Técnicas de projeto de algoritmos

→ não existe fórmula mágica para resolver problemas.

→ Principais paradigmas:
- Divisão e conquista
- Algoritmos gulosos (greedy)
- Programação dinâmica
- Recursão
- Redução

Problemas de otimização

→ Dadas algumas restrições, vários soluções são viáveis.

→ Dada uma função que atribui um valor a cada solução viável, qual é uma melhor solução?
↳ Pode ser uma que minimize ou maximize o valor.
↳ Solução ótima

PROBLEMA ≡ INSTÂNCIA DO PROBLEMA
SUBPROBLEMA ≡ INSTÂNCIA MENOR DO PROBLEMA

→ As análises que veremos envolvem manipular uma solução ótima sem conhecê-la.

→ A corretude de um algoritmo envolverá:
↳ mostrar que ele sempre devolve solução viável.
↳ mostrar se ele é ótimo ou não.

→ Existe um conj. finito porém muito grande de soluções viáveis
↳ As variáveis possuem domínios discretos
↳ Admitem, portanto, algoritmos de força bruta.

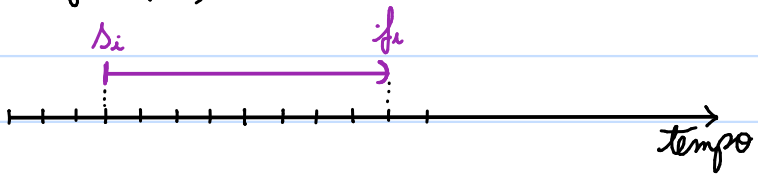
Algoritmos gulosos

→ Constroem soluções por uma sequência de decisões que visam o melhor cenário de curto prazo, sempre tendo em vista o objetivo do problema.

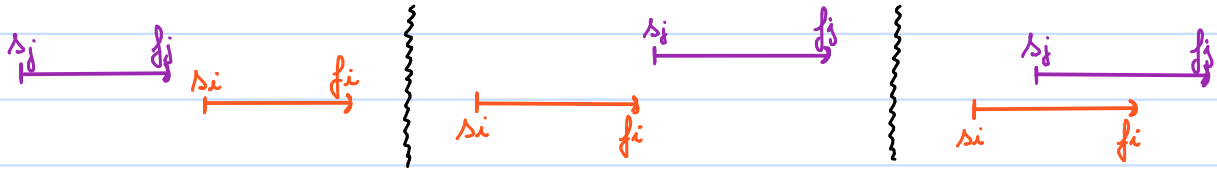
→ Não há garantias que isso levará a uma solução ótima
↳ Muitas vezes, na verdade, não leva!

Um primeiro problema

→ Seja t_i uma tarefa que, se selecionada, acontece no intervalo $[s_i, f_i)$.

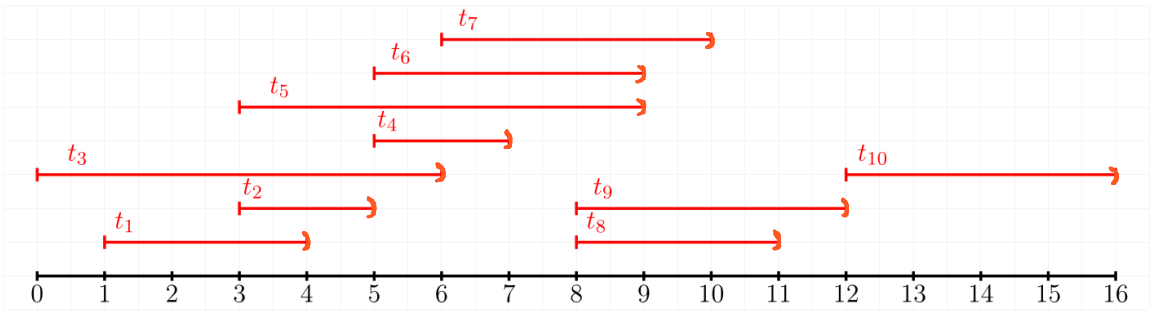


→ Duas tarefas t_i e t_j são compatíveis se esses intervalos não se sobrepõem, isto é, se $s_i \geq f_j$ ou se $s_j \geq f_i$



Problema *Escalonamento de Tarefas Compatíveis*

Entrada: $\langle T, n, s, f \rangle$, onde $T = \{t_1, \ldots, t_n\}$ é um conjunto com n tarefas onde cada $t_i \in T$ tem tempo inicial s_i e tempo final f_i.
Saída: Maior subconjunto $S \subseteq T$ de tarefas mutuamente compatíveis.



