Introduction to Algorithms

Department of Computer Science and Engineering

East China University of Science and Technology

Graph searching
- ***BFS***
- ***DFS***
- ***Heuristics Search***

Flow network

Graph algorithms

- Single-source shortest paths
  - Dijkstra Algorithm
  - Bellman-Ford Algorithm
- All-pairs shortest paths
  - Floyd-Warshall Algorithm
  - Janson Algorithm

Graph searching

- 图的基本概念
- Breadth-first search（BFS)
- Depth-First Search(DFS)
- 应用
- ***Heuristics Search***

Breadth-First Search

- Again will associate vertex "colors" to guide the algorithm
  - **White vertices** = discovered
    - All vertices start out white
  - **Grey vertices** = discovered, but not fully explored
    - They may be adjacent to white vertices
  - **Black vertices** = discovered and fully explored
    - They are adjacent only to black and gray vertices
- Explore vertices by scanning adjacency list of grey vertices

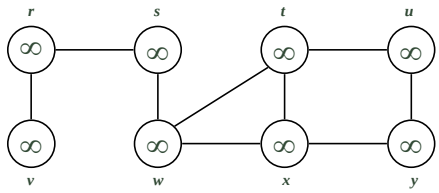
Breadth-First Search

```
BFS(G, s) {
   initialize vertices;
   Q = {s};          // Q is a queue (duh); initialize to s
   while (Q not empty) {
      u = RemoveTop(Q);
      for each v ∈ u->adj {
         if (v->color == WHITE)
            v->color = GREY;
            v->d = u->d + 1;
            v->p = u;
            Enqueue(Q, v);
      }
      u->color = BLACK;
   }}
```

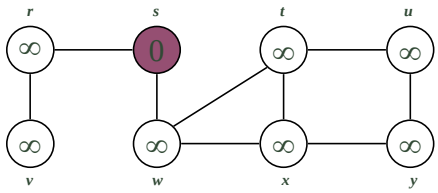
What does `v->d` represent?

What does `v->p` represent?

