U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Desenho de Algoritmos

Primeiro Trabalho de Grupo (T1)

Bernardo Ferreira, up201806581
Daniela Tomás, up202004946
Hélder Bessa, up201503035
LEIC • 2021/2022

Descrição do Problema

- Neste trabalho, foi-nos pedido para resolver problemas de minimização e maximização de entregas de mercadorias por uma empresa de logística a partir de um armazém.
- Nesta empresa, existem dois tipos de entregas, normais e expresso. Para as entregas normais, que são as encomendas que têm mais pedidos e não precisam de ser entregues no mesmo dia, a empresa recorre à subcontratação de estafetas. As entregas expresso têm uma viatura no armazém de capacidade unitária.
- Foi-nos proposto resolver três cenários: um que otimiza o numero de estafetas, outro que otimiza o lucro da empresa e o último que otimiza as entregas expresso.

Cenário 1: Formalização

- Dados de entrada:

  - Conjunto E com n estafetas, e_1, e_2, …, e_n:

    id → cada estafeta tem um id único   c → cada estafeta tem um custo associado

    v → cada estafeta tem um volume máximo associado   w → cada estafeta tem um peso máximo associado

  - Conjunto P com n pedidos normais:

    id → cada pedido tem um id único r_p → cada pedido tem uma recompensa associada

    v_p → cada pedido tem um volume w_p → cada pedido tem um peso t_p → tempo estimado de entrega

- Resultados:

  - numDeliverMen → numero de estafetas necessários para poder entregar todas as encomendas no armazém.

- Variáveis de decisão:

  - Entregas que foram realizadas.
  - Os estafetas usados para as entregas.

- Função objetivo:

  - número mínimo de estafetas necessários
  - minimização $\sum_{i=1}^{DeliverMan}(e_i)$

- Restrições e domínios de valores para as variáveis:

  - todos os valores tem de ser positivos.

Cenário 1: Algoritmos Relevantes

- Estruturas de dados usados:

  - std::vector → Redimensiona-se automaticamente e, para além disso, tem vários métodos nativos úteis para a resolução do problema.

  - std::set → Usado para guardar elementos únicos e cada elemento é ordenado. Útil para este algoritmo para ir guardando os estafetas já usados.

- Algoritmo(s): First Fit Decreasing

  → Bin Packing

  → Minimização do número de estafetas usados.

  → É usado um algoritmo de aproximação.

```
Inputs: P = {p_1, p_2, ..., p_n}; E = {e_1, e_2, ..., e_n}
sort(P)

R = {}
For each p_i in P do:
   For each ei in E do:
      If remaningVolume(e_i) - volume(p_i) < 0
      and remainingWeight(e_i) - weigth(p_i) <0 then:
            Continue
      EndIf
      remaningVolume(e_i) = remaningVolume(e_i) – volume(p_i)
      remainingWeight(e_i ) =  remainingWeight(e_i) - weigth(p_i)
      R = R ∪ {e_i}
      Endfor
 Endfor
```

Cenário 1: Complexidade

First fit decreasing:

std::sort no vetor de entregas → $\Theta(N \log_2 N)$

Loop que percorre encomendas → $\Theta(N)$

Loop que percorre os estafetas → $\Theta(T)$

Nested Loops → $\Theta(N * T)$

Temporal:
- $\Theta(N * T)$

Espacial:
- $\Theta(N)$

Cenário 1: Resultados

```
Choose a Scenery:1
 Number of used deliver man: 11
```

Teste usando os datasets "carrinhas_test_1" e "encomendas_test_1"

```
Choose a Scenery:1
 Number of used deliver man: 16
```

Teste usando os datasets "carrinhas_test_2" e "encomendas_test_2"

```
Choose a Scenery:1
 Number of used deliver man: 24
```

Teste usando os datasets "carrinhas" e "encomendas"

Cenário 2: Formalização

- Dados de entrada:
  - Conjunto E com n estafetas, $e_1, e_2, \ldots, e_n$:

    id → cada estafeta tem um id único   c – cada estafeta tem um custo associado

    v → cada estafeta tem um volume máximo associado   w – cada estafeta tem um peso máximo associado
  - Conjunto P com n pedidos normais:

    id → cada pedido tem um id único r_p → cada pedido tem uma recompensa associada

    v_p → cada pedido tem um volume w_p → cada pedido tem um peso t_p → tempo estimado de entrega
- Resultados:
  - Lucro → diferença entre a receita total dos pedidos entregues e a despesa correspondente ao custo total de cada estafeta.
- Variáveis de decisão:
  - Entregas que foram realizadas e os estafetas usados para as entregas.
- Função objetivo:
  - Lucro máximo da empresa.
- Restrições e domínios de valores para as variáveis:
  - todos os valores tem de ser positivos.

