Partição de Conjuntos (PARTITION)

Marcos Straub
marcos.straub@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul
Junho de 2012

O problema

- O problema consiste em decidir se, dado um conjunto de números inteiros S, é possível dividir S em dois subconjuntos s1 e s2 de forma que a soma dos elementos em s1 seja igual a soma dos elementos de s2.

S := {2,3,5,7,13}
s1 := {3,5,7}
s2 := {2,13}
soma(s1) = soma(s2) = 15

PARTITION pertence à NP

- Para a prova, precisamos de um algorimo de verificação em tempo polinomial que verifique:
  - Se a união de s1 e s2 resulta em S.
  - Se o somatório de s1 é igual ao de s2.

PARTITION pertence à NP

- Algoritmo de verificação em tempo polinomial

```
verificapartition(S,s1,s2){
// Verifica se s1Us2 = S
if( uniao(s1,s2) != S)
     false

//Verifica se somatorio(s1) == somatorio(s2)
foreach i1 in s1:
     somas1= somas1 +i1;
foreach i2 in s2:
     somas2= somas2 +i2;

if(somas1 == somas2)
     return true;
else
     return false;
```

PARTITION pertence à NP

- Análise de complexidade

```
verificapartition(S,s1,s2){
// Verifica se s1Us2 = S
if( uniao(s1,s2) != S)
     false

//Verifica se somatorio(s1) == somatorio(s2)
foreach i1 in s1:
     somas1= somas1 +i1;
foreach i2 in s2:
     somas2= somas2 +i2;

if(somas1 == somas2)
     return true;
else
     return false;
```

O(1)

Atualização de descritores para listas encadeadas

PARTITION pertence à NP

- Análise de complexidade

```
verificapartition(S,s1,s2){
// Verifica se s1Us2 = S
if( uniao(s1,s2) != S)
     false

//Verifica se somatorio(s1) == somatorio(s2)
foreach i1 in s1:
     somas1= somas1 +i1;
foreach i2 in s2:
     somas2= somas2 +i2;

if(somas1 == somas2)
     return true;
else
     return false;
```

O(N)

Percorrer vetor união e marcando elementos encontrados de S

PARTITION pertence à NP

- Análise de complexidade

```
verificapartition(S,s1,s2){
// Verifica se s1Us2 = S
if( uniao(s1,s2) != S)
     false

//Verifica se somatorio(s1) == somatorio(s2)
foreach i1 in s1:
     somas1= somas1 +i1;
foreach i2 in s2:
     somas2= somas2 +i2;

if(somas1 == somas2)
     return true;
else
     return false;
```

O(N)

Percorre s1 e s2

PARTITION pertence à NP

- Análise de complexidade

```
verificapartition(S,s1,s2){
// Verifica se s1Us2 = S
if( uniao(s1,s2) != S)
     false

//Verifica se somatorio(s1) == somatorio(s2)
foreach i1 in s1:
     somas1= somas1 +i1;
foreach i2 in s2:
     somas2= somas2 +i2;

if(somas1 == somas2)
     return true;
else
     return false;
```

O(1)

PARTITION é NP-Completo

- Redução em tempo polinomial de um problema NP-Completo conhecido: soma de subconjuntos (SUBSET-SUM)
  - Instâncias de SUBSET-SUM (I) deverão ser transformadas em instâncias de PARTITION (I’)

O Problema SUBSET-SUM

- Dado uma conjunto de inteiros S e um número t.
- Existe um subconjunto S' $\subseteq$ S que a soma dos seus elementos é igual a t?
- Ex: S := {**3**,**5**,11} e t := 8
  S' := {3,5}

Algoritmo de Redução - Teoria

- Sendo I=(S,t) uma instância de SUBSET-SUM, onde:
- S = {v1,v2,v3,...,vn}
- t = somatorio(s' <= S)

- Iremos mapear para instâncias de PARTITON I'= (S'), onde
- S'={v1,v2,v3,...,vn,x}
- x=v1 + v2 + … + vn − 2t

