Algoritmos Gulosos ("Greedy Algorithms")
OBI'2003—Olimpíada Brasileira de Informática

04 de julho de 2003
guilherme.pinto@ic.unicamp.br

O termo "Algoritmo Guloso" se aplica normalmente à uma *classificação* de algoritmos; ou seja, um determinado algoritmo pode ser, ou não, guloso de acordo com um determinado critério. Contudo, de fato, no exercício de resolver um problema computacional de otimização (minimizar ou maximizar uma determinada função objetivo), muitas vezes propomos um algoritmo guloso naturalmente, inconscientemente. Quer dizer, ser guloso é algo comum em programação, mesmo que a gente não conheça a teoria por trás da classificação e análise de algoritmos. O que ocorre, então, é que, mais do que uma classificação, "Algoritmo Guloso" pode ser visto como uma *técnica* de projeto de algoritmos. Assim, conhecendo um pouco da técnica, podemos nos habituar a fazer mentalmente a seguinte pergunta, quando nos depararmos com um problema de otimização: *será que existe uma solução gulosa eficiente para esse problema?*

A palavra "eficiente" é a chave da questão. Ela responde à pergunta: *por que devo conhecer um pouco da técnica de algoritmos gulosos?* A resposta é que, geralmente, uma solução gulosa é simples (fácil de implementar) e eficiente (o algoritmo executa rapidamente). Algumas outras técnicas para projeto de algoritmos, como projeto por indução, programação dinâmica ou "Branch and Bound", resultam, geralmente, em algoritmos mais complicados e não tão eficientes. Para alguns problemas, usar programação dinâmica, por exemplo, pode ser como dar um tiro de canhão para matar um pardal. Usar uma solução gulosa pode gerar um algoritmo mais simples e eficiente; e que mata o pardal do mesmo jeito.

Mas isso tem um preço...

Como veremos, ser guloso é, de modo bem geral, propor um algoritmo iterativo no qual, a cada passo, aplicamos um critério guloso de otimização *local* para construir uma parte da solução. Veremos com mais clareza, nos exemplos, o que significa otimizar localmente. Essa parte da solução não será mudada mais. Na próxima iteração, uma nova parte da solução será construída usando o mesmo critério, e assim em diante. Ao final das iterações, se o critério guloso é *correto*, a solução construída será, de fato, uma solução ótima *global*, que é o que pede o problema. O termo "guloso" (em inglês "greedy", cuja melhor tradução talvez seja "ganancioso") vem exatamente do fato de que o algoritmo constrói parte da solução, a cada iteração,

otimizando localmente, e depois não volta atrás. Algoritmos gulosos são geralmente eficientes justamente porque não voltam atrás naquilo que já foi construído, não há o que chamamos de "backtracking". A técnica, em si, consiste em propor um critério guloso e mostrar que ele é correto.

O preço é esse: é preciso garantir que o critério guloso de fato leva à solução ótima. Isso pode não ser simples. Com a experiência, como em tudo na vida, a tarefa de provar, ou mostrar com argumentos rigorosos, que o critério é correto, vai ficando cada vez mais fácil e cada vez mais prazerosa.

A técnica gulosa possui essa diferença em relação àquelas outras técnicas já citadas. Por exemplo, projeto por indução ou programação dinâmica estão baseados numa solução recursiva para o problema. Propor um algoritmo de programação dinâmica, por exemplo, é definir o preenchimento de uma tabela baseado numa relação de recorrência. A relação de recorrência em si é o argumento a ser usado para mostrar a correção do algoritmo. Não existe, em programação dinâmica, por exemplo, a tentativa e erro. Não podemos "chutar" uma relação de recorrência e ver no que vai dar. O algoritmo provavelmente não dará uma solução correta, que dirá uma solução ótima. No caso de algoritmos gulosos existe algo parecido com tentativa e erro. Podemos inventar um critério qualquer, mas precisamos, depois, mostrar que o critério leva à solução ótima. Se não temos um argumento convincente para a correção do critério, não podemos chamar nosso procedimento de "algoritmo". Mas isso também tem um lado bom. Um critério guloso pode não levar ao ótimo em todos os casos, mas pode conseguir soluções próximas da solução ótima, ou mesmo a solução ótima, em alguns casos. E isso pode eventualmente atender à necessidade de quem propôs o problema. Nesse caso, chamamos nosso procedimento de "heurística". Com isso, para finalizar essa introdução, notamos que a técnica gulosa é também uma boa técnica para a obtenção de heurísticas.

