



INSTITUTO SUPERIOR TÉCNICO

## Introdução aos Algoritmos e Estruturas de Dados 2017/2018 - 2º semestre

Enunciado do 2º Projecto (v1.1)  
Data de entrega: 18 de Maio de 2018 (23h59m)

### 1. Introdução

O planeamento de projectos é uma componente fundamental da gestão de projectos, sejam projectos informáticos, de engenharia civil ou outros. O planeamento de projectos consiste no escalonamento temporal das diversas tarefas por forma a evidenciar o seu paralelismo ou sequencialidade. Dado um conjunto de tarefas, as suas durações estimadas e as suas dependências, é possível estimar quando a tarefa poderá começar a ser executada. Desta forma, é possível reservar atempadamente os recursos necessários à sua realização, sejam recursos humanos ou materiais.

Uma tarefa apenas se pode iniciar assim que todas as suas *dependências* estejam concluídas. Este tempo designa-se por *early start*. Se todas as tarefas se iniciarem assim que possível a duração do projecto será mínima. Mas para que tal seja possível, pode ser necessário reservar um grande número de recursos em certos períodos de tempo. Por exemplo, se o escalonamento por *early start* obrigar, em certo momento do projecto, a desenvolver três tarefas em simultâneo, pode não haver recursos suficientes disponíveis.

A alternativa consiste em atrasar algumas tarefas até que haja recursos suficientes. Contudo, se algumas tarefas forem atrasadas, a duração total do projecto poderá aumentar. Podem no entanto existir tarefas que, por serem curtas, possam ser atrasadas sem afectar a duração total do projecto. O momento mais tarde em que uma tarefa se pode iniciar sem atrasar a duração total do projecto é designado por *late start*. Se uma tarefa tiver um *early start* igual ao *late start* diz-se *crítica*, pois qualquer atraso no seu início tem consequências imediatas na duração total do projecto. Uma sequência contendo todas as tarefas críticas de um projecto designa-se por *caminho crítico* (*critical path*).

O objectivo deste projeto é o desenvolvimento, em linguagem C, de um programa para determinar um caminho crítico de um projecto e as suas tarefas críticas. Mais especificamente, a interacção com o programa deverá ocorrer através de uma sequência de linhas compostas por uma palavra (comando), e eventualmente uma sequência de argumentos, que permitem a inserção e análise de tarefas. *O caminho crítico a obter será aquele determinado pela ordem de inserção das tarefas*. Os possíveis comandos são listados na Tabela seguinte e indicam as operações a executar.

Comando	Descrição
add	Adiciona uma tarefa.
duration	Lista as tarefas de duração igual ou superior ao indicado.
depend	Lista as tarefas que dependem da indicada.
remove	Remove uma tarefa.

path	Lista todas as tarefas no caminho crítico.
exit	Termina o programa.

## 2. Especificação do programa

Cada tarefa de um projecto é representada por um *identificador* (um inteiro positivo com pelo menos 32 bits), uma *descrição* (uma cadeia de caracteres delimitada por *aspas*), uma *duração* (um inteiro positivo com pelo menos 32 bits), e uma *lista de tarefas* das quais esta tarefa depende. A tarefa só se pode iniciar depois de todas as tarefas de que depende terem terminado.

**Nota:** Quando a execução das tarefas 9 e 12 depende da tarefa 10, dizemos que 9 e 12 são *dependentes* de 10; alternativamente a tarefa 10 *precede* as tarefas 9 e 12.

A descrição da tarefa terá, no máximo, 8000 caracteres, mas na grande maioria dos casos será muito inferior a esse valor. Não existe limite para o número de tarefas, para o número de dependências de uma tarefa, nem para o número de tarefas dependentes de uma tarefa. O limite é apenas imposto pela memória disponibilizada pelo sistema operativo para a execução do programa. Este limite será manipulado pelo ambiente de teste Mooshak.

## 3. Dados de Entrada

Durante a execução do programa as instruções devem ser lidas do terminal (standard input) na forma de uma sequência de linhas iniciadas por uma palavra, que se passa a designar por *comando*, seguido de um número variável de argumentos; o comando e cada um dos argumentos são separados por um espaço.

