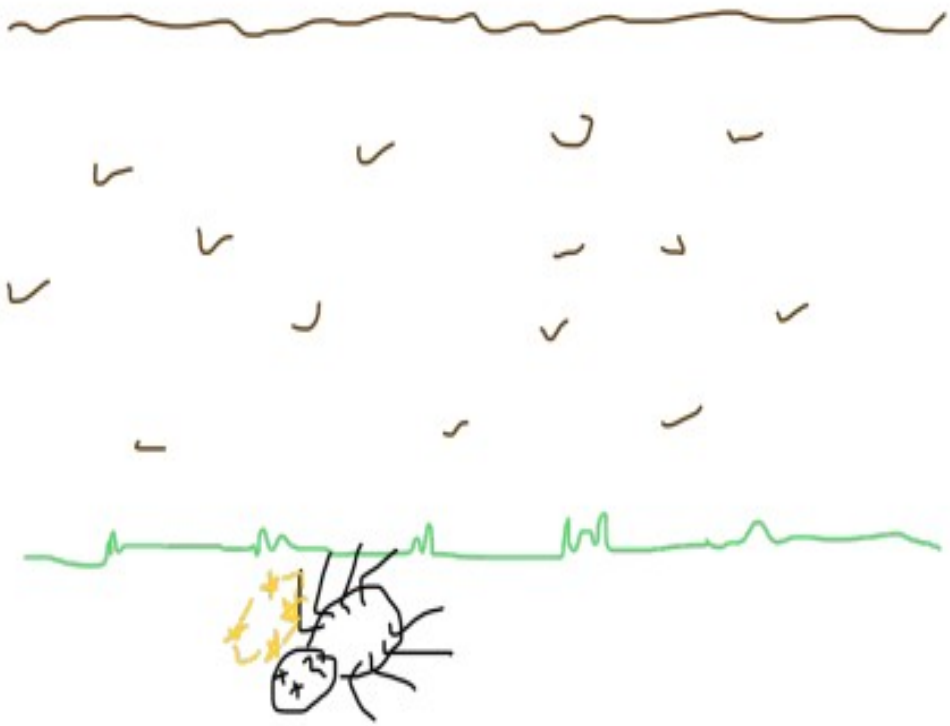
UNIVERSIDADE DE ÉVORA

Problem: Hill, the Climber



Abril 2022

Rafael Silva 45813

Vicente Romão 45467

(Grupo 208)

1 Introdução

Este problema consiste num alpinista que tem como objetivo chegar do chão ao topo de um penhasco. Para isso, existem várias formas de chegar ao cimo mas com atenção ao intervalo entre os *hold points* (pontos estratégicos que são escolhidos de forma a tentar chegar ao topo) onde o alpinista pode chegar porque ele pode não chegar lá. A partir destas restrições, pretende-se calcular o melhor caminho até ao topo, atingindo o menor número de *hold points* pelo alpinista.

A resolução deste problema, baseia-se na **construção de um grafo não orientado** com o objetivo de chegar a uma solução ótima do problema através de uma **pesquisa em largura(BFS)**.

2 Desenvolvimento do Algoritmo

2.1 Input

O nosso algoritmo começa por ler a primeira linha do input, com o auxilio da classe **BufferedReader** e **System.in**.

A primeira linha do input contem três inteiros separados por um "espaço", através do método **split** são guardados num *array* de *Strings* e a partir daí define-se que a primeira variável corresponde ao número de *hold points,* a segunda variável corresponde à altura da parede e a terceira variável corresponde ao número de casos teste.

2.2 Criação do Grafo

Na criação do grafo começou-se por ler do input todos os *hold points* sendo cada um representado através de uma classe (*holdpoint)* com as instâncias int x, int y, (representando as coordenadas) e um ArrayList do tipo aresta (classe com instâncias *hold point* e int peso). Esses *hold points* são adicionadas como nós ao grafo e também são adicionados a uma lista que irá ser usada para ordenar *hold points.* Posteriormente, calculámos o reach máximo do conjunto de testes de forma a que não seja necessário criar um grafo para cada teste, fazendo apenas um grafo com o reach máximo.

Percorre-se a lista ordenada do primeiro ponto até ao fim, verificando todos os seguintes pontos da lista. Calcula-se as respetivas distâncias até ao final da lista ou até chegar a um ponto em que a distância das ordenadas é superior ao reach. Quando reach é possível, adiciona-se uma aresta ao grafo com os dois pontos.

2.3 Ordenação de *hold points*

Todos os *hold points* são adicionados a uma lista e ordenados tendo em conta as ordenadas de modo a facilitar o processo de criação de arestas do grafo.

