Algorithm Design & Analysis (CSE222)

Lecture-12

Recap

- Edit Distance
- Weighted Subset
  - 0/1 Knapsack

0/1 Knapsack

Let there are n items {1, 2, …, n}, a weight function w: [n] → $R_{>0}$ and a value function v: [n] → $R_{>0}$. Let W > 0.

Goal: Find subset S ⊆ {1, 2, …, n} such that $\sum_{i \in S} w_i \leq W$ and the value of the subset if maximum.

0/1 Knapsack

Subproblem: Val(i,T): Maximum value from first i elements with weight limited to T.

Recurrence: $Val(i, T) = \max\{Val(i-1, T), Val(i-1, T-w_i) + v_i\}$

Final Solution: Val(n, W)

Running time?

Recurrence Problem

Is the following relation is true?

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Prove without showing LHS = RHS.

Outline

- Graphs
- Depth First Search

Graph Notations

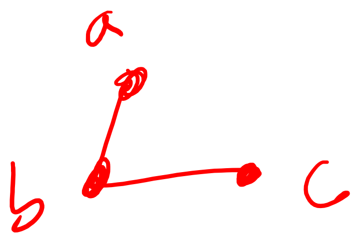
Graphs: Captures relationship (a.k.a edges) between objects (a.k.a nodes).
Consists of collection of nodes (a.k.a vertices) V and a collection of edges E (sets of nodes).

$G = (V, E)$

<u>Simple Graphs</u>

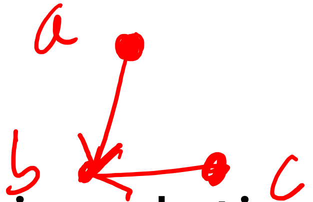
- At most one edge between pair of vertices (i.e., each edge is a pair of nodes).
- Undirected (or symmetric) and unweighted edges.
- No self loops.

Graph Notations



$(a,b) / (b,a) \in E$

Undirected graph for symmetric relationship, $G = (V, E)$. Every $e \in E$ is not an ordered pair (u, v).



$(a,b) \in E$

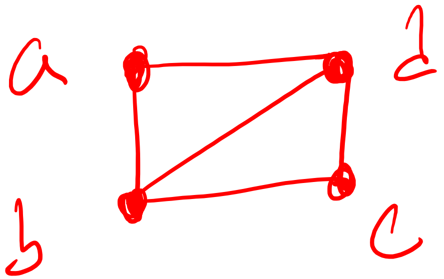
$(b,a) \notin E$

Directed graph for asymmetric relationship, $G = (V, E)$. Every $e \in E$ is an ordered pair (u, v).

Graph Notations

Given a graph $G = (V, E)$, a **path** p is a sequence of vertices $v_1, v_2, \ldots, v_n$ such that each pair of consecutive vertices (v_i, v_{i+1}) is an element in E.

A **simple path** is a sequence of vertices, where none of them are repeated.



