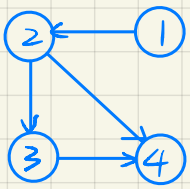
Lecture 7 Graph I



Def: $G=(V,E)$ V: set of vertice $|V|$: size of V

E: set of edge $|E|$: size of E

e.g. $V=\{1,2,3,4\}$ $E=\{(1,2),(2,3),(3,4),(2,4)\}$

Adjacency list $Adj[u]=\{v:(u,v)\in E\}$ storage space: $O(|V|+|E|)$

e.g. 1: [2] 3: [4]
2: [3|4] 4: [ ]

Adjacency Matrix $a_{ij}=1$ if $(i,j)\in E$

$=0$ otherwise

storage space: $O(|V|^2)$

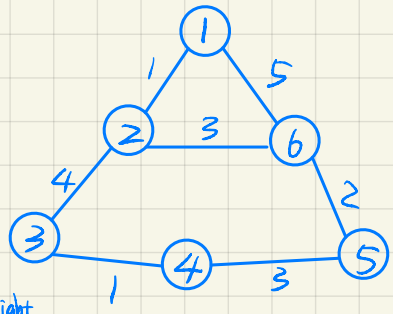
e.g.

u\v	1	2	3	4
1	0	1	0	0
2	0	0	1	1
3	0	0	0	1
4	0	0	0	0

Edge Direction

directed graph edge (u,v) is an arrow

undirected graph edge (u,v) is a line.



Edge Weight $w(u,v)$.

store in adjacency list

vertex: weight

1: [2:1 | 6:5]
2: [1:1 | 3:4 | 6:3]
3: [2:4 | 4:1]
4: [3:1 | 5:3]
5: [4:3 | 6:2]
6: [1:5 | 2:3 | 5:2]

Loop and multiedges

* simple graph: no loop no edge

Undirected graph terms

neighbor: vertix adjacent

neighborhood $N(v)$ set of all neighbors

Degree of vertex v $Deg(v)$ number of edge connected to v.

Theorem 1. $\sum_{v\in V} deg(v)=2|E|$

Theorem 2. G has even number of odd-degree vertices

Directed graph terms

edge (u,v) initial vertex u

end vertex v

Degree in-degree of v $\deg^-(v)$

out-degree of v $\deg^+(v)$

↓ adjacent to ↑ adjacent from

$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v)$

Subgraph

Add/remove vertices/edge of graph

Operation	The resulting vertex set	The resulting edge set
Add an edge e	V	$E \cup \{e\}$
Remove an edge e	V	$E - \{e\}$
Add a vertex v	$V \cup \{v\}$	E
Remove a vertex v	$V - \{v\}$	the set of edges of E that do not contain v

Special graph

Complete (each pair one edge) cycle d-dimensional cubes (bit string)

Bipartite graph

vertex set divided into 2 disjoint sets

Graph Isomorphism

a b adjacent in G_1 $\leftrightarrow$ $F(a)$ $F(b)$ adjacent in G_2

* test 1. define function F in G_2 * same adjacency matrix
  2. check each edges

* not isomorphic: different no. of vertices/edge
  not corresponding degree

Connectivity.

path: $x_0 \Rightarrow x_n$ if $x_0 = x_n$, path = cycle

connected graph connected component

cut vertex, cut edge disconnected

vertex cut: subset V': removal from connected graph make graph disconnected.

vertex connectivity: minimum size of vertex cut

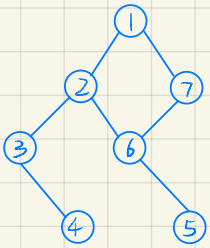
Graph Traversal

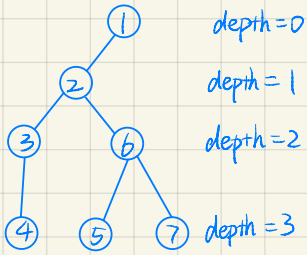
Start from sourse vertex s

visit all vertices of graph in certain order

produce search tree

Depth-first Search (DFS)





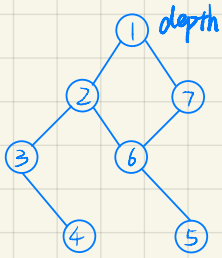
Source vertex s
visit a vertex deeper in graph
root: s parent of u: u.p
tree depth of u: u.depth

