

## Trabalho Prático III

# Soluções em algoritmos força bruta, guloso e programação dinâmica

Algoritmos e Estruturas de Dados III

Tatiana Santos Camelo de Araujo – 2015086298

### 1. Introdução

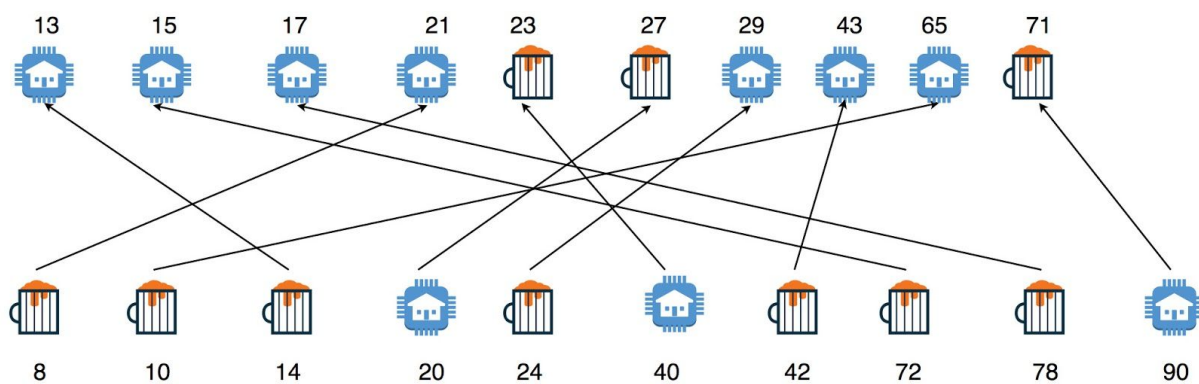
Em uma rua com bares e residências, os moradores querem pendurar bandeiras que vão de um bar até a residência do dono do bar. Queremos que tenham o maior número de bandeiras penduradas mas sem se cruzarem.

A entrada do problema segue o padrão abaixo:

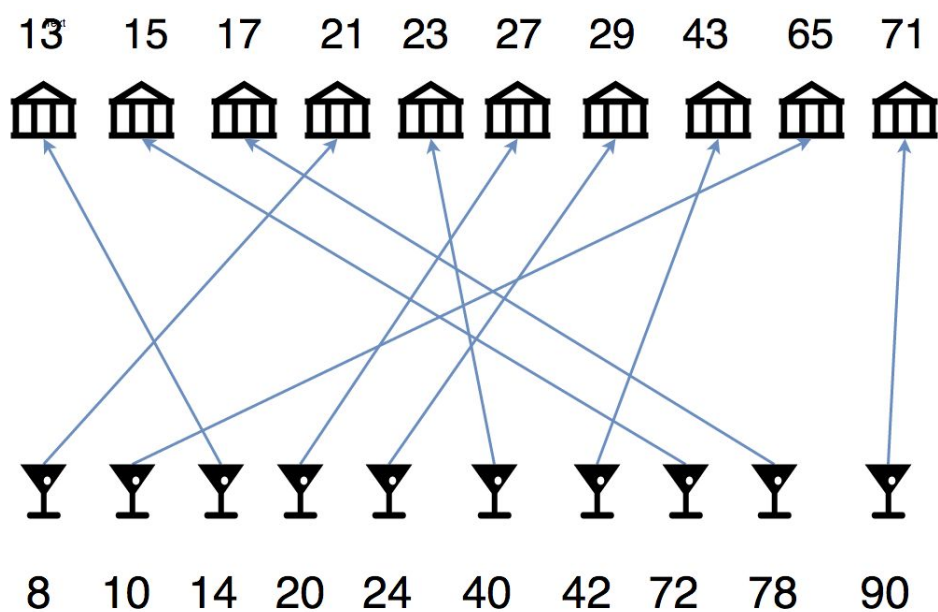
```
d/g/b    // aqui é a escolha do algoritmo usado, dinâmico, guloso ou força bruta
10       // o valor seguinte determina quando bares existem
8 21     // todos os pares de valores seguintes são a relação de bar e casa
10 65
14 13
23 40
24 29
27 20
42 43
71 90
72 15
78 17
```

Todos números pares ficam no mesmo lado da rua, podendo ser bar ou casa. Para facilitar o entendimento e os algoritmos, foi considerado que todo número par corresponde a um bar e todo número ímpar a uma casa. Como todo par (bar-casa) tem um número ímpar e um par, e a bandeira pendurada vai de um a outro, não há nenhuma alteração significativa no problema ou no algoritmo ao usar essa condição.

