MC658 - Projeto e Análise de Algoritmos III

Prof. Flávio Keidi Miyazawa / PED: Edson Ticona Zegarra

Laboratório 2 - 1o. Semestre de 2017

Os sistemas de comunicação móveis são baseados na utilização eficiente da largura de banda B (bandwidth em inglês). Infelizmente, a largura de banda é um recurso limitado e, portanto, deve ser distribuída da melhor maneira possível entre os usuários, maximizando assim o lucro obtido pelas companhias telefônicas.

Um usuário pode ser descrito por uma classe c; um valor p, representando o pagamento pelo serviço, e um requerimento de largura de banda w. Quando o sistema faz a distribução da largura de banda, temos a restrição de que dois usuários de classes diferentes não podem estar juntos. Assim, deve existir uma separação d entre eles. Isto ocorre por conta da interferência que dois canais com exigências diferentes podem causar um ao outro, e com isso, temos os intervalos de guarda.

Você é contratado para projetar dois algoritmos: um de **backtracking** e um **branch and bound** que distribui os usuários da melhor maneira possível.

Entrada

A primeira linha contém os parâmetros do problema: n, d e B, indicando a quantidade de usuários, o tamanho da separação (ou divisor) e a largura total, respectivamente. Cada uma das n linhas seguintes tem tres inteiros positivos, p w c, indicando o pagamento, a largura e a classe do usuário.

Saída

Um vetor binário de n elementos, sendo que o usuário da posição i possui valor 1 se foi atribuido á solução, e 0 caso contrário, para $i = 0, 1, \ldots, n-1$.

Exemplo de entrada	**Exemplo de saída**
5 2 20 4 6 1 5 9 1 3 3 2 8 7 2 6 8 3	0 0 1 1 1

Para o exemplo, o valor ótimo é $p_3 + p_4 + p_5 = 17$. Repare que a solução $(1, 0, 1, 0, 1)$ não é viável pois mesmo que a soma dos pesos w_1, w_3 e w_5 seja 17 (e portanto abaixo da capacidade da mochila), será necessário usar dois divisores pois os três usuários são de classes diferentes. Com isso, será necessário uma largura total de $17+2\cdot 2 = 21$, e claramente não é viável, pois a capacidade da mochila é 20. Na Figura 1 temos uma distribução dos usuários.

Execução

Para compilar o código pode se usar o comando:

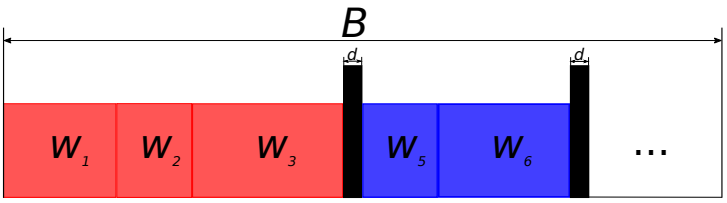

Figure 1: Exemplo de como os usuários são distribuidos. Usuários de classes diferentes (cores distintas na imagem) têm que estar separados por um divisor de largura d. Então tem que se maximiar o lucro, i.e. $\max \sum_{i=1}^{n} p_i x_i$ onde x_i é 1 se o i-ésimo usuário i foi atribuído à solução, e 0 caso contrário.

```
$ make
```

Para executar o seu código, pode se usar a seguinte sintaxe:

```
$ ./knapsack -i <input_filename> [-bt|-bb] -t <max_time>
```

onde `<input_filename>` é o nome do arquivo de entrada; `-bt` é usado para resolver o problema utilizando a rotina de backtracking e `-bb` é usado para resolver o problema utilizando a rotina de branch and bound; e `<max_time>` é o tempo em segundos permitido para se executar a respectiva rotina. Por exemplo, no comando seguinte

```
$ ./knapsack -i 5_2_20.in -bt -t 10
```

O arquivo de entrada tem o nome `5_2_20.in`, cujo problema será resolvido através da rotina de backtracking com tempo máximo de 10 segundos.

Código

O arquivo `knapsack_bt_bnb.cpp` contém as funções:

- `bool knapsack_bt (int n, int d, int B, vector<int> &p, vector<int> &w, vector<int> &c, vector<int> &sol, int t)`
- `bool knapsack_bnb (int n, int d, int B, vector<int> &p, vector<int> &w, vector<int> &c, vector<int> &sol, int t)`

onde

- `n`: contém o número de usuários (número inteiro não negativo)
- `d`: contém o tamanho da divisão (número inteiro não negativo)
- `B`: contém a largura total (número inteiro não negativo)
- `p`: contém o vetor de pagamentos (número inteiro não negativo)
- `w`: contém o vetor de larguras (número inteiro não negativo)
- `c`: contém o vetor de classes (número inteiro onde a menor classe é 0)
- `t`: contém o tempo máximo permitido, em segundos, para a execução da rotina)

onde vocês devem implementar os algoritmos de **backtracking** e **branch and bound**, respectivamente. As funções devolvem `true` se acharam uma solução ótima e a variável `sol` é o vetor solução, representado por um vetor binário que indica os itens que pertencem à solução. As funções devolvem `false` caso a respectiva rotina tenha parado pela limitação de tempo, e neste caso, a variável `sol` contém a melhor solução encontrada até o momento.

Relatório

O relatório deve apresentar a descrição clara dos dois algoritmos implementados, em alto nível e com descrição clara das principais características dos métodos implementados, como a forma de ramificação e limitantes usados. O relatório também deve apresentar comparações computacionais mostrando o tempo utilizado e qualidade das soluções encontradas pelos dois algoritmos para diversas entradas. Para isso, apresente uma tabela com colunas indicando o nome do arquivo de entrada, tempo limite utilizado, e para cada algoritmo/rotina, o valor da solução obtida pela rotina seguido pelo tempo de execução da rotina; valor 1 ou 0, se a rotina obteve solução ótima ou se parou por tempo.

Os trabalhos podem ser feitos individualmente ou em duplas, porém neste caso, seu dupla não pode ser o mesmo dupla dos últimos dois laboratórios.

As implementações com os melhores desempenhos terão um bônus adicionados na nota final do laboratório: de 1.5 pontos para o melhor; 1.0 pontos para o segundo melhor; e 0.5 pontos para o terceiro melhor.

A **data de entrega máxima é até o final do dia 26/04/2017.**