Os comandos disponíveis são descritos de seguida e correspondem às palavras `add`, `duration`, `depend`, `remove`, `path`, `exit`. Se os argumentos não forem válidos deverá ser impressa a mensagem: `illegal arguments`.<sup>1</sup> Cada comando indica uma determinada acção que se passa a caracterizar em termos de objectivo, número de argumentos, sintaxe e output:

add: adiciona uma nova tarefa ao projecto (nargs >= 3)

add id descrição duração ids

- id: número inteiro positivo com pelo menos 32 bits;

- descrição: cadeia de caracteres com um máximo de 8000 caracteres iniciada e terminada pelo carácter *aspas*;

- duração: número inteiro positivo com pelo menos 32 bits;

- ids: sequência, possivelmente nula, de identificadores das dependências desta tarefa. Estes identificadores são separados por um espaço e deverão corresponder a tarefas previamente adicionadas.

*Output:* Não tem qualquer output.

---

<sup>1</sup> Note que neste projecto, contrariamente ao primeiro projecto, deverá efectuar validação dos parâmetros.

Adiciona a tarefa ao projecto, invalidando o caminho crítico anteriormente calculado.

No caso de introdução de uma tarefa com um identificador já existente deverá ser impressa a mensagem de erro

```
id already exists
```

No caso de introdução de uma tarefa que tenha uma dependência que não existe deverá ser impressa a mensagem de erro

```
no such task
```

duration: imprime as tarefas com duração igual ou superior ao indicado (nargs <= 1)

duration

duration value

- value: número inteiro positivo com pelo menos 32 bits.

*Output:* imprime as tarefas de duração igual ou superior a value pela ordem em que foram introduzidas. Se o argumento value for omitido, deverão ser impressas todas as tarefas (pela ordem em que foram introduzidas).

Cada tarefa deve ser impressa com o mesmo formato do comando path. Caso o comando path ainda não tenha sido executado, ou o caminho crítico esteja inválido, a sequência [<early> <late>] referida abaixo deverá ser omitida.

depend: lista as tarefas dependentes da tarefa indicada (nargs = 1)

depend id

- id: número inteiro positivo.

*Output:* deve ser impresso o id da tarefa seguido dos id<sub>i</sub> das tarefas dependentes dela separados por um espaço

```
<id>: <id1> <id2> <id3>
```

Caso a tarefa não tenha tarefas dependentes dela deverá ser impressa a mensagem

```
<id>: no dependencies
```

Caso não exista uma tarefa com o identificador id deverá ser impressa a mensagem

```
no such task
```

No caso de existirem 2 ou mais dependências, deverão ser listadas pela ordem em que foram inseridas.

remove: remove uma tarefa sem tarefas dependentes (nargs = 1)

remove id

- id: número inteiro positivo.

*Output:* Não tem qualquer output.

Remove uma tarefa sem dependentes, tornando o caminho crítico calculado anteriormente inválido.

Caso a tarefa com o identificador id tenha dependentes deverá ser impressa a mensagem

task with dependencies

Caso não exista uma tarefa com o identificador id deverá ser impressa a mensagem

no such task

path: lista todas as tarefas do caminho crítico (nargs = 0)

path

*Output:* lista todas as tarefas do caminho crítico do projecto pela ordem de introdução e com a forma

<id> <descrição> <duração> [<early> <late>] <ids>

onde <id>, <descrição>, <duração> e <ids> são os valores introduzidos aquando da criação da tarefa; <early> e <late> correspondem respectivamente ao *early start* e *late start* da tarefa; caso o *early start* seja igual ao *late start*, então <late> deverá ser substituído por CRITICAL)

No fim da listagem das tarefas deverá ser impressa a duração total do projecto:

project duration = <value>

A *determinação do caminho crítico* deve analisar todas as tarefas finais, ie, aquelas de que nenhuma outra tarefa depende.

