Tratamento de problemas $\mathcal{NP}$-difíceis: *backtracking*.

- ▷ Casos de aplicação:
  - Problemas cujas soluções podem ser representadas por tuplas (vetores) de tamanho fixo ou variável da forma $(x_1, \ldots, x_n)$.
  - Solucionar o problema equivale a encontrar uma tupla que otimiza uma *função critério* $P(x_1, \ldots, x_n)$ **ou** encontrar todas as tuplas que satisfaçam $P(x_1, \ldots, x_n)$.
- ▷ Restrições:
  - *Explícitas*: especificam os domínios (finitos) das variáveis na tupla.
  - *Implícitas*: relações entre as variáveis da tupla que especificam quais delas respondem ao problema.

Backtracking: conceitos básicos

- **Espaço de soluções**: conjunto de todas as tuplas satisfazendo as restrições *explícitas*.
- **Espaço de estados**: conjunto de todas as subseqüências das tuplas do *espaço de soluções*.
- **Algoritmo Força Bruta**: enumera todas tuplas do espaço de soluções e verifica quais delas satisfazem às restrições *implícitas*.
- **Algoritmo Backtracking**:
  - busca sistemática no espaço de estados do problema que é organizado segundo uma *estrutura de árvore*, denominada **árvore de espaço de estados**.
  - uso de *funções limitantes* para restringir a busca na árvore.

Backtracking: conceitos básicos (cont.)

- Métodos de exploração do espaço de estados (EE):
  - *nós ativos*: aqueles que ainda têm filhos a serem gerados.
  - *nós amadurecidos*: aqueles em que todos os filhos já foram gerados ou não devam ser mais expandidos de acordo com a *função limitante*.
  - *nó corrente*: aquele que está sendo explorado.
- *Backtracking*: busca no EE é feita em *profundidade*.
- *Branch-and-Bound*: durante a busca no EE a geração de todos os filhos do nó corrente assim como o cálculo da função limitante em cada um deles é feita de uma vez só (e.g., busca em *largura*).

Backtracking: o algoritmo (recursivo)

BACK(k);
    (* Entrada: $x_1, x_2, \ldots, x_{k-1}$ (já escolhidos). *)
    (* Saída: todas as soluções serão impressas. *)

    (* Obtém o domínio de x_k e o seu tamanho. *)
    $(T, \ell) \leftarrow$ Domínio$(x_1, x_2, \ldots, x_{k-1})$;
    Para $i = 1$ **até** ℓ **faça** (* testa todos valores do domínio *)
        $x_k \leftarrow T[i]$;
        Se $(x_1, \ldots, x_k)$ satisfaz às restrições implícitas **então**
            IMPRIMA$(x_1, \ldots, x_k)$; (* solução encontrada *)
        (* Verifica se busca deve prosseguir nesta subárvore *)
        Se $B_k(x_1, \ldots, x_k)$ é verdadeiro **então** BACK$(k+1)$;
    fim-para.
fim.

Backtracking: problema da soma de subconjuntos (SOS)

- SOS: dado um conjunto $S = \{w_1, \ldots, w_n\}$ de n valores inteiros positivos e um valor inteiro positivo W, existe $U \subseteq \{1, \ldots, n\}$ tal que $\sum_{i \in U} w_i = W$?
- Exemplo: Se $n = 4$, $S = \{11, 13, 24, 7\}$ e $W = 31$ tem-se $U = \{1, 2, 4\}$ e $U = \{3, 4\}$.
- SOS é $\mathcal{NP}$-completo. (PAR $\propto_{\text{poli}}$ SOS)
- Representação das soluções:
  1. tupla de tamanho variável: como no exemplo acima.
  2. tupla de tamanho fixo n: $x_i = 1$ se $i \in U$ e $x_i = 0$ caso contrário.

Backtracking: problema SOS (cont.)

- *Funções Limitantes*:
  1. $B_k(x_1, \ldots, x_k) = \texttt{true} \Leftrightarrow \sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq W$.
  2. Suponha que $0 < w_1 \leq w_2 \leq \ldots \leq w_n$. Então $(x_1, \ldots, x_k)$ não pode levar a uma nova solução se $\sum_{i=1}^{k} w_i x_i + w_{k+1} > W$. Logo, uma outra função limitante seria:

$$B'_k(x_1, \ldots, x_k) = \texttt{true} \quad \Leftrightarrow \quad B_k(x_1, \ldots, x_k) = \texttt{true} \textbf{ e } \sum_{i=1}^{k} w_i x_i + w_{k+1} \leq W.$$

- Componentes do algoritmo:
  - Tuplas de tamanho n (fixo) onde todo x_i está em $\{0, 1\}$.
  - Hipóteses: $0 < w_1 \leq \ldots \leq w_n < W$ e $\sum_{i=1}^{n} w_i \geq W$.
  - Os parâmetros $s = \sum_{i=1}^{k-1} w_i x_i$ e $r = \sum_{i=k}^{n} w_i$ tais que $s + r \geq W$ são passados para a k-ésima chamada recursiva.

