UNIVERSIDADE FEDERAL DE MINAS GERAIS

INSTITUTO DE CIÊNCIAS EXATAS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

BRUNO ANTONIO VIEIRA

TRABALHO PRÁTICO 3 DE AEDS 3

BELO HORIZONTE

2017

SUMÁRIO

1 INTRODUÇÃO

O problema apresentado nesse trabalho prático se baseia em uma rua que possui bares e casas em ambos os lados, cada casa em um lado da rua está associada a um bar do outro lado dessa mesma rua. O objetivo da solução é encontrar a maior quantidade possível de ligações entre uma casa e um bar, por meio de linhas de bandeirolas, sem que essas linhas se cruzem.

Para isso, são apresentadas três soluções de resolução: resolução por força bruta, resolução por algoritmo guloso e resolução por programação dinâmica. Na resolução por força bruta, todas as 2^n possibilidades de ligação entre casas e bares são testadas e a que tiver o maior número de ligações e não possuir cruzamentos é retornada como resposta. Na resolução por algoritmo guloso… Na resolução por programação dinâmica um lado da rua é tomado como referência, nesse caso o lado que possui as casas ou bares com número ímpar, e nesse vetor de números ímpares deve-se encontrar a maior subsequência crescente possível, esse valor é então retornado como resposta.

2 SOLUÇÃO DO PROBLEMA

- Solução por força bruta:

  Na solução por força bruta é criado um vetor de uma estrutura que representa a rua:

```
estrutura rua{
        int lado esquerdo da rua;
        int lado direito da rua;
}
```

  No lado esquerdo da rua são armazenados as casas ou bares de número par e no lado direito as de número ímpar. Esse vetor então é ordenado, com base no lado esquerdo, utilizando o algoritmo quicksort. A partir disso, toma-se o lado direito como referência para os próximos passos da resolução.

  Um loop é feito de i=1 até 2^N-1, onde N é o número de casas ou bares em um lado da rua. Para cada iteração do loop, é chamada a função converte_bin() que transforma o valor de i em um vetor de números binários de tamanho N. Em uma entrada de tamanho 10, por exemplo, quando i = 200, a função retorna o seguinte vetor:

0	0	1	1	0	0	1	0	0	0

  Para cada posição desse vetor verifica-se se o valor existente é igual a 1, o que significa que será feita uma ligação entre a casa e o bar que estão nessa posição. Se o número do bar/casa for maior do que o

testado anteriormente (o maior valor é iniciado com zero), uma ligação pode ser feita com sucesso e o resultado parcial é incrementado em 1, caso contrário quebra-se o loop, pois existe cruzamento de linha nessa possibilidade testada. No fim, o resultado parcial de cada teste é armazenado na variável solution (inicializada com zero) caso a parcial seja maior do que o que já esteja armazenado em solution anteriormente. Com isso, a solução final é encontrada.

Exemplo com N = 3 e entradas: (1 2), (5 8), (9 4)

2	8	4

0	0	1	Não há cruzamento. Parcial = 1
0	1	0	Não há cruzamento. Parcial = 1
0	1	1	Há cruzamento. 4 < 8
1	0	0	Não há cruzamento. Parcial = 1
1	0	1	Não há cruzamento. Parcial = 2
1	1	0	Não há cruzamento. Parcial = 2
1	1	1	Há cruzamento. 4 < 8

Como a maior parcial encontrada foi 2, a solução do problema é 2.

- Solução por algoritmo guloso:

  Nesse método, o algoritmo tenta buscar uma solução ótima e muitas vezes consegue com sucesso. Assim como no método anterior, um vetor de uma estrutura é criado e ordenado utilizando quicksort. Em seguida, um vetor auxiliar de tamanho N, sendo N o número de casas/bares dado na entrada, é criado para auxiliar na contagem de “cruzamentos” entre as linhas de bandeirolas de um bar e outro.