Breadth-First Search: Example

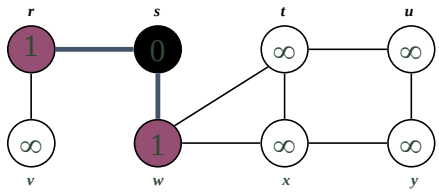


Breadth-First Search: Example



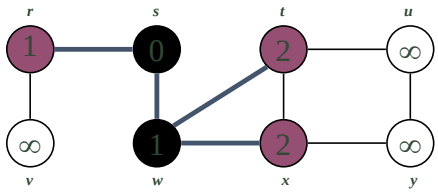
Q: | *s* |

Breadth-First Search: Example



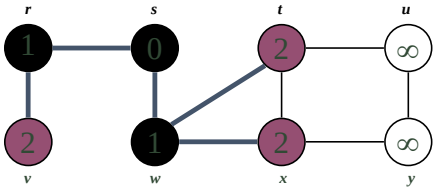
Q: | *w* | *r* |

Breadth-First Search: Example



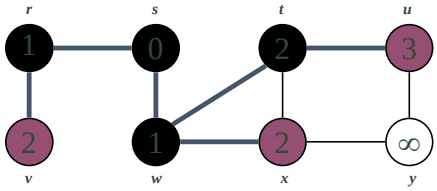
Q: | *r* | *t* | *x* |

Breadth-First Search: Example



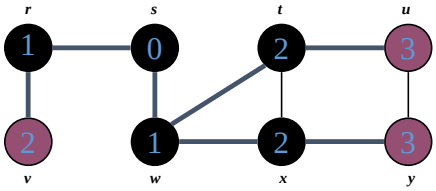
Q: | *t* | *x* | *v* |

Breadth-First Search: Example



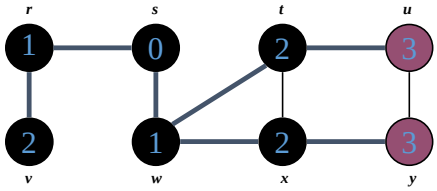
Q: | *x* | *v* | *u* |

Breadth-First Search: Example



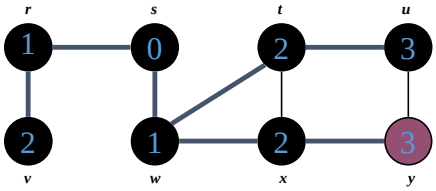
Q: | *v* | *u* | *y* |

Breadth-First Search: Example



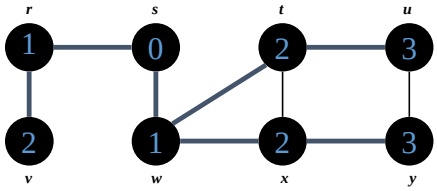
Q: | *u* | *y* |

Breadth-First Search: Example



Q: | *y* |

Breadth-First Search: Example



Q: Ø

BFS: The Code Again

```
BFS(G, s) {
    initialize vertices;
    Q = {s};
    while (Q not empty) {
        u = RemoveTop(Q);
        for each v ∈ u->adj {
            if (v->color == WHITE)
                v->color = GREY;
                v->d = u->d + 1;
                v->p = u;
                Enqueue(Q, v);
        }
        u->color = BLACK;
    }
}
```

Touch every vertex: O(V)

u = every vertex, but only once (Why?)

So v = every vertex that appears in some other vert's adjacency list

What will be the running time?

Total running time: O(V+E)

BFS: The Code Again

```
BFS(G, s) {
    initialize vertices;
    Q = {s};
    while (Q not empty) {
        u = RemoveTop(Q);
        for each v ∈ u->adj {
            if (v->color == WHITE)
                v->color = GREY;
                v->d = u->d + 1;
                v->p = u;
                Enqueue(Q, v);
        }
        u->color = BLACK;
    }
}
```

What will be the storage cost in addition to storing the graph?

Total space used: O(max(degree(v))) = O(E)

Breadth-First Search: Properties

- BFS calculates the *shortest-path distance* to the source node
  - Shortest-path distance $\delta(s,v)$ = minimum number of edges from s to v, or ∞ if v not reachable from s
- BFS builds *breadth-first tree*, in which paths to root represent shortest paths in G
  - Thus can use BFS to calculate shortest path from one vertex to another in O(V+E) time

Depth-First Search

- As DFS progresses, every vertex has a ***color***:

WHITE = undiscovered

GRAY= discovered, but not finished (not done exploring from it)

BLACK = finished (have found everything reachable from it)

- **Input:** *G = (V, E)*, directed or undirected. No source vertex given!
- **Output:** 2 ***timestamps*** on each vertex:

  d[*v*] = ***discovery time***

  f[*v*] = ***finishing time***

Discovery and finish times:

- Unique integers from 1 to 2|*V*|.
- For all *v*, *d*[*v*] < *f* [*v*].

➢$1 \leqslant d[v] < f[v] \leqslant 2|V|$

DFS*(V, E)*

- DFS*(V, E)*
- for each *u* ∈ *V*
- do *color*[*u*] ← WHITE
- *time* ← 0
- for each *u* ∈ *V*
- do if *color*[*u*] = WHITE
- then DFS-VISIT*(u)*

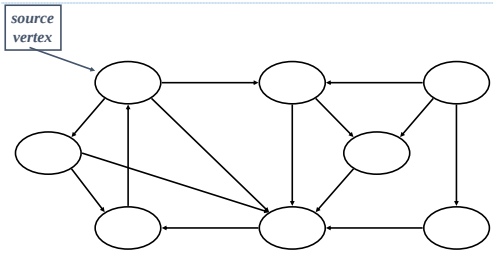
❖O(V+E)

DFS-VISIT*(u)*
color[*u*] ← GRAY — discover *u*
time ← *time*+1
d[*u*] ← *time*
for each *v* ∈ *Adj*[*u*] — explore *(u, v)*
do if *color*[*v*] = WHITE
then DFS-VISIT*(v)*
color[*u*] ← BLACK
time ← *time*+1
f [*u*] ← *time* — finish *u*

