Introduction to Algorithms and Data Structures

11. Graph Algorithms (1) Breadth-first search and Depth-first search

Professor Ryuhei Uehara,

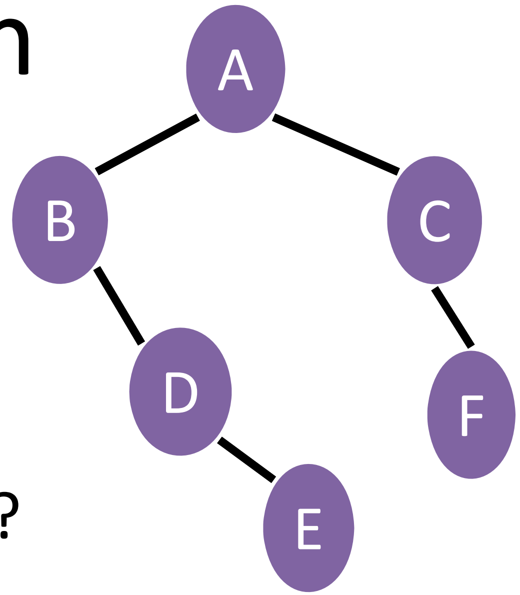
School of Information Science, JAIST, Japan.

uehara@jaist.ac.jp

http://www.jaist.ac.jp/~uehara

http://www.jaist.ac.jp/~uehara/course/2020/myanmar/

Search in Graph



- How can we check all vertices in a graph systematically, and solve some problem?
  - e.g., Do you have a path from A to D?
- Two major (efficient) algorithms:
  - Breadth First Search: A -> B -> C -> D -> F -> E
    it starts from a vertex v, and visit all (reachable) vertices from the vertices closer to v.
  - Depth First Search: A -> B -> D -> E -> C -> F
    it starts from a vertex v, and visit every reachable vertex from the current vertex, and back to the last vertex which has unvisited neighbor.

BFS (Breadth-First Search)

- For a graph G=(V,E) and any start point s∈V, all reachable vertices from s will be visited from s in order of distance from s.
- Outline of method: color all vertices by white, gray, or black as follows;
  - White: Unvisited vertex
  - Gray: It is visited, but it has unvisited neighbors
  - Black: It is already visited, and all neighbors are also visited

  - Search is completed when all vertices got black
  - Color of each vertex is changed as white→gray→black

BFS (Breadth-First Search): Program code

```
BFS(V,E,s){
  for v∈V do toWhite(v); endfor
  toGray(s);
  Q={s};
  while( Q!={} ){
    u=pop(Q); // Q → Q' where Q={u}∪Q'
    for v∈{v∈V|(v,u)∈E}
      if isWhite(v) then
        toGray(v); push(Q,v);
      endif
    endfor
    toBlack(u);
  }
}
```

Queue of gray nodes

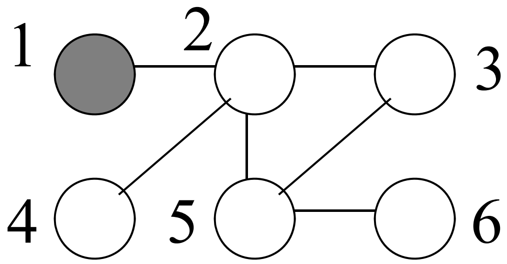
Pop u from left side, which became in gray first

If v, neighbor of u, is white

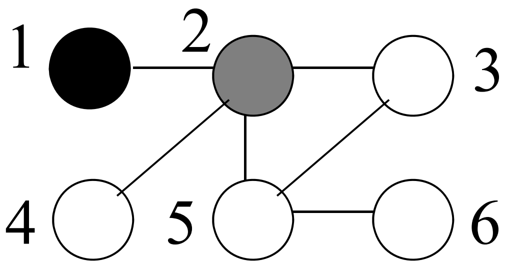
Put v into Q from right, which will be processed last

