Trabalho de Grafos
Parte 2

Grafos ponderados

Código

- Feito em Go (golang)
    - Compilada (Execução mais rápida)
    - Tipada (Melhor uso de memória)
- Bibliotecas próprias
    - Otimização de estruturas (“classes”)
    - Sintaxe homogênea

```go
func dijkstraList(adjacency [][]*Neighbor, start uint32, end uint32) []*TreeNode {
    // initialize the list
    list := List{}
    list.vertexes = make([]*Node, len(adjacency)+1) // 0 is not used



    list.Update(start, 0, 0)

    // tree
    tree := make([]*TreeNode, 0, len(adjacency))

    node := list.Pop()
    for node != nil && node.id != end {
        // add node to tree
        tree = append(tree, &TreeNode{node.id, node.father, node.cost})

        // update neighbors
        for _, v := range adjacency[node.id] {
            list.Update(v.vertex_id, node.id, node.cost+v.weight)
        }
        node = list.Pop() // get the node with the lowest cost
    }

    if node != nil { // add the end node
        tree = append(tree, &TreeNode{node.id, node.father, node.cost})
    }

    return tree
}
```

```go
func dijkstraHeap(adjacency [][]*Neighbor, start uint32, end uint32) []*TreeNode {
    // initialize the heap
    heap := Heap{}
    heap.vertexPos = make([]int, len(adjacency))
    for i := range heap.vertexPos { // set all vertexes as unexplored
        heap.vertexPos[i] = -1
    }
    heap.Update(start, 0, 0)

    // tree
    tree := make([]*TreeNode, 0, len(adjacency)+1) // 0 is not used

    node := heap.Pop()
    for node != nil && node.id != end {
        // add node to tree
        tree = append(tree, &TreeNode{node.id, node.father, node.cost})

        // update neighbors
        for _, v := range adjacency[node.id] {
            heap.Update(v.vertex_id, node.id, node.cost+v.weight)
        }
        node = heap.Pop() // get the node with the lowest cost
    }

    if node != nil { // add the end node
        tree = append(tree, &TreeNode{node.id, node.father, node.cost})
    }

    return tree
}
```

Dijkstra Lista

Estruturas:

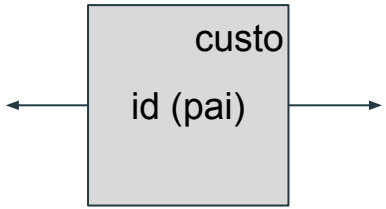
```
13  type List struct {
14      vertexes []*Node
15      head     *Node
16      tail     *Node
17  }
```

vértices []*node
head *node
tail *node

vértices	n*64	bits
head	64	bits
tail	64	bits

(2+n) * 64
≈ 64*n

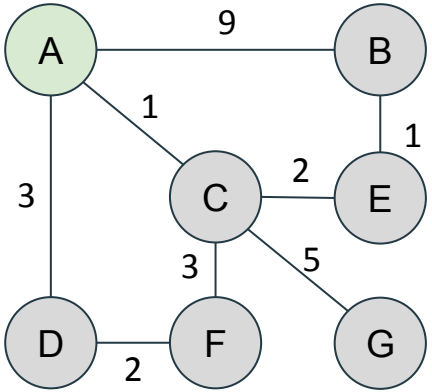
```
5   type Node struct {
6       id     uint32
7       father uint32
8       cost   float64
9       prev   *Node
10      next   *Node
11  }
```



id	32	bits
pai	32	bits
custo	64	bits
ant	64	bits
prox	64	bits

32*2 + 64*3
= 256

Dijkstra Lista



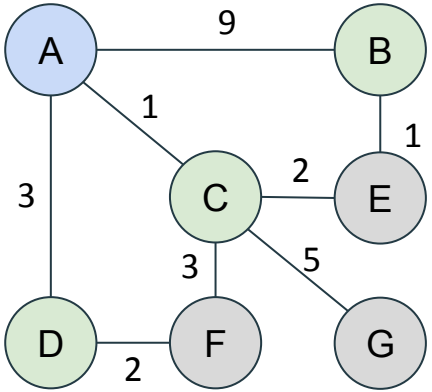
Lista encadeada ordenada dos vértices descobertos:

