added: 10 Jun 2008 19:30
Edited: 10 Dec 2012 16:05

Finding the shortest paths from a given vertex to all other vertices Dijkstra algorithm

Contents [hide]

Statement of the Problem

Given a directed or undirected weighted graph with $n$vertices and $m$edges. The weights of all edges are non-negative. Contains some starting vertex s. Required to find the length of the shortest paths from the vertex $s$to all other vertices, as well as provide a way to output the shortest paths themselves.

This problem is called "the problem of the shortest paths from a single source" (single-source shortest paths problem).

Algorithm

Here we describe an algorithm that offered Dutch researcher **Dijkstra** (Dijkstra) in 1959

Zavedëm array $d[]$, in which each vertex $v$will store the current length of $d[v]$the shortest path from $s$to v. Initially $d[s] = 0$, and for all other vertices, this length is equal to infinity (when implemented on a computer usually as infinity simply choose a sufficiently large number, certainly more possible path length):

$$d[v] = \infty, v \neq s$$

Furthermore, for each vertex $v$we store, it is still marked or not, i.e. zavedëm boolean array $u[]$. Initially, all vertices are labeled, ie,

$$u[v] = \text{false}$$

Dijkstra's algorithm itself consists of **iterations** . For the next iteration is chosen vertex with the lowest value of is not marked, ie .:$n\ vd[v]$

$$d[v] = \min_{p:\ u[p]=\text{false}} d[p]$$

(It is understood that on the first iteration will be selected starting vertex s.)

Selected so the top $v$is marked marked. Furthermore, at the current iteration, the vertex v produced **relaxation** : See all edges (v, to)emanating from the vertex v, and for each such vertex toalgorithm tries to improve the value $d[to]$. Suppose the length of this edge is lenthen in the form of relaxation of the code looks like this:

$$d[to] = \min(d[to], d[v] + \text{len})$$

At the current iteration is finished, the flow proceeds to the next iteration (again selected vertex with the smallest value d, out of it produced relaxation, etc.). In the end, after n iterations, all vertices will be labeled, and the algorithm completes its work. It is argued that the values found $d[v]$are the desired length of the shortest paths $s$in v.

It is worth noting that, if not all the vertices reachable from the vertex s, then the values $d[v]$for them will remain endless. It is clear that the last few iterations of the algorithm will just choose these peaks, but no useful work to produce these iterations are not (as an infinite distance can not prorelaksirovat others, even the same infinite distance). Therefore, the algorithm can be immediately stopped as soon as the selected vertex is taken from the top of an infinite distance.

Restoration paths . Of course, you usually need to know not only the length of the shortest paths, but also to get yourself the way. We show how to preserve the information sufficient for restoration of the shortest path $s$to each vertex. It's enough so-called **array of ancestors** : the array $p[]$, in which each vertex $v \neq s$is stored number of the vertex $p[v]$, which is the penultimate in the fast track to the top v. Here we use the fact that if we take the shortest path to some vertex v, and then remove from the last vertex of the path, you get a way, ending a vertex $p[v]$, and this will be the shortest path for the top $p[v]$. So, if we have this array of ancestors, then the shortest path can be restored to him, each time taking a ancestor of the current node, until we shall come to the starting vertex s- so we get the desired shortest path, but written in reverse order. Thus, the shortest path $P$to the top $v$is equal to:

$$P = (s, \dots, p[p[p[v]]], p[p[v]], p[v], v)$$

It remains to understand how to build the array of ancestors. However, this is very simple: at every successful relaxation, ie when the selected vertex $v$is an improvement of the distance to a vertex to, we record that the ancestor of the vertex $to$is the vertex v:

$$p[to] = v$$

Proof

The main assertion , which is based on the correctness of Dijkstra's algorithm is as follows. It is alleged that after some vertex $v$becomes marked, the current distance to it $d[v]$is already the shortest, and therefore more will not change.

The proof will produce by induction. For the first iteration of its validity is obvious - for the top $s$we have $d[s] = 0$, which is the length of the shortest path to it. Suppose now that it holds for all previous iterations, ie all already labeled vertices; prove that it is not broken after the current iteration. Let v- vertex selected in the current iteration, ie vertex, which is going to mark the algorithm. We prove that $d[v]$indeed is the length of the shortest path to it (we denote this length through $l[v]$).