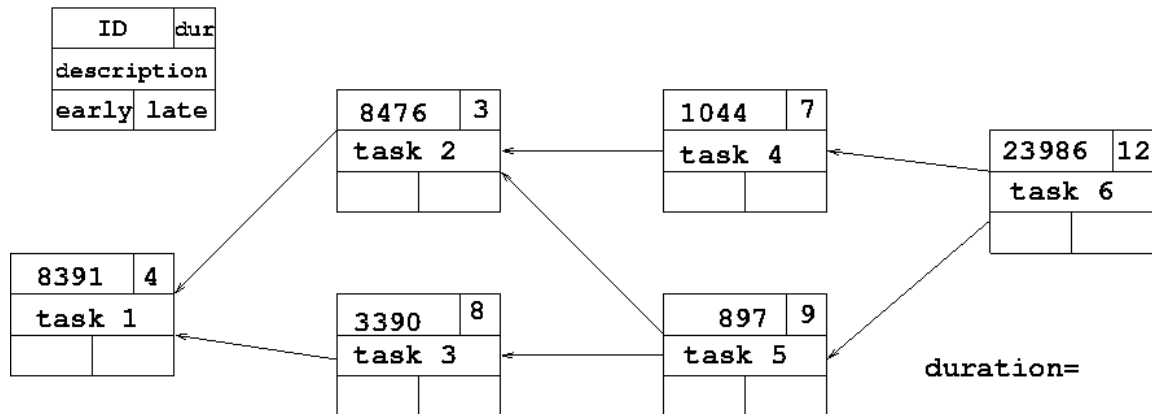
Para cada tarefa final deve determinar os *early start* percorrendo todas as suas dependências, e recursivamente as dependências delas, calculando, do início para o fim, o tempo que resulta da sua dependência que termina mais tarde (maior *early finish*). O *early start* de uma tarefa origem é 0.

A *duração do projecto* é o maior valor entre os *early finish* das tarefas finais, onde o *early finish* é a soma da *duração* da tarefa com o seu *early start*.

Para a determinação dos *late start*, deve subtrair a duração do projecto às tarefas finais e proceder como anteriormente com as dependências, subtraindo as durações dessas tarefas.

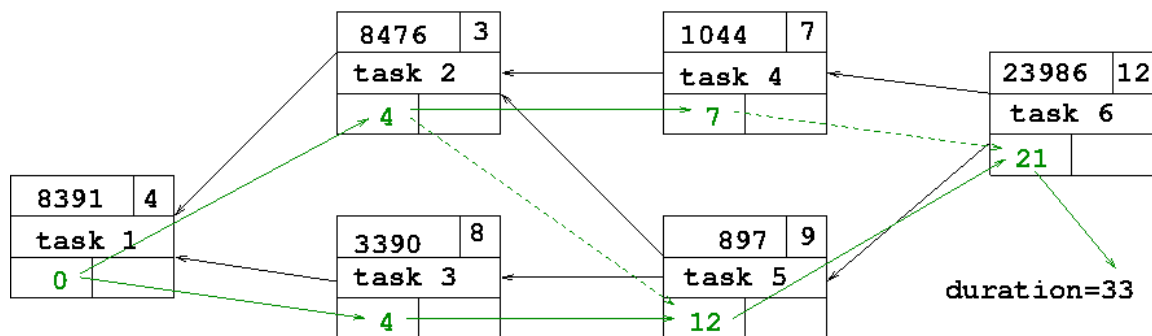
O caminho crítico corresponde às tarefas onde o *early start* é igual ao *late start*.

Exemplo: Considere as 6 tarefas abaixo, onde uma seta de *a* para *b* significa que a tarefa *a* é dependente da tarefa *b*.

