Circuit Optimization

e.g., energy, performance, latency, area, throughput

- Typically involves multiple objective functions as optimization goals.
- There are typically multiple possible solutions, i.e., different potential circuit implementations.

Heuristics: problem solving based on experience

- Optimizations can be approximate or exact
- We are typically interested in Pareto-optimal design points
  - Points at which it is impossible to make one design goal better without making at least one individual worse off.
  - Also known as 'trade-off' points.

Scheduling problems

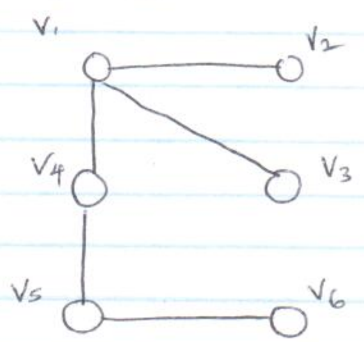
E.g., Minimize circuit area under latency constraints

Minimize circuit latency under area constraints.

In most cases, exact solutions are hard to find, and typically, only approximate solutions are found.

Graphs Overview

G:



~~ver~~ Vertices

$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$

edges

$E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_4, v_5\}, \{v_5, v_6\}\}$

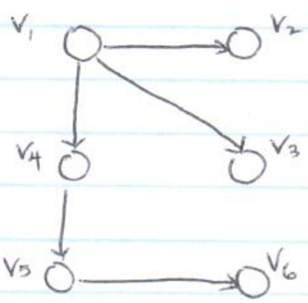
Degree of a vertex: no. of edges incident to it; e.g., $\deg(v_1) = 3$

Cardinality: no. of vertices

$|G| = 6$

Adjacent vertices: all vertices connected by an edge.

Undirected graph : $E(u, v)$ is equivalent to $E(v, u)$



Directed graph specifies source vertex and destination vertex.

Weighted graphs: edges and/or vertices assigned with numerical weights (Edge weighted or ~~ver~~ vertex weighted)

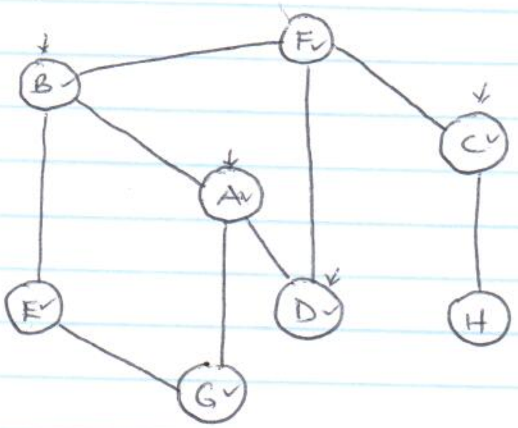
E.g. of weights: cost of flight between two cities.

Why Graph Optimizations?

- Many problems are easily expressed as a set of transitions between states of a system. Such problems can naturally be approached as graph search.
  - E.g., flight paths, cost of different layover options
  - Planning the actions of an autonomous agent, e.g., robot.
- Nodes can represent the states of a system, and edges represent the actions that will transition the system from one state to another.

<u>Graph algorithms</u> <u>Breadth-first</u> search

Can be implemented with Queue — linked list



~~B~~
~~D~~
~~G~~
~~E~~
~~F~~
~~C~~

BFS(v):
Initialize queue Q
Mark v as visited, push to queue
While Q is not empty
  Take front element, call it w
  For each w → u
    If u is not visited, mark it as visited and push it to Q

Visited: A, B, D, G, E, F, C, H

<u>Depth-first search</u> : Uses stack; (recursion)
1, push onto stack
2, visit
3, Mark as visited

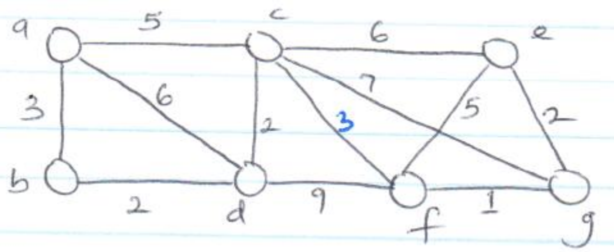
DFS(v):
Mark v as visited
For each edge v → u :
  if u is not visited, call DFS(u)

Stack : ~~H~~ ~~G~~ ~~X~~ ~~E~~ ~~F~~ ~~B~~ ~~A~~

Visited; A, B, E, G, F, C, H, D

Shortest Path Problem

Dijkstra's algorithm : used for graphs with positive edge weights.
Priority queue or heap.