Original da entrada:

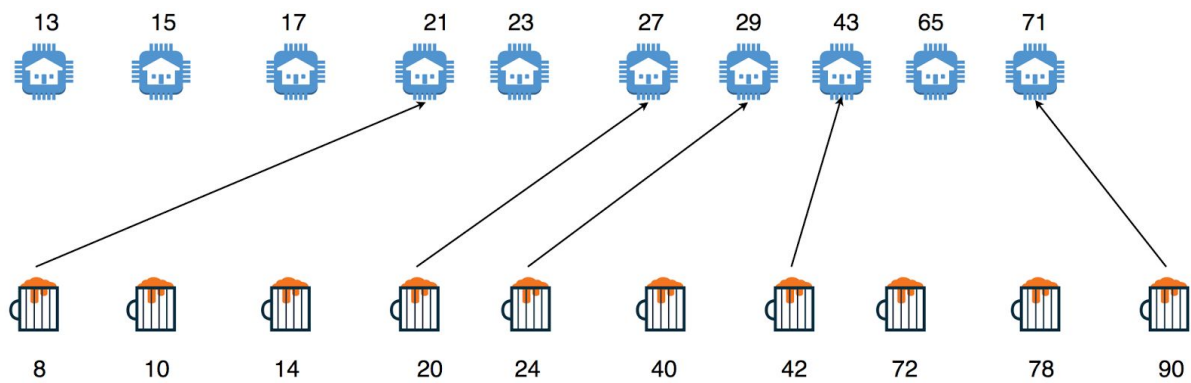


Consideração nessa documentação e no algoritmo:



Os copos representam os bares e as casas as residências. Os números são seus respectivos números na rua. Todos valores pares estão embaixo e representam os bares, o oposto para as residências. As setas azuis representam um exemplo da relação dos pares (bar, casa) com algumas colisões.




Neste exemplo, a solução é 5. A solução são o número máximo de bandeiras sem colisão e não pede o bar. Note que a solução abaixo poderia ter o par 14-13 em vez do 8-21, que a solução seria ainda a melhor, embora com pares diferentes.



## 2. Solução do problema

A solução deste problema é dada em três algoritmos diferentes, força bruta, guloso e programação dinâmica.

A modelagem é feita por meio de um vetor de pares (bar, casa) e que em cada relação tem um contador de colisões. Isso foi feito por meio de um struct. A representação do vetor com estas três informações segue assim:

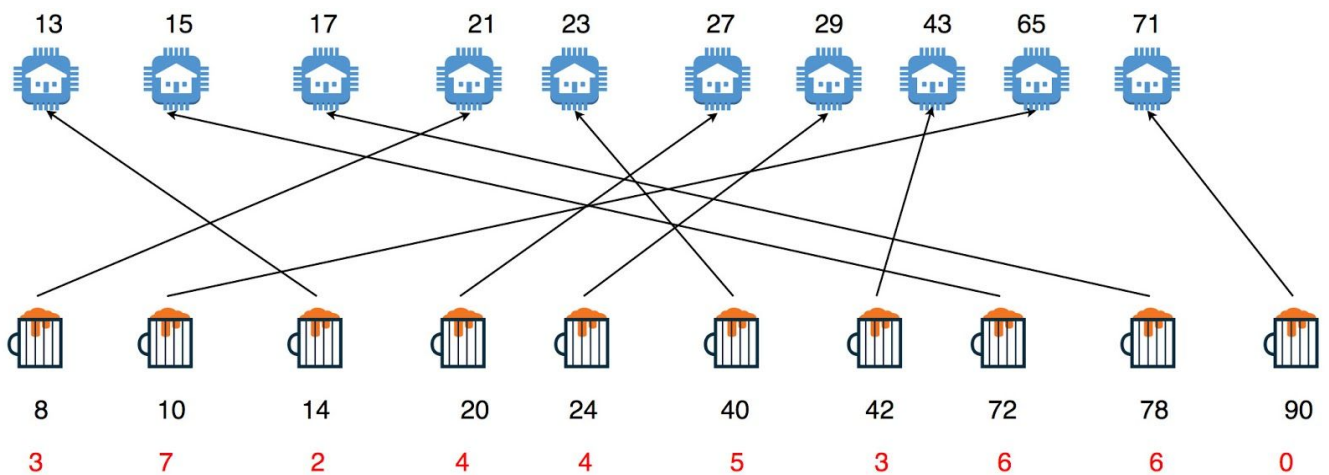
| Bar   | Casa  | Colisão   |
|---|---|---|
|  |  |  |