Consider the shortest path $P$to the top v. Clearly, this path can be divided into two ways: P_1 consisting only of vertices labeled (at least the starting vertex $s$is in the way) and the rest of the way P_2(it may also include the marked vertex, but certainly begins with untagged). We denote $p$the first vertex of the path P_2, and through q- the last point of the path P_1.

We first prove our claim for the top p, ie, prove equality $d[p] = l[p]$. However, it is almost obvious: after all, one of the previous iterations, we chose the top $q$and perform the relaxation of it. Since (by virtue of the choice of the vertex p) the shortest path to $p$the shortest path is to $q$plus edge (p, q), then if the relaxation of $q$the quantity $d[p]$actually be set to the desired value.

Due to the non-negativity values of edges length of the shortest path $l[p]$(and she just proved equal $d[p]$) does not exceed the length of $l[v]$the shortest path to the top v. Given that $l[v] \le d[v]$(because Dijkstra's algorithm could not find a shorter route than it is at all possible), as a result we obtain the relations:

$$d[p] = l[p] \le l[v] \le d[v]$$

On the other hand, since both p, and v- unmarked vertices, so that both the current iteration been chosen peak v, and not the top p, then obtain another inequality:

$$d[p] \ge d[v]$$

From these two inequalities we conclude equality $d[p] = d[v]$, and then found out before we receive and relations:

$$d[v] = l[v]$$

QED.

Implementation

Thus, Dijkstra's algorithm is the $n$iteration, each of which is selected unlabeled vertex with the lowest value $d[v]$, the vertex is marked, and then looked through all the edges emanating from a given vertex, and along each edge is an attempt to improve the value $d[]$at the other end of the edge.

Time of the algorithm consists of:

- nJust search the top with the lowest value $d[v]$of all unlabelled vertices, ie of $O(n)$ vertices
- m time an attempt is made relaxations

At the simplest implementation of these operations on the top search will be spent $O(n)$ operations, and one relaxation - $O(1)$operations, and the final **asymptotic behavior of** the algorithm is:

$$O(n^2 + m)$$

Implementation :

```
const int INF = 1000000000;

int main() {
        int n;
        ... чтение n ...
        vector < vector < pair<int,int> > > g (n);
        ... чтение графа ...
        int s = ...; // стартовая вершина

        vector<int> d (n, INF),  p (n);
        d[s] = 0;
        vector<char> u (n);
        for (int i=0; i<n; ++i) {
                int v = -1;
                for (int j=0; j<n; ++j)
                        if (!u[j] && (v == -1 || d[j] < d[v]))
                                v = j;
```

```
                if (d[v] == INF)
                        break;
                u[v] = true;

                for (size_t j=0; j<g[v].size(); ++j) {
                        int to = g[v][j].first,
                                len = g[v][j].second;
                        if (d[v] + len < d[to]) {
                                d[to] = d[v] + len;
                                p[to] = v;
                        }
                }
        }
}
```

Here the graph g is stored in the form of adjacency lists: for each vertex v list $g[v]$ contains a list of edges emanating from this vertex, ie a list of pairs $\text{pair} < \text{int}, \text{int} >$, where the first element of the pair - the top, which leads the edge, and the second element - the weight of the edge.

After reading infest arrays distances $d[]$, labels $u[]$ and ancestors $p[]$. Then executed n iterations. At each iteration, first is the vertex v having the smallest distance $d[]$ of unlabelled vertices. If the distance to the selected vertex v is equal to infinity, then the algorithm stops. Otherwise, the vertex is marked as tagged, and searched all the edges emanating from a given vertex, and along each edge are performed relaxation. If relaxation is successful (ie, the distance $d[to]$ changes), then the distance is recalculated $d[to]$ and stored ancestor $p[]$.

After all the iterations in the array $d[]$ are the length of the shortest paths to all vertices, and in the array $p[]$ - the ancestors of all vertices (except the home page s). Restore the path to any vertex t in the following way:

```
vector<int> path;
for (int v=t; v!=s; v=p[v])
        path.push_back (v);
path.push_back (s);
reverse (path.begin(), path.end());
```

Literature

- Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein. **Algorithms: Design and Analysis** [2005]
- Edsger Dijkstra. **A Note on Two problems in connexion with Graphs** [1959]

Присоединиться к обсуждению...

