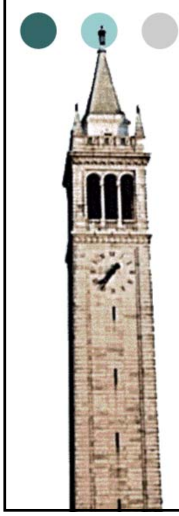


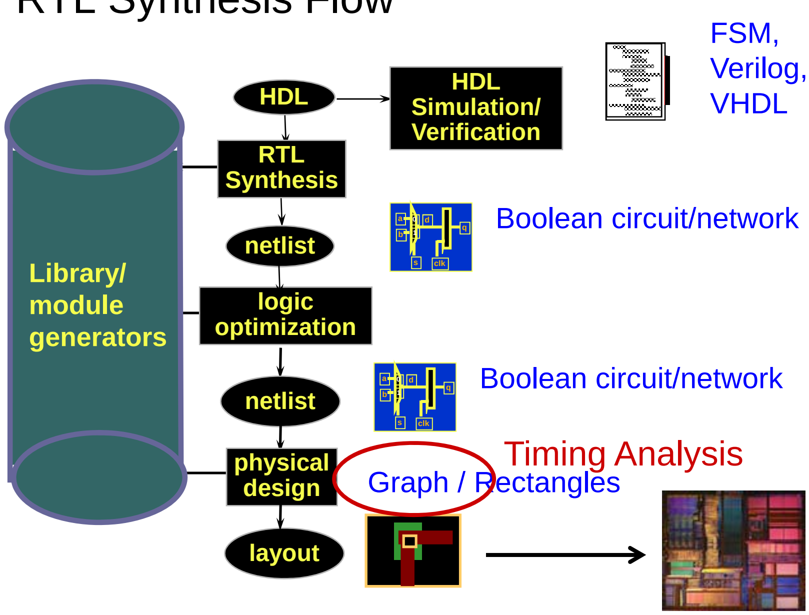
Fundamental Algorithms for System Modeling, Analysis, and Optimization

Edward A. Lee, Jaijeet Roychowdhury, Sanjit A. Seshia
UC Berkeley
EECS 144/244
Fall 2013

Lecture 3: Timing Analysis – Part 1

Thanks to Kurt Keutzer for several slides

RTL Synthesis Flow



Timing Analysis / Verification

Verifying a property about **system timing**

Arises in many settings:

- Integrated circuits
- Embedded software
- Distributed embedded systems
- Biological systems
- …

Illustrates many concepts of this course

- Graph algorithms
- Optimization
- SAT solving
- Numerical simulation

Timing Analysis for Digital ICs

(Clock) Speed is one of the major performance metrics for digital circuits

Timing Analysis = the process of verifying that a chip meets its speed requirement

- E.g., 1 GHz means that next-state function must be computed within 1 ns

Timing Analysis for Embedded Software

Latency from reading sensor values to writing actuator commands is determined by execution time of compute()

```
while(1) {
    read_sensors();

    compute();

    write_actuators()
}
```

This code is known to be terminating:

- loops with finite bounds
- no unbounded recursion

Typically:

- No interrupts/threads

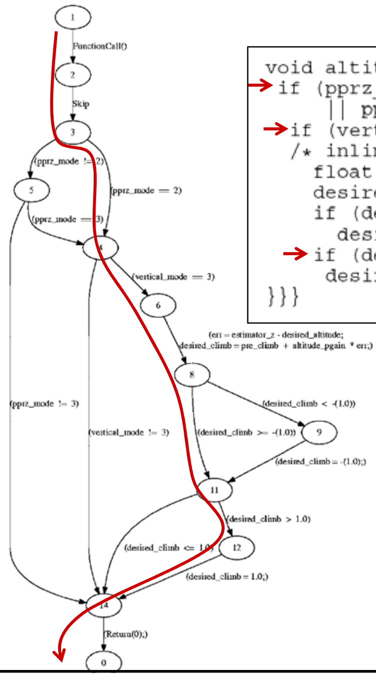
Example of ComputationalTask

altitude_control_task() from implementation of software controller of "Paparazzi UAV"

```
main.c:
…
while(1) {
  …
  periodic_task(…);
  …
}
```

```
switch(…) {
  case 0: …
    altitude_control_task(…);
    …
}
```

```
void altitude_control_task(void) {
 if (pprz_mode == PPRZ_MODE_AUTO2
     || pprz_mode == PPRZ_MODE_HOME) {
  if (vertical_mode == VERTICAL_MODE_AUTO_ALT) {
  /* inlined below: function  altitude_pid_run(); */
    float err = estimator_z - desired_altitude;
    desired_climb = pre_climb + altitude_pgain * err;
    if (desired_climb < -CLIMB_MAX)
      desired_climb = -CLIMB_MAX;
    if (desired_climb > CLIMB_MAX)
     desired_climb = CLIMB_MAX;
}}}
```



