Projeto e Análise de Algoritmos
Trabalho Prático 1 - Paradigmas

Vagner Clementino[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)

vagnercs@dcc.ufmg.br

Resumo. *TO DO*

1. Introdução

Apesar da crença um cérebro maior não indica maior inteligência. Um contraexemplo bastante conhecido é Albert Einsten, cujo volume de seu cérebro não era maior do que o da média. Com a melhoria das imagens de ressonância magnética (IRM), diversos estudos vêm sendo proposto com o objetivo de correlacionar o *volume* do cérebro com o *Quociente de Inteligência* (QI). Um estudo utilizando imagens concluiu que a correlação entre QI e o volume do cérebro é consistente, todavia, as correlações fracas, e não há como comprovar uma relação direta. Em síntese: ser um *"cabeção"* não é indicativo de inteligência.

A fim de refutar de vez esta crença, este trabalho se propõem em analisar os dados de *peso* e *QI* de um determinada população, com o objetivo de encontrar o maior número de pessoas cujo o peso do seu cérebro é menor, contudo, com um QI maior. O problema será formalmente definido na Seção 2.

Este documento está estruturado como segue. A Seção 2 define formalmente o problema, fazendo uma relação do mesmo com o problema da *Longest increasing subsequence*; a Seção 3 apresenta as três soluções propostas para resolver o problema: Força Bruta (subseção 3.1), Gulosa (subseção 3.2) e Programação Dinâmica (subseção 3.3); a Seção 4 faz uma discussão sobre resultados os experimentais de cada solução; a Seção 5 discorre sobre as conclusões do trabalho.

2. Definição Formal do Problema

O problema pode ser definido formalmente da seguinte forma. Dados $C = \langle c_1, c_2, \ldots, c_n \rangle$ um conjunto de indivíduos de tamanho n, bem como $P = \langle p_1, p_2, \ldots, p_n \rangle$ e $Q = \langle q_1, q_2, \ldots, q_n \rangle$, onde p_i e q_i representa, respectivamente, o peso e o QI do i-ésimo indivíduo. O problema consiste em encontrar o subconjunto $S \subseteq C$ tal que para todo $s_i, s_j \in S$ onde $i < j$ $p_i < p_j$ **e** $q_i > q_j$. Além disso, $|S|$ deve ser *máximo*.

O problema proposto neste trabalho pode ser facilmente mapeado para um bem conhecido problema de otimização denominado *Longest Increasing Subsequence - LIS*[Knuth 2013]. Para tanto, basta ordenar um dos conjuntos P ou Q de forma crescente ou decrescente. Por exemplo, caso o conjunto P de forma decrescente,

o problema se transforma em encontrar a maior *LIS* no novo conjunto $Q^{'}$ que foi gerado pela ordenação de P.[1]

O LIS é um problema bastante estudado e existem diversas abordagens na literatura para resolvê-lo. Este trabalho utilizou algumas destas referências nas soluções propostas na Seção 3.

3. Soluções propostas

Neste trabalho foi proposto três soluções para o problema utilizando os paradigmas Força Bruta, Gulosa e Programação Dinâmica. Para cada paradigma discute-se como foi realizada a modelagem, o funcionamento do algoritmo proposto e a análise da complexidade de tempo e de espaço. A descrição dos algoritmos será feita em mais alto nível por meio de pseudocódigo.

3.1. Força Bruta

3.1.1. Modelagem

O paradigma de Força Bruta é o único que garantidamente encontra uma solução ótima para qualquer problema computável. Contudo, o preço que se paga por esta aplicabilidade universal é o alto custo de tempo e/ou espaço necessário. A principal característica de uma abordagem Força Bruta é que ela faz uma *busca integral no espaço de soluções* [Kleinberg and Tardos 2005]. Neste sentido, ao desenvolvermos um algoritmo força bruta, ele deverá listar todas as possíveis soluções do problema para posteriormente definir a melhor entre as soluções válidas.

No contexto do problema estudado, o espaço de soluções consiste de todas as permutações dos elementos do *power set* de C, cuja a notação é dada por 2^C. Desta forma, o algoritmo a ser proposto deverá ser capaz de criar cada permutação possível de tamanho $1, 2, \ldots, n$. Para cada permutação/solução criada deverá ser verificado se a solução é válida a fim de encontrar a melhor entre as válidas. Na próxima subseção descreveremos o algoritmo proposto.

3.1.2. O Algoritmo Força Bruta

O algoritmo 1 apresenta em alto nível a abordagem Força Bruta utilizada. Por meio do método GENERATE-ALL-SOLUTIONS todas as possível soluções são geradas. A partir delas a solução ótima (de maior tamanho) é encontrada e posteriormente retornada S_{best}. O método IS-VALID verifica se uma da solução é valida verificando se cada par de item respeita as restrições do problema.

