Método Guloso

Notas de aula da disciplina IME 04-10823
ALGORITMOS E ESTRUTURAS DE DADOS II

Paulo Eustáquio Duarte Pinto
(pauloedp arroba ime.uerj.br)

junho/2012

Troco mínimo

"Dados os tipos de moedas de um país, determinar o número mínimo de moedas para dar um troco de valor **n**."

Exemplo: M = {1, 5, 10, 25, 50, 100} n = 37

O número mínimo de moedas é 4:

T(37) = 1 + T(37 - 25) = 1 + 1 + T(12 - 10) =
2 + 1 + T(2 - 1) = 3 + T(1) = 4

Basta, a cada passo, usar a moeda de maior valor possível.

Troco mínimo

Mas...
nem sempre essa estratégia funciona!

Exemplo: M = {1, 5, 12, 24, 50, 100} n = 20

Usando a estratégia anterior:

T(20) = 1 + T(20 - 12) = 1 + 1 + T(8 - 5) =
2 + 1 + T(3 - 1) = 3 + 1 + T(2 - 1) =
4 + T(1) = 5

O resultado é ERRADO, pois é possível dar um troco usando 4 moedas de 5.

Troco mínimo

Quando o método guloso funciona, o algoritmo é, em geral, eficiente.

Figurativamente, a solução gulosa consiste em, a cada passo, escolher o melhor pedaço possível e não se arrepender.

Para saber se o guloso funciona, é necessário PROVAR que o algoritmo resolve o problema.

Porque o método funciona com as moedas brasileiras?

M = {1, 5, 10, 25, 50, 100}

1. A tabela abaixo mostra o máximo de moedas de cada tipo usado em um troco mínimo, pois, para cada aumento nesses valores, existe outra opção com menos moedas. Adicionalmente, não se pode usar simultâneamente 2 moedas de 10 e 1 de 5.

1(1)	2(5)	3(10)	4(25)	5(50)	6(100)
4	1	2	1	1	∞

2. O valor máximo conseguido com as moedas tipos 1 a 5 é 99. Logo, qualquer troco x > 99 usa tantas moedas de 100 quanto necessário, transformando o problema para um troco y = x - 100*⌊x/100⌋ < 100.

3. O valor máximo conseguido com as moedas tipos 1 a 4 é 49. Logo, qualquer troco x, 100 > x > 49, usa 1 moeda de 50, transformando o problema para um troco y = x - 50 < 50.

Porque o método funciona com as moedas brasileiras?

M = {1, 5, 10, 25, 50, 100}

4. O valor máximo conseguido com as moedas tipos 1 a 3 é 24. Logo, qualquer troco x, 50 > x > 24, usa 1 moeda de 25, transformando o problema para um troco y = x - 25 < 25.

5. O valor máximo conseguido com as moedas tipos 1 e 2 é 9. Logo, qualquer troco x, 25 > x > 9, usa 1 ou 2 moedas de 10, transformando o problema para um troco y = x - 10*⌊x/10⌋ < 10.

6. O valor máximo conseguido com as moedas do tipo 1 é 4. Logo, todo valor x, 10 > x > 4 usa 1 moeda de 5.

Conclusão: o troco mínimo obtido pelas considerações anteriores é exatamente aquele obtifo vom o algoritmo guloso. Logo, o método guloso funciona corretamente para esse conjunto de moedas.

Troco mínimo - Conjunto Guloso de Moedas

Teorema 1: (Cowen, Cowen & Steinberg)

"Suponha que $C_1 = \{a_1, a_2, \ldots a_k\}$ seja um conjunto de moedas guloso e seja $C_2 = \{a_1, a_2, \ldots a_k, a_{k+1}\}$ e $m = \lceil a_{k+1}/a_k \rceil$. Então C_2 é guloso sse $G(C_2, m.a_k) \leq m$".

