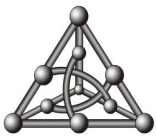


UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
Faculdade de Computação

Implementação Algorítmica

Atividade 4 — Problema do corte da barra de aço

1 Descrição

Uma empresa compra longas barras de aço e as corta em barras menores para revenda. Os cortes são realizados de tal modo que as barras resultantes têm sempre comprimento inteiro. Um corte não tem valor e não é cobrado pela empresa. Para cada $i = 1, 2, \ldots$, a empresa cobra o preço p_i por uma barra de comprimento i.

Como exemplo, suponha que temos uma barra de comprimento 10 metros com os valores dos seus pedaços de 1 a 10 metros dados pela seguinte tabela:

comprimento i	1	2	3	4	5	6	7	8	9	10
preço p_i	1	5	8	9	10	17	17	20	24	30

Dessa forma, temos o seguinte:

> **Problema do corte da barra de aço**: Dadas uma barra de n metros de comprimento e uma tabela de preços p_i para $i = 1, 2, \ldots, n$, determine o valor máximo de venda r_n que é possível obter pelo corte da barra e venda de seus pedaços.

Um exemplo para uma barra de $n = 4$ metros é mostrado na figura 1.

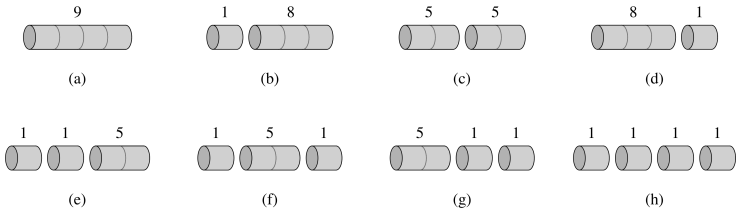


Figura 1: Exemplo com todos os cortes possíveis para uma barra de comprimento $n = 4$ e preços de venda dados pela tabela acima. Observe que o valor máximo de venda é obtido quando cortamos a barra em dois pedaços de 2 metros cada (letra (c)) e obtemos o valor de venda $p_2 + p_2 = 5 + 5 = 10$.

Observe que, para solucionar o problema original do corte de uma barra de comprimento n, devemos resolver problemas menores do mesmo tipo, isto é, solucionar o problema do corte de barras de comprimentos menores. Uma vez que fazemos o primeiro corte, podemos considerar os dois pedaços resultantes como instâncias independentes do problema do corte da barra de aço. A solução ótima do problema todo incorpora soluções ótimas para os dois subproblemas relacionados, maximizando o valor de venda para cada um dos dois pedaços. Dessa forma, dizemos

que o problema do corte da barra de aço exibe *subestrutura ótima*: soluções ótimas para um problema incorporam soluções ótimas para os subproblemas relacionados, que podemos solucionar independentemente.

Simplificamos uma decomposição de tal forma que consista de um primeiro pedaço à esquerda de comprimento i e um pedaço remanescente de comprimento $n - i$. Apenas o pedaço remanescente à direita pode ser dividido e, assim, a decomposição de uma barra de comprimento n é composta de um primeiro pedaço mais a decomposição do pedaço remanescente. A solução em que não há cortes pode ser vista como o primeiro pedaço de comprimento $i = n$ e preço de venda p_n e o pedaço remanescente com comprimento 0 com preço de venda correspondente 0. Dessa forma, escrevemos o valor máximo de venda de uma barra de comprimento n como

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i}) ,$$

onde r_n é o valor máximo de venda obtido quando cortamos uma barra de aço de comprimento n. Nessa formulação, uma solução ótima incorpora a solução de somente um subproblema relacionado (o remanescente).

Traduzindo diretamente a fórmula acima, obtemos o seguinte algoritmo recursivo:

```
CORTE-DA-BARRA(p, n)
01.  se n = 0
02.    devolva 0
03.  q ← −∞
04.  para i ← 1 até n
05.    q ← max(q, p[i] + CORTE-DA-BARRA(p, n − i))
06.  devolva q
```

No entanto, se codificamos o algoritmo CORTE-DA-BARRA em uma linguagem de programação e o executamos em um computador, percebemos que o programa fica lento quando o tamanho da entrada torna-se moderadamente grande. Para $n = 40$, esse programa deve gastar algo em torno de uma hora e a cada incremento de 1 em n, o tempo de execução do programa dobra, aproximadamente. Isso devido às repetidas chamadas recursivas de CORTE-DA-BARRA com os mesmos parâmetros.

Tendo observado que a solução recursiva ingênua é ineficiente porque soluciona os mesmos subproblemas repetidamente, fazemos com que cada problema seja solucionado uma única vez, gravando sua solução em uma tabela. Se precisamos da solução desse subproblema de novo posteriormente, podemos apenas consultá-la ao invés de recomputá-la.

