Problema da Mochila Relacional

André Almeida *164047*
MC658 - Lab 02

O problema

A entrada do problema é dada por:

- Número de itens;
- Matriz de relações de itens;
- Vetor de valor de itens;
- Vetor de peso de itens;
- A capacidade da mochila e
- O tempo máximo disponível para o cálculo da solução

Todos os resultados mostrandos foram executados em uma máquina com processador Intel i7 de 3ª geração.

Algoritmo exato

A abordagem exata para o problema foi criar uma estrutura de solução global, com a soma dos itens e um vetor de índices de itens da solução. Esta estrutura é atualizada sempre que é encontrada uma solução melhor.

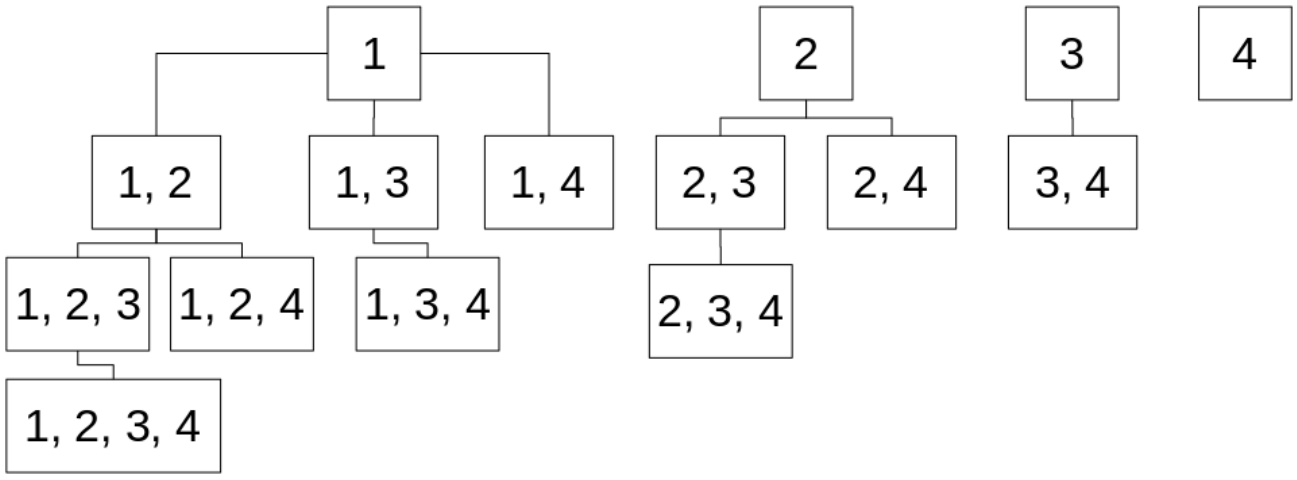
O algoritmo cria uma árvore de combinações de índices para cada item da entrada. As combinações geradas por cada nó inserem somente índices que sejam maiores que o último índice do nó pai, de forma a evitar permutações. Por exemplo, seja n o número de itens e dado um nó com índices, $i_1, i_2, i_3, ..., i_j$, onde $i_k < i_{k+1}$, esse nó gerará $n - j$ folhas. Para cada $m \in [j + 1, n]$, o nó ira uma folha inserindo em cada folha os índices do nó pai ($i_1, i_2, i_3, ..., i_j$) mais o índice m.

Se os itens de um nó não cabem na mochila, esse nó vira uma folha e não gera

mais combinações. Por exemplo, se um conjunto de itens (i_1, i_3, i_7) já ocupa a capacidade máxima da mochila, nenhum item i_k (já que o peso de i_k é maior que 0) será capaz de ocupar a mochila e não faz sentido calcularmos soluções com o conjunto inicial.

Agora, dada uma raíz (que pela definição anterior possúi apenas um item), verificamos se este item cabe na mochila e, se cabe, chamamos uma rotina para calcular os valores de seus nós-filhos. Essa rotina passa por parâmetro o valor do item e a capacidade total da mala subtraída do peso do item. Para cada filho que couber na mala, é calculada a soma dos itens selecionados (levando em conta a soma das relações entre os itens) e é feita uma chamada recursiva passando a soma atual e reduzindo a capacidade da mala com o peso do novo item.

