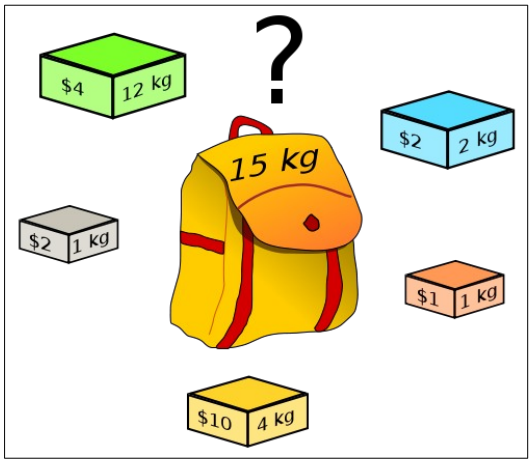
Sistemas Operativos
2019 / 2020

Licenciatura em Engenharia Informática

Trabalho Prático #1

Introdução

O problema do *Knapsack*, ou da mochila, é um problema de optimização combinatória. O seu nome tem a ver com o problema de alguém que é limitado por uma mochila de tamanho limitado e deve preenchê-la com os itens mais valiosos (https://en.wikipedia.org/wiki/Knapsack_problem).



De uma forma simples, dado um conjunto de vários itens, cada um com um determinado peso e valor, o objectivo é determinar quais os itens que devem ser incluídos numa coleção de modo a que o peso total seja menor ou igual a um determinado limite e o valor total seja o maior possível.

O problema mais comum é o *0-1 Knapsack* que restringe o número de cópias $\mathbf{x_i}$ de cada item aos valores **zero** e **um**. Dado um conjunto de ***n*** itens, cada um com um peso $\mathbf{w_i}$ e um valor $\mathbf{v_i}$, juntamente com a capacidade máxima da mochila **W**, o objectivo é:

$$maximizar \quad \sum_{i=1}^{n} v_i x_i$$

$$sujeito\ a \quad \sum_{i=1}^{n} w_i x_i \leq W \quad com \quad x_i \in \{0,1\}$$

Portanto o objectivo é maximizar a soma dos valores dos itens que estão dentro da mochila de modo a que a soma dos seus pesos seja menor ou igual à capacidade da mochila.

Algoritmos de resolução

O algoritmo mais óbvio é o algoritmo de força bruta, em que todas as soluções são testadas até se encontrar uma solução válida (com peso $\leq W$) com o melhor valor possível.

Infelizmente a complexidade deste algoritmo é de $O(2^n)$. Por exemplo, para $n=50$, o número de valores possíveis para a tabela é de 2^{50} o que torna o seu cálculo bastante lento. Para valores superiores a $n=50$, o tempo de resolução torna-se praticamente incomportável.

O algoritmo proposto para este trabalho é o algoritmo **AJ KP Aleatório (AJ-KPA)**, com o seguinte funcionamento:

1. Inicializa-se uma tabela de itens de forma aleatória. Esta tabela poderá corresponder a uma solução inicial inválida.

2. Aplica-se uma operação que remove *n1* itens na mochila e de seguida insere *n2* itens. Os valores *n1* e *n2* deverão ser aleatórios. Note que poderão resultar soluções inválidas.

3. Calcula-se a soma dos valores de todos os itens incluídos na mochila, assim como a soma dos seus pesos.

4. O algoritmo volta ao ponto 2 enquanto não houver uma condição de término dada pelo tempo ou número de iterações máxima.

No fim, o algoritmo deverá ser capaz de retornar o valor máximo da soma dos valores dos itens que estão dentro da mochila, encontrado durante a sua execução. Este valor máximo só deve ser calculado dentro das soluções válidas!

Note que a melhor solução encontrada pelo algoritmo pode ou não ser a melhor solução em termos globais.

Implementação concorrencial do algoritmo AJ-KPA

Dado que o algoritmo AJ-KPA tem uma forte componente aleatória, um dos grandes factores que pode influenciar a solução é o número de iterações realizadas pelo algoritmo (ou de forma indirecta, o tempo que se dá ao algoritmo para tentar encontrar a melhor solução).

Desta forma, propomos a implementação paralela e concorrencial do algoritmo nas suas versões *Base* e *Avançada*.

Versão Base

1. Criar *m* processos (número parametrizável) em que cada processo corre o algoritmo AJ-KPA.

2. À medida que cada processo encontra uma solução melhor que a anterior, coloca a sua solução na memória partilhada.

