PROBLEMAS PROGRAMAÇÃO DINÂMICA

Caminho simples mais longo em um grafo acíclico direcionado
Suponha que nos seja dado um grafo acíclico direcionado G = (V, E) com pesos de aresta de valor real e dois vértices distintos s e t. Descreva uma abordagem de programação dinâmica para encontrar um caminho simples ponderado mais longo de s para t. Como é o grafo de subproblema? Qual é a eficiência do seu algoritmo?

Maior subsequência de palíndromo
Um palíndromo é uma sequência não vazia sobre algum alfabeto que lê o mesmo para a frente e para trás. Exemplos de palíndromos são todas as sequências de comprimento 1, civic, racecar e aibohphobia (medo de palíndromos). Dê um algoritmo eficiente para encontrar o maior palíndromo que é uma subsequência de uma determinada sequência de entrada. Por exemplo, dado o caractere de entrada, seu algoritmo deve retornar carac. Qual é o tempo de execução do seu algoritmo?

Imprimindo ordenadamente
Considere o problema de imprimir ordenadamente um parágrafo com uma fonte monoespaçada (todos os caracteres tendo a mesma largura) em uma impressora. O texto de entrada é uma sequência de n palavras de comprimentos $l_1, l_2, \ldots, l_n$, medidos em caracteres. Queremos imprimir este parágrafo ordenadamente em um número de linhas que contenham no máximo M caracteres cada. Nosso critério de "ordenação" é o seguinte. Se uma determinada linha contém palavras de i a j , onde i <= j , e deixamos exatamente um espaço entre as palavras, Pj o número de caracteres de espaço extra no final da linha é $M - j + i - \Sigma^{j}_{k=i} l_k$, que deve ser não negativo para que as palavras caibam na linha. Desejamos minimizar a soma, em todas as linhas, exceto a última, dos cubos dos números de caracteres de espaço extra nas extremidades das linhas. Dê um algoritmo de programação dinâmica para imprimir um parágrafo de n palavras ordenadamente em uma impressora. Analise os requisitos de tempo de execução e espaço do seu algoritmo.

Editar distância
Para transformar uma sequência de texto de origem x[1..m] em uma sequência de destino y[1..n], podemos executar várias operações de transformação. Nosso objetivo é, dados x e y, produzir uma série de transformações que mudam x para y. Usamos um array Z — assumido como grande o suficiente para conter todos os caracteres de que precisará — para conter os resultados intermediários. Inicialmente, Z está vazio e, no término, devemos ter Z[j]= y[j] para j = 1, 2,...,n. Mantemos os índices atuais i em x e j em Z, e as operações podem alterar Z e esses índices. Inicialmente, i = j = 1. Precisamos examinar cada caractere em x durante a transformação, o que significa que, no final da sequência de operações de transformação, devemos ter i = m + 1.
Podemos escolher entre seis operações de transformação:
Copiar um caractere de x para Z definindo Z[j]= x[i] e, em seguida, incrementando i e j. Esta operação examina x[i].
Substituir um caractere de x por outro caractere c, definindo Z[j]=c e, em seguida, incrementando i e j. Esta operação examina x[i]
Excluir um caractere de x incrementando i, mas deixando j sozinho. Esta operação examina x[i].
Inserir o caractere c em Z definindo Z[j] = c e, em seguida, incrementando j, mas deixando i sozinho. Esta operação não examina nenhum caractere de x.
Alterar (ou seja, trocar) os próximos dois caracteres copiando-os de x para Z, mas na ordem oposta; fazemos isso definindo Z[j]= x[i+1] e Z[j+1] = x[i] e, em seguida, definindo i = i + 2 e j = j + 2. Esta operação examina x[i] e x[i + 1].
Eliminar o restante de x definindo i = m + 1. Esta operação examina todos os caracteres em x que ainda não foram examinados. Esta operação, se realizada, deve ser a operação final.