Por exemplo, para uma entrada com 4 itens, o algoritmo irá gerar a seguinte árvore de combinações:



Implementação

A implementação do algoritmo é feita da seguinte maneira, começando pela rotina principal:

> *1.* Ordenar decrescentemente os `n` itens pelo seu *valor relativo*[1] e iniciar o alarme com o tempo máximo
> *2.* Criar uma estrutura global `solução`, com a soma igual à 0 e o conjunto de itens vazio

> *3.* Chamar a rotina `calculaSolução(items, valores, pesos, relações, capacidade)`
> *4.* Quando a rotina acabar ou quando o alarme acabar, retorna o conjunto de items e a soma da solução atual

[1]O *valor relativo* é calculado fazendo a soma do valor de um item mais todas os valores de suas relações e dividindo pelo seu peso:

> valor_relativo = (somaTodasRelações(item) + valor[item]) / peso[item]

Usamos um vetor ordenado para, no caso em que o tempo estourar, ter uma chance que tenhamos começado por um item que venham a ter soluções mais próximas da ótima.

A rotina `calculaSolução(...)` por sua vez é definida por:

> *1.* Para cada índice `i` no vetor ordenado:
> *2.* Verificar se o item cabe na mochila e, se couber:
> *3.* Cria uma solução com apenas este elemento e compara com a solução global. Se for melhor, substituiu ela.
> *4.* Se ainda houver espaço na mochila, chama a rotina `calculaFilhos([i], valor[i], capacidade - v[i])`

Nessa rotina `calculaSolução(...)` criamos soluções com apenas um item e, caso ainda exista espaço disponível, chamamos uma subrotina para verificar as combinações que são "filhas" desta rotina, de forma a percorrer todas as combinações que são válidas. Para cada `i`, passamos o `valor[i]` como uma soma parcial e reduzimos a capacidade da mochila subtraindo o peso de `i`. A rotina `calculaFilhos(itens[], soma_parcial, capacidade)` funciona de maneira semelhante, mas sem a etapa de buscar os itens ordenados de forma a evitar consultas a vetor que iriam pesar no desempenho do algoritmo.

Rotina `calculaFilhos(itens[], soma_parcial, capacidade_parcial)`:

> *1.* Para `j`, começando no último índice de `itens[] + 1` até `número total de itens`:
> *2.* Se o alarme esgotou, retornamos a solução global do momento
> *3.* Se o `peso[j]` cabe na mochila:

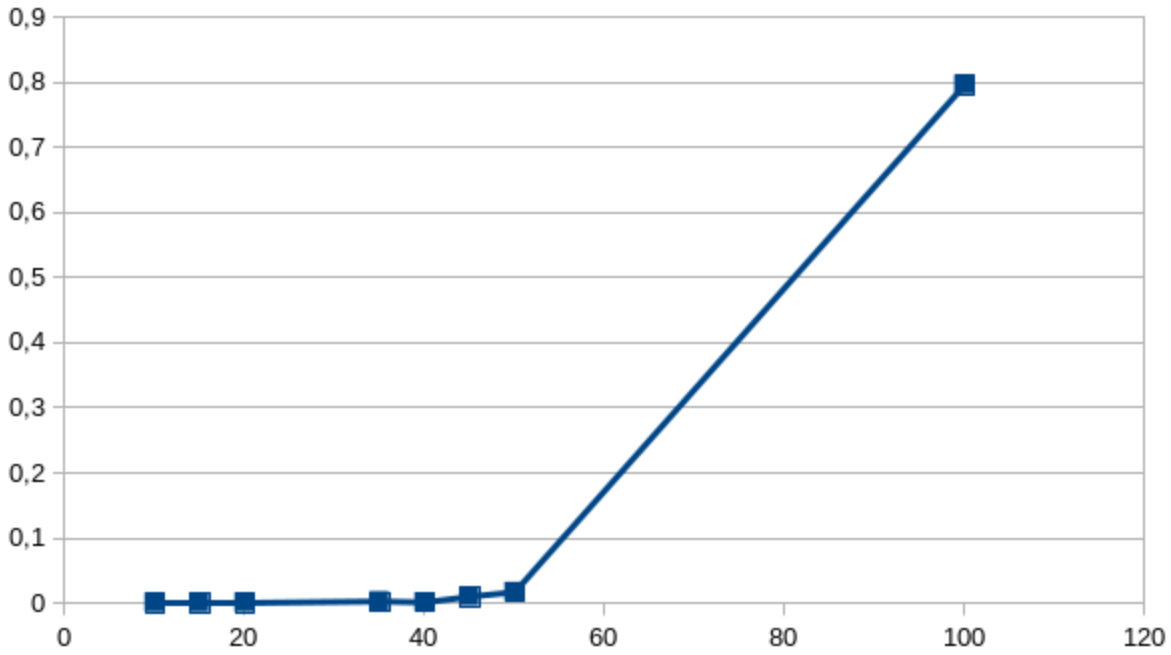
> 4. Adiciona o `j` no vetor `itens`
> 5. Calculamos a `soma_atual`, somando a `soma_parcial` + `valor[j]` + `valorRelacoes(j, itens)`
> 6. Se esta soma for maior do que a solução global, substituiu a global por ela
> 7. Se ainda existe espaço na mochila:
> 8. Chamamos `calculaFilhos(itens[], soma_atual, capacidade - peso[j])`
> 9. Retiramos `j` do vetor `itens`