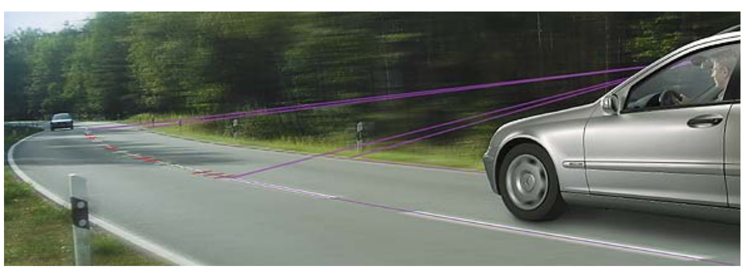
Path Space in Program

```
void altitude_control_task(void) {
 if (pprz_mode == PPRZ_MODE_AUTO2
      || pprz_mode == PPRZ_MODE_HOME) {
  if (vertical_mode == VERTICAL_MODE_AUTO_ALT) {
   /* inlined below: function  altitude_pid_run(); */
     float err = estimator_z - desired_altitude;
     desired_climb = pre_climb + altitude_pgain * err;
     if (desired_climb < -CLIMB_MAX)
       desired_climb = -CLIMB_MAX;
     if (desired_climb > CLIMB_MAX)
       desired_climb = CLIMB_MAX;
}}}
```

Must find:
Longest path in the control-flow graph (CFG)

Distributed Embedded Systems

- **Lane Keeping System (LKS)**



"… the system becomes an active lane keeping assistant as LKS, through an intervention in the steering. … the LKS measures the vehicle position relative to the lane, but offers active support in keeping the vehicle to the lane. However, the driver always retains the driving initiative, meaning that although he can feel the recommended steering reaction as a gentle movement of the steering wheel, his own decision takes priority at all times..."

Source: Continental Website

[Haibo Zeng, GM]

Automotive Architecture Design Process

- **Functional Model**

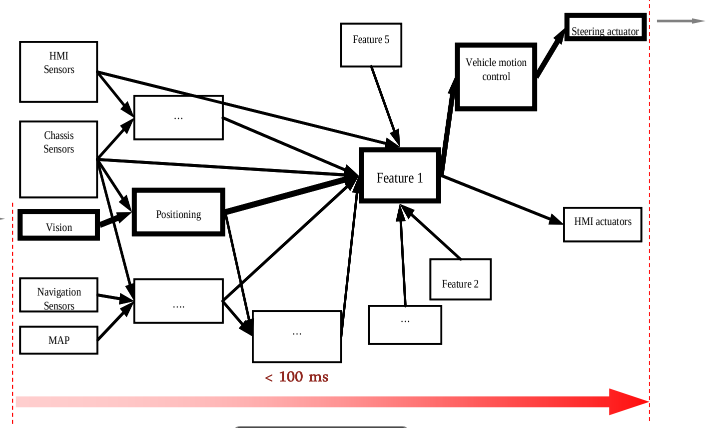
Functional Design

Input:

Context diagram with signal interfaces and timing requirement

Example:

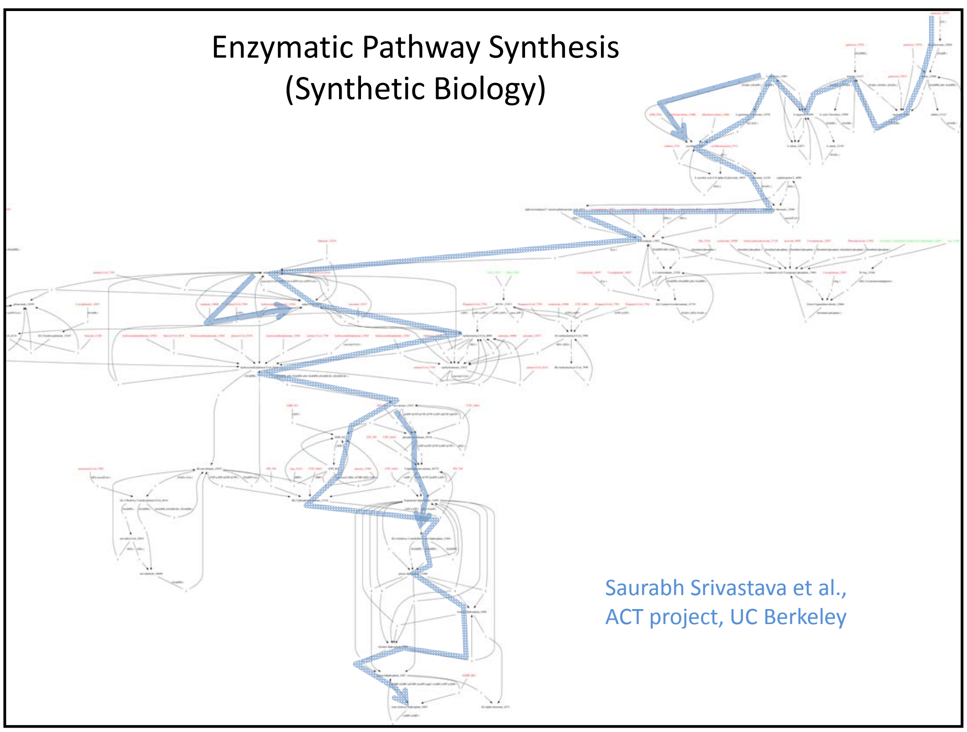
Lane keeping system



Requirements and specification definition

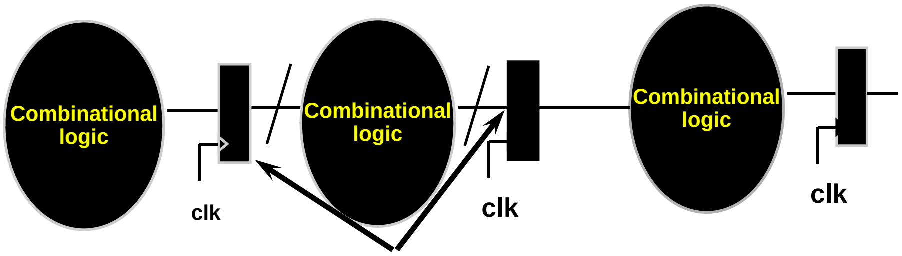
[Haibo Zeng, GM]

Enzymatic Pathway Synthesis (Synthetic Biology)



Saurabh Srivastava et al., ACT project, UC Berkeley

Static Timing Analysis for Circuits



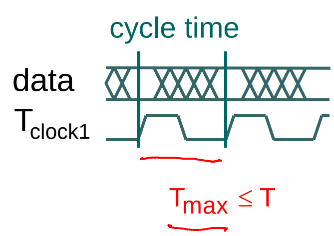
Determine fastest permissible clock speed (e.g. 1 GHz) by determining delay of longest path from register to register (e.g. 1ns.)

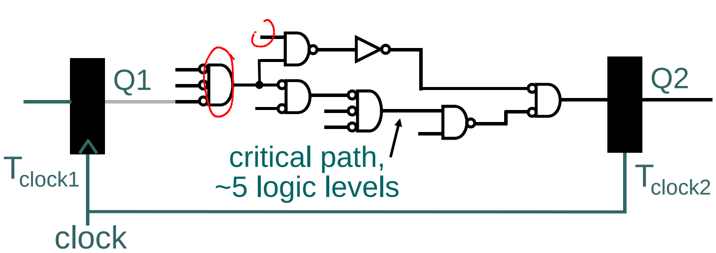
Cycle Time - Critical Path Delay

Cycle time (T) cannot be smaller than longest path delay (T_{max})

Longest (critical) path delay is a function of:

Total gate, wire delays

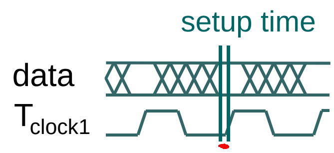




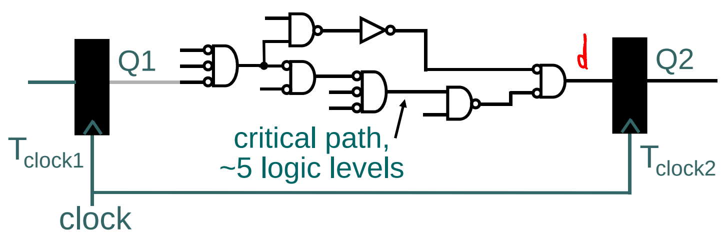
Cycle Time - Setup Time

For FFs to correctly latch data, it must be stable during:

Setup time (T_{setup}) *before* clock arrives



$T_{max} + T_{setup} \leq T$

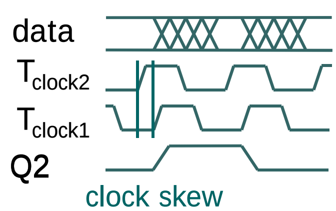


EECS 144/244, UC Berkeley: 13

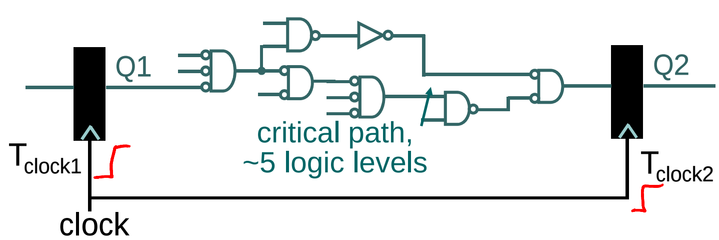
Cycle Time - Clock-skew

If clock network has unbalanced delay – clock skew

Cycle time is also a function of clock skew (T_{skew})



$T_{max} + T_{setup} + T_{skew} \leq T$

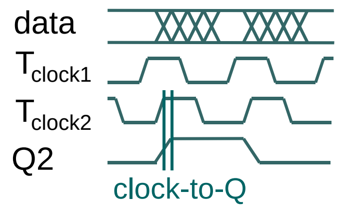


EECS 144/244, UC Berkeley: 14

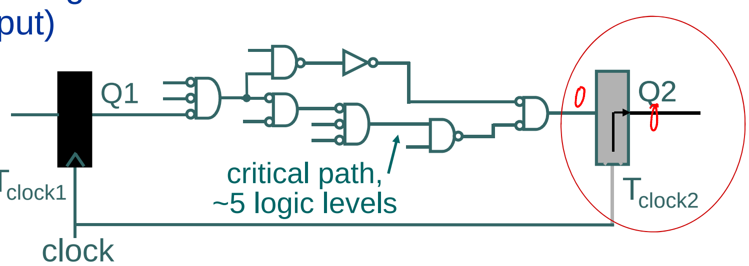
Cycle Time - Clock to Q

Cycle time is also a function of propagation delay of FF (*$T_{clk\text{-}to\text{-}Q}$*)

$T_{clk\text{-}to\text{-}Q}$: time from arrival of clock signal till change at FF output)



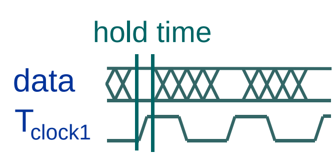
$T_{max} + T_{setup} + T_{skew} + T_{clk-to-Q} \le T$



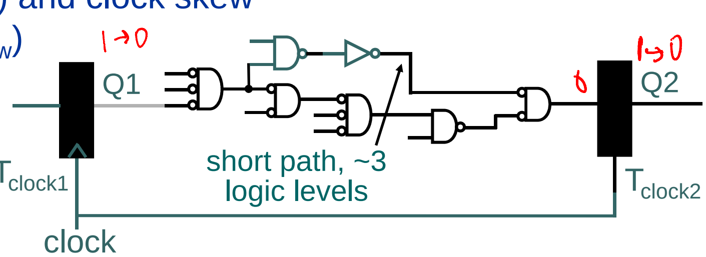
Min Path Delay - Hold Time

For FFs to correctly latch data, data must be stable during Hold time (T_{hold}) *after* clock arrives