Observe várias soluções viáveis

ótima : $S_1 = \{t_1, t_4, t_8, t_{10}\}$, $S_2 = \{t_1, t_4, t_9, t_{10}\}$

Algoritmos gulosos para escalonamento compatível

Enquanto puder escolher tarefas,
1) escolha a de menor duração
2) escolha a de menor s_i
3) escolha a de menor f_i

que seja compatível com as tarefas já escolhidos

Invariante: antes de qualquer iteração, o conj. já escolhido só tem tarefas mutuam. compatíveis

Note como todos têm intenção de alcançar o objetivo e são viáveis

Algoritmos gulosos para escalonamento compatível

1) escolha a de menor duração
2) escolha a de menor s_i
3) escolha a de menor f_i

→ Esses algoritmos são ótimos?

1) Não, por exemplo:

2) Não, por exemplo:

3) Parece que sim

Algoritmo guloso que parece bom

```
ESCALONACOMPATIVELREC(T)
1 se |T| == 1 então
2     devolve T
3 Seja t_x a tarefa que termina primeiro em T
4 T' = {t_z ∈ T : s_z ≥ f_x}
5 S' = ESCALONACOMPATIVELREC(T')
6 devolve S' ∪ {t_x}
```

1) Ele devolve uma solução viável?

Por indução em $n = |T|$, vamos provar que $P(n)$ = "ESCALONACOMPATREC(T), em que $n = |T|$, devolve uma solução viável" vale.

Se $n = 1$, o alg. devolve o próprio T, que só tem uma tarefa e portanto é viável. ∴ $P(1)$ vale.

Se $n > 1$, suponha que $P(k)$ vale para $1 \leq k < n$.

O algoritmo encontra t_x e calcula T', que só tem tarefas que começam depois que x termina. Então qualquer tarefa em T' é compatível com t_x.

Em seguida o algoritmo chama a recursão. Por HI, S' é solução viável para T' (mutuam. compatíveis). Mas então $S' \cup \{t_x\}$ é um conj. de tarefas mutuam. compat. ∴ é solução viável p/ T e $P(n)$ vale.

2) Quanto tempo ele leva?

Vamos supor que T é representada por dois vetores s e f, em que $s[i], f[i]$ são início/fim da tarefa t_i, se $t_i \in T$, e $s[i] = f[i] = -1$ se $t_i \notin T$. Vamos supor também que a solução S é um vetor em que $S[i] = 1$ sse i foi esc.

Vamos supor ainda que $f[1] \leq f[2] \leq \ldots \leq f[n]$.

Seja $X(n)$ o tempo do alg. sobre n tarefas:

$$X(1) = 1 \quad \text{e} \quad X(n) \leq X(n-1) + \Theta(n)$$
$$\Rightarrow O(n^2).$$

Linhas 3 e 4:

sejam s', f' dois vetores de tam. n $\quad s'[i] = f'[i] = -1$

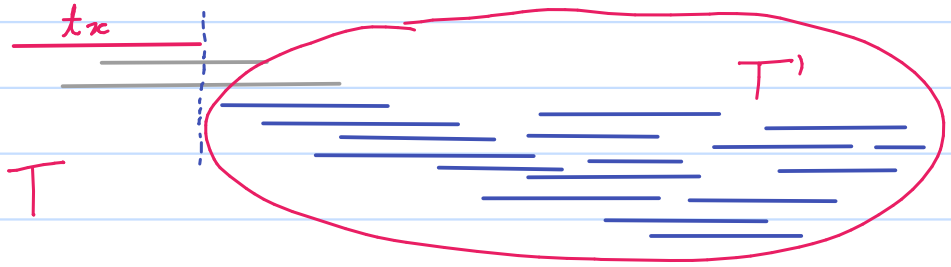
$x = 1$ enquanto $f[x] == -1$, $x{+}{+}$