1 Critérios Gulosos

O elementos, ou palavras-chave, da técnica gulosa são:

> (**Otimização**) Aplica-se a problemas de otimização (minimizar ou maximizar alguma função objetivo). Às vezes o problema é de decisão mas existe subjacente uma otimização;
>
> (**Solução**) O que constitui, para o problema em questão, uma solução? O que constitui uma solução ótima? A solução, geralmente, tem a forma de um conjunto de números; ou uma seqüência de arestas de um grafo; ou uma seqüência de elementos;
>
> (**Critério Guloso**) Qual é o critério que será usado localmente, a cada iteração, para construir parte da solução;

(**Correção**) Qual é o argumento que mostra que a solução dada pelo critério guloso é ótima?

Há certos tipos de problemas que não admitem solução gulosa. Entre eles estão aqueles que não possuem uma característica que é necessária para a aplicação eficiente da técnica gulosa: é o que chamamos de *subestrutura ótima*. Essa mesma característica é necessária para a aplicação eficiente de outras técnicas, como programação dinâmica. O que é subestrutura ótima? Significa, em termos gerais, que a solução ótima de uma instância do problema é composta por subsoluções que também são ótimas para subinstâncias daquela instância. Isso ficará mais claro nos exemplos das seções seguintes. Por enquanto, vale a pena somente dizer que é precisamente essa subestrutura ótima que deve ser explorada para se obter o argumento que mostra que um dado critério guloso leva à solução ótima. Veremos isso também.

Até agora falamos muito sobre critérios gulosos sem apresentar um exemplo mais concreto. Bem, daremos agora um exemplo interessante. Alertamos de antemão que o procedimento que resulta do critério guloso que daremos não está bem definido; mas ele é interessante como ilustração.

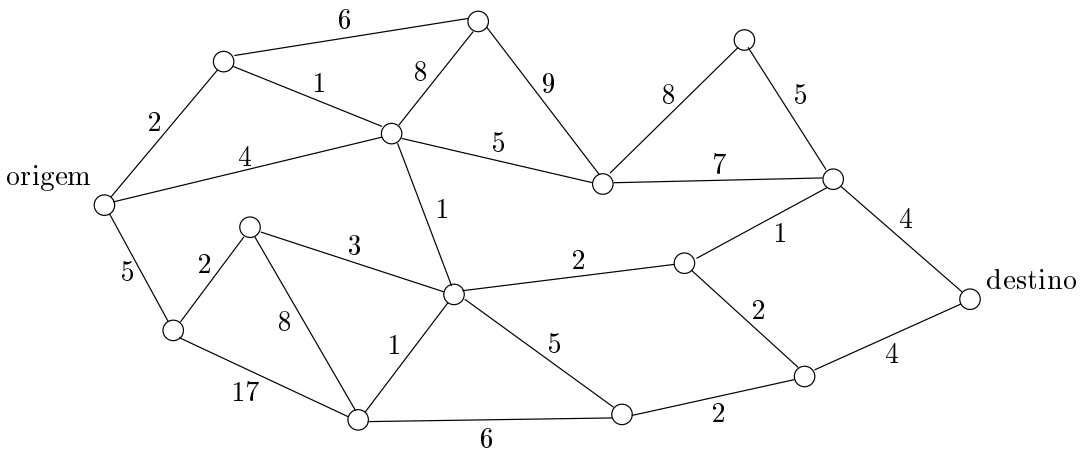
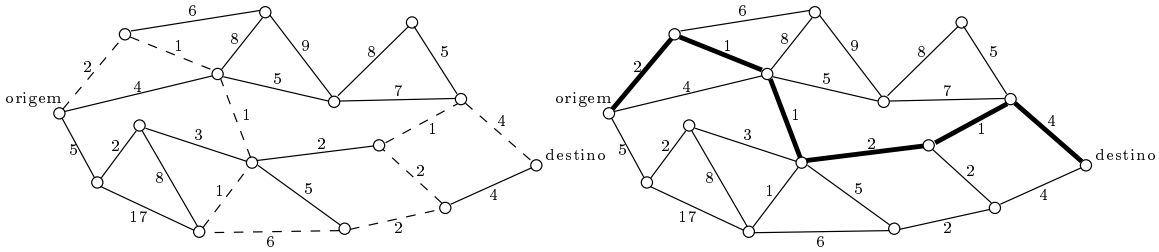


