Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Algoritmos e Estruturas de Dados

Semestre de Inverno 2009/10

Trabalho Final

---

1 Introdução

O objectivo deste trabalho é a realização dum programa para a resolução de instâncias do *problema do escalonamento de tarefas de tempo constante*, caracterizado por:

- Uma *tarefa* consiste numa acção que demora uma unidade de tempo a ser executada.
- Existe apenas *uma unidade de execução* de tarefas, ou seja, a execução das tarefas tem de ser realizada em série.
- Dado um conjunto de tarefas, um *escalonamento* define uma ordem de execução destas tarefas. A primeira tarefa no escalonamento *conclui* a sua execução na unidade de tempo 1; a segunda tarefa *conclui* na unidade de tempo 2, etc.
- Cada tarefa tem associado um *prazo* e uma *penalização*.
- O objectivo deste problema é obter um escalonamento que minimize a soma das penalizações associadas às tarefas que concluem a sua execução após o seu prazo.

Formalmente, uma instância do problema é descrita por:

- um conjunto $T = \{t_1, \ldots, t_n\}$ de n tarefas identificadas por $t_i \in \mathbb{N}$, com $1 \leq i \leq n$;
- uma sequência de n prazos $d_i \in \mathbb{N}$, com $1 \leq i \leq n$, $1 \leq d_i \leq n$ e tal que d_i é o prazo da tarefa t_i;
- uma sequência de n penalizações $p_i \in \mathbb{N}$, com $1 \leq i \leq n$ e tal que p_i é a penalização associada à tarefa t_i, caso esta não seja concluída dentro do prazo d_i.

Dado um conjunto de tarefas T, um escalonamento é uma permutação do conjunto T. Num escalonamento S, uma tarefa designa-se por *a tempo* caso a sua conclusão seja menor ou igual ao seu prazo. Caso contrário, a tarefa designa-se por *em atraso*.

O objectivo deste problema é a obtenção de escalonamentos que minimizem a soma das penalizações das tarefas em atraso. Note-se que:

- minimizar a penalização das tarefas em atraso equivale a maximizar a penalização das tarefas a tempo;
- a penalização associada a uma tarefa em atraso é independente do valor desse atraso.

Recomenda-se a leitura da secção 16.5 do livro de referência, até à apresentação do lema 16.12.

1.1 Exemplo

Apresenta-se de seguida um exemplo concreto duma instância deste problema.

Uma associação humanitária acaba de receber um carregamento de contentores com bens de durabilidade limitada, para distribuir junto de uma comunidade necessitada. Cada contentor contém bens com um prazo de entrega comum e tem um valor associado. Contudo, a comunidade em questão fica a um dia de viagem (ida e volta) e só é possível transportar um contentor de cada vez. Por cada contentor não entregue dentro do seu prazo, perde-se o seu valor total.

Neste caso, objectivo é determinar uma sequência óptima de transporte de contentores, isto é, uma sequência que minimize a soma dos custos (penalizações) dos contentores cuja entrega só pode ser concluída após o prazo de bens contidos.

2 Funcionalidades a implementar

As funcionalidades a implementar são as seguintes:

- Comando para carregar a informação das tarefas presentes num ficheiro de texto, dado o nome do ficheiro. Na primeira linha deste ficheiro está presente o número n de tarefas descritas nesse ficheiro. As n linhas seguintes descrevem cada uma das tarefas, sendo constituídas por três inteiros:
  1. identificador t_i da tarefas, em que $1 \leq i \leq n$;
  2. prazo d_i da tarefa;
  3. penalização p_i associada à tarefa, caso seja concluída em atraso.
- Comando para procurar e apresentar um escalonamento óptimo, recorrendo ao algoritmo GETSEQUENCE1.
- Comando para procurar e apresentar um escalonamento óptimo, recorrendo ao algoritmo GETSEQUENCE2.

A apresentação dos escalonamentos deve incluir o tempo de de processamento do algoritmo e a soma das penalizações das tarefas em atraso nesse escalonamento.

3 Algoritmos e Estruturas de Dados

Nesta secção apresentam-se os algoritmos referidos, assim como a estrutura de dados para *conjuntos disjuntos* utilizada no algoritmo GETSEQUENCE2.

3.1 GETSEQUENCE1

O algoritmo ganancioso GETSEQUENCE1 calcula uma sequência S óptima e define-se da seguinte forma:

- Entrada: um conjunto de tarefas $T = \{t_1, \ldots, t_n\}$ em que cada tarefa t_i tem associada um prazo d_i e um peso p_i.
- Saída: uma permutação de T, ou seja uma sequência de execução.
- Algoritmo:
  1. Iniciar as sequências S e R vazias (note que a sequência S, estando vazia, não tem tarefas em atraso)
  2. Para cada tarefa t em T, por ordem decrescente das penalizações p_i:
     2.1. Se for possível inserir t em S de forma a continuar a não existir tarefas em atraso em S, então inserir t em S por ordem crescente de prazo.
     Caso contrário, inserir t no final de R.
  3. Inserir as tarefas em R no final de S.
  4. Retornar S.