Como exemplo, uma maneira de transformar o algoritmo da string de origem na string de destino altruísta é usar a seguinte sequência de operações, onde os caracteres sublinhados são x[i] e Z[j] após a operação:

Operação	x	Z
strings iniciais	algoritmo	
copiar	**a**lgoritmo	a
copiar	a**l**goritmo	al
substituir por t	al**g**oritmo	alt
excluir	alg**o**ritmo	alt
copiar	algo**r**itmo	altr
inserir u	algor**i**tmo	altru
inserir í	algor**i**tmo	altruí
inserir s	algor**i**tmo	altruís
alterar	algorit**m**o	altruísti
inserir c	algorit**m**o	altruístic
excluir	algorit**m**o	altruístic
copiar	algoritm**o**	altruístico

Observe que há várias outras sequências de operações de transformação que transformam o algoritmo em altruísta.
Cada uma das operações de transformação tem um custo associado. O custo de uma operação depende da aplicação específica, mas assumimos que o custo de cada operação é uma constante que conhecemos. Também assumimos que os custos individuais das operações de copiar e substituir são menores do que os custos combinados das operações de excluir e inserir; caso contrário, as operações de copiar e substituir não seriam usadas. O custo de uma determinada sequência de operações de transformação é a soma dos custos das operações individuais na sequência. Para a sequência acima, o custo de transformar o algoritmo em altruísta é:

(4*custo(copiar))+(1*custo(substituir)) + (2*custo(excluir)) + (4*custo(inserir)) + (1*custo(alterar))

a) Dadas duas sequências x[1..m] e y[1..n] e um conjunto de custos de operação de transformação, a distância de edição de x para y é o custo da sequência de operação menos dispendiosa que transforma x para y. Descreva um algoritmo de programação dinâmica que encontra a distância de edição de x[1..m] para y[1..n] e imprime uma sequência de operação ótima. Analise os requisitos de tempo de execução e espaço do seu algoritmo.

O problema de distância de edição generaliza o problema de alinhamento de duas sequências de DNA (veja, por exemplo, Setubal e Meidanis [310, Seção 3.2]). Existem vários métodos para medir a similaridade de duas sequências de DNA alinhando-as. Um desses métodos para alinhar duas sequências x e y consiste em inserir espaços em locais arbitrários nas duas sequências (incluindo em cada extremidade) para que as sequências resultantes x’ e y’ tenham o mesmo comprimento, mas não tenham um espaço na mesma posição (ou seja, para nenhuma posição j são ambos x’[j] e y’[j] um espaço). Então atribuímos uma “pontuação” a cada posição. A posição j recebe uma pontuação da seguinte forma:
+1 se x' [j] = y´ [j] e nenhum for um espaço,
-1 se x' [j] ≠ y' [j] e nenhum for um espaço,

-2 se x' [j] ou y' [j] for um espaço.

A pontuação para o alinhamento é a soma das pontuações das posições individuais. Por exemplo, dadas as sequências x = GATCGGCAT e y = CAATGTGAATC, um alinhamento é

```
G ATCG GCAT
CAAT GTGAATC
-*++*+*+-++*
```

Um + sob uma posição indica uma pontuação de +1 para essa posição, um - indica uma pontuação de -1 e um * indica uma pontuação de -2, de modo que esse alinhamento tem uma pontuação total de 6*1 - 2*1 - 4*2 = -4.

b) Explique como lançar o problema de encontrar um alinhamento ótimo como um problema de distância de edição usando um subconjunto das operações de transformação copiar, substituir, excluir, inserir, alterar e eliminar.

Planejando uma festa da empresa

O professor Stewart está prestando consultoria para o presidente de uma corporação que está planejando uma festa da empresa. A empresa tem uma estrutura hierárquica; ou seja, a relação de supervisor forma uma árvore com raiz no presidente. O escritório de pessoal classificou cada funcionário com uma classificação de convivência, que é um número real. Para tornar a festa divertida para todos os participantes, o presidente não quer que um funcionário e seu supervisor imediato compareçam. O professor Stewart recebe a árvore que descreve a estrutura da corporação, usando a representação filho esquerdo e irmão direito descrita na Seção 10.4. Cada nó da árvore contém, além dos ponteiros, o nome de um funcionário e a classificação de convivência desse funcionário. Descreva um algoritmo para compor uma lista de convidados que maximize a soma das classificações de convivência dos convidados. Analise o tempo de execução do seu algoritmo.

