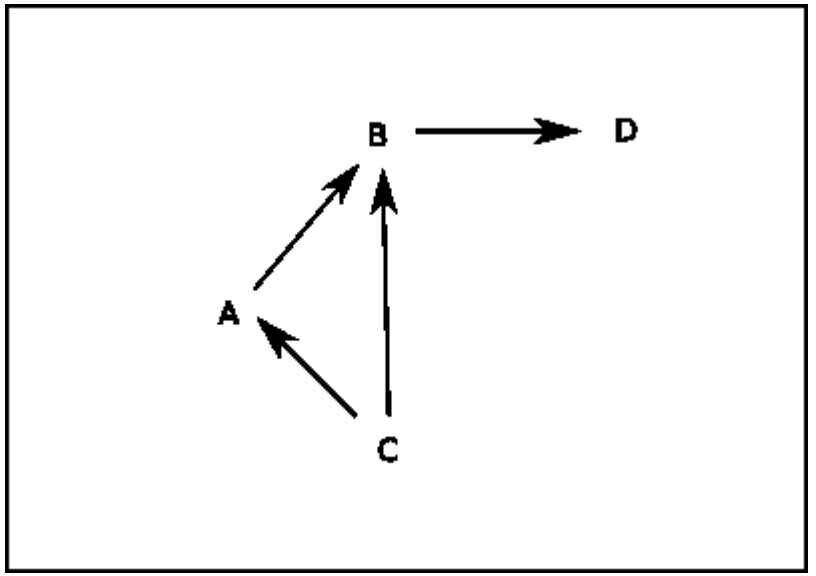
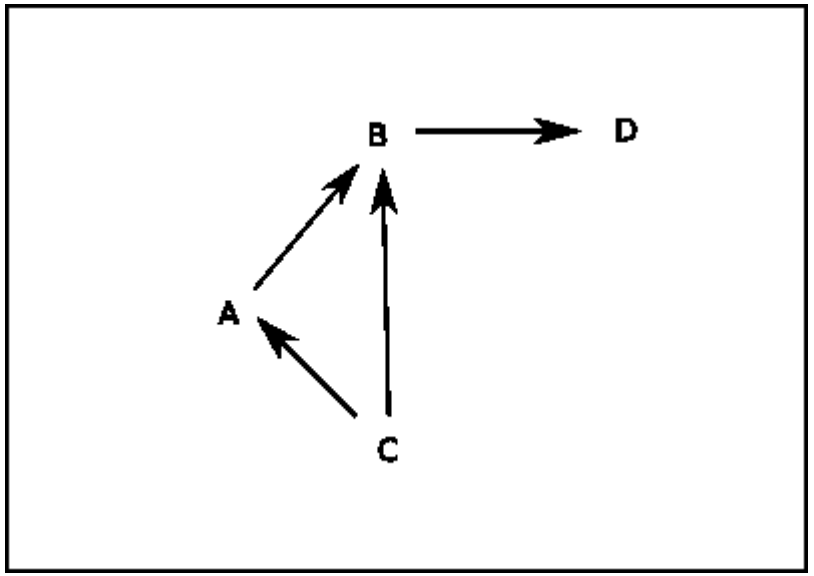
Encontrando rotas em Grafos

Grafo direcionados



- Interpretado como Mapa
- Os nodos podem ser vistos como locais.
- E os arcos como caminhos.

Representação



- Listas de adjacências (vizinhos)

```
(define Mapa
(list
  (list 'A (list 'B))
  (list 'B (list 'D))
  (list 'C (list 'A 'B))
  (list 'D (list ))))
```

Qual o contrato do Mapa?

Qual o contrato do Mapa?

```
;; nó é um simbolo


;; grafo é uma lista de
;;      (list n v)
;; onde n é um nó e
;;          v é uma lista de nós
```

Para percorrer um passo

```
;; vizinhos:nó grafo -> lista de nós
;; retorna os nodos vizinhos de n no grafo G
(define (vizinhos n G)
  (cond [(empty? G)
         (error 'vizinhos "nodo não existe no grafo") ]
        [(symbol=? n (first (first G))) (second (first G))]
        [else (vizinhos n (rest G))]))
```

Para criar um caminho no mapa:

;; find-route : nó nó grafo -> (listof nó)

;; para encontrar um caminho da origem até o destino em G

(define (find-route origem destino G) ...)

;; exemplos

(find-route 'B 'D Mapa)

;; deve retornar (list 'B 'D)

Exemplos (cont.)

```
(find-route 'A 'D Mapa)
;; deve retornar (list 'A 'B 'D)

(find-route 'A 'C Mapa) ?
```

Revisão contrato

;; find-route : nó nó grafo -> (listof nó) or false

;; para encontrar um caminho da origem até o destino em G

;; se não há nenhum caminho, então a função retorna falso

(define (find-route origem destino G) ...)

Template

```
(define (find-route origem destino G)
  (cond
    [(symbol=? origem destino) (list destino)]
    [else ... (find-route/list (vizinhos origem G)
   destino G) ...]))
```

Refinamento

```
(define (find-route origem destino G)
  (cond
    [(symbol=? origem destino) (list destino)]
    [else (local ((define possible-route
    (find-route/list (neighbors origem G)
                                    destino G)))
    (cond
      [(boolean? possible-route) ...]
      [else ;; (cons? possible-route)
...]))]))
```

Cabeçalho

```
;; find-route/list : (listof nó) nó grafo  ->  (listof nó) or false
;; para criar um caminho de algum nó na ld-origems
;; para o destino
;; se não há nenhum caminho, então a função retorna falso
(define (find-route/list ld-origems destino G) ...)
```

find-route : nó nó grafo -> (lista de nós) or false

```
;; para encontrar um caminho da origem até o destino em G
;; se não há nenhum caminho, então a função retorna falso
(define (find-route origem destino G)
  (cond
    [(symbol=? origem destino) (list destino)]
    [else (local ((define possible-route
    (find-route/list (vizinhos origem G) destino G)))
    (cond
      [(boolean? possible-route) false]
      [else (cons origem possible-route)]))])))
```

find-route/list : (lista de nós) nó grafo -> (lista de nós) or false

```
;; para criar um caminho de algum nó na ld-Os para D
;; se não há nenhum caminho, então a função retorna falso
(define (find-route/list ld-Os D G)
  (cond
    [(empty? ld-Os) false]
    [else (local ((define possible-route (find-route (first ld-Os) D G)))
    (cond
      [(boolean? possible-route) (find-route/list (rest ld-Os) D G)]
      [else possible-route]))]))
```