Algorithm DFS-Main(G,s)

```
for each vertex u ∈ G.V
    u.depth ← ∞
    u.p ← NIL
s.depth ← 0
DFS(G,s)
```

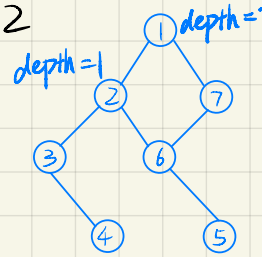
```
DFS(G,s)
for each v ← G.adj[u]
    if v.depth = ∞
        v.depth ← u.depth + 1
        v.p ← u.
        DFS(G,v)
```

Step 1:



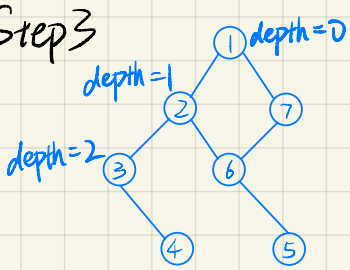
DFS(G,1)

Step 2



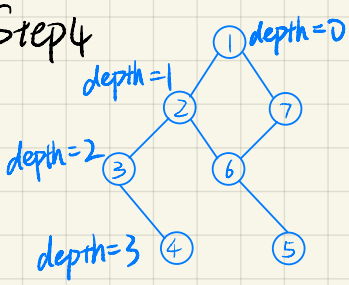
DFS(G,1)
→ DFS(G,2)

Step 3



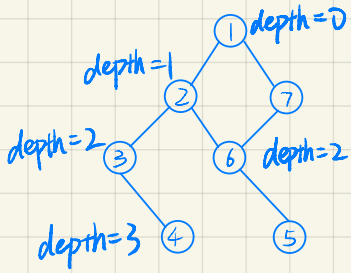
DFS(G,1)
→ DFS(G,2)
→ DFS(G,3)

Step 4



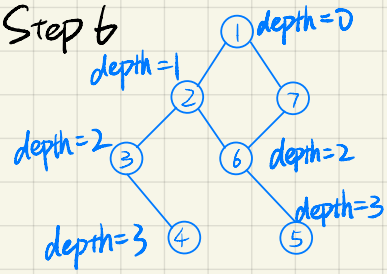
DFS(G,1)
→ DFS(G,2)
→ DFS(G,3)
→ DFS(G,4)

Step 5 backtrack to vertex 3, vertex 2



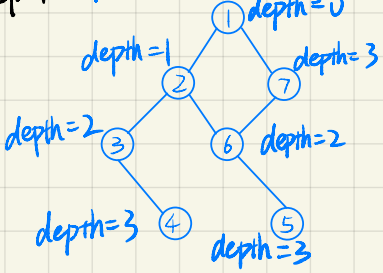
DFS(G,1)
→ DFS(G,2)
→ DFS(G,6)

Step 6



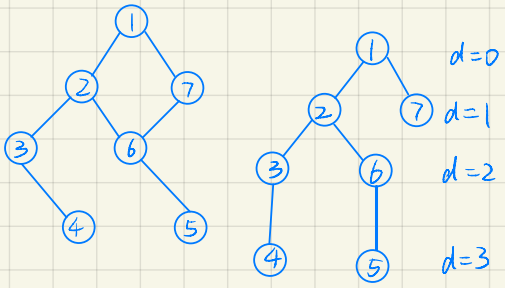
DFS(G,1)
→ DFS(G,2)
→ DFS(G,6)
→ DFS(G,5)

Step 7 backtrack to vertex 6.



DFS(G,1)
→ DFS(G,2)
→ DFS(G,6)

Breadth-First Search (BFS)



Source vertex s

visit all vertices closer to s

root: s

parent of u: u.p

distance from s to u: u.d

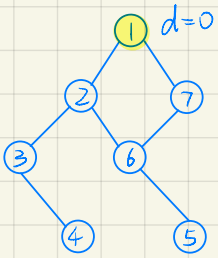
Queue used in BFS (FIFO property)

enqueue (S, x) O(1) dequeue (S) O(1)

Algorithm BFS (G, s)

```
for each vertex u ∈ G.V - {s}
    u.d ← ∞ ; u.p ← NIL
s.d ← 0 ; s.p ← NIL
Q ← create a FIFO queue
enqueue (Q, s)
while Q ≠ ∅
    u ← dequeue (Q)
    for each v ∈ G.adj[u]
        if v.d = ∞
            v.d ← u.d + 1
            v.p ← u
            enqueue (Q, v)
```

