1.1

Neste exercicio obtou-se por uma implementação recursiva ou seja em profundidade. O algoritmo recebe como parametro um nó,a root, e sabe-se que todos os nós à esquerda da root são menores e à direita da root são maiores (pois é uma BST) .É utilizado como comparador o compare que retorna um valor positivo que o primeiro parametro for maior que o segundo, negativo se o segundo for maior que o primeiro e zero se forem iguais. As condição de paragem são a chegada ao nó que é limitado pelos parametros min e max e a existência de uma root que tenha os filhos a null, isto significa que já não tem mais filhos para percorrer e retorna o valor existente no campo bool. Se o valor value estiver entre esses valores é returnado true. Este algoritmo não percorre a arvore toda, percorrendo apenas um nó por nível por isso podemos dizer que tem custo O(log(n)), semelhante ao algoritmo binary search.

1.3

Algoritmo recursivo que recebe como parametro a raiz da arvore binária e verifica se é realmente uma árvore binária retornando true se for e false se não for. Este método tem como condição de paragem a verificação se existe nós filhos da raiz (!=null). Retorna true se que a recursão chegou ao fim(não tem mais nós para verificar) indicando que aquele ramo segue as especificações de uma arvore binária de pesquisa, pois caso contrário é retornado false no nivel mais profundo da recurção até chegar ao topo e lançado como retorno final do método. A especificação é que o nó esquerdo filho da raiz que é passada seja menor que o nó da raiz e o nó direito filho seja maior se isto se verificar em toda a árvore podemos concluir que se trata de uma árvore binária de pesquisa. Neste caso o algoritmo começa por verificar tudo para a esquerda e só depois de chegar ao fundo é que verifica a direita. Este algoritmo tem custo O(n) pois tem que percorrer todos os nós para ver se estão de acordo com o pretendido.

3.

Para alem dos campos indicados no enunciado para a class Vertex foram ainda adicionados os seguintes campos:

double cost – o custo de cada vértice tendo como referência a posição atual do percurso.Ou seja, o custo necessário para se deslocar a cada um dos vértices adjacentes.

boolean isVisited – indica se o vértice já foi visitado durante o percurso.

Int parent – indica o indentificador do vértice que antece o vértice currente no percurso.

Obtou-se por uma MST de acordo com o algoritmo de Prim. Este é um algoritmo greedy que serve para encontrar a menor árvore geradora num grafo conexo, com valores e não orientado.

Começa-se no vértice com menor identificador e assume-se que todos os vértices adjacentes a esse têm custo infinito. Todas as arestas são colocadas numa linkedList , escolhe-se a aresta de menor custo e move-se a referência para o vértice adjacente dessa aresta. Essa aresta é removida da lista e adicionam-se todas as outras adjacentes a esse novo vértice e volta-se a colocar todos os vértices adjacentes com o custo à expepção do que ja foi visitado que fica a zero, pois todos os de custo zero não podem ser visitados de novo. Este processo é repetido até todos os vértices terem sido percorridos e sem haver loop no caminho, pois não se avança para vértices que já tenham sido visitados. Á medida que se avança na árvore vai sendo desinhado o caminho, atribuíndo o valor do identificador dos vértices que antecedem um determinado vértice, sabendo assim qual é o caminho a seguir na MST. Para verificar se os ids passados como parametro pertencem ao grafo foi necessário ver se o id inicial é parent do vértice adjacente e vice-versa.