MAC0338 - ANÁLISE DE ALGORITMOS

LISTA 6

CHECKLIST

entregue ▢ fiz ▢ incompleto ▢ não entendi ▢

- intervalos disjuntos

1 ▢

- coloração de intervalos

2 ▢

3 ▢

- escalonamento

4 ▢ ★

5 ▢

- problema da mochila fracionária

6 ▢

- coleção máxima de intervalos disjuntos

7 ▢

8 ▢ 9 ▢ 10 ▢

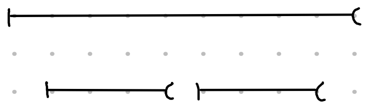
NÃO ESQUECER

- para criar critérios gulosos é interessante ordenar a entrada
- não se esquecer de seguir os passos para cada exercício
- o critério guloso ótimo pode ser um valor relativo
- pensar se a ordem das tarefas altera o prazo que termina.

1. Mostre um exemplo para cada um dos três primeiros critérios gulosos apresentados para o problema da coleção máxima de intervalos disjuntos visto em aula que prove que o algoritmo obtido usando estes critérios pode produzir um conjunto de intervalos disjuntos que não é máximo.

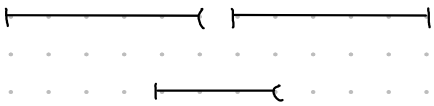
no PDF de estudos

- escolher o intervalo i em R com menor $s[i]$ (começam primeiro)



não funciona, pois joga fora 2 intervalos

- escolher o intervalo i em R com menor $f[i] - s[i]$ (menor tamanho)

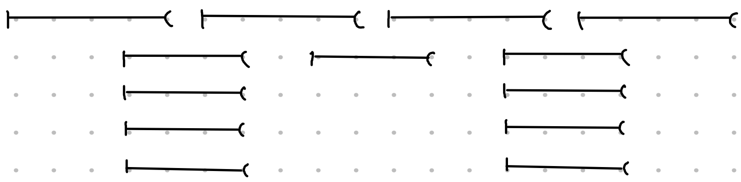


não funciona, pois joga fora 2 intervalos

- escolher o intervalo i tal que

$$|\{ j \in R : j \text{ intersecta } i \}|$$

é o menor possível (menor número de intersecções)



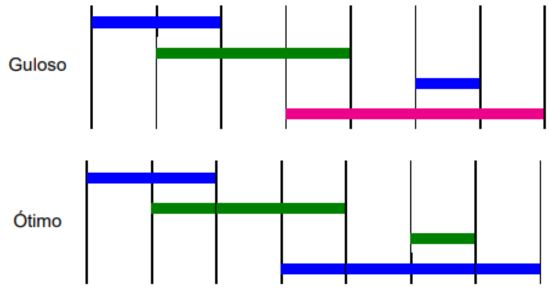
não funciona, pois descarta a primeira fileira.

2. Considere o algoritmo visto em aula para o problema da coloração de intervalos. Modifique-o para que, além da m-coloração c, ele devolva um conjunto S de m intervalos e um instante t tal que $s[i] \leq t < f[i]$ para todo i em S.

- devolve um conjunto S de m intervalos
- instante t tal que $s[i] \leq t < f[i]$

3. Considere o algoritmo do problema da coloração de intervalos visto em aula. Nele, os intervalos são ordenados no começo pelo valor de $s[i]$. O que acontece se ordenarmos os intervalos por $f[i]$ em vez de $s[i]$? O algoritmo continua correto? Prove, apresentando um certificado como o do exercício anterior para ele, ou dê um contra-exemplo.

- Ordene as atividades de maneira que $f[1] \leq f[2] \leq \dots \leq f[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.



não funciona

4. Seja $1, \ldots, n$ um conjunto de *tarefas*. Cada tarefa consome um dia de trabalho; durante um dia de trabalho somente uma das tarefas pode ser executada. Os dias de trabalho são numerados de 1 a n. A cada tarefa t está associado um *prazo* p_t: a tarefa deveria ser executada em algum dia do intervalo $1 \,..\, p_t$. A cada tarefa t está associada uma *multa* $m_t \geq 0$. Se a tarefa t é executada depois do prazo p_t, paga-se a multa m_t (mas a multa não depende do número de dias de atraso).