Algoritmo de Viterbi

Podemos usar programação dinâmica em um grafo direcionado G=(V,E) para reconhecimento de fala. Cada aresta(u,v) € E é rotulada com um som (u, v) de um conjunto finito Σ de sons. O grafo rotulado é um modelo formal de uma pessoa falando uma língua restrita. Cada caminho no grafo começando de um vértice distinto v_0 € V corresponde a uma possível sequência de sons produzidos pelo modelo. Definimos o rótulo de um caminho direcionado como a concatenação dos rótulos das arestas naquele caminho.

a) Descreva um algoritmo eficiente que, dado um grafo G rotulado por arestas com vértice distinto v_0 e uma sequência s = {1,2,...,k) de sons de Σ, retorna um caminho em G que começa em v_0 e tem s como seu rótulo, se tal caminho existir. Caso contrário, o algoritmo deve retornar NENHUM-CAMINHO. Analise o tempo de execução do seu algoritmo. (Dica: você pode achar úteis os conceitos do Capítulo 22.)

Agora, suponha que cada aresta(u,v) € E tenha uma probabilidade não negativa associada p(u,v) de atravessar a aresta (u, v) do vértice u e , portanto, produzir o som correspondente. A soma das probabilidades das arestas deixando qualquer vértice é igual a 1. A probabilidade de um caminho é definida como o produto das probabilidades de suas arestas. Podemos ver a probabilidade de um caminho começando em v_0 como a probabilidade de que uma "caminhada aleatória" começando em v_0 seguirá o caminho especificado, onde escolhemos aleatoriamente qual aresta tomar deixando um vértice u de acordo com as probabilidades das arestas disponíveis deixando u.

b)Estenda sua resposta para a parte (a) de modo que, se um caminho for retornado, seja um caminho mais provável começando em v_0 e tendo rótulo s. Analise o tempo de execução do seu algoritmo.

Compressão de imagem por entalhe de costura

Recebemos uma imagem colorida consistindo de uma matriz m x n A[1..m, 1..n] de pixels, onde cada pixel especifica um triplo de intensidades vermelho, verde e azul (RGB). Suponha que desejamos comprimir esta imagem ligeiramente. Especificamente, desejamos remover um pixel de cada uma das m linhas, para que a imagem inteira se torne um pixel mais estreita. Para evitar efeitos visuais perturbadores, no entanto, exigimos que os pixels removidos em duas linhas adjacentes estejam na mesma coluna ou em colunas adjacentes; os pixels removidos formam uma "costura" da linha superior para a linha inferior, onde pixels sucessivos na costura são adjacentes verticalmente ou diagonalmente.

a. Mostre que o número de tais costuras possíveis cresce pelo menos exponencialmente em m, assumindo que n > 1.

b. Suponha agora que, junto com cada pixel A[i,j], calculamos uma medida de interrupção de valor real d[i,j], indicando quão disruptivo seria remover o pixel A[i,j]. Intuitivamente, quanto menor a medida de interrupção de um pixel, mais semelhante o pixel é aos seus vizinhos. Suponha ainda que definimos a medida de interrupção de uma costura como a soma das medidas de interrupção de seus pixels.
Dê um algoritmo para encontrar uma costura com a menor medida de interrupção. Quão eficiente é seu algoritmo?