Obs:

1) $G(C,m)$ é o número mínimo de moedas para um troco n, usando C .
2) O conjunto $C_1 = \{1\}$ é guloso, pois só gera uma solução.
3) Os demais conjuntos são gulosos, pois:
   - $C_2 = \{1,5\}$ m = 5 e $G(C_2,5) = 1 \leq 5$.
   - $C_3 = \{1,5,10\}$ m = 2 e $G(C_3,10) = 1 \leq 2$.
   - $C_4 = \{1,5,10,25\}$ m = 3 e $G(C_4,30) = 2 \leq 3$.
   - $C_5 = \{1,5,10,25,50\}$, m = 2 e $G(C_5,50) = 1 \leq 2$.
   - $C_6 = \{1,5,10,25,50,100\}$, m = 2 e $G(C_6,100) = 1 \leq 2$.

Troco mínimo

No segundo exemplo, a solução é por PD:

M = {1, 5, 12, 24, 50, 100} n = 20

T(20) = min(T(20-12), T(20-5), T(20-1))+1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	0	1	2	3	4	1	2	3	4	5	2	3	4	5	6	3	4	5	6	7	4
3	0	1	2	3	4	1	2	3	4	5	2	3	1	2	3	3	4	2	3	4	4

Compactação de Dados: Huffman(1952)

O objetivo da compactação de dados é diminuir o tamanho de uma mensagem codificada.

Normalmente os métodos de compactação de dados usam códigos de tamanho variável, para atribuir códigos pequenos para símbolos frequentes e códigos maiores para símbolos raros.

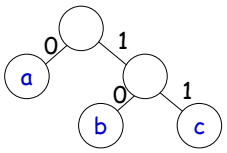
A vantagem de um código prefixos é que não existe ambiguidade na decodificação de dados.

Compactação de Dados: Huffman

O código de Huffman é um código ótimo de prefixos de tamanho variável, que utiliza uma árvore na criação do código e na decodificação.

Ex: a = 0; b = 10; c = 11;
abcbbac é codificado como 010111010011

Árvore de Huffman



Compactação de Dados: Huffman

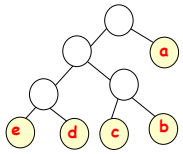
Cria um código de prefixos de tamanho variável, usando um algoritmo gulosocom complexidade O(n. log n).

Árvores de Huffman: árvores estritamente binárias enraizadas, com as codificações nas folhas, usadas na compactação e na descompactação.

Exemplo:

Símbolo/Freq.	Codificação
(a, 26)	1
(b, 15)	011
(c, 10)	010
(d, 10)	001
(e, 8)	000

Árvore de Huffman

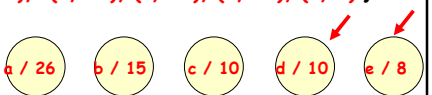


Compactação de Dados: Huffman

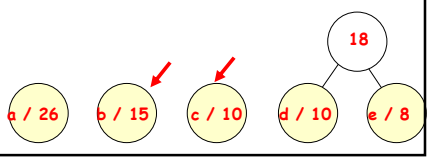
Algoritmo de Huffman: inicia com uma floresta de folhas, correspondentes aos símbolos e aglutina, sucessivamente, subárvores com soma total de frequências mínima.

Exemplo: {(a, 26), (b, 15), (c, 10), (d, 10), (e, 8) }

Passo 1



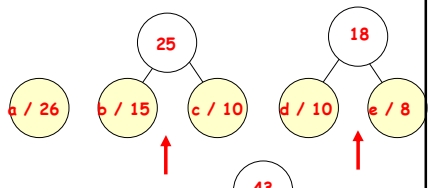
Passo 2



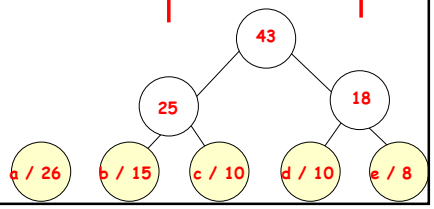
Compactação de Dados: Huffman

Exemplo: {(a, 26), (b, 15), (c, 10), (d, 10), (e, 8) }

Passo 3



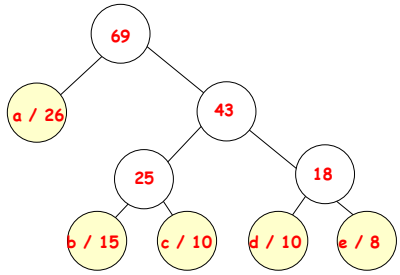
Passo 4



Compactação de Dados: Huffman

Exemplo: {(a, 26), (b, 15), (c, 10), (d, 10), (e, 8) }

Passo Final



Tamanho médio do codificação: =

$\Sigma\ d_i f_i\ /\ \Sigma f_i\ =\ (26x1+15x3+10x3+10x3+8x3)/69 = 2.24$

Compactação de Dados: Huffman

Exercício:

Criar a árvore de Huffman para a seguinte situação:

(a, 35)
(b, 26)
(c, 13)
(d, 13)
(e, 7)
(f, 6)

Calcular o tamanho médio da codificação.

Compactação de Dados: Huffman

Idéia do algoritmo: criar a árvore com o auxílio de um Heap

Algoritmo:

```
CriaFolhas;  CriaHeap (H);
Para i de 1 a n-1:
    p ← H[1].arv;
    Troca(1, n-i+1);  DesceHeap(1, n-i);
    q ← H[1].arv;
    Aloca(r);  r↑.le ← p;  r↑.ld ← q;
    r↑.f ← p↑.f + q↑.f;
    H[1].arv ← r;
    DesceHeap(1, n-i);
Fp;
T ← H[1].arv;
Fim;
```

Complexidade: O(n.log n)

Compactação de Dados: Huffman

Correção do Algoritmo

Lema: Para qualquer árvore de Huffman ótima existe outra árvore equivalente onde os 2 símbolos de menor frequência são irmãos.

Teorema: O algoritmo de Huffman é correto.

Compactação de Dados: Huffman

Correção do Algoritmo

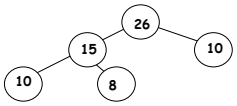
Teorema: O algoritmo de Huffman é correto.

Prova: Indução em n (número de símbolos).
Seja T_H uma árvore de Huffman e T_O uma árvore ótima. Em T_H, os 2 símbolos de menor frequência, f_1 e f_2 são irmãos. Caso em T_O esses símbolos não sejam irmãos, podemos remanejar para que sejam. Consideremos, respectivamente, as árvores T_H' e T_O' para n-1 símbolos onde os dois símbolos de menor frequência foram fundidos. Por hipótese, T_O e T_O' são ótimas.
Seria absurdo termos $c(T_O) > c(T_O') + f_1 + f_2$, pois poderíamos construir, a partir de T_O', uma árvore para n símbolos com custo menor que o de T_O. De forma análoga, seria absurdo $c(T_O) < c(T_O') + f_1+f_2$, pois agora T_O' é que não seria ótima. Portanto, $c(T_O) = c(T_O')+f_1+f_2$.
Mas $c(T_H) = c(T_H')+f_1+f_2$, por construção.
Segue-se que $c(T_H) = c(T_O)$ e, portanto, T_H também é ótima.
Logo, o algoritmo é correto.

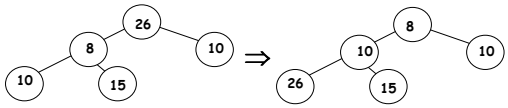
Compactação de Dados: Huffman

Situação do heap na execução de Huffman

Exemplo: {(a, 26), (b, 15), (c, 10), (d, 10), (e, 8) }



a) Criação do Heap, usando DesceHeap:

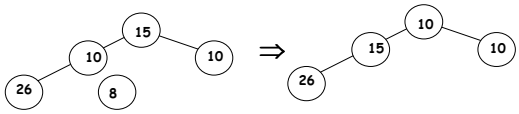


Compactação de Dados: Huffman

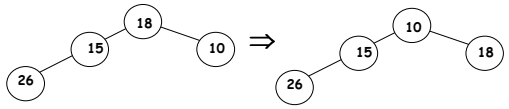
Situação do heap na execução de Huffman

b) Passos para fusão das subárvores:

b.1) Primeira fusão: retira o menor:



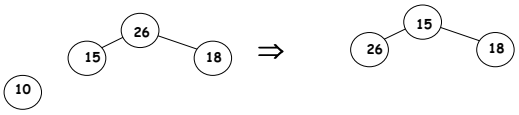
b.1) Primeira fusão: substitui o segundo menor pela soma 8 + 10:



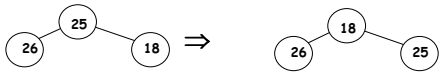
Compactação de Dados: Huffman

Situação do heap na execução de Huffman

b.2) Segunda fusão: retira o menor:



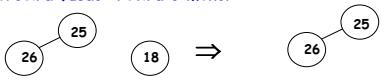
b.2) Segunda fusão: substitui o segundo menor pela soma 10+15:



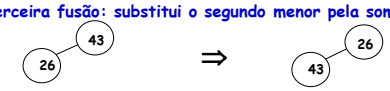
Compactação de Dados: Huffman

Situação do heap na execução de Huffman

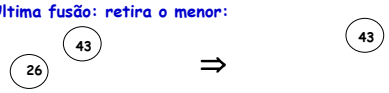
b.3) Terceira fusão: retira o menor:



b.3) Terceira fusão: substitui o segundo menor pela soma 18 + 25:



b.4) Última fusão: retira o menor:



b.4) Última fusão: substitui o segundo menor pela soma 26+43:

Compactação de Dados: Huffman

Outro algoritmo, quando os dados já estão ordenados

```
Usa duas filas Q1 e Q2:
Algoritmo:
    Esvazia filas;
    Cria raízes para símbolos e enfila em Q1;
    Enquanto (filas não vazias);
       Se Q1 vazia e Q2 só tem um elemento, termina.
       Senão
            Obtem subárvores a1 e a2 de Q1 e Q2 com
                 menores frequências.
            Desenfila a1 e a2.
            Junta a3 = a1 + a2;
            Enfila a3 em Q2;
    Fe;
Fim;
```

A raiz da árvore está no começo de Q2.

Merge ótimo

"Dados n arquivos com tamanhos $t_1, t_2 \ldots t_n$, determinar a sequência ótima (menor número de operações) de merge dos mesmos, para transformar em 1 único arquivo."

Exemplo: { 300, 500, 150, 400, 200}

Fazendo merges da esquerda para a direita:
T(n) =(300+500)+(800+150)+(950+400)+(1350+200)=4650

Fazendo merges da direita para a esquerda:
T(n) =(200+400)+(600+150)+(750+500)+(1250+300)=4150

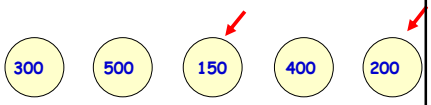
A solução ótima requer 3450 operações apenas !

Merge ótimo

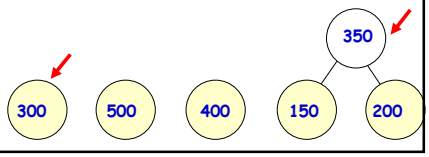
Merge ótimo: algoritmo e prova análogos a Huffman

Exemplo: {300, 500, 150, 400, 200 }

Passo 1



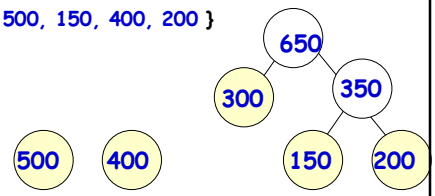
Passo 2



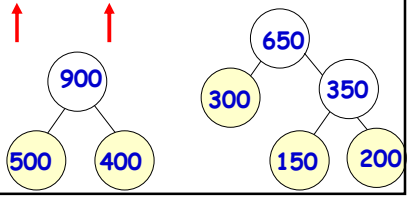
Merge ótimo

Exemplo: {300, 500, 150, 400, 200 }

Passo 3



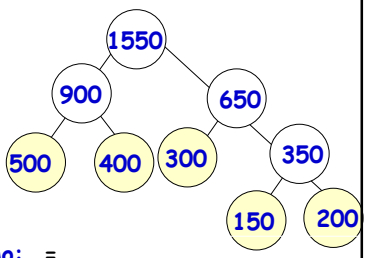
Passo 4



Merge ótimo

Exemplo: {300, 500, 150, 400, 200 }

Passo 5



Custo do merge ótimo: =

(150+200)+(350+300)+(500+400)+(900+650) = 3450

Merge ótimo

Exercício:

Calcular a melhor e a pior maneira de fazer o merge dos arquivos de tamanho:

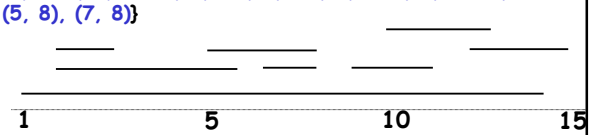
80
1000
300
100
90
150
410

Calcular o número de operações em cada caso.

