Final ONI 2014 - Sessão de Discussão dos Problemas

Esta sessão de discussão pretende apenas dar algumas linhas orientadoras sobre os problemas, e não providenciar soluções completas.

Para melhor a compreender é conveniente ter alguma noções básicas de análise de algoritmos e da notação Big O.

Problema A - Tabuleiros Axandrezados

Tipo de problema: Ad-Hoc, Programação Dinâmica
Autor: Pedro Ribeiro (DCC/FCUP)

Problema
Dada uma matriz com **L** linhas e **C** colunas cujas células são claras ou escuras, pretende-se descobrir qual a área máxima de um subrectângulo axadrezado, bem como o número de subrectângulos axadrezados diferentes que têm esse número de células.

Restrições
$1 \leq L, C \leq 1000$
Ao longo desta análise consideraremos, sem perda de generalidade, matrizes quadradas de lado N (N = L = C).

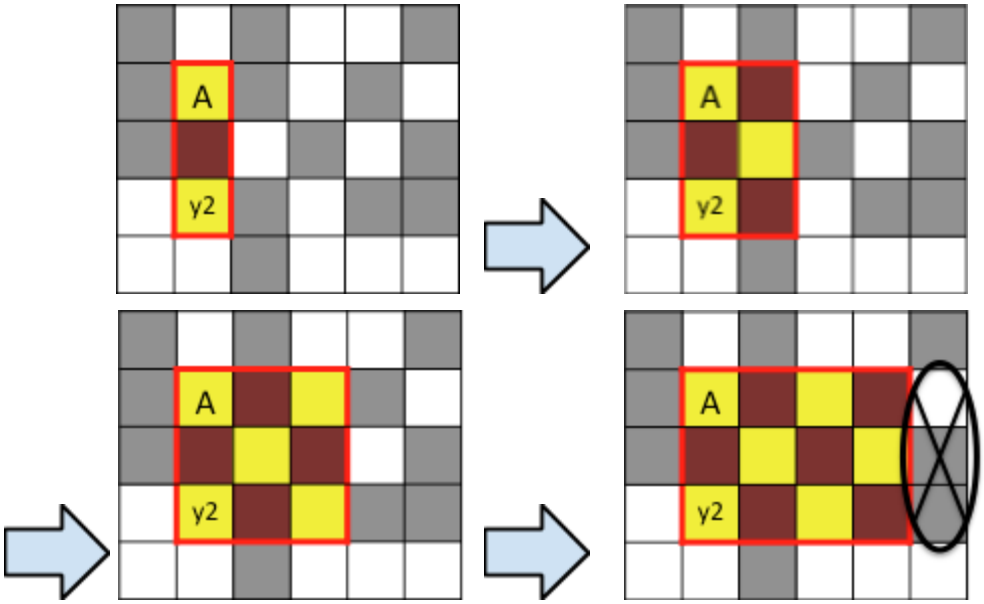
Solução base
Para cada par de células, A e B (ver figura), verificar que o rectângulo definido por estas células é axandrezado. Seria algo como:


```
1  Para cada linha y: 1 .. N
2     Para cada coluna x: 1 .. N
3        Para cada linha y2: y .. N
4           Para cada coluna x2: x .. N
5              Para cada linha i: y .. y2
6                 Para cada coluna j: x .. x2
7                    Verificar se a célula (i, j) não tem vizinhos no rectângulo com cores iguais
8              Se todas as células são válidas, comparar a área com o resultado
```

