Instituto Superior Técnico — LMAC/MEBiom

V1

Elementos de Programação

Novembro 2020 Mini-teste 2 Duração: 30m

Nesta avaliação não pode usar definições por compreensão nem métodos. As únicas operações sobre listas permitidas são: lista vazia ([]), acesso aos elementos da lista por posição (lista[posição]), seccionamento da lista (lista[posição:posição]), comparação com a lista vazia (==[]), cálculo do comprimento (`len`) e concatenação (+).

Existe um número arbitrário de **servidores**, identificados sequencialmente $(0, 1, 2, \dots)$, colocados inicialmente no plano nos seus pontos base. Esta informação (número de servidores e suas posições iniciais) é dada por uma lista como, por exemplo, `base=[(0,0),(10,10),(5,5)]`, que no caso identifica três servidores (o servidor 0 inicialmente colocado em $(0, 0)$, o servidor 1 inicialmente colocado em $(10, 10)$, e o servidor 2 inicialmente colocado em $(5, 5)$).

Há também uma lista de **pedidos** (pontos no plano), que devem ser atendidos sequencialmente. Por exemplo, `reqs=[(1,1),(10,9),(4,5),(10,0)]`, sendo que cada pedido deve ser atendido deslocando para o ponto pedido um dos servidores.

Finalmente, há uma lista de **alocação** dos servidores, que determina qual o servidor que deve atender cada pedido. Por exemplo, `alcs=[0,1,2,1]`.

Neste caso, ao atender o primeiro pedido, o servidor 0 desloca-se da sua base $(0, 0)$ para o ponto $(1, 1)$. Depois, ao atender o segundo pedido, o servidor 1 desloca-se da sua base $(10, 10)$ para o ponto $(10, 9)$. De seguida, ao atender o terceiro pedido, o servidor 2 desloca-se da sua base $(5, 5)$ para o ponto $(4, 5)$. Finalmente, ao atender o quarto e último pedido, o servidor 1 desloca-se da sua posição actual $(10, 9)$ para o ponto $(10, 0)$.

Considera-se a habitual distância Euclideana no plano, ou seja, $\text{dist}((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, que pode ser facilmente calculada em *Pyhton* pela seguinte definição, que pode usar livremente.

```
from math import sqrt
def dist(p,q):
    return sqrt((p[0]-q[0])**2+(p[1]-q[1])**2)
```

Deste modo o servidor 0 percorre uma distância $\text{dist}((0, 0), (1, 1)) = \sqrt{2}$, o servidor 1 percorre uma distância total $\text{dist}((10, 10), (10, 9)) + \text{dist}((10, 9), (10, 0)) = 1 + 9 = 10$, e o servidor 2 percorre uma distância $\text{dist}((5, 5), (4, 5)) = 1$. Pretende-se que os vários servidores percorram distâncias o mais aproximadas possível, pelo que o custo final das alocações é a diferença máxima entre as distâncias percorridas por eles, neste caso $10 - 1 = 9$.

(**a**) Defina imperativamente em *Python* uma função `maxmin` que dada uma lista não vazia de números calcula o par de valores correspondente ao máximo e ao mínimo da lista.

Por exemplo, `maxmin([1.41,10,1])` deverá ser `(10,1)`.

(**b**) Defina imperativamente em *Python* uma função `cost` que dada uma lista base dos servidores, uma lista de pedidos, e uma lista de alocações, calcula o custo total das alocações face à situação inicial dada pela base e à lista de pedidos.

Por exemplo, `cost(base,reqs,alcs)` deverá ser `9`.

(**c**) Defina imperativamente em *Python* uma função Booleana `anybetter` que dada uma lista base dos servidores, uma lista de listas de pedidos, uma lista de listas de alocações, e um valor alvo, devolve `True` se existe nas listas dadas uma lista de pedidos e uma lista de alocações cujo custo total a partir da base dos servidores seja inferior ao valor alvo, e devolve `False` caso contrário.

Por exemplo, `anybetter(base,[reqs,[(1,1),(10,9),(4,5),(10,10)]],[alcs,[1,1,1,1]],9)` deverá ser `True` (pois para os pedidos `[(1,1),(10,9),(4,5),(10,10)]` e as alocações `alcs` tem-se `cost(base,[(1,1),(10,9),(4,5),(10,10)],alcs)` igual a 1).

