EP 1: Aplicação de estruturas de dados e algoritmos relacionados

Entrega: 05/02/2018, até às 19h

Tópicos de Programação – IME – USP

Verão/2018

Prof. Arthur Casals (arthur.casals at usp.br)

1. Introdução

Uma das aplicações de árvores de busca binárias envolve situações nas quais existe uma alta volatilidade nos dados utilizados por um sistema (dados entrando e saindo do sistema constantemente). Por exemplo: imagine um sistema de navegação via GPS que deve armazenar, de forma ordenada, quais são os lugares mais próximos de um determinado referencial. Tal sistema poderia ser utilizado a bordo de um caminhão de entregas da seguinte forma:

- No início do dia, o caminhão sempre parte de uma garagem da empresa e deve fazer um certo número de entregas, em diferentes lugares da cidade.

- Com a utilização deste sistema, os destinos das entregas podem ser armazenados de forma ordenada (do mais próximo ao mais longe, tendo a garagem como referência).

- Ao se realizar uma entrega, o nó na árvore binária de busca correspondente ao lugar onde a entrega foi realizada é removido da árvore.

No entanto, em uma metrópole como São Paulo, "distância" normalmente não é o fator limitante: se todas as entregas devem acontecer dentro de um determinado período no dia (horário comercial), é interessante que a árvore binária de busca seja ordenada utilizando-se o critério de **tempo de deslocamento**, ao invés de distância. Nesse caso, os destinos de entrega estariam ordenados do que possui menor tempo de deslocamento ao que possui maior tempo de deslocamento. Adicionalmente, é importante considerar que a garagem de onde sai o caminhão pode ser acessada 24h por dia, 7 dias por semana; desta forma, se a última entrega do dia for realizada no último minuto do horário comercial, o sistema entende que todas as entregas foram feitas com sucesso, e o caminhão não tem prazo de retorno para a garagem.

A utilização de tal sistema requer que o tempo de deslocamento entre cada ponto seja constantemente monitorado, visto que este tempo pode aumentar ou diminuir de acordo com a

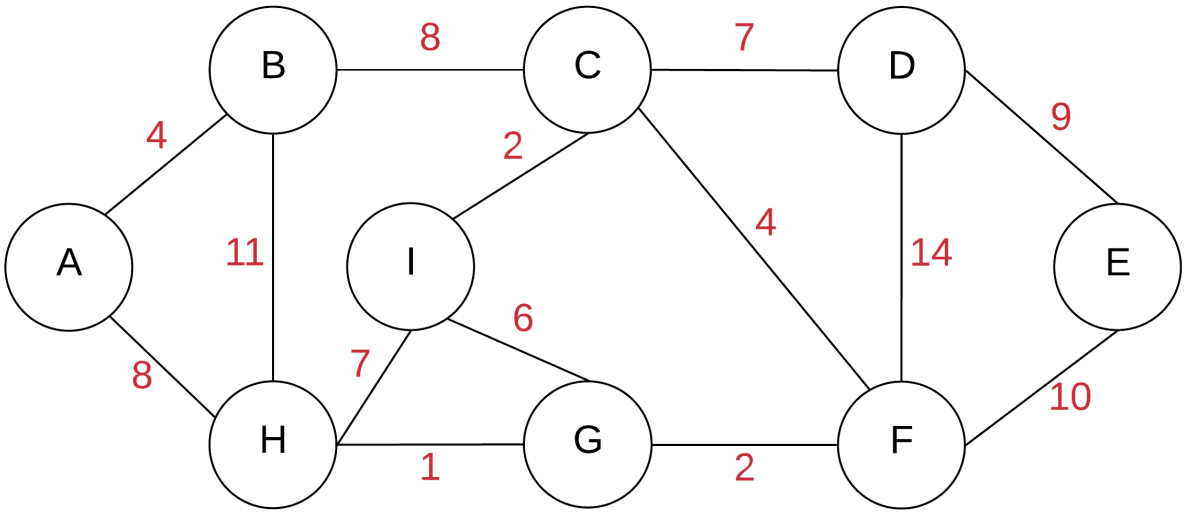
ocorrência de trânsito, acidentes, alagamentos, etc. Neste caso, o sistema passa a possuir dois problemas a serem resolvidos:

- O menor caminho entre todos os pontos de interesse (garagem e destinos de entrega) deve ser constantemente monitorado, de forma que seja sempre possível se obter uma rota ótima (mais rápida possível); e

- Uma vez que a rota ótima seja obtida, esta deve ser convertida em uma árvore binária de busca para que possa ser utilizada pelo sistema.

