

Fence-Insertion for Structured Programs

Arash Pourdamghani

Mohammad Taheri

Mohsen Lesani

Developing a Distributed System!



Looking Deeper

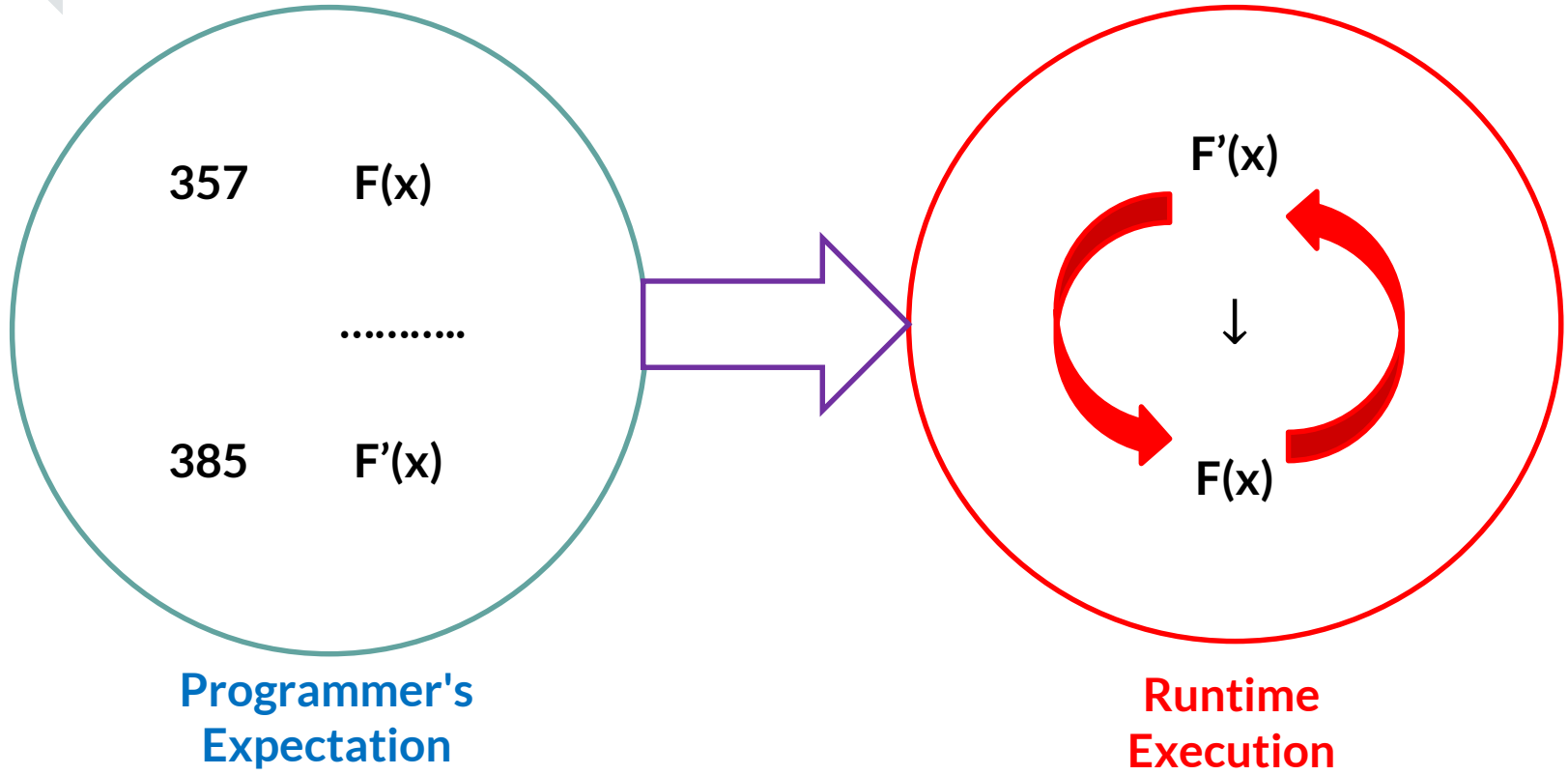
357 F(x)

.....

385 F'(x)

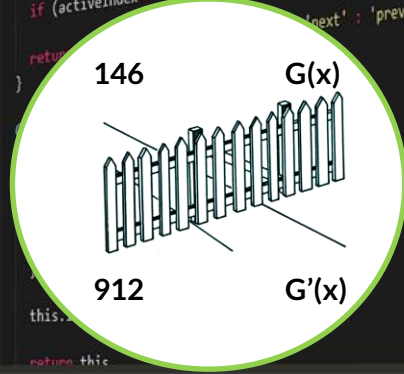
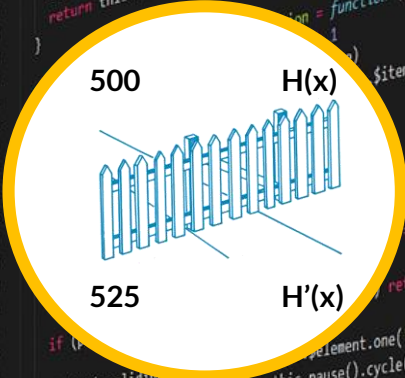
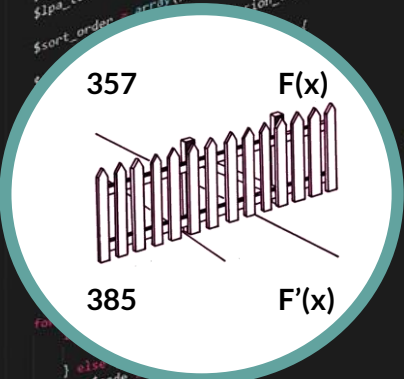
```
354 }
355 this.$items = item.parent().children();
356 return this.$items.index(item || this.$active);
357 }
358 }
359 }
360 Carousel.prototype.getItemForDirection = function (direction, active) {
361   var delta = direction === 'prev' ? -1 : 1;
362   var activeIndex = this.getItemIndex(active);
363   var itemIndex = (activeIndex + delta) % this.$items.length;
364   return this.$items.eq(itemIndex);
365 }
366 }
367 Carousel.prototype.to = function (pos) {
368   var that = this;
369   var activeIndex = this.getItemIndex(this.$active = this.$element.find('.item.active'));
370   if (pos > (this.$items.length - 1) || pos < 0) return;
371   if (this.sliding) return this.$element.one('slid.bs.carousel', function () { that.to(pos) });
372   if (activeIndex == pos) return this.pause().cycle();
373   return this.slide(pos > activeIndex ? 'next' : 'prev', this.$items.eq(pos));
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
```

Out of Order Execution



Fence Insertion

```
177 $totals = $total;
178 $taxes => $total;
179 'total' => $total;
180 );
181 $old_taxes = $taxes;
182 $ipa_tax = array();
183 $sort_order = array();
184 $extension->getExtensions('total');
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205 } else {
206     $code
207 }
208 if ($this->config->get($code -> '_status')) {
209     $this->load->model('extension/total/' . $code);
210
211     // We have to put the totals in an array so that they pass
212     // by reference.
213     $this->['model_extension_total_' . $code]->getTotal($
214         total_data);
215
216     if (!empty($totals[count($totals) - 1]) && !isset($totals[
217         count($totals) - 1]['code'])) {
218         $totals[count($totals) - 1]['code'] = $code;
219     }
220
221     $tax_difference = 0;
222
223     foreach ($taxes as $tax_id => $value) {
224         if (isset($old_taxes[$tax_id])) {
```



Our Solution

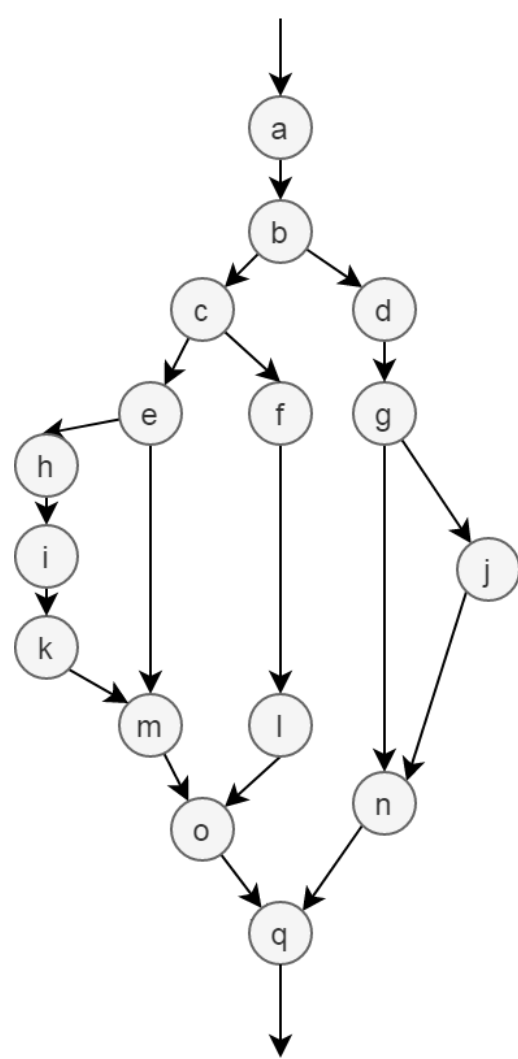




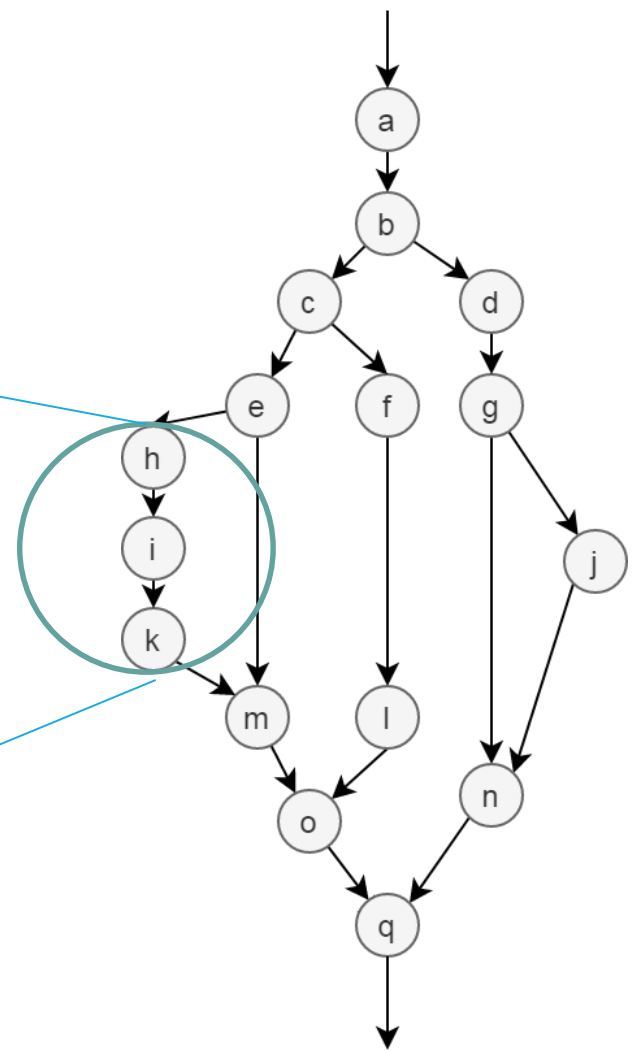
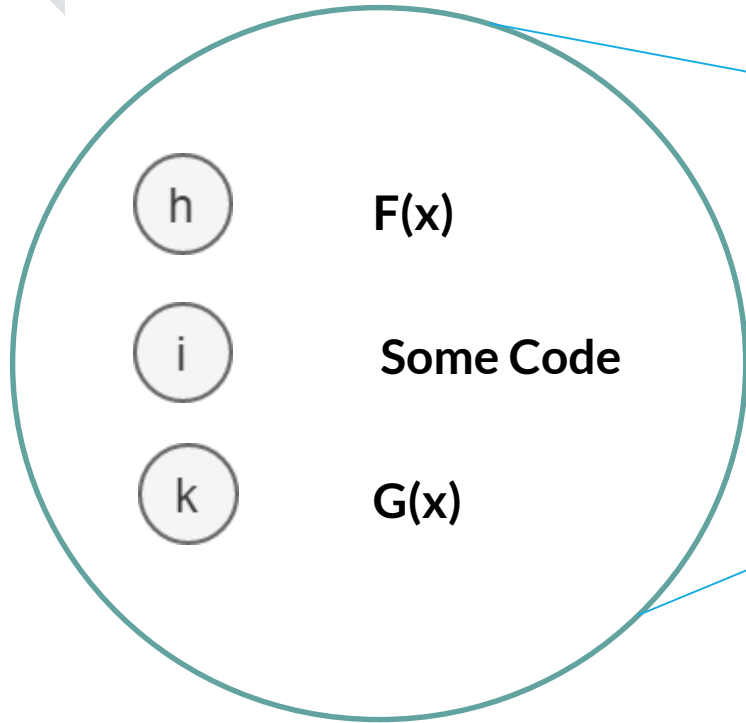
Outline