Instituto Superior Técnico — LMAC/MEBiom

V2

Elementos de Programação

Novembro 2020 Mini-teste 2 Duração: 30m

Nesta avaliação não pode usar definições por compreensão nem métodos. As únicas operações sobre listas permitidas são: lista vazia ([]), acesso aos elementos da lista por posição (lista[posição]), seccionamento da lista (lista[posição:posição]), comparação com a lista vazia (==[]), cálculo do comprimento (`len`) e concatenação (+).

Existe um número arbitrário de **servidores**, identificados sequencialmente $(0, 1, 2, \dots)$, colocados inicialmente numa linha nos seus pontos base. Esta informação (número de servidores e suas posições iniciais) é dada por uma lista como, por exemplo, `base=[0,-5,10]`, que no caso identifica três servidores (o servidor 0 inicialmente colocado em 0, o servidor 1 inicialmente colocado em -5, e o servidor 2 inicialmente colocado em 10).

Há também uma lista de **pedidos** (pontos na linha), que devem ser atendidos sequencialmente. Por exemplo, `reqs=[-7,0,3,20]`, sendo que cada pedido deve ser atendido deslocando para o ponto pedido, ao longo da linha, um dos servidores.

Finalmente, há uma lista de **alocação** dos servidores, que determina qual o servidor que deve atender cada pedido. Por exemplo, `alcs=[1,0,2,2]`.

Neste caso, ao atender o primeiro pedido, o servidor 1 desloca-se da sua base -5 para o ponto -7. Depois, ao atender o segundo pedido, o servidor 0 desloca-se da sua base 0 para o ponto 0 (ou seja, não necessita de se mover). De seguida, ao atender o terceiro pedido, o servidor 2 desloca-se da sua base 10 para o ponto 3. Finalmente, ao atender o quarto e último pedido, o servidor 2 desloca-se da sua posição actual 3 para o ponto 20.

Considera-se a habitual distância na linha, ou seja, $\text{dist}(x, y) = |x - y|$, que pode ser facilmente calculada em *Pyhton* pela seguinte definição, que pode usar livremente.

```
def dist(p,q):
    return abs(p-q)
```

Deste modo o servidor 0 percorre uma distância $\text{dist}(0, 0) = 0$, o servidor 1 percorre uma distância total $\text{dist}(-5, -7) = 2$, e o servidor 2 percorre uma distância $\text{dist}(10, 3)+\text{dist}(3, 20) = 7 + 17 = 24$. Pretende-se calcular o custo total das alocações, que é a soma das distâncias totais percorridas pelos vários servidores. No exemplo, tem-se custo $0 + 2 + 24 = 26$.

(**a**) Defina imperativamente em *Python* uma função `calcdist` que dada uma lista base dos servidores, uma lista de pedidos, e uma lista de alocações, calcula o custo total.

Por exemplo, `calcdist(base,reqs,alcs)` deverá ser `26`.

(**b**) Defina imperativamente em *Python* uma função `matrixdist` que dado o número de servidores a considerar, uma lista de listas de pedidos, e uma lista de listas de alocações, calcula a matriz que tem em cada entrada (i,j) o valor do custo total relativo ao pedido na posição i e alocações na posição j das respectivas listas, sempre em relação à lista base dos servidores em que todos estão colocados no ponto 0.

Por exemplo, `matrixdist(3,[reqs,[0,1,1,1]],[alcs,[0,0,0,0],alcs]))` deverá ser `[[27, 34, 27], [2, 1, 2]]`.

(**c**) Defina imperativamente em *Python* uma função `posmin` que dada uma matriz calcula (um)a posição na matriz do seu valor mínimo; e use-a para definir uma função `bestcomb`, que dado o número de servidores a considerar, uma lista de listas de pedidos, e uma lista de listas de alocações, calcula o par (i,j) a que corresponde (um)a combinação, pedidos na posição i e alocações na posição j, a que corresponda um menor custo total relativamente à lista base dos servidores em que todos estão colocados no ponto 0.

