Projeto e Análise de Algoritmos
Problema 5 - Algoritmos Gulosos
Angelina Gomes RA: 120194

Problema: Você precisa dirigir um carro de uma cidade A até uma cidade B. Um tanque cheio do carro possui autonomia para viajar m quilômetros e você sabe as distâncias, a partir de A, dos postos de combustível existentes no caminho. Sejam d1 < d2 < … < dn as distâncias dos n postos do caminho. Você deve encontrar onde você deve parar para abastecer o carro de forma a fazer o menor número de paradas para chegar até B.

a) Forneça um algoritmo guloso para resolver o problema.

A cada passo escolher a maior distância possível para compor o vetor solução, ou seja, a maior distância entre o ponto atual e o próximo posto, ou cidade B se for o caso, que seja menor que m. Para isso, inicialmente, procura-se pelo posto mais distante de A sendo essa distância menor que m e para os demais passos, procura-se, analogamente, pela maior distância entre o posto que está e os próximos postos possíveis, utilizando suas distâncias até A para calcular a distância que deve ser percorrida sendo que esta deve ser menor que m, como já mencionado anteriormente.

Para o código em C, os casos de teste foram considerados possuindo as seguintes formas de entrada e saída:

- *Entrada:*

A primeira linha de um caso de teste contém os inteiros AB, indicando a distância entre as cidades A e B, N, indicando o número de postos no caminho, e M, indicando a distância que se pode percorrer com um tanque cheio. Em cada uma das próximas N linhas são apresentadas as distâncias, a partir de A, dos N postos no caminho.

- *Saída:*

A saída de cada caso de teste possui duas linhas. A primeira apresenta o menor número de postos onde deve-se parar e a segunda apresenta os números de cada posto (ordinalidade partindo de A) onde deve-se parar.

Exemplos:

Entrada	Saída
230 7 60	4
20 40 100 130 150 170 220	2 3 5 6

Entrada	Saída
230 7 130	1
20 40 100 130 150 170 220	4

Entrada	Saída
120 4 130	0
40 70 100 110	

Código:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void busca_posto(int d[], int n, int m, int *atual){
   int i;

   for(i = *atual+1; i < n; i++){
      if(*atual < 0){ //encontrar primeiro posto
         if(d[i] > m) //posto que nao da pra ir => parar no anterior
            break;
      } else{ //encontrar demais postos
         if(d[i]-d[*atual] > m) //posto que nao da pra ir => parar no anterior
            break;
      }
   }
   *atual = --i;
}

int main(){
        int AB, N, M, i, atual = -1, nPostos = 0; //atual: ultimo posto considerado,
nPostos: numero de postos onde deve parar

   scanf("%d %d %d", &AB, &N, &M);

   int d[N], postos[N];
   for(i = 0; i < N; i++)
      scanf("%d", &d[i]);

   memset(postos, 0, N*sizeof(int)); //inicializando com 0

   if(AB <= M){ //nao precisa parar em nenhum posto
      printf("0\n\n");
      return 0;
   }
   busca_posto(d, N, M, &atual); //primeiro posto
   if(atual < 0){ //impossivel ir de A a B com um tanque dessa capacidade
```

```
        printf("-1\n\n");
        return 0;
    } else{
        nPostos = 1;
        postos[0] = atual+1; //indices do vetor: 0 a n-1, resposta: 1 a n

        if(AB - d[atual] <= M){ //da pra ir do posto atual direto para B
            printf("1\n%d\n", postos[0]);
            return 0;
        }

        while(AB - d[atual] > M){ //ainda precisa parar
            busca_posto(d, N, M, &atual);
            postos[nPostos] = atual+1;
            nPostos++;
        }
        printf("%d\n", nPostos);
        for(i = 0; i < nPostos; i++)
            printf("%d ", postos[i]);
        printf("\n");
        return 0;
    }
}
```

b) Mostre que o algoritmo encontra a solução ótima.

Seja S a solução do algoritmo guloso, que escolhe sempre o posto mais distante possível em cada passo, e C alguma solução ótima para o problema, sendo p e q, respectivamente, os tamanhos de S e C, têm-se os seguintes casos:

- se $s_i \in S$ e $s_i \in C \mid \forall_{i,\ 0<i\leq p}$, então não existe nenhum elemento $c_k \in C$ tal que $c_k \notin S$, caso contrário, teria-se $q > p$ e portanto C não seria uma solução ótima;
- suponha que existe algum $s_i \in S$ tal que $s_i \notin C$, então existe algum $c_k \in C$ tal que $c_k \notin S$, pois, para todos os $s_j \in S \mid j < i$, tem-se que $s_j \in C$ e comparando s_i com todos $c_k \mid k \geq i$ tem-se que $c_k < s_i$, uma vez que $c_k = s_i \Rightarrow s_i \in C$, o que contradiz a suposição inicial, e $c_k > s_i \Rightarrow c_k \in S$, uma vez que S possui os maiores elementos possíveis e, como todos os elementos anteriores a c_k estão em S, c_k pertenceria a S em detrimento de s_i, já que $c_k > s_i$, o que também contradiz a suposição inicial. Dessa forma, confirma-se que $s_i > c_k$ e portanto $c_k \notin S$, já que s_i é o elemento que o algoritmo guloso escolheria.

Portanto, $p = q$ e, sendo q o tamanho da solução ótima, tem-se que S também é uma solução ótima.