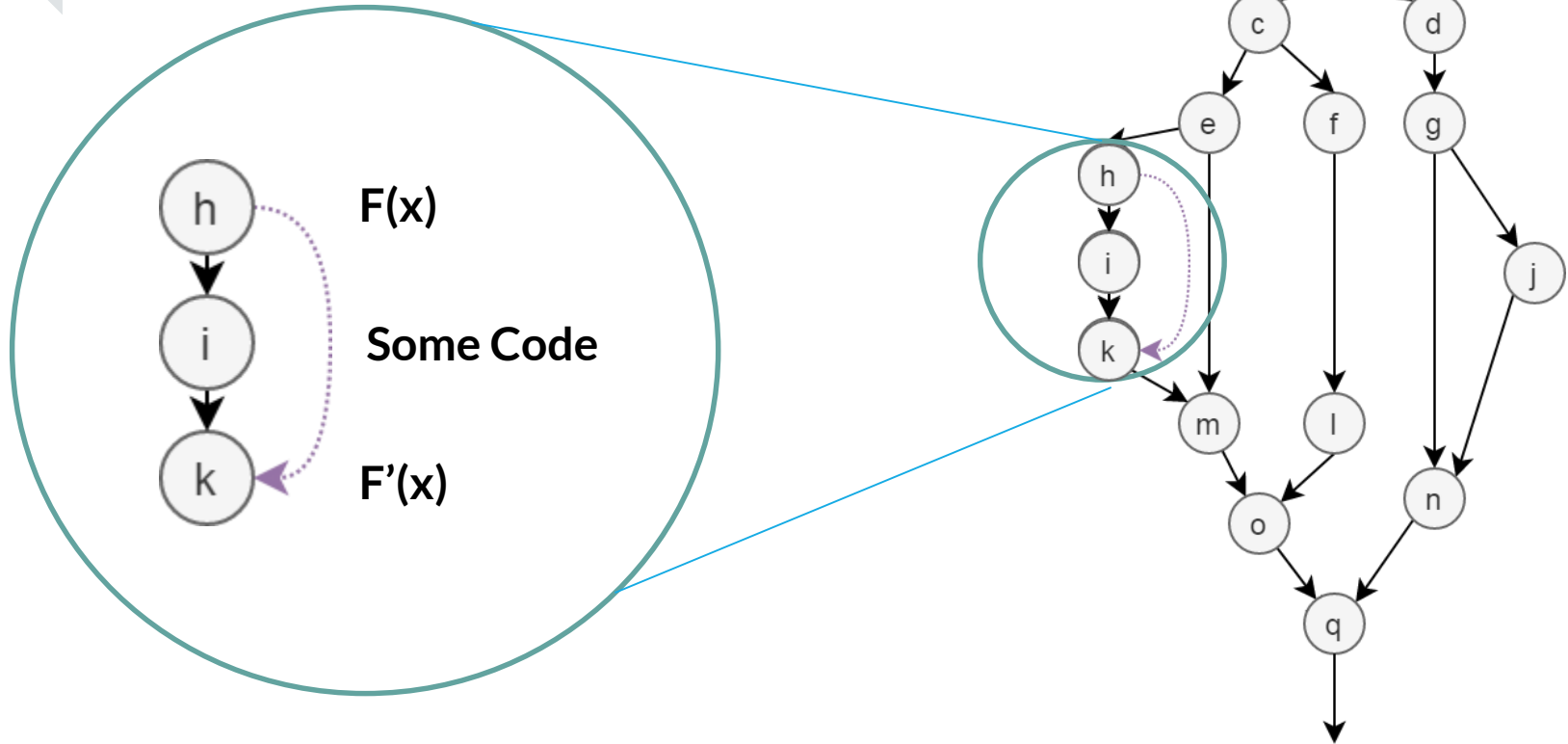
CFG



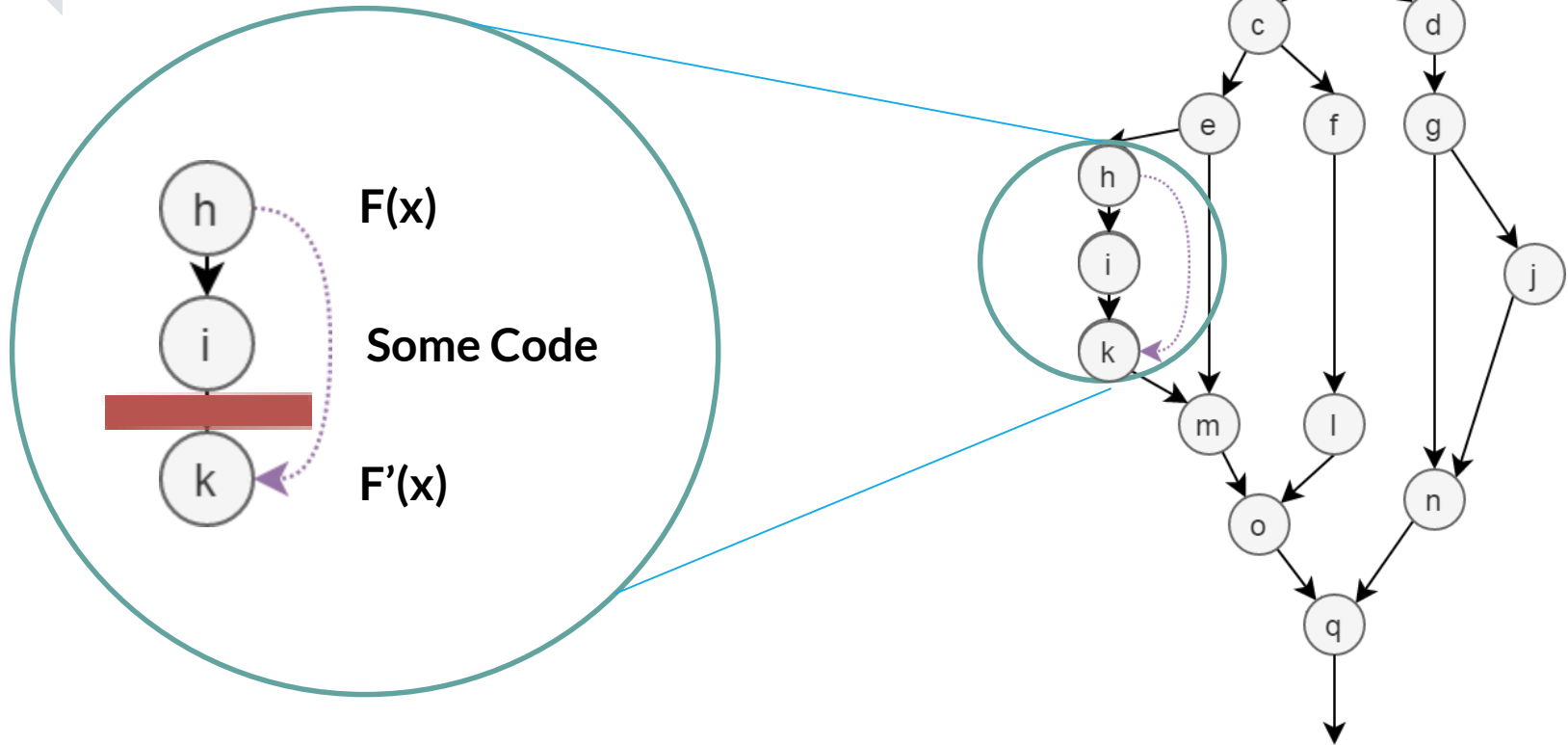
A Closer Look



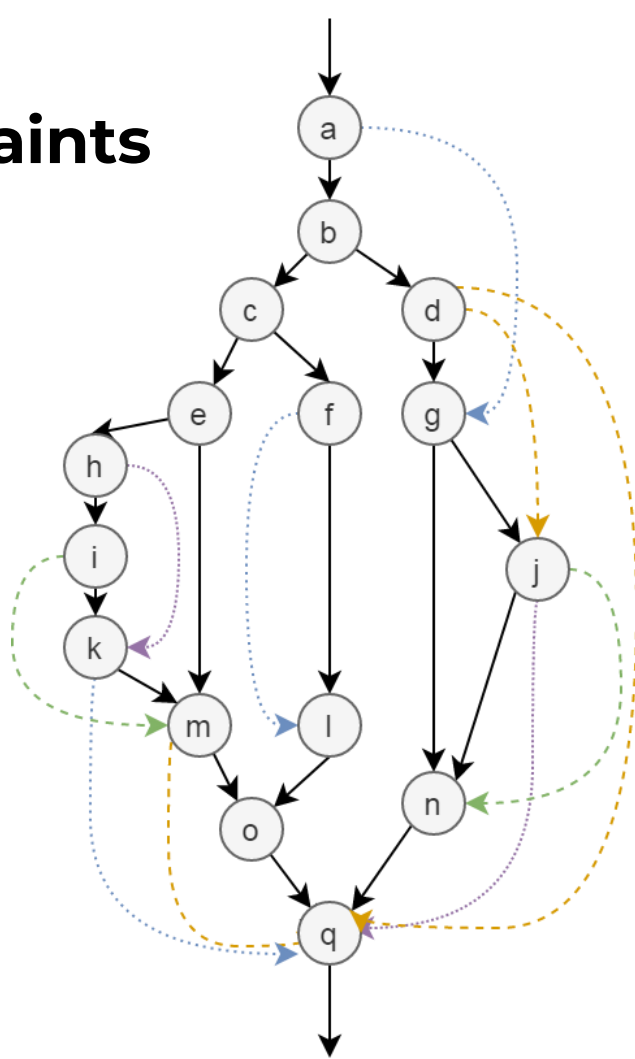
Maintaining Correctness



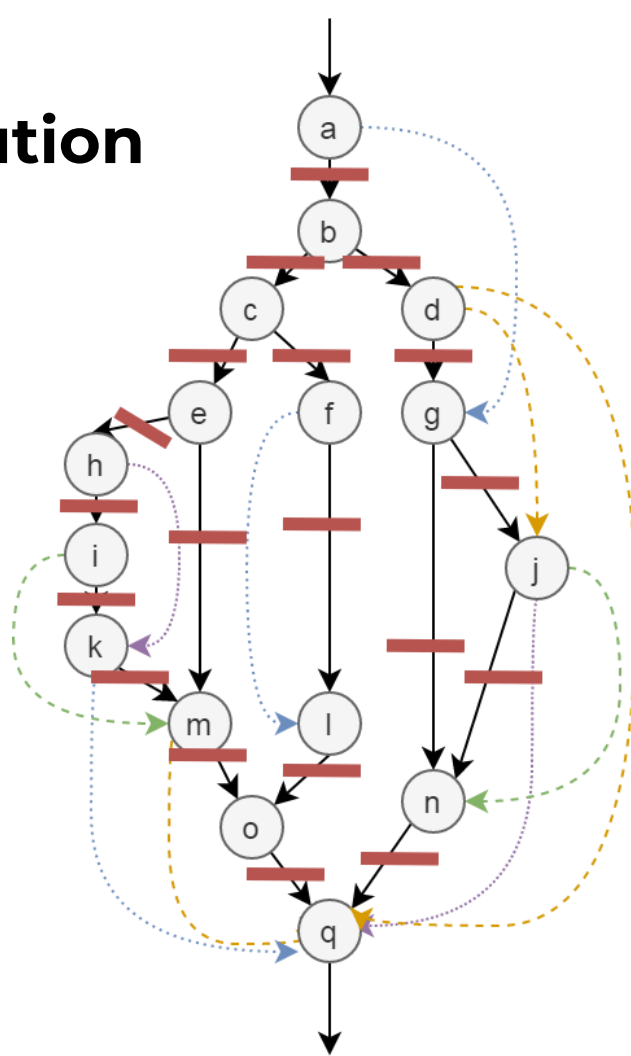
Fence Insertion



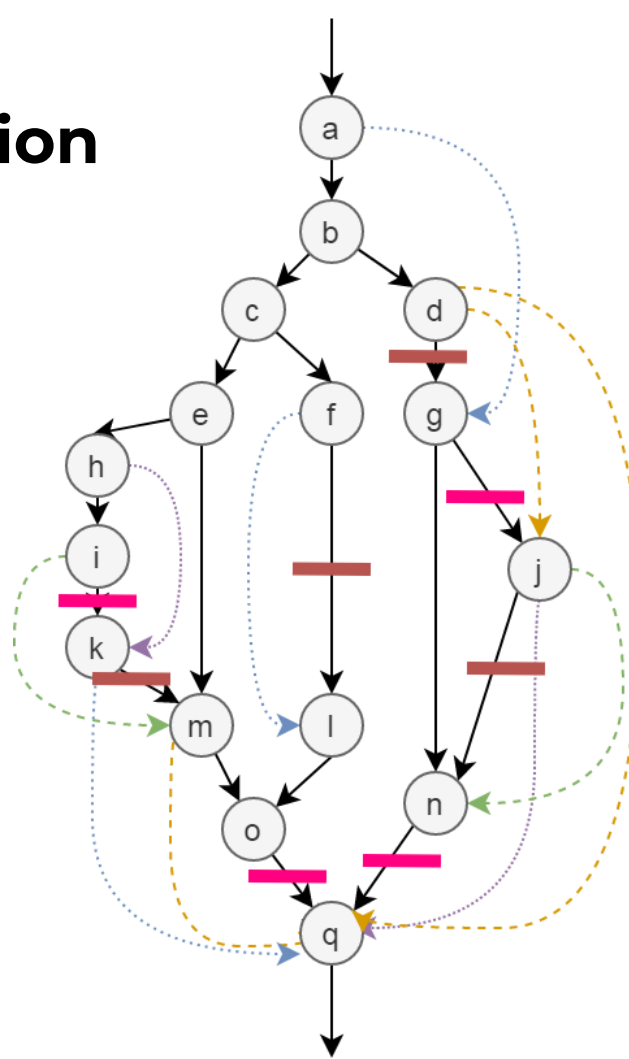
All Constraints



Naive Solution

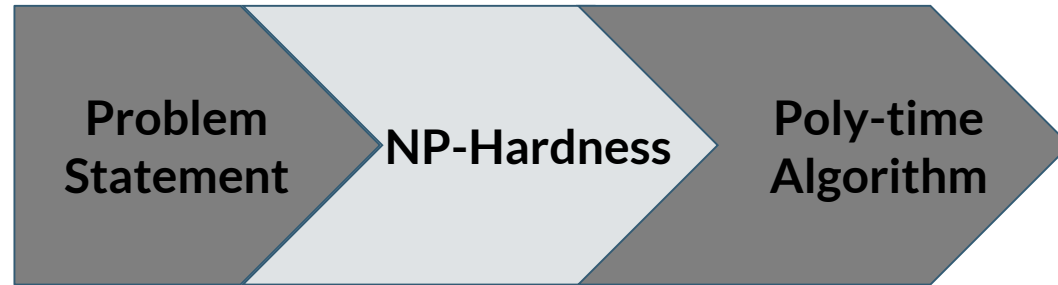


Optimization





Outline





Reduction from Minimum Set Cover

$$U = \{u_1, u_2, u_3, u_4, \dots, u_n\}$$

$$S = \{S_1, S_2, \dots, S_m\}$$

Example: $S_1 = \{u_1, u_2, u_3\}, S_2 = \{u_2, u_4\}$

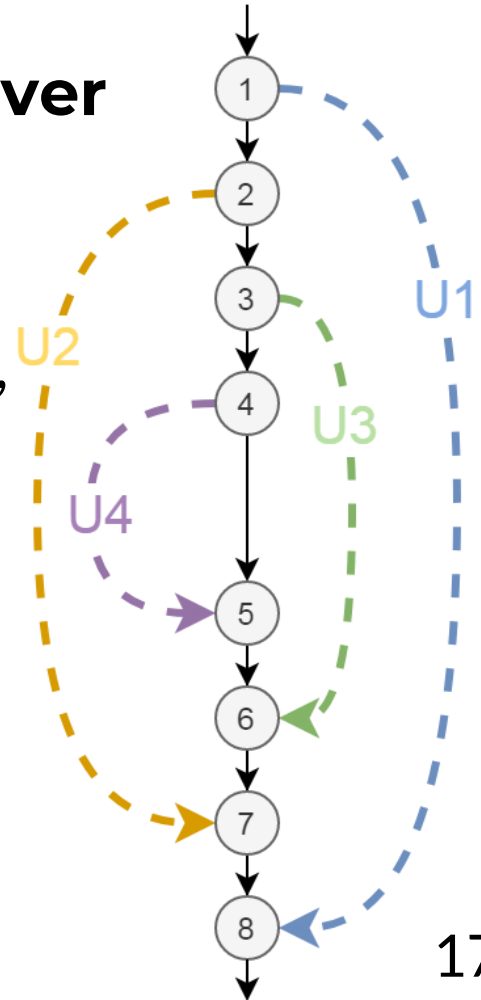
Reduction from Minimum Set Cover

$U = \{u_1, u_2, u_3, u_4, \dots, u_n\}$: Constraint Types,

Straight line program with $2n$ instructions,

Constraints with type u_i on (v_i, v_{2n-i+1}) ,