  A seguir, um loop de i=0 até N-1 é executado, e aninhado a este é executado outro loop de j=i+1 até N-1. O objetivo desse trecho é verificar se há cruzamento entre um valor e todos os seus valores seguintes. Caso o valor na posição j seja menor do que aquele na posição i, significa que houve um cruzamento entre essas duas casas/bares, logo, o vetor auxiliado na posição i e na posição j é incrementado em 1. Se nenhum valor de j entrar nessa condição, significa que a posição de i não faz nenhum cruzamento e , portanto, está no conjunto solução. Após o incremento, procura-se a posição que causa o maior número de cruzamentos, pois isso é necessário para tomar a decisão gulosa a cada passo do algoritmo.

  for *i=0 até N-1{*

```
        for j=i+1 até N-1{
            se rua[j] for menor que rua[i]{
                aux[i] e aux[j] são incrementados em 1
                se aux[i] for maior que maior{
                    maior = aux[i]
                    imaior = i
                }
                se  aux[j] for maior que maior{
                    maior = aux[j]
                    imaior = j
                }
            }
        }
        se aux[i] continuar sendo igual a zero
            solução++
    }
```

Feito isso, enquanto o maior número de cruzamentos for maior que zero (verificado por um loop while), no vetor auxiliar é escolhido de forma gulosa a posição que causou mais cruzamentos. Para todas as posições anteriores à escolha gulosa, se o valor da posição for maior que a escolha gulosa e diferente de zero, ele é decrementado em 1. Para todas as posições posteriores à escolha gulosa, se o valor da posição for menor que a escolha gulosa e diferente de zero, ele é decrementado em 1. Feito isso, a escolha gulosa é retirada do processo, marcando-a com o valor zero. Para ambos os casos anteriores, se houver um decréscimo e o valor da posição passar a ser zero, significa que aquela posição não cruza com nenhuma outra e, portanto, faz parte do conjunto solução. É feita então uma busca pelo novo maior valor para ser tomado como escolha gulosa e prosseguir com a solução até o fim.

Exemplo com N = 3 e entradas: (1 2), (5 8), (9 4)

2	8	4
0	0	0

Todas as posições do vetor aux são inicializadas com zero (não há cruzamento ainda)

2	8	4
0	1	1

a posição zero do vetor, que contém 2, não faz cruzamento, desde o início, logo, já está dentro do conjunto solução. 8 é maior que 4, então houve cruzamento entre os dois. **SOLUÇÃO PARCIAL= 1**

2	8	4
0	0	0

1 é o maior valor e portanto é escolhido gananciosamente (na mesma posição onde se

encontra o número 8). como 4 é menor que 8, o valor presente no vetor auxiliar é decrementado e se torna 1, significando que aquela posição não faz mais nenhum cruzamento e, portanto, está no conjunto solução. **SOLUÇÃO = 2**

- Solução por programação dinâmica:

  Na solução dinâmica também é criado um vetor de uma estrutura como na solução por força bruta e esse vetor é ordenado com base nos valores da esquerda da rua. Após isso, é criado um vetor auxiliar de tamanho N, que armazena os resultados dos sub-problemas do problema geral. O objetivo nessa parte é encontrar a maior subsequência crescente dentre os números das casas/bares presentes do lado direito da rua. Para isso, o vetor auxiliar é inicialmente preenchido com “uns” em todas as suas posições. A partir daí, para i=1 até N-1 são testadas todas as subsequências de j=0 até i. Se o número presente no vetor na posição j for menor do que o presente na posição i, significa que essa subsequência em aux[i] tem no mínimo o tamanho do valor presente em aux[j] + 1, mas, se o valor de aux[i] for maior do que o de aux[j] + 1, o valor não deve ser atualizado. Quando j alcança i, j volta a posição inicial j=0 e i é incrementado em 1. O processo se repete até i chegar à última posição i=N-1. A seguir, está o pseudocódigo que descreve esse algoritmo:

```
for i=1 até N-1{
        for j=0 até i{
                se rua[j] for menor que rua[i]{
                        aux[i] = max(aux[i], aux[j]+1)
                }
        }
}
```

  Feito esse procedimento, o vetor aux possui agora o tamanho da maior subsequência crescente armazenada em alguma das posições, basta realizar uma busca nesse vetor para encontrar o maior valor e encontrar a solução para o problema geral.

