Enumeração de subconjuntos

Marcela Ribeiro de Oliveira
GRR20157372
Ciência da Computação
Universidade Federal do Paraná
Email: mro15@inf.ufpr.br

I. Compilação e Execução

Para compilar basta executar o comando *make*.

Para executar o programa que gera todos os subconjuntos:

`./subconjuntos <n> <k>`

Para executar o programa que gera o próximo subconjunto:

`./prox-subconj <n> <k> <subconjunto>`

Os elementos do subconjunto devem estar separados por espaços. Por exemplo: 1 2 3

II. Descrição do Problema

Dados um **n** e um **k**, sendo k≤n, o problema consiste em enumerar todos os subconjuntos de tamanho **k** do conjunto $\{1, \dots n\}$.

Desta forma, deve-se enumerar $\binom{n}{k}$ subconjuntos.

III. Algoritmos Implementados

Dois diferentes algoritmos foram implementados, o primeiro gera todos os subconjuntos dado um **n** e um **k**, e o segundo que gera o próximo subconjunto dado um **n**, um **k** e um subconjunto de tamanho **k**.

A ordem dos subconjuntos devolvidos nos dois programas é diferente, pois 2 estratégias diferentes foram utilizadas para enumerá-los. Estas estratégias estão detalhadas a seguir.

A. Subconjuntos

Este algoritmo devolve na saída padrão todos os subconjuntos de tamanho k de um dado n.

O algoritmo funciona da seguinte maneira:

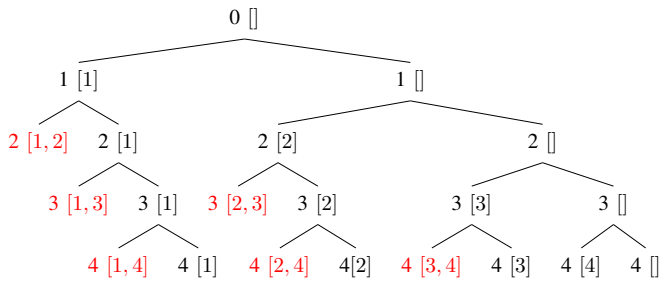
Na primeira chamada passa-se n, k, o conjunto (set$\{1, \dots n\}$), o offset começando em 0 e um conjunto (subsets) que inicia vazio.

A cada chamada da recursão são feitas 2 verificações, a primeira verifica se o tamanho do subconjunto subsets é igual a k, se for, então o subconjunto é impresso, e retorna-se para a função chamadora. A segunda verifica se o offset (deslocamento) é igual a n, se isso for verdade então todos os elementos de set foram percorridos, então também retorna-se para a função chamadora.

Se nenhuma dessas verificações for verdadeira, então prossegue-se a execução do algoritmo em 2 chamadas recursivas. Antes da primeira chamada é adicionado no fim do subconjunto subsets o elemento de set na posição offset, e então chama-se recursivamente a função enumerate com o offset incrementado de 1. Antes da segunda chamada é removido do subconjunto subsets o elemento correspondente a set na posição offset, e então chama-se recursivamente a função enumerate com o offset incrementado de 1.

Assim sendo, a cada nova chamada serão criadas duas novas chamadas, onde na primeira o elemento do conjunto está contido no subconjunto e na segunda o mesmo elemento do conjunto não está contido no subconjunto. Dessa forma é possível enumerar todos os subconjuntos sem que haja repetição.

Abaixo encontra-se um exemplo da árvore de execução para n=4 e k=2. Onde os nodos em vermelho representam as soluções.



O pseudocódigo do algoritmo implementado pode ser visto abaixo.

