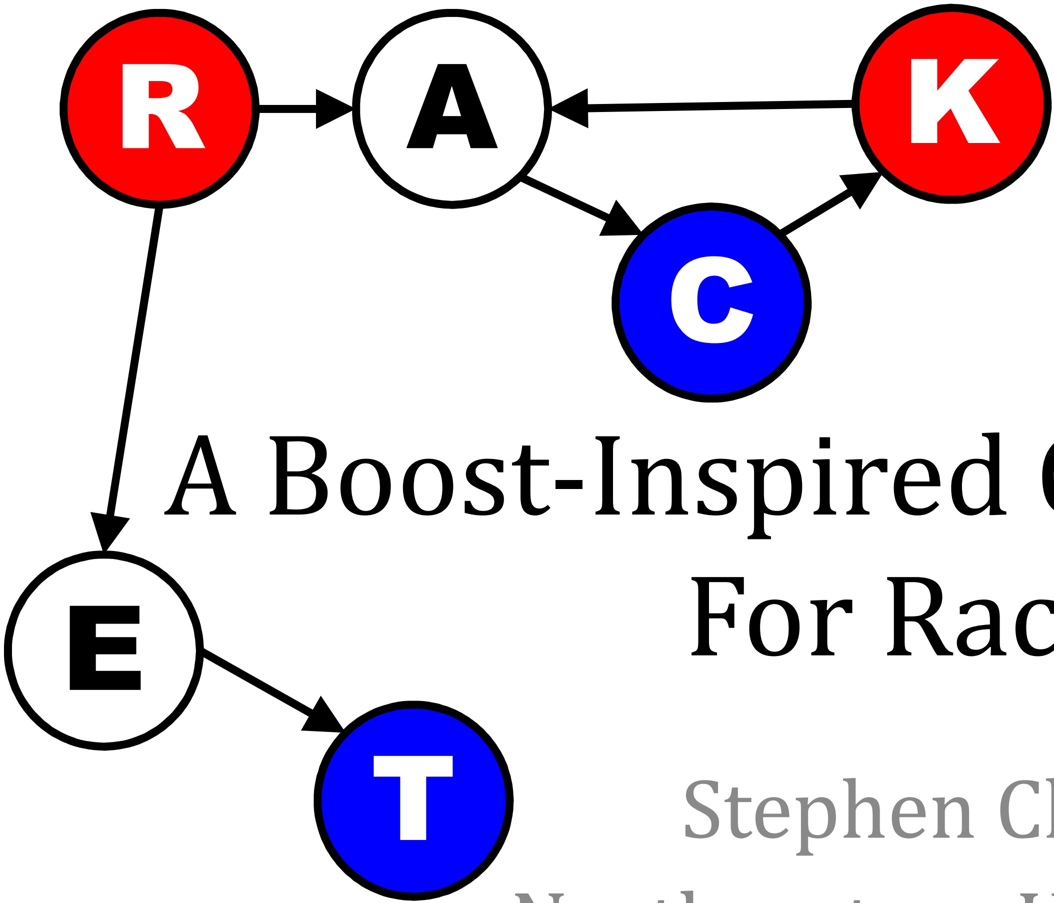
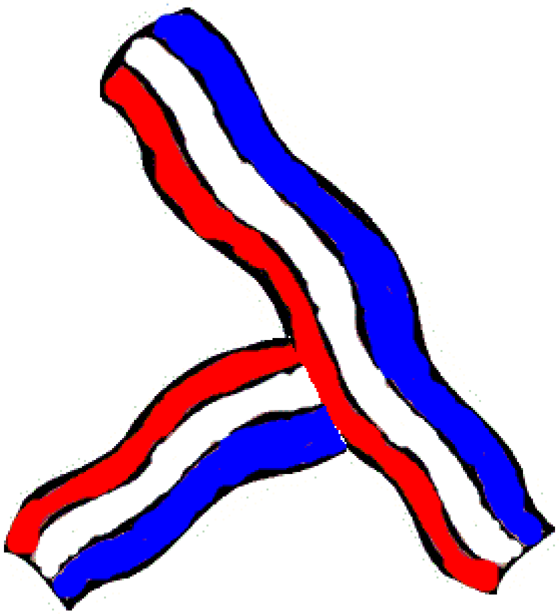