  Exemplo com N = 3 e entradas: (1 2), (5 8), (9 4)

j	**i**	
2	8	4
1	1	1

  vetor[j] é menor que vetor[i], então aux[i] = max(aux[i], aux[j]+1). Como j já alcançou i, j volta a ser 0 e i avança uma posição.

j		**i**

2	8	4
1	2	1

vetor[j] é menor que vetor[i], então aux[i] = max(aux[i], aux[j]+1). j é incrementado em uma posição.

	j	**i**
2	8	4
1	2	2

vetor[j] não é menor que vetor[i], então nada é feito. Como j já alcançou i e i está na última posição possível o algoritmo termina.

A resposta é o maior valor presente no vetor aux, no caso, 2.

3 ANÁLISE DE COMPLEXIDADE

- Função cmp():

  A complexidade de tempo e espaço é O(1), pois o tamanho da entrada é sempre fixo. A função só serve como auxílio de comparação para o quicksort.

- Função converte_bin():

  A complexidade de tempo do algoritmo é O(n), onde n é o tamanho do vetor que será passado como argumento à função. Após a conversão de decimal para binário que consiste em divisões sucessivas por 2, o vetor de tamanho n recebe n inserções.

  A complexidade de espaço também é O(n), já que são alocados n espaços para compor um vetor que armazena os bits da string binária.

- Função max():

  A complexidade de tempo e espaço dessa função é O(1), pois o tamanho da entrada é sempre fixo (2 entradas).

- Solução força bruta:

  A complexidade temporal dessa solução é $O(n2^n)$, pois, no pior caso, um vetor de tamanho n é completamente percorrido para avaliar a possibilidade de ligação entre casas/bares e isso é repetido dentro de um loop que vai de i=1 até 2^n para testar todas as possibilidades.

  A complexidade espacial é O(n) pois é feita a alocação de dois vetores de tamanho n.

- Solução algoritmo guloso:

  A complexidade temporal dessa solução é $O(n^3)$, devido aos dois loops aninhados do algoritmo e ao fato do pior caso do loop while apenas retirar a cada passada a maior escolha gulosa e isso não interferir nas outras escolhas, apenas diminuir um valor em cada.

  A complexidade espacial é O(n) por conta das duas alocações de vetores de tamanho realizadas.

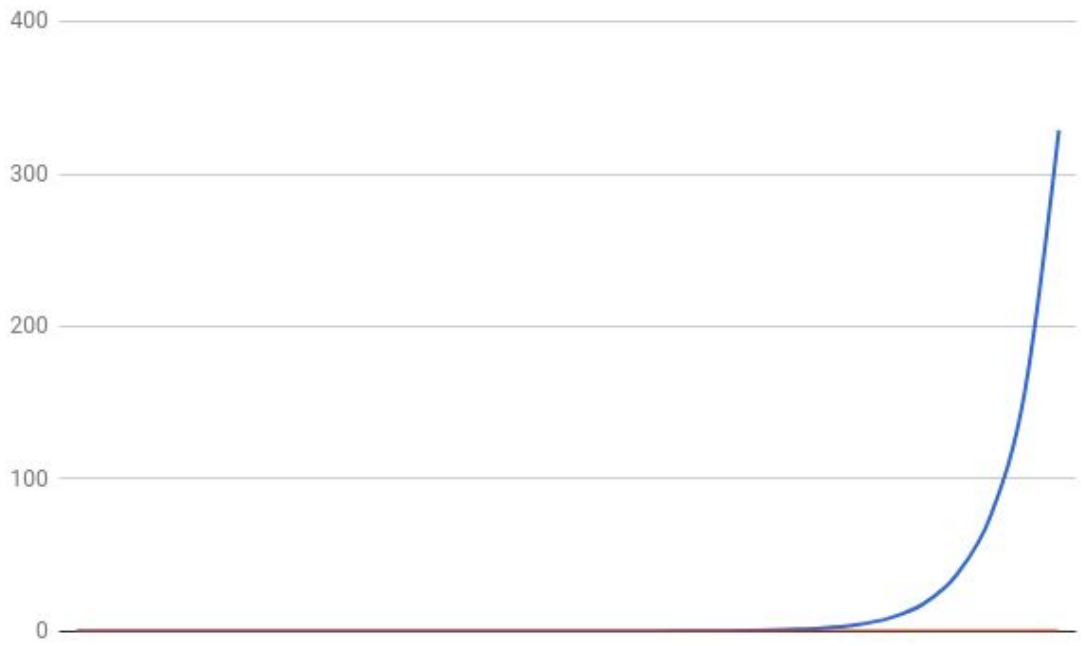
- Solução programação dinâmica:

  A complexidade temporal dessa solução é $O(n^2)$ onde n é o tamanho da entrada que forma o tamanho do vetor de casas/bares, pois existem dois loops aninhados, o mais interno percorre o vetor de j=0 até i, e o mais externo de i=1 até n-1.

  A complexidade espacial é O(n), por conta da alocação de dois vetores de tamanho n.

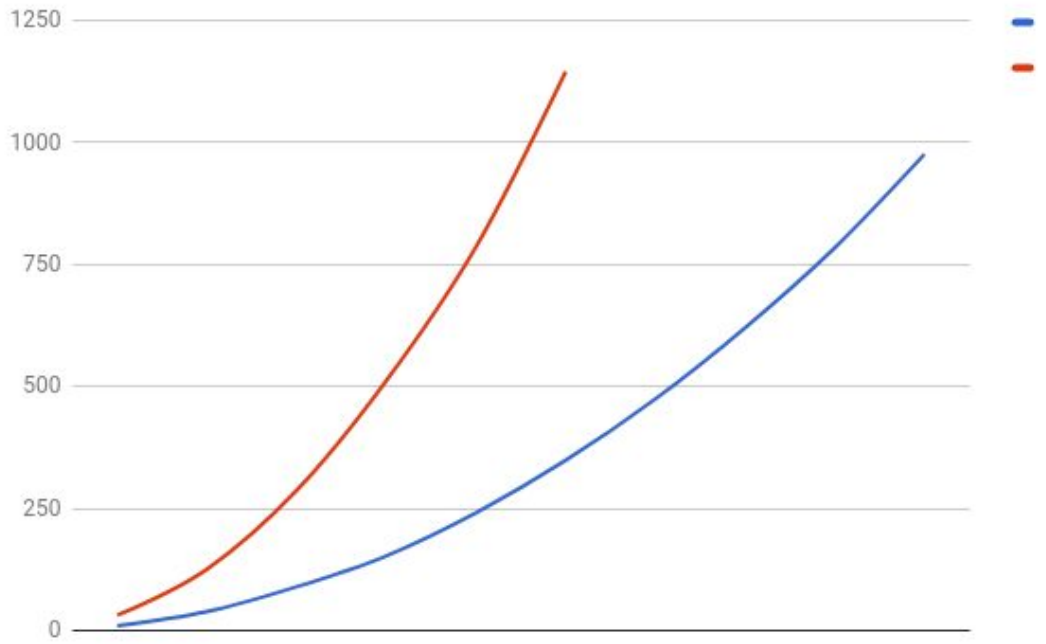
4 TESTES

Os testes foram realizados no sistema operacional Ubuntu 17.04 rodando num processador Intel Core i3 de 3.10 Ghz e 6 GB de memória RAM, utilizando o comando time do UNIX no terminal para avaliação do tempo de execução dos testes.