Quebrando uma string

Uma certa linguagem de processamento de string permite que um programador quebre uma string em duas partes. Como essa operação copia a string, custa n unidades de tempo para quebrar uma string de n caracteres em duas partes. Suponha que um programador queira quebrar uma string em muitas partes. A ordem em que as quebras ocorrem pode afetar a quantidade total de tempo usada. Por exemplo, suponha que o programador queira quebrar uma string de 20 caracteres após os caracteres 2, 8 e 10 (numerando os caracteres em ordem crescente da extremidade esquerda, começando em 1). Se ele programar as quebras para ocorrerem na ordem da esquerda para a direita, então a primeira quebra custa 20 unidades de tempo, a segunda quebra custa 18 unidades de tempo (quebrando a string dos caracteres 3 a 20 no caractere 8) e a terceira quebra custa 12 unidades de tempo, totalizando 50 unidades de tempo. Se ela programar as quebras para ocorrerem na ordem da direita para a esquerda, no entanto, a primeira quebra custa 20 unidades de tempo, a segunda quebra custa 10 unidades de tempo e a terceira quebra custa 8 unidades de tempo, totalizando 38 unidades de tempo. Em outra ordem, ela poderia quebrar primeiro em 8 (custando 20), depois quebrar a peça da esquerda em 2 (custando 8) e, finalmente, a peça da direita em 10 (custando 12), para um custo total de 40.
Projete um algoritmo que, dado o número de caracteres após os quais quebrar, determine uma maneira de menor custo para sequenciar essas quebras. Mais formalmente, dada uma string S com n caracteres e uma matriz L[1..m] contendo os pontos de quebra, calcule o menor custo para uma sequência de quebras, junto com uma sequência de quebras que atinja esse custo.

Planejando uma estratégia de investimento

Seu conhecimento de algoritmos ajuda você a obter um emprego empolgante na Acme Computer Company, junto com um bônus de assinatura de $ 10.000. Você decide investir esse dinheiro com o objetivo de maximizar seu retorno ao final de 10 anos. Você decide usar a Amalgamated Investment Company para gerenciar seus investimentos. A Amalgamated Investments exige que você observe as seguintes regras. Ela oferece n investimentos diferentes, numerados de 1 a n. Em cada ano j , o investimento i fornece uma taxa de retorno de r_{ij} . Em outras palavras, se você investir d dólares no

investimento i no ano j , então no final do ano j , você tem dr_{ij} dólares. As taxas de retorno são garantidas, ou seja, você recebe todas as taxas de retorno para os próximos 10 anos para cada investimento. Você toma decisões de investimento apenas uma vez por ano. No final de cada ano, você pode deixar o dinheiro ganho no ano anterior nos mesmos investimentos, ou pode transferir dinheiro para outros investimentos, transferindo dinheiro entre investimentos existentes ou movendo dinheiro para um novo investimento. Se você não movimentar seu dinheiro entre dois anos consecutivos, você paga uma taxa de f1 dólares, enquanto se você movimentar seu dinheiro, você paga uma taxa de f2 dólares, onde f2 > f1.

a. O problema, como declarado, permite que você invista seu dinheiro em vários investimentos a cada ano. Prove que existe uma estratégia de investimento ótima que, a cada ano, coloca todo o dinheiro em um único investimento. (Lembre-se de que uma estratégia de investimento ótima maximiza a quantidade de dinheiro após 10 anos e não se preocupa com nenhum outro objetivo, como minimizar o risco.)
b. Prove que o problema de planejar sua estratégia de investimento ótima exibe subestrutura ótima.
c. Projete um algoritmo que planeje sua estratégia de investimento ótima. Qual é o tempo de execução do seu algoritmo?
d. Suponha que a Amalgamated Investments impôs a restrição adicional de que, a qualquer momento, você não pode ter mais do que $ 15.000 em qualquer investimento. Mostre que o problema de maximizar sua renda ao final de 10 anos não exibe mais subestrutura ótima.

Planejamento de estoque
A Rinky Dink Company fabrica máquinas que recapeiam pistas de gelo. A demanda por esses produtos varia de mês para mês, então a empresa precisa desenvolver uma estratégia para planejar sua fabricação, dada a demanda flutuante, mas previsível. A empresa deseja elaborar um plano para os próximos n meses. Para cada mês i, a empresa P conhece a demanda d_i , ou seja, o número de máquinas que ela venderá. Seja $D = \Sigma^n_{i=1} d_i$ a demanda total nos próximos n meses. A empresa mantém uma equipe em tempo integral que fornece mão de obra para fabricar até m máquinas por mês. Se a empresa precisar fabricar mais de m máquinas em um determinado mês, ela pode contratar mão de obra adicional em meio período, a um custo que equivale a c dólares por máquina. Além disso, se, no final de um mês, a empresa estiver segurando alguma máquina não vendida, ela deve pagar os custos de estoque. O custo para manter j máquinas é dado como uma função h(j) para j =1, 2,...,D, onde $h(j) \geq 0$ para $1 \leq j \leq D$ e $h(j) \leq h(j+1)$ para $1 \leq j \leq D - 1$.
Dê um algoritmo que calcule um plano para a empresa que minimize seus custos enquanto atende toda a demanda. O tempo de execução deve ser polinomial em n e D.