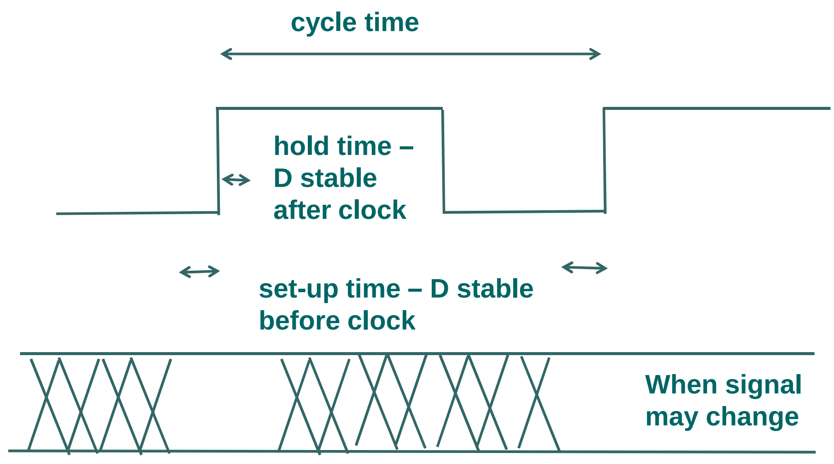
Determined by delay of shortest path in circuit (T_{min}) and clock skew (T_{skew})



$T_{min} \ge T_{hold} + T_{skew}$



A Quick Recap



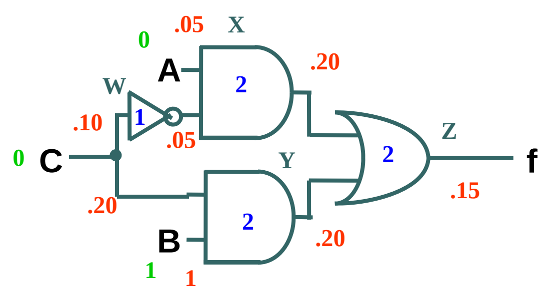
Example of a single phase clock

Modeling Timing in a Combinational Circuit

Arrival time in green

Interconnect delay in red

Gate delay in blue



What's the right mathematical object to use to represent this physical object?

Modeling - 1

Use a labeled *directed* graph

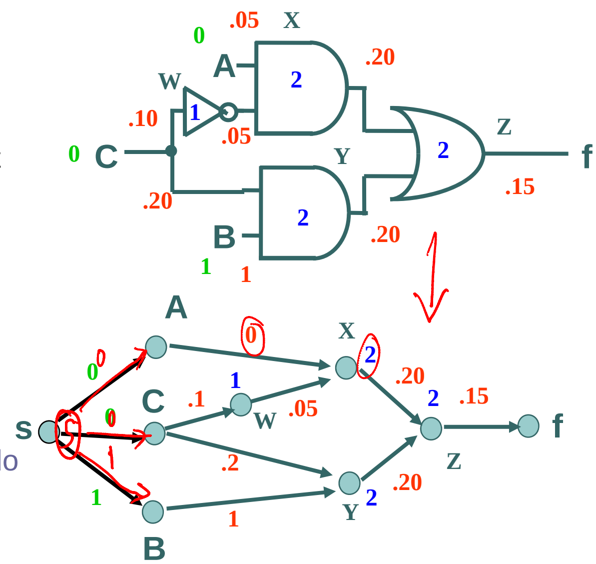
G = <*V*,*E*>

Vertices represent gates, primary inputs and primary outputs

Edges represent wires

Labels represent delays

Now what do we do with this?