   Problema: Programar as tarefas (ou seja, estabelecer uma bijeção entre as tarefas e os dias de trabalho) de modo a minimizar a multa total.

   Considere o algoritmo brevemente descrito na aula, que primeiramente ordena as tarefas pela multa. A seguir, considera uma a uma cada tarefa, escalonando-a no último dia livre dentro do seu prazo. Se não houver dia livre dentro do prazo, escalona a tarefa no último dia livre.

   Prove que esse algoritmo está correto, ou seja, produz um escalonamento com multa mínima. Compare o consumo de tempo deste algoritmo e do algoritmo guloso visto em aula.

- conjunto de tarefas de 1 a n
- cada tarefa consome 1 dia de trabalho, dias de trabalho de 1 a n
- cada tarefa tem um prazo pt, deve ser executada em algum dia de 1... pt
- multa se ultrapassar o prazo pt., sem depender dos dias de atraso

OBJETIVO: programar as tarefas para diminuir a multa total

- ordena as tarefas pela multa
- coloca da maior para a menor no último dia do prazo
- se não tiver, coloca no dia disponível

5. A entrada é uma sequência de números $x_1, x_2, \ldots, x_n$ onde n é par. Projete um algoritmo que particione a entrada em $n/2$ pares da seguinte maneira. Para cada par, computamos a soma de seus números. Denote por $s_1, s_2, \ldots, s_{n/2}$ as $n/2$ somas. O algoritmo deve encontrar uma partição que minimize a máximo das somas e deve ser tão eficiente quanto possível. Explique porque ele funciona e determine a sua complexidade.

- entrada: sequência de números $x_1, x_2, \ldots, x_n$ onde n é par

OBSETIVO: particionar a entrada em $n/2$ pares e minimizar o máximo das somas.

(1) possivel critério guloso

- a sequência de números é ordenada tal que $X[1] \leq X[2] \leq \ldots \leq X[n]$ e cada par é formado com o primeiro e o último x_i disponível, assim, minimizamos a maior soma que é a do último número na sequência e repetimos o processo até o último par que estará no meio da sequência.
- é importante lembrar que esse critério tem subestrutura ótima.

(2) algoritmo

seja X o vetor dos números $x_1 \ldots x_n$ e S o conjunto de pares

```
MINIMIZA-SOMA (X, n)
ordena o vetor X em ordem crescente
i ← 1
j ← n
soma_maxima ← 0
enquanto i < j faça
    S ← S ∪ {X[i], X[j]}
    se soma_maxima < X[i] + X[j]
        soma_maxima ← X[i] + X[j]
    i ← i + 1
    j ← j + 1
devolva S e soma_maxima
```

complexidade: $O(n \lg n)$

Sabrina Araújo da Silva nUSP: 12566182

MAC0338 - ANÁLISE DE ALGORITMOS

LISTA 6

exercícios 6 e 10

6. Descreva um algoritmo eficiente que, dado um conjunto $\{x_1, x_2, \dots, x_n\}$ de pontos na reta real, determine o menor conjunto de intervalos fechados de comprimento um que contém todos os pontos dados. Justifique informalmente o seu algoritmo e analise a sua complexidade.

seja A uma coleção de intervalos ótimos do conjunto $X = \{x_1, x_2, \dots, x_n\}$

```
INTERVALO-OTIMO (X, n
ordene X de forma que X[1] ≤ X[2] ≤ ... ≤ X[n]
A ← ∅
enquanto X ≠ ∅ faça
    escolha por um critério guloso um intervalo i de X
    A ← A ∪ {i}
    X ← X \ {j ∈ X : j intersecta i}
devolva A
```

Agora, é preciso encontrar o critério guloso para preencher o algoritmo.

possível critério guloso:

(1) com base no primeiro ponto do conjunto determinar o intervalo que ele irá fazer parte. Por exemplo: $X[1] = 0{,}8$, então todos os $X[i]$ seguintes tal que $0{,}8 \leq X[i] \leq 1{,}8$ estarão no intervalo. Assim, esses pontos do conjunto serão retirados e o algoritmo irá repetir para os pontos restantes.

