ECS 122A

Lecture 10
4/30/2020

Notion of graphs

Basic terminology

- Graph $G = (V, E)$:
  - $V = \{v_i\}$ = set of vertices
  - E = set of edges = a subset of $\underline{V \times V = \{(v_i, v_j)\}}$
- $|E| = O(|V|^2)$
  - dense graph: $|E| \approx |V|^2$
  - sparse graph: $|E| \approx |V|$
- Some variants
  - undirected: edge $(u, v) = (v, u)$
  - directed: (u, v) is edge from u to v.
  - weighted: weight on either edge or vertex
  - multigraph: multiple edges between vertices
- Reading: Appendix B.4, pp.1168-1172 of [CLRS,3rd ed.]

Notion of graphs

Representing a graph by an Adjacency Matrix

- $A = (a_{ij})$ is a $|V| \times |V|$ matrix, where

$$a_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$

- If G is undirected, A is symmetric, i.e., $A^T = A$.
- A is typically very sparse
  use a sparse storage scheme in practice

Notion of graphs

Representing a graph by an Incidence Matrix

- $B = (b_{ij})$ is a $|V| \times |E|$ matrix, where

$$b_{ij} = \begin{cases} 1, & \text{if edge } e_j \text{ enters vertex } v_i \\ -1, & \text{if edge } e_j \text{ leaves vertex } v_i \\ 0, & \text{otherwise} \end{cases}$$

Notion of graphs

Representing a graph by an Adjacency List

- For each vertex v,

$$\text{Adj}[v] = \{ \text{ vertices adjacent to } v \}$$

- Variation: could also keep second list of edges coming into vertex.
- How much storage is needed?
  Answer: $\Theta(|V| + |E|)$ ("sparse representation")

Notion of graphs

Degree of a vertex

- undirected graph:
  - The degree of a vertex $=$ the number of incident edges
  - total $\#$ of items in the adj. list $= \sum_{v \in V} \text{degree}(V) = 2|E|$
- directed graph (digraph):
  - out-degree and in-degree
  - total $\#$ of items in the adj. list $= \sum_{v \in V} \text{out-degree}(V) = |E|$

Review: queue and stack data structure

- Queues and stacks are dynamic sets in which the elements removed from the set is prescribed.
- The queue implements a First-In-First-Out (FIFO) policy.
  The stack implements a Last-In-First-Out (LIFO) policy.
- Queue supports the following operations:

Q.Enqueue(x) -- pushes X on into last place in the queue
Q.Dequeue-- returns and removes the first element of the queue

See chapter 10.1 for implementation details

Breadth-First Search (BFS)

- An archetype for many important graph algorithms

Shortest!!!

- Input: Given $G = (V, E)$ and a source vertex s,

  Output: $d[v] =$ distance from s to v for all $v \in V$.
- distance $=$ fewest number of edges $=$ shortest path
- BFS basic idea:
  - discovers all vertices at distance k from the source vertex before discovering any vertices at distance $k+1$
  - or expanding frontier – "greedy" – propagate a wave 1 edge-distance at a time.

Breadth-First Search (BFS)

```
BFS(G,s)
for each vertex u in V-{s}
   d[u] = +infty
endfor
d[s] = 0
Q = empty  // create FIFO queue
Enqueue(Q, s)
while Q not empty
   u = Dequeue(Q)
   for each v in Adj[u]
       if d[v] = +infty,
          d[v] = d[u] + 1
          Enqueue(Q, v)
       endif
   endfor
endwhile
return d
```

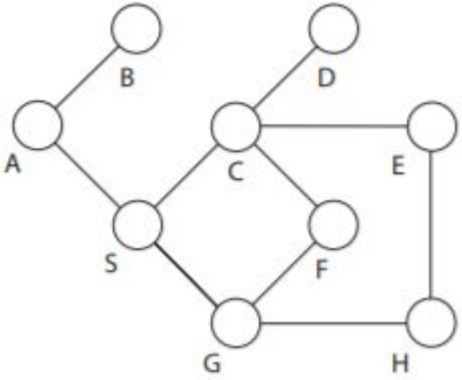
- Breadth-First spanning tree
- Running time: $O(|V| + |E|)$

  $O(|V|)$: every vertex enqueued at most once

  $O(|E|)$: every vertex dequeued at most once and we examine (u, v) only when u is dequeued at most once if directed, at most twice if undirected.