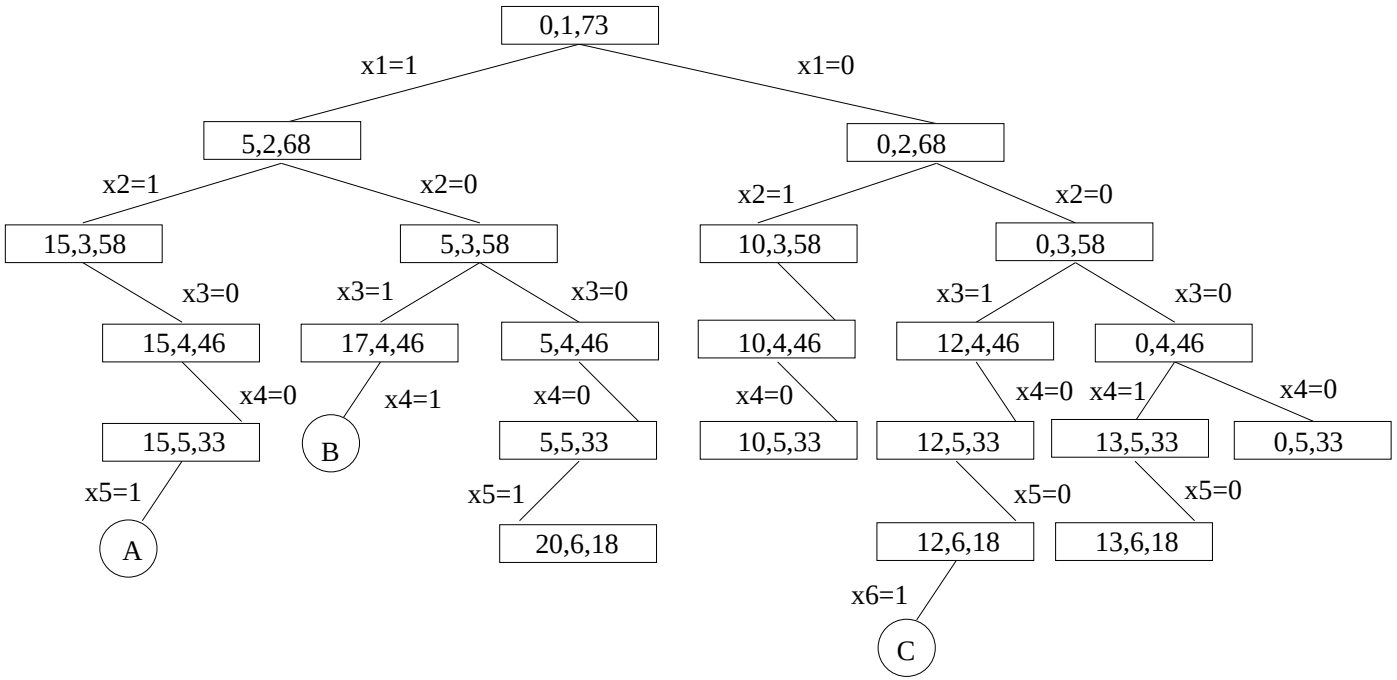
Backtracking: problema SOS (cont.)

```
SOS(s, k, r);   (* Domínio de x_k será sempre {0, 1}. *)
    x_k ← 1;   (* Caso x_k = 1. *)
    Se (s + w_k = W) então
        IMPRIMA (x_1, ..., x_k, 0, ..., 0) (* não precisa de recursão *)
    se não
        Se (s + w_k + w_{k+1} ≤ W) então SOS(s + w_k, k + 1, r − w_k);
    fim-se
    x_k ← 0;   (* Caso x_k = 0. *)
    Se (s + r − w_k ≥ W) e (s + w_{k+1} ≤ W) então
        SOS(s, k + 1, r − w_k);
    fim-se
fim.
```

Backtracking: problema SOS (cont.)

- Exemplo: $n = 6$, $W = 30$, $S = \{5, 10, 12, 13, 15, 18\}$.
- Espaço de estados completo: $2^{6+1} - 1 = 127$.
- Parte da árvore de espaço de estados gerada por `SOS(0,1,73)` (próxima transparência ...)
- Legenda:
  - triplas (s, k, r);
  - $\bigcirc$ são os nós-resposta.

