Breadth First Search Algorithm using C++ STL

Hello people..! This is a special extension for my discussion on **Breadth First Search (BFS) Algorithm**. Here, I give you the code for the Breadth First Search Algorithm using C++ STL . Well, it makes no sense if the algorithm is using STL if the input graph isn't built by STL..! So, essentially this is the Breadth First Search algorithm designed for my code in **Adjacency List using C++ STL**. A couple of features of this code are –

- The basic input differs from the code in our initial discussion.
- The flag variables is of the type **bool**.

```
/*
 * Breadth First Search
 * Algorithm for Graph
 * implemented using C++ STL
 *
 * Authored by,
 * Vamsi Sangam.
 *
 */

#include <cstdio>
#include <vector>
#include <list>
#include <utility>

using namespace std;

void breadthFirstSearch(vector< list< int > > adjacencyList, int parent[], int level[])
{
    list<int>::iterator itr;
    int i, par, lev;
    bool flag = true;
    //'lev' represents the level to be assigned
    //'par' represents the parent to be assigned
    //'flag' indicates if graph is unexplored or not

    lev = 0;
    level[1] = lev;
    /* We start from node 1
     * So, Node 1 is at level 0
```

```
31          * All immediate neighbours are at
32          * level 1 and so on.
33          */
34
35         while (flag) {
36             flag = false;
37             for (i = 1; i < adjacencyList.size(); ++i) {
38                 if (level[i] == lev) {
39                     flag = true;
40                     itr = adjacencyList[i].begin();
41                     par = i;
42
43                     while (itr != adjacencyList[i].end()) {
44                         if (level[*itr] != -1) {
45                             ++itr;
46                             continue;
47                         }
48
49                         level[*itr] = lev + 1;
50                         parent[*itr] = par;
51                         ++itr;
52                     }
53                 }
54             }
55
56             ++lev;
57         }
58     }
59
60     int main()
61     {
62         int vertices, edges, v1, v2, weight;
63
64         printf("Enter the Number of Vertices -\n");
65         scanf("%d", &vertices);
66
67         printf("Enter the Number of Edges -\n");
68         scanf("%d", &edges);
69
70         // Adjacency List is a vector of lists.
71         vector< list<int> > adjacencyList(vertices + 1);
72
73         printf("Enter the Edges V1 -> V2\n");
74
75         for (int i = 1; i <= edges; ++i) {
76             scanf("%d%d", &v1, &v2);
77
78             // Adding Edges
79             adjacencyList[v1].push_back(v2);
80             adjacencyList[v2].push_back(v1);
81         }
82
83         printf("\nThe Adjacency List-\n");
84         // Printing Adjacency List
85         for (int i = 1; i < adjacencyList.size(); ++i) {
86             printf("adjacencyList[%d] ", i);
87
88             list<int>::iterator itr = adjacencyList[i].begin();
89
90             while (itr != adjacencyList[i].end()) {
91                 printf(" -> %d", *itr);
92                 ++itr;
93             }
94             printf("\n");
95         }
96
97         int parent[vertices + 1];
98         //Each element of Parent Array holds the Node value of its parent
99         int level[vertices + 1];
100        //Each element of Level Array holds the Level value of that node
101
102        for (int i = 0; i <= vertices; ++i) {
103            //Initialising our arrays
104            parent[i] = 0;
105            level[i] = -1;
106        }
107
108        breadthFirstSearch(adjacencyList, parent, level);
109
```

```
110     //Level Array
111     printf("\nLevel and Parent Arrays -\n");
112     for (int i = 1; i <= vertices; ++i) {
113         printf("Level of Node %d is %d, Parent is %d\n", i, level[i], parent[i]);
114     }
115 
116     return 0;
117 }
```

I could have just simply put the algorithm procedure.. But the reason I put the whole code, is that the graph I am using is an unweighted graph… Obviously, BFS needs an unweighted graph..! So, this graph is a little different from my C++ STL implementation of Adjacency List. The difference, is that, each edge is simply an integer corresponding to the vertex 'V' in an edge U → V… It is not a pair… You should've guessed that by now.. But, it doesn't harm to mention… 😛

Feel free to comment if you have any doubts..! Keep practising..! Happy Coding..! 😀

16 thoughts on "Breadth First Search Algorithm using C++ STL"

Pragati

FEBRUARY 10, 2017 AT 11:58 PM

Why is it that you have used stl here?bfs could be implemented in c or even c++ whithout using stl..so why use stl?

REPLY

Vamsi Sangam

FEBRUARY 25, 2017 AT 8:29 AM

Yes it can be done in a number of other ways. I was just providing one implementation here. In my post on BFS, I have given the implementations for other languages.

REPLY

Kasper

APRIL 23, 2016 AT 9:39 PM

I don't see C++ code in your example, though!