$(v_{\frac{n}{2}}, v_{\frac{n}{2} + 1})$: The middle edge





Reduction from Minimum Set Cover

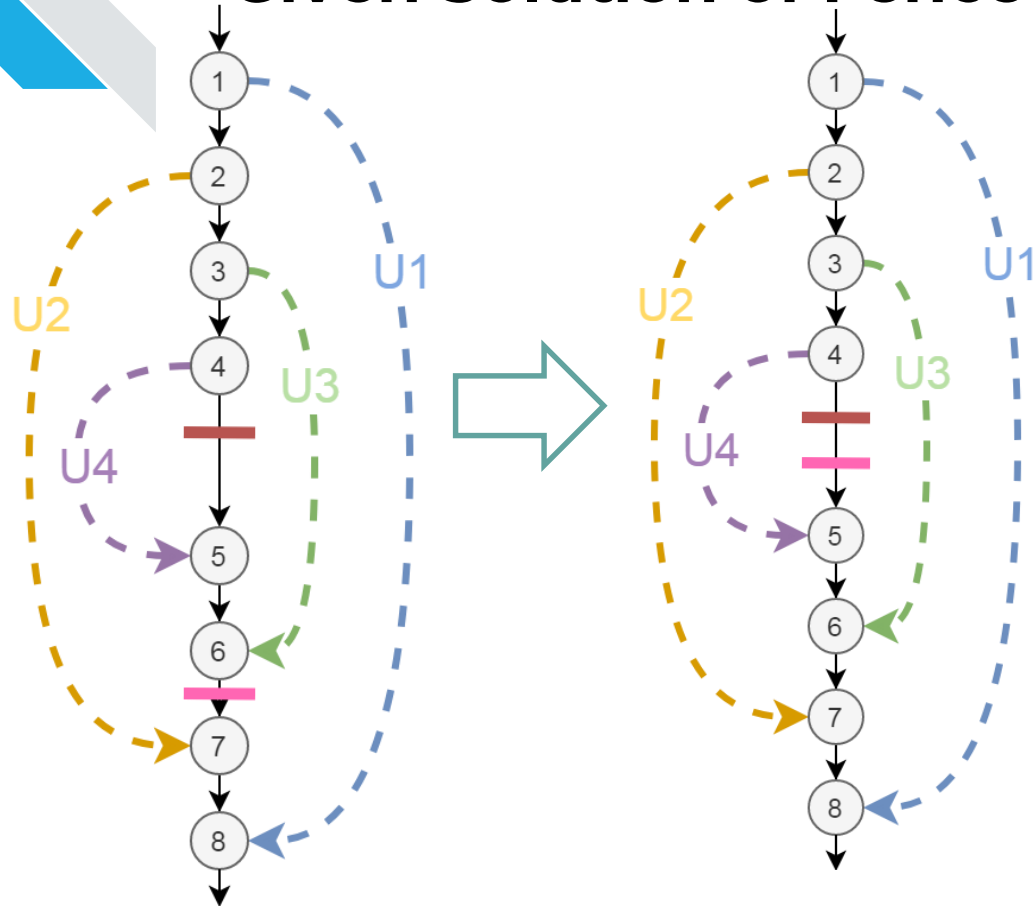
$S = \{S_1, S_2, \dots\}$: Fence Types

We could have many fences of S_i

Example: $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_2, u_4\}$



Given Solution of Fence Insertion



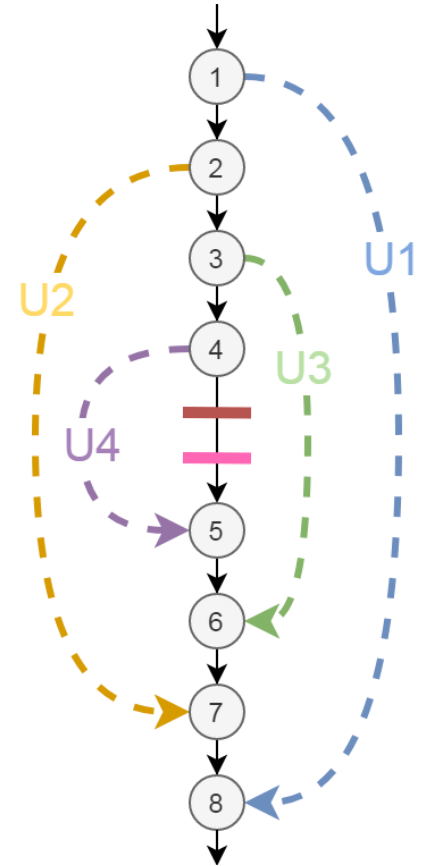
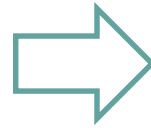
$$U = \{u_1, u_2, u_3, u_4, \dots\}$$

$$S = \{S_1, S_2, \dots\}$$

Given Solution of Minimum Set Cover

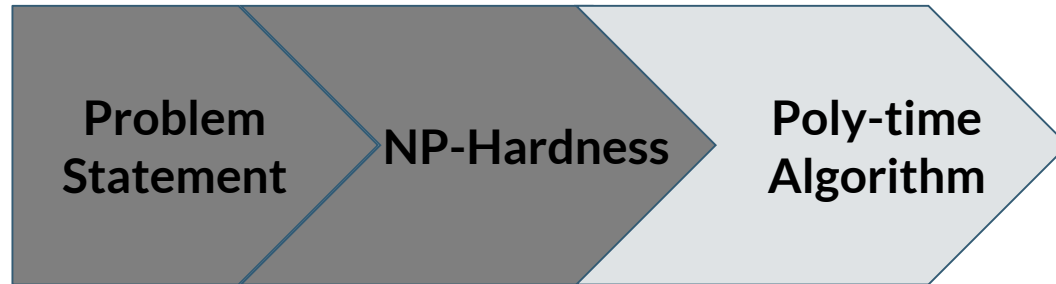
$$U = \{U_1, U_2, U_3, U_4, \dots\}$$

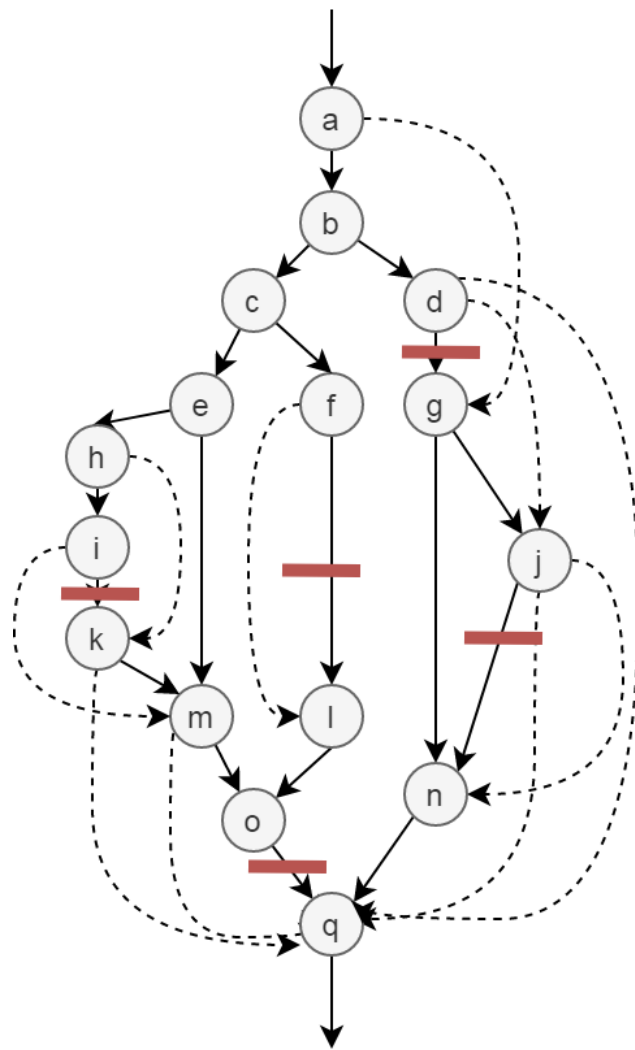
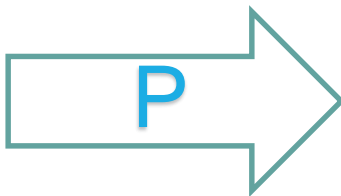
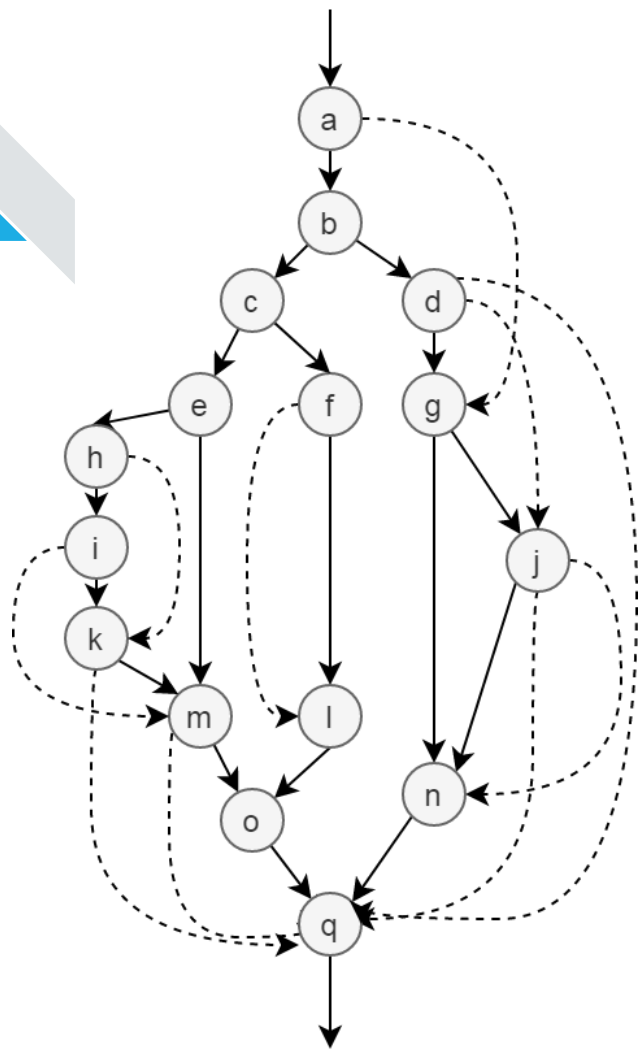
$$S = \{S_1, S_2, \dots\}$$