Bar 8      Casa 21      Colisão 3

Bar 10      Casa 65      Colisão 7

Bar 14      Casa 13      Colisão 2

Bar 20      Casa 27      Colisão 4



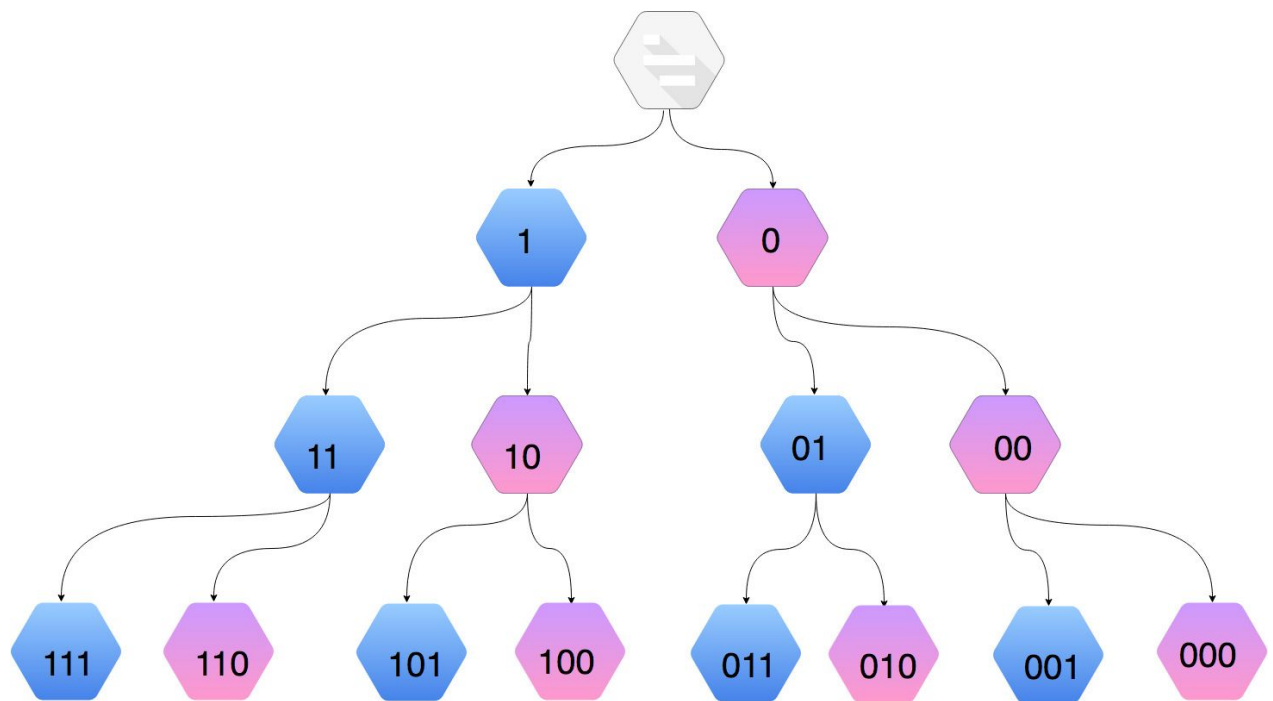
A colisão é considerada quando dada uma bandeira, 24-29, por exemplo é cortada por uma outra. Essa bandeira tem o total de 4 colisões. Para isso acontecer, deve haver uma bandeira que tenha ou:

- número do bar **menor** e número de residência **maior** do que 24-29  
exemplo: bandeira (10-65)
- número do bar **maior** e número de residência **menor** do que 24-29  
exemplo: bandeiras (72-15), (40-23), (78-17).

Qualquer outra opção – sabendo que não há duas bandeiras saindo do mesmo par (bar, casa) – não configura colisão. Vemos que no exemplo acima a bandeira 24-29 tem, de fato, quatro opções de bandeiras que colidem.