Contratando jogadores de beisebol free-agent
Suponha que você seja o gerente geral de um time de beisebol da liga principal. Durante a entressafra, você precisa contratar alguns jogadores free-agent para seu time. O dono do time lhe deu um orçamento de $X para gastar com agentes livres. Você tem permissão para gastar menos de $X no total, mas o dono irá demiti-lo se você gastar mais do que $X.
Você está considerando N posições diferentes, e para cada posição, P jogadores free-agent que jogam naquela posição estão disponíveis.8 Como você não quer sobrecarregar seu elenco com muitos jogadores em qualquer posição, para cada posição você pode contratar no máximo um agente livre que joga naquela posição. (Se você não contratar nenhum jogador em uma posição específica, então você planeja ficar com os jogadores que você já tem naquela posição.)
Para determinar o quão valioso um jogador será, você decide usar uma estatística sabermétrica9 conhecida como “VORP” ou “valor sobre jogador substituto”. Um jogador com um VORP mais alto é mais valioso do que um jogador com um VORP mais baixo. Um jogador com um VORP mais alto não é necessariamente mais caro para contratar do que um jogador com um VORP mais baixo, porque outros fatores além do valor de um jogador determinam quanto custa contratá-lo.
Para cada jogador free-agent disponível, você tem três informações:

a posição do jogador,
a quantia de dinheiro que custará para contratar o jogador, e
o VORP do jogador.

Crie um algoritmo que maximize o VORP total dos jogadores que você contratar, gastando não mais do que $X no total. Você pode assumir que cada jogador contrata por um múltiplo de $100.000. Seu algoritmo deve gerar o VORP total dos jogadores que você contratar, a quantia total de dinheiro que você gasta e uma lista de quais jogadores você contratar. Analise o tempo de execução e o requisito de espaço do seu algoritmo.

PROBLEMAS ALGORITMOS GULOSOS

Troco de moedas
Considere o problema de fazer troco para n centavos usando o menor número de moedas. Suponha que o valor de cada moeda seja um inteiro.
a. Descreva um algoritmo ganancioso para fazer troco consistindo de moedas de 25 centavos, 10 centavos, 5 centavos e 1 centavo. Prove que seu algoritmo produz uma solução ótima.

b. Suponha que as moedas disponíveis estejam nas denominações que são potências de c, ou seja, as denominações são c0, c1,..., ck para alguns inteiros $c > 1$ e $k \geq 1$. Mostre que o algoritmo ganancioso sempre produz uma solução ótima.

c. Dê um conjunto de denominações de moedas para as quais o algoritmo ganancioso não produz uma solução ótima. Seu conjunto deve incluir um centavo para que haja uma solução para cada valor de n.

d. Dê um algoritmo de tempo O(nk) que faça troco para qualquer conjunto de k denominações de moedas diferentes, assumindo que uma das moedas seja um centavo.

Agendamento para minimizar o tempo médio de conclusão
Suponha que você tenha um conjunto S={a1, a2,...,an} de tarefas, onde a tarefa ai requer pi unidades de tempo de processamento para ser concluída, uma vez iniciada. Você tem um computador no qual executar essas tarefas, e o computador pode executar apenas uma tarefa por vez. Seja ci o **tempo de conclusão** da tarefa ai , ou seja, o tempo em que a tarefa ai conclui o processamento. Seu objetivo é minimizar o tempo médio de conclusão, ou seja, n para minimizar $(1/n)\ \Sigma^{n}_{i=1}$ ci . Por exemplo, suponha que haja duas tarefas, a1 e a2 , com p1 = 3 e p2 = 5, e considere o cronograma em que a2 é executado primeiro, seguido por a1 . Então c2 = 5, c1 = 8, e o tempo médio de conclusão é (5 + 8)/2 = 6,5. No entanto, se a tarefa a1 for executada primeiro, então c1 = 3, c2 = 8 e o tempo médio de conclusão é (3 + 8)/2 = 5,5.