2. Informações adicionais:

Os vértices de um grafo, depois de ordenados, podem ser armazenados em uma árvore de busca binária de diferentes formas. Considere, por exemplo, o grafo visto na aula do dia 23/01/2018 (slide 30):



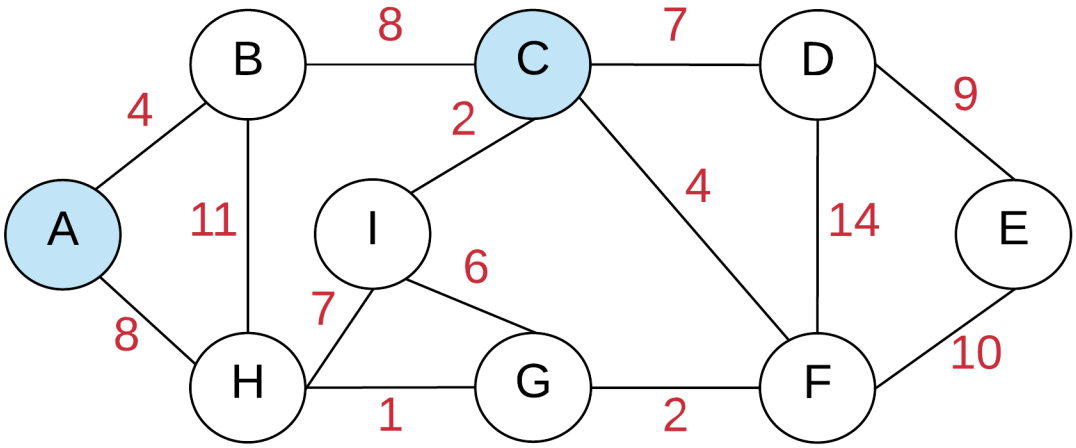
Este grafo foi utilizado para se demonstrar a utilização do algoritmo de Dijkstra na resolução de um problema de caminho mínimo. Na demonstração, chegou-se à seguinte tabela de caminho mínimo, utilizando-se como ponto de partida o vértice A:

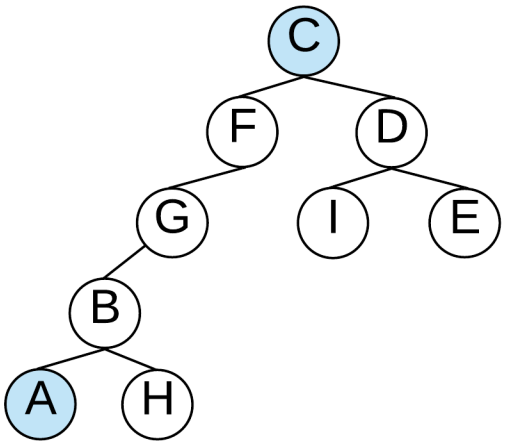
vértices	A	B	C	D	E	F	G	H	I
caminho	0	4	12	19	21	11	9	8	14
predecessor	null	A	B	C	F	G	H	A	C

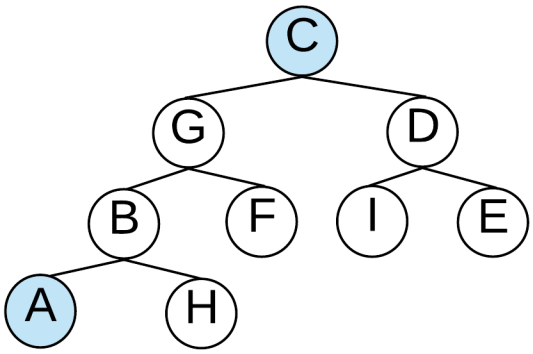
Utilizando-se esta tabela, se ordenarmos os vértices do grafo de acordo com sua distância em relação ao vértice A (do menos distante para o mais distante), obtemos a seguinte lista:

A -> B -> H -> G -> F -> C -> I -> D -> E

A representação desta ordenação utilizando uma árvore de busca binária, no entanto, vai depender do número de níveis existentes na árvore e do vértice escolhido como raiz. Por exemplo: imagine que o vértice escolhido como raiz seja o C. Desta forma, temos abaixo duas possíveis representações para a mesma árvore binária de busca:







Observe que a altura das árvores também varia! No entanto, se utilizarmos o algoritmo de inserção explicado na aula do dia 18/01 (slides 33), teremos a seguinte situação:

1. Considere cada vértice **w** no *array* {C, G, D, B, F, A, H, I, E}, de forma que o vértice C está na posição 0, o vértice G está na posição 1, o vértice D está na posição 2, e assim por diante.
2. Utilizando a ordenação dada por A -> B -> H -> G -> F -> C -> I -> D -> E , faça:
    a. Insira **w** na árvore inicial;
    b. Se a árvore ainda não existir, crie uma árvore utilizando o vértice **w**, sem filhos;

c. Se a árvore já existir e **w** estiver ordenado de forma que venha antes da raiz da árvore, insira **w** na árvore à esquerda da raiz;
d. Caso contrário (ou seja, se a árvore já existir e **w** estiver ordenado de forma que venha depois da raiz da árvore), insira **w** na árvore à direita da raiz.