A Boost-Inspired Graph Library For Racket

Stephen Chang
Northeastern University

(fourth RacketCon)
September 20, 2014

Demo: `plt-bacon-erdos`



`http://pasterack.org/bacon`

plt-bacon-erdos

```
(     ...
 ; Scrape data:
 (define PLT-PUBS-URL (string->url "http://www.ccs.neu.edu/racket/pubs/"))
 (define neu-pubs-port (get-pure-port PLT-PUBS-URL))
 (define authors+title      ...  neu-pubs-port      ...)

 ; Populate graph and edge property:
 (define PLT-GRAPH (unweighted-graph/undirected null))
 (define-edge-property PLT-GRAPH papers)
 (for ([as+t authors+title])
   (define authors (cdr (reverse as+t)))
   (define title (car (reverse as+t)))
   (for* ([auth1 authors]
          [auth2 authors]
          #:unless (string=? auth1 auth2))
     (define papers-curr (papers auth1 auth2 #:default null))
     (add-edge! PLT-GRAPH auth1 auth2)
     (papers-set! auth1 auth2 (cons title papers-curr))))

 ; Compute:
 (define (plt-bacon auth erdos bacon)
   (define erdos-path (fewest-vertices-path PLT-GRAPH auth erdos))
   (define bacon-path (fewest-vertices-path PLT-GRAPH auth bacon))
        ...))
```

On Existing Graph Libraries

"none of the libraries applied the principles of generic programming and were far more rigid than necessary"

--- Lee, Siek, Lumsdaine, OOPSLA 1999,
The Generic Graph Component Library

Genericity Goals

- Define new graph representations
  that work with existing algorithm impls
- Attach graph properties without hardcoding
- Reuse common algorithmic patterns

“we found as many graph representations as graph applications” [LSL99]

`define-generics`

gen:graph

- has-vertex?
- has-edge?
- vertex=?
- add-edge!
- add-directed-edge!
- remove-edge!
- remove-directed-edge!
- graph-copy

- add-vertex!
- remove-vertex!
- rename-vertex!
- in-vertices
- in-neighbors
- in-edges
- edge-weight
- transpose

Algorithms

- BFS
- DFS
- Bellman-Ford
- Dijkstra
- Kruskal
- Prim
- Floyd-Warshall
- Johnson
- max flow
- bipartite-matching
- coloring

Genericity Goals

- Define new graph representations that work with existing algorithm impls
- Attach graph properties without hardcoding
- Reuse common algorithmic patterns

A Standard Textbook Algorithm

“During depth-first search, vertices are colored during the search to indicate their state:

- Each vertex is initially white,
- is grayed when it is discovered,
- and is blackened when finished …”

In Racket

```
(define (dfs G)
  (define-vertex-property
    G color #:init WHITE)
  (for ([v (in-vertices G)]
        #:when (white? (color v)))
    (color-set! v GRAY)
    (for ([u (in-neighbors G v)])
        ...)))
```

Genericity Goals

- Define new graph representations
  that work with existing algorithm impls
- Attach graph properties without hardcoding
- Reuse common algorithmic patterns

More Textbook Algorithms

BFS

- Add start vertex to FIFO queue
- For v in queue:
  - For neighbors of v:
    - Add to queue if not seen before

Dijkstra

- Add start vertex to heap
- For v in heap:
  - For neighbors of v:
    - Add to heap if it improves currently known shortest path
    - Update current known shortest path