Seleção de Tarefas

"Dadas n tarefas com datas de início e fim $(c_1, f_1), (c_2, f_2), \ldots (c_n, f_n)$, determinar o máximo de tarefas que podem ser executadas por 1 processador."

Exemplo:
{ (1, 14), (10, 13), (2, 3), (2, 6), (9, 11), (12, 15), (5, 8), (7, 8)}

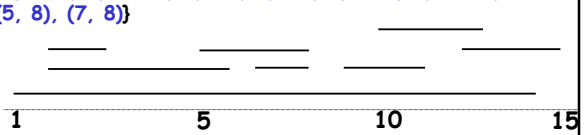


A solução ótima é selecionar 4 tarefas !

Seleção de Tarefas

Algoritmo: ordenar as tarefas por data de fim e selecionar, gulosamente, tal que cada tarefa em avaliação não conflite com o conjunto já escolhido

Exemplo:
{ (1, 14), (10, 13), (2, 3), (2, 6), (9, 11), (12, 15), (5, 8), (7, 8)}



Uma solução ótima é: (2, 3), (7, 8), (9, 11) e (12, 15)

Seleção de Tarefas

Algoritmo:

```
Ordenar tarefas por data fim;
S ← T[1];  r ← T[1].f;
Para i de 2 a n:
        Se (T[i].c  >  r) Então
         S ←  S + T[i];    r ← T[i].f;
Fp;
Fim;
```

Complexidade: O(n.log n)

Correção do Algoritmo

Teorema: O algoritmo de Seleção de Tarefas é correto.

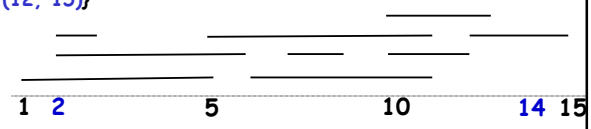
Prova: Seja S o conjunto encontrado pelo algoritmo e So um conjunto ótimo, ambos ordenados por data de fim. Seja j o primeiro índice tal que as tarefas dos 2 conjuntos sejam diferentes. Então podemos substituir a tarefa to_j de So pela t_j de S. Podemos fazer isso sucessivamente. Ao final não poderá sobrar nenhuma tarefa em So, pois o algoritmo teria selecionado essa tarefa. Logo, os dois conjuntos têm o mesmo número de elementos e, portanto, S também é ótimo.

Cobertura mínima de Segmentos

"Dados dois pontos a e b e n segmentos com extremos $(c_1, f_1), (c_2, f_2), \ldots (c_n, f_n)$, determinar o número mínimo de segmentos que cobre o intervalo (a, b)."

Exemplo: a = 2, b = 14

{ (1, 5), (6,11), (2, 6), (7, 9), (10, 12), (2, 3), (5,11), (12, 15)}



A cobertura mínima é de 4 segmentos !

Cobertura mínima de Segmentos

Exercício:

Escrever um algoritmo para determinar se o conjunto de n segmentos $S = \{(c_1, f_1), \ldots (c_n, f_n)\}$ cobre ou não o intervalo (a, b).

Cobertura mínima de Segmentos

Solução:

```
VerificaCobertura;
     Ordenar S por ci;
     p ← a;   cobre ← V;
     Para i de 1 a n:
        Se (p < b) Então
            Se (ci > p)  Então  cobre ← F
            Senão Se (fi > p)  Então  p ← fi;
     Fp;
     Se (p < b) Então cobre ← F
     Retornar cobre;
Fim;
```


Cobertura mínima de Segmentos

Exemplo: a=1, b=12

(S = {(3,6), (2,5), (4,9), (7,13), (5,6),(6,7), (8,12), (5,9),(4,8),(1,4),(3,4)}

Ordenando:

(S'= {(1,4),(2,5), (3,6), (3,4), (4,9), (4,8), (5,9), (5,6), (6,7), (7,13),(8,12)}

i	c_i	f_i	p	cobre
			1	1
1	1	4	4	1
2	2	5	5	1
3	3	6	6	1
4	3	4	6	1
5	4	9	9	1
6	4	8	9	1
7	5	9	9	1
8	5	6	9	1
9	6	7	9	1
10	7	13	13	1
11	8	12	13	1

Cobertura mínima de Segmentos

Determinação da cobertura, supondo-se que o conjunto de segmentos cobre o intervalo dado.

