int subsetsum(int v[], int n)

Alg n det.

2 fases

[ORACULO] produz alguma coisa

o que produz? Há que definir

- array com alguns elementos de V[]
- { p elementos de v[]

Parte

Determ.:

```
int teste(int aux[], int p)
{ int sum = 0;
  for(int i=0; i<p; i++)
      sum += aux[i];
  return (sum == 0);
}
```

devia-se verificar que aux ⊆ v[]

Garantir que teste $\Theta(N)$ → como $P < N$, e teste é $\Theta(P)$, tá bem e oraculo tb

//

Outro só Determinístico

```
int subsetsum
{ return (f(v, n, 0)); }
```

f: a soma de um subconj de v é 0 (ou outro x)

```
int f( int v[], int n, int s){
{   if(v[0]==s) return 1;
    if (n==1) return (v[0]==s);
    else return ( f(v+1, n-1, s-v[0])
                  || f(v+1, n-1, s));
}
```

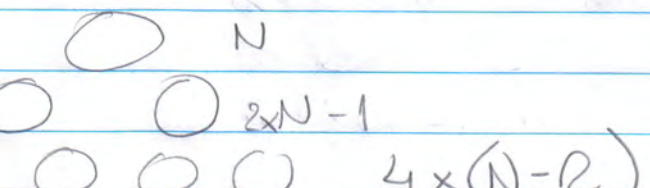
// incluindo o 1º elem na soma com os restantes somar s-v[0]

v[0] = 1
s = 10
incluindo s = 9

pior caso

$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ 2 \times Tf(N-1) & N > 1 \end{cases}$$

$$\sum_{i=0}^{} \begin{matrix} \Theta(1) \\ 2 \times \Theta(1) \\ 4 \times \Theta(1) \end{matrix}$$



$$\sum_{i=0}^{N} 2^i \Theta(1) = 2^N \times \Theta(1)$$

$$T(N) = \Theta(1) \cdot (2^{n+1} - 1)$$

$$= \Theta(2^n)$$

//

λ = nº chaves armazenadas / dimensão do array

em chaining, $\lambda \in [0, +\infty[$

λ, o que mede? o comprimento médio de chaves por listas, o tempo a obter um elemento

em open-addressing

$$0 \leq \lambda \leq 1$$

$$\lambda = P(\text{colisão})$$

Exame 07/08

Parte 2

② b)

clique : grafo com todos os vértices orientados

a)

```
int eClique(Grafo g, int v[], int n)
{
    int r=1;
    int i,j;
    for(i=0; i<n-1; i++)
        for(j=i+1; j<n; j++)
            if(!aresta(g,v[i],v[j])) return 0;
    return 1;
}
```

$O(N^2)$

(xV se tiver em lista de adjacências)

clique Maximal

optimização

Prob de decisão complexidade (boolP)

Dado G (grafo) e v (vertice)

saber se existem em G um
clique com K vertices

é NP-completo equivalente a SAT

Solução: gerar todos os que
incluem (v) e ver quais
são cliques, mais ou menos

10/11 Exame

(5)

```
typedef struct edge{
    int cost;
    int dest;
    struct edge *next;
} Edge;

typedef Edge * Graph [MAXV]

int capacidades [MAXV]

void calcula-capacidades (Graph g)
{
    for(int i=0; i<MAXV; i++)
        capacidades [i] = 0;
}
```

$\Theta(V)$

```
{
    for(i=0; i<MAXV; i++)
    {
        for(Edge e = g[i]; e, e=e->next)
        {
            capacidades [i] -= e->cost;
            [e->dest] += e->cost;
        }
    }
}
```

$\Theta(V+E)$

```
f = 1
while( x>0 ){
    f = f * x;
    x = x - 1;
}
```

// $x = x \geq 0 \wedge f = 1$

Pré-condição: $\{ x = x_0 \in \mathbb{Z} \wedge x_0 \geq 0 \}$

Pós- " : $\{ fact(x_0) = f \}$

Se ~~se puser na~~ while fosse $(x \geq 0)$, não precisava da segunda cond. na pré-cond.

Invariante:

$$\frac{P \Rightarrow I \qquad \{I \wedge x>0\}\ f = f * x;\ x = x-1\ \{I\} \qquad I \wedge x < 0 \Rightarrow Q}{\{P\}\ \text{while}(x>0)\ \{f = f * x;\ x = x-1\}\ \{Q\}}$$

$x_0 = 5$

x	f
5	1
4	5
3	20
2	60
1	120
0	120

$I = (fact(x) * f = fact(x_0))$

mas é útil?

para 3º passo

$I = \{ fact(x_0) = f * fact(x) \wedge x \geq 0 \}$

1º $P \Rightarrow I = \dots$ | $P \Rightarrow fact(x_0) = fact(x_0)$ ✓

3º $I \wedge x \leq 0 \Rightarrow Q$ |

2º

$I \wedge e \Rightarrow I[x-1/x][f*x/f]$

$\{ fact(x_0) = f * fact(x) \wedge x \geq 0 \wedge x > 0 \}$ $\{ f = f * x; \; x = x - 1; \}$ $\{ \dots \}$

$I \wedge e$ I

$I[x-1/x] = fact(x_0) = f * fact(x-1) \wedge x - 1 \geq 0$

$I[x-1/x][f*x/f] =$
$fact(x_0) = f * x * fact(x-1) \wedge x - 1 \geq 0$

$fact(x_0) = f * fact(x) \wedge x > 0$ ($fact(x)$)
$\Rightarrow fact(x_0) = f * x * fact(x-1)$
$\wedge \; x - 1 \geq 0$

09/10 II (1)

BalOK:

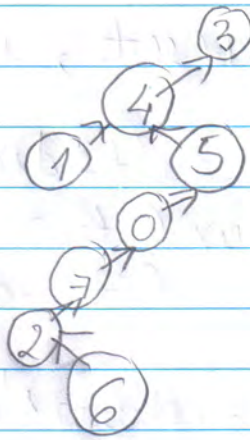
```
int BalOK (AVLTree t){
    int aux;
    return BalOKAux(t, &aux);
}

int BalOKAux(AVLTree t, int * alt)
{
    int alt_esq, alt_din, OK_esq, OK_din;
    OK_esq = BalOKAux(t->esq, &alt_esq);
    OK_din =      "       (t->din ; &alt_din);

    if(!t){ *alt=0, return 1};

    if (alt_esq > alt_din)
        *alt = alt_esq+1;
    else
        *alt = alt_din+1;
    return( OK_esq && OK_din &&
            t->bfactor == alt_esq - alt_din);
}
```

07/08 Teste

Última da parte II

0	1	2	3	4	5	6	7
5	4	7	-1	3	4	2	0	...

raiz ; esq ; dir



tem as setas ao contrário das definições tradicionais de árvores

1º

$\Theta(V)$ inserções à cabeça { O melhor é converter primeiro numa árvore normal (uma lista de adjacências)

raiz=3 (identificada a meio, quando encontrar o -1)

0 → 7
1 →
2 → 6
3 → 4
4 → 5, 1
5 →

2º Fazer pre-order, $\Theta(N)$