Prim

- Add start vertex to heap
- For v in heap:
  - For neighbors of v:
    - Add to heap if it improves currently known MST
    - Update currently known MST

More Textbook Algorithms

BFS

- Add start vertex to ***generic*** **queue**
- For v in **queue**:
  - For neighbors of v:
    - **visit v**

Dijkstra

- Add start vertex to ***generic*** **queue**
- For v in **queue**:
  - For neighbors of v:
    - **visit v**

Prim

- Add start vertex to ***generic*** **queue**
- For v in **queue**:
  - For neighbors of v:
    - **visit v**

Generalized BFS

```
(define (bfs/generalized
         G v-start Q
         init
         enqueue?
         on-enqueue)
  (enqueue Q v-start) ; generic queue
  (init)

  (for ([v (in-queue Q)])
    (for ([neigh (in-neighbors G v)]
          #:when (enqueue? v neigh))
      (on-enqueue v neigh)
      (enqueue Q neigh))))
```

events

BFS-based Dijkstra

```
(define (dijkstra G v-start)
  ; dist[v] is current known distance from v-start to v
  (define-vertex-property G dist #:init +inf.0)
  ; pred[v] is v's predecessor in the shortest path
  (define-vertex-property G pred #:init #f)

  (bfs/generalized
   G v-start
   (mk-priority (λ (u v) (< (dist u) (dist v)))) ; heap
   (λ _ (dist-set! v-start 0)) ; init
   (λ (v neigh) ; enqueue?
     (< (+ (dist v) (wgt v neigh))
        (dist neigh)))
   (λ (v neigh) ; on-enqueue
     (dist-set! neigh (+ (dist v) (wgt v neigh)))
     (pred-set! neigh v)))

  (pred->hash))
```

New Binding Forms

```
(do-bfs ; Dijkstra
 G v-start
 #:init-queue
  (mk-priority (λ (u v) (< (dist u) (dist v))))
 #:init (dist-set! v-start 0)
 #:enqueue? (from to)
   (< (+ (dist from) (wgt from to)) (dist to))
 #:on-enqueue (from to)
   (dist-set! to (+ (dist from) (wgt from to)))
   (pred-set! to from))
```

syntax-parse Makes it Easy

```
(define-syntax (do-bfs stx)
  (syntax-parse stx
    [(_ G start-v
     (~or
      (~optional (~seq #:init-queue Q:expr))
      (~optional (~seq #:init i:expr))
      (~optional (~seq #:enqueue? (e?-from:id e?-to:id) e?))
      (~optional (~seq #:on-enqueue (e-from:id e-to:id) e)))
     ...)
     #'(bfs/generalized
             ...
        #:on-enqueue (λ (e-from e-to) e)
             ...)]))
```

Create Special Identifiers

```
(do-bfs ; Dijkstra
 G v-start
 #:init-queue:
   (mk-priority (λ (u v) (< (dist u) (dist v))))
 #:init: (dist-set! v-start 0)
 #:enqueue?:
   (< (+ (dist $from) (wgt $from $to)) (dist $to))
 #:on-enqueue:
   (dist-set! $to (+ (dist $from) (wgt $to $from)))
   (pred-set! $to $from))
```

syntax-parameterize

```
(define-syntax (do-bfs stx)
  (syntax-parse stx
    [(_ G start-v
     (~or
      (~optional (~seq #:init-queue Q:expr))
      (~optional (~seq #:init i:expr))
      (~optional (~seq #:enqueue? (e?-from:id e?-v:id) e?))
      (~optional (~seq #:on-enqueue (e-from:id e-v:id) e)))
     ...)
     #'(bfs/generalized
             ...
        #:on-enqueue
        (λ (from to)
          (syntax-parameterize
           ([$from (syntax-id-rules () [_ from])]
            [$to (syntax-id-rules () [_ to])])
           e))
             ...)]))
```

Thanks!

```
raco pkg install graph
https://github.com/stchang/graph
```