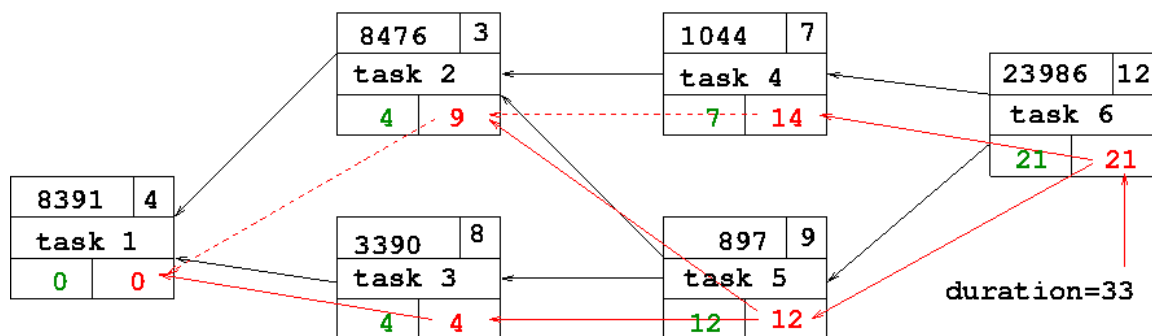


O **cálculo do *early start*** começa nas *tarefas origem* (sem precedentes, task 1) e caminha na direcção das *tarefas finais* (sem dependentes, task 6). O *early start* de uma tarefa é o maior valor ( $\text{duração}_i + \text{early-start}_i$ ) entre todas as dependências *i* da tarefa.

Notar que os ponteiros a preto dirigem-se das  *finais* para as *origens*.



O **cálculo do *late start*** começa nas tarefas  *finais* e caminha para as tarefas  *origem* subtraindo a sua duração ao menor tempo dos seus sucessores.



exit: abandona a aplicação (nargs = 0)  
exit

*Output*: Não imprime output.

## 4. Dados de Saída

O programa deverá escrever no standard output as respostas a certos comandos apresentados no standard input. As respostas são igualmente linhas de texto formatadas conforme definido anteriormente neste enunciado. Tenha em atenção o número de espaços entre elementos do seu output, assim como os espaços no final de cada linha. Procure respeitar escrupulosamente as indicações dadas.

## 5. Exemplos (Input/Output)

### Exemplo 1 (./proj2)

#### Dados de Entrada:

```
add 8391 "task 1" 4
add 8476 "task 2" 3 8391
add 3390 "task 3" 8 8391
add 1044 "task 4" 7 8476
add 897 "task 5" 9 8476 3390
add 23986 "task 6" 12 1044 897
duration
exit
```

#### Dados de Saída:

```
8391 "task 1" 4
8476 "task 2" 3 8391
3390 "task 3" 8 8391
1044 "task 4" 7 8476
897 "task 5" 9 8476 3390
23986 "task 6" 12 1044 897
```

### Exemplo 2 (./proj2)

#### Dados de Entrada:

```
add 8391 "task 1" 4
add 8476 "task 2" 3 8391
add 3390 "task 3" 8 8391
add 1044 "task 4" 7 8476
add 897 "task 5" 9 8476 3390
add 23986 "task 6" 12 1044 897
duration 8
depend 8476
exit
```

#### Dados de Saída:

```
3390 "task 3" 8 8391
897 "task 5" 9 8476 3390
23986 "task 6" 12 1044 897
8476: 1044 897
```

## Exemplo 3 (./proj2)

### Dados de Entrada:

```
duration
add 8391 "task 1" 4
add 8476 "task 2" 3 8391
add 3390 "task 3" 8 8391
add 1044 "task 4" 7 8476
add 897 "task 5" 9 8476 3390
add 23986 "task 6" 12 1044 897
remove 23986
remove 1044
duration
exit
```

### Dados de Saída:

```
8391 "task 1" 4
8476 "task 2" 3 8391
3390 "task 3" 8 8391
897 "task 5" 9 8476 3390
```

## Exemplo 4 (./proj2)

### Dados de Entrada:

```
add 8391 "task 1" 4
add 8476 "task 2" 3 8391
add 3390 "task 3" 8 8391
add 1044 "task 4" 7 8476
add 897 "task 5" 9 8476 3390
add 23986 "task 6" 12 1044 897
path
duration
exit
```

### Dados de Saída:

```
8391 "task 1" 4 [0 CRITICAL]
3390 "task 3" 8 [4 CRITICAL] 8391
897 "task 5" 9 [12 CRITICAL] 8476 3390
23986 "task 6" 12 [21 CRITICAL] 1044 897
project duration = 33
8391 "task 1" 4 [0 CRITICAL]
8476 "task 2" 3 [4 9] 8391
3390 "task 3" 8 [4 CRITICAL] 8391
1044 "task 4" 7 [7 14] 8476
897 "task 5" 9 [12 CRITICAL] 8476 3390
23986 "task 6" 12 [21 CRITICAL] 1044 897
```