paths

a, b, c ✓

b c d ✓

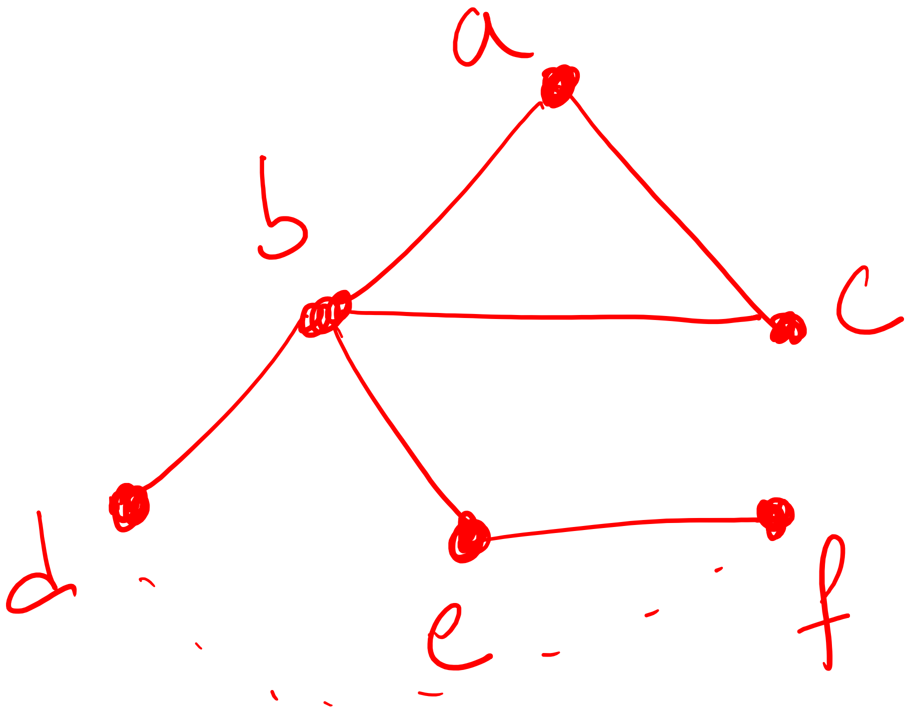
a c b X

b c d b

Graph Notations

Given a graph $G = (V, E)$, a **<u>path</u>** p is a sequence of vertices $v_1, v_2, \ldots, v_n$ such that each pair of consecutive vertices (v_i, v_{i+1}) is an element in E.
A **<u>simple path</u>** is a sequence of vertices, where none of them are repeated.
A **<u>cycle</u>** is a path if all the vertices are unique except for the first and the last.
A graph is called **<u>connected</u>** if there is a path between every pair of vertices.
A **<u>tree</u>** is a connected graph without any cycle.
A directed graph with no cycle is called **<u>directed acyclic graph</u>** (DAG).
The shortest path length from u to v is called the **<u>distance</u>** from u to v, i.e., $\delta_G(u, v)$.
The maximum distance between any pair of vertices is the **<u>diameter</u>** of a graph.
<u>Eccentricity</u> of a vertex u $\{e(u)\}$ is maximum distance of any other vertex v from u.
The node with the smallest eccentricity is the **<u>center</u>** and its value is the **<u>radius</u>** of the graph.

Examples



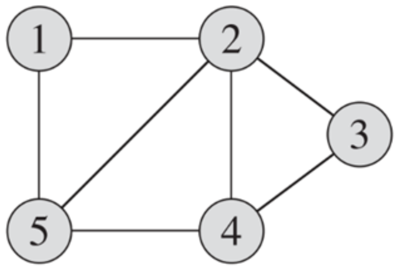
dia = 3

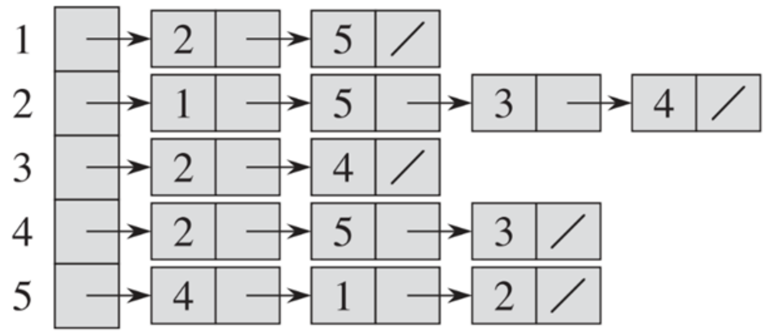
ecc(d) = 3

center = b

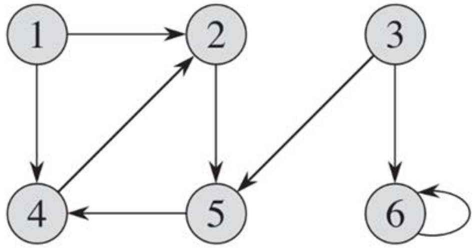
radius = 2

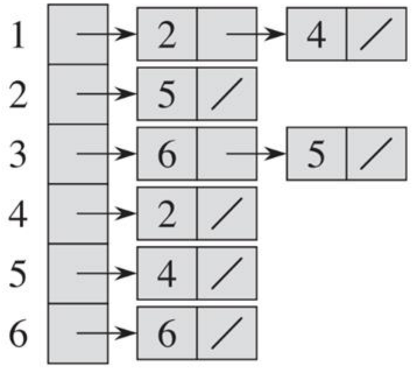
Graph Representation





	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0





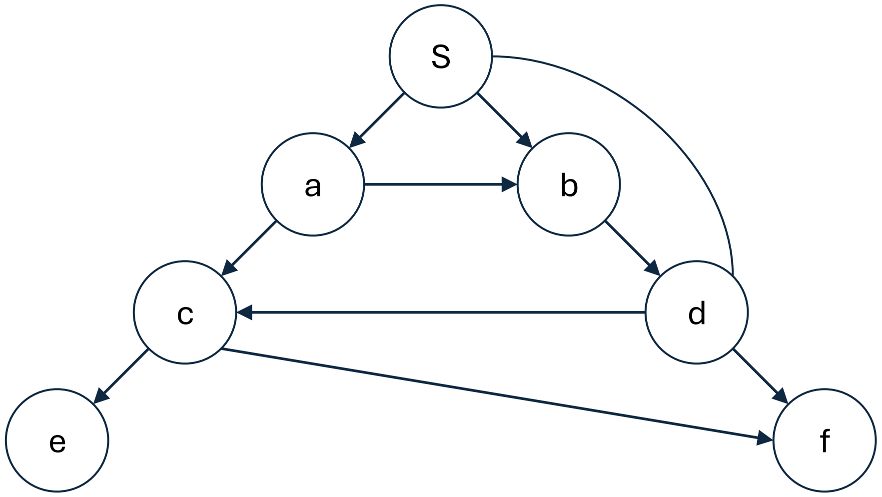
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Graph Properties

A graph G = (V, E) is sparse if $|E| \ll |V|^2$.