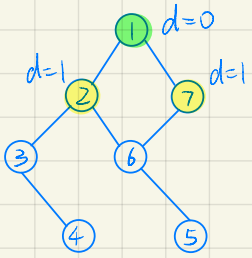
Steps: [Beginning] Q = (1 : 0)



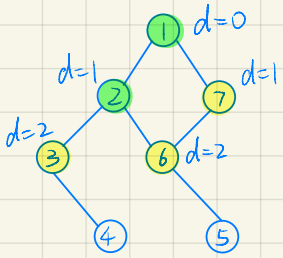
vertex in Q

dequeued vertex

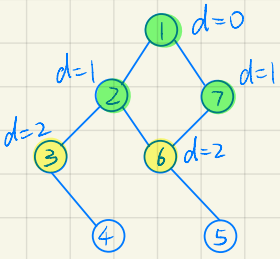
[Iteration 1] Q = (2 : 1), (7 : 1)



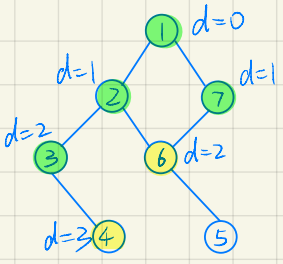
[Iteration 2] Q = (7 : 1), (3 : 2), (6 : 2)



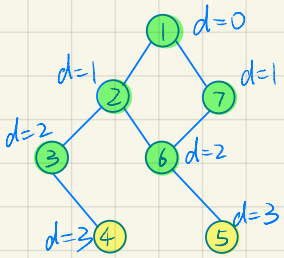
[Iteration 3] Q = (3 : 2), (6 : 2)



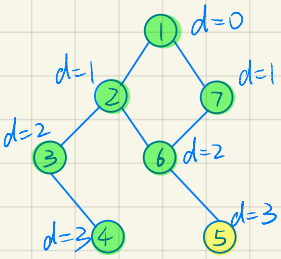
[Iteration 4] Q = (6 : 2), (4 : 3)



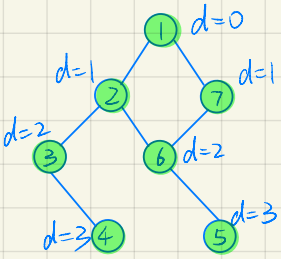
[Iteration 5] Q = (4 : 3), (5 : 3)



[Iteration 6] Q = (5 : 3)



[Iteration 7] Q = ∅



DFS/BFS: Print Path

trace the parent vertex iteratively

print in reverse order

Algorithm PrintPath(G, s, v)

```
create a list L
repeat
    insert v into list L
    v ← v.p
until v.p = NIL
print L in reverse order.
```

Application:

decide path from vertex t to s

find edge that serve as bridge(connection)

check graph is bipartite graph

Lecture 8 Graph II

Euler path and Circuits

no repeat edge
begin vertex same as end vertex

Euler circuit: simple circuit contain every edge of graph

Euler path simple path contain every edge of graph

Theorem 1: each vertex has even degree

Theorem 2: exactly two vertices with odd degree

How to find ··· circuit find cycle and delete, repeat until empty, merge cycle

How to find ··· path add dummy edge to join 2 odd-degree vertices, remove dummy edge

Hamilton path and circuits

Hamilton circuit: simple circuit pass through every vertex of graph

Hamilton path: simple path pass through every vertex of graph

test ··· circuit: n vertices — no-adjacent vertex v & u

$deg(u) + deg(v) \geq n$ → has Hamilton circuit

Shortest path problem

edge weight $w(u,v)$ *adjacency list

path → adjacent vertice sequence.

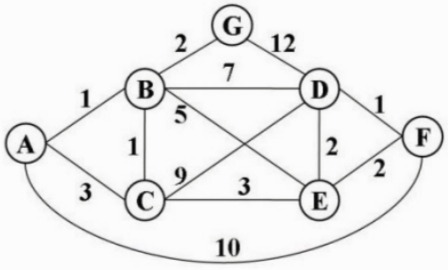
* Min-heap
node value ≤ child's value

Dijkstra's Algorithm

```
init(G,s)
  for each vertex v∈G.V
      v.d ← ∞
      v.p ← NIL
  s.d ← 0
relax(u,v)
  if v.d > u.d + w(u,v)
    v.d ← u.d + w(u.v)
    v.p ← u
```

```
Dijkstra(G,s)
  init(G,s)
  S ← ∅
  build heap(Q,G.V)
  while Q ≠ ∅
      u ← extract min(Q)
      S ← S∪{u}
      for each vertex v ∈ G.adj[u]
          relax(u,v)
          if v.d is updated
              decrease key(Q,v)
```