Backtracking: problema SOS (cont.)



$$n = 6, \quad W = 30, \quad S = \{5, 10, 12, 13, 15, 18\}$$

$$s = \sum_{i=1}^{k-1} w_i x_i \text{ e } r = \sum_{i=k}^{n} w_i$$

Backtracking: p-coloração de grafos (COR)

- COR: dado um grafo não-orientado G com n vértices e representado por sua matriz de adjacências A, encontrar todas as colorações de G com p cores ou menos.
- Representação das soluções: tuplas de tamanho n (fixo) onde $x_i \in \{0, 1, \ldots, p\}$ representa a cor do vértice i.
- A cor *zero* significa que o vértice ainda não está colorido.
- O algoritmo inicializará a tupla com *zeros*.
- *Função Limitante*: recursão só é interrompida se não for possível alocar uma cor para o k-ésimo vértice.

Backtracking: p-coloração de grafos (COR)

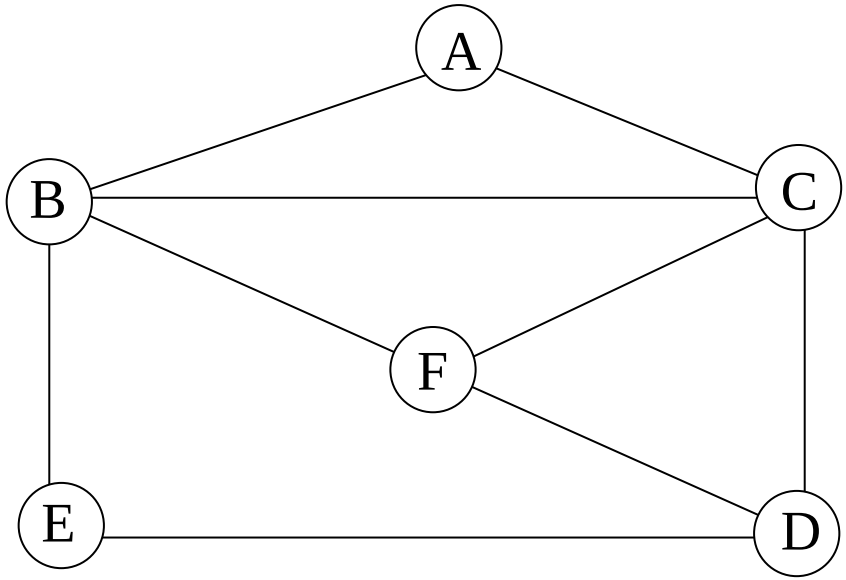
Domínio$(x_1, \ldots, x_{k-1})$;
    Para $i = 1$ **até** p **faça** pode$[i] = 1$;
    Para $j = 1$ **até** $k - 1$ **faça**
        Se $(A[k, j] = 1$ e $x_j \neq 0)$ **então**
            pode$[x_j] \leftarrow 0$; (* não pode ter cor igual a um vizinho *)
    fim-para;
    $\ell \leftarrow 0$; (* Constrói o domínio *)
    Para $i = 1$ **até** p **faça**
        Se (pode$[i] = 1$) **então**
            $\ell \leftarrow \ell + 1$; $T[\ell] \leftarrow i$;
        fim-se;
    fim-para;
    Retornar (T, ℓ).
fim.

Backtracking: p-coloração de grafos (COR)

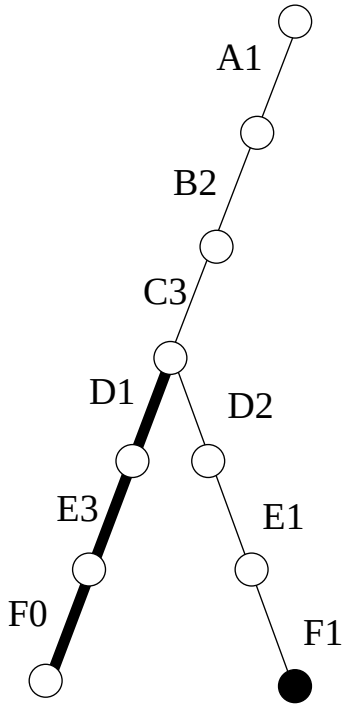
```
COR(k);
    (T, ℓ) ← Domínio(x_1, x_2, ..., x_{k-1});
    Para i = 1 até ℓ faça
        x_k ← T[i];
        Se k = n então (* todos vértices estão coloridos *)
            IMPRIMA(x_1, ..., x_k)
        se não COR(k + 1);
    fim-para.
```

Backtracking: p-coloração de grafos (COR)

- Exemplo: $p = 3$.



Backtracking: *p*-coloração de grafos (COR)



Exemplo de *backtracking*: árvore de espaço de estados