ALGORITMOS E ESTRUTURAS DE DADOS III

DOCUMENTAÇÃO TP2

12 de Novembro de 2018

Matrícula: 2015079208
Universidade Federal de Minas Gerais

1 INTRODUÇÃO

O problema consiste em descobrir qual o maior resultado que um jogador pode obter somando ou subtraindo números de uma sequência. Ele também oferece limitantes como: o número, enquanto está sendo calculado, não pode ser negativo nem passar do valor máximo estipulado.

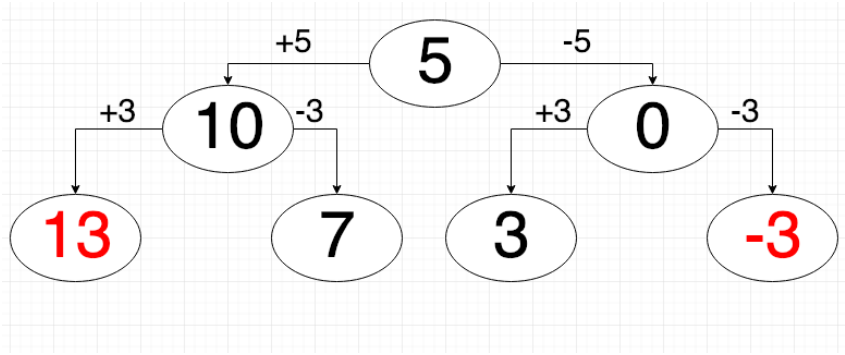
Dado entradas bastante grandes o desafio é implementar um algoritmo capaz de encontrar uma resposta ótima ou acusar que determinado problema é impossível de se resolver.

2 MODELAGEM E SOLUÇÃO DO PROBLEMA

A fim de resolver o problema foram criados dois arquivos que usam dois paradigmas de programção diferentes, o de força bruta que pode ser encontrado no arquivo bruteForce.cpp e o de programação dinâmica que pode ser encontrado no arquivo dynamicProgramming.cpp.

Em alto nível para entender e explorar o problema primeiro o desenhei como uma árvore que contém o nó raiz igual ao valor inicial e que sempre contém dois filhos, o filho da direita é um produto da soma e o da esquerda da subtração, essa árvore pode ser melhor visualizada com o seguinte exemplo:

Valor inicial: 5
Valor máximo que a árvore pode atingir: 10
Valor mínimo que árvore deve atingir para que haja vitória: 10



Os números em vermelho na árvore acima nem chegam a ser consideradas no código real. Foram feitas podas de acordo com a especificação do TP que já foram descritas na sessão anterior.

O programa começa lendo a entrada do arquivo, como especificado na documentação. Aqui eu escolhi guardar a sequência de número S em um vector em detrimento de um array por causa da facilidade de manipulação dos dados. Em seguida o código diverge em duas implementações diferentes.

2.1 Força bruta

```
int computeBruteForce(int value, const vector<int>& sequence, int index,
    int topValue, int minimumValue) {

    if (value > topValue || value < 0) return -1;

    if (index >= sequence.size()) return value;

    int left, right;
    right = computeBruteForce(value + sequence[index], sequence, index+1,
        topValue, minimumValue);
    left = computeBruteForce(value - sequence[index], sequence, index+1,
        topValue, minimumValue);

    return max(left, right);
}
```

Nessa implementação a função computeBruteForce recebe "const vector<int>& sequence"que é um vetor constante passado como referência para que o cálculo de entradas muito grandes sejam calculadas em tempo hábil.

No que tange a implementação real é bastante simples, primeiro o algoritmo faz as podas necessárias com os dois primeiros ifs. Depois chama a função computeBruteForce recursivamente, criando o lado direito da árvore, depois chama novamente a função computeBruteForce dessa vez passando como o que deverá ser computado os valores negativos, criando o lado esquerdo da árvore. O retorno dessa função é o valor máximo entre a direita e esquerda computada daquela chamada recursiva, porque afinal de contas o objetivo do algoritmo é retornar o maior valor possível, mesmo que esse não garanta a vitória.

2.2 Programação dinâmica

```
int computeDynamicProgramming(int value, const vector<int>& sequence, int index,
    int topValue, int minimumValue, int **memoization) {

    if (value > topValue || value < 0) {
        return -1;
    }

    if (memoization[value][index] != -2) {
        return value;
    }

    if (index >= sequence.size()) {
        memoization[value][index] = value;
        return value;
    }

    int left, right;
    right = computeDynamicProgramming(value + sequence[index], sequence,
        index+1,
        topValue, minimumValue, memoization);
    left = computeDynamicProgramming(value - sequence[index], sequence, index+1,
        topValue, minimumValue, memoization);

    memoization[value][index] = max(left, right);
    return max(left, right);
}
```

Para implementar a resolução dinâmica a única coisa que fiz foi adicionar memorização ao algoritmo. Criei uma matriz simples que funciona como dicionário. Os índices value e index funcionam exatamente da mesma forma que as chaves de um dicionário e o que está contido no endereço desses índices é o que pode ser chamado de valor de um dicionário, nesse caso é o valor retornado pela função. A primeira coisa que e função faz quando é chamada é checar se aquele valor e aquela chave já foram computados, se foram, basta usar o resultado que já foi computado e está guardado na matriz ao invés de computá-lo novamente.

Para que esse código funcione foi necessário incializar a matriz com valores inválidos (ou que não são usados pelo programa, no caso -2), para indicar uma matriz vazia. Além disso,

toda vez que uma computação é feita na função também é necessário adicioná-la na matriz de memorização, fora isso, o código funciona exatamente da mesma forma que o de força bruta.

3 ANÁLISE TEÓRICA DO CUSTO ASSINTÓTICO

Para o algoritmo de força bruta a complexidade será sempre O(2^N) sendo N o tamanho da sequência recebida. Claro que a poda faz alguma diferença nessa complexidade, mas nada que seja de grande ajuda.

Para o algoritmo de programação dinâmica acredito que a complexidade seja por volta da dimensão da matriz O(n*m) sendo n o número de valores possíveis para serem guardados na matriz e m o número de índices. Essa estimativa foi feita no meu conhecimento prévio de quanto algoritmos dinâmicos costumam conseguir otimizar de soluções como a apresentada acima.

4 ANÁLISE DE EXPERIMENTOS

A máquina usada tem as seguintes configurações: OS: MacOS High Siera Version 10.13.6 (17G65) Memória: 16 GB 1600 MHz DDR3 Processador: 2,6 GHz Intel Core i7

Todos os testes que usam programação dinâmica rodaram em 0.00s, não consegui medí-los de forma precisa o suficiente. Contudo os testes que usam força bruta tiveram variância no tempo, por exemplo o teste 2 rodou em 1.81s o 3 em 0 e os demais demoraram mais de 1minuto então não esperei para que terminassem de rodar. Esse comportamento era esperado porque o tempo é exponencial.

5 CONCLUSÃO

Os algoritmos funcionaram como esperado, apresentando tempos de processamento condizentes com suas implementações.