Initialize source to zero and every other vertex to ∞

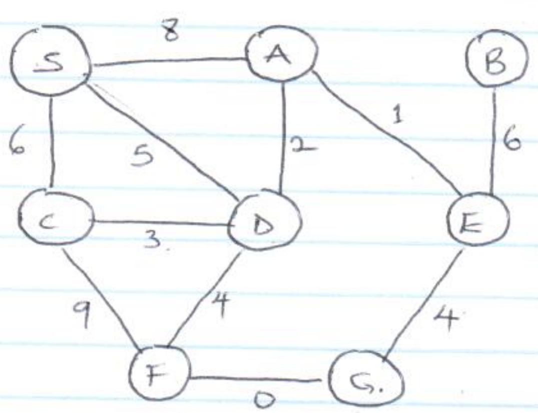
Relax all out-edges from source

- Check if edge can give shorter path to connected vertex than current shortest path.
- Next vertex to relax: vertex closest to source.

V	a	b	c	d	e	f	g
a	$\boxed{0a}$	3a	5a	6a	∞	∞	∞
b		$\boxed{3a}$	5a	5b	∞	∞	∞
c			$\boxed{5a}$	5b	11c	8c	12c
d				$\boxed{5b}$	11c	8c	12c
f					$\boxed{11c}$	$\boxed{8c}$	9f
g							$\boxed{9f}$

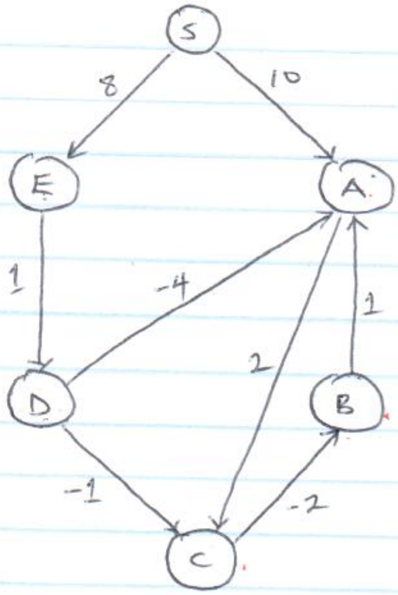
$$T = O(\underbrace{E}_{\text{edges}} \log \underbrace{V}_{\text{vertices}})$$

Example 2



	S	a	b	c	d	e	f	g
s	$\boxed{0s}$	8s	∞	6s	5s	∞	∞	∞
d		7d	∞	6s	$\boxed{5s}$	∞	9d	∞
c		7d	∞	$\boxed{6s}$		∞	9d	∞
a		$\boxed{7d}$	∞			8a	9d	∞
e			14e			$\boxed{8a}$	9d	12e
f			14e				$\boxed{9d}$	9f
g			$\boxed{14e}$					$\boxed{9f}$

Bellman-Ford Single-source Shortest Path Algorithm



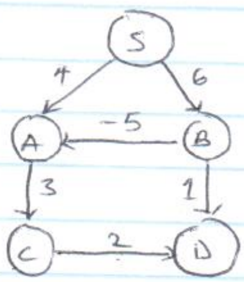
maximum V − 1 iterations.

Repeatedly relax all edges in the graph.

$T = O(VE)$

Iterations	S	A	B	C	D	E
1	0	10	∞	∞	∞	8
	0	10	10	12	9	8
2	0	5	10	8	9	8
3	0	5_D	5_C	7_A	9_E	8_S

Example 2



	S	A	B	C	D
1	0	4_S	6_S	∞	∞
		1_B	6_S	7_A	7_B
2	0	1_B	6_S	4_A	6_A
3		1_B	6_S	4_A	6_C ⇐

Directed Acyclic Graphs (DAGs)

$G = (V, E)$, a directed graph

No cycles, i.e., no directed path from any vertex back to itself

Acyclic.

