

# Algoritmos e Estruturas de Dados 2

## Trabalho Unidade 2

Rafael Beserra Gomes

Universidade Federal do Rio Grande do Norte

1 de maio de 2013

## Unidade 2

# Implementações Unidade 2

## Sobre as implementações da unidade 2

A implementação de cada estrutura de dados é livre. Entretanto, o grupo deve fornecer uma classe para cada estrutura de dado com um método main que leia um arquivo de entrada (cada estrutura terá uma especificação de entrada) e gere um arquivo de saída (também especificado). Utilize as classes `DataInputStream` e `DataOutputStream` para realizar essas operações com arquivos.

## Arquivos para testes

Arquivos de entrada (.in) e arquivos de saída correspondentes (.out) estão disponíveis na página do professor para testes. Observe que os testes podem não incluir casos em que o algoritmo do grupo não funciona. Testes diferentes serão aplicados na avaliação da implementação.

# Implementações Unidade 2

## Especificações

Você deve assumir que o arquivo de input a ser utilizado segue corretamente cada especificação. Se houver uma *Sequência de operações* significa que nesse trecho da especificação poderá ocorrer qualquer uma das operações no bloco delimitado por chaves, em qualquer ordem e inúmeras vezes.

## Observações importantes que se aplicam a todos os algoritmos

- 1) Você deve assumir que em qualquer operação qualquer referência a um nó é um índice válido no momento da ocorrência dessa operação.
- 2) Qualquer valor definido como  $\infty$  ou  $-\infty$  no seu código, caso venha a ser escrito em file output, deve ser escrito com a string `inf` e `-inf`, respectivamente.

# Busca BFS e DFS

## Busca BFS e DFS

# Implementação BFS e DFS

## Input

<n>: inicializa o grafo com os nós indexados de 0 a n-1

Sequência de operações {

**edge** <i> <j>: especifica que há uma aresta entre i e j **não** orientada,  $0 \leq i, j < n$

}

Sequência de operações {

**shortest** <i> <j>: escreve no file output o menor caminho de i até j e o custo associado,

$0 \leq i, j < n$

**path** <i> <j>: escreve no file output um caminho (qualquer) entre o nó i e o nó j,  $0 \leq i, j < n$

}

## Sample Input

4

edge 1 2

edge 2 3

edge 3 2

shortest 2 1

path 3 0

# Busca BFS e DFS

## Observações importantes

- 1) Caso não haja um caminho existente entre dois nós do grafo, a operação shortest e a operação path deve escrever em file output a string *"No path"* ao invés do caminho e o custo associado (no caso de shortest).
- 2) Toda operação deve escrever uma linha (e somente uma) em file output. Caso alguma operação não especifique saída, escreva em file output um traço "-". Assim é possível associar o número de linha do arquivo de input com o mesmo número de linha do arquivo de output.

# Fila de Prioridades

## Fila de Prioridades



# Implementação Fila de Prioridades

## Input

<min | max>: especifica se é uma fila de prioridade máxima ou mínima

Sequência de operações {

**insert** <id> <chave>: insere a chave com um id externo associado,  $id \geq 0$

**extract**: extrai o min ou max da heap e escreve seu id e chave no file output

**decrease** <id> <delta>: decrementa a chave associada ao id em delta,  $id \geq 0$ ,  $\delta > 0$

**increase** <id> <delta>: incrementa a chave associada ao id em delta,  $id \geq 0$ ,  $\delta > 0$

}

## Sample Input

max

insert 3 5

insert 5 8

insert 8 2

extract

increase 8 4

extract

extract

# Implementação Fila de Prioridades

## Observações importantes

- 1) Utilize uma heap para a fila de prioridades.
- 2) Utilize um vetor para implementar as chaves da heap.
- 3) Como saber de imediato a que elemento externo está associado o mínimo ou máximo da heap?
- 4) Dado um elemento externo, como saber de imediato o índice associado na heap?
- 5) Na inserção, se o elemento externo já estiver presente na heap, não efetue a inserção.
- 6) Caso a heap esteja vazia, ao extrair o mínimo ou máximo, escreva no file output "empty" ao invés do identificador e chave.
- 7) Toda operação deve escrever uma linha (e somente uma) em file output. Caso alguma operação não especifique saída, escreva em file output um traço "-". Assim é possível associar o número de linha do arquivo de input com o mesmo número de linha do arquivo de output.
- 8) Operações increase constam apenas se a heap for máxima e decrease constam apenas se a heap for mínima.

# Conjuntos Disjuntos

## Conjuntos Disjuntos

# Implementação Conjuntos Disjuntos

## Input

<n>: inicializa os conjuntos de 0 até n-1

Sequência de operações {

**compare** <i> <j>: escreve no output true ou false se i e j estiverem no mesmo conjunto,

$0 \leq i, j < n$

**union** <i> <j>: une os dois conjuntos relativos a i e a j,  $0 \leq i, j < n$

}

## Sample Input

4

compare 0 1

compare 1 2

compare 0 3

union 1 2

compare 1 2

# Conjuntos Disjuntos

## Observações importantes

**1)** Toda operação deve escrever uma linha (e somente uma) em file output. Caso alguma operação não especifique saída, escreva em file output um traço "-". Assim é possível associar o número de linha do arquivo de input com o mesmo número de linha do arquivo de output.

