Anderson Andrei da Silva, 8944025

Relatório EP1- Conjectura de Collatz

1.0 CONJECTURA DE COLLATZ

Considere a seguinte operação em um número inteiro positivo arbitrário tal que:

- Se o número é par, divida-o por 2;
- Se é ímpar, multiplique-o por 3 e some 1.

A conjectura diz que se aplicarmos repetidas vezes a função acima em um número interiro, vamos, necessariamente, convergir para 1 (onde a função é estacionária).

2.0 TRATAMENTO DE DADOS

Como para cada n ímpar que recebermos teremos que multiplicá-lo por 3 e somar 1, prevê-se que este número, se já arbitrariamente grande, ficará maior ainda, podendo então exceder a capacidade de armazenamento da memório RAM do computador, que é limitada.

A fim de tratar esse problema foram aplicados dois métodos:

a) Sabendo que é impossível retardar para sempre o uso total da memória RAM, foi utilizada um tipo de variável que dobrará o valor máximo que seria permitido inicialmente e não irá considerar números negativos, então, ao invés de se utilizar variável do tipo int, utilizaremos variáveis do tipo unsigned long int para armazenar os números da sequência dada pela função de Collatz.

b) Tratando matemáticamente o problema, se a função f(x) = 3x+1 será utilizada para um x inteiro ímpar, seja x = k, teremos então que f(k) = 3k+1, que necessáriamente será um número par, e então em um próximo passo aplicaríamos f(x) = x/2 , onde x é um inteiro par, e então teríamos f(k) = (3k+1)/2 = (3k)/2 + ½ = 1,5k + 0,5.

Dessa forma, sabendo que utilizar f(x) = 3x+1 fará com que um k ímpar arbitrariamente grande fique muito maior, utilizaremos a função f(x) = 1,5x + 0,5 de forma que cada vez que usada atualizaremos o contador de iterações em 2, pois estamos "adiantando" um passo, mas então faremos com que aquele k ímpar arbitrariamente grande não fique tão maior como algo na ordem de 3k e sim metade disso, 1,5k.

c) A fim de agilizar o processamento dos dados e evitar operações aritméticas repetitivas e portanto desnecessárias será um utilizado um vetor onde cada posição armazenará o próximo valor resultante da função quando o valor daquele índice em questão for posto na função. Dessa forma, se o valor já estiver lá basta olhar qual será o próximo e então ir para a próxima posição até que o valor seja um ou não exista um valor calculado em uma delas, nesse momento será calculado e armazenado. Assim quando for necessário utilizar um valor que eventualmente já tenha sido calculado basta utilizá-lo no vetor e então teremos o seu resultado e respectivamente a sequência de números que parte dele até chegar em 1.

Novamente, não iremos aqui com esses 3 métodos sanar o problema da limitação física da memória RAM, apenas desejamos retardar ao máximo o seu esgotamento

utilizando uma variável com capacidade maior e trabalhando com números razoávelmente menores.

Usaremos ainda nosso vetor com alocação dinâmica, verificando se foi possível alocar a quantidade de memória necessária para executar o programa, nesse caso, foi estipulado 256 MB.

3.0 IMPLEMENTAÇÃO

Na implimentação desse programa utilizamos as seguintes funções com seus respectivos escopos:

- **unsigned long int collatz (int n) :** Recebe um inteiro n e verifica se o número recebido é ímpar ou par e então retorna o resultado de f(n) onde f é a função da conjectura de Collatz.

- **void initVetor(unsigned long int *prox) :** Recebe um ponteiro para um vetor de unsigned long int e irá inicializar as posições de 2 até MAX do vetor com o valor -2.

O programa se sustenta na utilzação de um vetor que em cada posição guarda o resultado do valor da função aplicada naquele índice, ou seja, cada posição guarda o próximo valor a ser gerado pela função. Dessa forma, uma vez calculada a sequência toda a partir de um inteiro x, não será necessário calculá-los novamente, basta apenas percorrer o vetor e será possível encontrar a sequência dada pelas posições do vetor. Ex:

x = 16 : 16 → 8 → 4 → 2 → 1

```
   0  1  2   3  4   5   6   7   8   9  10 11 12 13 14 15 16 17
v[ |  | 1 |  |  |  |  |  | 4 |  |  |  |  |  |  |  | 8 |  ]
```

Para tratar as casa vazias, saberemos que elas assim estão quando tiverem o valor -2 nelas, sendo postos no vetor pela função initVetor. E as posições 0 e 1 receberão -1 para se diferirem, pois não são consideradas vazias, mas também não serão utilizadas. Então nosso vetor ficará:

```
   0   1  2  3  4   5 6  7  8   9 10 11 12 13 14 15 16 17
v[-1|-1| 1 |-2|-2|-2|-2|-2|-2| 4 |-2|-2 |-2 |-2 |-2 |-2 | 8 |-2| ...]
```

E como dito à cima os casos ímpares serão tratados de forma diferente fazendo com que sejam acessadas apenas as posições pares de memória, então, entre um ímpar e o próximo par da sequência que será apontada na segunda interação da função, aqui terá um apontamento direto. Ex:

x = 10: 10 → 5 → 16 → 8 → 4 → 2 → 1

```
   0   1  2   3   4   5    6    7    8    9   10  11  12  13  14   15  16
v[-1|-1 | 1 |-2| 2 | -2 | -2 | -2 | 4 | -2 | 16 | -2 | -2 | -2 | -2 | -2 | 8  | ...]
```

Note que: o valor na posição v[5] = -2, ou seja, se mantém como se não tivesse sido acessado, assim sendo v[10] = 16 diretamente, e tratamos essa passagem no programa atualizando o contador em 2 nesses casos.

Para especificar o tamanho máximo de posições do vetor foram feitos os seguintes passos:

a) Determinado o mínimo de memória no computador: 256 MB, para também não ficarmos limitados a números muito pequenos.

b) Depois multplicado 256 por 1024 duas vezes seguidas para tomar a dimensão de mega bytes.

c) Dividido por 8 pois é o tamanho que é comportado pela variável unsigned long int.

E assim chegamos em MAX = 33554432.