Trabalho 3: Programação dinâmica
Caio Cesar Aguiar Alarcon n° 7241109
22 de novembro de 2018

1. Introdução

O trabalho atual tem como propósito a resolução de três problemas com o uso de técnicas de programação dinâmica para que a solução tenha complexidade de tempo polinomial. O primeiro problema consiste em, dada uma sequência de caracteres, encontrar a maior subsequência que seja um palíndromo. O segundo problema trata-se de encontrar o maior subconjunto que seja uma progressão aritmética, a partir de um conjunto dado. Já o terceiro tem a intenção de encontrar o número de possibilidades da soma S de N dados de M lados ser um valor dado.

2. Os problemas

-Primeiro problema:

Primeiramente uma solução recursiva foi implementada onde a função fun(), que recebe como parâmetros uma string ‘s’ e seu tamanho ‘l’ e retorna um vetor inteiro contendo apenas 1 ou 0, onde o 1 representa um particionamento da string representado por um espaço branco no momento da impressão. Por exemplo, se a string é “ABAABAAABA” e a tabela de particionamento vem com {0,0,0,0,1,0,0,1,0,1} isso representa: “ABAAB AAA BA ”. Isso foi feito desta forma pois facilita a concatenação.

Esta função verifica (com o uso de uma função auxiliar chamada pal) se a string recebida trata-se de um palíndromo, caso afirmativo, retorna uma tabela de particionamento referente ao palíndromo encontrado. Por exemplo, se receber a string “ANA” ela retorna {0,0,1}. Se receber “AA” retorna {0,1}; Se for “B”; {1}.

Caso contrário, divide a string recebida em duas partes de todas as maneiras diferentes e efetua uma chamada recursiva para cada parte obtida. As respostas que fornecem somadas o menor número de partições são reservadas. Ao término do processo, a solução encontrada é a que minimiza o número de palavras, então a tabela de particionamento da primeira divisão é concatenada à tabela da segunda e é o que a função retorna. Na função principal um laço imprime a string e utiliza a tabela de particionamento para inserir os espaços em branco que dividem a string principal em palíndromos.

Posteriormente a parte dinâmica foi implementada utilizando uma tabela de hash, onde resultados previamente calculados são armazenados para uso em posteriores chamadas. Não foi usado tratamento de colisões pois a tabela criada era suficientemente grande. Com o uso da programação dinâmica a função, depois de verificar se a string recebida é o caso base (palíndromo), verifica se a solução já foi encontrada alguma vez buscando na tabela de hash. Caso afirmativo ela apenas retorna diretamente a solução. Caso contrário ela busca recursivamente por uma solução.

-Segundo problema:

Em primeiro lugar, o tipo de dado utilizado foi um vetor onde o primeiro elemento v[0] ou *v representa o número de elementos contidos no vetor. Este tipo de dado é chamado neste trabalho por 'set'. O primeiro procedimento é ordenar o set utilizando o algoritmo Bolha, pois o tamanho dos sets que o programa recebe não são muito grandes.

Depois disso a função MaiorPA é chamada. Esta função percorre com um laço do segundo até o penúltimo elemento do set considerando-o o elemento médio de uma PA de três elementos. Os outros dois começam sendo os elementos adjacentes ao do meio e, conforme a necessidade, vão se afastando. Se uma trinca testada for uma PA, chama a função MaxSet (explicada posteriormente) e guarda o maior resultado encontrado das diversas vezes que chamar MaxSet. Caso contrário, ela avalia qual índice precisa ser incrementado, incrementa e o ciclo recomeça. Até que algum índice exceda o tamanho do set, quando isso ocorrer um próximo elemento é testado. Por exemplo, se a entrada for {1, 2, 7, 12, 111}. O primeiro elemento testado é 2 como sendo o elemento médio. O 1 e o 7 são os elementos adjacentes. Como {1,2,7} não é uma PA, ou seja, 1+7 != 2*2, é necessário substituir algum elemento das pontas. Isso é feito comparando o dobro do elemento atual com a soma dos outros dois elementos. Se a soma for menor do que o dobro do elemento testado, o elemento da direita é substituído pelo próximo, caso contrário é o da esquerda que é substituído. No caso do exemplo, 1+7 > 2*2, então o elemento da esquerda que será substituído. Porém ele é o primeiro, seu índice (i) agora será negativo (-1), condição de quebra do while. Assim sendo, o laço mais externo é acionado e o elemento do meio será alterado, antes era o 2, agora será o 7. Os adjacentes serão o 2 e o 12. {2,7,12} forma uma PA pois 2+12 = 2*7. Nesse caso a função MaxSet é chamada e o resultado, caso seja o maior, será guardado. Depois disso os dois elementos da ponta são substituídos pelo próximo, ou seja, a trinca agora é o {1,7,111}, que também não forma uma PA, e o ciclo se repete até que o 12 seja testado e algum elemento da ponta saia do intervalo compreendido. Quando isso acontecer, a função simplesmente retorna a maior resposta que tiver encontrado nas diversas vezes que chamou MaxSet.

