Lecture 13

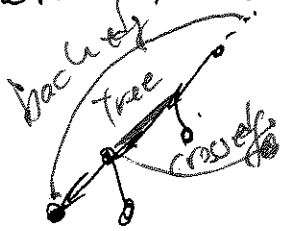
Last Time: Reachability problem in graphs.. $G=(V,E)$ a node u which nodes are reachable i.e. there is a path from u?

- Depth first Search tree.
- Connected components in case of Directed graphs Strongly connected components

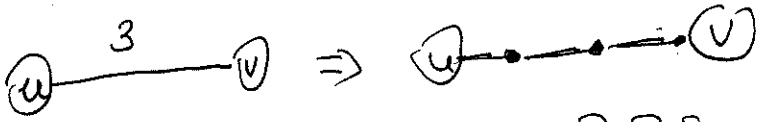
DFS uses FILO

- DAG
- Linearization / topological sorting of G. based on pre and post order visits during DFS

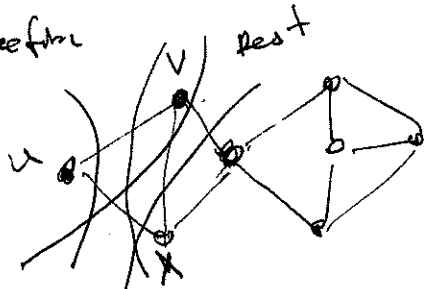
first time last time



Today: Distance / shortest paths in graphs $G=(V,E,w_e)$ weight as distance.



1. unweighted graphs ⇒ BFS. tree.
2. weighted graphs ⇒ Djkestra's shortest path Alg.
3. " " with some weights negative value ⇒ Bellman-Ford.
4. "negative cycles" ⇒ detection

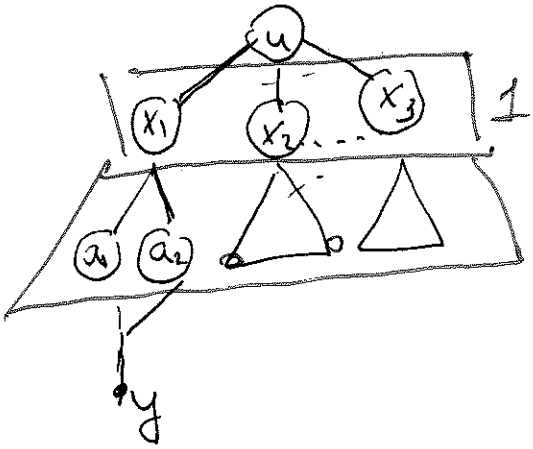


Sub set of the nodes that are processed

Rest what is the best data structure to organize these nodes ⇒ priority queue ⇒ Heaps several options.

unweighted graph $G = (V, E)$ given u

Q: what are the shortest paths to all other nodes in G
(assume G is connected)



BFS

FIFO

1 hop away from u

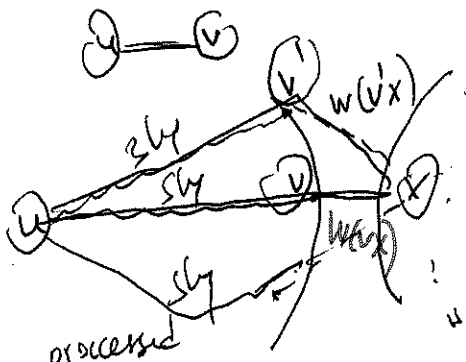
2 hops away from u

$dist(x,y) = 1$ for and edge (x,y)

time complexity of the BFS

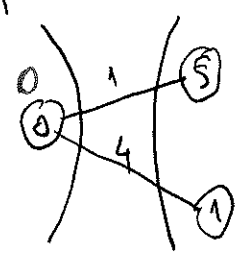
$O(|V| + |E|)$.

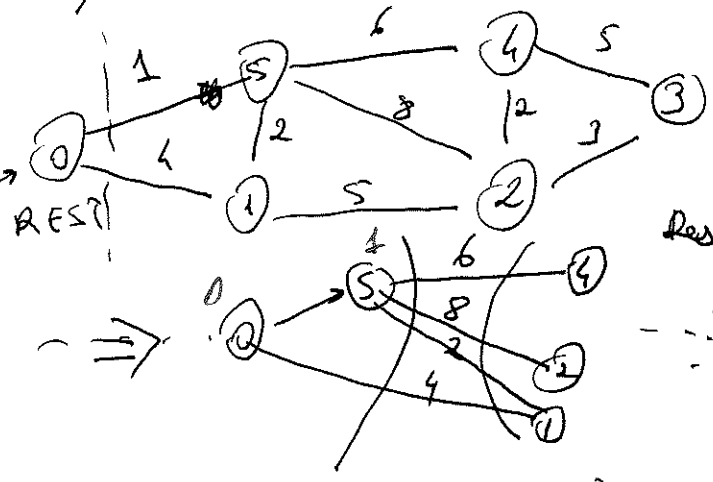
how do we show that BFS is a shortest path Tree?