## Exemplo 5 (./proj2)

### Dados de Entrada:

```
add 5 "task15" 5
add 4 "task14" 4 5
add 3 "task13" 3 4 5
add 2 "task12" 2 3 4 5
add 1 "task11" 1 2 4
depend 1
depend 2
depend 5
path
exit
```

### Dados de Saída:

```
1: no dependencies
2: 1
5: 4 3 2
5 "task15" 5 [0 CRITICAL]
4 "task14" 4 [5 CRITICAL] 5
3 "task13" 3 [9 CRITICAL] 4 5
2 "task12" 2 [12 CRITICAL] 3 4 5
1 "task11" 1 [14 CRITICAL] 2 4
project duration = 15
```

## 6. Compilação do Programa

O compilador a utilizar é o `gcc` com as seguintes opções de compilação: `-Wall -ansi -pedantic`. Para compilar o programa deverá executar o seguinte comando:

```
$ gcc -Wall -ansi -pedantic -o proj2 *.c
```

o qual deve ter como resultado a geração do ficheiro executável `proj2`, caso não haja erros de compilação. A execução deste comando não deverá escrever qualquer resultado no terminal. Caso a execução deste comando escreva algum resultado no terminal, considera-se que o programa não compilou com sucesso. Por exemplo, durante a compilação do programa, o compilador não deve escrever mensagens de aviso (warnings).

## 7. Execução do Programa

O programa deve ser executado da forma seguinte:

```
$ ./proj2 < test01.in > test01.myout
```

Posteriormente poderá comparar o seu output com o output previsto usando o comando `diff`

```
$ diff test01.out test01.myout
```

### 7.1. Testes Auxiliares

Para testar o seu programa poderá executar os passos indicados acima ou usar os scripts `run.sh` e `run_all.sh` distribuídos no ficheiro `exemplos-p2.zip`.

Se quiserem executar apenas o `test01.in` deverão executar

```
$ ./run.sh <vosso_ficheiro_c> test01.in
```

Para executarem todos os testes deverão executar

```
$ ./run_all.sh <vosso_ficheiro_c>
```

Estes scripts compilam o ficheiro indicado e comparam o resultado obtido com o resultado esperado. Se apenas indicar o tempo de execução é porque o comando `diff` não encontrou nenhuma diferença. Caso indique mais informação, então é porque o resultado obtido e o resultado esperado diferem. Para obter a informação detalhada das diferenças poderá remover a opção `-q` da linha 11 do ficheiro `run.sh`.



## 8. Entrega do Projecto

A entrega do projecto deverá respeitar o procedimento seguinte:

- Na página da disciplina aceda ao sistema para entrega de projectos. O sistema será activado uma semana antes da data limite de entrega. Instruções acerca da forma de acesso ao sistema serão oportunamente fornecidas.
- Efectue o upload de um ficheiro arquivo com extensão `.zip` que inclua todos os ficheiros fonte que constituem o programa. Se o seu código tiver apenas um ficheiro o zip conterá apenas esse ficheiro. Se o seu código estiver estruturado em vários ficheiros (`.c` e `.h`) não se esqueça de os juntar também ao pacote.
- Para criar um ficheiro arquivo com a extensão `.zip` deve executar o seguinte comando na directoria onde se encontram os ficheiros com extensão `.c` e `.h` (se for o caso), criados durante o desenvolvimento do projecto:  

```
$ zip proj2.zip *.c *.h
```
- Como resultado do processo de upload será informado se a resolução entregue apresenta a resposta esperada num conjunto de casos de teste.
- O sistema **não permite submissões com menos de 10 minutos de intervalo** para o mesmo utilizador. **Tenha especial atenção a este facto na altura da submissão final.** Exemplos de casos de teste serão oportunamente fornecidos.
- Data limite de entrega do projecto: **18 de Maio de 2018 (23h59m)**. Até à data limite poderá efectuar o número de submissões que desejar, sendo utilizada para efeitos de avaliação a última submissão efectuada. Deverá portanto verificar cuidadosamente que a última submissão corresponde à versão do projecto que pretende que seja avaliada. Não existirão excepções a esta regra.