Portanto, temos:

```
INTERVALO-OTIMO (X, n)
0   ordene X de forma que X[1] ≤ X[2] ≤ ... ≤ X[n]
1   A ← ∅
2   i ← 1
3   inicio ← 0
4   fim ← 0
5   enquanto X ≠ ∅ faça
6       inicio ← X[i]
7       fim ← X[i] + 1
8       enquanto i ≤ n e X[i] ≤ fim
9           intervalo ← intervalo ∪ {X[i]}
10          i ← i + 1
11      A ← A ∪ intervalo
12      X ← X \ {j ∈ X : j intersecta intervalo}
13      intervalo ← ∅
14  devolva A
```

ALGORITMO

O algoritmo ordena todos os pontos do menor para o maior. Com base no primeiro ponto dessa ordem, o início e o fim do intervalo são definidos e todos os pontos do conjunto tal que inicio $\leq X[i] \leq$ fim são encaixados nesse intervalo. Em seguida, o intervalo é colocado na coleção A de intervalos e cada item que foi encaixado nele é retirado de X. O intervalo é resetado e o algoritmo repete para as instâncias menores.

O algoritmo funciona, porque todos os pontos têm que estar necessariamente em um intervalo e ao pegar o início dos intervalos como o valor do primeiro ponto disponível garante que no intervalo de X[i] estará todos os pontos possíveis para aproveitar o intervalo.

COMPLEXIDADE: a linha 0 consome tempo $O(n \lg n)$ para ordenar; linhas 1 a 4: $O(1)$; linhas 5 a 13: $O(n)$. Portanto, o algoritmo consome tempo $O(n \lg n)$

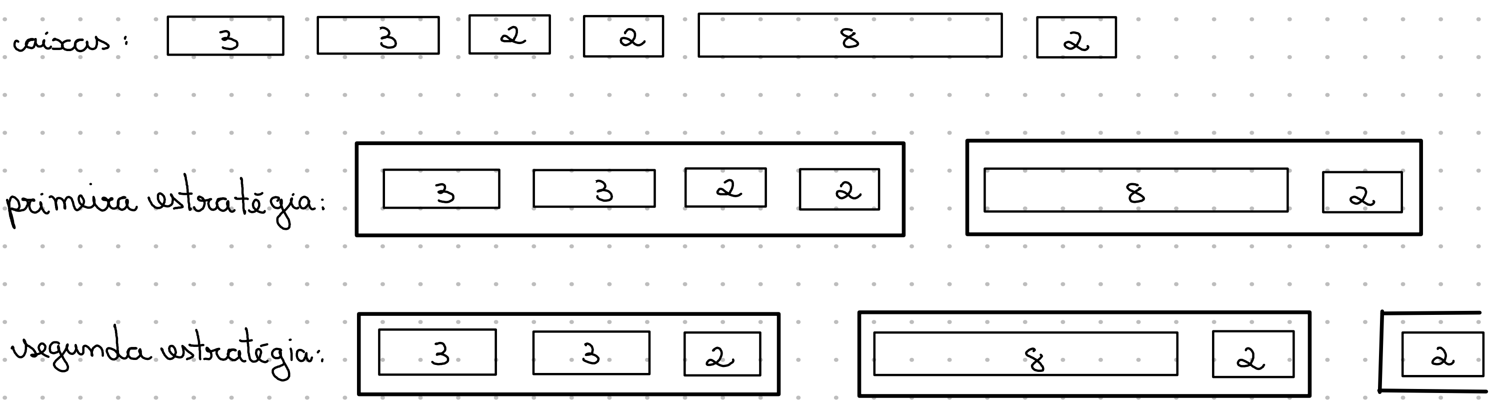
7. Você foi contratado como consultor para uma companhia de caminhões que faz uma grande quantidade de entregas entre Rio e São Paulo. O volume de entregas é grande o suficiente para que a companhia tenha que enviar um número de caminhões entre as duas localidades. Os caminhões têm uma quantia máxima de peso W que eles podem carregar. Caixas chegam a São Paulo uma a uma, e cada caixa i tem um peso w_i. A estação de caminhões é pequena, então no máximo um caminhão pode estar na estação por vez. A política da companhia é que as caixas que chegam primeiro são enviadas primeiro; do contrário um cliente pode ficar chateado ao saber que uma caixa que chegou depois da dele foi entregue primeiro no Rio. No momento, a companhia está usando uma estratégia gulosa para carregar os caminhões: eles colocam no caminhão as caixas na ordem em que chegam e, quando a próxima caixa não cabe no caminhão, eles liberam o caminhão para seguir viagem.