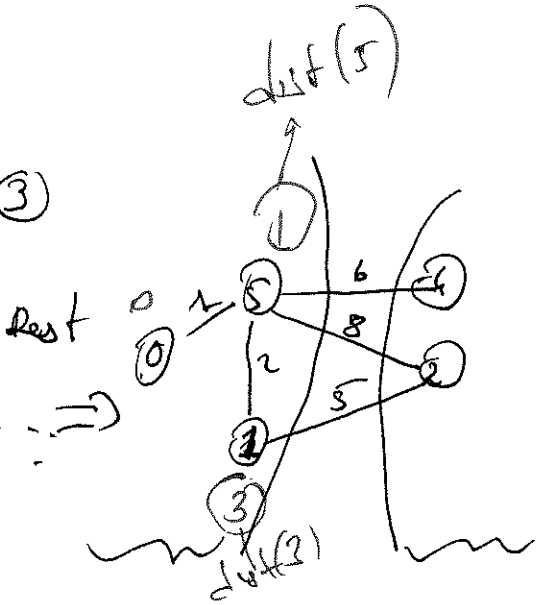


$shy(u \to v) + l \Rightarrow shy(u \to x)$

example!
undirected graph.







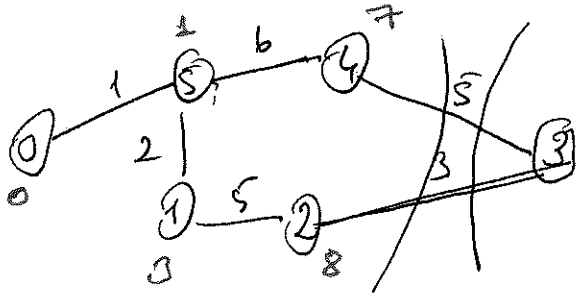
graph Cut

partition of the nodes of a graph

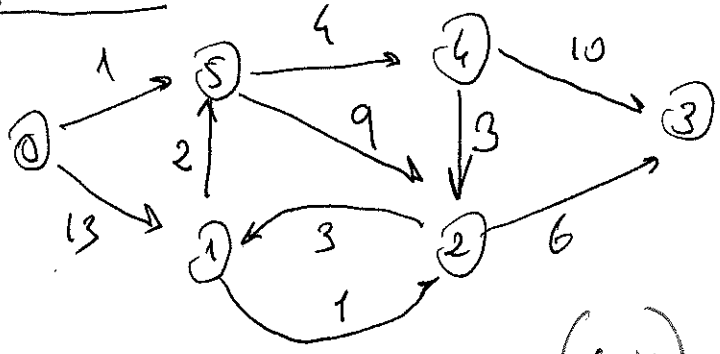
choose an edge (u,v) in the cut s.t.