Make u in black after visiting all neighbors

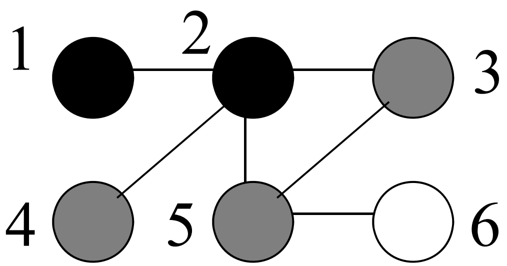
BFS (Breadth-First Search): Example



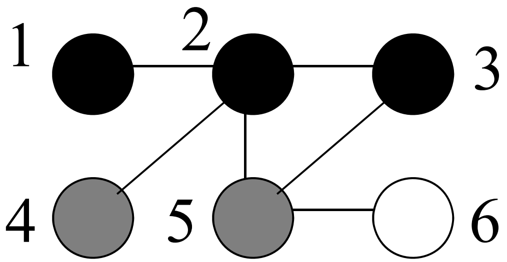
Q={1}



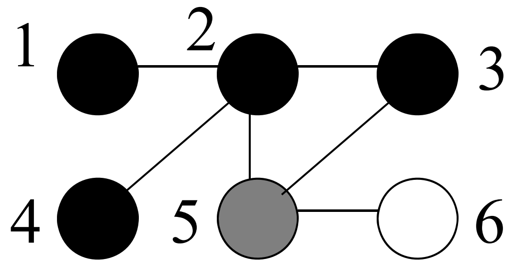
u=1,
visit 2
Q={2}
black 1



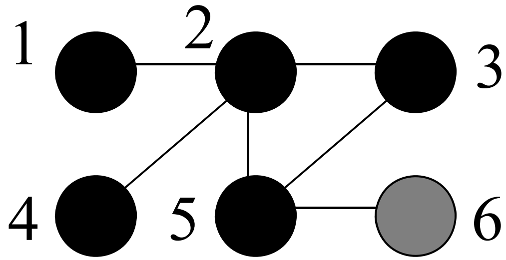
u=2,
visit 3,4,5
Q={3,4,5}
black 2



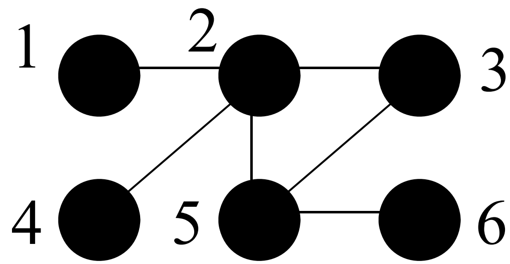
u=3,
visit null
Q={4,5}
black 3



u=4,
visit null
Q={5}
black 4



u=5,
visit 6
Q={6}
black 5



u=6,
visit null
Q={}
black 6

BFS: Time complexity

Time complexity is not easy from program…

Consider from

<u>the viewpoints of vertices and edges</u>

- Each vertex never gets white again after initialization.
- Each vertex gets into Q and gets out from Q at most once
- Each edge is checked at most once
  - when one endpoint vertex is taken from Q and its neighbors are checked along edges
- $\therefore O(|V|+|E|)$

```
BFS(V,E,s){
  for v∈V do
    toWhite(v);
  endfor
  toGray(s);
  Q={s};
  while( Q!={} ){
    u=pop(Q);
    for v∈{v∈V|(v,u)∈E}
      if isWhite(v) then
        toGray(v);
        push(Q,v);
      endif
    endfor
    toBlack(u);
}}
```

