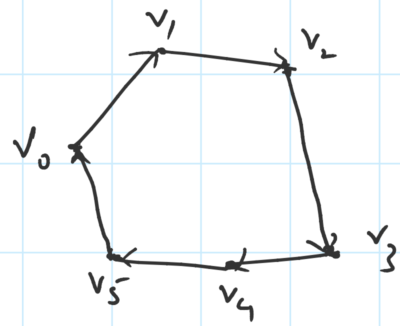
Lecture 9

Friday, February 4, 2022 5:27 AM

Feb 4, 2022

Graph Algorithms

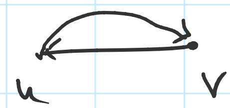


Graph: V - vertices

$E \subseteq V \times V$

$V = \{v_0, v_1, v_2, v_3, v_4, v_5\}$

$E = \{(v_i, v_{i+1}) : i\}$



Directed graphs.

Undirected graphs

Weighted graphs → each edge e is assigned a weight/length $l_e \in \mathbb{R}$ ✓

Unweighted graphs.

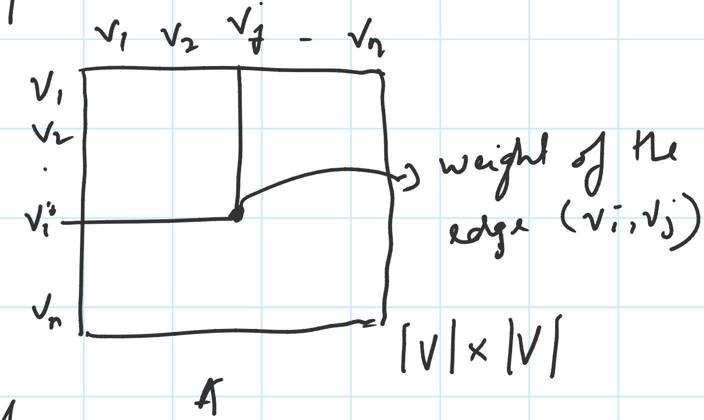
Representation of graphs

(1) Adjacency matrix of a graph

Notation:

$\{|V| = n$

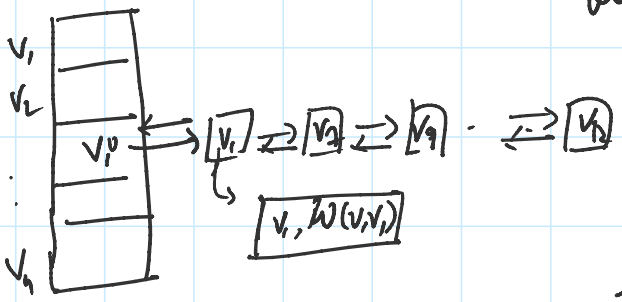
$|E| = m\}$



(2) Adjacency list of a graph

Out-neighbors(v)

(2) Adjacency list of a graph



Out Neighbors$(v) = \{ u : (v,u) \in E \}$

$v \longrightarrow u$

In Neighbors

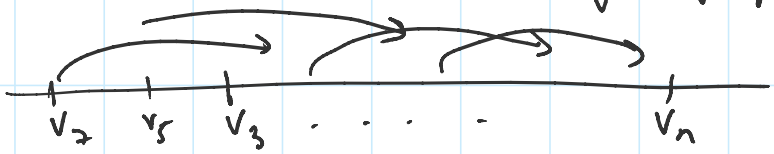
(*) Adjacency list might be more space efficient compared to Adjacency matrix representation — Eg: $m << n^2$, $m = O(n)$

Basic Graph Algorithms that you ought to know

(1) Breadth First Search (BFS)

(2) Depth First Search (DFS)

(3) Topological sort: Given a directed acyclic graph



(4) Connected components (undirected)

(5) Strongly connected components

(6) Dijkstra's algorithm for single source shortest paths.

X X

Shortest path algorithms (Directed, without loss of generality)

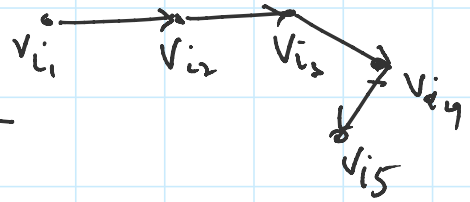
ALL NONNEGATIVE WEIGHTS.

① Dijkstra's algorithm

- given a graph G, $s \in V(G)$, weighted
- find $dist(s,u)$:= length of the shortest (min length) directed path from $s \to u$ in G.

path = seq $(v_{i_1}, v_{i_2}, \dots, v_{i_k})$ of vertices in G

such that



- $(v_{i_j}, v_{i_{j+1}}) \in E$

[- simple paths → all distinct vertices.]

[Here, we allow vertices to repeat.]

weight/length (P) = sum of weights of all edges

Dijkstra's Algorithm

Computes $L : V \to \mathbb{R}$

At termination $L(u) = dist(s,u)$

Init. $\begin{cases} X = \{s\} \\ L(s) := 0, \quad L(u) = +\infty \ \forall\, u \in V, u \neq s \end{cases}$

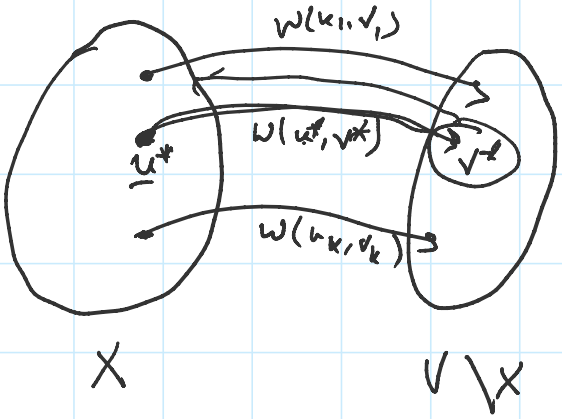
while there is an edge (u,v) for $u \in X$, $v \in V \setminus X$

$(u^*, v^*) :=$ such an edge which minimizes $L(u) + w(u,v)$

add v^* to X

$L(v^*) := L(u^*) + w(u^*, v^*)$

Picture



Running Time: $O(mn)$

Faster implementations possible — use a heap.

