UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

Gabriel Simmel Nascimento
Rafael Augusto Monteiro

Relatório UVA 10003 - Cutting Sticks

1. O Programa

1.1. Instruções de Execução

Utilize o comando python com o nome do programa para executá-lo. O programa pode receber diferentes entradas. Os casos de teste descritos no pdf estão no arquivo “1.in”. Para executar o programa com o caso de teste citado acima, utilize:

```
python t2.py < 1.in
```

O código foi feito em Python 3. Caso Python 3 não seja a versão Python padrão do seu sistema, utilize o comando adequado

1.2. Visão Geral do Programa

Um pseudo algoritmo que descreve o funcionamento do programa seria:

```
1.  Lê um comprimento L:
   1.1.  Caso L == 0, finaliza a execução
2.  Lê um número N, que descreve o número de cortes
3.  Lê N números, que descrevem as posições onde os
    cortes serão feitos.
4.  Calcula custo mínimo de cortes
5.  Imprime qual é o valor mínimo encontrado
6.  Retorna ao passo 1.
```

Todas as leituras são feitas via *stdin*, e todas as escritas são feitas via *stdout*..

1.3. Regra de recorrência

O problema é: dado um conjunto de *n* cortes necessários num bastão de tamanho *L*, dê o custo mínimo para realizar os cortes. O custo de um corte de uma posição *ai* até uma posição *aj* é dado por *cost(ai,aj) = aj - ai*.

Considere uma função *C(ai,aj)*, que dá o custo mínimo para cortar um bastão, dadas as possibilidades de corte de uma posição *ai* até uma posição *aj*. Então, o problema torna-se calcular *C(a1,an)*, onde *a1* = 0, e a posição final *an* = *L.*

O problema *C(ai,aj)* pode ser dividido na soma de dois subproblemas. Ao cortar um bastão ao meio, o seu custo será mínimo caso o custo do corte dos bastões restantes do corte também sejam mínimos(no caso, os custos de corte do bastão do lado direito e do lado esquerdo devem ser mínimos). Portanto, a regra de recorrência para o custo mínimo *C(ai,aj)* pode ser dada por:

$$C(ai,aj) = min[C(ai,ak) + C(ak, aj)] + cost(ai, aj)$$

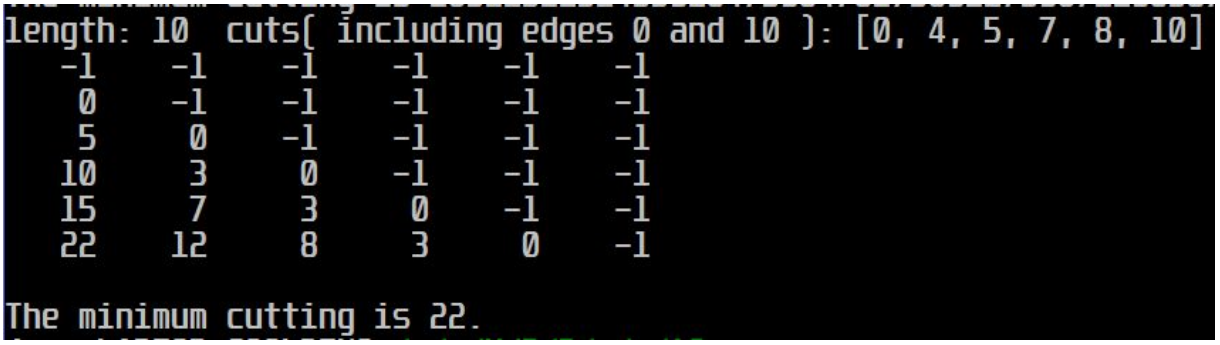
Onde *ak* é um dos cortes possíveis entre *ai* e *aj* (*k* varia entre *i+1* e *j-1*). O caso base da recursão é:

$$C(ai, ai+1) = 0$$

Pois o custo para cortar um bastão que não pode ser cortado é zero(o bastão deve ter extremidades *ai* e *ai+1*, logo não existem posições de corte intermediárias).

1.4. Estratégia de programação dinâmica

Utilizando a regra de recorrência descrita acima, foi utilizada uma abordagem *bottom-up*, utilizando uma tabela de memoização. Para cada possível corte *j*, calcula-se o custo mínimo desse corte ao verificar todos os cortes possíveis anteriores. A imagem abaixo exemplifica uma tabela gerada:



Cada posição da tabela representa o custo mínimo de corte *C(ai, aj)* onde a linha da tabela é a posição *aj*, e a coluna da tabela é a posição *ai*. Por exemplo, o custo mínimo para cortar de 0 a 10 é dado por tabela[0][5] (pois 0 é o corte de índice 0, e 10 é o corte de índice 5).

Nota-se que não são calculados *C(ai,ai)*, pois não faz sentido um corte que começa e termina no mesmo ponto (gerando um bastão de tamanho 0), além dos cortes *C(ai, ai+1) = 0*, do caso base da recorrência.