0
A (-)

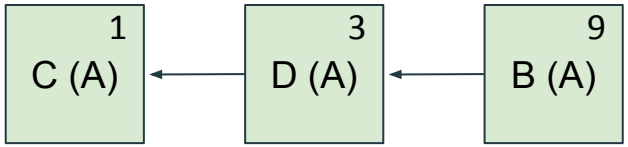
Vértices explorados:

Lista:	A	B	C	D	E	F	G
vértices	[&A, nil, nil, nil, nil, nil, nil]						
head	&A						
tail	&A						

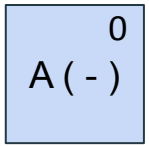
Dijkstra Lista (pop)



Lista encadeada ordenada dos vértices descobertos:

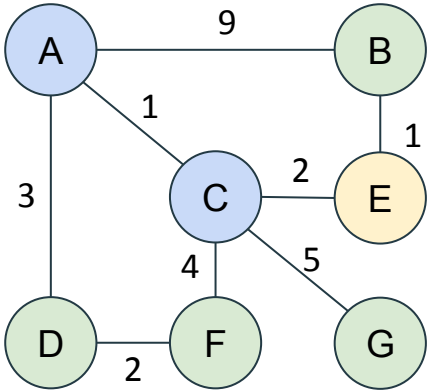


Vértices explorados:

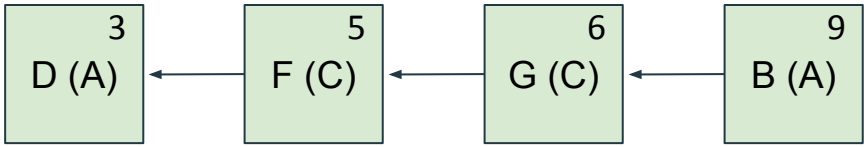


Lista:	A	B	C	D	E	F	G
vértices	[&A,	&B,	&C,	&D,	nil,	nil,	nil]
head	&C						
tail	&B						

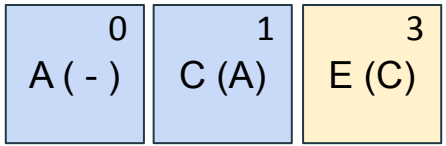
Dijkstra Lista (update)



Lista encadeada ordenada dos vértices descobertos:

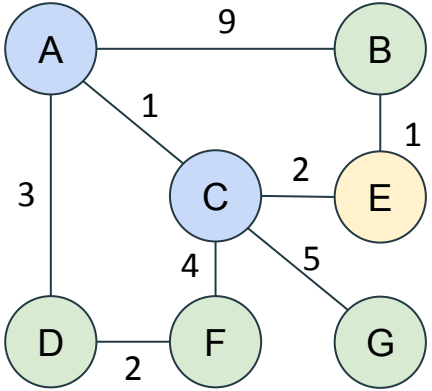


Vértices explorados:

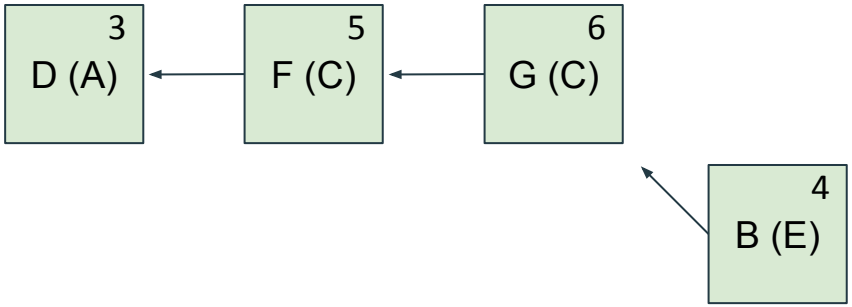


Lista:	A	B	C	D	E	F	G
vértices	[&A,	&B,	&C,	&D,	&E,	&F,	&G]
head	&D						
tail	&B						

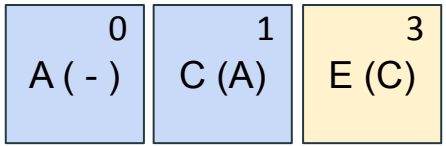
Dijkstra Lista (update)