3.2 Conjuntos disjuntos

O algoritmo GETSEQUENCE2 requer a manipulação de *colecções de conjuntos disjuntos*. No contexto deste trabalho, estes conjuntos disjuntos vão ser constituídos por tarefas. As secção 21.1 e 21.3 do livro de referência caracterizam estas colecções e apresentam algoritmos eficientes para a sua representação e manipulação.

Em seguida, apresenta-se apenas a caracterização funcional destas colecções:

- Cada conjunto da colecção é identificado por um dos seus elementos, designado por *elemento representante*.
- A operação MAKESET(x) cria e insere na colecção o conjunto constituído apenas pelo elemento x;
- A operação UNION(x, y) cria e insere na colecção o conjunto $T_x \cup T_y$, onde T_x é o conjunto da colecção que contem x e T_y é o conjunto que contém y. Note-se que, porque a colecção é constituída por conjuntos *disjuntos*, os conjuntos T_x e T_y são também removidos da colecção.

- A operação FINDSET(x) retorna o elemento representante do conjunto presente na colecção e que contém o elemento x.

No contexto do algoritmo GETSEQUENCE2, a colecção deve também suportar as seguintes operações, específicas de conjuntos disjuntos de tarefas:

- SIZE(x), que retorna o número de tarefas presentes no conjunto que contém x;
- LARGEST(x) que retorna o maior prazo associada a qualquer das tarefas presentes no conjunto que contém x.

A descrição de algoritmos eficientes para a representação e manipulação de colecções de conjuntos disjuntos está presente na secção 21.3 do livro de referência.

3.3 GETSEQUENCE2

A ideia subjacente ao algoritmo GETSEQUENCE2 é atribuir tarefas directamente a posições de S, inicialmente vazias, por forma a minimizar a soma do peso das tarefas que não possam ser efectuadas a tempo. Para cada tarefa t_i, por ordem decrescente da penalização:

1. Tenta-se colocar t_i na maior posição de S menor ou igual que o prazo d_i.
2. Caso esta colocação seja impossível (todas as posições menores ou iguais que d_i estão ocupadas), insere-se t_i na última posição livre de S.

O algoritmo ganancioso GETSEQUENCE2, apresentado em seguida, usa colecções de conjuntos disjuntos para determinar de forma eficiente a maior posição de S menor ou igual que o prazo d_i. A ideia subjacente é a de que cada conjunto disjunto representa uma sequência de tarefas escalonadas contiguamente.

- Entrada: um conjunto de tarefas $T = \{t_1, \ldots, t_n\}$ em que cada tarefa t_i tem associada um prazo d_i e um peso p_i.
- Saída: uma permutação do conjunto T, ou seja uma sequência de execução.
- Seja S uma vector de dimensão n em que cada posição pode estar vazia ou conter uma tarefa.
- Algoritmo:
  1. Iniciar o vector S com todas as posições vazias.
  2. Iniciar a estrutura de dados para conjuntos disjuntos DS, contendo um conjunto por tarefa, ou seja, realizar MAKESET(t_i) para $1 \leq i \leq n$.
  3. Para cada tarefa t em T, por ordem decrescente das penalizações p_i, onde d é o prazo de t:
     3.1. Se $S[d]$ estiver livre, realizar as afectações $S[d] \leftarrow t$ e $pos \leftarrow d$.
     Caso contrário,
        i. Afectar s com o conjunto em DS que contém a tarefa $S[d]$: $s \leftarrow$ FINDSET($S[d]$)
        ii. Afectar pos com a diferença entre o maior índice em S onde está inserido um elemento de s e o tamanho de s: $pos \leftarrow$ LARGEST(s) $-$ SIZE(s)
        iii. Se $pos > 0$, afectar $S[pos] \leftarrow t$.
        Caso contrário, a tarefa t fica em atraso e é colocada na última posição ainda livre de S. Afectar pos com a posição em que ficou t.
     3.2. Garantir que não existem dois conjuntos em DS cujas tarefas estejam contíguas em S:
        i. Se $S[pos-1]$ não está livre, então UNION(FINDSET($S[pos]$), FINDSET($S[pos-1]$)).
        ii. Se $S[pos+1]$ não está livre, então UNION(FINDSET($S[pos]$), FINDSET($S[pos+1]$)).
  4. Retornar S.

4 Data de entrega

A data limite para a entrega da resolução é 15 de Fevereiro de 2010.

ISEL, 18 de Dezembro de 2009.