Cenário 2: Algoritmos Relevantes

- Estruturas de dados usados:
  - std::vector → Redimensiona-se automaticamente e, para além disso, tem vários métodos nativos úteis para a resolução do problema.
  - std::set → Usado para guardar elementos únicos e cada elemento é ordenado.

```
Inputs: P = {p_1, p_2, ..., p_n}; E = {e_1, e_2, ..., e_n}
sort(P,racio)
sort(E,cost)
For each p_i in P
   For each e_i in E
        If deliverMan.addDeliver(p_i) then
           payment ← p_i.getReward()
            If usedDeliverMen.find(e_i) == usedDelivermen.end() then
                cost ← e_i.getCost()
            End If
            UsedDeliverMen.insert(e_i)
            Break
        End If
    End for
End for
```

Cenário 2: Complexidade

std::sort no vetor de entregas -> Θ(N log2 N)

std::sort no vetor dos estafetas -> Θ(N log2 N)

Loop que percorre encomendas -> Θ(N)

Loop que percorre os estafetas -> Θ(T)

Temporal:

- Θ(N * T)

Espacial:

- Θ(N)

Cenário 2: Resultados

```
Choose a Scenery:2
 Profit: 103760
```

Teste usando os datasets "carrinhas_test_1" e "encomendas_test_1"

```
Choose a Scenery:2
 Profit: 65201
```

Teste usando os datasets "carrinhas_test_2" e "encomendas_test_2"

```
Choose a Scenery:2
 Profit: 111726
```

Teste usando os datasets "carrinha" e "encomendas"

Cenário 3: Formalização

- Dados de entrada:
  - Conjunto P com n pedidos expresso, $p_1, p_2, \ldots, p_n$:

    id → cada entrega tem um id único r_p → recompensa do pedido v_p → volume do pedido

    w_p → peso do pedido t_p → tempo estimado de entrega
- Resultados:
  - avgTime → tempo médio de execução maximizando o número de entregas (9h00-17h00 → 28800 segundos)
- Variáveis de decisão:
  - Entregas a serem realizadas: $p_1, p_2, \ldots, p_{maxDelivers}$
  - Duração das tarefas a serem realizadas: $f_1 = t_{p1}, f_2 = f_1 + t_{p2}, \ldots, f_{maxDelivers} = f_{maxDelivers\,-\,1} + t_{p(maxDelivers\,-\,1)}$
- Funções objetivo:
  - Máximo de entregas num dia
  - avgTime: $\sum_{i=1}^{maxDelivers}(f_i/maxDelivers)$
- Restrições e domínios de valores para as variáveis:
  - todos os valores devem ser positivos
  - $\sum_{i=1}^{maxDelivers}(f_i/maxDelivers) \leq 28800$ segundos, $f_i \in]0,28800]$
  - $p_1, p_2, \ldots, p_n$ devem ser ordenados por duração e, caso algum pedido não seja realizado, é descartado e volta para o fornecedor passando a ficar com maxDelivers ≤ n pedidos, $p_1, p_2, \ldots, p_{maxDelivers}$

Cenário 3: Algoritmos Relevantes

- Estrutura(s) de dados selecionada(s):

  - std::vector → redimensiona-se automaticamente e, para além disso, tem vários métodos nativos úteis para a resolução do problema

- Algoritmo(s):

  - Escalonamento de atividades (minimizar tempo médio):
    → Garante uma solução ótima
    → Apenas se consegue resolver este cenário com um algoritmo ganancioso

```
Inputs: P = {p_1 , p_2, …, p_i, …, p_n }
sort(P.begin(),P.end(),compareDuration)
sum ← 0
maxDelivers ← 0
For i = 1 to P.size() do
    If sum + p_i.getDuration() ≤ 28800 then
        sum ← sum + p_i.getDuration()
        maxDelivers ← maxDelivers + 1
    Else
        P ← P.erase(P.begin()+i,P.end())
        break
    EndIf
EndFor
avgTime ← sum / maxDelivers
Return avgTime
```

Cenário 3: Complexidade

Temporal:

std::sort$\rightarrow \Theta\,(N \log_2 N)$

ciclo de 1 até n $\rightarrow \Theta\,(N)$

vector::size() $\rightarrow \Theta\,(1)$

getDuration() $\rightarrow \Theta\,(1)$

vector::erase $\rightarrow \Theta\,(N)$

- Melhor caso: $\Theta\,(N)$
- Pior/médio caso: $\Theta\,(N \log_2 N)$

Espacial:

- $\Theta\,(N)$

Cenário 3: Resultados

```
Choose a Scenery:3
 Average time: 282.87 seconds
```

Teste usando os datasets "carrinhas_test_1" e "encomendas_test_1"

```
Choose a Scenery:3
 Average time: 431.15 seconds
```

Teste usando os datasets "carrinhas_test_2" e "encomendas_test_2"

```
Choose a Scenery:3
 Average time: 231.06 seconds
```

Teste usando os datasets "carrinha" e "encomendas"

Solução Algorítmica a Destacar

- Algoritmos gananciosos:

- Apesar de nem sempre ser possível encontrar uma solução ótima, os algoritmos gananciosos são úteis para resolver problemas de otimização e também são simples de implementar e interpretar.

Conclusão / Autoavaliação

- Logo no inicio do trabalho, pensámos em resolver cada cenário usando diferentes tipos de algoritmos. No primeiro cenário, baseamo-nos no problema de Bin Packing adaptando o algoritmo First Fit Decreasing. No segundo cenário, inspiramo-nos no Problema da Mochila que usa um algoritmo ganancioso. Por fim, no terceiro cenário, decidimos basear-nos no problema Escalonamento de Atividades, resolvendo-o também com um algoritmo ganancioso.
- A nosso ver, a parte mais exigente foi associar cada cenário a um algoritmo. Apesar disso, achamos que o objetivo principal do trabalho foi alcançado.

Bernardo Ferreira - 33,(3)%

Tarefas:

- Descrição do problema
- Primeiro cenário
- Conclusão

Daniela Tomás - 33,(3)%

Tarefas:

- Descrição do problema
- Terceiro cenário
- Conclusão

Hélder Bessa - 33,(3)%

Tarefas:

- Descrição do problema
- Segundo cenário
- Conclusão