REPLY

Vamsi Sangam

APRIL 26, 2016 AT 12:32 PM

Yeah! 😛 … I'm not a much C++ programmer myself but I can't avoid STL either! 😉

REPLY

Shivang Bansal

MARCH 7, 2016 AT 12:47 AM

First of all thanks for such detailed explaination. I now have understood BFS more clearly . I just did not get one thing in this code…
Why does the for loop in the main function runs till less than adjacentList.size() while in the breathFirstSearch function it runs till less than 'equal to' adjacentList.size()?

REPLY

Vamsi Sangam

MARCH 7, 2016 AT 1:09 AM

The loop in the breadthFirstSearch function must run till less than adjacencyList.size()… It was a mistake which doesn't cause much trouble in C.. But it would be an exception in Java… I have corrected the code… Thanks for pointing it out… 🙂

REPLY

Shivang Bansal

MARCH 7, 2016 AT 1:12 AM

No worries 🙂

REPLY

Neil

AUGUST 24, 2015 AT 5:24 AM

Why not pass adjacencyList by const reference in order to avoid copying it?

REPLY

Vamsi Sangam

AUGUST 24, 2015 AT 3:27 PM

I just wanted to keep it simple so that everyone can understand it easily… Yes, it would make a lot of sense to use const reference if the graph is big, so that we can avoid copying… Though modern compilers generally detect this opportunity of optimization, I prefer to do that explicitly… So, you are right, but here, I wanted to keep things simple… 🙂

REPLY

Pingback: Breadth First Search Algorithm

Yogendra

JUNE 27, 2015 AT 3:12 AM

Why don't you use queue?

REPLY

Vamsi Sangam

JUNE 27, 2015 AT 5:45 PM

Well… It does make a lot of sense to use a queue… But I wanted the algorithm to be as simple as possible… But BFS runs better if we use a queue, can't really neglect it.. 😛 .. So, I have added this topic… Check out **Breadth First Search Algorithm using Queue**..! 🙂

REPLY

Yogendra

JUNE 27, 2015 AT 6:17 PM

I read it and I noticed you didn't use STL queue, is there some reason behind it?

REPLY

Vamsi Sangam

JUNE 27, 2015 AT 6:51 PM

Nice question..! 🙂 … I took a little time to think which one would be better… A queue or a list… By default the underlying container of a queue is a dequeue… Personally, I don't fancy the way a dequeue grows (the internal memory allocation)… It is chunks of contiguous data… But we are not interested in the contiguous memory allocation, or benefits of random access… So, the dequeue doesn't offer us anything we are interested in… On the other hand a list grows decently, its a doubly linked list… In C too, traditionally

we implement queues using doubly linked lists… So, the memory allocation thing made me choose a list.
On the other hand, we can specify the underlying container of queue to be a list while instantiation… But I didn't do that either… Call me lazy, but I didn't do that because I didn't want to include another header file when the job can be done decently with the resources I already have… 🙂
You can always use the STL Queue… We can't know which is the best, unless we are given really massive amounts of data… 😉

REPLY

competitivecoder

JUNE 24, 2015 AT 11:01 PM

I stare you blog almost the whole day 🙂 .Btw along with segment tree request(I hope you remember that).If you could make a post on cycle detection using DFS 🙂 That is another hot topic for every programmer 🙂 Just a suggestion and request you can say.And one more thing i would like to know is that have you prepared any list of topics according to which you make your posts ? If yes please let us know

REPLY

Vamsi Sangam

JUNE 25, 2015 AT 12:27 AM

I'm really glad you find my blog useful..! I do remember your request.. 😉 … Yes, cycle detection is an interesting topic.. I just wrote a short note in my post regarding DFS… It would be nice to have the code too..! Thanks for the suggestion.. 🙂
As for the list of topics… Nothing specific like that.. 😛 … I do have a bunch of topics in mind… But the real challenge is to balance my day-to-day curriculum with the blog activities… So it all comes down to priorities… What's easy and effective for me in the given situation.. 🙂
Adjacency List Data Structure, BFS and DFS have been the heart of the traffic to my blog… So, I am boosting it by putting more implementations. Then I'm thinking of starting the Object Oriented Section of my Java Tutorials… The post is half ready… Then I guess your request for Segment Tree problems is on the queue.. 🙂 … After that, I'll stop posting for a while and go through all my codes once again… Because every now-and-then when people ask doubts and I go through my codes, I find a few statements which can be removed to cut-short the code… So I'll be working on optimizing the codes for a while…
Currently that's what I have in mind… 😀 …. Let's see what I actually end up doing..! 😛

REPLY

Leave a Reply

Enter your comment here...

CATEGORIES

Select Category

ARCHIVES

Select Month

SUBSCRIBE TO BLOG VIA EMAIL

Enter your email address to subscribe to this blog and receive notifications of new posts by email.
Join 132 other subscribers

Email Address

Subscribe