Por exemplo, `posmin([[27, 34, 27], [2, 1, 2]])` deverá ser `(1,1)`, e portanto também `bestcomb(3,[reqs,[0,1,1,1]],[alcs,[0,0,0,0],alcs]))` deverá ser `(1,1)`.

V3

Instituto Superior Técnico — LMAC/MEBiom

Elementos de Programação

Novembro 2020 Mini-teste 2 Duração: 30m

Nesta avaliação não pode usar definições por compreensão nem métodos. As únicas operações sobre listas permitidas são: lista vazia ([]), acesso aos elementos da lista por posição (lista[posição]), seccionamento da lista (lista[posição:posição]), comparação com a lista vazia (==[]), cálculo do comprimento (`len`) e concatenação (+).

Consideramos tarefas da forma `(d,p)` onde `d` é um inteiro que indica o prazo máximo em que a tarefa deve ser executada, e `p` é a penalização a pagar se a tarefa não for executada dentro do prazo.

Uma *calendarização* é uma lista de tarefas que estabelece o instante em que cada tarefa é executada. Por exemplo, com a calendarização

```
calend=[(1,30),(2,10),(1,100),(3,5)]
```

está-se a estipular que a tarefa `(1,30)` é executada no instante 0 (o que cumpre o prazo e não paga penalidade), depois a tarefa `(2,10)` é executada no instante 1 (o que também cumpre o prazo e não paga penalidade), de seguida a tarefa `(1,100)` é executada no instante 2 (o que não cumpre o prazo e paga 100 de penalização), e finalmente a tarefa `(3,5)` é executada no instante 3 (o que cumpre o prazo e não paga penalização).

A penalização total associada à calendarização é portanto 0+0+100+0=100.

(**a**) Defina imperativamente em *Python* uma função `penal` que dada uma lista de calendarização de tarefas calcula o par de valores correspondentes à penalização total associada e à posição na lista de um(a) tarefa que paga a penalização mais alta (a posição deve ser -1 se nenhuma tarefa for penalizada).

Por exemplo, `penal(calend)` deverá ser `(100,2)`, e `penal([(1,100),(1,30),(2,10),(3,5)])` deverá ser `(0,-1)`.

(**b**) Defina imperativamente em *Python* uma função `melhor` que dada uma lista de listas de calendarização, calcula (um)a calendarização na lista com menor penalização total.

Por exemplo, `melhor([calend,[(1,100),(1,30),(2,10),(3,5)])` deverá ser a calendarização `[(1,100),(1,30),(2,10),(3,5)]`.

(**c**) Defina imperativamente em *Python* uma função `recalendar` que dada uma lista de calendarização, e uma posição nessa lista, calcula todas as listas de calendarização que resultam da lista dada atrasando ou adiantando a tarefa na posição dada de todas as maneiras possíveis mas mantendo a ordem relativa das restantes tarefas; e use-a para definir uma função Booleana `recalhighpen` que dada uma lista de calendarização verifica se a calendarização dada pode ser melhorada recalendarizando a sua tarefa mais penalizadora.

Por exemplo, `recalendar(calend,2)` resulta na lista `[[(1,100),(1,30),(2,10),(3,5)], [(1,30),(1,100),(2,10),(3,5)], [(1,30),(2,10),(3,5),(1,100)]]`, e portanto o resultado de `recalhighpen(calend)` deverá ser `True`.

Instituto Superior Técnico — LMAC/MEBiom

Elementos de Programação

V4

Novembro 2020 Mini-teste 2 Duração: 30m

Nesta avaliação não pode usar definições por compreensão nem métodos. As únicas operações sobre listas permitidas são: lista vazia ([]), acesso aos elementos da lista por posição (lista[posição]), seccionamento da lista (lista[posição:posição]), comparação com a lista vazia (==[]), cálculo do comprimento (`len`) e concatenação (+).

Consideramos tarefas da forma `d` onde `d` é um inteiro que indica o prazo máximo em que a tarefa deve ser executada, considerando-se uma falha caso a tarefa seja executada fora do prazo.