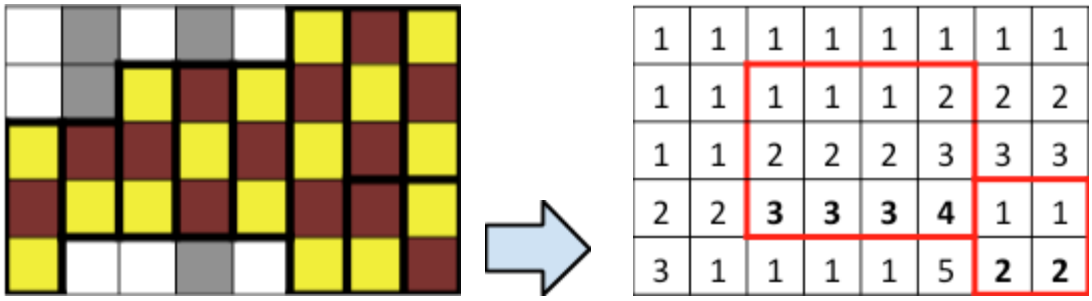
Esta abordagem teria uma complexidade temporal de $O(N^6)$ e daria cerca de 40 pontos.

Programação dinâmica
Na solução base, estamos a repetir diversas vezes a verificação se uma porção de uma coluna ou linha é axandrezada. Uma abordagem melhor é:

1. Fixar a célula A nas linha y, coluna x
2. Testar todas as alturas possíveis para o rectangulo, ie todos os y2 de y a N como acima
3. Avançar coluna a coluna e parar se o rectângulo for inválido - basta verificar a coluna que estamos a adicionar



Aplicando esta abordagem directamente, a complexidade seria $O(N^5)$, mas podemos fazer melhor. Podemos pré-calcular uma matriz auxiliar onde cada célula contém a altura do maior rectângulo axandrezado de largura 1 (só nessa coluna) que termina nessa célula, exemplo:



1	1	1	1	1	1	1	1
1	1	1	1	1	2	2	2
1	1	2	2	2	3	3	3
2	2	**3**	**3**	**3**	**4**	1	1
3	1	1	1	1	5	**2**	**2**

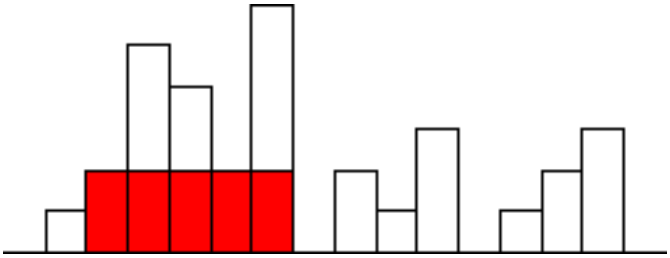
Para verificar se uma coluna é válida, podemos simplesmente ver se a altura no canto inferior direito é suficiente e se a côr dessa célula é diferente da anterior. Esta abordagem teria uma complexidade temporal de $O(N^4)$ e daria cerca de 60 pontos.

Maior rectângulo num histograma - abordagem base $O(N^2)$

Mesmo assim, ainda estamos a fazer muitas verificações desnecessárias. Olhando com atenção para a matriz acima com as altura de cada célula, podemos ver que esses valores limitam a altura dos rectângulos axandrezados cuja base se situa na linha da célula.

Considerando apenas uma linha da matriz, podemos determinar o rectângulo de maior área que assenta nessa linha. Para isso, podemos usar uma estratégia semelhante ao Problema A: Publicidade da Final das ONI 2006.

As alturas podem representar os valores num histograma, exemplo do enunciado desse problema:



Os rectângulos estão limitados pelo menor valor numa sequência contigua de alturas.
Assim, uma abordagem simples é testar cada altura e expandir o rectângulo para a esquerda e direita dessa coluna enquanto as alturas são maiores ou iguais à altura que estamos a testar e as cores das células batem certo.

Esta abordagem para determinar o maior rectângulo num histograma funciona em $O(N^2)$. Como temos de fazer esta verificação para cada uma das N linhas, esta abordagem tem um tempo total de $O(N^3)$ que teria cerca de 80 pontos.

Maior rectângulo num histograma em O(N)
No entanto, o limite para N é 1000 e $O(N^3)$ é demasiado lento. Felizmente, existe um algoritmo mais eficiente para determinar o maior rectângulo num histograma.