Figura 1: Uma malha viária com os custos dos pedágios

Considere uma malha viária interligando centenas de cidades. Entre cada par de cidades há um pedágio com um determinado custo. Dada uma cidade origem e uma cidade destino, você deve escolher o trajeto de menor custo entre essas duas cidades, onde o custo é definido como a soma dos valores dos pedágios, desconsiderando qualquer outro fator como, por exemplo, consumo de gasolina. Veja que o caminho de menor custo não é necessariamente o caminho que passa pelo menor número de cidades intermediárias entre a origem e o destino. Considere que você tem um mapa com as cidades, as estradas e o custo dos pedágios, como na figura 1.

Considere o seguinte procedimento. Partindo da cidade origem, verifique no mapa os custos dos pedágios para cada cidade ligada diretamente à cidade origem e escolha

o *menor* pedágio. Dirija até a cidade correspondente e escolha novamente o menor pedágio, considerando apenas cidades adjacentes pelas quais você ainda não passou. Repita esse passo até chegar na cidade destino. Veja que o critério é local, ou seja, você escolhe olhando apenas os custos das cidades adjacentes àquela na qual você está. O guloso aqui, ou ganancioso, é que você decide pagar o menor pedágio possível a cada iteração. Mas como é fácil ver, na figura 2, esse critério não escolhe necessariamente o trajeto de menor custo[1].



Trajeto do Guloso Trajeto ótimo

Figura 2: O Guloso **não** leva ao ótimo

2 O Problema da Seleção de Atividades

O problema da seleção de atividades é o seguinte. É dado como entrada um conjunto de n atividades, representado pelo conjunto $S = \{1, 2, 3, \ldots, n\}$. Cada atividade i possui um tempo de início s_i e um tempo de finalização f_i, sempre respeitando $s_i < f_i$. O objetivo é selecionar um subconjunto $A \subseteq S$ das atividades, com a quantidade máxima possível de atividades, para serem realizadas por uma mesma pessoa. Para isso, todo par (i, j) de atividades em A deve ser *compatível*: $s_i \geq f_j$ ou $s_j \geq f_i$. Chamamos um tal conjunto A também de *compatível*.

Assuma que o conjunto S é dado ordenado pelo tempo de finalização das atividades, ou seja, $f_1 \leq f_2 \leq \ldots \leq f_n$. A figura 3 dá um exemplo de instância (ou entrada) para este problema.

Bem, o problema pode ser resolvido pelo seguinte método *força bruta* (neste caso, força brutíssima):

[1]Para o exemplo da figura, esse procedimento não escolhe o trajeto de menor custo, mas, pelo menos, leva até a cidade destino. Veja porém que o procedimento não está bem definido. Para certos pares de cidades, você ficaria sem opção e não chegaria à cidade destino. Uma nota interessante é que este problema possui, de fato, um algoritmo guloso. O critério é um pouco mais sofisticado mas sempre escolhe o trajeto de menor custo. É o bastante famoso algoritmo de *Dijkstra* para caminhos mínimos em grafos.

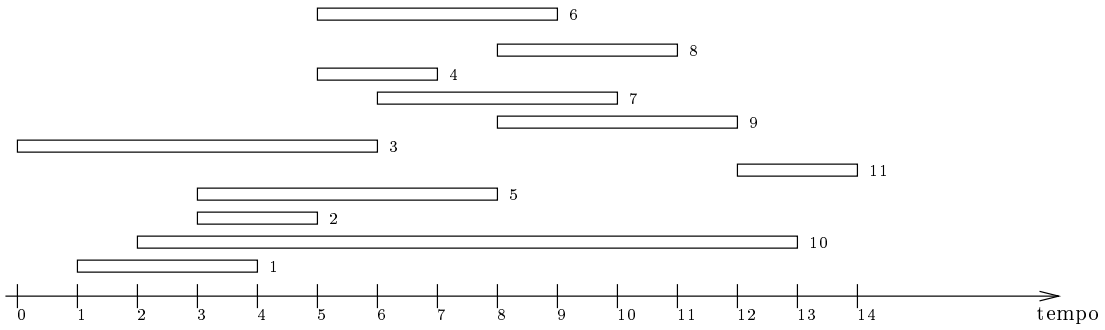


Figura 3: Uma instância para o problema da Seleção de Atividades

1. $A = \emptyset$;
2. para cada subconjunto B de S:
   (a) Se B for compatível e $|B| > |A|$, faça $A = B$;
3. retorne A.