Uma *calendarização* é uma lista de tarefas que estabelece o instante em que cada tarefa é executada. Por exemplo, com a calendarização

```
calend=[1,2,1,0]
```

está a estipular-se que a tarefa `1` é executada no instante 0 (o que cumpre o prazo), depois a tarefa `2` é executada no instante 1 (o que também cumpre o prazo), de seguida a tarefa `1` é executada no instante 2 (o que não cumpre o prazo e corresponde a uma falha, pois o prazo é falhado por 1 instante), e finalmente a tarefa `0` é executada no instante 3 (o que não cumpre o prazo e corresponde a outra falha, pois o prazo é falhado por 3 instantes).

O número total de falhas associado à calendarização é portanto 2.

(**a**) Defina imperativamente em *Python* uma função `falhas` que dada uma lista de calendarização de tarefas calcula o par de valores correspondente ao número total de falhas associado, e à posição na lista de calendarização de (um)a das tarefas falhadas cujo prazo é falhado mais largamente (a posição deve ser -1 se não houver falhas).

Por exemplo, `falhas(calend)` deverá ser `(2,3)`, e `falhas([0,2,5,3])` deverá ser `(0,-1)`.

(**b**) Defina imperativamente em *Python* uma função `menosfalhas` que dada uma lista de listas de calendarização, calcula (um)a posição nessa lista da calendarização com número mínimo de falhas.

Por exemplo, `menosfalhas([calend,[0,2,5,3]])` deverá ser `1`.

(**c**) Defina imperativamente em *Python* uma função `antecip` que dada uma lista de calendarização, e uma posição nessa lista, calcula todas as listas de calendarização que resultam da lista dada antecipando a tarefa na posição dada de todas as maneiras possíveis mas mantendo a ordem relativa das restantes tarefas; e use-a para definir uma função `antpior` que dada uma lista de calendarização verifica se a calendarização dada pode ser melhorada antecipando a sua tarefa cujo prazo é falhado mais largamente, devolvendo `False` se não for o caso, e se for o caso devolvendo o par `(alt,f)` onde `alt` é a melhor calendarização alternativa obtida por antecipação e `f` o seu número de falhas.

Por exemplo, `antecip(calend,3)` resulta na lista `[[0,1,2,1],[1,0,2,1], [1,2,0,1]]`, e portanto o resultado de `antpior(calend)` deverá ser `([0,1,2,1],1)`.

Instituto Superior Técnico — LMAC/MEBiom **V5**

Elementos de Programação

Novembro 2020 Mini-teste 2 Duração: 30m

Nesta avaliação não pode usar definições por compreensão nem métodos. As únicas operações sobre listas permitidas são: lista vazia ([]), acesso aos elementos da lista por posição (lista[posição]), seccionamento da lista (lista[posição:posição]), comparação com a lista vazia (==[]), cálculo do comprimento (`len`) e concatenação (+).

Consideramos configurações de memória dadas por listas de valores (sem repetições). Por exemplo,

`mem=[23,5,14,77]`.

O custo de aceder a cada valor numa dada configuração da memória é a posição que ocupa. Assim, no exemplo, aceder ao valor 23 tem custo 0, aceder ao valor 5 tem custo 1, aceder ao valor 14 tem custo 2, e aceder ao valor 77 tem custo 3.

Uma lista de *pedidos* é uma lista (com possíveis repetições) de valores em memória, que devem ser acedidos. O custo total de aceder aos pedidos é a soma dos custos associados ao acesso a cada um dos valores. Assim, por exemplo, para a lista de pedidos

`req=[5,14,5,5,77,23]`.

tem-se um custo total igual a 1+2+1+1+3+0=8.

(**a**) Defina imperativamente em *Python* uma função `custo` que dada uma configuração de memória, e uma lista de pedidos, calcula o seu custo total.

Por exemplo, `custo(mem,req)` deverá ser `8`.

(**b**) Defina imperativamente em *Python* uma função `maisvisitada` que dada uma configuração de memória, e uma lista de pedidos, calcula (um)a posição da célula de memória mais visitada para satisfazer os pedidos.

Por exemplo, `maisvisitada(mem,req)` deverá ser `1` (a posição da memória correspondente ao valor `5`, que é o que mais vezes ocorre na lista de pedidos).

