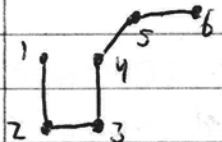
Analysis of Algorithms - CS 323
Lecture #8 - March 30, 2016
Notes by Jules Skrill

Graph algorithms
$G = (V, E)$
V - vertices, E - edges

Cartesian product
$A \times B = \{[a, b] \mid a \in A, b \in B\}$
Relations (on $A \times B$) - subset of cartesian product
Ex: If relation is students taking 2 classes together
we can have $[s_1, s_2] \in R$
$[s_3, s_5] \in R$

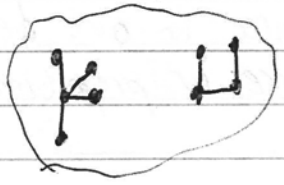


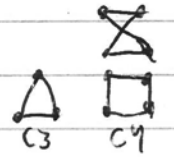
$V = \{1, 2, 3, 4, 5, 6\}$
$E = \{[1,2], [2,3], [3,4], [4,5], [5,6]\}$
$G = (V, E)$

Directed graph - Edge has direction (ie: 1 to 2 (1,2))
Undirected graph - Edges is bidirectional (ie: 1 to 2/2 to 1 ({1,2}))

DAG - Directed acyclic graph (no cycles)
cycle - A path which forms a closed circuit

Trees are graphs
Trees aren't necessarily directed } not necessarily DAGs
Trees do not have cycles

Graphs can have components while trees can not
Ex:
$G =$ 

C_n - only cycle 

K_n - Each vertice has an edge to every other vertice

Not carteslan product because each vertice does not have an edge to itself

Edge count in K_n is $\frac{n(n-1)}{2} = (n,2) = {}_nC_2 = \binom{n}{2}$

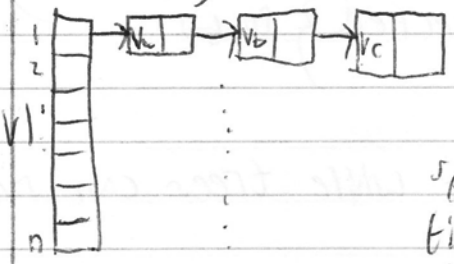
Graph can model relationships (ex: is a brother / network connectivity)

Representation:
Adjacency matrix (A) $|V| \times |V|$

$A[i,j] = \{1$ if $\exists$ edge between v_i and v_j otherwise $0\}$
$\deg^+(v_i)$ is out degree or edges directed away from vertice
$\deg^-(v_i)$ is indegree or edges directed toward vertice.

To find the out degree of a vertice, sum the row of A
To find the in degree of a vertice, sum the column of A
space is $\Theta(|V|^2)$
Diagonals are loops
If $A = A^T$ (is symetric over the diagonal), the graph is undirected

Adjacency Table (list)

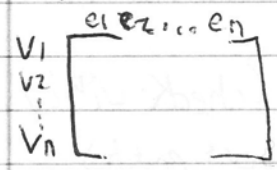


$n = |V|$

$V_a, V_b, V_c \in V$

space $= \Theta(|V| + |E|)$
time out degree is out degree
time in degree is proportional to $|V|$

Incidence matrix



1 if vertex in row is incident to edge in column

space = $\theta(|V| \cdot |E|)$

weighted Graph

C[i,j] = weight/cost of edge (i,j)

If v_i isn't adjacent to v_j, the cost is ∞

connected to - there is some path between 2 vertices

adjacent to - $(\exists e \in E)\ (v_1, v_2) = e)$

Searching a graph (traversal)

Depth first search (DFS)

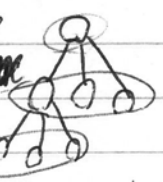
1) Pick a vertice
2) Visit it and its children in depth first order if it hasn't yet been visited
3) Iterate until search condition has been met or no more vertices (back to 2)

$\theta(|V|+|E|)$

```
main
  for i=1 to n
    if !v.visited:
      DFS(v)
DFS(vertex)
```

Breadth First search (BFS)

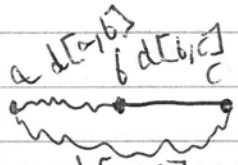
1) Pick a vertex an putin queue
2) dequeue queue and visit vertex if not visited then add children to queue
3) Iterate until search is completed or no unvisited vertices (back to 2)



$\theta(|V|+|E|)$

Topological search $\Theta(|V|+|E|)$
Flattened tree
can be implemented as DFS or BFS (check wiki)

shortest path { SSSP (single-source shortest path)
APSP (All-pairs shortest path)



path
edge

d is distance

negative-weight edges is possible but not common
Dijkstras SSSP

init
```
For i=1 to |V|                          pred[]=> Θ(n^2)
  distance[i] = c[s,i]                  Θ(n^2)
visit neighbors of i                    Θ(|E| log |V|)
  if (distance[i] + c[i,j] < distance[j])
     distance[j] = distance[i] + c[i,j]
     pred[j] = i
```

goto

brute $\Theta(n^2)$

heap $\Theta(|V| \log |V| + |V|^2) \log |V|)$
$\Theta(|V| \log |V| + |E|) \log |V|)$