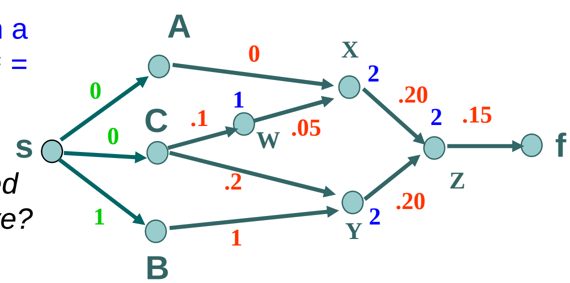


Modeling - 2

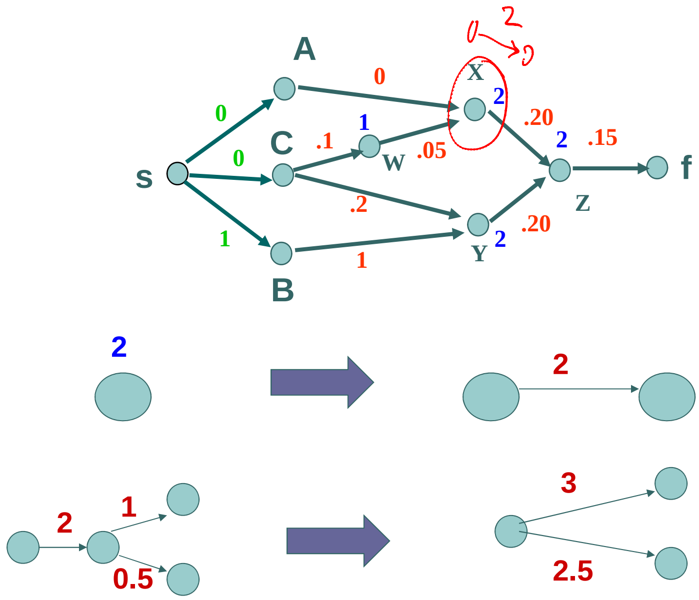
Find longest path in a *directed* graph *G* = <*V*,*E*>

What sort of directed graph do we have?

Is this in the standard form for a longest/shortest path problem?

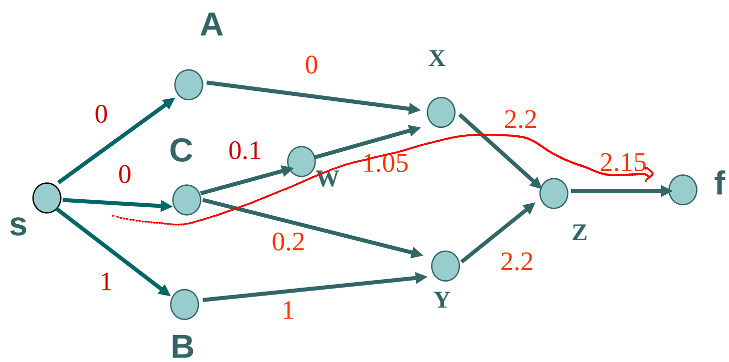


Split Nodes into Edges



EECS 144/244, UC Berkeley: 21

DAG with Weighted Edges



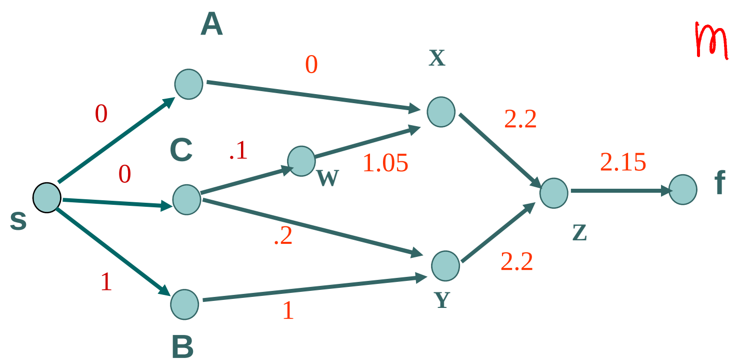
Problem: Find the longest (critical) path
from source s to sink f.