Algoritmo 1: enumerate

Entrada: $int\ n, int\ k, conjunto\ set, int\ offset = 0, conjunto\ subsets$
Saída: Todos os subconjuntos de tamanho k do conjunto $\{1, \dots n\}$

```
1  início
2      se subsets.size = k então
3          Imprime subconjunto retorna 0
4      fim
5      se offset = n então
6          retorna 0
7      fim
8      subsets := subsets ∪ {set[offset]}
9      enumerate(set, n, k, offset+1, subsets)
10     subsets := subsets − {set[offset]}
11     enumerate(set, n, k, offset+1, subsets)
12     retorna 0
13 fim
```

A recorrência deste algoritmo mantém uma propriedade importante, a toda chamada recursiva, se não cair na base, duas novas chamadas serão criadas. Na primeira delas (a esquerda) o elemento do conjunto (set) na posição offset estará contido no subconjunto de solução. Na segunda (a direita) o elemento do conjunto (set) na posição offset não estará contido no conjunto solução. Como a cada duas novas chamadas o offset é incrementado de 1, garante-se que todos os elementos do conjunto set serão percorridos e adicionados aos subconjuntos de solução, quando a recursão desce a esquerda.

O algoritmo para quando caí em uma de duas condições. A primeira delas é quando o tamanho do subconjunto é igual a k, ou seja, um subconjunto foi encontrado. A segunda é quando offset é igual a n, que indica que todos os elementos conjunto $\{1, \dots n\}$ foram percorridos e compõem-se parte da solução.

Assim pode-se concluir que o algoritmo para e devolve a solução para o problema.

B. Próximo Subconjunto

Este algoritmo devolve na saída padrão o próximo subconjunto de tamanho k de um dado n e um subconjunto de tamanho k.

O algoritmo funciona da seguinte maneira:

São gerados 2 vetores de tamanho n, o primeiro (set) representa o conjunto $\{1, \dots n\}$ e o segundo é um vetor de bits (bit_set) que será utilizado para gerar o próximo subconjunto, este vetor começa inicialmente com zeros.

Então percorre-se o subconjunto (subsets) lido da entrada e, nas posições de bit_set que representam os valores de subsets, é atribuido 1. Ou seja, o vetor bit_set tem 1 nas posições dos elementos do subconjunto de entrada. Por exemplo se n=5, k=3 e o subconjunto que deseja-se encontar o próximo é $1, 2, 3$, o vetor bit_set é formado pelos valores $0, 0, 1, 1, 1$.

A partir destas informações então inicia-se um laço, em que a cada iteração é somado +1 ao vetor de bits (bit_set). Esse laço termina quando a quantidade de bits 1 em bit_set é igual a k, que é quando o próximo subconjunto foi encontrado.

Então o subconjunto encontrado é mostrado na saída padrão e o programa encerra a sua execução.

Este algoritmo apesar de simples não é tão eficiente, pois gera conjuntos de tamanho diferente de k, que são subconjuntos do conjunto $\{1, \dots n\}$ e que como não são resposta para o problema são apenas descartados. Mas, este algoritmo resolve o problema e gera todos os subconjuntos também, basta executá-lo $\binom{n}{k}$ vezes passando na entrada o subconjunto anterior.

O pseudocódigo do algoritmo implementado pode ser visto a seguir.

Algoritmo 2: prox_subconj

Entrada: $int\ n, int\ k, conjunto\ subsets$
Saída: O próximo subconjunto de tamanho k do subconjunto de entrada

```
1  início
2      para i = n ate 1 faça
3          set := set ∪ {i}
4          bit_set := bit_set ∪ {0}
5      fim
6      para i = subsets.begin ate subsets.end faça
7          bit_set[subsets[i]] := 1
8      fim
9      found := 0
10     enquanto found == 0 faça
11         some 1 em bit_set
12         se quantidade de 1's em bit_set = k então
13             found := 1
14         fim
15     fim
16     para i = 1 ate n faça
17         se bit_set[i] == 1 então
18             imprime set[i]
19         fim
20     fim
21 fim
```

O laço deste algoritmo é executado até que um novo subconjunto de tamanho k seja encontrado. Como cada iteração soma-se 1 ao vetor de bits, em um dado momento este vetor terá novamente k bits 1 no vetor. Isso garante que o laço termina e que o programa também termina.

Somando +1 no vetor de bits, faz com que ele represente o próximo subconjunto do conjunto $\{1, \dots n\}$. Como a cada iteração é verificado se a quantidade de bits 1 é igual a k, e isso é o critério de parado do laço da soma, é evidente que o laço só parará de executar quando o próximo subconjunto de tamanho k for encontrado.

Portanto pode-se concluir que este algoritmo para e devolve a solução do problema.