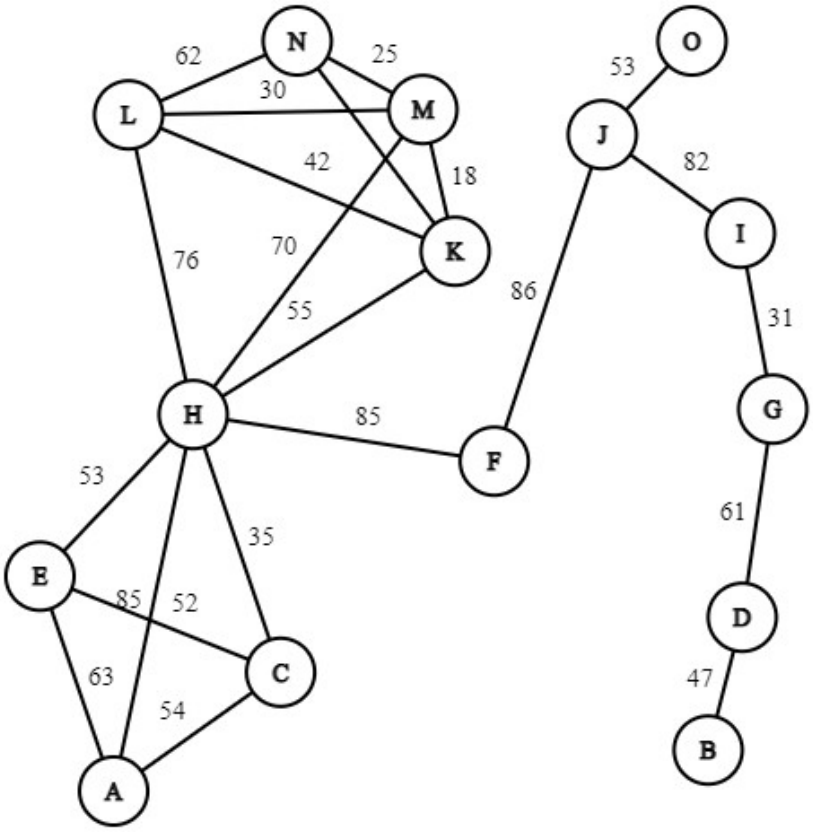
A partir desta lista é criada outra lista com os *hold points* em que podemos começar a pesquisa.

2.4 Descoberta do número mínimo de nós para chegar ao topo

Começamos por utilizar uma queue do tipo *holdpoint* onde vai adicionar todos os pontos que permitem começar e prosseguir com a pesquisa em largura (BFS), ao mesmo tempo esses pontos vão ficar com a cor GREY e distância 0. Depois vai começar a esvaziar a queue e a cada *hold point* retirado da queue, vai verificar todos os pontos de adjacência, ou seja, se têm cor WHITE (significa que são pontos que não foram visitados) e se o peso da aresta entre esses dois pontos é menor ou igual ao reach. Se isso acontecer, o ponto de adjacência vai mudar para GREY e a distância desse ponto vai ser a distância do ponto que foi removido da queue+1 de modo a obtermos a distância do inicio aquele ponto. A pesquisa termina quando houver um ponto em que se pode chegar ao topo da parede ou quando não for possível atingir o topo (“unreachable”).

3 Descrição do grafo

Nós optamos por criar um grafo não orientado pois no caso da ilustração do grafo tanto podemos ir do nó A para o C como podemos ir do nó C para o A de forma a tentar descobrir qual será o caminho com menos nós para chegar ao topo. O grafo é pesado para tornar o nosso algoritmo mais eficiente. No inicio é calculado o reach máximo de modo a que seja feito um só grafo, em vez de um grafo para cada teste.



3 Análise de Complexidades

Linhas	Temporal	Espacial
19-30 beginNode()	O(n)	-----
n é o tamanho da lista ordenada de pontos (listOrderly)		
47-53	O(n)	$O(n^2)$
n representa o número de *hold points*		

55 Collections.sort()	O(n*log n)	O(n)
n é o tamanho da lista ordenada de pontos (listOrderly)		
58-63	O(n)	O(n)
n representa o número de casos teste		
69-84	$O(n^2-n)$	$O(n^2)$
n é o tamanho da lista ordenada de pontos (listOrderly)		
85-109	O(n*(m+l)	O(v)
n representa o tamanho da lista de casos teste (reachList) m representa o tamanho da lista (holdPointBegin) (**191-198**) l representa o número de aresta do grafo (**200-219**) v representa o número de *hold points*		
181-186 removeQueue()	O(n)	O(n)
n é o tamanho da lista ordenada de pontos (listOrderly)		

4 Conclusão

Em conclusão, neste trabalho tivemos como objetivo determinar o melhor caminho até ao topo, atingindo o menor número de *hold points* pelo alpinista. Começamos por desenvolver em papel um grafo do exemplo mencionado no enunciado e chegamos à conclusão que durante a execução do algoritmo seria melhor implementar apenas um grafo não orientado pesado tendo em conta o maior reach inserido no input. Depois disso, restava apenas saber de que forma é que deveríamos fazer a pesquisa para descobrir o melhor caminho e como tal, optámos por implementar uma pesquisa em largura (BFS).

Foram realizadas todas as tarefas propostas pelo enunciado do trabalho e não encontramos nenhum problema ao implementar este algoritmo.