Lista encadeada ordenada dos vértices descobertos:

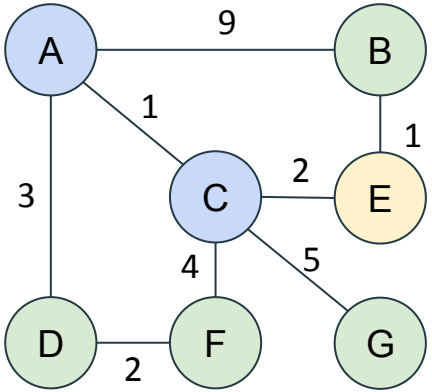


Vértices explorados:

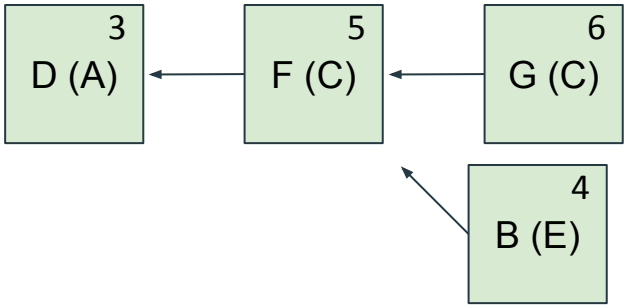


Lista:	A	B	C	D	E	F	G
vértices	[&A,	&B,	&C,	&D,	&E,	&F,	&G]
head	&D						
tail	&G						

Dijkstra Lista (reorder)



Lista encadeada ordenada dos vértices descobertos:

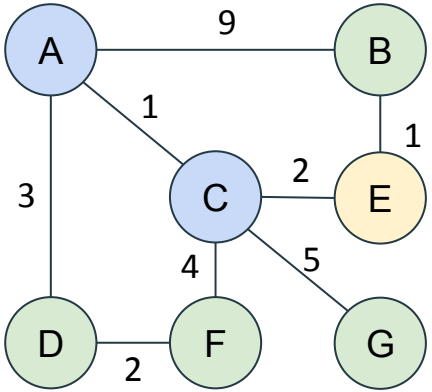


Vértices explorados:

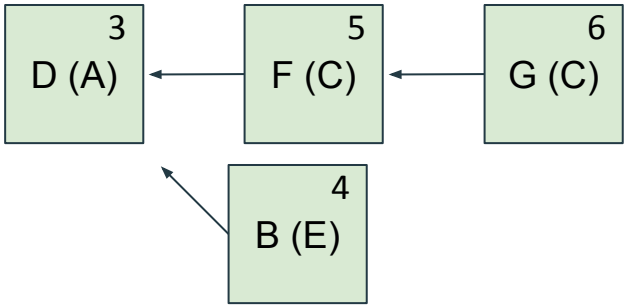
0	1	3
A (-)	C (A)	E (C)

Lista:	A	B	C	D	E	F	G
vértices	[&A, &B, &C, &D, &E, &F, &G]						
head	&D						
tail	&G						

Dijkstra Lista (reorder)



Lista encadeada ordenada dos vértices descobertos:

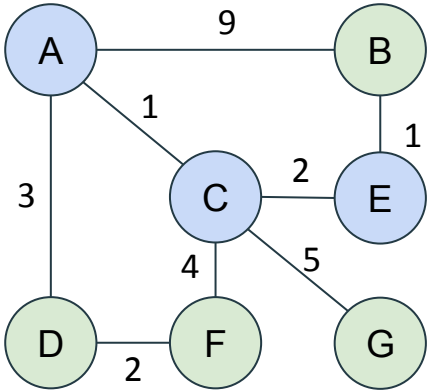


Vértices explorados:

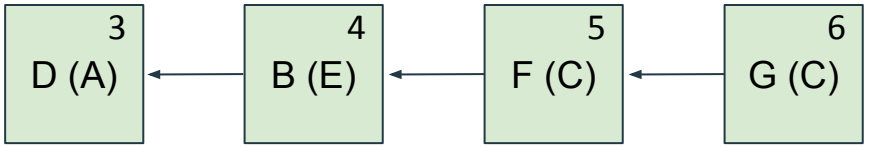
0	1	3
A (-)	C (A)	E (C)