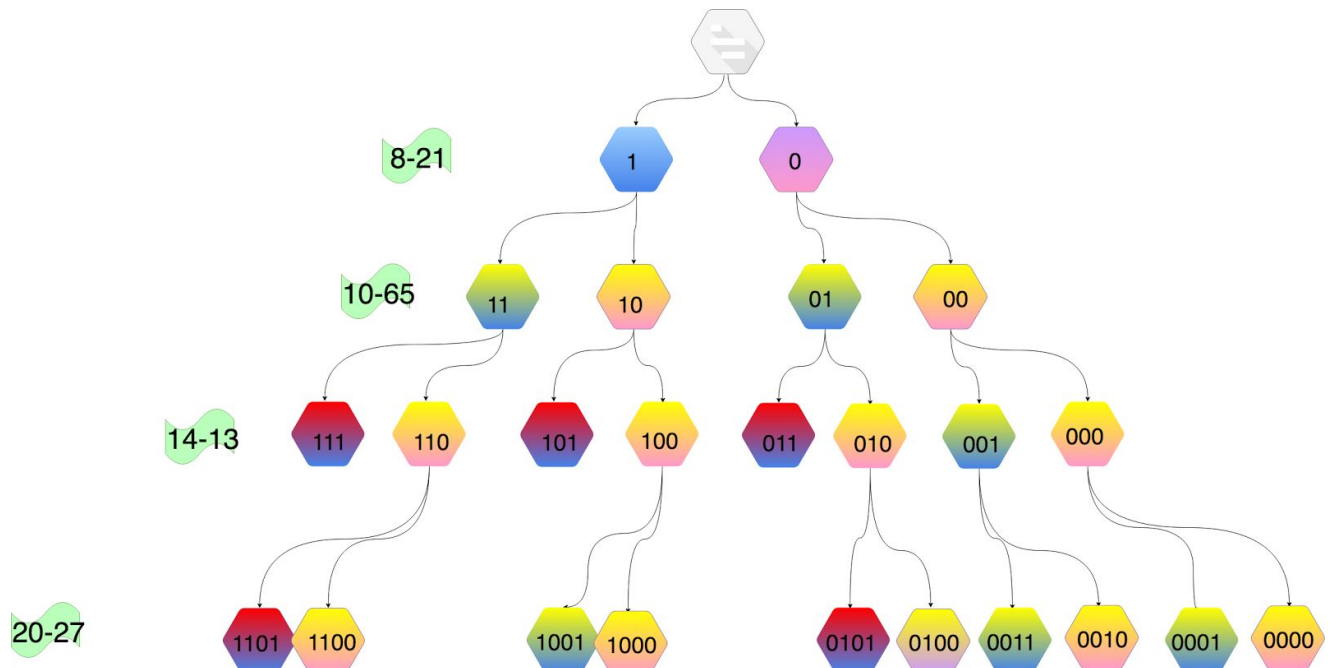
### Algoritmo Força Bruta

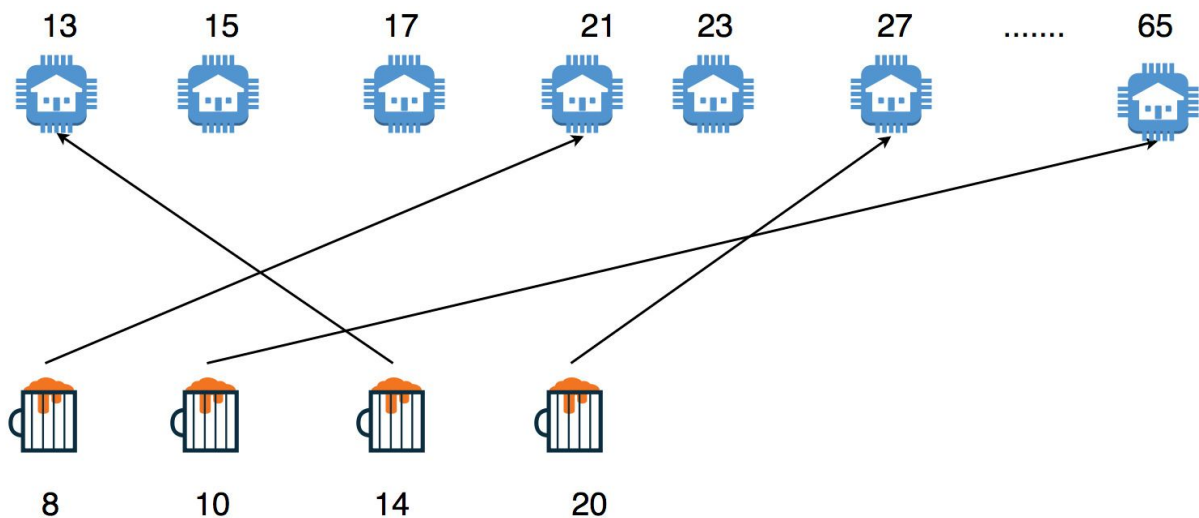
A solução em força bruta percorre todas as opções de caminhamento em busca da solução. Para isso, foi implantado uma árvore, já que a cada bandeira temos duas opções: pendurá-la ou não. À esquerda, o nodo que representa a bandeira colocada e a direita a bandeira não foi colocada.