Proof of correctness:

Proof by induction — can induct on # vertices.
— # iterations of the while loop

$P(k)$: for the k^{th} vertex v added to X in the algorithm, $L(v) = dist(s,v)$.

Base case: $X = \{s\}$

$L(s) = 0$

Induction Step:

- Assume $P(1), P(2), \ldots, P(k-1)$ are true.
- Want to prove $P(k)$ is true.

Suppose $v^* \in V$ is added to X in the k^{th} iteration

from the Algorithm $\Leftrightarrow$

$\exists u^* \in X$,

1) $(u^*, v^*) \in E$

2) minimizes $L(u^*) + w(u^*, v^*)$ for all edges from X to $V \setminus X$.

Want to show: at the end of this iteration

$$L(v^*) := L(u^*) + w(u^*, v^*) = dist(s, v^*)$$

1) $L(u^*) + w(u^*, v^*) \geq dist(s, v^*)$

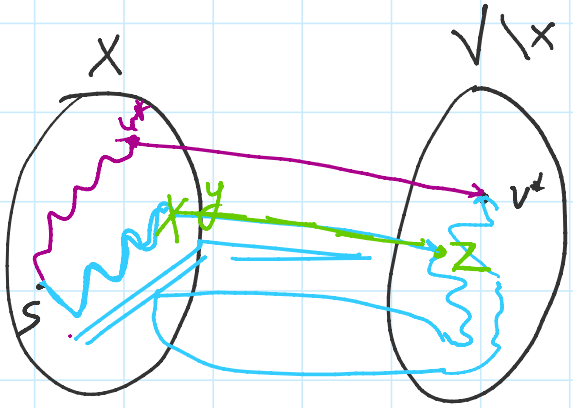
By I.H. $L(u^*) = dist(s, u^*)$

$\Rightarrow \exists$ path P from s $\xrightarrow{L(u)}$ u^* in G $\xrightarrow{w(u^*, v^*)}$

$\Rightarrow$ $\exists$ path P from s ⇝ u^* → v^* in G, with $w(u^*, v^*)$

P'

② $\quad dist(s, v^*) \geq L(u^*) + w(u^*, v^*)$



$dist(s, v^*)$ = (length of blue path from $s \to y$) + $w(y,z)$ + (length of blue path from z to v^*)

Blue path

$\geq dist(s,y) + w(y,z) + dist(z, v^*)$ ✓

Non-negativity of $dist(z, v^*)$ ✓

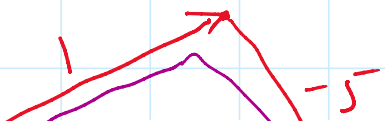
$\geq dist(s,y) + w(y,z) + 0$

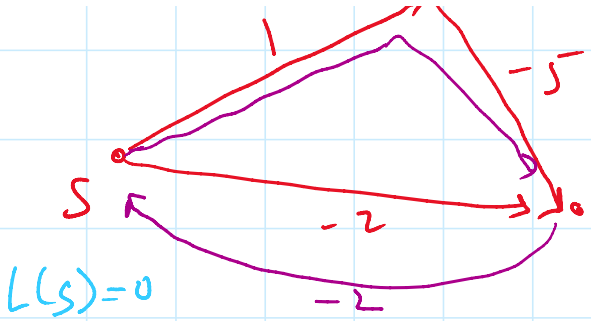
$= L(y) + w(y,z)$

$\geq L(u^*) + w(u^*, v^*)$

choice of v^*

Ex:





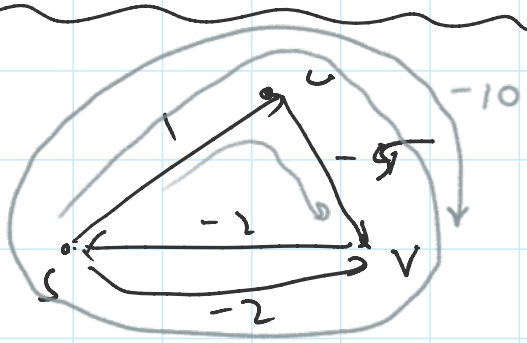
$L(s) = 0$

$dist(s,s) = 0$

$L(v) = -2$

$dist(s,v) = -4$

Shortest paths on graphs with negative edge weights



-4 $s \longrightarrow u \longrightarrow v$

-10 $s \xrightarrow[1]{} u \xrightarrow[-5]{} v \xrightarrow[-2]{} s \xrightarrow[1]{} u \xrightarrow[-5]{} v$

$s \longrightarrow u \longrightarrow v \longrightarrow s \longrightarrow u \longrightarrow v \longrightarrow s \longrightarrow u \longrightarrow v$

Negative edge weights could lead to negative weight cycles $\longrightarrow$ hard to define what shortest path means.

Fix 1: Restrict to cycle-free paths

Problem becomes NP-hard
(Don't expect a polynomial time algorithm for this version)

Fix 2: Either:

① Compute shortest paths correctly

OR

② Declare that there is a negative weight cycle.

Input: Directed, weighted graph, source vertex
- negative weights allowed
- negative weight cycles also allowed

Output: Either

① List of shortest paths

OR

② Assert that there is a negative weight cycle.

Bellman-Ford's algorithm Dynamic Programming based.