Lista:	A	B	C	D	E	F	G
vértices	[&A, &B, &C, &D, &E, &F, &G]						
head	&D						
tail	&G						

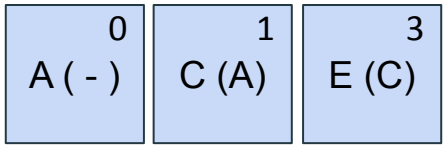
Dijkstra Lista (reorder)



Lista encadeada ordenada dos vértices descobertos:



Vértices explorados:



Lista:	A B C D E F G
vértices	[&A, &B, &C, &D, &E, &F, &G]
head	&D
tail	&G

Dijkstra Heap

Estruturas:

```
9   type Heap struct {
10      vertexPos []int
11      heap      []HeapNode
12  }
```

verticePos []int
heap []nós

verticePos	n*64	bits
heap	n*128	bits

(128 + 64) * n = 192*n

```
3   type HeapNode struct {
4       id     uint32
5       father uint32
6       cost   float64
7   }
8
```

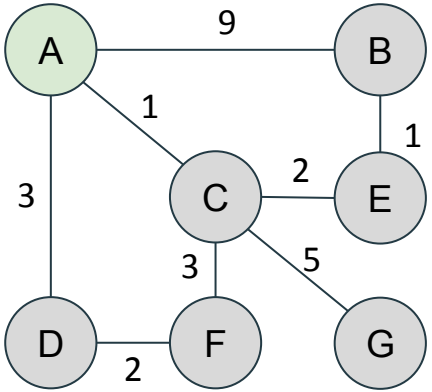
custo

id (pai)

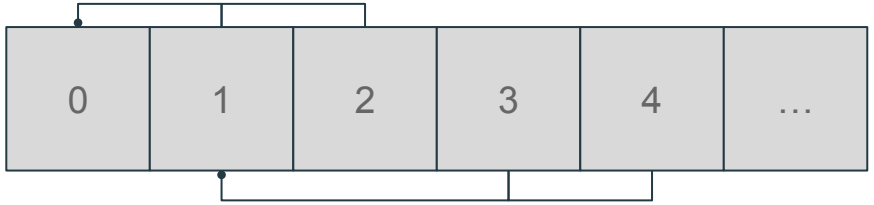
id	32	bits
pai	32	bits
custo	64	bits

(32*2 + 64) = 128

Dijkstra Heap



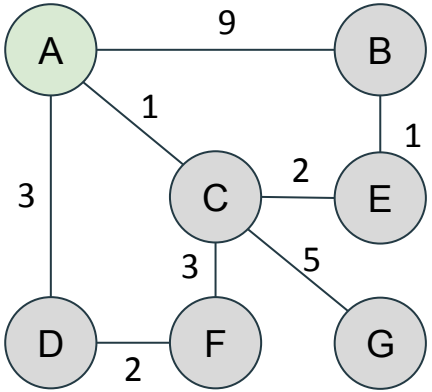
Árvore de Heap:



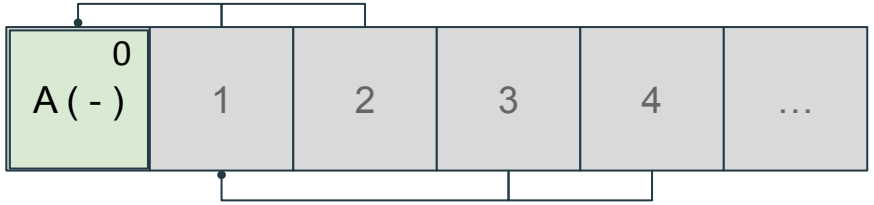
Vértices explorados:

	A	B	C	D	E	F	G
verticePos	-1	-1	-1	-1	-1	-1	-1
heap	[ ]						