Processamos as alturas da esquerda para a direita. Vamos manter uma pilha com informação das alturas já processadas. Inicialmente, coloca um par de inteiros na pilha, representando a altura na coluna 0 e o indice dessa altura (zero). Depois, para cada altura:

- Se a cor da célula actual for igual à cor da célula anterior, sabemos que não podemos formar um rectângulo com células anteriores. Assim, removemos os elementos da pilha enquanto estes forem maiores ou iguais à altura actual. Quando removemos uma altura, sabemos que podemos formar um rectângulo com dimensões altura_na_pilha * (coluna_dessa_altura - coluna_actual)
- Se a altura no topo da pilha for menor do que a altura actual, adicionamos simplesmente um par de inteiros com a altura actual e a respectiva coluna.
- Se a altura no topo da pilha for menor ou igual à altura actual, quer dizer que esta altura limita qualquer rectângulo que fosse formado por células anteriores. Logo, removemos as alturas da pilha que são maiores ou iguais à altura actual.

Tome-se como exemplo a linha 4 da matriz dada anteriormente: “2, 2, 3, 3, 3, 4, 1, 1”
0) Inicialmente a pilha começa com os valores <2, 0>
1) A[1] = 2 e a côr bate certo. O topo da pilha tem altura 2, logo remove-o porque é maior ou igual à altura actual - é possível formar um rectângulo 2x1 (altura na pilha * diferença da coluna actual e a coluna na pilha). Adiciona <2, 0> à pilha pois 0 era a coluna do valor da pilha.
2) A[2] = 3 mas a côr é igual: esvazia a pilha. O topo é <2, 0>, logo é possível formar um rectângulo 2x2. Adiciona <3, 2> à pilha.
3) A[3] = 3 e côr ok. Faz o mesmo que em 1), remove o topo e adiciona <3, 2>.

4) A[4] = 3 e côr ok. Faz o mesmo que em 3), remove o topo e adiciona <3, 2>.
5) A[5] = 4 e côr ok. A altura no topo da pilha é menor, por isso apenas adiciona <4, 5>.
6) A[6] = 1 e côr não ok. A altura no topo da pilha é maior, por isso esvazia a pilha (não há alturas na pilha menores que 1):
- Topo <4, 5> é possível formar um rectângulo 4x1
- Topo <3, 2> é possível formar um rectângulo 3x4 (vai ser a solução)
Adiciona <1, 6> à pilha.
7) A[7] = 1 e côr ok. A altura no topo da pilha é igual, remove o topo e adiciona <1, 6>.
Fim) Esvazia a pilha. Encontra <1, 6> logo é possível formar um rectângulo 1x2.

Para cada uma das N linhas, determinamos o maior rectângulo no histograma dessa linha em O(N), por isso, o total é $O(N^2)$ que seria merecedor dos 100 pontos!

Ligações interessantes

- Estrutura de dados Stack (pilha) na Wikipedia.
- Maior rectângulo num histograma no site geeksforgeeks
- Programação Dinâmica na Wikipedia e no TopCoder.

Problema B - Comunicação Fragmentada

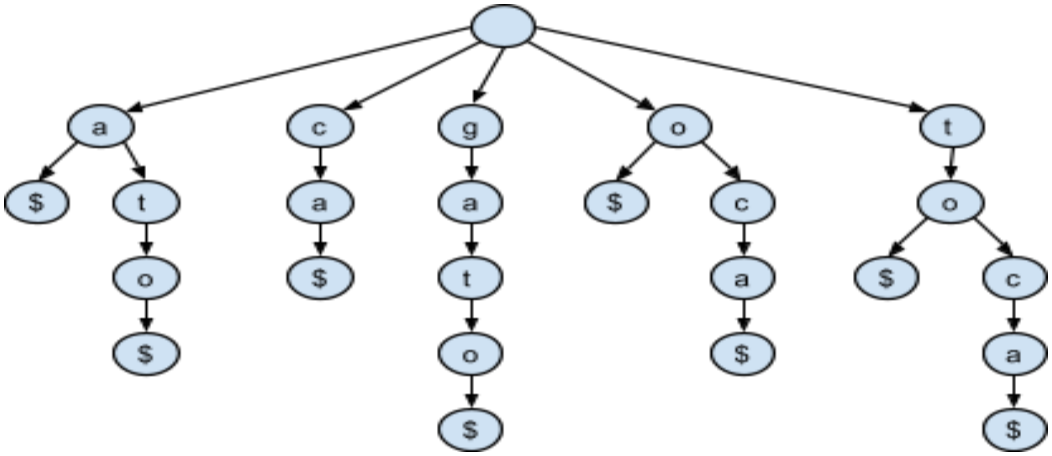
Tipo de problema: String, Data Structures
Autor: João Ramos (FCUP)