Topological Sorting (ordering) of DAGs — for Only possible DAGs. $O(V+E)$

Given a DAG $G = (V, E)$, $V = (1, 2, \dots n)$

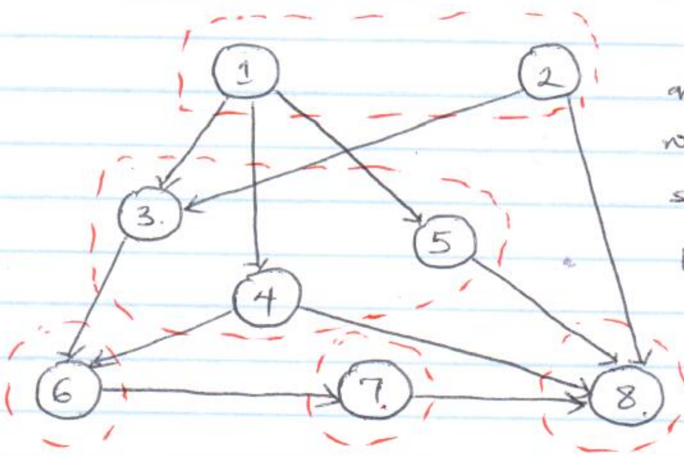
Ennmerate the vertices as $\{i_1, i_2, \dots i_n\}$ so that

for any edge (j, k) in E,

j appears before k in the ennmeration

$\Rightarrow$ Tasks with dependencies : before executing k, j is executed.

E.g.,



Assuming nodes are courses and edges are prerequisites, what is the minimum number of semesters to complete the program?

= 5 semester

Equivalent to finding the longest path in a DAG

If indegree $(j) = 0$, ~~loge~~ longest_path_to$(j) = 0$

If indegree $(k) > 0$, longest_path_to (k) is

$\underline{1}$ + max (longest_path_to(j) among all incoming neighbors j of k.

- Start with a vertex v with indegree of 0
- Print v, remove vertex v and all edges coming out of it.
- Repeat till all vertices are removed.

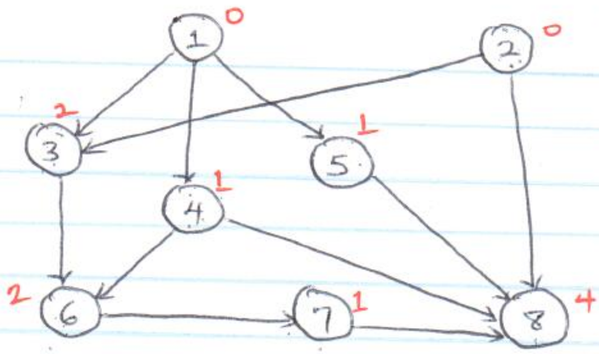
Alternative algorithm: run DFS, with time stamps → begin/end and place completed vertices on a stack.

* No known efficient solution for non-DAG graphs

Topological sort algorithm

```
function TopSort (G)
  for i = 1 to n
      indegree [i] = 0 ;
      LPT [i] = 0 ;
  for i = 1 to n
      for (i, j) in E
          indegree [j] = indegree [j] + 1
  for i = 1 to n
      if indegree [j] == 0
          add i to Queue
  while Queue is not empty
    j = remove_head (Queue)
    for (j, k) in E
        indegree [k] = indegree [k] - 1
        LPT [k] = max (LPT [k], 1 + LPT [j])
        if indegree [k] == 0
            add k to Queue.
```

Topological sort / longest path example



Indegree

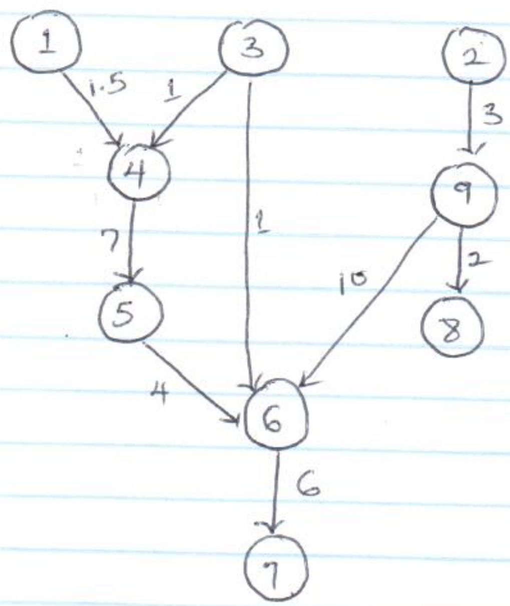
[redacted]