Outline







Algorithm Overview

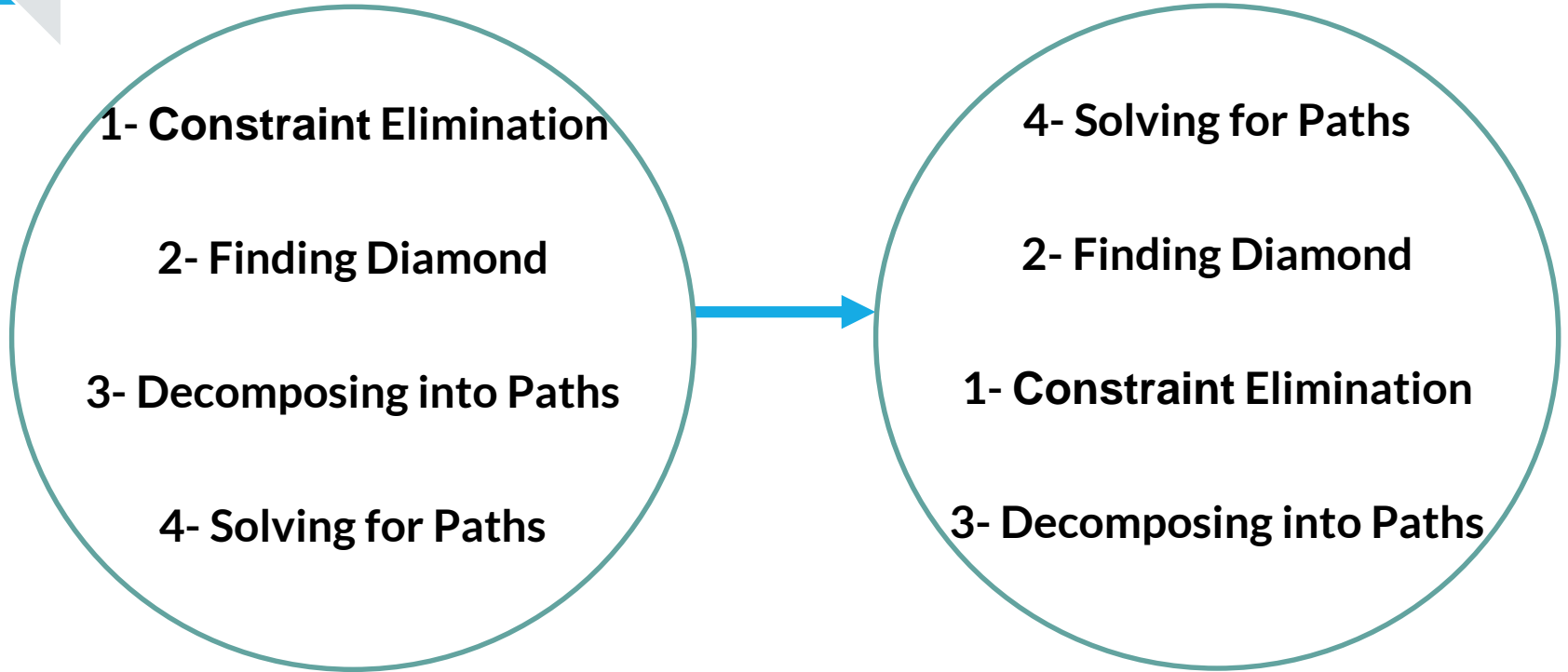
1- Constraint Elimination

2- Finding Diamond

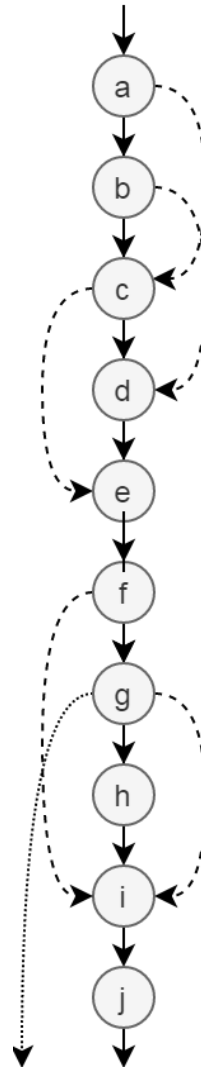
3- Decomposing into Paths

4- Solving for Paths

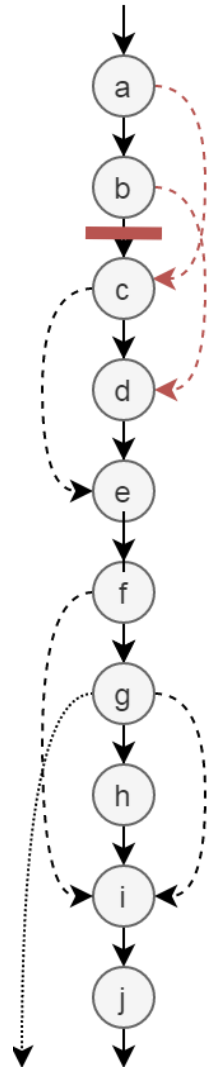
Out of Order Execution!



