›