Mas a companhia está questionando se não estão usando muitos caminhões e querem a sua opinião sobre um possível modo de melhorar a situação. Eles estão pensando no seguinte. Talvez eles possam diminuir o número de caminhões necessários mandando algumas vezes um caminhão menos cheio de modo que os próximos caminhões estejam melhor carregados.

Prove que, dada uma sequência de caixas com os seus pesos especificados, o método guloso correntemente em uso minimiza o número de caminhões usados. Sua prova deve seguir o tipo de análise usado na coleção máxima de intervalos disjuntos: estabeleça que a solução gulosa é ótima identificando uma medida sobre a qual o algoritmo guloso fica "sempre à frente".

O método guloso em vigor funciona, porque todas as caixas possíveis que cabem no caminhão e que chegaram primeiro serão enviadas, assim, aproveitando ao máximo o espaço e respeitando a política da empresa. Caso siga o método de encher menos um caminhão para poder encher outro, pode acontecer o seguinte exemplo: um caminhão de capacidade 10 tem que levar as seguintes caixas: 3, 3, 2, 2, 8. A empresa decide encher o primeiro caminhão com 3, 3 e 2 para poder encher ao máximo o seguinte. Quando o primeiro caminhão sai da estação, chega uma caixa com peso 2, que não cabe no próximo. Nesse caso, a empresa deve usar 3 caminhões para realizar o transporte, quando poderia usar apenas 2 na seguinte ordenação: 1º com 3, 3, 2, 2 e 2º com 8, 2.

caminhão tem capacidade 10

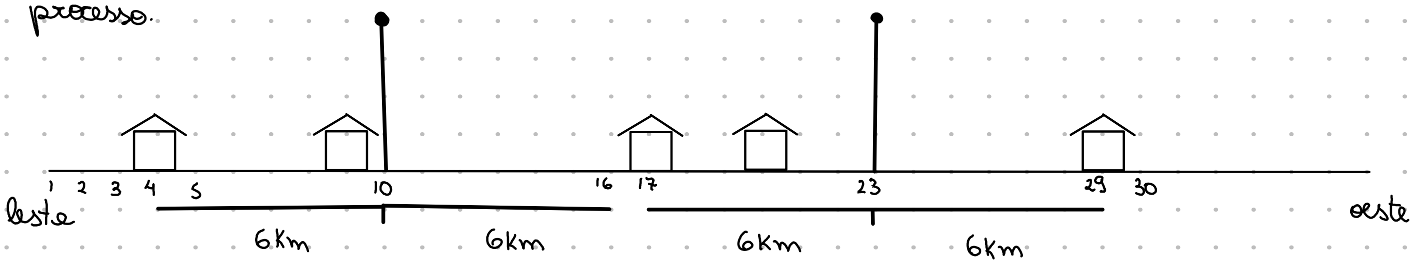


8. Considere uma estrada calma e longa, com algumas poucas casas à beira. (Podemos imaginar a estrada como uma linha reta, com uma extremidade leste e uma extremidade oeste.) Suponha que, apesar da atmosfera bucólica, os moradores dessas casas são ávidos usuários de telefones celulares. Você quer colocar estações-base de telefonia celular ao longo da estrada, de maneira que toda casa esteja a no máximo 6 quilómetros de uma das estações-base.

Dê um algoritmo eficiente que atinge esse objetivo usando um número mínimo de estações-base. Justifique sua resposta.

(1) possível estratégia gulosa

a partir da primeira casa no lado oeste percorrer 6Km e instalar a torre, percorrer mais 6Km e procurar a primeira casa após esse percurso e repetir o processo.