O cálculo na linha *5* de `valorRelacoes()` é feito tomando cuidado para não somar duas vezes a mesma relação. Somamos apenas as relações na coluna da matriz do item novo, pegando os itens que já estão na mocilha. Como a coluna do item novo nunca foi percorrida antes, os itens que iremos somar não serão repetidos.

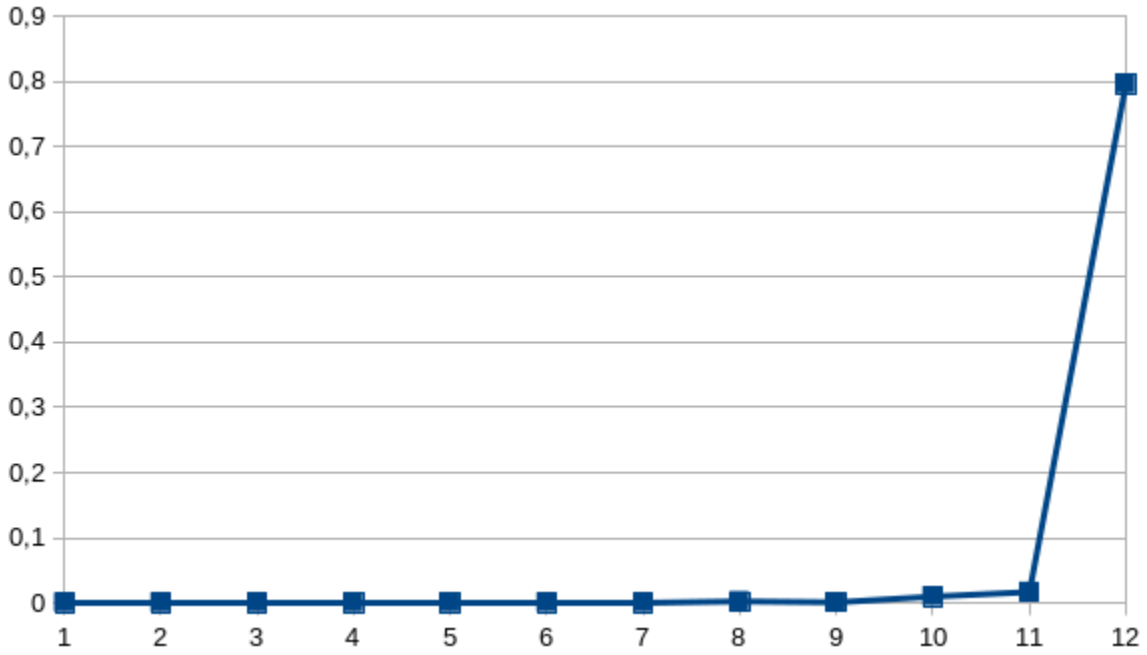
Resultados

Casos pequenos

Para os casos de teste de 01 à 12, o algoritmo exato executou abaixo de um segundo e obteve a resposta esperada. O tempo de execução do algoritmo crescia exponencialmente conforme o número de itens da entrada, como é possível ver nos gráficos abaixo:



Número de itens x Tempo de execução



Caso teste x Tempo de execução

Casos grandes

Algoritmo aproximado