Q1) Consider the following undirected & weighted graph:



(a) Step through Dijkstra's algorithm to calculate the single-source shortest paths from A to every other vertex. Show your steps below. Update the old values and write in new ones, from left to right within each cell, as the algorithm proceeds. Also list the vertices in the order which you extracted (i.e. marked them known). Finally, indicate the lowest-cost path from node A to node F. Write down:

[Initialization]	Q = (A: 0), (B: ∞), (C: ∞), (D: ∞), (E: ∞), (F: ∞), (G: ∞)
[Iteration 1]	Extract vertex **A**, update vertices B, C, F A: d=0 Q = (B: 1), (C: 3), (D: ∞), (E: ∞), (F: 10), (G: ∞)
[Iteration 2]	Extract vertex **B**, update vertices C, D, E, G A: d=0, B: d=1 Q = (C: 2), (D: 8), (E: 6), (F: 10), (G: 3)
[Iteration 3]	Extract vertex **C**, update vertices D, E A: d=0, B: d=1, C: d=2 Q = (D: 8), (E: 5), (F: 10), (G: 3)
[Iteration 4]	Extract vertex **G**, update vertices D A: d=0, B: d=1, C: d=2, G: d=3 Q = (D: 8), (E: 5), (F: 10)
[Iteration 5]	Extract vertex **E**, update vertices D, F A: d=0, B: d=1, C: d=2, G: d=3, E: d=5
[Iteration 6]	Q = (D: 7), (F: 7) Extract vertex **D & F** (regardless their order) A: d=0, B: d=1, C: d=2, G: d=3, E: d=5, D: d=7, F: d=7 Q = ∅

* shortest path tree

Planar graph

planar ⇒ no crossing edge

Euler's formula

no. of region = edge − vertices + 2

Graph coloring

no adjacent vertices have same color.

Lecture 9 Graph III

Flow network : $G=(V,E)$

producer : source vertexs

consumer : sink vertex t.

vertex, edge → capacity $c(u,v)$, flow $f(u,v)$

Format : flow/capacity.

$c(u,v) \geq f(u,v) \geq 0$

flow conversion : $\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v)$

Modeling : add dummy vertex for both edge

Maximum Flow Problem

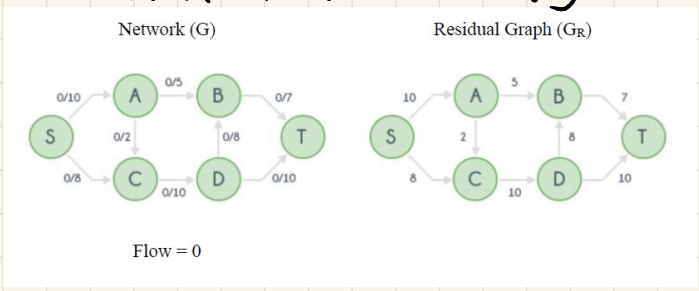
value of flow f $|f|$ find max value of $|f|$