  Note: not $\Theta(|V| + |E|)$!

- Correctness of BFS
  shortest path proof – see pp.597-600 of [CLRS,3rd ed.]
  similar with weighted edges – Dijkstra's algorithm – *to be discussed*



Take S to be the source vertex.

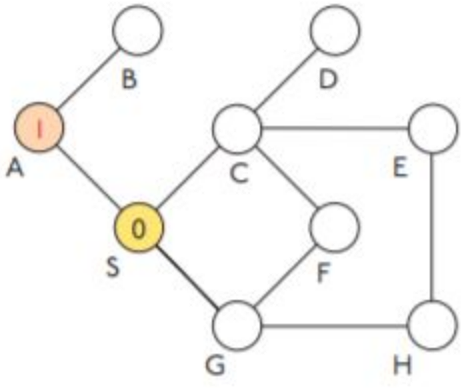
Adjacency Lists:

$$\begin{aligned} A_S &= \{A, C, G\} \\ A_A &= \{B, S\} \\ A_B &= \{A\} \\ A_C &= \{D, E, F, S\} \\ A_D &= \{C\} \\ A_E &= \{C, H\} \\ A_F &= \{C, G\} \\ A_G &= \{F, H, S\} \\ A_H &= \{E, G\} \end{aligned}$$

1. *Set things up:*
   $d(v) \leftarrow \infty$ for all $v \neq s \in V$
   $d(s) \leftarrow 0$ and $Q \leftarrow \{s\}$

❷ *Main loop: continues until the queue is empty*

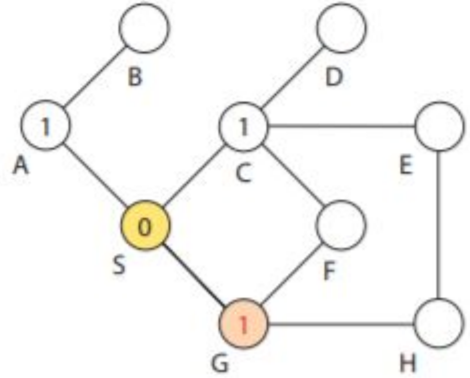
```
While ( Q ≠ ∅ ) {
        Pop a vertex v off the left end of Q.

        Examine each of v's neighbours
        For each  w ∈ A_v {
            If(  d(w) = ∞  ) then {
                Set d(w) and get ready to process w's neighbours
                d(w) ← d(v) + 1
                Push w on to the right end of Q.
            }
        }
}
```



v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$

Yellow vertex is v, red is w.

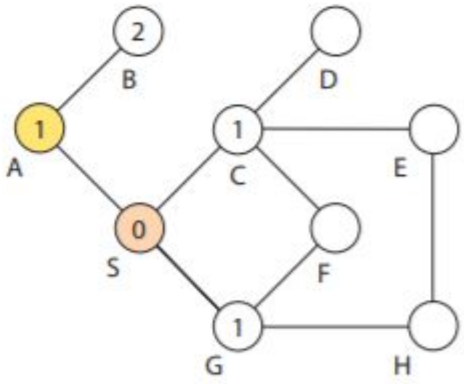
$$A_v = A_S = \{A, C, G\}$$



v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$

Yellow vertex is v, red is w.

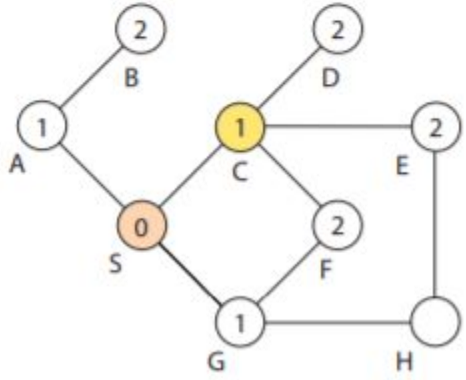
$$A_v = A_S = \{A, C, G\}$$



v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$

Yellow vertex is v, red is w.