Idéia do Algoritmo:

Supõe-se que o conjunto de segmentos S cobre o intervalo (a, b) dado (ver exercício).
Ordena-se S pelos começos dos segmentos e, para pontos de referência, definidos em ordem crescente, seleciona-se os segmentos que cobrem esses pontos e têm o maior extremo direito.
Inicialmente o ponto de referência é a. Cada vez que se escolhe um segmento e acrescenta-se ao conjunto solução R, muda-se o ponto de referência para o final do segmento escolhido. O algoritmo pára quando o ponto de referência é ≥ b.

Cobertura mínima de Segmentos

Determinação da cobertura, supondo-se que o conjunto de segmentos cobre o intervalo dado.

```
CoberturaMinima;
    Ordenar S por c_i;  R ← Ø;
    c_0 ← -∞;  f_0 ← -∞;  c_{n+1} ← ∞;  f_{n+1} ← ∞;
    p ← a;    q ← 0;
    Para i de 1 a n+1:
      Se (c_i > p) Então
          R ← R + (c_q, f_q);  p ← f_q;    q ← i;
          Se (p ≥ b)  Então  Sair do loop;
      Senão Se (f_i > f_q) Então q ← i;
    Fp;
    Imprimir R;
Fim;
```


Cobertura mínima de Segmentos

Exercício:

Preencher a tabela de atribuição de valores às variáveis do algoritmo de Cobertura Mínima para os segmentos:

(S = {(3,6), (2,5), (4,9), (7,13), (5,6),(6,7), (8,12), (5,9),(4,8),(1,4),(3,4)}
a=1, b=12

i	c_i	f_i	p	q	R
0	-∞	-∞	1	0	Ø
1					
2					
.					
.					
10					
11					
12					

Cobertura mínima de Segmentos

Exercício:

Demonstrar a corretude do algoritmo de cobertura mínima de segmentos.

Sequenciamento de Tarefas com receita máxima

"Dadas n tarefas unitárias com datas limite de fim e receitas dadas, (l_1, r_1), (l_2, r_2),... (l_n, r_n), determinar a receita máxima que se pode ter, sabendo-se que a receita de uma tarefa só é considerada se ela for realizada dentro do tempo limite."

Exemplo:

Tarefa	T1	T2	T3	T4	T5	T6
l_i	1	1	2	3	3	4
r_i	7	8	4	6	5	10

A receita ótima é 29 !

Sequenciamento de Tarefas com receita máxima

Algoritmo: ordenar as tarefas, de forma decrescente por receita e selecionar, gulosamente, tal que cada tarefa em avaliação não conflite com o conjunto já escolhido.

Exemplo:

Tarefa	T5	T2	T1	T4	T6	T3
l_i	4	1	1	3	3	2
r_i	10	8	7	6	5	4

Seleciona T5

Tarefa	T5
l_i	4
r_i	10

Sequenciamento de Tarefas com receita máxima

Exemplo:

Tarefa	T5	T2	T1	T4	T6	T3
l_i	4	1	1	3	3	2
r_i	10	8	7	6	5	4

Seleciona T2

Tarefa	T2	T5
l_i	1	4
r_i	8	10

Descarta T1

Seleciona T4

Tarefa	T2	T4	T5
l_i	1	3	4
r_i	8	6	10

Seleciona T6

Tarefa	T2	T4	T6	T5
l_i	1	3	3	4
r_i	8	6	5	10

Descarta T3

Receita máxima = 8+6+5+10=29

Sequenciamento de Tarefas com receita máxima

Algoritmo:

Ordenar tarefas por receita;
S ← ∅;
Para i de 1 a n:
  Se (ViavelIncluir (S, T[i]) Então
    Incluir (S, T[i]);
Fp;
Fim;

Complexidade: $O(n^2)$

Incluir (S, T): inclui ordenadamente por l.