Adj. matrix is not good

Real example code for BFS

```
using System.Collections.Generic;
using static System.Console;

public class i111_12_p7 {
    public static void Main(){
        List<int>[] edges = new List<int>[7];
        edges[1] = new List<int>{2};
        edges[2] = new List<int>{1,3,4,5};
        edges[3] = new List<int>{2,5};
        edges[4] = new List<int>{2};
        edges[5] = new List<int>{2,3,6};
        edges[6] = new List<int>{5};

        bfs(6,edges,1);
    }
```

```
    static void bfs(int n, List<int>[] edges, int s) {
        int[] color = new int[n+1];
        color[s] = 1;
        List<int> q = new List<int>();
        q.Add(s);

        while (q.Count > 0) {
            int u = q[0]; q.RemoveAt(0);
            foreach (int v in edges[u]) {
                if (color[v] == 0) {
                    color[v] = 1;
                    q.Add(v);
                }
            }
            color[u] = 2;

            Write("u="+u+", Q={ ");
            foreach (int w in q) Write(w+" ");
            Write("}, color={ ");
            for(int i=1; i<=n; i++) Write(color[i]+" ");
            WriteLine("}");
        }
    }
}
```

Initialize adj. list for each node

Specify the number of nodes and start node

Initialize by 0 = white

Queue is realized by List

Refer the first node and remove it

Output

Application of BFS: Shortest path problem on graph

Definition of “distance”

- Start vertex v has distance 0
- Except start vertex, each vertex u has distance d+1, where d is the distance of parent of u.

- On BFS, modify that each gray vertex receives its “distance” from black neighbor, then you get (shortest) distance from v to it.

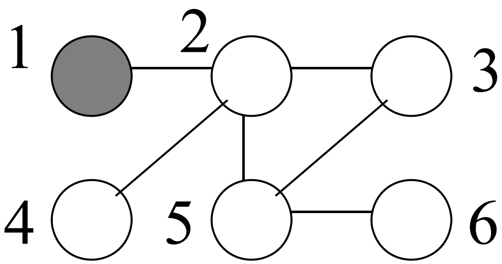
DFS (Depth-First Search)

- For a graph G=(V,E) and start point s∈V, it follows reachable vertices from s until it reaches a vertex that has no unvisited neighbor, and returns to the last vertex that has unvisited neighbors.

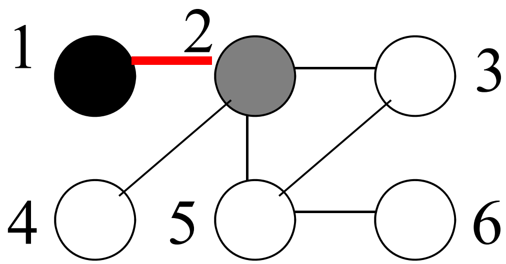
```
dfs(V, E, s) {
   visit(s) // in gray
   for (s, w)∈E do
      if notVisited(w) then
         dfs(V, E, w)          ←recursive call of dfs
   toBlack(u)                  ←make it black after check
}
```

> Program code is relatively simple, and vertices are put into a stack when dfs makes a recursive call.

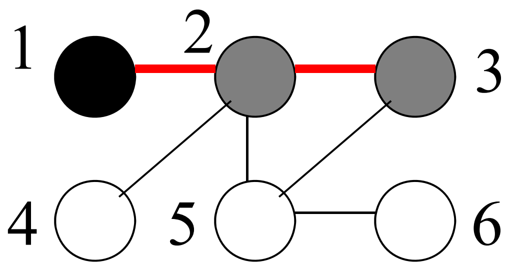
DFS: Example



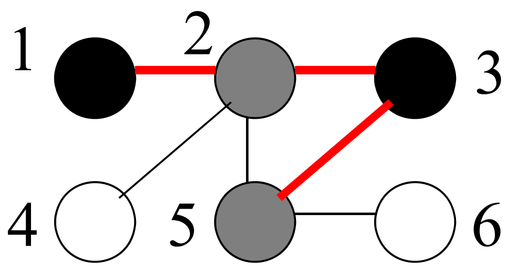
DFS(1)



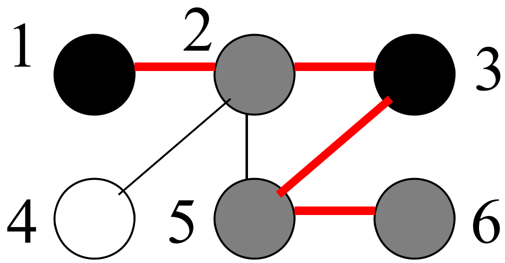
DFS(2)