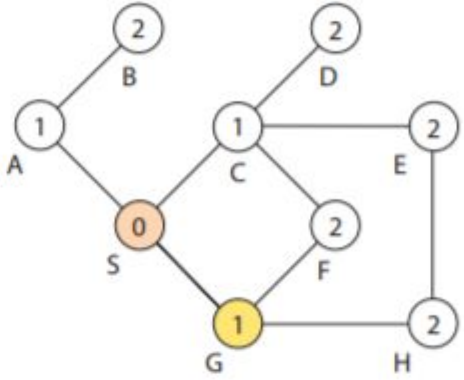
$$A_v = A_A = \{B, S\}$$



v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$

Yellow vertex is v, red is w.

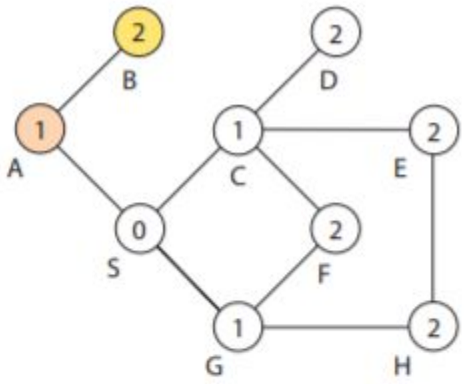
$$A_v = A_C = \{D, E, F, S\}$$



v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$

Yellow vertex is v, red is w.

$$A_v = A_G = \{F, H, S\}$$

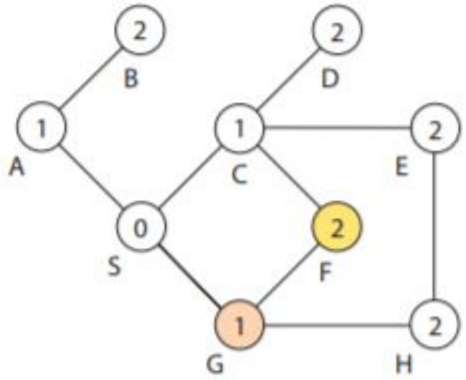


Yellow vertex is v, red is w.

$$A_v = A_B = \{A\}$$

v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$
B	A	none	$\{D, E, F, H\}$

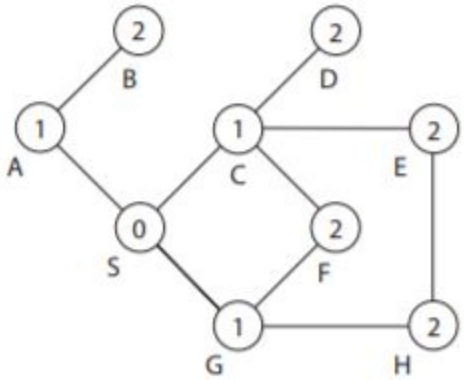
...



Yellow vertex is v, red is w.