É fácil ver que esse algoritmo é pra lá de ineficiente. Como existem 2^n subconjuntos de S, o número de passos será algo mais ou menos proporcional a $n2^n$.

Primeira tentativa "gulosa". Considere o seguinte procedimento, que tem o critério guloso a) "inclua na solução a atividade de *menor* duração". A idéia deste critério é que, ao escolher a atividade de menor duração, deixamos mais tempo livre na solução A:

1. $A = \emptyset$;
2. Selecione a atividade i, do conjunto das atividades de S que não estão em A, de menor duração, compatível com o conjunto A:
   (a) Se não existe tal atividade, vá para o passo 3);
   (b) Caso contrário, faça $A = A \cup \{i\}$ e repita o passo 2);
3. retorne A.

Bem, dá para ver que esse procedimento vai retornar um conjunto A de tamanho pelo menos 1, desde que $S \neq \emptyset$. Mas será que o conjunto A retornado sempre será de tamanho máximo? A resposta é não, como podemos verificar no contra-exemplo da figura 4. O guloso retorna o subconjunto $\{2, 4\}$, mas existe um subconjunto compatível maior: $\{1, 3, 5\}$.

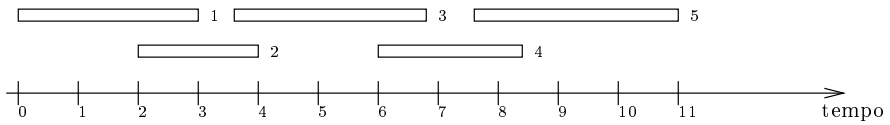


Figura 4: O critério a) falha para esta instância

Segunda tentativa "gulosa". Considere agora o mesmo procedimento, com o critério guloso b) "inclua na solução a atividade que termina mais cedo":

1. $A = \emptyset$;
2. Selecione a atividade i, do conjunto das atividades de S que não estão em A, com o menor tempo de finalização, compatível com o conjunto A:
   (a) Se não existe tal atividade, vá para o passo 3);
   (b) Caso contrário, faça $A = A \cup \{i\}$ e repita o passo 2);
3. retorne A.

Se nós executarmos este novo procedimento no exemplo da figura 3 veremos que ele realmente retorna um subconjunto de tamanho máximo possível. Mas isso sempre vai acontecer? A resposta, neste caso, é sim. Veremos agora uma argumentação que pode ser usada para mostrar que esse critério guloso sempre levará à uma solução ótima. Notamos também que o número de passos desse algoritmo será algo proporcional a n, um algoritmo linear.

Argumentando pela correção do Guloso. A idéia do argumento é, partindo de uma solução ótima, ou seja, de um subconjunto $O \subseteq S$ compatível de tamanho máximo, mostrar que a solução A construída pelo guloso terá, também, tamanho $|O|$; portanto será também ótima. Faremos isso substituindo as atividades da solução O pelas atividades escolhidas pelo guloso, uma a uma. O primeiro passo do argumento é substituir a primeira atividade de O pela atividade de número 1, que é a que o guloso escolhe primeiro.

Então, temos a seqüência de passos:

1. Por hipótese, O é uma solução ótima qualquer para o conjunto S;
2. Seja k o número da primeira atividade em O (a atividade que termina mais cedo em O):
   (a) Se $k = 1$, então a primeira atividade de O já é a que o guloso escolheu;

(b) Senão, então veja que pela definição do problema, $f_1 \leq f_k$, portanto podemos substituir a atividade k pela atividade 1, na solução O, que teremos uma outra solução ótima B^1 para S (B^1 tem o mesmo número de atividades que O). Veja a ilustração na figura 5.