4	$x + y = z$
6	$z + i = j$
7	$j + e = f$
8	$z + f = k$

E.g. of dependencies

Node:	1	4	2	5	3	6	7	8
Longest path:	0	1	0	1	1	2	3	4

Weighted DAG example

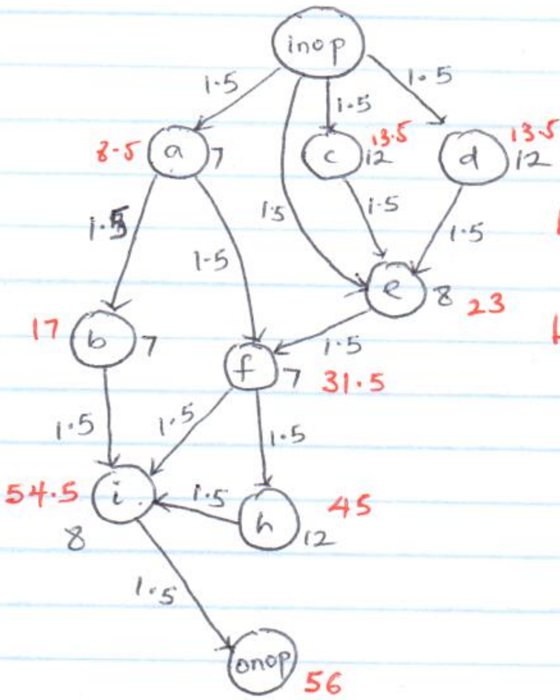


Topological ordering
3, 2, 9, 8, 1, 4, 5, 6, 7
or 1, 3, 4, 5, 2, 9, 8, 6, 7

Longest path = 19

Quiz 4

Longest path: DAG

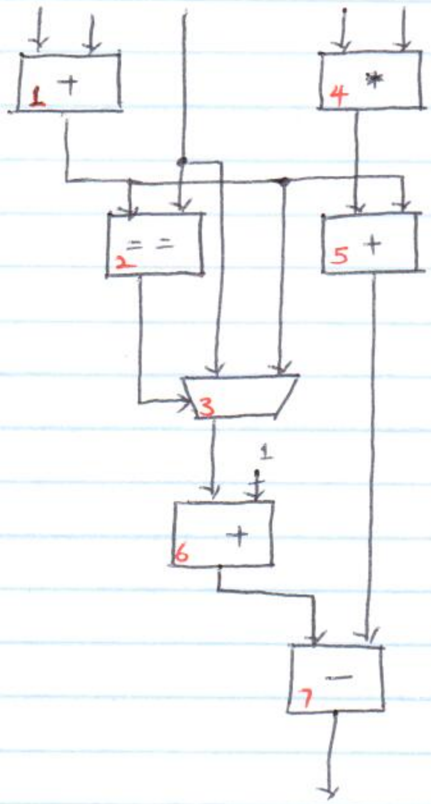


Topological ordering:
inop, a, c, d, e, b, f, h, i, onop

Longest path:
<inop, c, e, f, h, i, onop>
Length: 56

Graph representation of datapaths