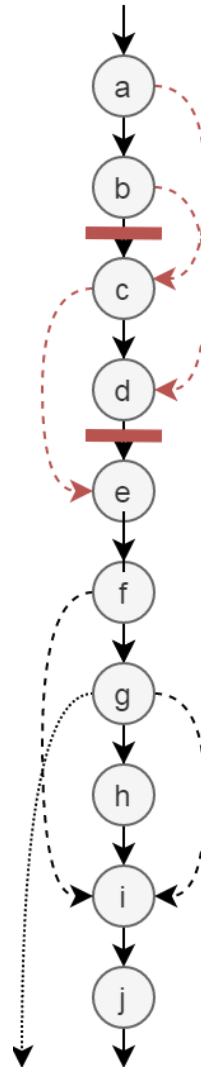
A Simple Path



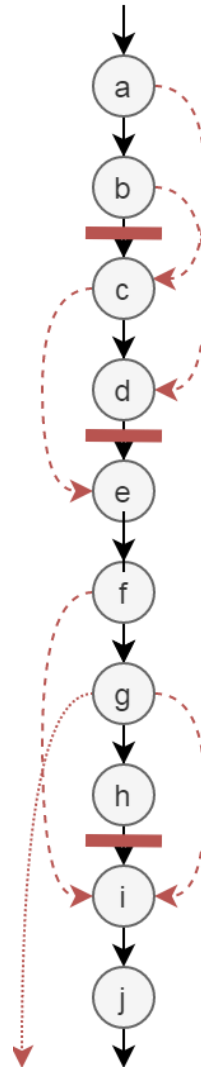
A Simple Path



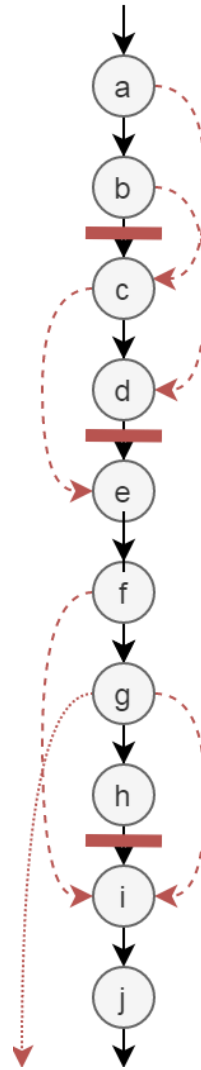
A Simple Path



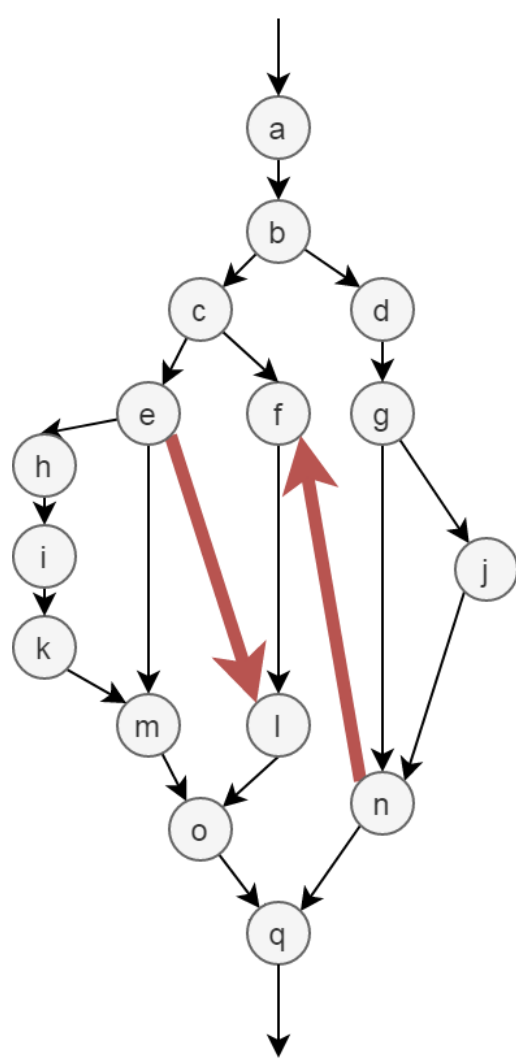
A Simple Path



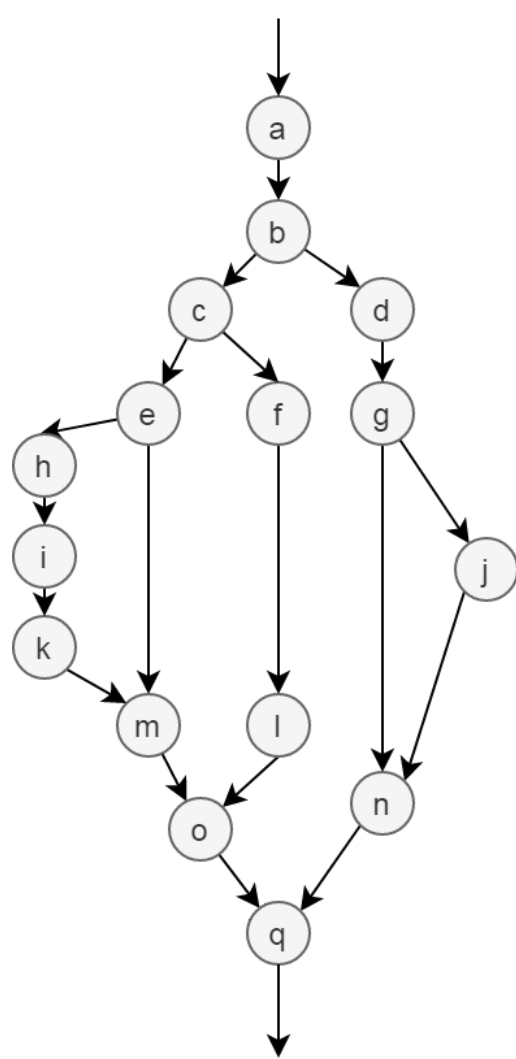
A Simple Path



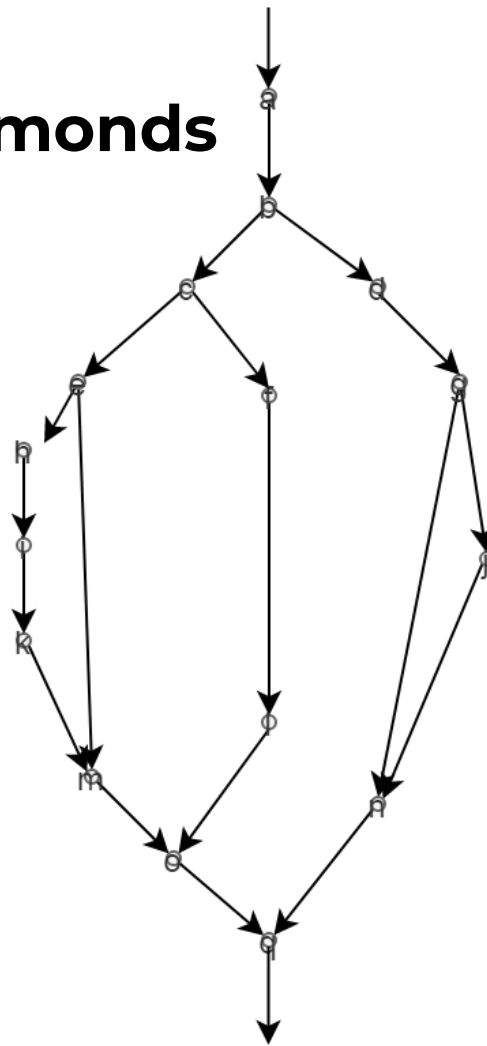
Structured



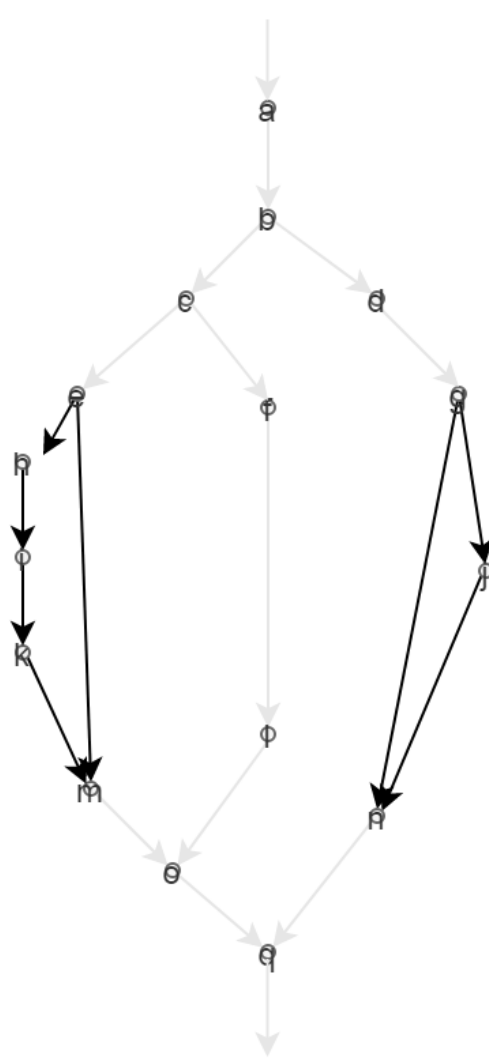
A(C)FG



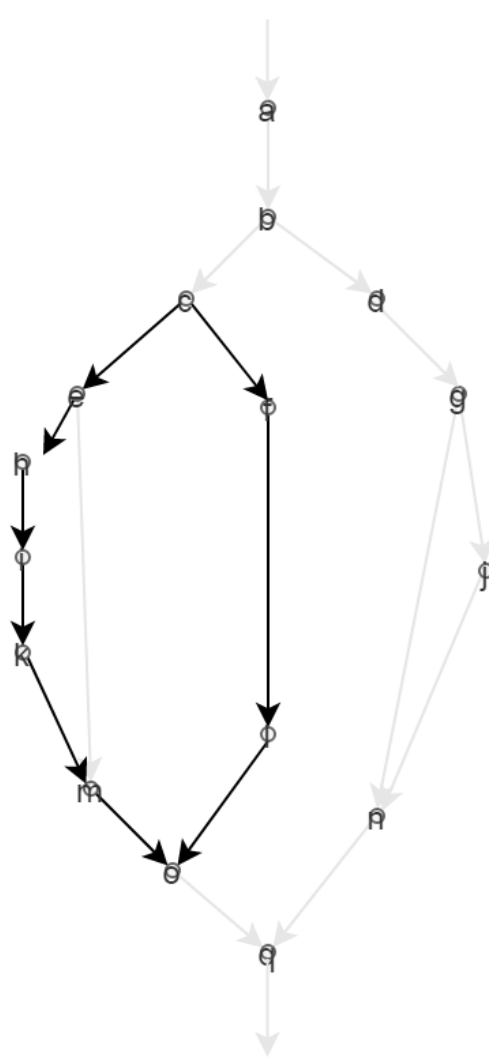
Finding Diamonds



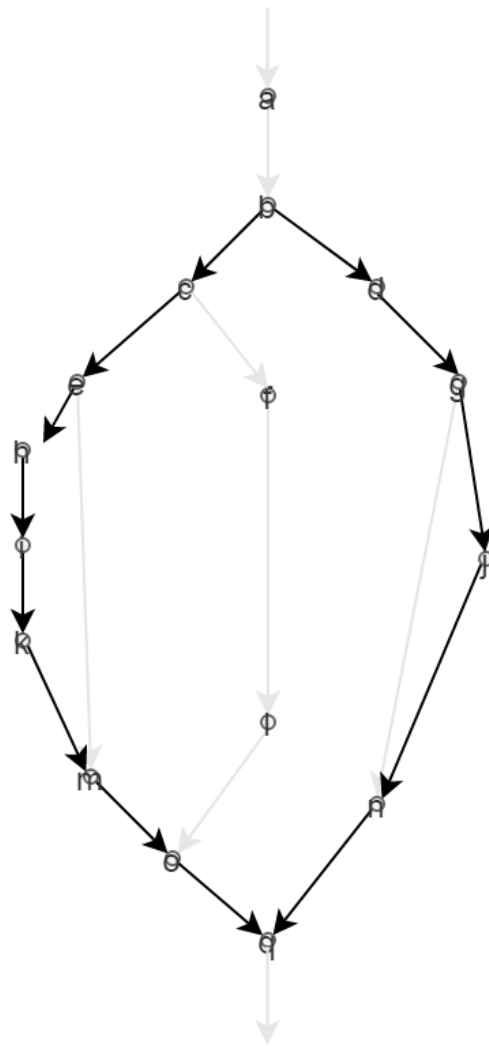
Level 0



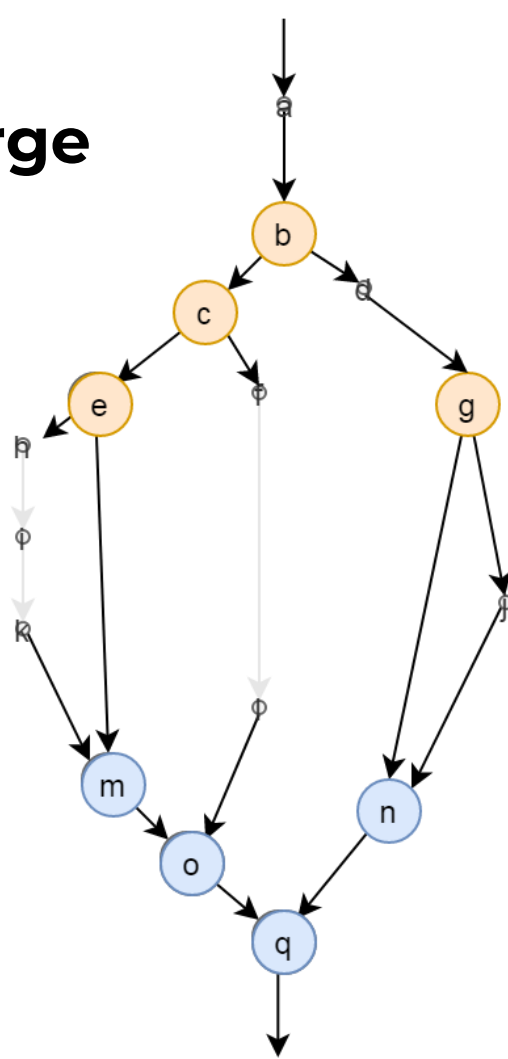
Level 1



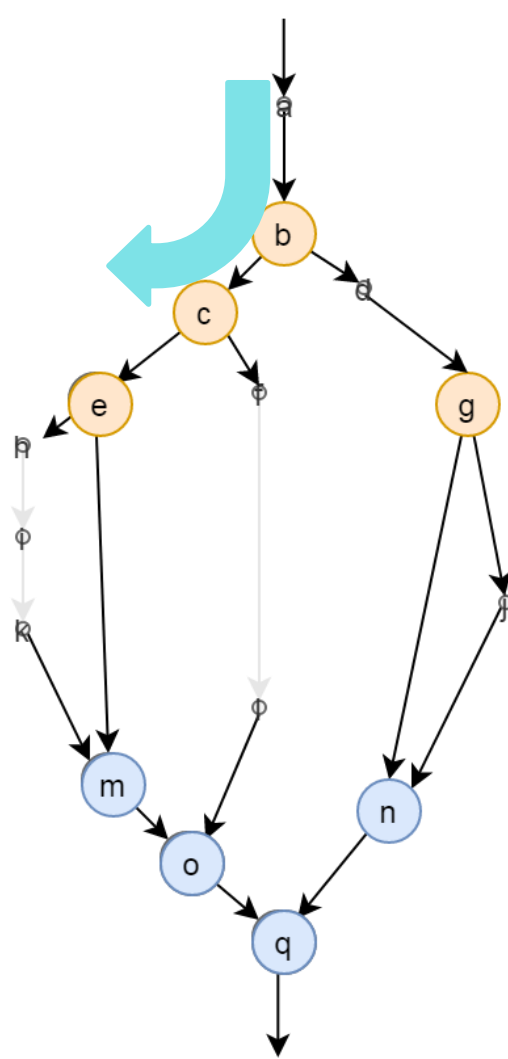
Level 2



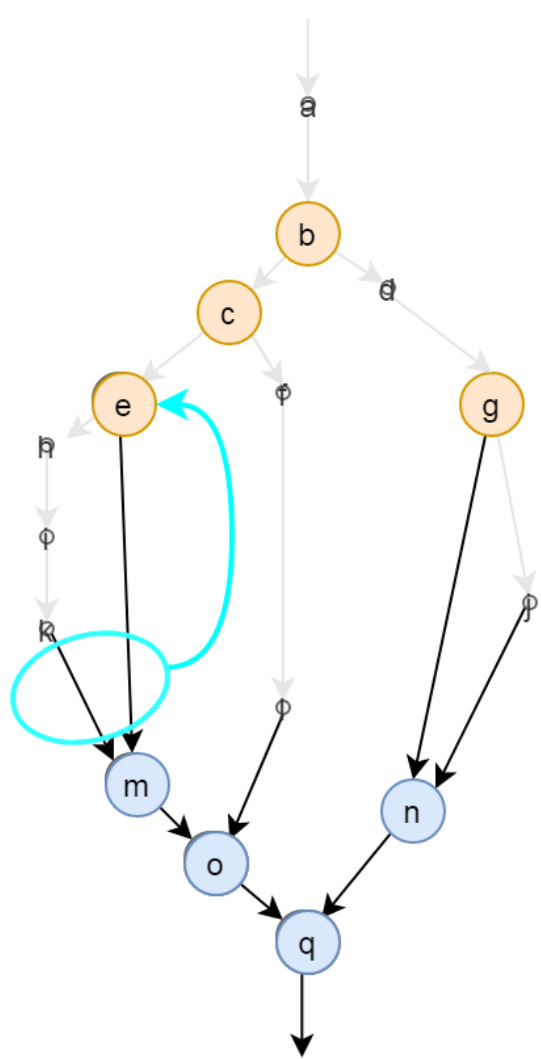
Branch/Merge



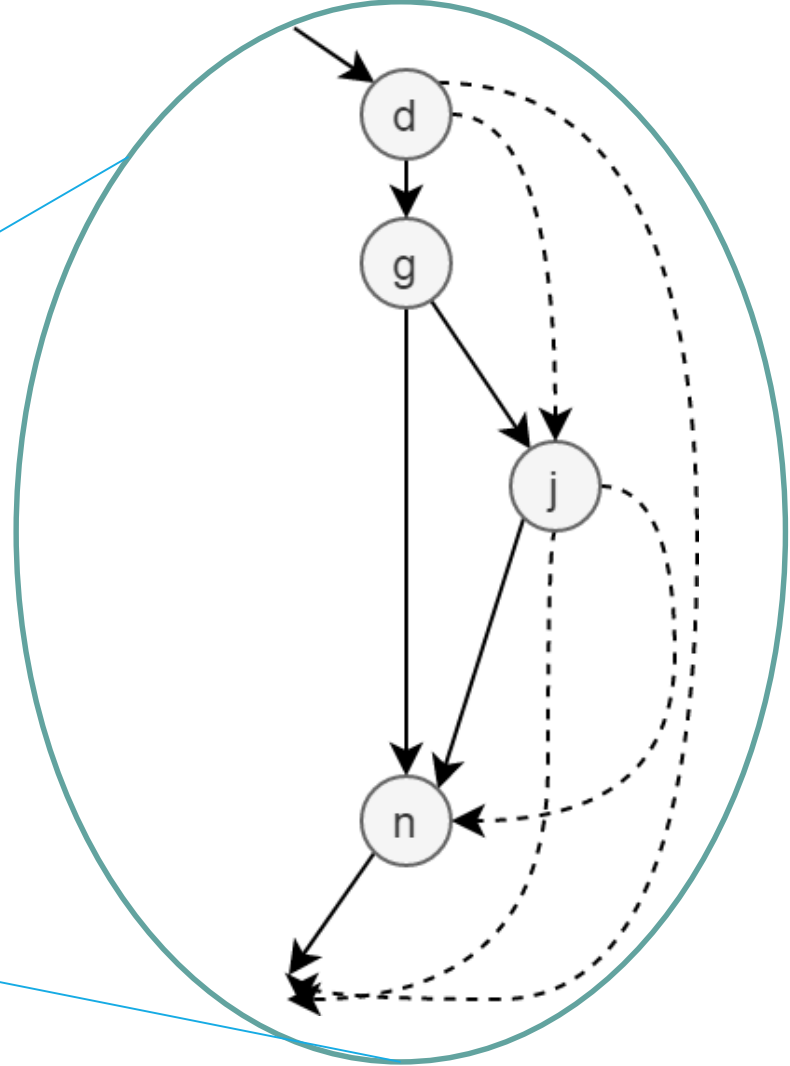
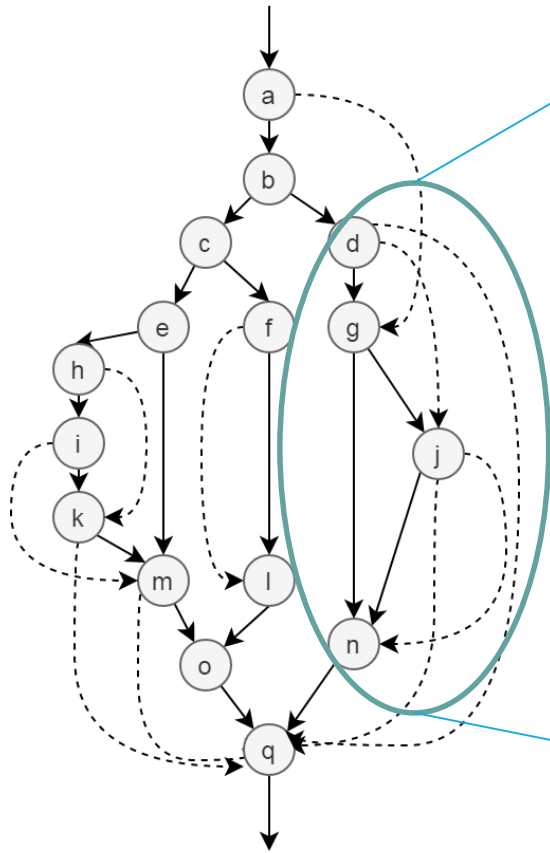
Algorithm