a. Dê um algoritmo que agende as tarefas de modo a minimizar o tempo médio de conclusão. Cada tarefa deve ser executada de forma não preemptiva, ou seja, uma vez que a tarefa ai comece, ela deve ser executada continuamente por pi unidades de tempo. Prove que seu algoritmo minimiza o tempo médio de conclusão e informe o tempo de execução do seu algoritmo.

b. Suponha agora que as tarefas não estejam todas disponíveis ao mesmo tempo. Ou seja, cada tarefa não pode ser iniciada até seu tempo de liberação ri. Suponha também que permitimos a preempção, para que uma tarefa possa ser suspensa e reiniciada posteriormente. Por exemplo, uma tarefa ai com tempo de processamento pi = 6 e tempo de liberação ri = 1 pode começar a ser executada no tempo 1 e ser preemptada no tempo 4. Ela pode então retomar no tempo 10, mas ser preemptada no tempo 11, e pode finalmente retomar no tempo 13 e ser concluída no tempo 15. A

tarefa ai foi executada por um total de 6 unidades de tempo, mas seu tempo de execução foi dividido em três partes. Neste cenário, o tempo de conclusão de ai é 15. Dê um algoritmo que escalona as tarefas de modo a minimizar o tempo médio de conclusão neste novo cenário. Prove que seu algoritmo minimiza o tempo médio de conclusão e declare o tempo de execução do seu algoritmo.

Cache off-line

Os computadores modernos usam um cache para armazenar uma pequena quantidade de dados em uma memória rápida. Mesmo que um programa possa acessar grandes quantidades de dados, ao armazenar um pequeno subconjunto da memória principal no cache — uma memória pequena, mas mais rápida — o tempo geral de acesso pode diminuir muito. Quando um programa de computador é executado, ele faz uma sequência (r1, r2,...,rn) de n solicitações de memória, onde cada solicitação é para um elemento de dados específico. Por exemplo, um programa que acessa 4 elementos distintos {a, b, c, d} pode fazer a sequência de solicitações <d, b, d, b, d, a, c, d, b, a, c, b>. Seja k o tamanho do cache. Quando o cache contém k elementos e o programa solicita o $(k + 1)^o$ elemento, o sistema deve decidir, para esta e cada solicitação subsequente, quais k elementos manter no cache. Mais precisamente, para cada solicitação ri, o algoritmo de gerenciamento de cache verifica se o elemento ri já está no cache. Se estiver, então temos um acerto de cache; caso contrário, temos uma falha de cache. Em caso de falha de cache, o sistema recupera ri da memória principal, e o algoritmo de gerenciamento de cache deve decidir se deve manter ri no cache. Se ele decidir manter ri e o cache já contiver k elementos, então ele deve despejar um elemento para abrir espaço para ri. O algoritmo de gerenciamento de cache despeja dados com o objetivo de minimizar o número de falhas de cache em toda a sequência de solicitações.

Normalmente, o cache é um problema on-line. Ou seja, temos que tomar decisões sobre quais dados manter no cache sem saber as solicitações futuras. Aqui, no entanto, consideramos a versão off-line deste problema, na qual nos é dada antecipadamente toda a sequência de n solicitações e o tamanho do cache k, e desejamos minimizar o número total de falhas de cache.

Podemos resolver este problema off-line por uma estratégia gananciosa chamada de mais distante no futuro, que escolhe despejar o item no cache cujo próximo acesso na sequência de solicitações vier mais distante no futuro.

a. Escreva um pseudocódigo para um gerenciador de cache que use a estratégia do futuro mais distante. A entrada deve ser uma sequência (r1, r2,..., rn) de solicitações e um tamanho de cache k, e a saída deve ser uma sequência de decisões sobre qual elemento de dados (se houver) remover em cada solicitação. Qual é o tempo de execução do seu algoritmo?

b. Mostre que o problema de cache off-line exibe subestrutura ótima.

c. Prove que o futuro mais distante produz o número mínimo possível de erros de cache.