Example 1



Delays

Adder : 5ns

Comp : 3.5ns

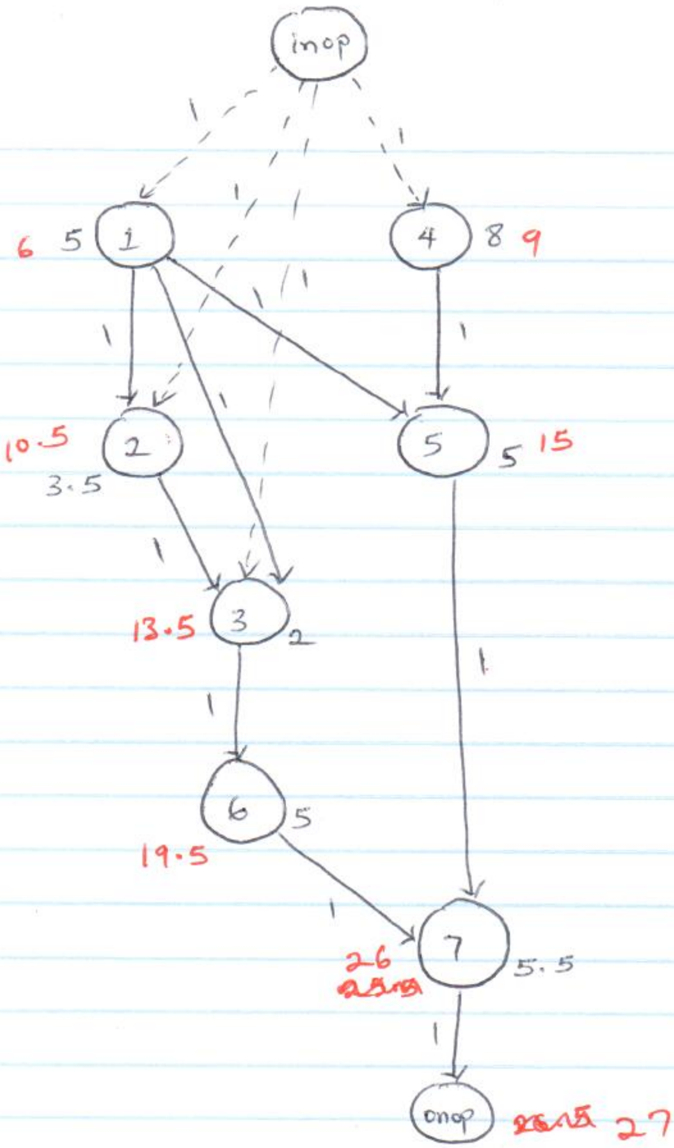
Multiplier : 8ns

Subtractor : 5.5ns

Multiplexer : 2ns

Wires : 1ns

Example 1 soln

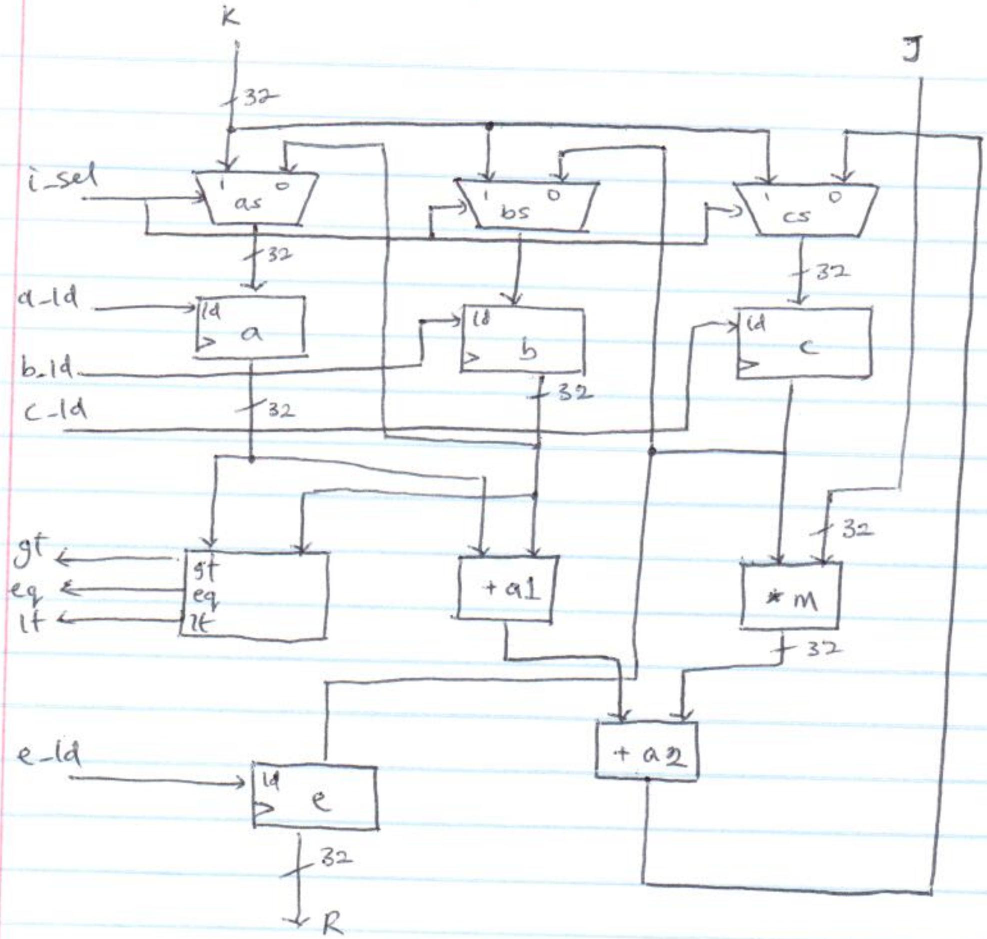


Topological order : inop, 1, 2, 3, 4, 5, 6, 7, onop

Longest path : ~~26.5ns~~ 27ns

Graph representation of datapaths

Example 2



Component delays

Parallel load register : 11

Multiplexer : 8

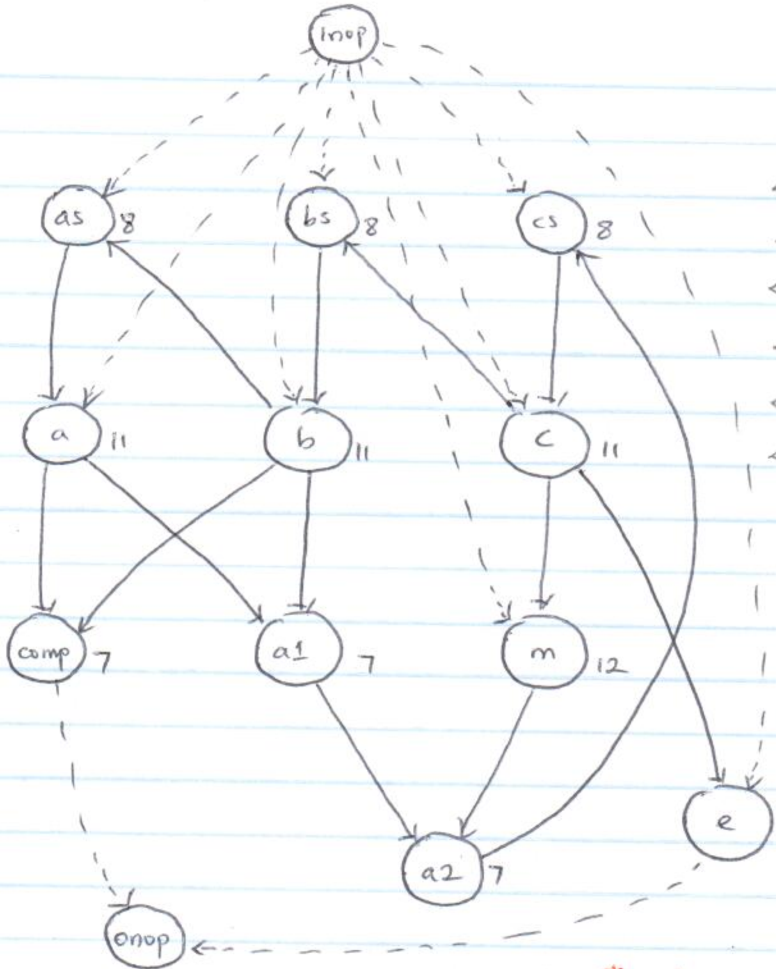
Multiplier : 12

Adder : 7

Comparator : 7

Graph representation of datapaths 2

Example 2 cont'd

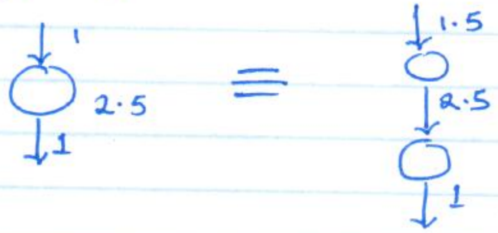


<inop, as, a>
<inop, a>
<inop, bs, b>
<inop, b>
<inop, cs, c>
<inop, c>
<inop, m, a2, cs, c>
<inop, r>
<a, comp, onop>
<a, a1, a2, cs, c>
<b, a1, a2, cs, c>
<c, m, a2, cs, c>*
<b, comp, onop>
<c, e>
<e, onop>

*Critical path = 38

~~Virt~~ Vertex weights can be translated to a graph with only edge weights.

E.g.,



- Find subgraphs that start at inop or reg and end at onop or reg.
- Find longest path in subgraph.