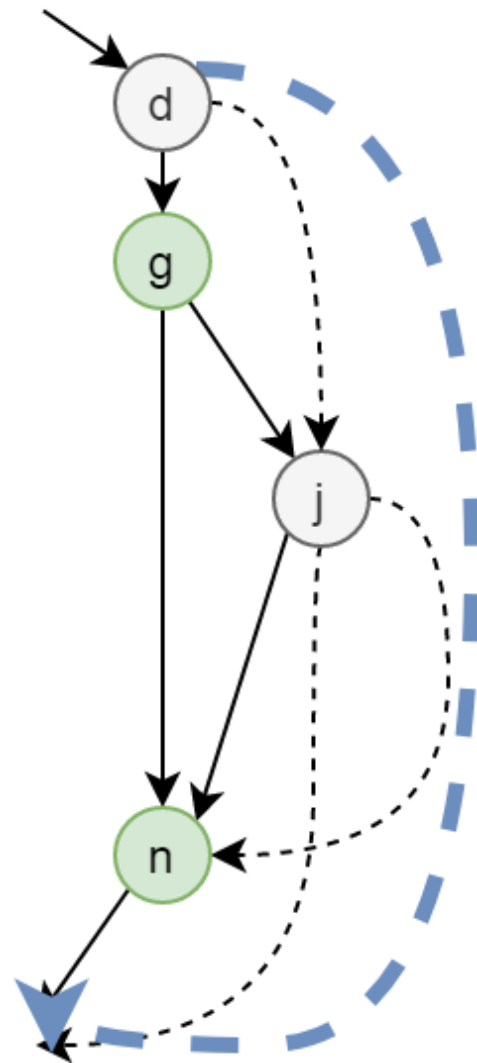
Algorithm



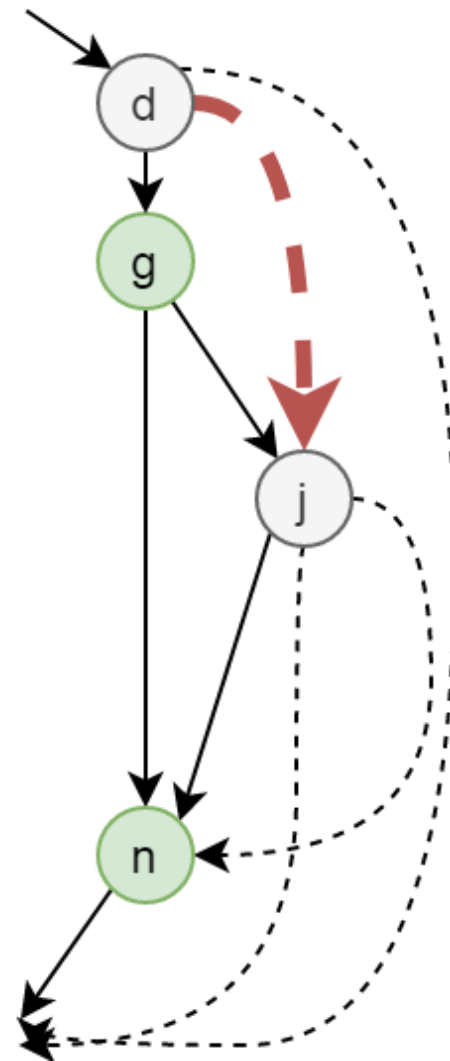
Fence Elimination



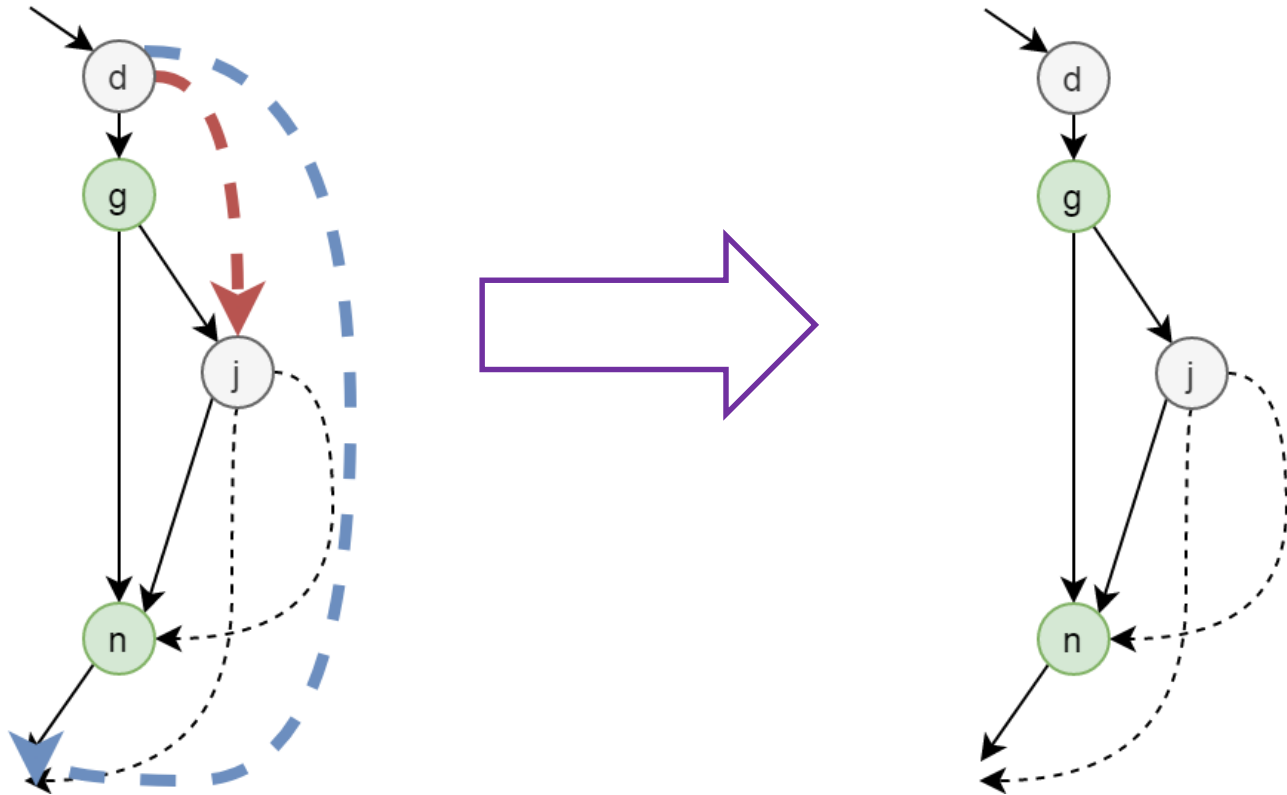
Passing Constraints



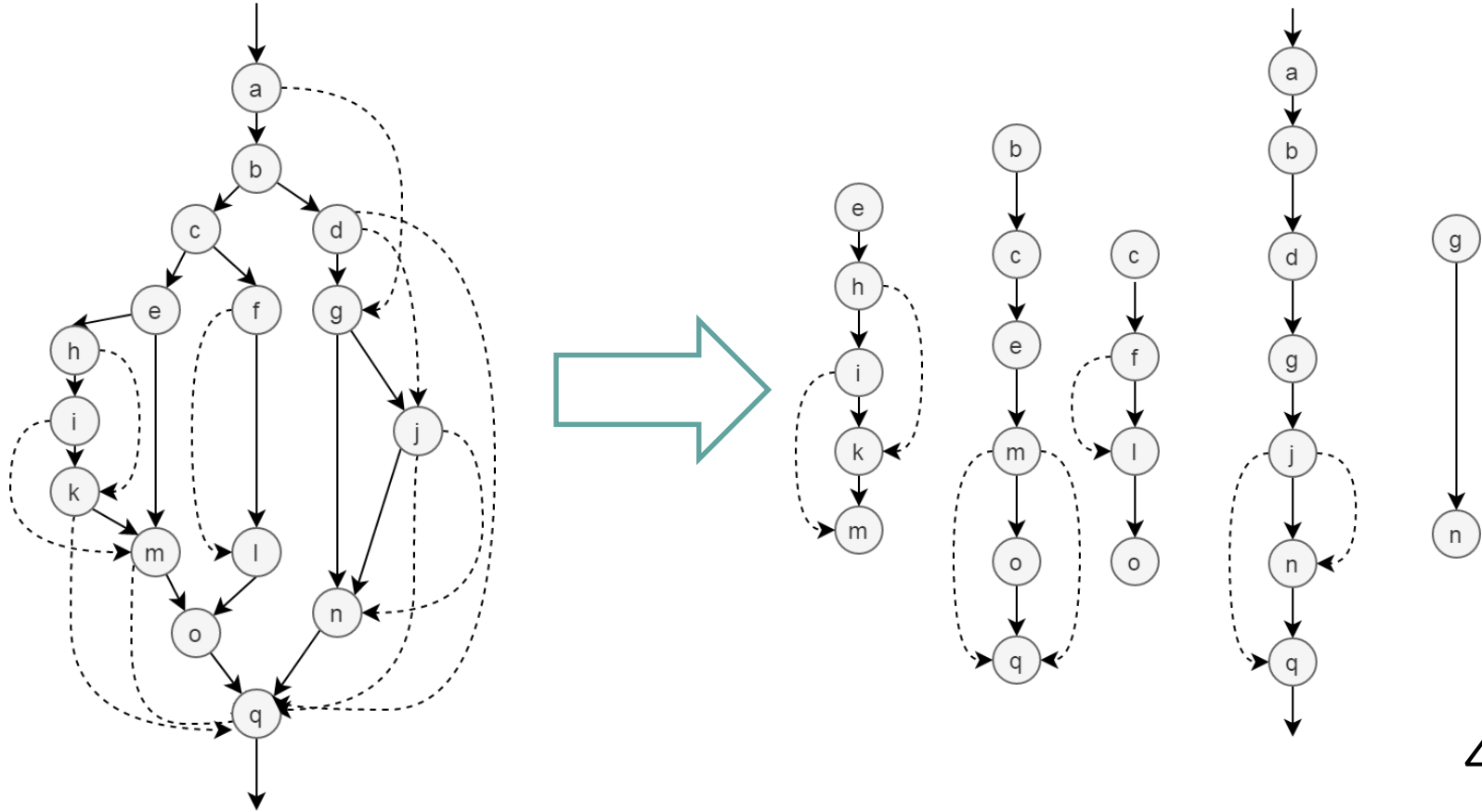
Spanning Constraints



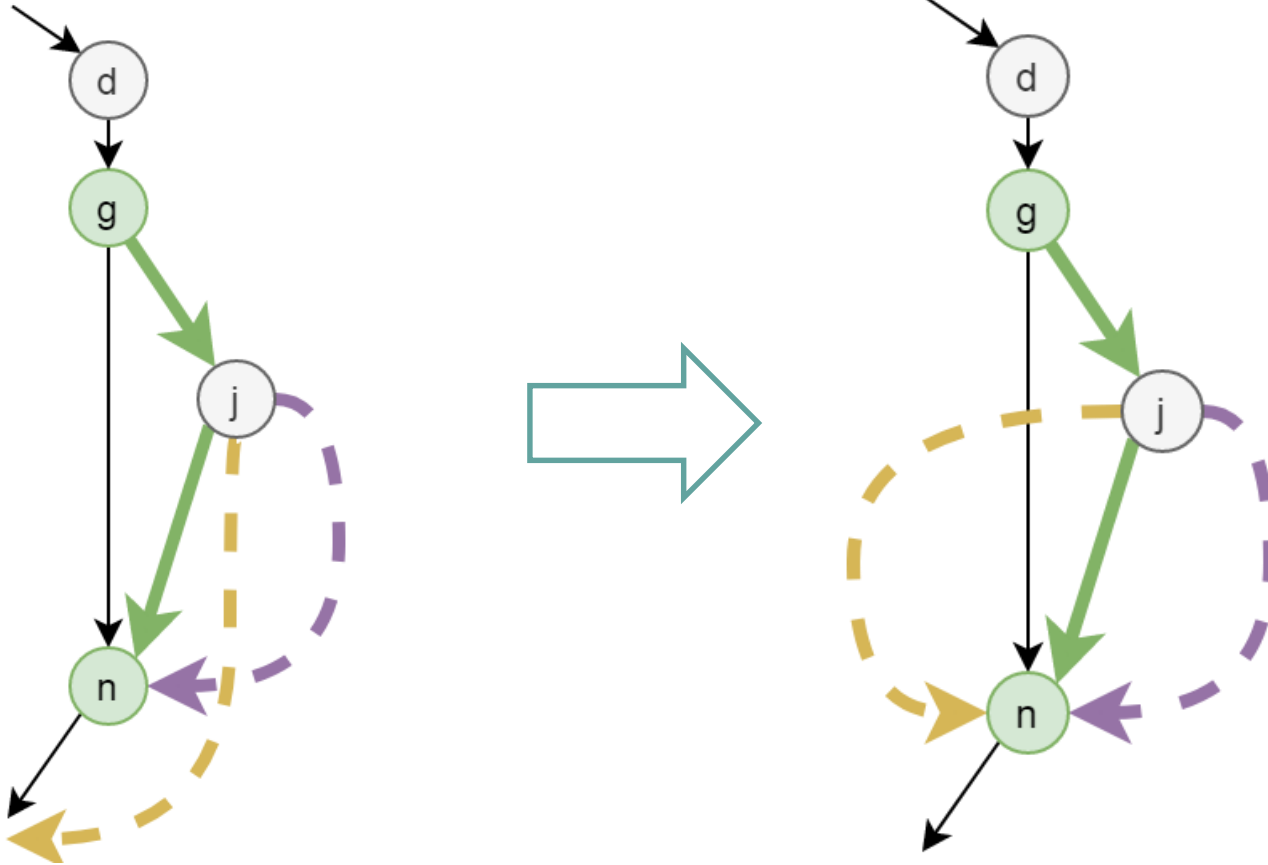
Control Dependency Preservation



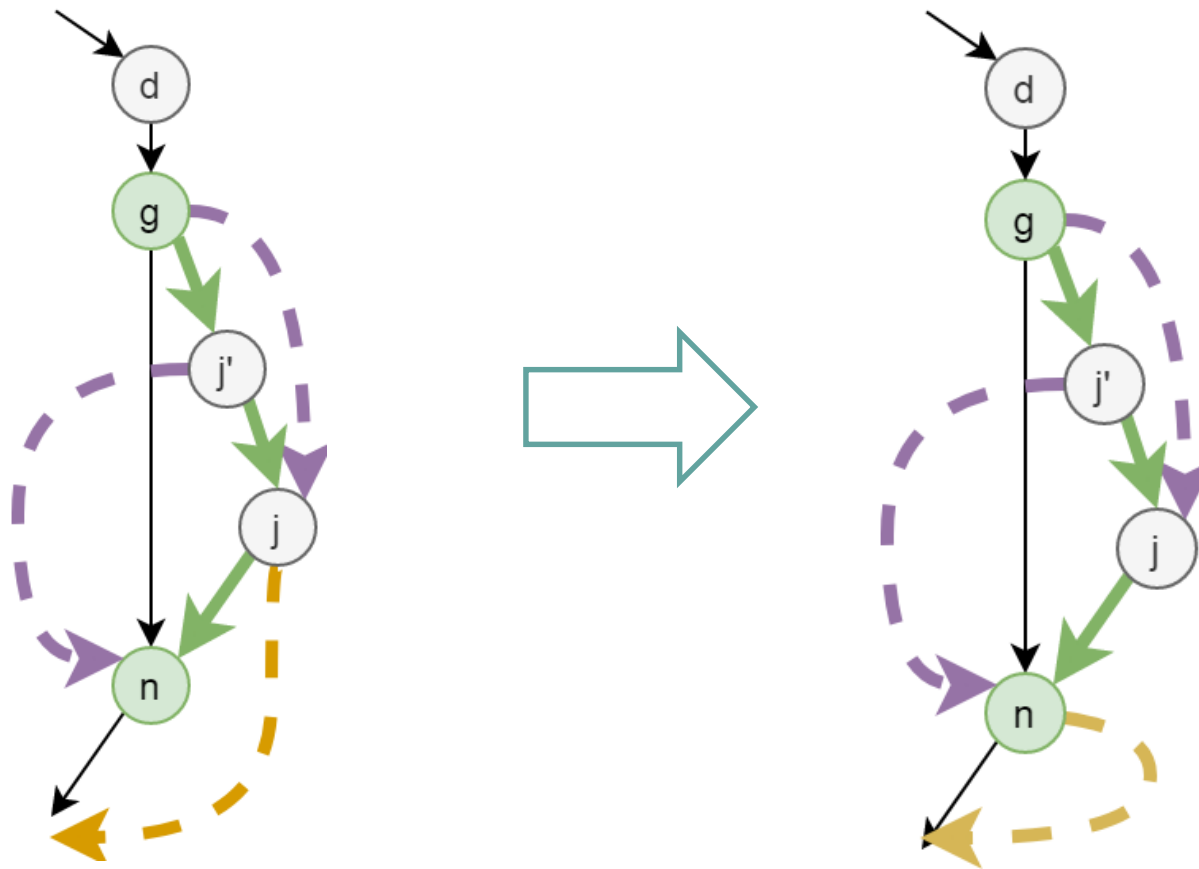
Decomposing into simple paths



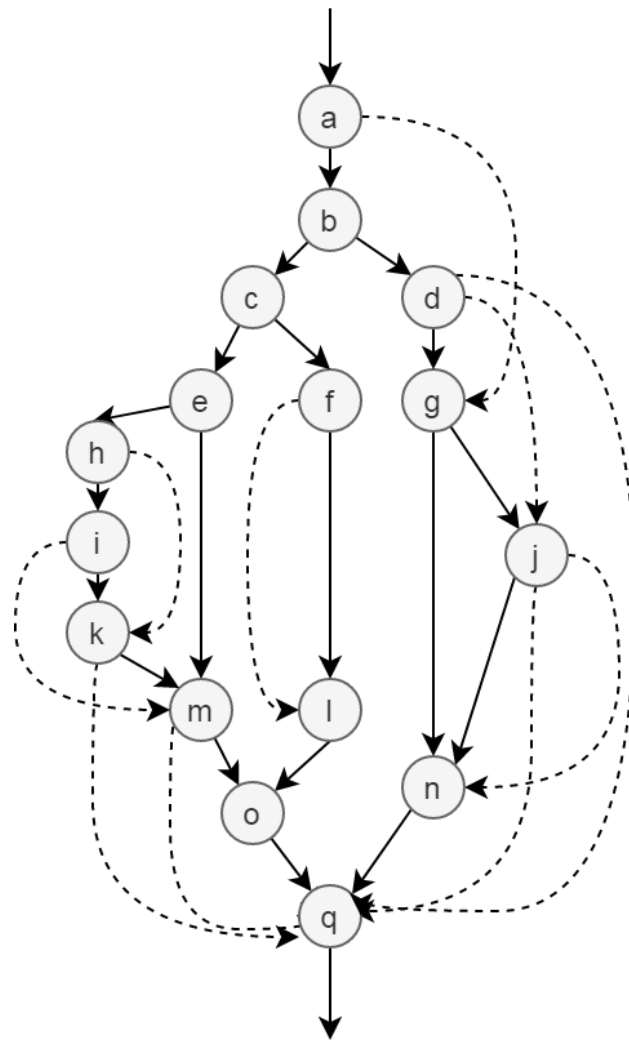
Absorption in a Path



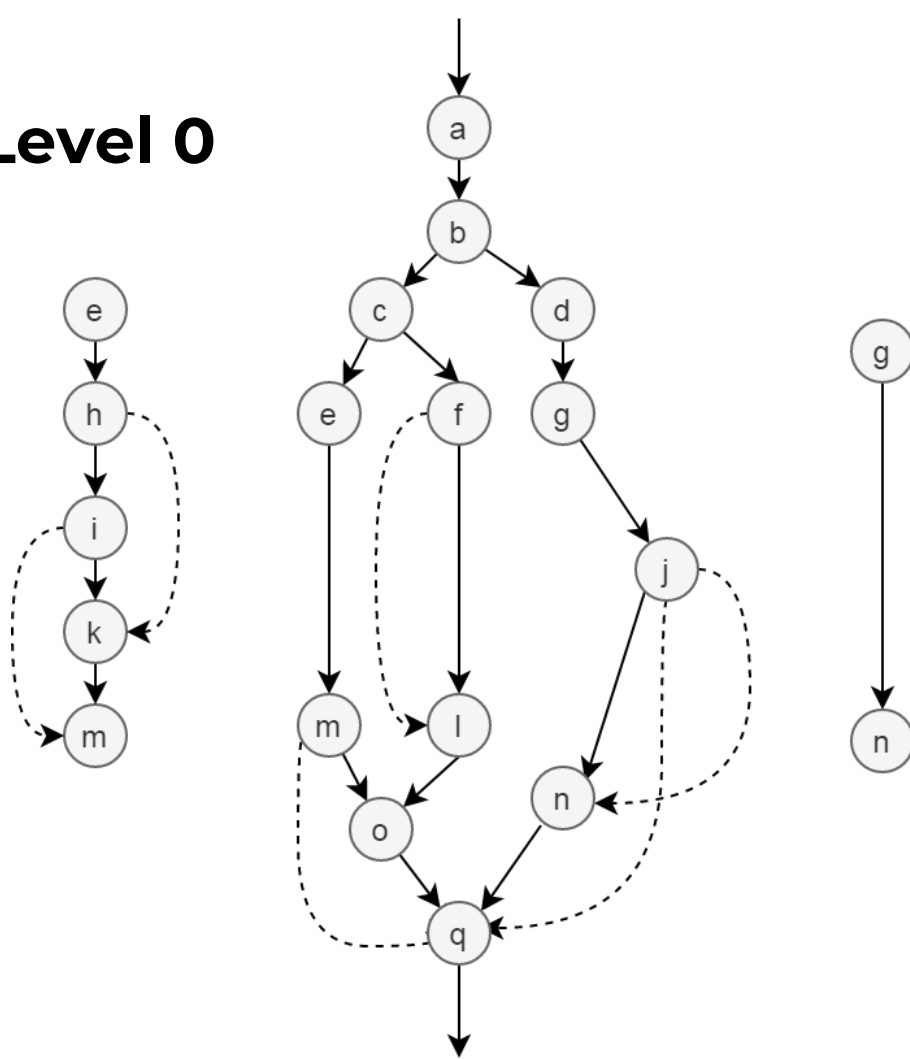
Emission in a Path