Algoritmo de Redução - Teoria

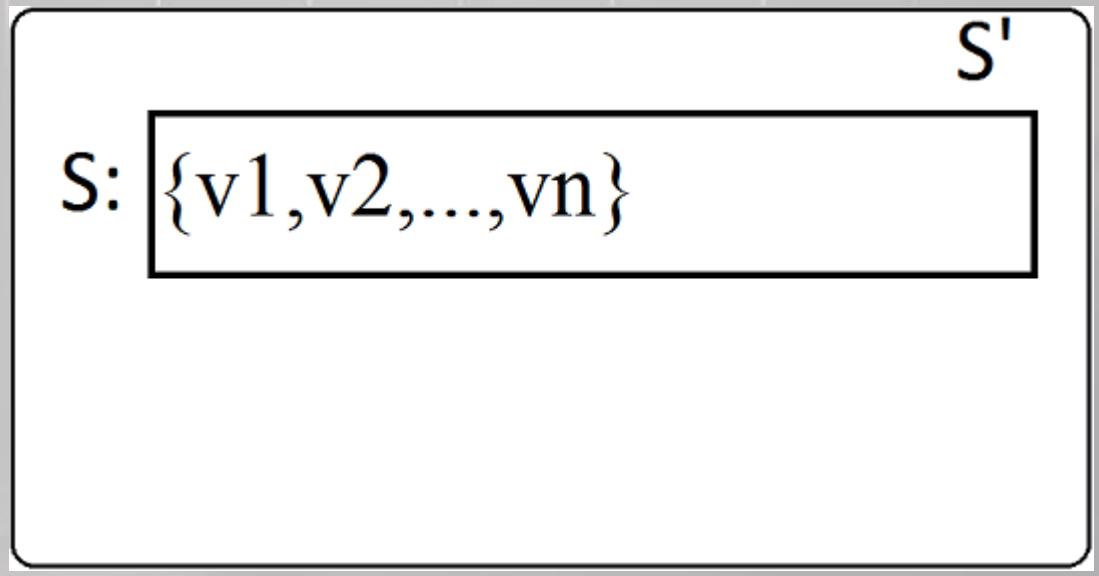
- Nesse caso, o somatário de S' totalizará:
  - (v1 + v2 +…+ vn + x) ou
  - (2v1 + 2v2 +…+ 2vn - 2t)
- E cada um dos subconjuntos de S' (s1,s2) deverão ter como somatório a metade de S'
  - (2v1 + 2v2 +…+ 2vn - 2t)/2 ou
  - (v1 + v2 +...+ vn - t)

Algoritmo de Redução - Teoria

- Considerando que um dos conjutos (s1) deverá conter o elemento x:
  - s1 = {x, alguma_coisa)
  - somatorio(s1) = (v1 + v2 +...+ vn - t)
- Sendo que:
  - somatorio(alguma_coisa) =
  - somatorio(s1) – x =
  - (v1 + v2 +...+ vn - t) – x =
  - (v1 + v2 +...+ vn - t) - (v1 + v2 + … + vn − 2t) = t
- Portanto, quando a partição conseguir ser resolvida, conseguiremos um subconjunto de S cujo somatório seja t.

