Universidade Federal de São Paulo - Campus São José dos Campos
Disciplina: Projeto e Análise de Algoritmos

Problema 5: Migração de Aves

Rosângela Miyeko Shigenari, 92334

3 de julho de 2018

1 Introdução

Este trabalho visa a resolução de um problema da migração de aves com a aplicação dos conceitos e programação vistos em aula, em linguagem C/C++ e em seguida, realizar o estudo de sua complexidade.

2 Problema

Todos os anos, o icetê (Avium profugus), uma avezinha verde e branca, com pouco menos de 40 centímetros de tamanho e 2 quilos de peso, realiza um longo vôo de pólo a pólo. Durante três meses, ele percorre milhares e milhares de quilômetros, do Circulo Polar Ártico até o limite da Antártida, ao longo do Oceano Pacífico. Quando o inverno se aproxima no Hemisfério Norte, milhares de aves dessa espécie deixam seu lugar de origem à procura de alimentos e temperaturas mais elevadas. E haja fôlego para voar durante vários dias sem parar. Isso só é possível porque a hipófise dá uma força, ativando funções hormonais que produzem uma camada de gordura debaixo da pele das aves, proporcionando-lhes o combustível necessário. Essa gordura representa a metade do peso total de seu corpo. Um icetê não consegue voar sem esse estoque de gordura, embora ele possa voar até que toda essa gordura seja queimada. Estudos mostram cálculos interessantes sobre o rendimento dessas reservas de gordura. Com seu depósito de gordura, um icetê é capaz de voar durante 24 horas, percorrendo uma distância de 200 quilômetros, sem parar. Ao longo do percurso, existem n ilhas com quilometragens iguais a $a_1 < a_2 < ... < a_n$, em que cada ai é medido a partir do ponto inicial (0 km), nas quais os indivíduos dessa espécie costumam parar para um merecido descanso. Os únicos lugares que um icetê pode pousar e se alimentar são essas ilhas, mas ele não precisa parar em todas. Sua viagem termina no quilômetro an, que é o seu destino. Estudos mostram que indivíduos dessa espécie seguem uma determinada rota, orientando-se pelo campo magnético terrestre, de forma a minimizar o número de paradas para se alimentar. Escreva um programa que leia as quilometragens dessas n ilhas e determina a quantidade de escalas efetuadas por um icetê durante o processo migratório.

Entrada: A entrada consiste de n + 1 linhas, sendo a primeira contendo o valor de n (n<300). As demais linhas contém as quilometragens $a_1, a_2, ..., a_n (a_n < 25000)$, representadas por um número inteiro, sendo um único número em cada linha.

Saída: A saída deverá conter um número inteiro indicando a quantidade de escalas no percurso.

Exemplos de entrada e saída

- **Entrada 1**:
  11

75
179
193
223
407
605
684
709
767
962
1000
Saída 1:
6

- **Entrada 2**:
9
74
160
177
240
318
405
433
486
600
Saída 2:
3

3 Solução

A ideia implementada na solução é, primeiramente, calcular as distâncias em relação as ilhas adjacentes como pode ser observado na figura 1

```
for(i = 0; i<n; i++){
  dist[i] = a[i] - aux; //calcula a distancia em relacao a ilha anterior
  aux = a[i];
}
```

Figura 1: Cálculo da distância de cada ilha

Dada a entrada do vetor a de entrada, o laço atribuirá ao vetor auxiliar *dist* a distância relativa de cada ilha.
O problema descrito afirma que não é necessário parar em todas as ilhas, portanto para otimizar a o percurso até o destino da ave, a melhor solução seria realizar uma parada apenas quando seu estoque de gordura não for suficiente para chegar até a próxima ilha.
Portanto, foi criado um laço que realiza essa verificação como apresentada na figura 2 É realizada uma contagem

```
for(i = 0; i< n - 1; i++){
  cont_distance = dist[i] + cont_distance;
  prox = cont_distance + dist[i+1];
  if(prox > 200){ //verifica a distancia do ponto seguinte
    cont_distance = 0; // reabastece o estoque de gordura na ilha i
    result++; //segue caminho
  }
```