3. Dado que dois ou mais processos podem aceder simultaneamente à memória partilhada e corrompê-la, a actualização desta deve ser feita de forma concorrencial.

4. Ao fim de algum tempo, termina-se a execução e informa-se o utilizador da melhor solução encontrada.

Versão Avançada

A versão avançada é semelhante à versão base com as seguintes alterações:

1. À medida que cada processo encontra uma solução melhor que a anterior, coloca a sua solução na memória partilhada e envia um sinal ao processo principal avisando-o que houve uma actualização da melhor solução.

2. O processo principal, sendo informado que um dos processos encontrou uma solução melhor, deverá enviar um sinal a todos os processos filhos de modo a estes actualizarem internamente as suas melhores soluções a partir da actual.

3. Todos os processos passarão a estar com a mesma solução, e todos deverão resumir a sua actividade a partir daí.

Dada a possibilidade que um processo pode estar no meio da execução do algoritmo AJ-KPA e ser informado da actualização de caminho por parte do processo pai (invalidando ou corrompendo os dados) considere utilizar mecanismos de sincronização, como por exemplo semáforos e mutex.

Desenvolvimento

A aplicação deverá ser feita na linguagem de programação C, em Linux, usando as técnicas de programação concorrencial utilizadas nas aulas laboratoriais, nomeadamente *fork*, funções de manipulação de memória partilhada, semáforos, mutex, etc.

Entradas

A entrada de informação é feita usando ficheiros de texto, um para cada problema.

Cada ficheiro de texto está separado por linhas, em que na primeira linha é dado o número de itens ***n***, na segunda linha a capacidade máxima ***W*** da mochila, e nas restantes *n* linhas é dado um par de valores que corresponde ao valor e peso de cada item. Na última linha está o valor óptimo que se pretende obter para esse problema.

```
4
11
6 2
10 4
12 6
13 7
23
```

O programa deverá ser capaz de ser invocado pela linha de comandos passando como argumentos o número de ordem do teste, o nome do ficheiro de texto com o problema, o número de processos a

serem criados e o tempo máximo de execução do algoritmo (em segundos). Por exemplo, o comando ***kp 1 ex23.txt 10 60*** deverá executar o teste número 1 do ficheiro de teste "ex23.txt" usando 10 processos em paralelo durante 60 segundos.

Resultados

De modo a se validar a qualidade do algoritmo, deverá ser construída uma tabela com as seguintes colunas:

a) Número do teste (de 1 a 10).
b) Nome do teste e número de itens.
c) Tempo total de execução.
d) Número de processos usado (parâmetro *m* na descrição dos algoritmos).
e) Melhor valor da soma dos itens encontrado.
f) Valor da soma dos pesos da melhor solução.
g) Número de iterações necessárias para chegar ao melhor valor encontrado.
h) Tempo que demorou até o programa atingir o melhor valor encontrado.

Cada teste deverá ser repetido 10 vezes para os mesmos parâmetros de entrada, e deverá ser possível obter valores médios de tempo e número de iterações, assim como o número de vezes em que se encontrou a soma óptima.

Os ficheiros de teste a utilizar serão disponibilizados no *moodle* da disciplina, assim como um exemplo de um ficheiro com resultados e respectivas estatísticas.

Entrega e avaliação

Os trabalhos deverão ser realizados em grupos de 2 alunos da mesma turma de laboratório, e deverão ser originais. Trabalhos plagiados ou cujo código tenha sido partilhado com outros serão atribuídos nota **zero**.

Todos os ficheiros deverão ser colocados num **ficheiro zip** (com o número de todos os elementos do grupo) e submetido via *moodle* **até às 23:55 do dia 01/Dezembro/2019**. Deverá também ser colocado no zip um **relatório em pdf** com a identificação dos alunos, as tabelas de resultados e a descrições das soluções que considerarem relevantes. Este documento deverá ser mantido curto e directo (2-3 páginas).

Irá considerar-se a seguinte grelha de avaliação:

Algoritmo AJ-KPA	04 val.
Algoritmo concorrencial	
Versão Base	04 val.
Versão Avançada	03 val.
Outra versão original	02 val.
Utilização de memória partilhada	01 val.

Utilização de semáforos	01 val.
Relatório com a tabela de testes	02 val.
Qualidade da solução e código	03 val.

As discussões dos projectos serão realizadas na semana seguinte à entrega do projecto, no horário das aulas laboratoriais. As notas poderão ser atribuídas aos alunos de forma individual.

Bom trabalho!