ViavelIncluir (S, T): verdadeiro se, à direita do ponto de inclusão nenhuma tarefa T_i está em posição j = l_i.

Sequenciamento de Tarefas com receita máxima

Correção do Algoritmo

Teorema: O algoritmo de Sequenciamento... é correto.

Prova: Seja S o conjunto encontrado pelo algoritmo e So um conjunto ótimo, ambos ordenados por data limite. Podemos remanejar as tarefas comuns tal que fiquem em mesma posição nos dois conjuntos . Seja t_i a tarefa de receita máxima de S que não está em So. Podemos substituir to_i por t_i, em So. Ao final do processo, $S \subseteq So$. Mas não podemos ter $S \subset So$ nem $So \subset S$. Portanto S = So, ou seja, S é ótimo.
Logo, o algoritmo é correto.

Sequenciamento de Tarefas com receita máxima

Exercício:

Indicar o sequenciamento e a receita ótima para as tarefas:

Tarefa	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
l_i	2	4	2	5	4	3	1	9	5	3
r_i	10	7	12	8	5	6	9	1	5	4

Sequenciamento de Tarefas com receita máxima

Exercício:

Indicar o sequenciamento e a receita ótima para as tarefas:

Solução:

Tarefa	T3	T1	T3	T2	T4	T8
l_i	2	2	3	4	5	9
r_i	12	10	6	7	8	1

Receita máxima = 12+10+6+7+8+1= 44

Sequenciamento de Tarefas c/ penalidade mínima

Dadas n tarefas unitárias com datas limite de fim e penalidades dadas, $(l_1, p_1), (l_2, p_2), \ldots (l_n, p_n)$, determinar a penalidade mínima para realizar todas as tarefas, sabendo-se que a penalidade se aplica quando a tarefa é realizada após o tempo limite.

Exemplo:

Tarefa	T1	T2	T3	T4	T5	T6
l_i	1	1	3	4	3	4
p_i	7	8	4	6	10	5

Sequenciamento de Tarefas c/ receita máxima (2)

Dadas **n** tarefas unitárias com datas limite de fim, receitas e penalidades dadas, $(l_1, r_1, p_1), (l_2, r_2, p_2), \ldots (l_n, r_n, p_n)$, determinar a receita máxima para realizar todas as tarefas, sabendo-se que a penalidade se aplica quando a tarefa é realizada após o tempo limite.

Exemplo:

Tarefa	T1	T2	T3	T4	T5	T6
l_i	1	1	3	4	3	4
r_i	20	8	14	6	10	15
p_i	7	8	4	6	10	5

Sequenciamento de Tarefas c/ penalidade mínima (2)

Dadas **n** tarefas unitárias com datas limite de fim e penalidades dadas, $(l_1, p_1), (l_2, p_2), \ldots (l_n, p_n)$, determinar a penalidade mínima para realizar todas as tarefas, sabendo-se que a penalidade se aplica quando a tarefa é realizada após o tempo limite, diariamente.

Exemplo:

Tarefa	T1	T2	T3	T4	T5	T6
l_i	3	2	1	1	5	4
p_i	5	10	6	9	3	5

Exercício: Escrever algoritmos para os seguintes problemas

Problema 1: Travessia

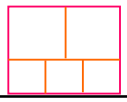
Tem-se n pessoas para atravessar uma ponte, numa noite escura, e uma única lanterna. No máximo duas pessoas podem atravessar de cada vez. São dados os tempos de travessia de cada um. Qual o tempo mínimo total de travessia?

Ex: n=4 tempos: 1, 2, 5, 10 tempo mínimo = 17.

Problema 2: Cortes quadrados

Tem-se uma chapa retangular de dimensões inteiras p x q e quer-se transformar esse retângulo no mínimo de quadrados, fazendo-se sempre cortes em toda a extensão da chapa. Qual o mínimo de quadrados?

Ex: corte de uma chapa 5 x 6:



FIM