Considere a representação em C de uma árvore de busca binária utilizada na aula do dia 18/01 (slide 29). Este algoritmo pode ser representado **recursivamente** pelo seguinte pseudocódigo:

```
InsereVertice(ARVOREPTR *arvore, vértice w)
    Se *arvore for igual a nulo:
        Aloca espaço em memória para a variável *arvore
        *arvore->info recebe w
        *arvore->esq e *arvore->dir recebem valor nulo
    Senão
        Se w vier antes de *arvore->info
            InsereVertice(&((*arvore)->esq),w)  //notação em C
        Senão
            InsereVertice(&((*arvore)->dir),w)  //notação em C
    Fim
```

Ao executarmos este pseudocódigo, iremos obter a árvore à direita na figura anterior. Observe que:

- O primeiro vértice no *array* utilizado vai ser sempre o nó raiz da árvore obtida; e

- A estrutura da árvore obtida também depende da ordem em que os outros vértices são processados (ou seja, suas posições no *array*).

3. Problemas a serem resolvidos neste EP:

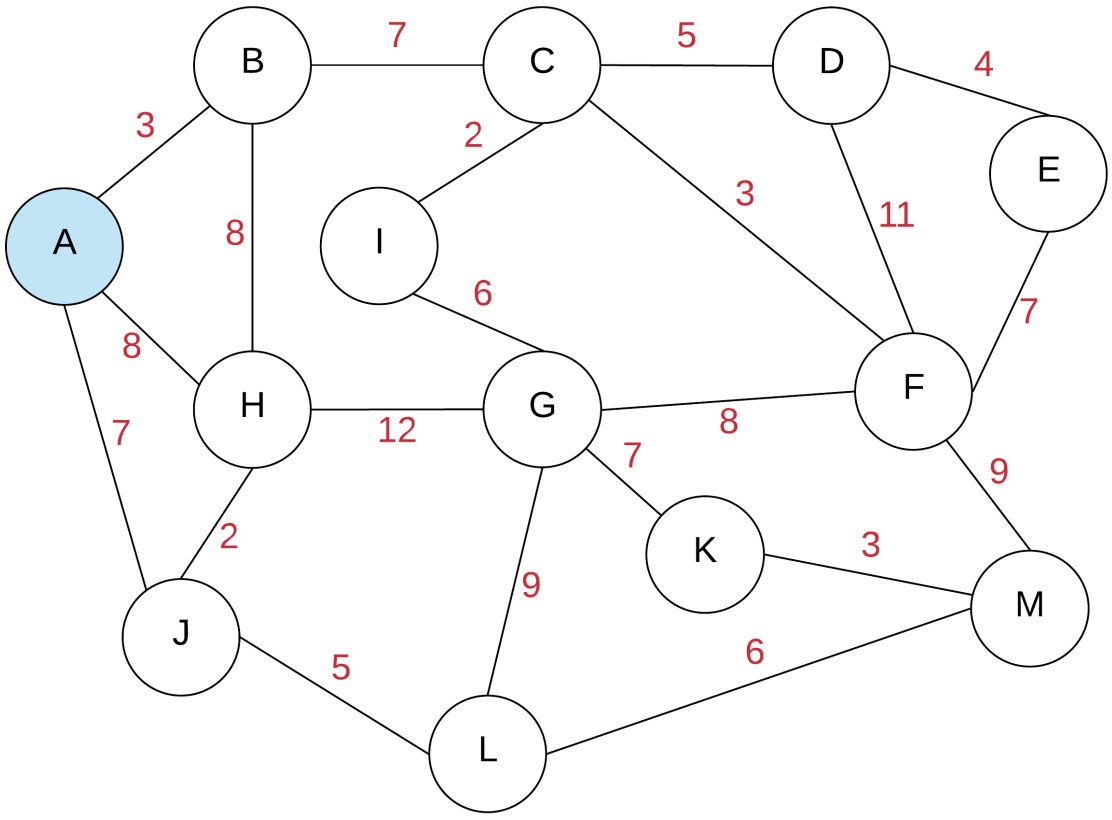
Considerando o cenário exemplificado mais acima, existem os seguintes problemas a serem resolvidos:

- **Problema 1:** O menor caminho entre o vértice de origem (garagem) e todos os outros vértices (destinos de entrega) deve ser achado através do algoritmo de Dijkstra.