3.1.3. Análise de Complexidade

Conforme exposto anteriormente, na abordagem de Força Bruta é realizada uma busca em cada item do espaço de soluções do problema. Na subseção 3.1.1 discutiu-se que este espaço de solução $O(2^n)$. Neste contexto, o método responsável por varrer

[1]Considera-se que para toda posição i em P e Q represente os dados do mesmo indivíduo

Algorithm 1: BRUTE-FORCE encontra a solução ótima listando todas elas.

Input: As sequências finitas $C = \langle c_1, c_2, \ldots, c_n \rangle$, $P = \langle p_1, p_2, \ldots, p_n \rangle$ e $Q = \langle q_1, q_2, \ldots, q_n \rangle$

Output: Uma sequência $S_{best} \subseteq C$ tal que atende as restrições do problema e seja máxima

```
1 U ← GENERATE-ALL-SOLUTIONS(C)
2 max ← 0
3 S_best ← ∅
4 for each S in U do
5     if IS-VALID(S) then
6         if S.legth() > max then
7             max ← S.legth()
8             S_best ← S
9 return S_best
```

o espaço de soluções necessariamente terá sua complexidade igual $O(2^n)$. No algoritmo 1 este trabalho é realizado pelo método GENERATE-ALL-SOLUTIONS. Como não as demais funções do algoritmo possuem complexidade inferiores, podemos concluir que o algoritmo de força bruta proposto possui ondem de complexidade de tempo igual a $O(2^n)$.

No tocante a complexidade de espaço, o algoritmo necessita carregar os conjuntos C, P e Q a fim de poder gerar todas as soluções e verificar se elas são válidas. Apesar de no pior caso existir $O(2^n)$ possíveis soluções, o sistema apenas armazena a melhor solução encontrada no momento. Partindo desta estratégia teremos um cutso de espaço igual a $O(1)$. Neste sentido, o algoritmo têm uma complexidade de espaço igual a $O(n)$.

3.2. Uma abordagem Gulosa

Apesar do algoritmo de força bruta resolver o problema, conforme poderá ser observado na Seção 4 onde descreve a análise experimental dos algoritmos, tal abordagem torna-se impraticável quando o tamanho da entrada cresce. Com o objetivo de encontrar uma solução de melhor desempenho, esta seção descreve uma solução baseada no paradigma *Guloso*.

3.2.1. Modelagem

Um problema é passível de ser resolvido utilizando a abordagem Gulosa caso ele possua a propriedade da *subestrutura ótima*. Um problema é dito ter subestrutura ótima se uma solução ótima pode ser construído de forma eficiente a partir de soluções ótimas de seus subproblemas [Cormen et al. 2009]. Vamos provar que o problema em questão possui subestrutura ótima.

Iniciemos, sem perda de generalidade, ordenando o conjunto de entrada C de forma descrescente pelo seu conjunto de pesos P. Conforme exposto na Seção 2 ao

realizarmos a ordenação da entrada conforme proposto, o problema transforma-se em encontrar uma *LIS* no novo conjunto de QI's $Q^{'}$ gerado.

Seja S_{ij} a maior LIS entre os elementos i e j do conjunto $Q^{'}$, conforme descrito no parágrafo anterior, tal que $1 \leq i \leq (n-1)$ e $i < j$. Desta forma, S_{1n} é a solução ótima para o problema. Suponha que escolhamos um valor k qualquer de modo que $i \leq k < j$. As soluções S_{ik} e $S_{(K+1)j}$ *são ótimas* o que pode ser provado por absurdo. Suponha sem perda de generalidade que exita uma solução S'_{ik} tal $|S'_{ik}| > |S_{ik}|$, a existência de S'_{ik} vai contra a suposição de que S_{ij} seja ótima.

Após provado que o problema apresenta subestrutura ótima já é possível propor um algoritmo guloso para o problema. A solução proposta é detalhada na subseção 3.2.2.

3.2.2. O Algoritmo Guloso

O *Patience Sorting* é um algoritmo de ordenação baseado no jogo de cartas de mesmo nome. O funcionamento básico do jogo é descrito a seguir:

1. Inicialmente, não há pilhas. A primeira carta formará uma nova pilha de tamanho 1.
2. A cada nova carta retirada do baralho, caso ela seja menor que alguma carta no topo de qualquer uma das pilhas disponíveis, a carta será colocada no topo da pilha; caso contrário uma nova pilha será criada.
3. Quando não há mais cartas no baralho, o jogo termina.

A Figura 1 exibe o resultado da aplicação do Patience Sorting em um conjunto de números. Conforme pode ser observado ao final do algoritmo cada pilha terá uma *sequência decrescente*. Utilizando esta propriedade do algoritmo foi realizada uma versão modificada do algoritmo para resolver o problema proposto neste trabalho. Basicamente realiza-se a ordenação do conjunto P de forma crescente; a partir do novo conjunto $Q^{'}$ resultante aplica-se uma versão gulosa do Patience Sorting de que modo que ao final a maior pilha gerada será a solução do problema. O Algoritmo apresenta de forma genérica a solução proposta.