Acima uma representação da árvore. Os nodos azuis são os nodos em que foram colocadas bandeiras e os rosas onde não foram. Se seguirmos os nodos 1 - 11 - 111, significa que colocamos todas bandeiras até a situação atual. Os valores com 0 representam onde não temos bandeiras. No nodo 010, não foi colocada nem a primeira nem a terceira bandeira, mas a segunda foi pendurada.

Isso garante que procuremos todas soluções possíveis pro problema, testando todas soluções. A cada novo nodo criado, é testado se houve colisão com as sequências de bandeiras implementadas. Se houve, este novo vira uma folha e para de gerar nodos.









Esta árvore é uma representação da entrada até a quarta bandeira. Os nodos com apenas um número 1 nunca serão conflitantes, pois só há uma bandeira colocada mas provavelmente não serão nosso melhor caso. Os nodos vermelhos em cima representam conflito e não geram novas soluções, já que queremos otimizar o número de bandeiras sem colisão. Os nodos amarelos em cima são os que seguem sem conflito e vão gerando novos pares.

Em entradas relativamente grandes, como 100 bandeiras, já não é um algoritmo de bom funcionamento.

## Algoritmo Guloso

A estratégia do algoritmo guloso é a cada interação, buscar a melhor opção. O algoritmo soma todas as colisões de cada par (bar, casa) e coloca a informação no vetor e o ordena de ordem crescente, então a última posição  $\text{vetor}[\text{tamanho}-1]$  terá o par bar-casa com maior colisão.

Ele empilha os bares que já tiveram as bandeiras penduradas e testa se há colisão entre a próxima bandeira do vetor e a pilha. A pilha é usada como um marcador, setada como 0 e recebe o valor 1 na posição correspondente à o par empilhado. Se não houver colisão, este novo valor entra na pilha e o número de bandeiras penduradas incrementa.

|  |  |  |  |
|---|---|---|---|
| 10  | 65  | 7   | 0   |
| 72  | 15  | 6   | 0   |
| 78  | 17  | 6   | 0   |
| 40  | 23  | 5   | 1   |
| ...   |   |   |   |

Bar, casa e colisões ordenados. O ícone vermelho e verde representa a pilha, que vai ser atualizada caso não haja colisão entre o par testado e os pares empilhados.

## Programação dinâmica

O algoritmo dinâmico cria um vetor que vai salvar a maior subsequência em um vetor auxiliar. Ordenando meus pares bar-casa em função de bares, só é necessário testar o vetor de número das casas. A maior sequência crescente dentro deste vetor será o maior número possível de bandeiras que podem ser colocadas sem se cruzarem. Na última posição do vetor, estará a maior subsequência existente no sistema.

|                     |    |    |    |    |    |    |    |    |    |  |
|---------------------|----|----|----|----|----|----|----|----|----|--|
| <b>bares</b>        |    |    |    |    |    |    |    |    |    |  |
| 0                   | 10 | 14 | 20 | 24 | 40 | 42 | 72 | 78 | 90 |  |
| <b>casas</b>        |    |    |    |    |    |    |    |    |    |  |
| 21                  | 65 | 13 | 40 | 29 | 20 | 43 | 90 | 15 | 17 |  |
| 1                   | 2  | 1  | 2  | 3  | 2  | 4  | 2  | 3  | 5  |  |
| <b>subsequencia</b> |    |    |    |    |    |    |    |    |    |  |

**Equação de recorrência:**

$$R_0(\text{maior Sequencia}) = 0$$

$$R_i(\text{maior Sequência}) = \max_{0 < j < i} T_j + 1$$

$$(Soma_j < Soma_i)$$

### 3. Análise de complexidade

Na análise de resultados dissertem sobre o compromisso entre complexidade assintótica e otimalidade da solução.