Dijkstra Heap



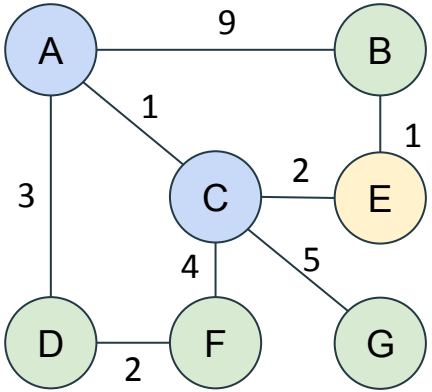
Árvore de Heap:



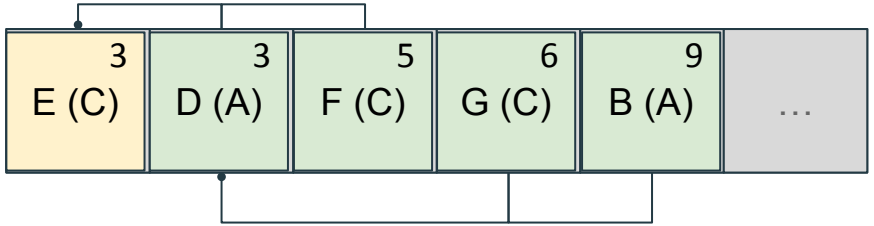
Vértices explorados:

	A	B	C	D	E	F	G
verticePos	[0, -1, -1, -1, -1, -1, -1]						
heap	[ ]						

Dijkstra Heap (pop)



Árvore de Heap:

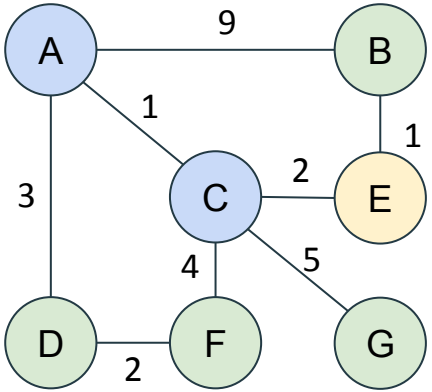


Vértices explorados:

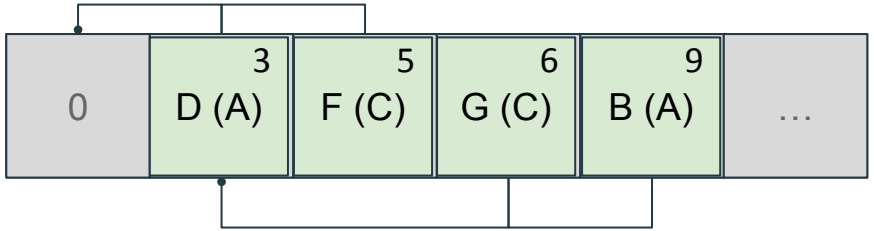
0	1
A (-)	C (A)

	A	B	C	D	E	F	G
verticePos	-2	4	-2	1	0	2	3
heap							

Dijkstra Heap (pop)



Árvore de Heap:

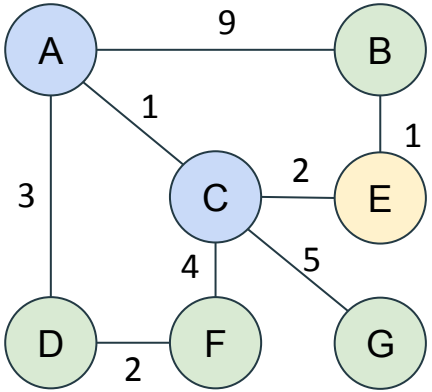


Vértices explorados:

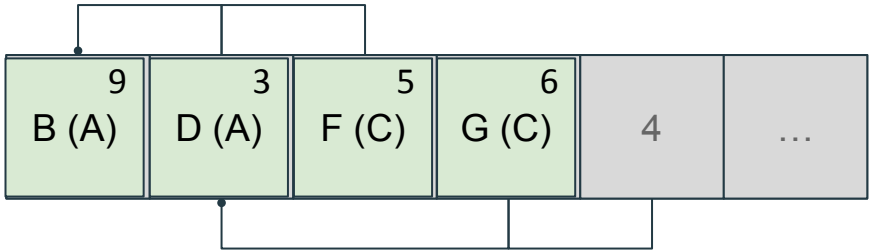
0	1	3
A (-)	C (A)	E (C)