$dist(u) + w(uv)$ is minimum

dist(u): distance from source/origin node to node u



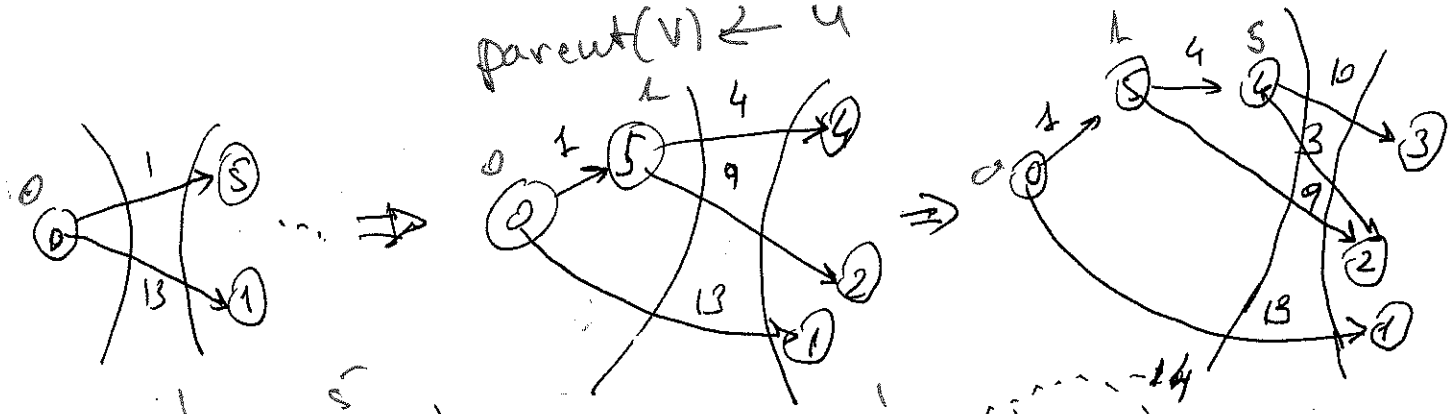
Directed Case

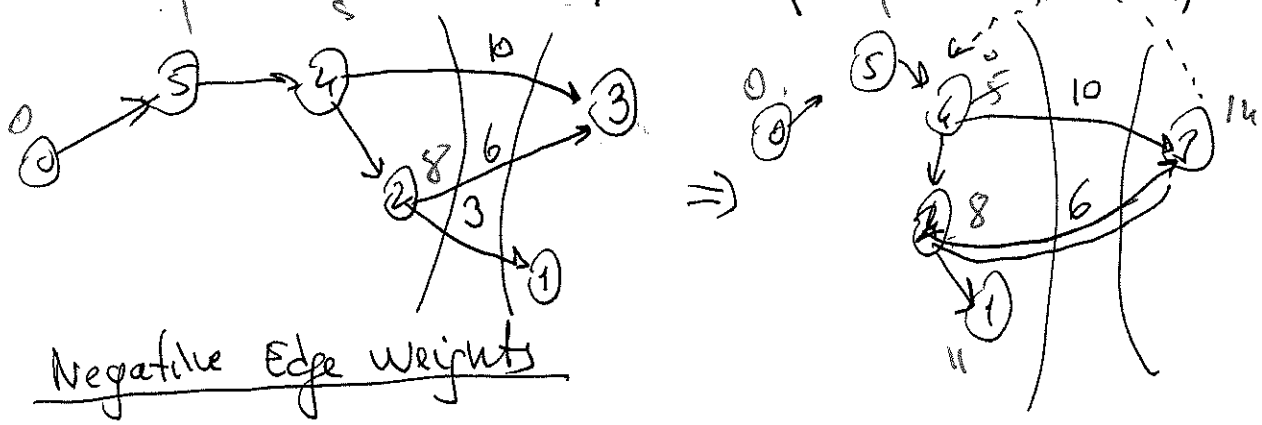


$\forall (u,v) \in E$ $\quad Dist(u) + w(u,v) < dist(v)$ then

$dist[v] \leftarrow dist(u) + w(u,v)$

$parent(v) \leftarrow u$





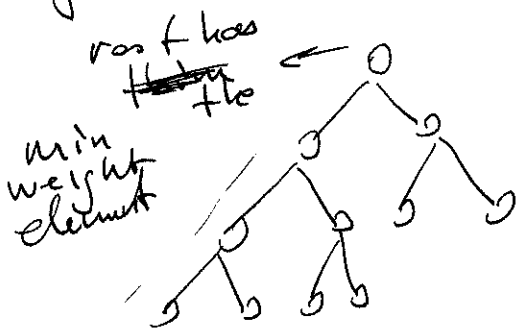
Negative Edge Weights

negative weight cycles

Heaps

Binary Heap

Idea: organize the elements into a binary tree (full)



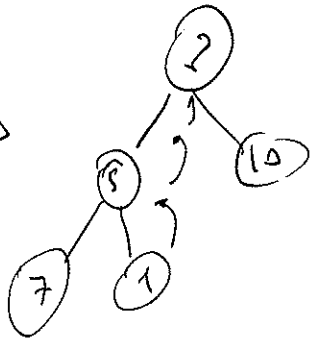
heap property:

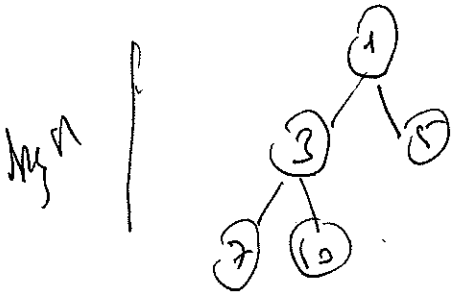
a parent node has weight/key value smaller / larger than both of its children

min heap / max heap.

10 3 5 7 1 12 13 2

ensure the Heap property ⇒





$\log n$

d - any tree.

d children.

$O(n \log n)$ building a heap.

not tight bound ⇒ $O(n)$