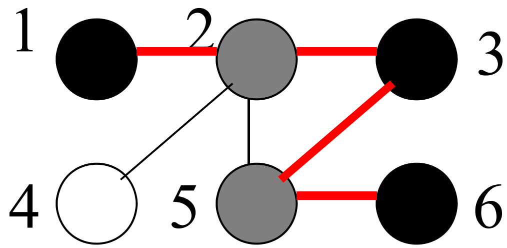
DFS(3)



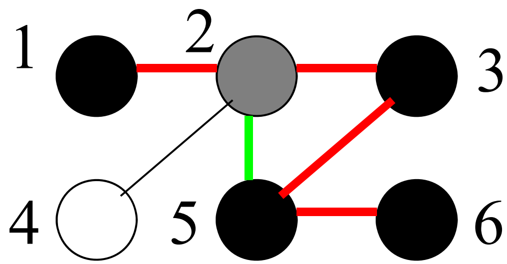
DFS(5)



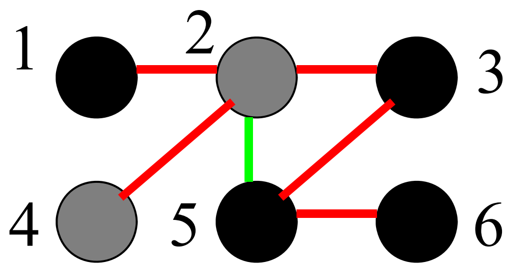
DFS(6)



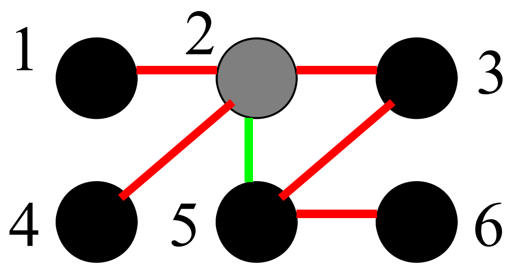
DFS(6)



DFS(5)
DFS(3)
DFS(2)



DFS(4)



DFS(2)

Real example code for DFS (by recursive call)

```
using System.Collections.Generic;
using static System.Console;

public class i111_12_p10 {
    public static void Main(){
        List<int>[] edges = new List<int>[7];
        edges[1] = new List<int>{2};
        edges[2] = new List<int>{1,3,4,5};
        edges[3] = new List<int>{2,5};
        edges[4] = new List<int>{2};
        edges[5] = new List<int>{2,3,6};
        edges[6] = new List<int>{5};

        depth = 0;
        color = new int[7];
        WriteLine("dfs(1)");
        dfs(edges,1);
    }

    static int depth;
    static int[] color;
```

```
    static void dfs(List<int>[] edges, int u) {
        depth ++;
        color[u] = 1;

        foreach (int v in edges[u]) {
            if (color[v]==0) {
                for (int i=0; i<depth; i++) Write("    ");
                WriteLine("->dfs("+v+")");
                dfs(edges, v);
            }
        }
        depth --;
    }
}
```

Initialize adj. list for each node

Specify the start node

Body is quite compact!

Colors are kept outside of function
depth is not required (only for output)

Output

Implementation of BFS
without recursive call; We can use stack

```
DFS(V,E,s){
  for v∈V do toWhite(v); endfor
  toGray(s);
  S={s};
  while( S!={} ){
    u=pop(S);
      for v∈{v∈V|(u,v)∈E}
      if isnotBlack(v) then
        toGray(v); push(S,v);
      endif
    endfor
    toBlack(u);
  }
}
```

Stack of gray nodes

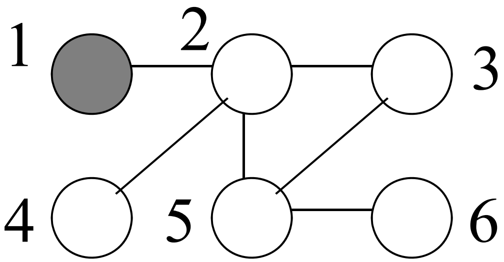
Pop u from the top
which becomes in gray at last