Problema
Dado um conjunto de **N** palavras a tua tarefa é descobrir, para cada palavra, qual o seu fragmento únivoco de tamanho mínimo, ou seja, uma subsequência contígua de letras que não aparece em nenhuma das outras palavras. Se para uma mesma palavra existirem vários possíveis fragmentos mínimos, deves escolher aquele que seja alfabeticamente menor.

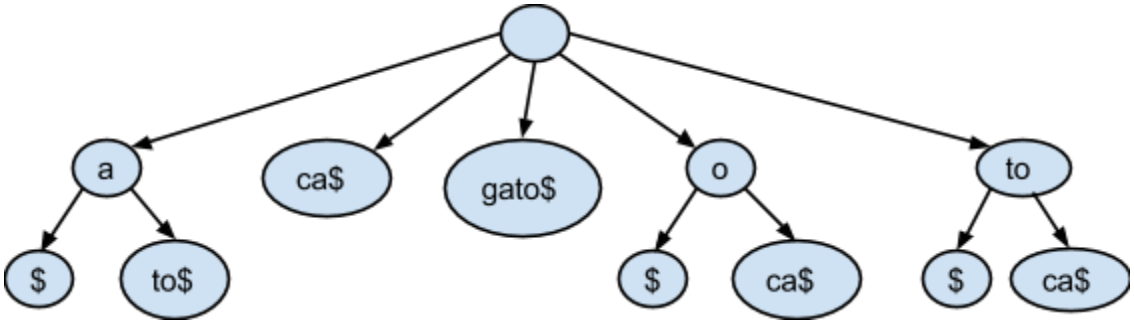
Restrições
$1 \leq N \leq 1\,000$ Número de palavras
$1 \leq L \leq 100$ Tamanho das palavras

Solução base
Para cada cada palavra e cada uma das suas subsequências contíguas, verificar se é uma subsequência contígua de uma das outras palavras. Verificar se é uma subsequência contígua de uma palavra pode ser feito em $O(L^2)$ fazendo "à bruta" ou em $O(L)$ utilizando *KMP* ou *Z-algorithm*. Ficando a complexidade final $O(N^2 * L^4)$ ou $O(N^2 * L^3)$ respectivamente.
Esta solução daria cerca de 40 e 70 pontos (com alguns cortes) respectivamente.

Usando uma Suffix Trie
Para melhorar seria necessário utilizar uma estrutura de dados que ajude a verificar se uma subsequência contígua existe em uma outra palavra. Para isso construímos uma Suffix Trie que é uma àrvore que contém todos os sufixos de cada palavra. Depois de construída a Suffix Trie podemos pesquisar cada sufixo de uma palavra e ver qual o menor prefixo do sufixo que identifica unicamente a palavra. A construção pode ser feita em $O(N * L)$ e cada pesquisa em $O(L)$, resultando numa complexidade final de $O(N * L^2)$, pois cada palavra tem L sufixos.
Esta solução daria cerca de 90 pontos.



Com compressão
Para conseguir os 100 pontos seria necessário comprimir a Suffix Trie para evitar ultrapassar os limites de memória.