| Função      | Funcionamento   | Complexidade |
|-------------|---|--------------|
| Força bruta | Testa todas opções de combinação de <b>n</b> bandeiras em árvore. | $O(n^2)$     |

|          |   |          |
|----------|---|----------|
| Guloso   | Testa melhor opção e empilha o valor. Verifica se o próximo valor do vetor tem conflito com algum valor da pilha. | $O(n^2)$ |
| Dinâmico | Ordena o vetor de par bar-casa, acha a maior sequência crescente em relação ao número de casas.                   | $O(n^2)$ |

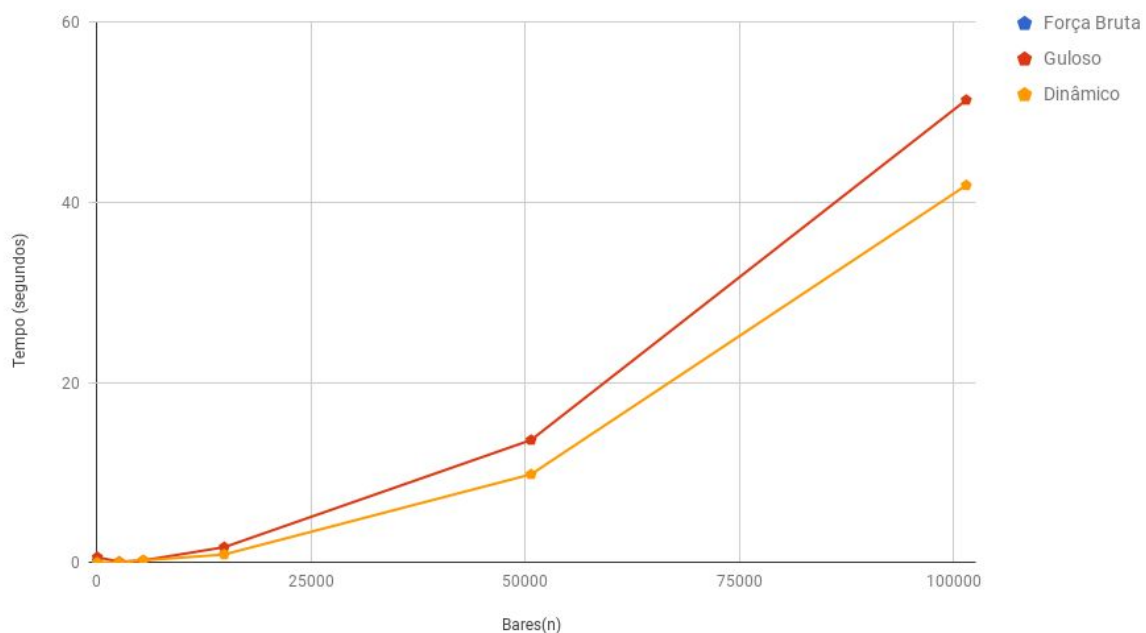
Todas complexidades estão em função de entradas,  $n$ , ou seja, o número de pares bar-casa.

Força bruta sem essa complexidade pois abre toda a árvore de possibilidade, em que cada nodo abre mais dois. O guloso reordena o vetor e o dinâmico tem essa complexidade por percorrer o vetor de casas completamente e ordenar o vetor de bares.

#### 4. Avaliação experimental

|   | A           | B     | C    | D    | E     | F     | G      |
|---|-------------|-------|------|------|-------|-------|--------|
| 1 | bares(n)    | 50    | 2600 | 5419 | 14857 | 50668 | 101447 |
| 2 | força bruta | 0.178 | -    | -    | -     | -     | -      |
| 3 | guloso      | 0.6   | 0.09 | 0.24 | 1.71  | 13.61 | 51.36  |
| 4 | dinâmico    | 0.02  | 0.03 | 0.24 | 0.9   | 9.8   | 41.89  |
| 5 |             |       |      |      |       |       |        |

Tamanho de Entrada x Tempo



Embora os tempos dos algoritmos guloso e dinâmico não sejam muito diferentes em tempo e complexidade, a otimização da programação dinâmica é incomparável. O algoritmo guloso para casos muito grandes gerava resultado errado, embora relativamente perto do



correto. Além disso, a implementação dinâmica foi a que teve a programação mais rápida, após observado uma implementação boa.

O força bruta rodou até o toy\_8, de entrada 50, e com entrada 100, não foi possível sua finalização.