(**c**) Defina imperativamente em *Python* uma função `melhorcomb` que dada uma lista de configurações de memória, e uma lista de listas de pedidos, calcula o par `(i,j)` a que corresponde (um)a combinação da configuração de memória na posição `i` com a lista de pedidos na posição `j`, nas respectivas listas, que minimiza o custo.

Por exemplo, `melhorcomb([mem,[5,14,77,23],[77,23,14,5]],[req,[23,23,23]])` deverá ser `(0,1)` (correspondente à configuração de memória `mem` e à lista de pedidos `[23,23,23]`, cujo custo associado é 0).

Instituto Superior Técnico — LMAC/MEBiom **V6**

Elementos de Programação

Novembro 2020 Mini-teste 2 Duração: 30m

Nesta avaliação não pode usar definições por compreensão nem métodos. As únicas operações sobre listas permitidas são: lista vazia ([]), acesso aos elementos da lista por posição (lista[posição]), seccionamento da lista (lista[posição:posição]), comparação com a lista vazia (==[]), cálculo do comprimento (`len`) e concatenação (+).

Consideramos configurações de memória dadas por listas de valores (sem repetições). Por exemplo,

`mem=[16,8,11,20]`.

O custo de aceder a cada valor numa dada configuração da memória é a posição que ocupa. Assim, no exemplo, aceder ao valor 16 tem custo 0, aceder ao valor 8 tem custo 1, aceder ao valor 11 tem custo 2, e aceder ao valor 20 tem custo 3.

Uma lista de *pedidos* é uma lista (com possíveis repetições) de valores em memória, que devem ser acedidos. Por exemplo,

`req=[11,8,11,11,20,16]`.

Dada uma configuração de memória, associa-se a cada lista de pedidos a lista dos custos correspondentes aos acessos. No caso, ter-se-ia

`custos=[2,1,2,2,3,0]`.

O custo total dos acessos é a soma dos valores na lista de custos, neste caso 2+1+2+2+3+0=10.

(**a**) Defina imperativamente em *Python* uma função `custotal` que dada uma lista de custos, calcula o seu custo total.

Por exemplo, `custotal(custos)` deverá ser `10`.

(**b**) Defina imperativamente em *Python* uma função `melhores` que dada uma lista de listas de custos, calcula o par formado pela lista das listas de custos com custo total dos acessos mínimo (de entre as que são dadas), e pelo respectivo custo.

Por exemplo, `melhores([custos,[25],[2,2,2,2,2]])` deverá ser `([custos,[2,2,2,2,2]],10)`.

(**c**) Defina imperativamente em *Python* uma função `troca` que dada uma lista e dois valores distintos `x,y` calcula a lista que resulta de trocar as ocorrências de `x` por `y` e as ocorrências de `y` por `x`; e use-a para definir uma função Booleana `melhoresportroca` que dada uma lista de custos, e o seu custo máximo, calcula a lista das listas de custos obtidas por troca de dois valores a partir da lista de custos dada que têm menor custo.

Por exemplo, `troca(custos,1,3)` resulta na lista `[2,3,2,2,1,0]`, e `melhoresportroca(custos)` resulta na lista `[[0,1,0,0,3,2]]` (que resulta da troca de `0` com `2`, que é a única troca cujo resultado tem custo `6`, tendo todas as outras possibilidades custos superiores).

V7

Instituto Superior Técnico — LMAC/MEBiom

Elementos de Programação

Novembro 2020 Mini-teste 2 Duração: 30m

Nesta avaliação não pode usar definições por compreensão nem métodos. As únicas operações sobre listas permitidas são: lista vazia ([]), acesso aos elementos da lista por posição (lista[posição]), seccionamento da lista (lista[posição:posição]), comparação com a lista vazia (==[]), cálculo do comprimento (`len`) e concatenação (+).

Consideram-se listas de valores numéricos. Em muitas situações é importante que trabalhemos com listas ordenadas (por ordem crescente, em sentido lato). Quando numa dada lista o valor numa certa posição é menor que o valor anterior dizemos que se trata de uma inversão.

Por exemplo, a lista `w=[1,2,3,4,3,5,5,0]` tem 2 inversões (4>3 e 5>0).