## 9. Avaliação do Projecto

### a. Componentes da Avaliação

Na avaliação do projecto serão consideradas as seguintes componentes:

1. A primeira componente avalia o desempenho da funcionalidade do programa realizado. Esta componente é avaliada entre 0 e 16 valores.
2. A segunda componente avalia a qualidade do código entregue, nomeadamente os seguintes aspectos: comentários, indentação, estruturação, modularidade e divisão em ficheiros, abstracção de dados, entre outros. Esta componente poderá variar entre -4 valores e +4 valores relativamente à classificação calculada no item anterior. Nesta componente será também utilizado o sistema `valgrind` de forma a detectar fugas de memória (“memory leaks”) ou outras incorrecções no código, que serão penalizadas. Aconselha-se por isso que os alunos utilizem este sistema para fazer

debugging do código e corrigir eventuais incorrecções, antes da submissão do projecto. Para utilizar o valgrind,<sup>2</sup> comece por compilar o seu código com o gcc mas com a flag adicional -g:

```
$ gcc -g -Wall -ansi -pedantic -o proj2 *.c
```

Esta flag adiciona informação para debugging que será posteriormente utilizada pelo valgrind. Depois disso, basta escrever

```
$ valgrind --tool=memcheck --leak-check=yes ./proj2 < test01.in
```

Neste exemplo, estaria a usar o test01.in como input. Deverá estar particularmente atento/a a mensagens como

```
Invalid read Invalid write
```

que indicam está a tentar ler ou escrever fora da área de memória reservada por si. O valgrind também detecta a utilização de variáveis não inicializadas dentro expressões condicionais. Nesse caso receberá a mensagem

```
Conditional jump or move depends on uninitialised value(s).
```

Por fim, o valgrind oferece a preciosa informação sobre a quantidade de memória alocada e libertada na heap, indicando quando essas duas quantidades não são iguais. Nesse caso terá memory leaks. Aqui fica um exemplo de output do valgrind quando tudo corre bem:

```
HEAP SUMMARY:
in use at exit: 0 bytes in 0 blocks
total heap usage: 1,024,038 allocs, 1,024,038 frees, 28,682,096 bytes allocated
All heap blocks were freed -- no leaks
```

## **b. Atribuição Automática da Classificação**

- o A classificação da primeira componente da avaliação do projecto é obtida através da execução *automática* de um conjunto de testes num computador com o sistema operativo GNU/Linux. Torna-se portanto essencial que o código compile correctamente e que respeite o formato de entrada e saída dos dados descrito anteriormente. Projectos que não obedeçam ao formato indicado no enunciado serão penalizados na avaliação automática, podendo, no limite, ter 0 (zero) valores se falharem todos os testes. Os testes considerados para efeitos de avaliação poderão incluir (ou não) os disponibilizados na página da disciplina, além de um conjunto de testes adicionais. A execução de cada programa em cada teste é limitada na quantidade de memória que pode utilizar, até um máximo de 64 Mb, e no tempo total disponível para execução, sendo o tempo limite distinto para cada teste.

---

<sup>2</sup> <http://www.cprogramming.com/debugging/valgrind.html>

- Note-se que o facto de um projecto passar com sucesso o conjunto de testes disponibilizado na página da disciplina não implica que esse projecto esteja totalmente correcto. Apenas indica que passou alguns testes com sucesso, mas este conjunto de testes não é exaustivo. É da responsabilidade dos alunos garantir que o código produzido está correcto.
- Em caso algum será disponibilizado qualquer tipo de informação sobre os casos de teste utilizados pelo sistema de avaliação automática. A totalidade de ficheiros de teste usados na avaliação do projecto serão disponibilizados na página da disciplina após a data de entrega.

## **LOG**

**04May2018** – Clarifica a ordem de apresentação nos comandos `depend` e `path`.