(2) algoritmo

seja q o tamanho da estrada, n o número de casas

seja C um vetor tal que C[1] é a distância da casa 1 até o início da estrada

seja E uma coleção de distâncias que as estações devem estar do início da estrada.

```
ESTACOES (n, C)
local_estacao ← C[1] + 6
E ← E ∪ local_estacao
final ← local_estacao + 6
para i ← 2 até n faça
    se C[i] > final então
        local_estacao ← C[i] + 6
        E ← E ∪ local_estacao
        final ← local_estacao + 6
devolva E
```

Esse algoritmo funciona, pois obrigatoriamente a primeira casa deve estar na região de cobertura, então colocamos essa casa no limite esquerdo para otimizar e aproveitar ao máximo o espaço restante, assim, abrangendo mais casas, e fazendo o mesmo para as instâncias restantes.

9. Seu amigo está trabalhando como coordenador de atividades do CEPÊ, e ele foi escolhido para organizar um mini-triatlon, onde cada participante deve nadar 20 voltas na piscina, depois correr 15km de bicicleta, e finalmente correr 5km. O plano é mandar os competidores, que não são muitos, em diferentes estágios, de acordo com a seguinte restrição: apenas uma faixa da piscina pode ser usada para o mini-triatlon, ou seja, apenas um competidor pode nadar de cada vez. O primeiro participante é liberado para nadar suas 20 voltas e, apenas quando ele termina, um segundo competidor é autorizado a nadar suas 20 voltas. Competidores podem fazer o percurso de bicicleta e a corrida ao mesmo tempo sem problemas. Apenas o uso da piscina tem essa restrição particular.

   Cada competidor tem um tempo esperado que leva para nadar suas 20 voltas, um tempo esperado para fazer 15km de bicicleta e um tempo esperado para correr 5km. Seu amigo, de posse dessa informação, deve montar um escalonamento dos competidores, que diz em que ordem os competidores iniciam sua prova, que minimize o término previsto. O término previsto da competição é o primeiro instante em que todos os competidores terminaram as três fases da prova considerando seu tempo esperado em cada prova.

(1) possíveis critérios gulosos

- o primeiro a nadar é o que demora mais nadando

	nado	corrida + bike		guloso	ótimo
participante 1	30	2	=	32	42
participante 2	10	80	=	120	90
				1 primeiro	2 primeiro

não funciona

- o primeiro a nadar é o que demora menos

	nado	corrida + bike	guloso	ótimo
part. 1	30	80	130	110
part. 2	20	40	60	90
			2 primeiro	1 primeiro

não funciona

- o primeiro a nadar é quem tem o maior valor relativo tempo total / por 2 (número de atividades) (tempo médio)

	nado	corrida	relativo	guloso	ótimo
part. 1	70	50	60	120	150
part. 2	30	70	50	170	100

(nem faz sentido)

não funciona

- o primeiro a nadar é quem demora mais de bicicleta e andando

	nado	corrida + bike	guloso
part 1.	15	40	55
part 2.	80	20	115

parece funcionar

O quarto critério guloso encontrado parece funcionar. Sabemos que o tempo de nado total sempre será o mesmo independente da ordem dos competidores. Portanto, a ideia desse critério guloso é otimizar o tempo que as outras atividades levam para serem completadas. Quanto mais cedo o competidor que leva mais tempo para andar de bicicleta e correr começar, mais cedo ele terminará as tarefas.

PROVA DO ALGORITMO

- seja i e j dois competidores tal que o tempo c_i de corrida e bicicleta de i é maior que c_j.
- na solução gulosa i vai primeiro

10. Um pequeno negócio, digamos, uma lojinha de xerox com uma única máquina de xerox, enfrenta o seguinte problema de escalonamento. Toda manhã, eles recebem uma coleção de tarefas de seus clientes. O dono do negócio quer executar essas tarefas em sua máquina, numa ordem que mantenha os seus clientes tão satisfeitos quanto possível. A tarefa do cliente i leva t_i unidades de tempo para ser completada. Dado um escalonamento (ou seja, uma ordem das tarefas), seja C_i o momento em que a tarefa do cliente i terminou de ser executada. Suponha ainda que cada cliente i tenha uma importância para o negócio, dada pelo número w_i. A satisfação do cliente i é dependente do tempo C_i em que sua tarefa é completada. O dono do negócio deseja determinar um escalonamento que minimize a soma ponderada $\sum_{i=1}^{n} w_i C_i$.

