added: 10 Jun 2008 19:17
Edited: 25 May 2012 14:18

Breadth-first search

Contents [hide]

- Breadth-first search
  - Description of the algorithm
  - Implementation
  - Application of the algorithm
  - Problem in online judges

Breadth-first search (bypassing the width, breadth-first search) - this is one of the basic graph algorithms.

As a result of breadth-first search is the shortest path length in the unweighted graph, ie path that contains the smallest number of edges.

The algorithm works for $O(n + m)$where n- number of vertices m- number of edges.

Description of the algorithm

The input to the algorithm is fed the given graph (unweighted), and number of starting vertex s. A graph can be either oriented or unoriented, for the algorithm is not important.

The algorithm itself can be understood as a process of "ignition" of the graph: at the zero step ignite only the tip s. At each step the fire already burning with each vertex spreads to all its neighbors; ie in one iteration of the algorithm is an expansion of the "ring of fire" in width by one (hence the name of the algorithm).

More precisely, this can be represented as follows. Let's create a place $q$in which to fit the top of the burning, and zavedём boolean array $\text{used}[]$, in which each vertex will be celebrating, she was lit or not (or in other words, whether she had).

Initially, only the top of the queue is put s, and $\text{used}[s] = \text{true}$, as for all the other vertices $\text{used}[] = \text{false}$. Then the algorithm is a loop: while queue is not empty, get out of her head one node to view all the edges emanating from this vertex, and if some of the more viewed vertices are off, then set them on fire and put it in the queue.

As a result, when the queue is empty, bypassing the width will visit all reachable from $s$the vertex, with each reaches to the shortest path. It is also possible to calculate the length of the shortest paths (which just need to have an array of lengths of paths $d[]$), and compact to save enough information to restore all of the shortest paths (that is, to have an array of "ancestors" $p[]$, in which each vertex to store number of the vertex at which we got to the top of this).

Implementation

Implement the above algorithm in C ++.

Input:

```
vector < vector<int> > g; // граф
int n; // число вершин
```

```
int s; // стартовая вершина (вершины везде нумеруются с нуля)

// чтение графа
...
```

Sam bypass:

```
queue<int> q;
q.push (s);
vector<bool> used (n);
vector<int> d (n), p (n);
used[s] = true;
p[s] = -1;
while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (size_t i=0; i<g[v].size(); ++i) {
                int to = g[v][i];
                if (!used[to]) {
                        used[to] = true;
                        q.push (to);
                        d[to] = d[v] + 1;
                        p[to] = v;
                }
        }
}
```

Now, if you need to restore and display the shortest path to some vertex to, it can be done as follows:

```
if (!used[to])
        cout << "No path!";
else {
        vector<int> path;
        for (int v=to; v!=-1; v=p[v])
                path.push_back (v);
        reverse (path.begin(), path.end());
        cout << "Path: ";
        for (size_t i=0; i<path.size(); ++i)
                cout << path[i] + 1 << " ";
}
```

Application of the algorithm

- Find **the shortest path** to the unweighted graph.
- Search **the connected components** in the graph for $O(n + m)$.

  To do this, we simply run a detour to a width of each vertex, except vertices remaining visited ($used = true$) after the previous runs. Thus, we perform normal start with a width of each vertex, but do not reset each time the array $used[]$, due to which we are every time we get a new connected component, and the total time of the algorithm will continue $O(n + m)$(such multiple runs on a graph traversal without zeroing array

usedcalled a series of rounds in width).

- Finding a solution to a problem (the game) **with the smallest number of moves** , if each state of the system can be represented by a vertex of the graph, and the transitions from one state to another - edges of the graph.

  A classic example - a game where the robot moves along the field, while it can move boxes that are on the same field, and requires for the least number of moves required to move boxes into position. Solved is a preorder traversal of the graph, where the state (vertex) is a set of coordinates: the coordinates of the robot, and the coordinates of all the boxes.

- Finding the shortest path in the **graph of 0-1** (ie, weighted graph, but with weights equal only 0 or 1): it is enough to slightly modify the breadth-first search, if the current edge of zero weight, and an improvement of the distance to some vertex, that this peak does not add to the end, and in the beginning of the queue.

- Finding **the shortest cycle** in a directed unweighted graph: make breadth-first search from each vertex; once during the traversal we try to go from the current vertex on some edge to an already visited vertex, then it means that we have found the shortest cycle and stop crawling in width; found among all such cycles (one from each run bypass) choose the shortest.