	A	B	C	D	E	F	G
verticePos	-2	4	-2	1	-2	2	3
heap	[ ]						

Dijkstra Heap (bubbleDown)



Árvore de Heap:

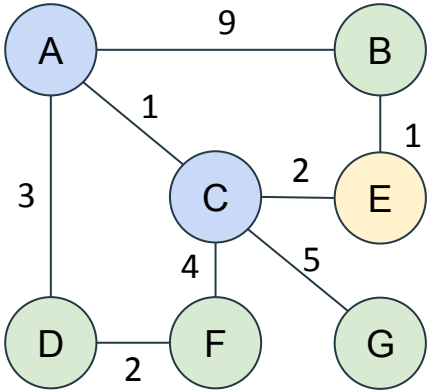


Vértices explorados:

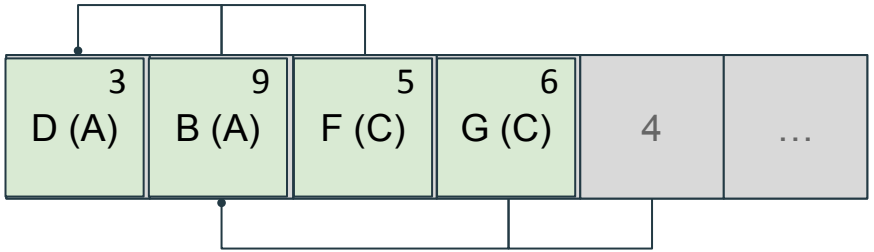
0	1	3
A (-)	C (A)	E (C)

	A	B	C	D	E	F	G
verticePos	-2	0	-2	1	-2	2	3
heap							

Dijkstra Heap (bubbleDown)



Árvore de Heap:

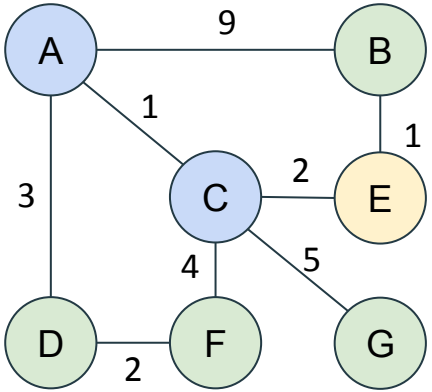


Vértices explorados:

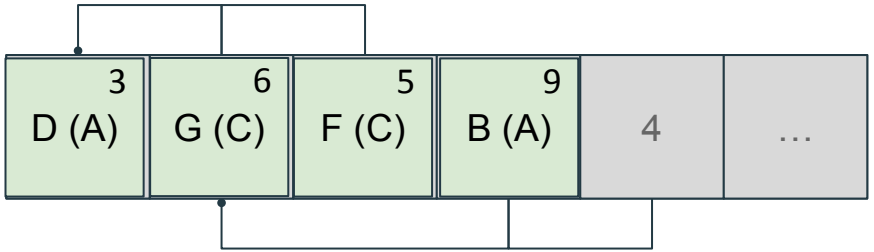
0	1	3
A (-)	C (A)	E (C)

	A	B	C	D	E	F	G
verticePos	-2	2	-2	1	-2	0	3
heap							

Dijkstra Heap (bubbleDown)



Árvore de Heap:

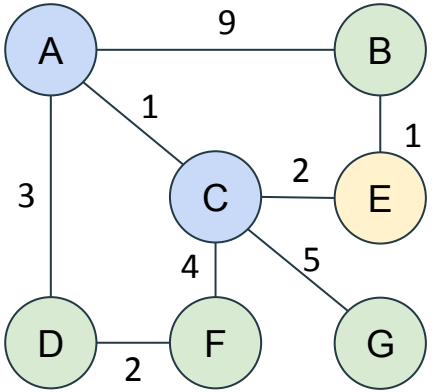


Vértices explorados:

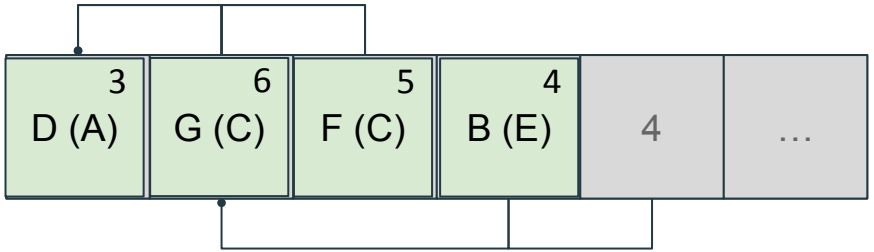
0	1	3
A (-)	C (A)	E (C)

	A	B	C	D	E	F	G
verticePos	-2	3	-2	0	-2	2	1
heap	[ ]						

Dijkstra Heap (update)



Árvore de Heap:

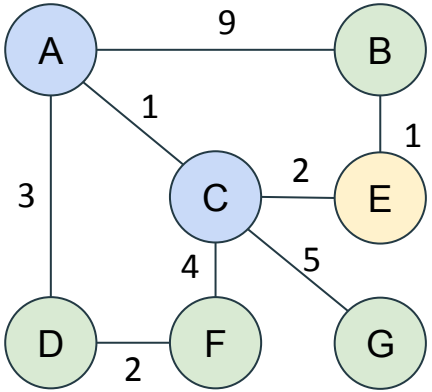


Vértices explorados:

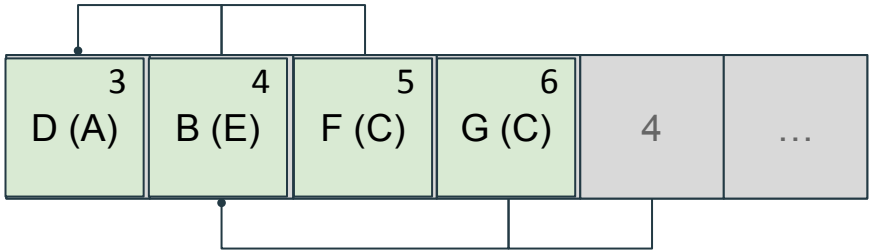
0	1	3
A (-)	C (A)	E (C)

	A	B	C	D	E	F	G
verticePos	-2	3	-2	0	-2	2	1
heap	[ ]						

Dijkstra Heap (bubbleUp)



Árvore de Heap:



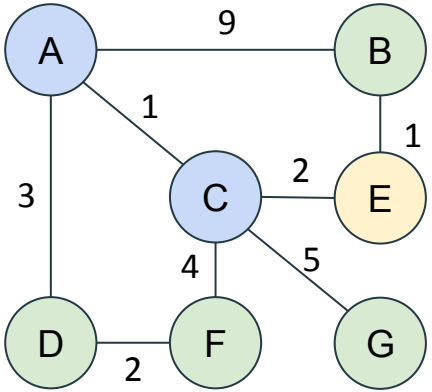
Vértices explorados:

0	1	3
A (-)	C (A)	E (C)

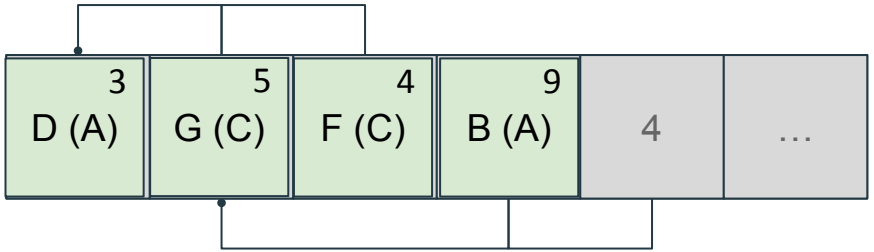
	A	B	C	D	E	F	G
verticePos	-2	1	-2	0	-2	2	3
heap	[ ]						

Árvore Mínima (Prim)

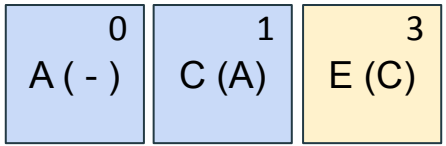
Muito parecido com o que acabamos de ver



Árvore de Heap:



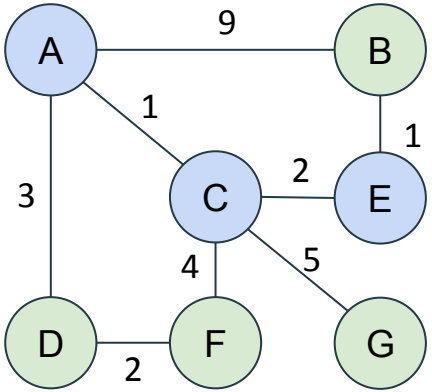
Vértices explorados:



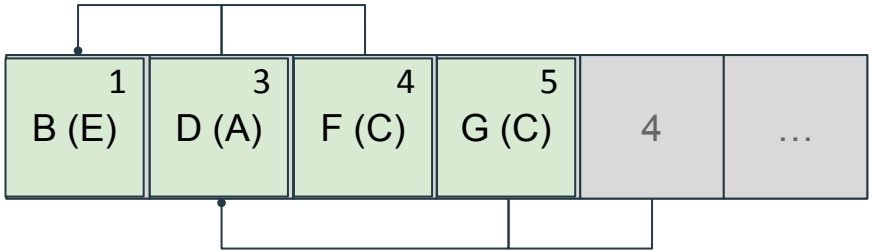
	A	B	C	D	E	F	G
verticePos	-2	3	-2	0	-2	2	1
heap	[ ]						

Árvore Mínima (Prim)

Muito parecido com o que acabamos de ver



Árvore de Heap:



Vértices explorados:

0	1	3
A (-)	C (A)	E (C)

	A	B	C	D	E	F	G
verticePos	[-2,	0,	-2,	1,	-2,	2,	3]
heap	[ ]						

Resultados

Grafo	Tempo médio das distâncias (seg)	Distância (10, 20)	Distância (10, 30)	Distância (10, 40)	Distância (10, 50)	Distância (10, 60)	Peso da MST
grafo 1	$7.95 * 10^{-4}$	1.52	1.48	1.52	1.39	1.38	220.11
grafo 2	$8.91 * 10^{-3}$	2.08	1.92	1.61	1.34	1.69	2247.07
grafo 3	0.173	1.98	2.08	2.14	1.82	2.23	22218.71
grafo 4	3.23	2.46	2.43	2.33	2.36	2.61	221937.54
grafo 5	4.40	14.03	12.01	13.66	9.22	14.47	4785814.85

Resultados

Pesquisador	Distância (até Dijkstra)	Caminho mínimo (com os nomes)
Alan M. Turing	-	(sem caminho)
J. B. Kruskal	3.48 (8 arestas)	**Edsger W. Dijkstra** > John R. Rice > Dan C. Marinescu > Howard Jay Siegel > Edwin K. P. Chong > Ness B. Shroff > R. Srikant > Albert G. Greenberg > **J. B. Kruskal**
Jon M. Kleinberg	2.71 (9 arestas)	**Edsger W. Dijkstra** > A. J. M. van Gasteren > Gerard Tel > Hans L. Bodlaender > Dimitrios M. Thilikos > Prabhakar Ragde > Avi Wigderson > Eli Upfal > Prabhakar Raghavan > **Jon M. Kleinberg**
Eva Tardos	2.75 (11 arestas)	**Edsger W. Dijkstra** > A. J. M. van Gasteren > Gerard Tel > Hans L. Bodlaender > Jan van Leeuwen > Mark H. Overmars > Micha Sharir > Haim Kaplan > Robert Endre Tarjan > Andrew V. Goldberg > Serge A. Plotkin > **Eva Tardos**
Daniel R. Figueiredo	2.94 (8 arestas)	**Edsger W. Dijkstra** > John R. Rice > Dan C. Marinescu > Chuang Lin > Bo Li > Y. Thomas Hou > Zhi-Li Zhang > Donald F. Towsley > **Daniel R. Figueiredo**

Resultados

	Lista encadeada	Árvore de heap	k
grafo 1	169.50ms	61.48ms	100
grafo 2	33.38s	1.44s	100
grafo 3	2h 25m 43s	31.00s	100
grafo 4	14h 45m 27s (~61d)	5.45s (9m 5s)	1
grafo 5	+48h	13.95s (23m 15s)	1