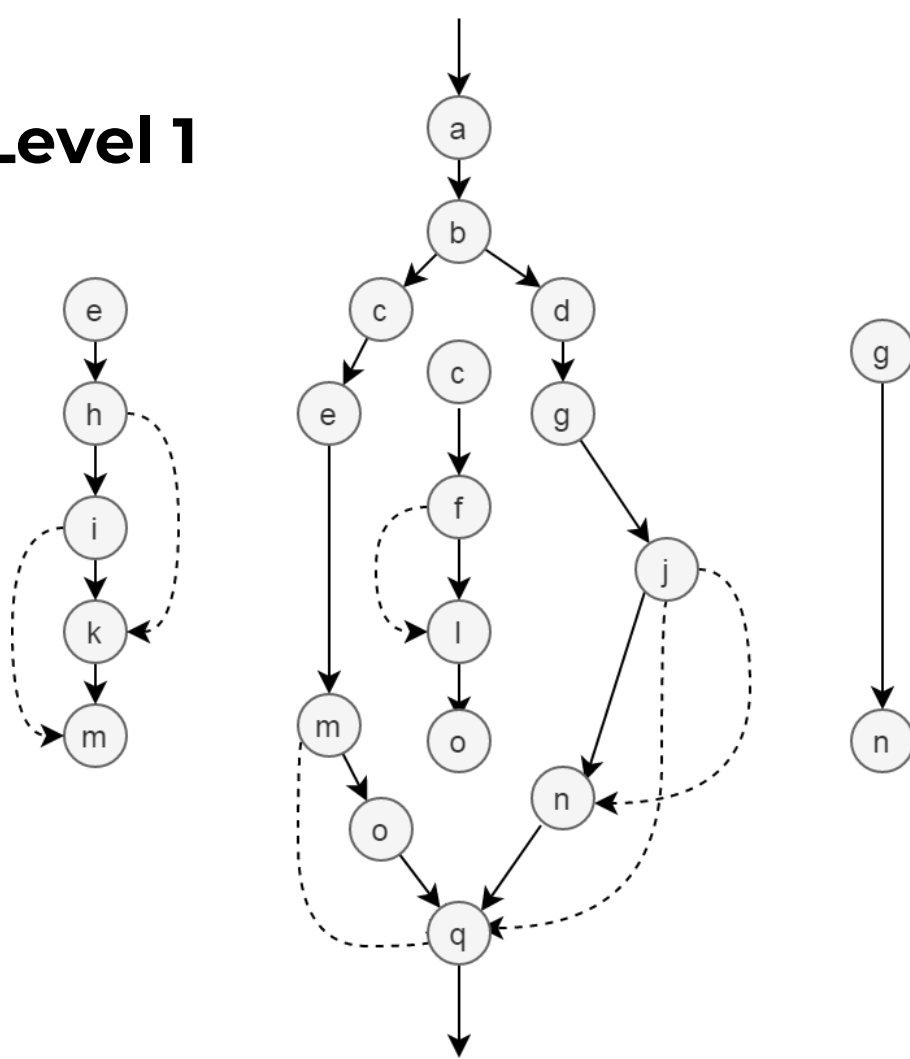
Before



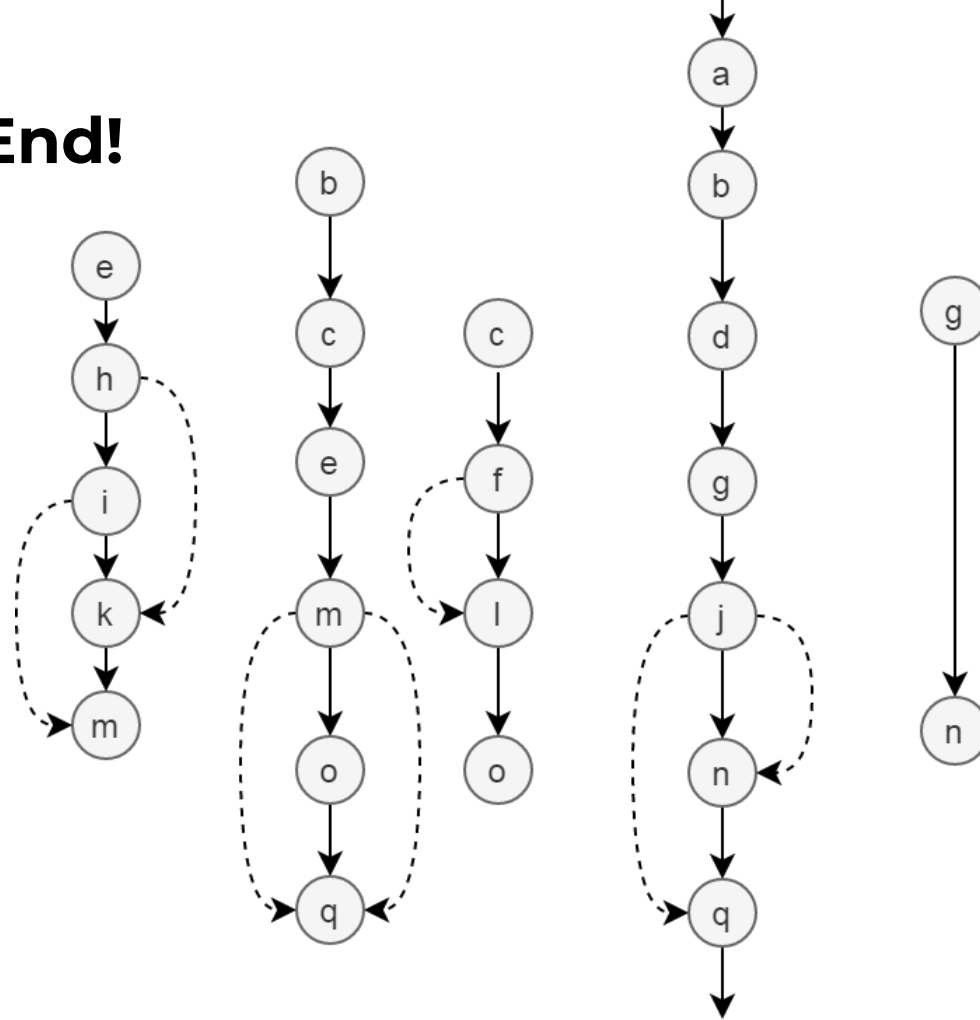
After Level 0



After Level 1



The End!





Complexity

1- $|E| \in O(V)$

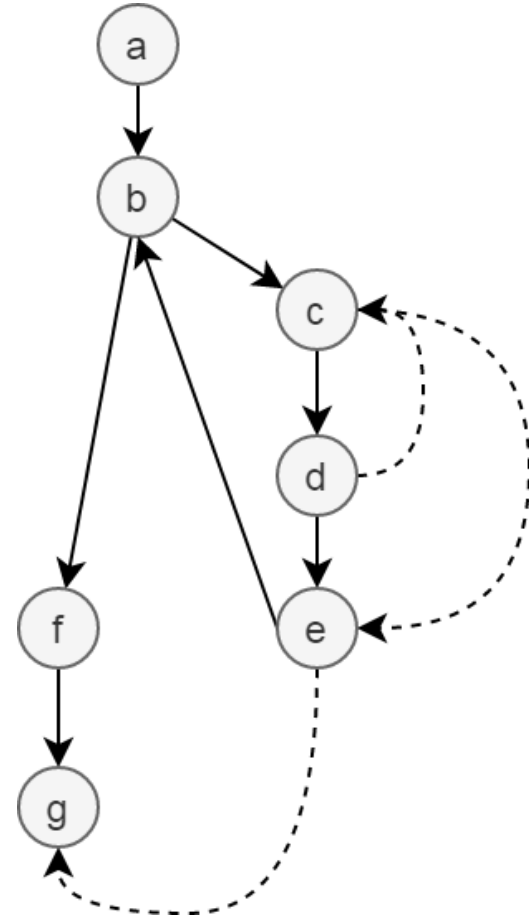
2-Elimination: $O(|C| + |V|)$

3-Finding Diamonds: $O(V \log V)$

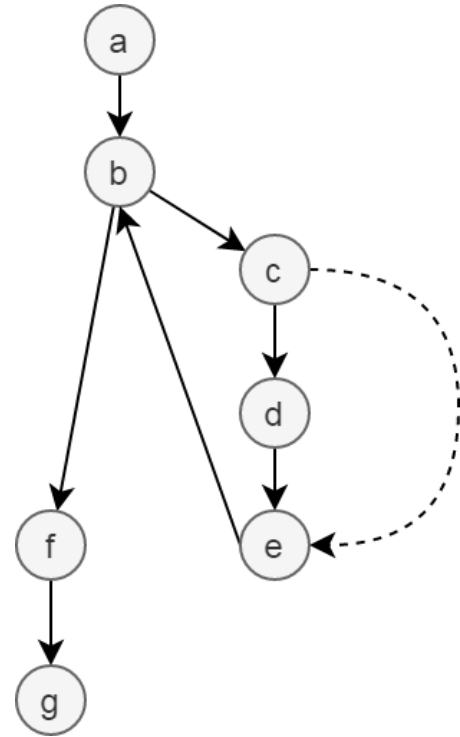
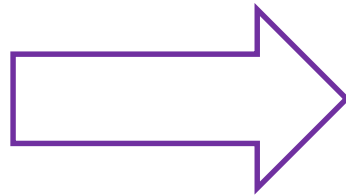
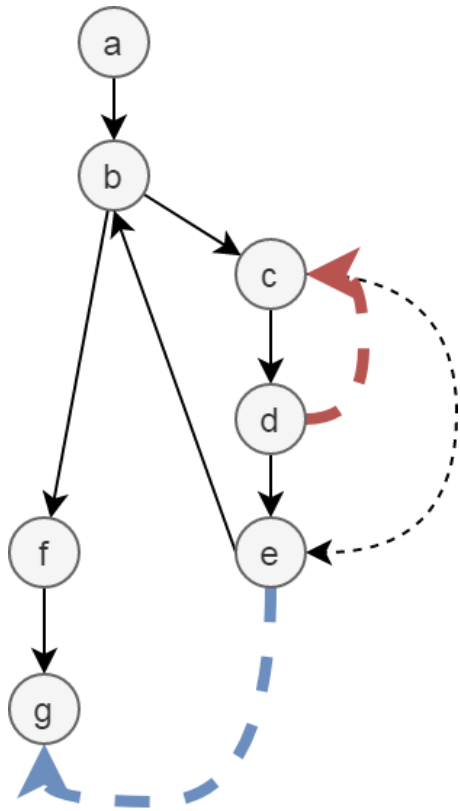
4-Decomposition and Insertion: $\sum_{|p|} |C_{p_i}| \log C_{p_i} + |V_{p_i}|$

\Rightarrow Polynomial Time!