Topological Sort of a DAG is a total ordering $v_1 < v_2 < \ldots < v_n$ of vertices in V such that for any edge $(v_i, v_j) \in E$, $j > i$.

If edges of DAG represents dependencies, then topological sort follows all dependencies.



Depth First Search

```
ITERATIVE-DFS(s)
Push(s)
while Stack is not empty
    u ← Pop()
    if u is unmarked
        mark u
        for every edge (u, v)
            Push(v)
```

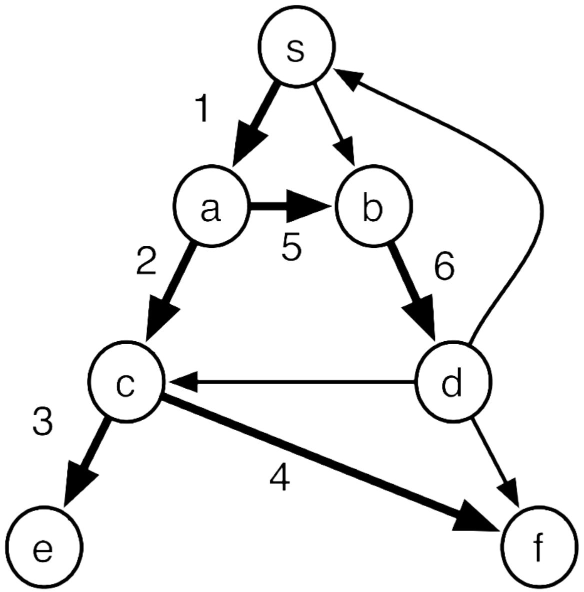
Graph Properties

A graph G = (V, E) is sparse if $|E| \ll |V|^2$.

Topological Sort of a DAG is a total ordering $v_1 < v_2 < \ldots < v_n$ of vertices in V such that for any edge $(v_i, v_j) \in E$, j > i.

If edges of DAG represents dependencies, then topological sort follows all dependencies.

Running time?



Outline

- Graphs

- Depth First Search

Depth First Search

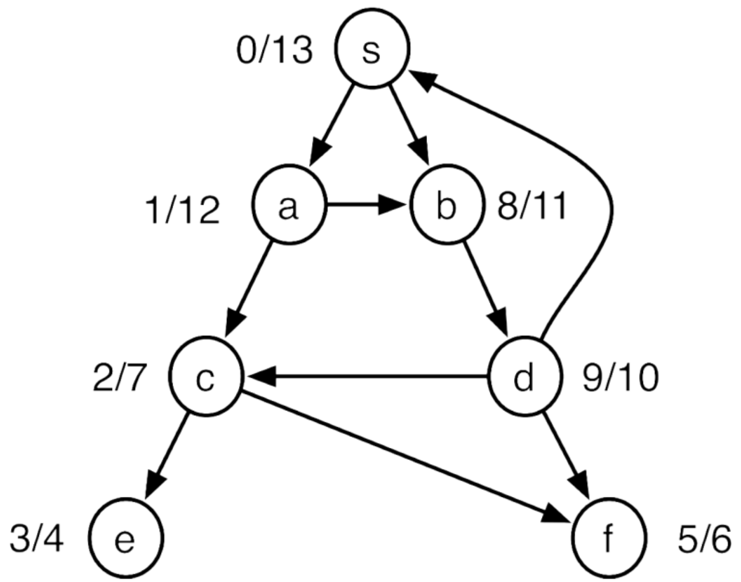
DFS(G)

for each $u \in G.V$
    $u.color = \text{WHITE}$
$time = 0$
for each $u \in G.V$
    if $u.color == \text{WHITE}$
        DFS-VISIT(G, u)

DFS-VISIT(G, u)

$time = time + 1$
$u.d = time$
$u.color = \text{GRAY}$ // discover u
for each $v \in G.Adj[u]$ // explore (u, v)
    if $v.color == \text{WHITE}$
        DFS-VISIT(v)
$u.color = \text{BLACK}$
$time = time + 1$
$u.f = time$ // finish u

Entry and Exit time



DFS Tree Edge Types

Let DFS starts at node s and returns a tree.

We call an edge (u, v) a **tree edge** if it is present it the returned tree.

The rest of the edges in the graph, which are non-tree edges, can further be classified as back edges, forward edges, and cross edges.

An edge $(u, v) \in E$ is called **back edge**, if v is an ancestor of u in DFS tree.
An edge $(u, v) \in E$ is called **forward edge**, if v is decentend of u in DFS tree.
An edge $(u, v) \in E$ is called **cross edge**, if v is neither ancestor nor a decentend of u in DFS tree.

Reference

Slides

Introduction to Algorithms by CLRS - Chp-22.3