A solução em programação dinâmica usando a estratégia *top-dowm* com *memoization* é apresentada a seguir:

```
CORTE-DA-BARRA-MEMORIZADO(p, n)
01.  seja r[0..n] um novo vetor
02.  para i ← 0 até n
03.    r[i] ← −∞
04.  devolva CORTE-DA-BARRA-MEMORIZADO-AUX (p, n, r)
```

```
CORTE-DA-BARRA-MEMORIZADO-AUX (p, n, r)
01.  se r[n] ≥ 0
02.    devolva r[n]
03.  se n = 0
04.    q ← 0
05.  senão
06.    q ← −∞
07.    para i ← 1 até n
08.      q ← max(q, p[i] + CORTE-DA-BARRA-MEMORIZADO-AUX(p, n − i, r))
09.  r[n] ← q
10.  devolva q
```

E a solução em programação dinâmica usando a estratégia *bottom-up* é a seguinte:

```
CORTE-DA-BARRA-DE-BAIXO-PARA-CIMA(p, n)
01.  seja r[0..n] um novo vetor
02.  r[0] ← 0
03.  para j ← 1 até n
04.    q ← −∞
05.    para i ← 1 até j
06.      q ← max(q, p[i] + r[j − i])
07.    r[j] ← q
08.  devolva r[n]
```

Observe finalmente que ambos algoritmos CORTE-DA-BARRA-MEMORIZADO e CORTE-DA-BARRA-DE-BAIXO-PARA-CIMA têm tempo de execução $\Theta(n^2)$.

Nesta atividade, você tem de implementar os dois algoritmos para solução do problema do corte das barras de aço: uma usando programação dinâmica (um dos algoritmos acima) e outra usando uma estratégia gulosa descrita a seguir.

Defina a **densidade** de uma barra de comprimento i como p_i/i, isto é, seu valor por metro. A estratégia gulosa para uma barra de comprimento n corta um primeiro pedaço de comprimento i, onde $1 \leq i \leq n$, com densidade máxima. Continua-se então aplicando a estratégia gulosa ao pedaço remascente de comprimento $n - i$.

2 Programa, entrada e saída

Você deve desenvolver e implementar os algoritmos conforme a descrição da seção 1.

Você deve construir conjuntos de dados de entrada (pseudo)aleatórios da seguinte forma. Primeiro, um valor do comprimento da barra n. Por exemplo, os comprimentos podem ser $n = 10, 20, \ldots, 100$. Para cada valor n do comprimento de uma barra, você deve determinar os preços de venda dos pedaços da barra de comprimentos inteiros de 1 a n. Depois disso, você deve executar os algoritmos propostos, mostrando o valor total de venda obtidos por cada um e

registrar seus tempos de execução. Ao final, mostre a porcentagem de acerto que a solução gulosa obtém em relação à solução da programação dinâmica. Limite os preços dos pedaços de uma barra no intervalo de 1 a 10.

2.1 Exemplo de entrada e saída

Um pequeno exemplo, para casos de teste com $n = 100, 200, \ldots, 2000$ é mostrado a seguir. Os tempos de execução dos algoritmos são mostrados em segundos. Os valores `vDP` e `tDP` indicam o valor total da venda obtido pelo algoritmo de programação dinâmica e o tempo de execução do algoritmo em segundos, respectivamente. Os valores `vGreedy` e `tGreedy` fazem as mesmas referências ao algoritmo guloso.

```
     n      vDP        tDP    vGreedy    tGreedy         %
-------------------------------------------------------------
   100      300   0.000016        300   0.000013    100.00
   200     1000   0.000057       1000   0.000046    100.00
   300      600   0.000128        300   0.000055     50.00
   400     1800   0.000263        800   0.000121     44.44
   500     1500   0.000344       1500   0.000279    100.00
   600     1800   0.000504       1800   0.000400    100.00
   700     3150   0.000680       1400   0.000273     44.44
   800     4800   0.000890       4800   0.000734    100.00
   900     3600   0.001138       3600   0.000890    100.00
  1000     5000   0.001435       5000   0.001398    100.00
  1100     3300   0.001802       3300   0.001454    100.00
  1200     7200   0.002149       7200   0.001744    100.00
  1300     5200   0.002296       5200   0.001828    100.00
  1400     2800   0.002875       1400   0.001445     50.00
  1500    10500   0.003582      10500   0.002428    100.00
  1600     6400   0.003477       3200   0.001619     50.00
  1700     5100   0.004034       5100   0.003378    100.00
  1800     4800   0.004634       1200   0.001195     25.00
  1900     4750   0.005101       1267   0.001493     26.67
  2000    16000   0.005844      16000   0.004757    100.00
```