Projete um algoritmo eficiente para resolver esse problema. Os dados são a duração $t_1, \dots, t_n$ das n tarefas e a importância $w_1, \dots, w_n$ de n clientes. Seu algoritmo deve produzir uma ordem das n tarefas que minimize a soma ponderada $\sum_{i=1}^{n} w_i C_i$. Mostre que seu algoritmo produz uma resposta correta.

Exemplo: Considere $n = 2$, $t_1 = 1$ e $w_1 = 10$, $t_2 = 3$ e $w_2 = 2$. Se a primeira tarefa for executada primeiro, o valor da solução é $10 \cdot 1 + 2 \cdot 4 = 18$, enquanto que se a segunda tarefa for executada primeiro, é $10 \cdot 4 + 2 \cdot 3 = 46$.

Seja A um vetor que armazena a ordem que as tarefas devem ser executadas, T o vetor da duração das tarefas e W o vetor da importância das tarefas.

Possíveis estratégias gulosas:

(1) ordernar as tarefas tal que $W[1] \geq W[2] \geq \dots \geq W[n]$ e executar primeiro as tarefas de maior importância.

i_1 $\boxed{}$ $t_1 = 5$, $w = 10$

i_2 $\boxed{}$ $t_2 = 1$, $w = 3$

executando i_1 e depois i_2 $\sum_{i=1}^{n} w_i C_i = 5 \cdot 10 + 6 \cdot 3 = 68$

executando i_2 e depois i_1 $\sum_{i=1}^{n} w_i C_i = 1 \cdot 3 + 6 \cdot 10 = 63$

=> não funciona

(2) ordernar as tarefas tal que $T[1] \leq T[2] \leq \dots \leq T[n]$ e executar as tarefas de menor duração primeiro.

i_1 $\boxed{}$ $t_1 = 5$, $w = 20$

i_2 $\boxed{}$ $t_2 = 1$, $w = 1$

executando i_1 e depois i_2 $\sum_{i=1}^{n} w_i C_i = 5 \cdot 20 + 6 \cdot 1 = 106$

executando i_2 e depois i_1 $\sum_{i=1}^{n} w_i C_i = 1 \cdot 1 + 6 \cdot 20 = 121$

=> não funciona

(3) ordernar as tarefas tal que $W[1]/T[1] \geq W[2]/T[2] \geq \ldots \geq W[n]/T[n]$

e executar primeiro as tarefas com maior importância por tempo primeiro.

i_1 $t_1 = 5$, $w = 20$, $w/t = 4$

i_2 $t_2 = 1$, $w = 1$, $w/t = 1$

executando i_1 e depois i_2 $\sum_{i=1}^{n} w_i C_i = 5 \cdot 20 + 6 \cdot 1 = 106$

executando i_2 e depois i_1 $\sum_{i=1}^{n} w_i C_i = 1 \cdot 1 + 6 \cdot 20 = 121$

i_1 $t_1 = 5$, $w = 10$, $w/t = 2$

i_2 $t_2 = 1$, $w = 3$, $w/t = 3$

executando i_1 e depois i_2 $\sum_{i=1}^{n} w_i C_i = 5 \cdot 10 + 6 \cdot 3 = 68$

executando i_2 e depois i_1 $\sum_{i=1}^{n} w_i C_i = 1 \cdot 3 + 6 \cdot 10 = 63$

=> essa estratégia gulosa parece funcionar

ALGORITMO

ORDEM-ÓTIMA (W, T, A)

1 ordena W e T de modo que $W[1]/T[1] \geq W[2]/T[2] \geq \ldots \geq W[n]/T[n]$

2 para $i \leftarrow 1$ até n faça

3     $A[i] \leftarrow$ tarefa armazenada em $W[i]$

4 devolva A

O algoritmo funciona com a estratégia gulosa escolhida, porque dá preferência às tarefas de maior importância por tempo, assim, a lojinha de xerox estará executando sempre a tarefa que tem mais satisfação por tempo de trabalho.

- consumo de tempo da ordenação na linha 1 é $\Theta(n \lg n)$
- consumo de tempo nas linhas 2 a 4 é $\Theta(n)$
- consumo de tempo total é $\Theta(n \lg n)$