EECS 144/244, UC Berkeley: 22

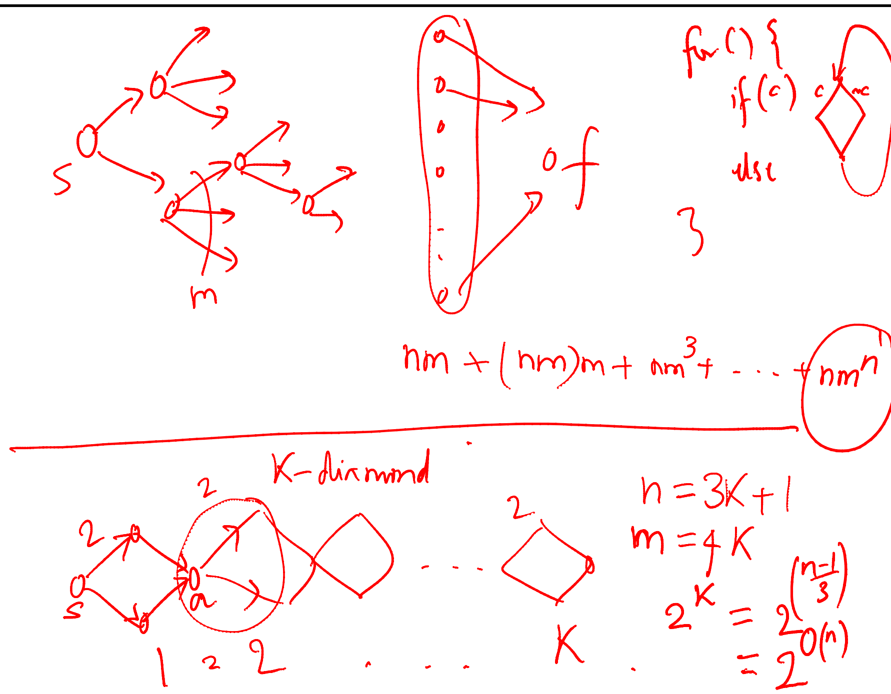
Naïve Approach: Enumerate Paths

How many paths in this example?
In the worst case?

n - nodes
m - edges



Problem:
Find the longest path from source s to sink f.



Algorithm 1: Longest path in a DAG

O(m+n)

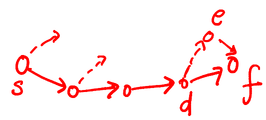
Critical Path Method **[Kirkpatrick 1966, IBM JRD]**

Let w(u,v) denote weight of edge from u to v

Steps:

1. Topologically sort vertices — DFS, finishing times (s . . . . . def)

   order: $v_1, v_2, \ldots, v_n$ $v_1 = s$, $v_n = f$



2. For each vertex v, compute

   d(v) = length of longest path from source s to v — O(m+n)

   $d(v_1) = 0$

   For i = 2..n (n-1)

   $d(v_i) = \max_{\text{all incoming edges } (u, v_i)} d(u) \oplus w(u,v_i)$ — O(m) + O(n)

   # incoming edges to v_i = in-degree

Algorithm 1: Longest path in a DAG

Critical Path Method **[Kirkpatrick 1966, IBM JRD]**

Let w(u,v) denote weight of edge from u to v — Find: d(f)

Steps:

1. Topologically sort vertices

   order: $v_1, v_2, \ldots, v_n$ $v_1 = s$, $v_n = f$

Time Complexity?

O(m+n)

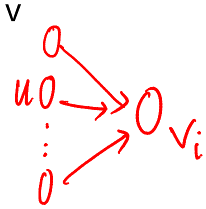
2. For each vertex v, compute

   d(v) = length of longest path from source s to v

   $d(v_1) = 0$

   For i = 2..n

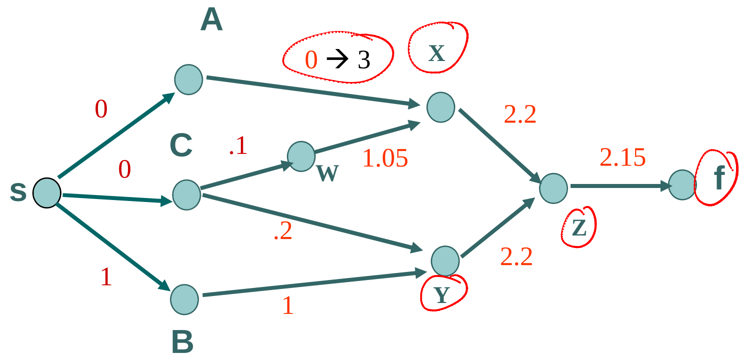
   $d(v_i) = \max_{\text{all incoming edges } (u, v_i)} d(u) + w(u,v_i)$



Run the CPM on our example

Algorithm 2: Incremental longest path in a DAG

Suppose only a few weights/nodes/edges change.
How do we recompute the longest path efficiently?



Exercise: READ HANDOUT

Algorithm 3: Top k longest paths in a DAG

Often, we don't want just the longest path

Want to find the *top* k *longest paths*

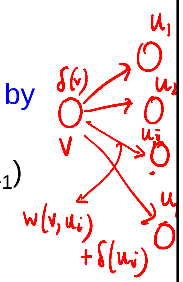
How to do this efficiently? (i.e., polynomial in n, m, k)

Key insight/idea:
- The 2nd longest path shares a prefix with the longest path.
- From each node along longest path, keep track of the "next longest" route to sink f.