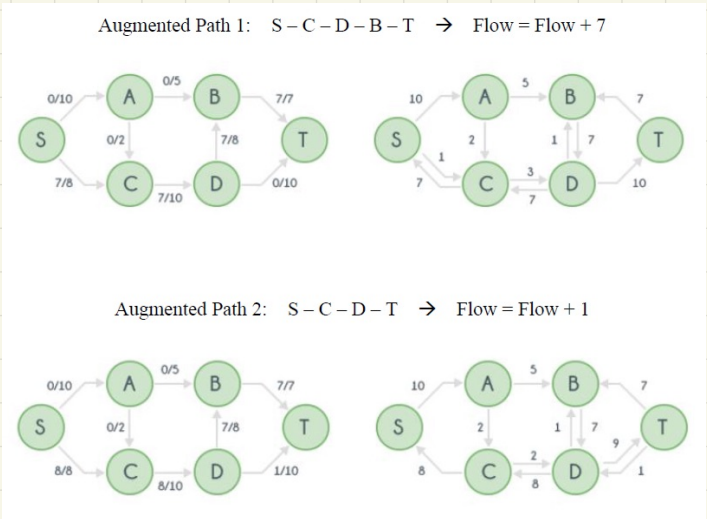
Residual Network $G_f=(G.V, E_f)$

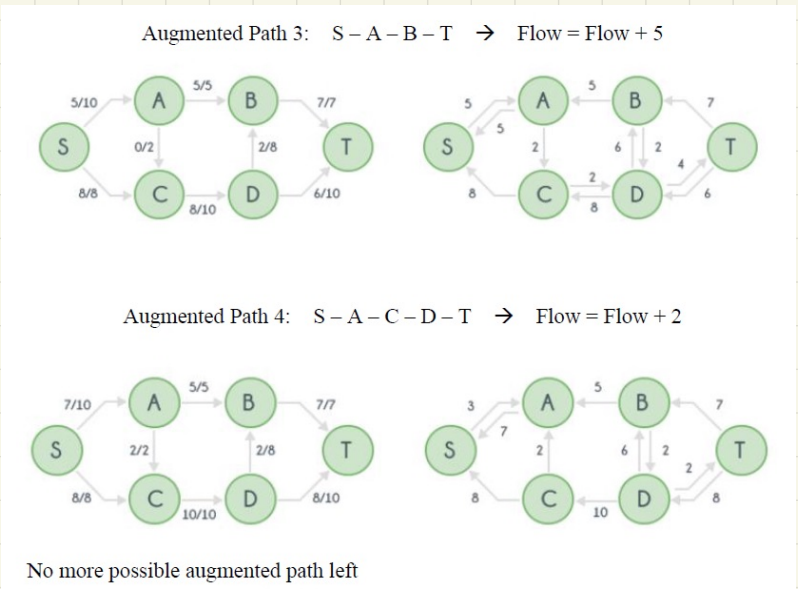
Augmenting path simple path from s to t in residual network

residual capacity min value of c_f in path.

Ford-Fulkerson algorithm



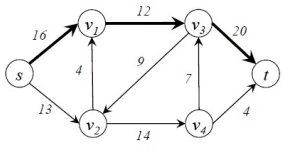




Edmonds-Karp Algorithm using BFS fewest edge less path

Iteration 1

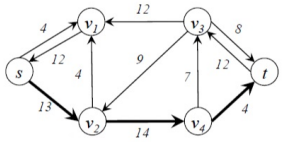
- 1. Find a path from s to t, by **BFS** on the residual network G_f
  - The path $<s, v_1, v_3, t>$ (with the fewest edges)
  - Distances from s to vertices $(s, v_1, v_2, v_3, v_4, t)$: 0, 1, 1, 2, 2, 3
- 2. The minimum residual capacity $c_f(u, v)$ on the path is 12
- 3. Update the flow



residual network G_f

Iteration 2

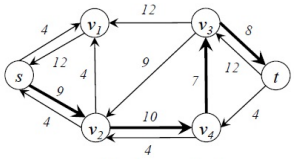
- 1. Find a path from s to t, by **BFS** on the residual network G_f
  - The path $<s, v_2, v_4, t>$ (with the fewest edges)
  - Distances from s to vertices $(s, v_1, v_2, v_3, v_4, t)$: 0, 1, 1, 3, 2, 3
- 2. The minimum residual capacity $c_f(u, v)$ on the path is 4
- 3. Update the flow on G



residual network G_f

Iteration 3

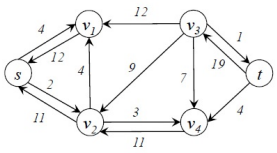
- 1. Find a path from s to t, by **BFS** on the residual network G_f
  - The path $<s, v_2, v_4, v_3, t>$ (with the fewest edges)
  - Distances from s to vertices $(s, v_1, v_2, v_3, v_4, t)$: 0, 1, 1, 3, 2, 4
- 2. The minimum residual capacity $c_f(u, v)$ on the path is 7
- 3. Update the flow on G



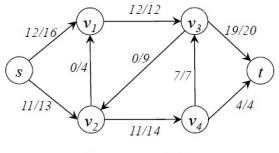
residual network G_f

Final result

- 1. BFS cannot find any path from s to t, on the residual network G_f
  - Distances from s to vertices $(s, v_1, v_2, v_3, v_4, t)$: 0, 1, 1, ∞, 2, ∞
  - Thus the algorithm terminates
  - We can derive the corresponding flow network G
- 2. The maximum flow $|f|$ is: 11 + 12 = 23



residual network G_f



flow network G