para $z = x+1$ até n se $s[z] \geq f[x]$, $s'[z] = s[z]$, $f'[z] = f[z]$

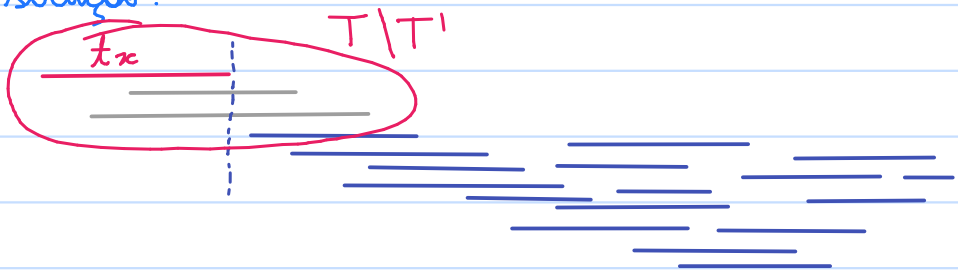
3) O algoritmo é ótimo?

Por indução em $n = |T|$, vamos provar que
$P(n)$ = "ESCALONACOMPATREC(T), com $n = |T|$, devolve solução ótima".

Se $n = 1$, note que o algoritmo simplesmente devolve T.
De fato, essa é a solução ótima nesse caso. ∴ $P(1)$ vale.
Seja então $n > 1$ e suponha que o algoritmo devolve uma solução ótima para qualquer conjunto que tenha menos do que n tarefas, isto é, $P(k)$ vale $\forall\ 1 \leq k < n$.
Com $n > 1$, o alg. encontra t_x, que termina primeiro em T, e monta T' com todas as tarefas compatíveis com t_x.



Como $|T'| < |T|$ pois ao menos $t_x \notin T'$, então por HI vale que S' é ótima para T'.
Agora suponha que $S = S' \cup \{t_x\}$ não é ótima para T.
Primeiro note que qualquer solução ótima para T deve conter alguma $t_y \in T \setminus T'$: se não, t_y poderia ser adicionada a tal solução.



Como $T \setminus T'$ contém todas as tarefas incompatíveis com t_x, concluímos que existe sol. ótima que contém t_x.
Denote uma tal sol. ótima por S^*.

Como S não é ótima, $|S^*| > |S|$.
Note que $R' = S^* \setminus \{t_x\}$ é tal que $R' \subseteq T'$ e, ademais, R' é ótima para T': se não fosse, haveria $R^* \subseteq T'$ ótima tal que $|R^*| > |R'|$ e, portanto, $R^* \cup \{t_x\}$ seria melhor para T.
Então $|R'| = |S^* \setminus \{t_x\}| = |S^*| - 1 > |S| - 1 = |S'|$, o que contradiz S' ser ótima para T'.
Então S é ótima para T. CQD

Versão iterativa

EscalonaCompativel(T, n)

```
1 Ordene as tarefas em ordem não-decrescente de tempo final
2 Renomeie-as para que f_1 ≤ f_2 ≤ ··· ≤ f_n
3 S = {t_1}
4 k = 1
5 para i = 2 até n, incrementando faça
6     se s_i ≥ f_k então
7         S = S ∪ {t_i}
8         k = i
9 devolve S
```

Invariante simples:
↳ antes de cada iteração, S contém um conjunto de tarefas mutuamente compatíveis.

→ A otimalidade pode ser observada da otimalidade de EscalonaCompativelRec, ou diretamente da seguinte invariante:

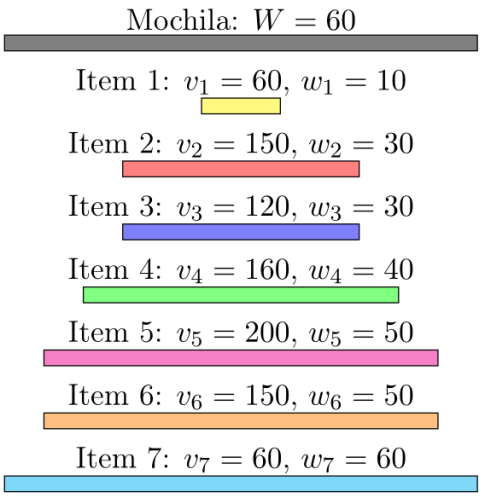
$P(x)$ = "antes da x-ésima iteração começar, Temos $i = x+1$ e
1) S é uma solução ótima para T_{i-1} que minimiza $f(S)$, e
2) $f_k \geq f_j$ para todo $t_j \in S$."