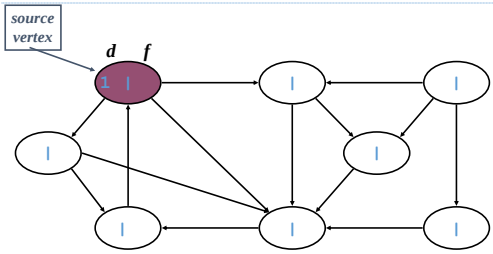
Depth-First Sort Analysis

- This running time argument is an informal example of *amortized analysis*
  - "Charge" the exploration of edge to the edge:
    - Each loop in DFS_Visit can be attributed to an edge in the graph
    - Runs once/edge if directed graph, twice if undirected
    - Thus loop will run in O(E) time, algorithm O(V+E)
      - Considered linear for graph, b/c adj list requires O(V+E) storage
  - Important to be comfortable with this kind of reasoning and analysis

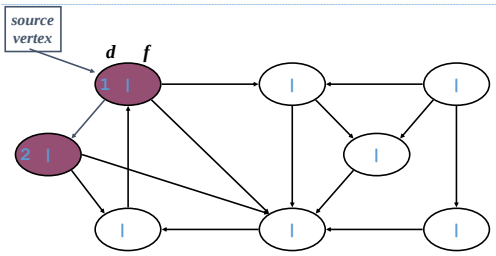
DFS Example



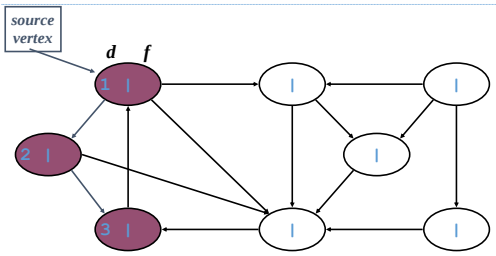
DFS Example



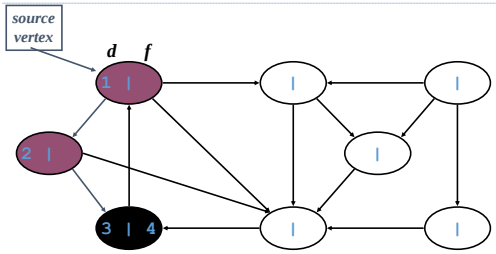
DFS Example



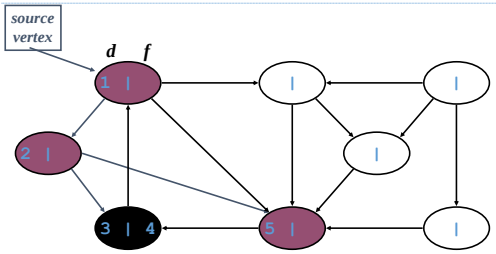
DFS Example