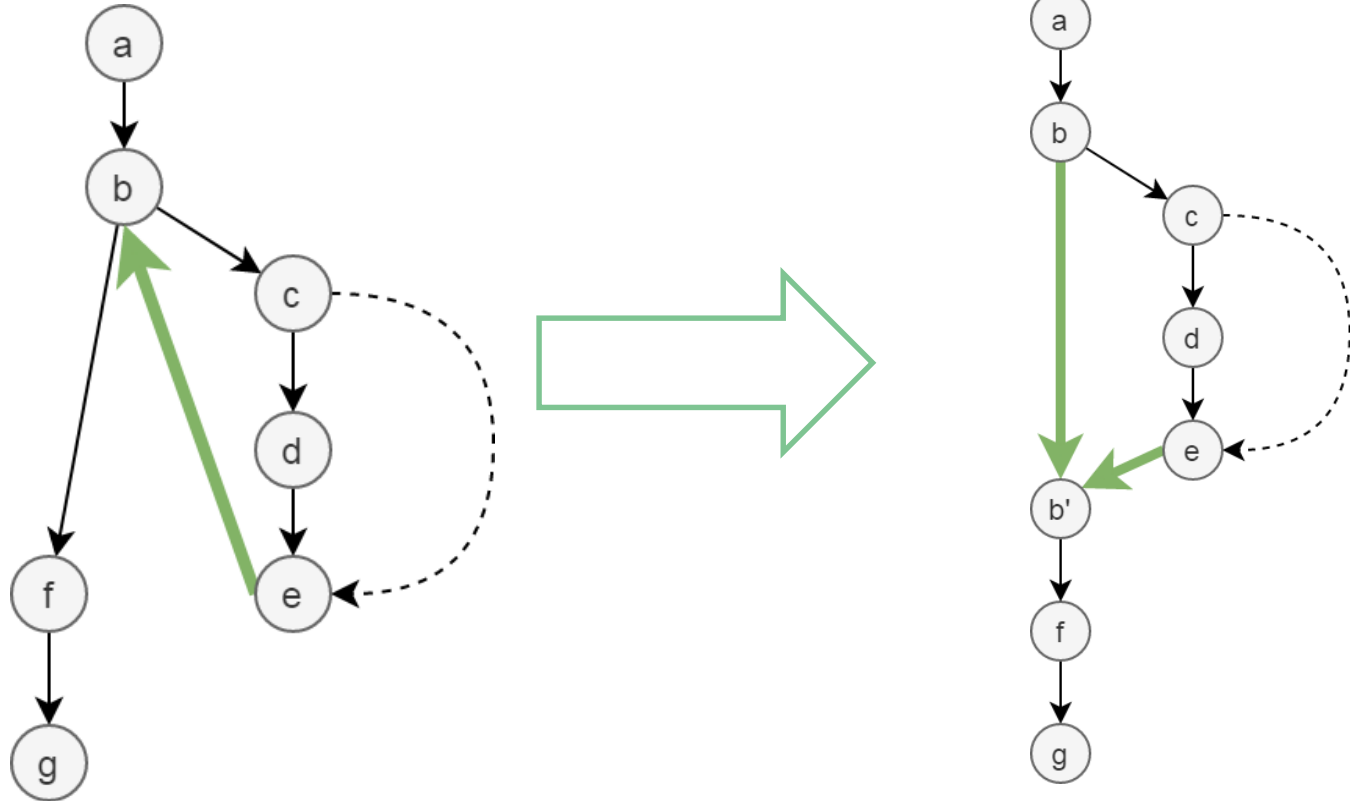
What about loops?



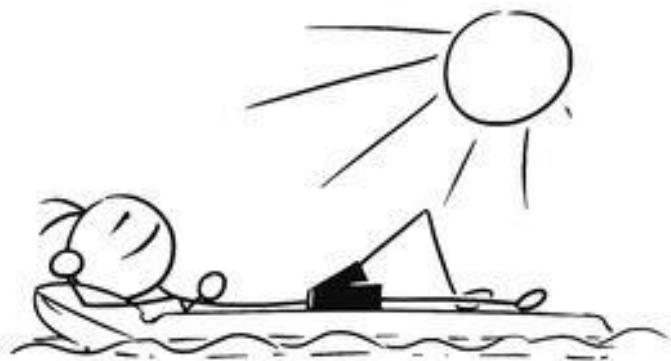
Elimination



Transformation



Thank you





Probabilistic Smart Contracts: Secure Randomness on the Blockchain
[ICBC 2019]

Propose a novel game-theoretic approach for generating provably unmanipulatable pseudorandom numbers on the blockchain.

Hybrid Mining: Exploiting Blockchain's Computational Power for
Distributed Problem Solving [SAC 2019]

A new mining protocol that combines solving real-world useful problems with Hashcash.