Ligações interessantes

- Trie na Wikipedia
- Suffix Trees com ilustração visual - http://www.allisons.org/ll/AlgDS/Tree/Suffix/

Problema C - Blocos Numéricos

Tipo de problema: Programação Dinâmica
Autor: Pedro Ribeiro (DCC/FCUP)

Problema
Dado o conteúdo de uma pilha de **B** blocos com dígitos, e um número **N**, a tarefa é descobrir qual o menor número maior ou igual a **N** que se consegue formar.

Restrições
$1 \leq B \leq 200$

Testar todos os números maiores ou iguais a N
Uma abordagem possível é começar no número N e verificar se é possível formar esse número com os digitos na pilha. Se não for possível, incrementa o número e verifica outra vez. A verificação pára se o número atingir mais de B digitos.

Para fazer a verificação, podemos percorrer a pilha de trás para a frente e manter 2 apontadores: A para o primeiro digito de N e B para o último digito.
Se o digito na pilha for igual ao digito de N na posição A
  Incrementa o apontador A
Senão, se o digito na pilha for igual ao digito na posição B
  Decrementa o apontador B
Senão
  É impossível formar este número.

Esta abordagem seria suficiente para cerca de 30 pontos.

Gerar todos os números possíveis de formar com a pilha de digitos
À medida que retiramos digitos da pilha, temos 2 hipóteses: colocar o digito à esquerda ou direita do número actual. Assim, podemos usar uma função recursiva para testar todas as hipoteses. Com **B** digitos, isto leva-nos a 2^B números possíveis. Para cada número, temos de o comparar com B e com a solução actual. Assim o tempo total é $O(B * 2^B)$ e daria ~50 pontos.

N = 000 … 000
Se N só contém zeros, o problema é mais simples: determinar qual é o menor número que podemos formar com a pilha. Neste caso, podemos usar uma estratégia gananciosa. Percorremos a pilha da esquerda para a direita. Para cada digito, se este for menor ou igual ao digito mais à esquerda do número actual, adiciona o digito à esquerda. Caso contrário, adiciona à direita.

Esta abordagem funciona para 20% dos pontos. Adicionando à abordagem anterior, seria suficiente para 70 pontos.

Aplicar Programação Dinâmica
Se percorrermos a pilha da direita para a esquerda, definimos logo a posição final do número em relação aos digitos mais à esquerda na pilha.
Podemos imaginar que dividimos o número que estamos em digitos adicionados à esquerda - prefixo - e digitos adicionados à direita - sufixo.

Considere-se um exemplo em que a pilha contém “45154” e o N é “45000”. Percorremos da direita para a esquerda.
Se adicionarmos todos os números à esquerda, o prefixo fica com “45154” e o sufixo vazio - “”.

Num caso mais geral, existem essencialmente 3 casos:
1) Imaginem que adicionamos 4 e 5 ao prefixo
Pilha com “451”, prefixo “45”, sufixo “” -> formamos um número “45 … ”
Como o prefixo já é igual ao início de N, então continuamos a querer o menor número maior ou igual ao resto de N.

2) Imaginem que adicionamos o 4 ao sufixo e o 5 ao prefixo
Pilha com “451”, prefixo “5”, sufixo “4” -> formamos um número “5 … 4”
Como o prefixo já é maior do que o início de N, então queremos o menor número possível de formar com os digitos na pilha -> “51454”

3) Imaginem que adicionamos o 4 e o 5 ao sufixo
Pilha com “451”, prefixo “”, sufixo “54” -> formamos um número “ … 54”
Como ‘1’ é menor que o primeiro digito de N, então é impossível formar um número válido se colocarmos o ‘1’ no prefixo, logo ignora essa hipótese.

Sabendo que temos X caracteres no prefixo, Y no sufixo e aquilo que estamos a procurar, podemos usar programação dinâmica para determinar e guardar qual é o melhor número que devemos formar.

Ligações interessantes:
- Programação Dinâmica na Wikipedia e no TopCoder.