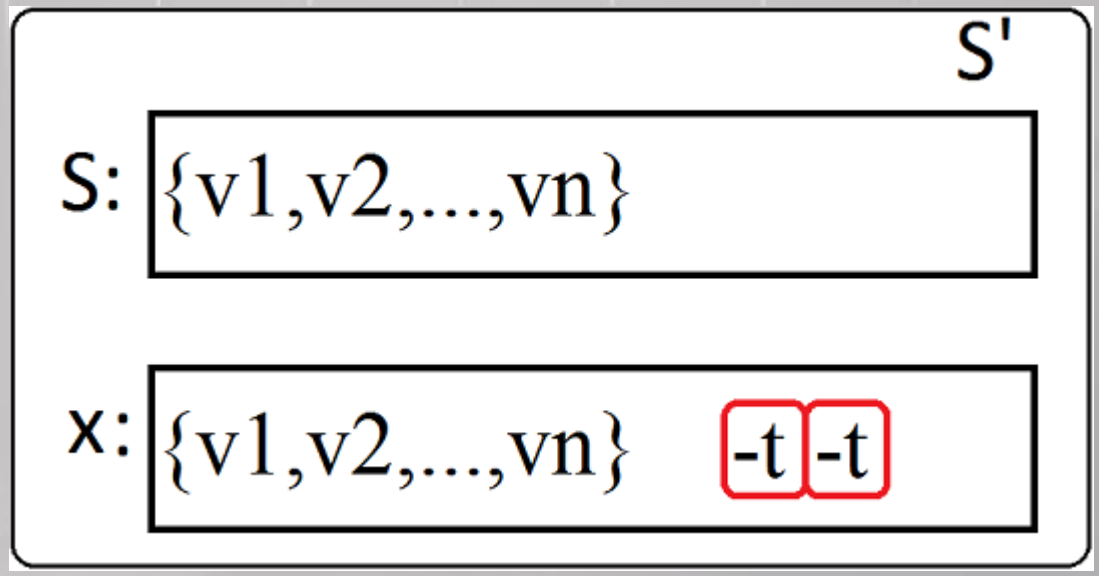
Algoritmo de Redução - Teoria

- Conjunto S inicial e S’ sendo contruído



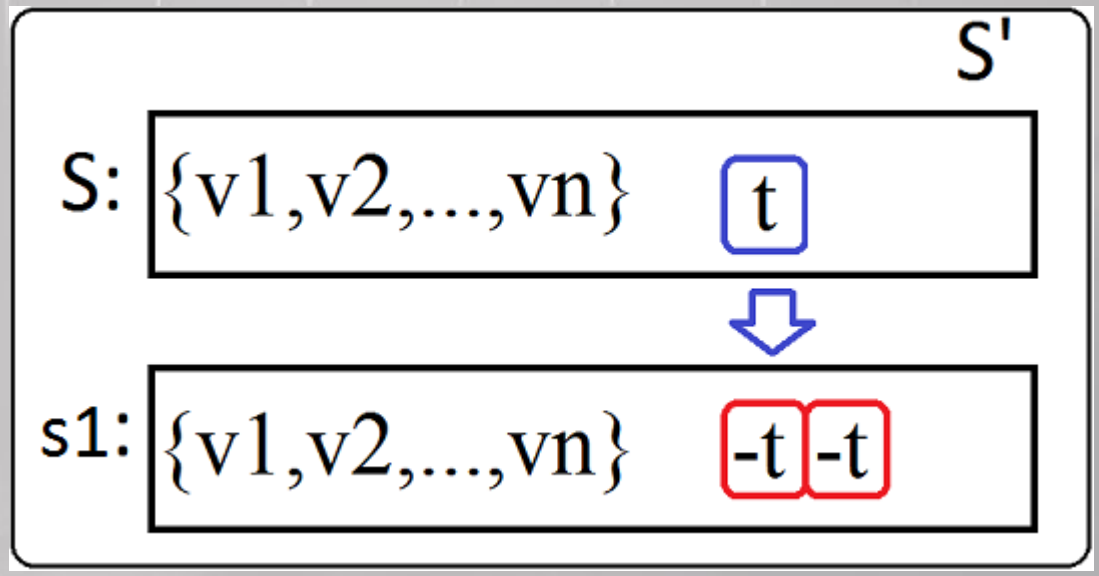
Algoritmo de Redução - Teoria

- X é adicionado faltando t para se igualar a metade



Algoritmo de Redução - Teoria

- s1 é formado com {x,t}
- s2 = S



Algoritmo de Redução - Algoritmo

```
subsetsum2partition(S,t){
    foreach i in S:
        somatorio = somatorio + i;

    x = somatorio - 2*t;
    S' = append(S,x);
    return S';
}
```

Algoritmo de Redução - Algoritmo

```
subsetsum2partition(S,t){
    foreach i in S:
        somatorio = somatorio + i;

    x = somatorio - 2*t;
    S' = append(S,x);
    return S';
}
```

O(N)

Percorre S

Algoritmo de Redução - Algoritmo

```
subsetsum2partition(S,t){
    foreach i in S:
        somatorio = somatorio + i;

    x = somatorio - 2*t;
    S' = append(S,x);
    return S';
}
```

O(1)

Algoritmo de Redução - Algoritmo

```
subsetsum2partition(S,t){
     foreach i in S:
          somatorio = somatorio + i;

     x = somatorio - 2*t;
     S' = append(S,x);
     return S';
}
```

O(1)

Atualização de descritores para final de vetor

Obrigado, Perguntas?

Artigo: http://inf.ufrgs.br/~msnascimento/partition_artigo.pdf
Apresentação: http://inf.ufrgs.br/~msnascimento/partition_apres.pdf