- Find all edges that lie **on any shortest path** between a given pair of vertices (a, b). To do this, run two breadth-first search: from a, to and from b. Denote $d_a[]$the array shortest distances obtained from the first bypass and through $d_b[]$- in a second bypass. Now for any edge $(u, v)$is easy to check whether it is in any shortest path: the criterion is a condition $d_a[u] + 1 + d_b[v] = d_a[b]$.

- Find all vertices lying **on some shortest path** between a given pair of vertices (a, b). To do this, run two breadth-first search: from a, to and from b. Denote $d_a[]$the array shortest distances obtained from the first bypass and through $d_b[]$- in a second bypass. Now, for each vertex $v$is easy to check whether it is in any shortest path: the criterion is a condition $d_a[v] + d_b[v] = d_a[b]$.

- Find **the shortest way to an even** in the graph (ie, the path of even length). For this we need to build an auxiliary graph, whose vertices are the state $(v, c)$where v- number of current peaks $c = 0 \dots 1$- the current parity. Each edge $(a, b)$of the original graph in this new graph is transformed into two ribs $((u, 0), (v, 1))$and $((u, 1), (v, 0))$. After that, it is necessary to bypass the column width to find the shortest path from the starting vertex to the terminal, with parity of 0.

Problem in online judges

List of tasks that can be taken using the bypass in width:

- SGU # 213 **"Strong Defence"** [Difficulty: Medium]

Лучшее вначале Поделиться Избранный

Присоединиться к обсуждению...

Ramil6085 • 2 года назад

Есть небольшая опечатка в начале статьи: написано "невзвешененном", надо написать "невзвешенном"

15 • Ответить • Поделиться ›

e_maxx Модератор ➚ Ramil6085 • 2 года назад

Исправлено, спасибо!

18 • Ответить • Поделиться ›

Ramil6085 ➚ e_maxx • 2 года назад

Вам спасибо - за замечательный сайт

27 • Ответить • Поделиться ›

Pavel Shevchuk • 2 года назад

Возможно, имеет смысл написать vector<char> used; вместо vector<bool>used;
P.S. Как мне убрать это:

7 • Ответить • Поделиться ›

loboq ➚ Pavel Shevchuk • год назад

фигню не неси

• Ответить • Поделиться ›

danyatinka • 2 года назад

Непонятно, зачем здесь массив d? Он здесь никак не используется. И хотелось бы больше комментариев в коде, а то очень трудно понять, что имеется в виду под той или иной операцией

5 • Ответить • Поделиться ›

Папков Никита ➚ danyatinka • год назад

Используется - d[to] = d[v] + 1;
В массив d записывается кол-во ребер в пути до какой-либо вершины.

3 • Ответить • Поделиться ›

Охтеров Егор • год назад

http://alenacpp.blogspot.ru/20... - Чем плох vector< bool >.

3 • Ответить • Поделиться ›

Cupper • год назад

Вы бы хоть указали что у вас граф g задан в виде вектора смежности (или как оно там называется)

2 • Ответить • Поделиться ›

лол ➚ Cupper • год назад

лол

4 • Ответить • Поделиться ›

Pro100vlad • 2 года назад

12355446699877889

5 • Ответить • Поделиться ›

Dagtom ➦ Pro100vlad • 2 года назад

Idiot!

19 • Ответить • Поделиться ›

stepanovep • 9 месяцев назад

Имейте в виду, что здесь матрица задается не матрицей смежности, а списком, где в i-ой строке выписаны номера вершин, с которыми соединена вершина i.

1 • Ответить • Поделиться ›

Vadim • 2 года назад

В строке с созданием булевого массива посещенных вершин я бы заполнил его false: vector<bool> used (n, false);

1 • Ответить • Поделиться ›

Aisultan • 2 года назад

А как считать двухмерный вектор?

1 • Ответить • Поделиться ›

stepanovep ➦ Aisultan • 9 месяцев назад

int a;
for (int i=0; i<n; ++i){="" for="" (int="" j="0;" j<n;="" ++j){="" cin="">> a;
g[i].push_back(a);
}
}

• Ответить • Поделиться ›

Erop • 3 месяца назад

А можно ли найти кратчайший цикл в НЕОРИЕНТИРОВАННОМ невзвешенном графе алгоритмом поиска в ширину?

• Ответить • Поделиться ›

Joynal • 4 месяца назад

Mr. @e_maxx

I try to find shortest with bfs. but i don't understand where is my problem. My code produce compiler error.
please check this http://paste.ubuntu.com/718916...

• Ответить • Поделиться ›

ExeKiller • 7 месяцев назад

Огромное спасибо за статью!

• Ответить • Поделиться ›

vladsv • 8 месяцев назад

Граф хранится списком смежных вершин?

• Ответить • Поделиться ›