3 Entrega

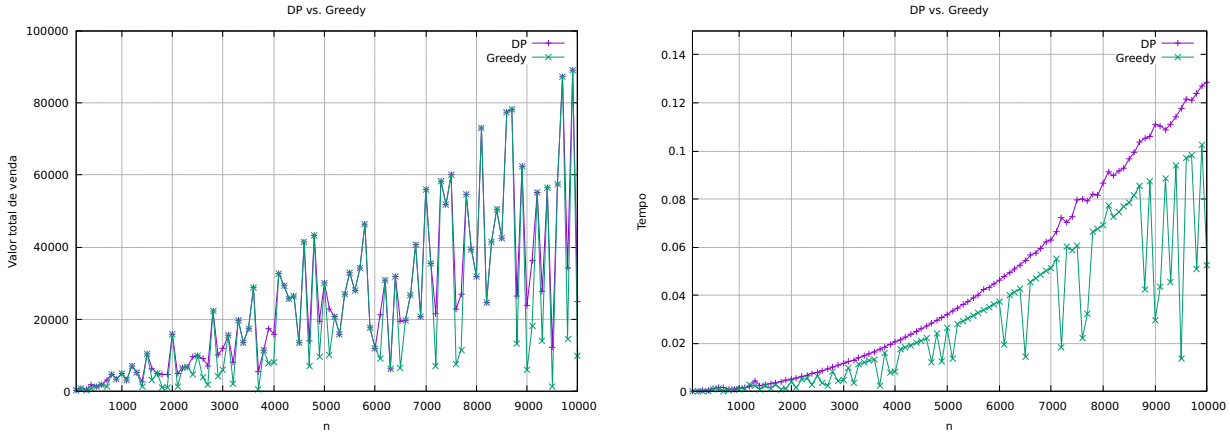
Instruções para entrega da sua atividade:

1. **O que entregar?**

   Um arquivo compactado a ser entregue deve conter o seguinte:

   - programa(s) desenvolvido(s) (e um arquivo **`Makefile`**, se for o caso),
   - tabela dos valores dos cortes das barras e tempos de execução dos algoritmos (em texto), de acordo com a formatação apresentada na seção 2.1, e

- gráficos (no formato **`.pdf`**) gerados a partir das tabelas dos valores e dos tempos de execução dos algoritmos. Exemplos relativos à tabela da seção 2.1 usando o software **`gnuplot`** são mostrados na Figura 2.



(a) Valor total de venda.

(b) Tempo.

Figura 2: Gráficos da execução dos algortimos do corte a barra de aço para o caso de teste com **`inc`** $= 100$, **`fim`** $= 10000$ e **`stp`** $= 100$.

Compacte todos esses arquivos com o compactador de sua preferência e entregue um único arquivo (com extensão **`.tgz`**, **`.bz2`**, **`.zip`**, **`.rar`**, ...).

2. **Forma de entrega**

   A entrega será realizada diretamente no Sistema (AVA/UFMS), na disciplina de Implementação Algorítmica – T01. Um fórum de discussão deste trabalho já se encontra aberto. Após abrir uma sessão digitando seu *login* e sua senha, vá até a sessão "Atividades" e escolha "Entrega da atividade 3". Você pode entregar a atividade quantas vezes quiser até às **23 horas e 59 minutos** do dia **27 de outubro de 2022**. A última versão entregue é aquela que será corrigida. Encerrado o prazo, não serão mais aceitos trabalhos.

3. **Linguagem de programação**

   Use a linguagem de programação de sua preferência para desenvolver esta atividade.

4. **Atrasos**

   Trabalhos atrasados não serão aceitos. Não deixe para entregar seu trabalho na última hora. Para prevenir imprevistos como queda de energia, problemas com o sistema, e/ou falha de conexão com a internet, sugerimos que a entrega do trabalho seja feita pelo menos um dia antes do prazo determinado.

5. **Erros**

   Trabalhos com erros de compilação/interpretação receberão nota ZERO. Faça todos os testes necessários para garantir que seu programa está livre de erros de compilação/interpretação.

6. **Arquivo com o programa fonte**

   Seu(s) arquivo(s) contendo o(s) fonte(s) do(s) programa(s) na linguagem escolhida deve(m) estar bem organizado(s). Um programa tem de ser muito bem compreendido por uma pessoa. Verifique se seu programa tem a indentação adequada, se não tem linhas muito longas, se tem variáveis com nomes significativos, entre outros. Não esqueça que um programa bem descrito e bem organizado é a chave de seu sucesso. Também não esqueça da documentação de seu programa.

7. **Conduta Ética**

   O trabalho deve ser feito INDIVIDUALMENTE/COM SEU GRUPO. Cada estudante tem responsabilidade sobre cópias de seu trabalho, mesmo que parciais. Não compartilhe seu programa ou trechos de seu programa. Você pode consultar seus colegas para esclarecer dúvidas e discutir idéias sobre o trabalho, ao vivo ou no fórum de discussão da disciplina, mas NÃO copie o programa!

   Trabalhos considerados plagiados terão nota ZERO.