If v, neighbor of u, is not black,

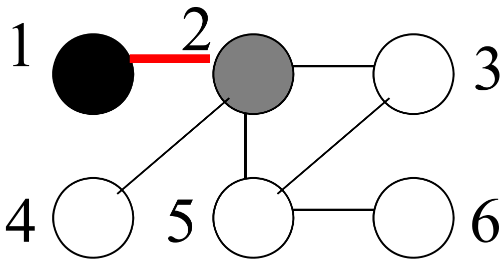
Push v onto S which
will be checked first

Make u in black after visiting all neighbors

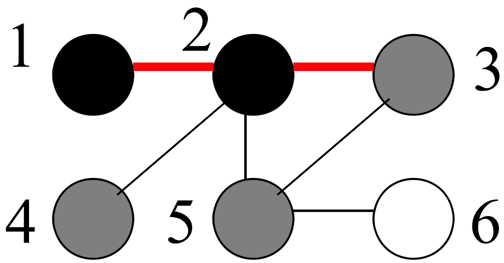
Example of BFS on stack



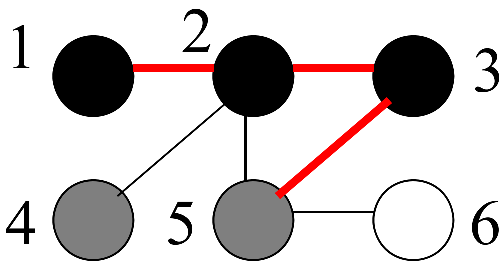
S={1}



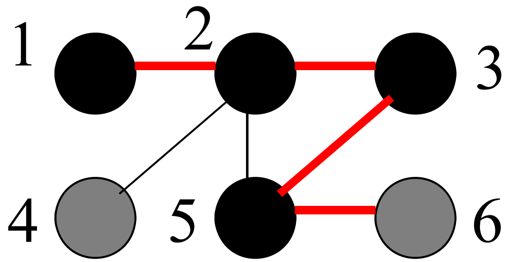
u=1
visit 2
S={2}
black 1



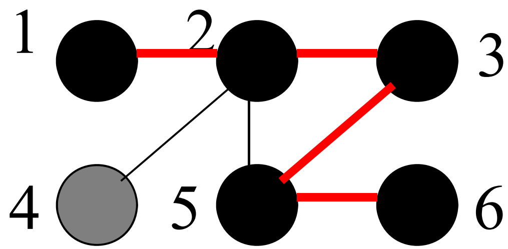
u=2
visit 5,4,3
S={5,4,3}
black 2



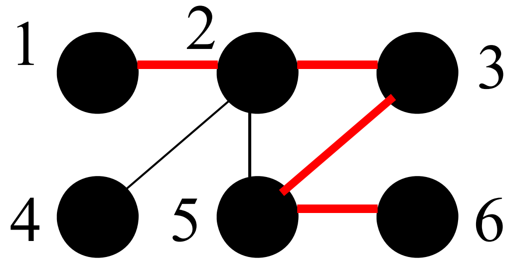
u=3
visit 5
S={5,4,5}
black 3



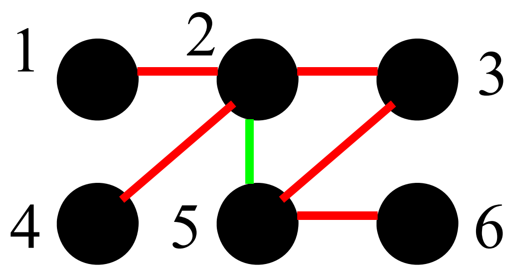
u=5
visit 6
S={5,4,6}
black 5



u=6
visit null
S={5,4}
black 6



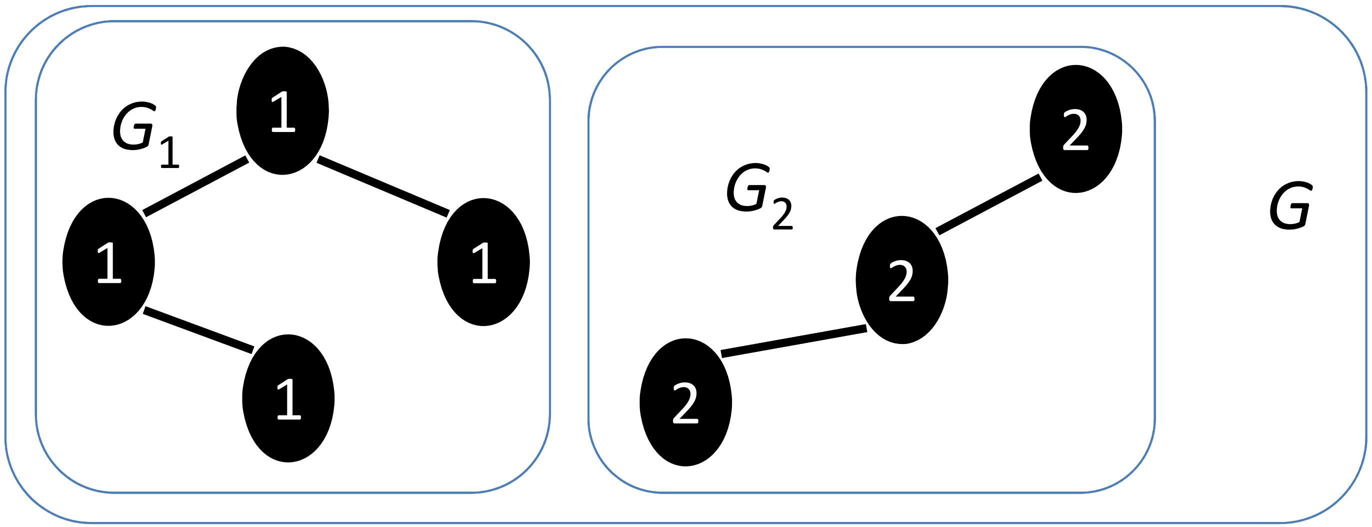
u=4
visit null
S={5}
black 4



u=5
visit null
S={}

Application of DFS: Find connected components in a graph

- For a given (disconnected) graph $G = (V, E)$, divide it into connected graphs $G_1 = (V_1, E_1)$, ..., $G_c = (V_c, E_c)$.
  - We will give a numbering array cn[] such that $\forall u,v \in V,\ u \in V_i \land v \in V_j \land i \neq j \Rightarrow cn[u] \neq cn[v]$



Application of DFS:
Find connected components of a graph

```
cc(V,E,cn){ //cn[|V|]
  for v∈V do
      cn[v] = 0; /*initialize*/
  endfor
  k = 1;
  for v∈V do
    if cn[v]==0 then
      dfs(V,E,v,k,cn);
      k=k+1;
    endif
  endfor
}
```

```
dfs(V,E,v,k,cn){
  cn[v]=k;
  for u∈{u|(v,u)∈E} do
    if cn[u]==0 then
      dfs(V,E,u,k,cn);
    endif
  endfor
}
```

BFS v.s. DFS on a graph (1)

From the viewpoint of algorithms:

Two major efficient & simple search algorithms

- Breadth First Search:

  It corresponds to "Queue"

- Depth First Search:

  It corresponds to "Stack"

- Both algorithms are easy to implement to run in $O(|V|+|E|)$ time. (In a sense, this time complexity is optimal since you have to check all input data.)

BFS v.s. DFS on a graph (2)

From the practical viewpoint

- BFS
  - Advantage: It can find shortest path
  - Disadvantage: It requires memory! (check binary tree of depth n)
- DFS
  - Advantage: It requires few memory (proportional to the depth of the graph)
  - Disadvantage: It may find non-shortest path.

Depending on applications, we choose better algorithm.