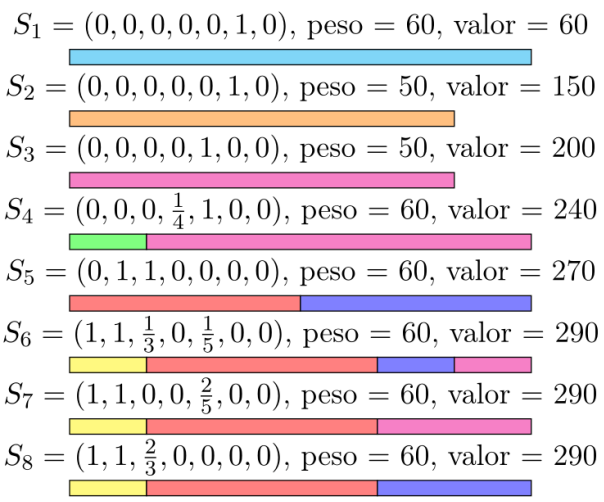
onde $T_a = \{t_1, t_2, \ldots, t_a\}$ para qualquer $a \in \{1, \ldots, n\}$ e
$f(X) = \max\{f_j : t_j \in X\}$ para qualquer "conj. compatível" $X \subseteq T$.

Problema *Mochila Fracionária*

Entrada: $\langle I, n, w, v, W\rangle$, onde $I = \{1, 2, \ldots, n\}$ é um conjunto de n itens sendo que cada $i \in I$ tem um peso w_i e um valor v_i associados, e W é a capacidade total da mochila.

Saída: Sequência $S = (f_1, \ldots, f_n)$ em que $f_i \in [0, 1]$, para todo $i \in I$, tal que $\sum_{i=1}^{n} f_i w_i \leq W$ e $\sum_{i=1}^{n} f_i v_i$ é máximo.





Algoritmos gulosos para Mochila Fracionária

Enquanto houver espaço na mochila:
Escolha a maior fração possível do item
① de maior valor
② de menor peso
③ de maior razão valor/peso

Note como os 3 algoritmos geram soluções viáveis e são gulosos, mas ① e ② não são ótimos:

① $W = 10$

$v_1 = 10000$ $w_1 = 10$

$v_2 = 10000$ $w_2 = 5$

$v_3 = 10000$ $w_3 = 4$
9999

② $W = 10$

$v_1 = 1$ $w_1 = 9$

$v_2 = 10000$ $w_2 = 10$

Algoritmo guloso que parece bom

```
MochilaFracionaria(I, n, W)
1  Ordene os itens pela razão valor/peso e os renomeie de forma que
   v_1/w_1 ≥ v_2/w_2 ≥ ··· ≥ v_n/w_n
2  capacidade = W
3  i = 1
4  enquanto i ≤ n e capacidade ≥ w_i faça
5      f_i = 1
6      capacidade = capacidade − w_i
7      i = i + 1
8  se i ≤ n então
9      f_i = capacidade/w_i
10 para j = i + 1 até n, incrementando faça
11     f_j = 0
12 devolve (f_1, ..., f_n)
```

① Solução viável?

Invariante: "antes da t-ésima iteração, $i = t$ e capacidade $= W - \sum_{j=1}^{t-1} f_j \cdot w_j$.

com capacidade $\geq$ ①"

② Tempo?

$\Theta(n \lg n)$, pois na linha 1 ordenamos os itens e os laços enquanto e para, juntos, percorrem todos os itens única vez, fazendo operações constantes. Como $\Theta(n \lg n) + \Theta(n) = \Theta(n \lg n)$, o resultado segue.

③ Ótimo?

Sim.

IDEIA DA PROVA:

	1	2	3 (i)	4	5	6	7	
Suponha que f =	1	1	1	½	⓪	⓪	⓪	é a sol. do alg.
Seja f^* =	1	1	½	¼	⅓	⓪	1/9	é uma sol. ótima $\neq f$
Vamos montar f' =	1	1	1	⓪	⅙	⅛	⓪	

e mostrar que f' é ótima → usando a escolha gulosa!