1.5. Complexidade de tempo

A solução possui basicamente três loops: Um loop que passa por todos os cortes possíveis (linhas da tabela). Há um loop interno a esse que busca o custo mínimo de corte dos cortes anteriormentes já feitos. Ainda, há um terceiro loop, interno ao segundo loop, que calcula o custo mínimo, verificando todas as possibilidades de cortes.

No pior caso, teremos N cortes, que verificam (N-1) cortes anteriores, que calculam seus valores testando ((n-1)-1) possíveis cortes, gerando uma complexidade O(N^3).

1.6. Complexidade de memória

Dado N possíveis cortes, é criada uma tabela de memoização que ocupa (N+2)^2 espaços de memória, além do vetor N que contém todas as posições possíveis para corte. Portanto, totaliza-se (N+2)^2 + N → O(N^2).

Por ser implementada em Python, a solução poderá utilizar diferentes tamanhos de memória para armazenar os custos de cada corte caso eles sejam muito grandes.

1.7. Escalabilidade

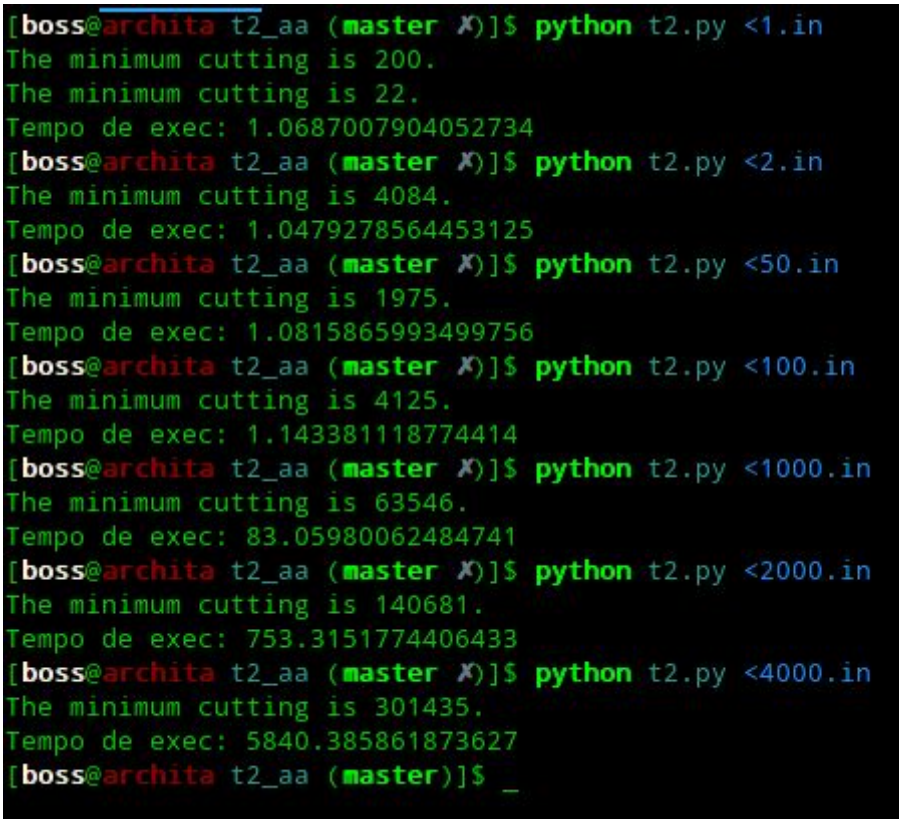
O programa criado pode ser executado para qualquer tamanhos de bastão e para qualquer números de cortes, devido à capacidade do Python de utilizar inteiros de tamanho variado. O tempo de execução não é impactado pelo comprimento do bastão, porém, a solução torna-se muito demorada ao utilizarmos mais de 5000 cortes, devido à complexidade cúbica. Os testes a seguir ilustram o tempo de execução para casos de tamanhos variados.

2. Execução e Testes

Testes para os casos descritos no pdf (contidos no arquivo 1.in), e um caso descrito manualmente via stdin:

```
[boss@archita t2_aa (master ✗)]$ python t2.py <1.in
The minimum cutting is 200.
The minimum cutting is 22.
[boss@archita t2_aa (master ✗)]$ python t2.py
1000
49
3 6 8 10 14 16 18 20 21 24 25 28 30 33 36 38 44 46 47 49 71 77 78 88 89 93 95 120 128
 150 159 174 188 199 222 233 255 277 288 404 424 454 487 543 547 567 578 589 678
0
The minimum cutting is 4084.
[boss@archita t2_aa (master ✗)]$ _
```

Testes para casos com 50, 100, 1000, 2000 e 4000 cortes e seus respectivos tempos de execução:



Para 10000 cortes, o programa não finalizou após 6h de execução e abortamos o teste.