Dados três números, A função MaxSet retorna o maior set de números que formam uma PA com os números já encontrados com os números fornecidos. Por exemplo, caso MaxSet receba, numa primeira chamada {1,2,3}, retornará {3,1,2,3} (Lembrando que o primeiro elemento corresponde ao número de elementos contidos). Caso, numa segunda chamada, receba {2,3,4}, retornará {4,1,2,3,4}. Se a segunda chamada for {4,5,6}, retornará {6,1,2,3,4,5,6} e assim por diante.

Na função MaxSet, o primeiro processo que acontece é a inicialização da Tabela, acontece apenas na primeira vez que a função é chamada e consiste em preencher com zeros todo o conteúdo da Tabela. Depois disso, uma busca na tabela revela se a trinca forncida pertence a alguma PA já encontrada antes. Caso afirmativo, a função apenas retorna o set correspondente. Caso exista uma PA correspondente, mas alguns elementos da trinca não pertencem a ela, eles são introduzidos e depois disso ela retorna o set. Se a PA não for encontrada, a trinca é inserida no local correspondente e a função retorna apenas a trinca inserida.

Desta forma, ao percorrer todas as trincas que formam uma PA, inevitavelmente a maior trinca será retornada pela função MaxSet, ficará retida na função MaiorPA e, ao final do todo o processo, será retornada para a função principal que imprimirá o set retornado com a função PrintSet.

-Terceiro problema:

Considerando que a resposta deste problema poderia ser um número muito grande que, eventualmente, não caberia em um inteiro de trinta e dois bits, o tipo de dado escolhido foi unsigned long long int e a ele dado o nome de ‘nat’, por ser um número natural. Para que o programa funcione é necessário que a máquina testada e o compilador estejam preparados para suportar tecnologia de pelo menos sessenta e quatro bits.

A primeira parte na solução do problema foi relacionada à pesquisa sobre a matemática envolvida no problema. A solução encontrada consiste na seguinte fórmula:

$$c = \sum_{k=0}^{\lfloor (p-n)/s \rfloor} (-1)^k \binom{n}{k} \binom{p - s\,k - 1}{n-1},$$

A partir disso, o problema consiste em um laço tipo for que percorre de k=0 até [(p-n)/s] onde [x] representa a função ‘floor’, ou seja, trunca o resultado eliminando a parte decimal. Como a linguagem C já faz isso por padrão sua implementação não é necessária.

Além disso a função ‘bin’ precisou ser implementada. Tal função tem por objetivo calcular o coeficiente binomial. Isso foi feito primeiramente utilizando recursão pura, posteriormente utilizando programação dinâmica, onde um cache foi inserido na função buscando otimizar os resultados registrando em uma matriz os coeficientes já encontrados.

3. Resultados

Antes de aplicar os conceitos da programação dinâmica, apenas os primeiros resultados eram exibidos pelo programa rapidamente, os últimos demoravam tanto que não valeria a pena o tempo gasto.

4. Conclusão

Sem dúvida o uso da programação dinâmica reduziu a complexidade dos programas fazendo-os passar de problemas intratáveis para problemas de complexidade polinomial.