Oleg Oshmyan • 2 года назад
> датский исследователь Дейкстра
нидерландский
5 • Ответить • Поделиться ›

e_maxx Модератор ➚ Oleg Oshmyan • 2 года назад
Исправлено.
1 • Ответить • Поделиться ›

Good ➚ e_maxx • 2 месяца назад
Вообще-то датский. Вот пруф: http://www.britannica.com/EBch...
• Ответить • Поделиться ›

Любитель пруфов ➚ Good • 2 месяца назад
как раз по вашей ссылке тащемта написано, что он голландский (dutch)
• Ответить • Поделиться ›

Emilbek Sulaymanov • 2 года назад
задача есть на acmp.ru http://acmp.ru/index.asp?main=...
4 • Ответить • Поделиться ›

miga • год назад
А вот строчку

if (d[v] == INF)
Не надо ли поправить на
if (v==-1 || d[v] == INF)
3 • Ответить • Поделиться ›

Людмила • 2 года назад
if (!u[j] && (v == -1 || d[j] < d[v])) опишите эту строчку ни как не могу разобраться(
2 • Ответить • Поделиться ›

e_maxx Модератор ➚ Людмила • 2 года назад
Если текущая вершина j не была ещё посещена, то мы её присваиваем в v в одном из двух случаев: либо v до сих пор содержит пустое значение (-1), либо v содержит вершину с бОльшим расстоянием, чем вершина j. Таким образом, после цикла в вершине v окажется вершина, которая не была ещё посещена (т.е. u[v] = false), а среди всех таких - с минимальным расстоянием d.
3 • Ответить • Поделиться ›

людмила ➚ e_maxx • 2 года назад
Спасибо
1 • Ответить • Поделиться ›

Snigir • 2 года назад
vector<char> u (n);

Можно было бы переименовать для наглядности в used или isUsed. Аналогично и для других переменных.
Больше спасибо. Лучшее изложение материала в Рунете.

2 ^ | v • Ответить • Поделиться ›

e_maxx Модератор ➚ Snigir • 2 года назад

Спасибо. Названия переменных я решил сделать совпадающими с обозначениями в формулах в тексте - а их, в свою очередь, я выбрал короткими, чтобы не загромождать формулы. Надо будет ещё подумать над этим.

1 ^ | v • Ответить • Поделиться ›

Сергей • 2 года назад

if (!u[j] && (v == -1 || d[j] < d[v])) в этой строчке пока вершина J не помечена и (v равно-1 или путь J<пути до v как то так?
подскажите пожалуйста по проще) а то что- ступор и зачем вершине v=-1 этого тоже не понял

1 ^ | v • Ответить • Поделиться ›

сергей • 2 года назад

зачем вершине в присваиваеться -1?

1 ^ | v • Ответить • Поделиться ›

e_maxx Модератор ➚ сергей • 2 года назад

Чтобы как-то отметить тот факт, что вначале v никак не выбрана. Мы не можем положить её равной, к примеру, нулю, потому что нулевая вершина уже могла стать посещённой (u[0] = true), а нам надо выбрать именно непосещённую.

1 ^ | v • Ответить • Поделиться ›

сергей ➚ e_maxx • 2 года назад

спасибо огромное)

1 ^ | v • Ответить • Поделиться ›

Bam • 2 года назад

Спасибо автору) я кажется закроюсь нормально ...

1 ^ | v • Ответить • Поделиться ›

Rolion • 2 года назад

vector<char> u (n);
почему char?

1 ^ | v • Ответить • Поделиться ›

e_maxx Модератор ➚ Rolion • 2 года назад

В смысле, почему не

```
vector<bool>
```

?

Потому что последний работает гораздо медленнее: он использует

побитовое сжатие для хранения своих значений.

1 • Ответить • Поделиться ›

Ignat Loskutov ➚ e_maxx • год назад

А почему не

```
bitset <n>
```

?

• Ответить • Поделиться ›

e_maxx Модератор ➚ Ignat Loskutov • год назад

По той же самой причине: он использует битовое сжатие, а потому гораздо медленнее простого массива/вектора байтов.

1 • Ответить • Поделиться ›

артур • 2 года назад

int to = g[v][j].first,
len = g[v][j].second;
if (d[v] + len < d[to]) {
d[to] = d[v] + len;
p[to] = v;
}опишите пожалуйста эту часть кода

• Ответить • Поделиться ›

Rasul • 2 года назад

Что такое

vector < vector < pair<int,int> > > g (n); ?

• Ответить • Поделиться ›