TEOREMA: Dado um conjunto $I = \{1, \ldots, n\}$ de n itens, em que cada $i \in I$ tem um peso w_i e um valor v_i associados, e dada uma mochila com capacidade de peso W, o algoritmo MochilaFracionaria(I, n, W) devolve uma solução ótima para o problema mochila fracionária.

PROVA: Seja $f = (f_1, \ldots, f_n)$ a sol. devolv. por MochilaFracionaria(I, n, W).
Seja $f^* = (f_1^*, \ldots, f_n^*)$ uma sol. ótima da mesma instância.
Se $f = f^*$, não há o que provar.
Então f difere de f^* em alguns valores e seja i o menor tal índice.
Como $f_j = f_j^*$ para todo $j < i$ (pela escolha de i), então só pode ser o caso que $f_i > f_i^*$, por causa da escolha do algoritmo.

Note que $\sum_{j=1}^{n} f_j w_j = W = \sum_{j=1}^{n} f_j^* w_j$.

Então, particularmente, $\sum_{j=i}^{n} f_j w_j = \sum_{j=i}^{n} f_j^* w_j$.

Vamos criar $f' = (f_1', \ldots, f_n')$ a partir de f^*.
Começamos fazendo $f_j' = f_j^* \ (= f_j)$ para $1 \leq j < i$.
Nesse momento, $\sum_{j=1}^{i-1} f_j' w_j = \sum_{j=1}^{i-1} f_j^* w_j$.

Fazemos então $f_i' = f_i$.

Como $f_i' = f_i > f_i^*$, não podemos apenas copiar o restante de f^*.
Precisamos, no entanto, garantir que $\sum_{j=i}^{n} f_j' w_j = \sum_{j=i}^{n} f_j^* w_j$.
Preenchemos então $f_{i+1}', \ldots, f_n'$ com valores que satisfaçam isso, o que é possível, já que existe ao menos uma solução que já satisfaz isso (a própria f).

Reescrevendo e isolando referente a i, temos que vale

$$f'_i w_i + \sum_{j=i+1}^{n} f'_j w_j = f^*_i w_i + \sum_{j=i+1}^{n} f^*_j w_j$$

$$w_i (f'_i - f^*_i) = \sum_{j=i+1}^{n} w_j (f^*_j - f'_j) \qquad (\star)$$

Então $\sum_{j=1}^{m} f'_j v_j = \sum_{j=1}^{i-1} f^*_j v_j + f'_i v_i + \sum_{j=i+1}^{m} f'_j v_j$

$$= \sum_{j=1}^{m} f^*_j v_j - f^*_i v_i - \sum_{j=i+1}^{m} f^*_j v_j + f'_i v_i + \sum_{j=i+1}^{m} f'_j v_j$$

$$= \sum_{j=1}^{m} f^*_j v_j + v_i (f'_i - f^*_i) - \sum_{j=i+1}^{m} v_j (f^*_j - f'_j)$$

$$= \sum_{j=1}^{m} f^*_j v_j + v_i (f'_i - f^*_i) \frac{w_i}{w_i} - \sum_{j=i+1}^{m} v_j (f^*_j - f'_j) \frac{w_j}{w_j}$$

$$= \sum_{j=1}^{m} f^*_j v_j + \frac{v_i}{w_i} (f'_i - f^*_i) w_i - \sum_{j=i+1}^{m} \frac{v_j}{w_j} (f^*_j - f'_j) w_j$$

$$\geqslant \sum_{j=1}^{m} f^*_j v_j + \frac{v_i}{w_i} (f'_i - f^*_i) w_i - \sum_{j=i+1}^{m} \frac{v_i}{w_i} (f^*_j - f'_j) w_j$$

$$= \sum_{j=1}^{m} f^*_j v_j + \frac{v_i}{w_i} \left((f'_i - f^*_i) w_i - \sum_{j=i+1}^{m} (f^*_j - f'_j) w_j \right)$$

$$= \sum_{j=1}^{m} f^*_j v_j$$

Concluímos então que $\sum_{j=1}^{m} f'_j v_j \geqslant \sum_{j=1}^{m} f^*_j v_j$, de forma que f' deve ser ótima também.

Fazendo essa transformação repetidamente, chegaremos a f.

Portanto, f também deve ser ótima.

CQD