Figura 2: Determinação de uma parada em uma ilha

para determinar a quilometragem do ponto inicial (km 0) até a ilha i (utilizando o contador *cont_ distance*), utilizando esse contador é verificado se o estoque de gordura na ave é suficiente para chegar até a próxima ilha, realizada a partir da soma do contador com a distância até a próxima ilha, se o valor ultrapassar 200 (limite que uma ave pode percorrer), ela deverá se alimentar nesta ilha, neste caso, o contador deve ser zerado (já que a ave está alimentada, a quilometragem pode ser zerada pois ela poderá voar, novamente, mais 200 km) e assim, poderá seguir para a próxima ilha contabilizando mais 1 no contador *result*, que armazena a quantidade de paradas realizadas pela ave. Na Figura 3 é apresentada o esboço da entrada 1 em representação de grafo, onde cada vértice representa uma ilha, onde 0 é a posição inicial da ave e cada aresta representa a distância entre 2 ilhas.

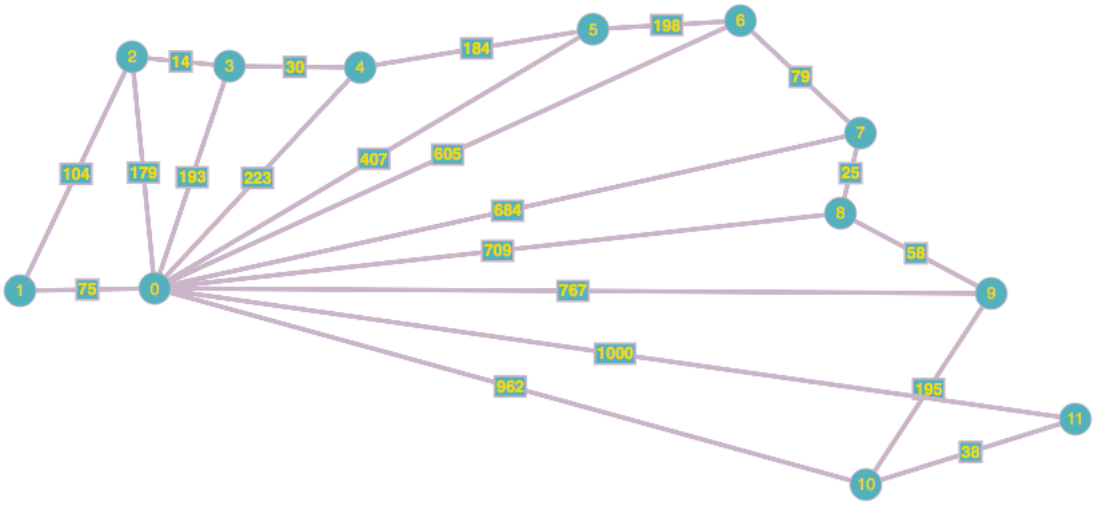

Figura 3: Representação em grafo da entrada 1

3.1 Função Principal

A função principal do algoritmo é apresentada na Figura 4. É inserido como entrada o n, que é o número de

```c
int main(){

   int n, i;
   scanf("%d", &n); //numero de ilhas
   int a[n];

   for(i = 0; i<n; i++)
     scanf("%d", &a[i]); // entrada da quilometragens

   printf("%d\n", CalcDistance(a, n));

  return 0;
}
```

Figura 4: Função Principal

ilhas, em seguida é inserido também em um vetor a os valores das quilometragens em relação ao ponto inicial (km 0), inseridas na função *CalcDistance*, que retorna a quantidade de paradas realizadas pela ave, imprimindo este valor para o usuário.

3.2 Análise do Algoritmo

A função que realiza o cálculo da solução consiste em 2 laços independentes, iterando n e n-1 vezes, respectivamente, logo o custo obtido na função *CalcDistance* é

$$C(n) = n + (n - 1) = 2n - 1$$

Logo, temos que a complexidade deste algoritmo é de **O(n)**.

3.3 Algoritmo Guloso

O problema tratado é um exemplo de algoritmo guloso pois tomam uma decisão baseada apenas em características do estado atual do problema. Buscando sempre percorrer a maior distância ustilizando o combustível sufiente.
Um algoritmo guloso é míope: ele toma decisões com base nas informações disponíveis na iteração corrente, sem olhar as consequências que essas decisões terão no futuro. Um algoritmo guloso jamais é revisado ou possui sua solução reconstruída: as escolhas que faz em cada iteração são definitivas.