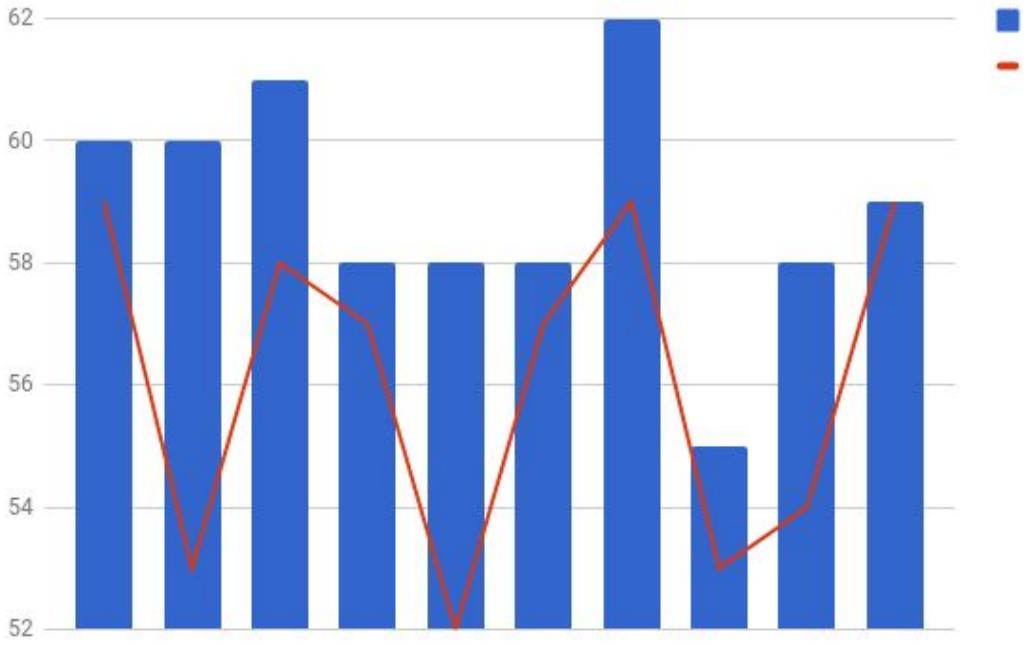
No gráfico acima (entrada x tempo em segundos) a linha azul representa o tempo de execução do algoritmo força bruta e a linha vermelha o tempo de

execução do algoritmo dinâmico. As entradas foram aleatórias e variaram de tamanho 1 até 30. É possível notar, claramente, a desvantagem da solução força bruta para entradas maiores pelo formato exponencial de sua curva, o que já era esperado por conta de sua complexidade $O(n2^n)$.



No gráfico acima (entrada x tempo em segundos), a linha azul representa o algoritmo dinâmico e a linha vermelha o algoritmo guloso. As entradas foram aleatórias e variaram de 50.000 até 300.000 para o algoritmo guloso e de 50.000 até 500.000 para o algoritmo dinâmico. A solução dinâmica, de fato, se sai melhor devido a sua complexidade n vezes menor do que a solução gulosa, pode-se ver isso pela diferença de inclinação entre as duas curvas.

O gráfico acima mostra a otimidade do algoritmo guloso. 0 significa que o algoritmo não entregou a solução ótima e 1 significa que a solução ótima foi entregue. A linha azul representa os testes feitos com entradas pequenas de tamanho 10, onde o algoritmo se saiu bem acertando 9 de 10 testes. Já a linha vermelha representa os testes feitos com entradas de tamanho 1000 e percebeu-se que o algoritmo não se sai bem em entradas muito grandes. Conclui-se então que para entradas grandes a solução ótima é dificilmente alcançada, pois o algoritmo acertou apenas 1 de 10 testes, porém a margem de erro é baixa como pode ser percebido no próximo gráfico:



As barras azuis representam a solução ótima e a linha vermelha representa os resultados entregues pelo algoritmo guloso.