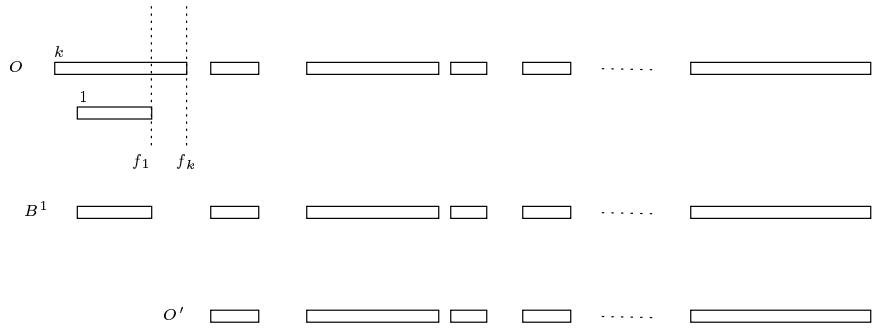


Figura 5: Argumentando pela correção do critério b)

Agora considere a seguinte subinstância: $S' = \{i \in S \mid s_i \geq f_1\}$. Ou seja, a subinstância que contém todas as atividades de S que iniciam após o término da atividade 1 de S. Considere, agora, a subsolução O' composta de todas as atividades em O, menos a atividade k (figura 5). Aqui vemos aquilo que chamamos antes de subestrutura ótima. O fato é que a subsolução O' é solução ótima para a subinstância S'. Para ver que isso é verdade, podemos argumentar por contradição. Suponha que houvesse uma solução C' para S' com tamanho maior do que $|O'|$. Ora, se isso fosse possível, teríamos então que o conjunto $C' \cup \{1\}$ seria uma solução para S de tamanho **maior** do que $|O|$. Mas isso é uma contradição, porque tomamos por hipótese que O é ótima!

Como O' é solução ótima para S', podemos repetir a mesma seqüência de passos (agora com O' no lugar de O e S' no lugar de S) e obter uma segunda solução B^2, que também é solução ótima para S, e que tem, pelo menos, as duas primeiras atividades iguais às do guloso. Repetindo a seqüência de passos $|O|$ vezes obtemos, ao final, a solução $B^{|O|} = A$; obtendo assim o argumento que queríamos.

3 O Problema da Cobertura de Pontos

Considere um conjunto de n pontos na reta real, nas posições $x_1, x_2, \ldots, x_n$ (assuma $x_1 \leq x_2 \leq \ldots \leq x_n$). Imagine agora que você tem n pedaços de esparadrapo, todos os pedaços de mesmo comprimento, igual a 1. Você precisa descobrir qual é a menor quantidade possível de pedaços de esparadrapo, com a qual é possível cobrir todos os n pontos. Note que no pior caso, você precisará de todos os n pedaços; e no melhor caso, de apenas 1 pedaço. Veja o exemplo na figura 6.

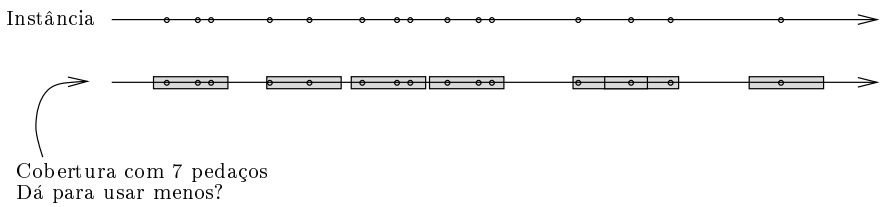


Figura 6: Cobertura de Pontos

Qual seria um possível critério guloso para esse problema? Note que, como não há pontos à esquerda de x_1, podemos colar a extremidade esquerda do primeiro esparadrapo em cima de x_1, na tentativa de cobrir o maior número possível de pontos com o primeiro esparadrapo. Para os pontos que restarem descobertos, nós repetimos esse passo, até que todos estejam cobertos. A figura 7 mostra o resultado desse procedimento guloso para a mesma instância da figura 6. Esse critério sempre leva ao ótimo? Intuitivamente, parece claro que sim. Mas precisamos de um argumento mais rigoroso.