# Árvore Geradora Mínima

## Árvore Geradora Mínima

# Implementação Kruskal e Prim

## Input

<n>: inicializa o grafo com os nós indexados de 0 a n-1

Sequência de operações {

**edge** <i> <j> <p>: especifica que há uma aresta entre i e j **não** orientada de peso p,  $0 \leq i, j < n$

<kruskal | prim>: escreve na tela as arestas que fazem parte da árvore geradora mínima utilizando o método kruskal ou prim

## Sample Input

4

edge 1 2 1

edge 2 3 2

edge 3 2 1

kruskal

# Árvore Geradora Mínima

## Observações importantes

- 1) O grafo especificado no arquivo de entrada é conectado.
- 2) O grafo de entrada é não orientado.
- 3) A escrita no arquivo de saída consiste em: (a) uma lista de  $n - 1$  arestas (cada aresta em uma linha) que fazem parte da árvore geradora mínima, cada aresta na forma  $x$   $y$  e (b) o custo total das arestas da árvore.



# Menor caminho: algoritmo de Bellman-Ford

Menor caminho: algoritmo de Bellman-Ford

# Implementação Bellman-Ford

## Input

<n>: inicializa o grafo com os nós indexados de 0 a n-1

Sequência de operações {

**edge** <i> <j> <p>: especifica que há uma aresta entre i e j orientada de peso p,  $0 \leq i, j < n$

**hasNegativeCicle**: escreve no file output true ou false se o grafo possui um ciclo de peso negativo

Sequência de operações {

**shortest** <i> <j>: escreve no file output o menor caminho de i até j e o custo associado,  $0 \leq i, j < n$

## Sample Input

```
6
edge 1 2 1
edge 2 3 2
edge 3 2 1
hasNegativeCicle
shortest 2 5
shortest 1 3
```

# Menor caminho: algoritmo de Bellman-Ford

## Observações importantes

- 1) Caso não haja um caminho existente entre dois nós do grafo, a operação shortest deve escrever em file output a string *"No path"* ao invés do caminho e o custo associado.
- 2) Toda operação deve escrever uma linha (e somente uma) em file output. Caso alguma operação não especifique saída, escreva em file output um traço "-". Assim é possível associar o número de linha do arquivo de input com o mesmo número de linha do arquivo de output.

# Menor caminho: algoritmo de Floyd

Menor caminho: algoritmo de Floyd

# Implementação Floyd

## Input

<n>: inicializa o grafo com os nós indexados de 0 a n-1

Sequência de operações {

**edge** <i> <j> <p>: especifica que há uma aresta entre i e j orientada de peso p,  $0 \leq i, j < n$

}

Sequência de operações {

**shortest** <i> <j>: escreve no file output o menor caminho de i até j e o custo associado,  $0 \leq i, j < n$

}

## Sample Input

4

edge 1 2 1

edge 2 3 2

edge 3 2 1

shortest 2 1

# Menor caminho: algoritmo de Floyd

## Observações importantes

- 1) Caso não haja um caminho existente entre dois nós do grafo, a operação shortest deve escrever em file output a string *"No path"* ao invés do caminho e o custo associado.
- 2) Toda operação deve escrever uma linha (e somente uma) em file output. Caso alguma operação não especifique saída, escreva em file output um traço "-". Assim é possível associar o número de linha do arquivo de input com o mesmo número de linha do arquivo de output.

# Menor caminho: algoritmo de Dijkstra

Menor caminho: algoritmo de Dijkstra

# Implementação Dijkstra

## Input

<n>: inicializa o grafo com os nós indexados de 0 a n-1

Sequência de operações {

**edge** <i> <j> <p>: especifica que há uma aresta entre i e j orientada de peso p,  $0 \leq i, j < n$ ,  $p > 0$

}

Sequência de operações {

**shortest** <i> <j>: escreve no file output o menor caminho de i até j e o custo associado,  $0 \leq i, j < n$

}

## Sample Input

6

edge 1 2 1

edge 2 3 2

edge 3 2 1

shortest 2 5



# Menor caminho: algoritmo de Dijkstra

## Observações importantes

- 1) Caso não haja um caminho existente entre dois nós do grafo, a operação shortest deve escrever em file output a string *"No path"* ao invés do caminho e o custo associado.
- 2) Toda operação deve escrever uma linha (e somente uma) em file output. Caso alguma operação não especifique saída, escreva em file output um traço "-". Assim é possível associar o número de linha do arquivo de input com o mesmo número de linha do arquivo de output.