Obviamente, uma lista ordenada não tem inversões. Por essa razão dizemos que uma lista está melhor ordenada que outra se tiver um menor número de inversões ou, caso tenham o mesmo número de inversões, se a primeira inversão for mais tardia (o que significa que tem um segmento inicial ordenado mais comprido).

(**a**) Defina imperativamente em *Python* uma função `invs` que dada uma lista de números, calcula o par formado pelo seu número de inversões e pela posição em que ocorre a primeira inversão (ou o comprimento da lista caso esta esteja ordenada).

Por exemplo, `invs(w)` deverá ser `(2,4)`.

(**b**) Defina imperativamente em *Python* uma função `melhorord` que dada uma lista de listas, calcula (um)a posição da lista (de entre as listas dadas) que está melhor ordenada.

Por exemplo, `melhorord([w,[1,2,3,3,4,5,5,0]])` deverá ser `1` (correspondendo à lista `[1,2,3,3,4,5,5,0]`, pois `invs([1,2,3,3,4,5,5,0])` é `(1,7)`).

(**c**) Defina imperativamente em *Python* uma função `todastrocas` que dada uma lista calcula a lista de todas as listas que resultam de trocar os elementos em quaisquer duas posições da lista; e use-a para definir uma função Booleana `valetrocar` que dada uma lista de valores determina se por uma troca de valores entre duas posições da lista seria possível melhorar a ordenação da lista dada.

Por exemplo, `todastrocas([1,2,3]))` resulta na lista `[[2,1,3],[3,2,1],[1,3,2]]`, e `valetrocar(w)` deverá ser `True` (pois a lista `[1,2,3,3,4,5,5,0]` é uma das trocas possíveis).

V8

Instituto Superior Técnico — LMAC/MEBiom

Elementos de Programação

Novembro 2020 Mini-teste 2 Duração: 30m

Nesta avaliação não pode usar definições por compreensão nem métodos. As únicas operações sobre listas permitidas são: lista vazia ([]), acesso aos elementos da lista por posição (lista[posição]), seccionamento da lista (lista[posição:posição]), comparação com a lista vazia (==[]), cálculo do comprimento (`len`) e concatenação (+).

Consideram-se listas de valores numéricos. Em muitas situações é importante que trabalhemos com listas ordenadas (por ordem crescente, em sentido lato). Qualquer lista, mesmo que não ordenada, pode ser decomposta em segmentos ordenados.

Por exemplo, à lista `w=[1,2,3,4,3,5,5,0]` corresponde a lista (de listas) de segmentos ordenados `[[1,2,3,4],[3,5,5],[0]]`.

Dizemos que a taxa de ordenação de uma lista é o comprimento do seu maior segmento ordenado, a dividir pelo comprimento total.

Obviamente, uma lista ordenada tem apenas um segmento e a sua taxa de ordenação é 1. A taxa de ordenação da lista `w` é 0.5.

(**a**) Defina imperativamente em *Python* uma função `segs` que dada uma lista de números, calcula a sua lista de segmentos.

Por exemplo, `segs(w)` deverá ser `[[1,2,3,4],[3,5,5],[0]]`.

(**b**) Defina imperativamente em *Python* uma função `taxa` que dada uma lista de valores calcula a sua taxa de ordenação.

Por exemplo, `taxa(w)` deverá ser `0.5`.

(**c**) Defina imperativamente em *Python* uma função `junta` que dada uma lista de listas, calcula a lista que resulta de concatenar todas essas listas; e use-a para definir uma função `rodando` que dada uma lista de valores calcula (uma)a lista com taxa de ordenação máxima de entre as que se podem obter a partir da lista dada rodando os seus segmentos (por rotação dos segmentos entende-se colocar sucessivamente o primeiro segmento no final da lista, até voltar à lista inicial).

Por exemplo, `junta([[1,2,3,4],[3,5,5],[0]]` deverá ser `[1,2,3,4,3,5,5,0]`, e `rodando(w)` deverá ser `[3,5,5,0,1,2,3,4]` (com uma taxa de ordenação `taxa([3,5,5,0,1,2,3,4])` igual a `0.625`, a maior de entre as listas `w`, `[3,5,5,0,1,2,3,4]`, `[0,1,2,3,4,3,5,5]` obtidas sucessivamente por rotação dos segmentos de `w`).