Figura 7: Solução Gulosa

Correção. O argumento para esse caso é o mesmo do problema anterior das atividades; ou melhor, segue a mesma idéia. Precisamos apenas notar que, neste caso, dada uma solução O por hipótese ótima: ou o primeiro esparadrapo de O já está com a extremidade esquerda sobre x_1; ou, então, podemos obter uma nova solução ótima B^1, empurrando o primeiro esparadrapo para a direita, até que sua extremidade esquerda esteja sobre x_1. Isso não descobre nenhum ponto, nem altera o número de pedaços de esparadrapo. Você pode verificar que as demais implicações, como a da subestrutura ótima, valem neste caso também.

4 O Problema do Troco

Você está trabalhando como caixa de um banco que faz o maior esforço para sempre agradar o cliente. Até regras aparentemente sem importância devem ser seguidas. Uma delas é a seguinte: ao dar um troco de n centavos em moedas para o cliente, você tem que dar a ele a *menor* quantidade possível de moedas. Assuma que o banco tem um estoque ilimitado de moedas dos seguintes valores $\{1, 5, 10, 25\}$. Qual procedimento você usaria? Faria vários montinhos com todas as possíveis combinações que somam n, pegaria o menor montinho e daria para o cliente?

O procedimento guloso que os caixas normalmente usam é o seguinte. Dão ao cliente moedas do maior valor, enquanto a soma total for menor do que n (se n for múltiplo do maior valor, melhor ainda); depois dão moedas do segundo maior valor, de novo enquanto a soma total for menor do que n; e assim por diante. Note que se tivéssemos que programar essa idéia, o algoritmo seria muito rápido.

Esse procedimento guloso sempre leva ao ótimo? Novamente, intuitivamente sim. Nesse caso, para os valores $\{1, 5, 10, 25\}$, de fato esse guloso leva sempre à menor quantidade possível de moedas. Veremos abaixo uma argumentação com algum rigor. Mas aqui temos uma boa oportunidade para ver por que "provas", "demonstrações" ou "argumentos rigorosos" são necessários. O ponto é o seguinte. Para alguns outros conjuntos de valores de moedas, esse guloso falha grosseiramente, muito embora a intuição de que ele está correto permaneça. Como exercício, encontre um conjunto de quatro valores de moedas e um n, tal que o guloso falha.

Correção. Vamos primeiro fazer um esforço de sistematização daquela idéia de argumentação da Seção 2. Para esse problema do Troco, novamente, a idéia do argumento é a mesma. Notamos três elementos naquela argumentação:

1. (**Subestrutura ótima**) Mostramos que o problema possui essa característica;

2. (**A primeira escolha do Guloso**) Mostramos que o procedimento possui a chamada "Greedy choice property", que dizer, dado uma solução ótima por hipótese, podemos construir uma outra solução ótima, que possui a escolha feita pelo guloso;

3. (**A Indução**) Usando esses dois elementos anteriores, argumentamos que uma seqüência de passos leva à demonstração do argumento. Nesta apostila não estamos explicitando a demonstração por indução, mas o mais rigoroso seria fazê-lo[2].

Vamos agora discutir o caso do problema do Troco. O problema possui, quase obviamente, subestrutura ótima. Se retirarmos uma moeda de valor k de uma solução ótima para um troco de n, as moedas que restaram são uma solução ótima para um troco de $n - k$.

Para mostrar o segundo elemento, primeiro notamos que a ordem das moedas numa solução não importa; claro, sempre a soma será a mesma. Com isso, podemos sempre assumir que, em qualquer solução, as moedas estão ordenadas por valor.

Dado uma solução ótima por hipótese, vamos mostrar algo bem forte, que implica o segundo elemento: a primeira moeda da solução ótima *tem* que ser a moeda que o guloso escolheu primeiro. Temos quatro casos a analisar. Se a primeira moeda do guloso é 25, 10, 5 ou 1. Faremos o caso para 25, que é o mais interessante. Os demais deixamos como exercício. Argumentamos por contradição. Suponha que a primeira escolha do guloso foi a moeda de 25 (portanto $n \geq 25$). Suponha, também, para

[2]A subestrutura ótima, em essência, é o "passo" da indução.

efeito de contradição, que a primeira moeda da solução ótima *não* é de 25. Sendo assim, podemos verificar, por inspeção, que os primeiros 25 centavos do troco, (a) ou estão cobertos por uma seqüência de três moedas de 10 (somando 30), (b) ou estão cobertos por um conjunto, de pelo menos três moedas, que somam exatamente 25. Ora, no caso (b) podemos substituir o conjunto de moedas que somam 25 por uma moeda de 25, e obter uma solução com *menos* moedas, uma contradição. No caso (a) podemos substituir as três moedas de 10 por uma de 25 e uma de 5; novamente obtendo uma contradição. Veja a ilustração na figura 8.