Algorithm 3: Top k longest paths in a DAG

Pre-compute phase:

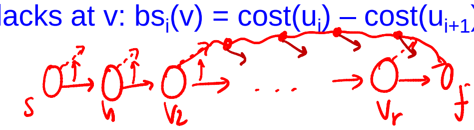
1. For all v, compute
   $\delta(v)$ = length of longest path from vertex v to sink f.
   How to compute this efficiently? Complexity?
2. At each vertex v: order successor vertices $u_1, u_2, \ldots, u_k$ by decreasing $cost(u_i) = w(v, u_i) + \delta(u_i)$
3. Compute 'branch' slacks at v: $bs_i(v) = cost(u_i) - cost(u_{i+1})$
   $bs_k(v) = cost(u_k)$



Algorithm 3: Top k longest paths in a DAG

Pre-compute phase:

1. $\delta(v)$ = length of longest path from vertex v to sink f.
   How to compute? Complexity?
2. At each vertex v: order successor vertices $u_1, u_2, \ldots, u_k$ by decreasing $cost(u_i) = w(v, u_i) + \delta(u_i)$
3. Compute 'branch' slacks at v: $bs_i(v) = cost(u_i) - cost(u_{i+1})$
   $bs_k(v) = cost(u_k)$



Main phase:

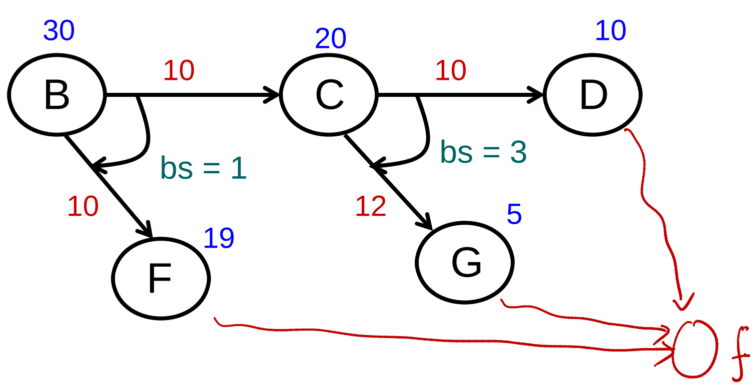
1. Let longest path $p = s, v_1, v_2, \ldots, v_r, f$
2. For 2nd (next) longest, order nodes according to branch slacks: $bs1(s), bs1(v_1), \ldots bs1(v_r)$, and pick the smallest. The corresponding successor indicates the next longest path.
3. For 3rd longest, add nodes along 2nd longest to the ordered node list, maintaining order. Go back to step 2 (check 'next' branch slack). (see handout for details)

Algorithm 3: Top k longest paths in a DAG

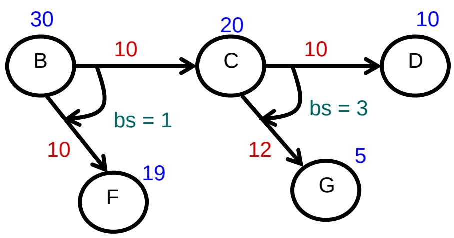
Main phase:

1. Let longest path p = s, v_1, v_2, …, v_r, f
2. For 2nd (next) longest, order nodes according to branch slacks: bs1(s), bs1(v_1), … bs1(v_r), and pick the smallest. The corresponding successor indicates the next longest path.
3. For 3rd longest, add nodes along 2nd longest to the ordered node list, maintaining order. Go back to step 2 (check 'next' branch slack). (see handout for details)

Example: Top k longest paths in a DAG



Example: Top k longest paths in a DAG



① B → C → D ⇝ f

② B → F ⇝ f

⋮

Next Lectures

Dealing with the non-idealities of the current model

- True and False paths
- Timing variability

Retiming (timing for sequential circuits)

References:

- In-class Handout: Chapter 5: “Timing Analysis for Combinatonal Circuits” of “Timing” by S. Sapatnekar
- Posted optional reading: Chapter 15 of “Introduction to Embedded Systems” by E. A. Lee and S. A. Seshia, http://leeseshia.org