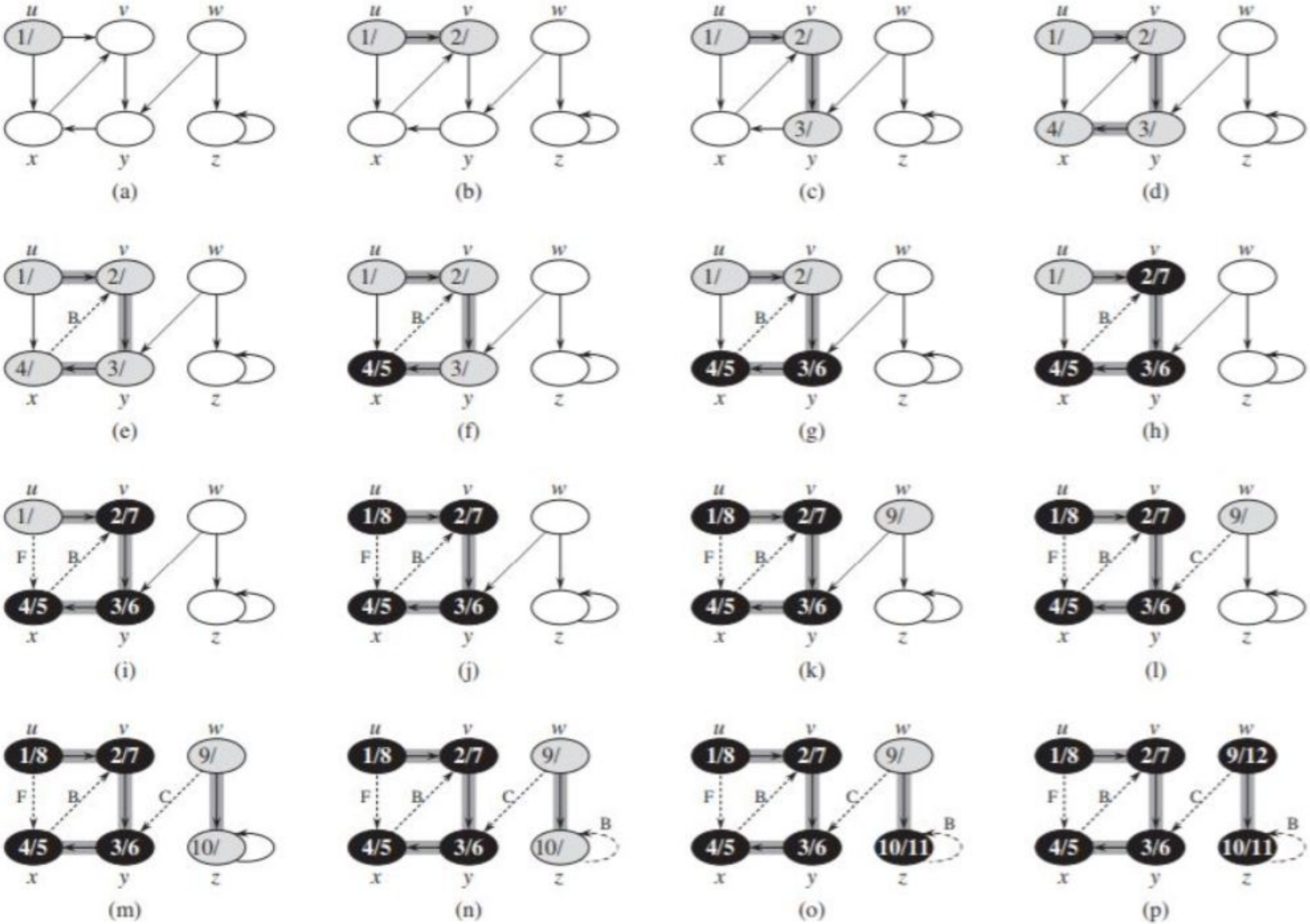
$$A_v = A_F = \{C, G\}$$

v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$
B	A	none	$\{D, E, F, H\}$
D	C	none	$\{E, F, H\}$
E	C	none	$\{F, H\}$
E	H	none	$\{F, H\}$
F	C	none	$\{H\}$
F	G	none	$\{H\}$



v	w	Action	Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$	$\{A\}$
S	C	set $d(C) = 1$	$\{A, C\}$
S	G	set $d(G) = 1$	$\{A, C, G\}$
A	B	set $d(B) = 2$	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$	$\{G, B, D\}$
C	E	set $d(E) = 2$	$\{G, B, D, E\}$
C	F	set $d(F) = 2$	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$
B	A	none	$\{D, E, F, H\}$
D	C	none	$\{E, F, H\}$
E	C	none	$\{F, H\}$
E	H	none	$\{F, H\}$
F	C	none	$\{H\}$
F	G	none	$\{H\}$
H	E	none	$\{\}$
H	G	none	$\{\}$

```
1  procedure DFS(startV: a vertex of G)
2  Initialize empty stack toInspect
3  toInspect.push(startV)
4  Initialized bool array visitedSet of size |V|
5  for ( i := 1 to n - 1 )
6    visitedSet[i] := false
7
8  while ( toInspect is not empty )
9    // Get top of stack & remove from stack
10   v := toInspect.pop()
11   if (visitedSet[v] = false)
12     visitedSet[v] := true
13     for each u in N(v)
14       // It could be u is already visited,
15       // check this on line 11 when u is popped
16       toInspect.push(u)
```


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
(j)
(k)
(l)
(m)
(n)
(o)
(p)
u
v
w
x
y
z
1/
2/
3/
4/
4/5
3/6
2/7
1/8
9/
10/
10/11
9/12
B
F
C