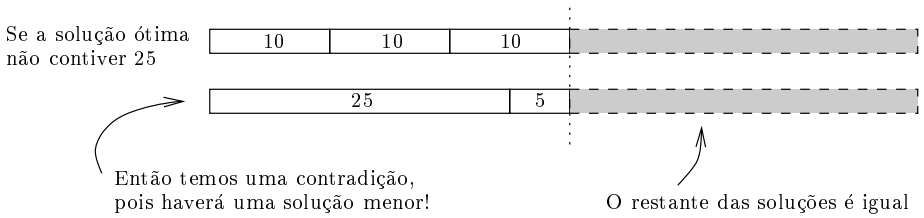


Figura 8: Demonstração da correção do guloso para o problema do Troco

5 Passatempo

1. Você está com um cachorro, um gato e um rato na margem esquerda de um rio e precisa levar todos os animais (vivos!) para a margem direita. Há uma ponte e você só pode atravessá-la carregando um animal por vez. Entretanto, o cachorro mata o gato se eles ficarem sozinhos de um lado da ponte; e o gato, por sua vez, mata o rato se eles também ficarem sozinhos. É possível atravessar?

2. A banda U2 vai começar um show daqui a 17 minutos. Os quatro integrantes estão na margem esquerda de um rio e precisam cruzar uma ponte para chegar ao palco na margem direita. É noite. Há somente uma lanterna. Só podem passar uma ou duas pessoas juntas pela ponte, e sempre com a lanterna. A lanterna não pode ser jogada, etc... **Não existe pegadinha neste problema!**

   Cada integrante possui um tempo diferente para atravessar a ponte: o Bono leva 1 minuto, o Edge 2 minutos, o Adam 5 minutos e o Larry 10 minutos. Quando dois atravessam juntos, eles vão pelo tempo do mais lento. Você precisa ajudá-los a atravessar. Por exemplo: se você mandar o Bono e o Larry passarem juntos, vai levar 10 minutos para eles chegarem ao palco. Se depois o Larry retornar com a lanterna, levará mais 10 minutos, 20 minutos terão passado e você falhou.

   (a) Encontre uma solução para o problema.

   (b) Vamos generalizar: agora há n pessoas do mesmo lado da ponte com diversos tempos de travessia (não necessariamente distintos) e uma lanterna.

Defina um critério guloso para resolver esse problema generalizado. Depois argumente que o algoritmo baseado no seu critério encontra, de fato, uma seqüência ótima; ou apresente um contra-exemplo para mostrar que seu critério não necessariamente leva ao ótimo.

3. O professor Midas dirige um automóvel de Porto Alegre para Fortaleza pela BR-116. Quando o tanque do carro dele está cheio, a autonomia é de n quilômetros. Ele tem um mapa que dá as distâncias entre os postos de gasolina em todo o trajeto. O objetivo dele é fazer o menor número possível de paradas para abastecer. Assuma que é possível viajar pelo trajeto com o carro dele; quer dizer, todo par consecutivo de postos no trajeto está separado por uma distância menor ou igual a n. Descreva um método guloso pelo qual o professor pode determinar em quais postos ele deve parar. Depois, tente achar um argumento que mostre que sua estratégia gulosa leva, de fato, à melhor solução.

6 Referência

1. Cormen, Leiserson, Rivest e Stein. *Introduction to Algorithms*. Segunda edição, MIT Press, 2001.

   Este livro, que é uma referência tradicional em cursos de Análise de Algoritmos, contém um excelente capítulo sobre Algoritmos Gulosos (há uma tradução para o português da Editora Campus). O livro é um bom ponto de partida para quem quiser se aprofundar no assunto. Algoritmos gulosos são, digamos, a superfície de um assunto muito interessante: Matróides.