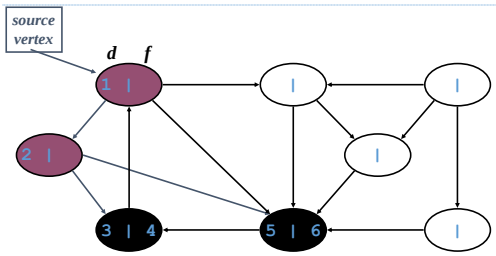
DFS Example



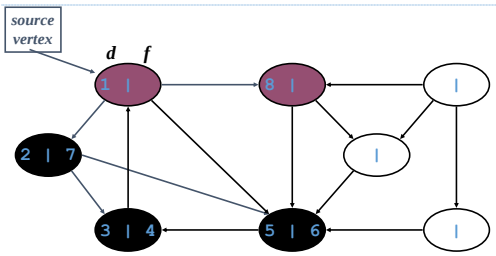
DFS Example



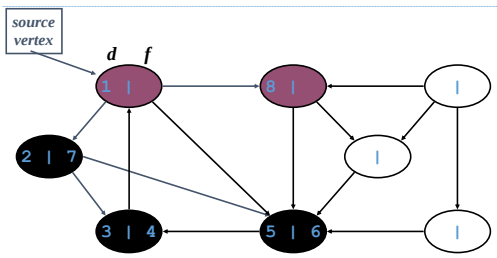
DFS Example



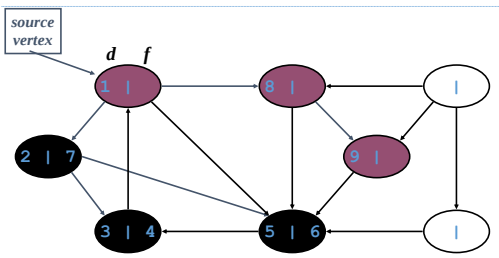
DFS Example



DFS Example

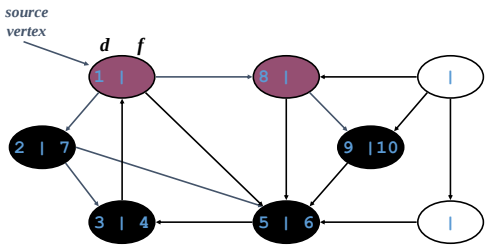


DFS Example

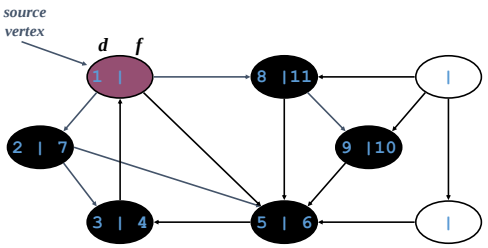


What is the structure of the grey vertices?
What do they represent?

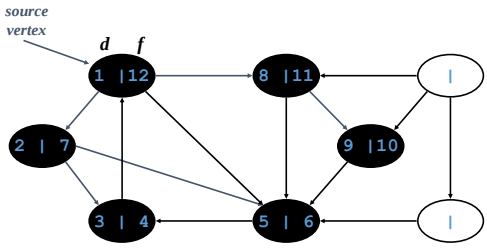
DFS Example



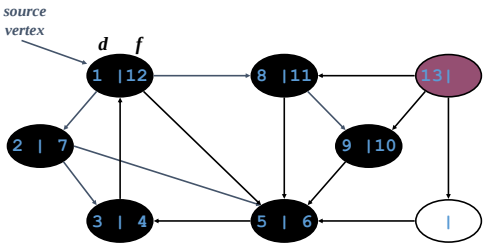
DFS Example



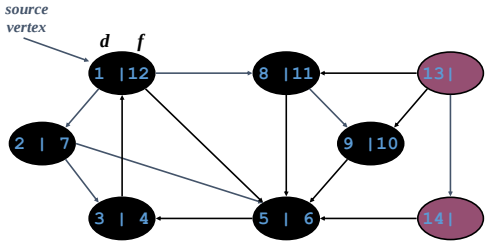
DFS Example



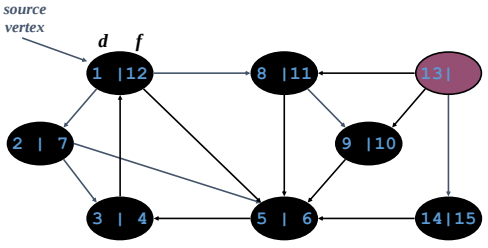
DFS Example



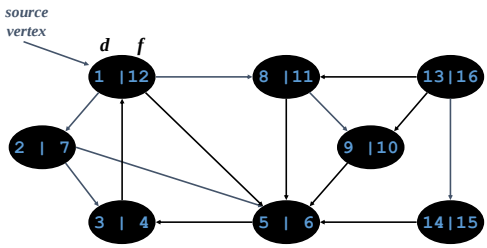
DFS Example



DFS Example



DFS Example



{1[2(3 4)(5 6)7][8(9 10)11]12}{13(14 15)16}