- **Problema 2:** A partir do resultado do problema anterior, uma ordem entre os vértices deve ser estabelecida, e uma **lista simples ligada** deverá ser criada contendo todos os vértices do grafo **de forma ordenada**.

- **Problema 3:** A partir da **lista ordenada** contendo todos os vértices do grafo e de uma **lista inicial de vértices**, construir a árvore binária de busca contendo todos os vértices.

Considere o grafo abaixo:



A partir deste grafo, e **considerando o vértice de origem como sendo o vértice A,** resolva os problemas 1, 2 e 3 apresentados. Estes problemas **devem** ser resolvidos da seguinte forma:

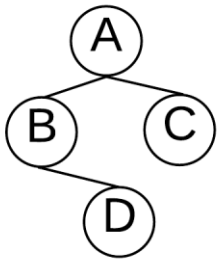
- **Problema 1:** deverá ser resolvido e entregue nos mesmos moldes do exercício dado em aula no dia 23/01/2018. **A resolução deste problema, portanto, requer tabelas com "caminho" e "predecessor" para cada passo da execução do algoritmo de Dijkstra. ESTE PROBLEMA NÃO REQUER IMPLEMENTAÇÃO DE CÓDIGO!**

- **Problema 2:** deverá ser resolvido utilizando-se os caminhos mínimos obtidos no problema anterior. A resolução deste problema requer:

    - Implementação de estrutura de lista simples ligada, com os métodos de ***Inserção*** e ***Busca*** de elementos;
    - Inserção dos vértices do grafo de forma ordenada. A ordem deve ser: o vértice de menor caminho mínimo em relação a A (o próprio A) deve ocupar o início da lista (*head*), e os próximos vértices devem ser ordenados de acordo com o tamanho do caminho mínimo (linha "*caminho*" da tabela obtida no Problema 1), de forma que o último elemento da lista seja o mais distante do vértice A (maior valor na linha "*caminho*"). **NÃO É NECESSÁRIO IMPLEMENTAR CÓDIGO PARA ORDENAR OS VÉRTICES!** Basta deixar indicado, em forma de comentário no código, a ordem de vértices utilizada no momento de se preencher a lista. **Caso dois vértices possuam o mesmo valor de "*caminho*", ordene-os em ordem alfabética.**

- **Problema 3:** deverá ser resolvido utilizando-se os conceitos vistos em aula e no enunciado deste EP. A resolução deste problema requer:
    - Implementação de uma lista inicial de vértices para preenchimento da árvore, **dada por: {I, D, B, F, C, G, J, L, H, M, K, A, E}** (implementada no código em forma de *array*, de forma que o elemento na **posição 0** seja o **vértice I**, o elemento na **posição 1** seja o **vértice D**, o elemento na **posição 2** seja o **vértice B** e assim por diante);
    - Implementação de um método para verificar se um dado vértice do *array* está antes ou depois de um determinado vértice na lista ligada;
    - Implementação de estrutura de árvore de busca binária;
    - Implementação do método de inserção de nós de forma ordenada em uma árvore de busca binária, conforme descrito no texto acima;
    - Implementação de código que exiba a estrutura da árvore de busca binária resultante. Basta que sejam exibidos na tela, **para cada nó da árvore**: o nome do nó atual, o nó imediatamente abaixo à esquerda e o nó imediatamente abaixo à direita. **Os nós à esquerda e à direita devem ser identificados!**

Para exibição de uma árvore na tela, observe o exemplo de árvore abaixo:



Para esta árvore, espera-se que apareça o seguinte texto na tela:

NÓ: A ESQ: B DIR: C

NÓ: B ESQ: NULL DIR: D

NÓ: C ESQ: NULL DIR: NULL

NÓ: D ESQ: NULL DIR: NULL

4. Regras:

Podem ser utilizados todos os códigos vistos em sala de aula. **NÃO PODEM** ser utilizadas bibliotecas de estruturas prontas, algoritmos, ou qualquer outra que reduza o esforço esperado para a solução do EP.

5. Entrega:

A entrega do EP 1 **deverá conter**:

A. **Um relatório,** contendo:
   - A resolução do problema 1, nos moldes do exercício entregue no dia 25/01 (com as tabelas de todos os passos – NÃO É NECESSÁRIO listar todos os menores caminhos encontrados)
   - A ordenação dos vértices encontrada e utilizada no problema 2
   - Uma representação (desenho ou figura) da árvore de busca binária encontrada na solução do problema 3
   - Maiores dificuldades encontradas durante a resolução do EP

B. **Código-fonte da implementação**, em C ou JAVA