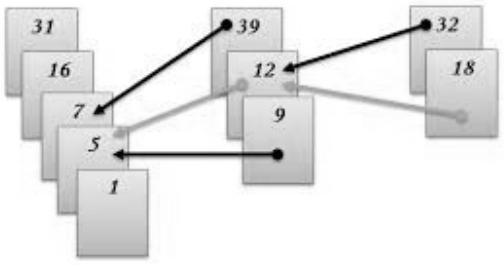

Figura 1. O resultado da aplicação Patience Sorting

Conforme exposto o algoritmo 2 é uma modificação gulosa do Patience Sorting. A parte gulosa do algoritmo está no método `GREEDY-CHOICE` (c_i, q_i) que consiste basicamente em buscar a pilha cujo elemento no topo seja maior do que q_i.

Algorithm 2: GREEDY encontra a solução de forma gulosa.

Input: As sequências finitas $C = \langle c_1, c_2, \ldots, c_n \rangle$, $P = \langle p_1, p_2, \ldots, p_n \rangle$ e $Q = \langle q_1, q_2, \ldots, q_n \rangle$

Output: Uma sequência $S_{best} \subseteq C$ tal que atende as restrições do problema e seja máxima

```
1  INITIALIZE o conjunto de n pilhas S_1, S_2, ..., S_n
2  SORT (C) //Ordenando pelos pesos
3  maxLength ← 0
4  bestStack ← 0
5  for i to n do
6      j ← GREEDY-CHOICE (c_i, q_i)
7      PUSH (S_j, c_i)
8      if |S_j| > maxLength then
9          maxLength ← |S_j|
10         bestStack ← j
11 return S_bestStack
```

Caso exista mais de uma pilha candidata será *retornada aquela cujo o elemento no topo seja o maior.*

Não obstante, cabe um questionamento se a solução gulosa aqui proposta leva à solução ótima.

3.3. Programação Dinâmica

Figure and table captions should be centered if less than one line (Figure **??**), otherwise justified and indented by 0.8cm on both margins, as shown in Figure 2. The caption font must be Helvetica, 10 point, boldface, with 6 points of space before and after each caption.

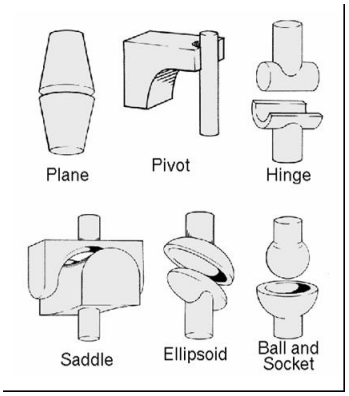

Figura 2. This figure is an example of a figure caption taking more than one line and justified considering margins mentioned in Section ??.

In tables, try to avoid the use of colored or shaded backgrounds, and avoid thick, doubled, or unnecessary framing lines. When reporting empirical data, do not use more decimal digits than warranted by their precision and reproducibility.

Table caption must be placed before the table (see Table 1) and the font used must also be Helvetica, 10 point, boldface, with 6 points of space before and after each caption.

Tabela 1. Variables to be considered on the evaluation of interaction techniques

	Chessboard top view	Chessboard perspective view
Selection with side movements	6.02 ± 5.22	7.01±6.84
Selection with in-depth movements	6.29±4.99	12.22±11.33
Manipulation with side movements	4.66± 4.94	3.47±2.20
Manipulation with in-depth movements	5.71 ±4.55	5.37 ±3.28

4. Análise Experimental

All images and illustrations should be in black-and-white, or gray tones, excepting for the papers that will be electronically available (on CD-ROMs, internet, etc.). The image resolution on paper should be about 600 dpi for black-and-white images, and 150-300 dpi for grayscale images. Do not include images with excessive resolution, as they may take hours to print, without any visible difference in the result.

5. Conclusões

Bibliographic references must be unambiguous and uniform. We recommend giving the author names references in brackets, e.g. [Knuth 1984], [Boulic and Renault 1991], and [Smith and Jones 1999].

The references must be listed using 12 point font size, with 6 points of space before each reference. The first line of each reference should not be indented, while the subsequent should be indented by 0.5 cm.

Referências

Boulic, R. and Renault, O. (1991). 3d hierarchies for animation. In Magnenat-Thalmann, N. and Thalmann, D., editors, *New Trends in Animation and Visualization*. John Wiley & Sons ltd.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition.

Kleinberg, J. and Tardos, E. (2005). *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Knuth, D. (2013). *Art of Computer Programming, Volume 4, Fascicle 4,The: Generating All Trees–History of Combinatorial Generation*. Pearson Education.

Knuth, D. E. (1984). *The TEX Book*. Addison-Wesley, 15th edition.

Smith, A. and Jones, B. (1999). On the complexity of computing. In Smith-Jones, A. B., editor, *Advances in Computer Science*, pages 555–566. Publishing Press.