Programação Dinâmica

Programação Dinâmica

- Quais são então as características que um problema deve apresentar para poder ser resolvido com PD?
  - Subestrutura ótima
  - Subproblemas coincidentes

- Subestrutura ótima

Quando a solução ótima de um problema contém nela própria Soluções ótimas para subproblemas do mesmo tipo.

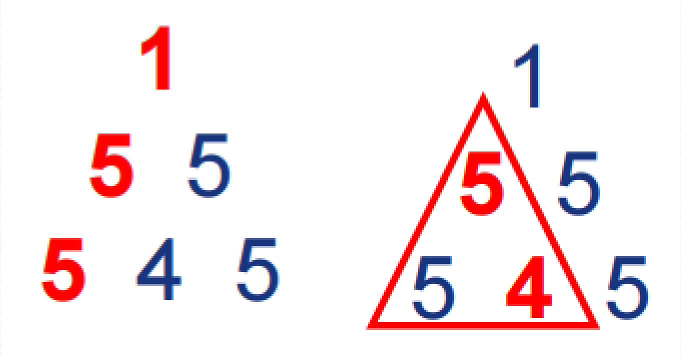
Exemplo: No problema das pirâmides de números, a solução ótima contém nela própria os melhores percursos de sub-pirâmides, ou seja, soluções ótimas de subproblemas.

PD

- . Subestrutura ótima (“optimal substructure”):

  É preciso ter cuidado porque isto nem sempre acontece!

- Exemplo:
  - se, no problema das pirâmides, o objetivo fosse encontrar a rota que maximizasse o resto da divisão inteira entre 10 e o valor dessa rota



PD

- Subproblemas coincidentes:

Quando um espaço de subproblemas é pequeno, isto é, não são muitos os subproblemas a resolver pois muitos deles são exatamente iguais uns aos outros.

- Exemplo:
  - no problema das pirâmides, para um determinada instância do problema, existem apenas $n+(n-1)+...+1<n^2$ subproblemas (crescem polinomialmente) pois, como já vimos, muitos subproblemas que aparecem são coincidentes.

PD

- Se um problema apresenta estas duas características, temos uma boa pista de que a PD se pode aplicar. No entanto, nem sempre isso acontece.
  - Subestrutura ótima
  - Subproblemas coincidentes

PD - Metodologia

1) Caracterizar a solução ótima do problema

2) Definir recursivamente a solução ótima, em função de soluções ótimas de subproblemas

3) Calcular as soluções de todos os subproblemas: “de trás para a frente” ou com memorização

4) Reconstruir a solução ótima, baseada nos cálculos efetuados (opcional)

PD Metodologia

1) Caracterizar a solução óptima de um problema

- Compreender bem o problema
- Verificar se um algoritmo que verifique todas as soluções com força bruta não é suficiente
- Tentar generalizar o problema

  (é preciso prática para perceber como generalizar da maneira correta)
- Procurar dividir o problema em subproblemas do mesmo tipo
- Verificar se o problema obedece ao princípio de otimalidade
- Verificar se existem subproblemas coincidentes

PD Metodologia

2) Definir recursivamente a solução ótima, em função de soluções ótimas de subproblemas.

Definir recursivamente o valor da solução óptima, com rigor e exatidão, a partir de subproblemas menores do mesmo tipo

Imaginar sempre que os valores das soluções ótimas já estão disponíveis quando precisamos deles

Não é necessário codificar. Basta definir matematicamente a recursão.

PD Metodologia

3) Calcular as soluções de todos os subproblemas: “de trás para a frente”

- Descobrir a ordem em que os subproblemas são precisos, a partir dos subproblemas menores até chegar ao problema global (“bottom-up”) e codificar, usando uma tabela.
- Normalmente, esta ordem é inversa à ordem normal da função recursiva que resolve o problema

PD Metodologia

Calcular as soluções de todos os subproblemas: “memorização”

- Existe uma técnica, chamada “memorização”, que permite resolver o problema pela ordem normal (“topdown”)
- Usar a função recursiva obtida diretamente a partir definição da solução e ir mantendo uma tabela com os resultados dos subproblemas.
- Quando queremos aceder a um valor pela primeira vez temos de calculá-lo e a partir daí basta ver qual é.

PD Metodologia

4) Reconstruir a solução ótima, baseada nos cálculos efetuados

- Pode ou não ser requisito do problema
- Duas alternativas:
  - Diretamente a partir da tabela dos sub-problemas
  - Nova tabela que guarda as decisões em cada etapa

Não necessitando de saber qual a melhor solução, podemos por vezes poupar espaço

Exemplo

- Dada uma sequência de números inteiros:

  7, 6, 10, 3, 4, 1, 8, 9, 5, 2

- Descobrir qual a maior subsequência crescente (não necessariamente contígua)

Exemplo

- 7, 6, 10, 3, 4, 1, 8, 9, 5, 2 – Tamanho 2
- 7, 6, 10, 3, 4, 1, 8, 9, 5, 2 – Tamanho 3
- 7, 6, 10, 3, 4, 1, 8, 9, 5, 2 – Tamanho 4

- Caracterizar solução ótima
  - Seja N o tamanho da sequência, e num[i] o i-ésimo número
  - Força bruta, quantas opções? (binomial theorem: 2^{n-1})

- Generalizar e resolver com subproblemas iguais:
  - seja best(i) o valor da melhor subsequência a partir da i-ésima posição
  - Caso fácil: a melhor subsequência a começar da última posição tem tamanho 1
  - Caso geral: para um dado i, podemos seguir para todos os números entre i+1 e N, desde que **sejam maiores**

- N – tamanho da sequência
- num[i] – número na posição i
- best(i) – melhor subsequência a partir da posição i

best(N) $= 1$

best(i) $= 1 + \text{máximo}\{\text{best}(j): i<j\leq N,\ \text{num}[j]>\text{num}[i]\}$ para $1\leq i<N$

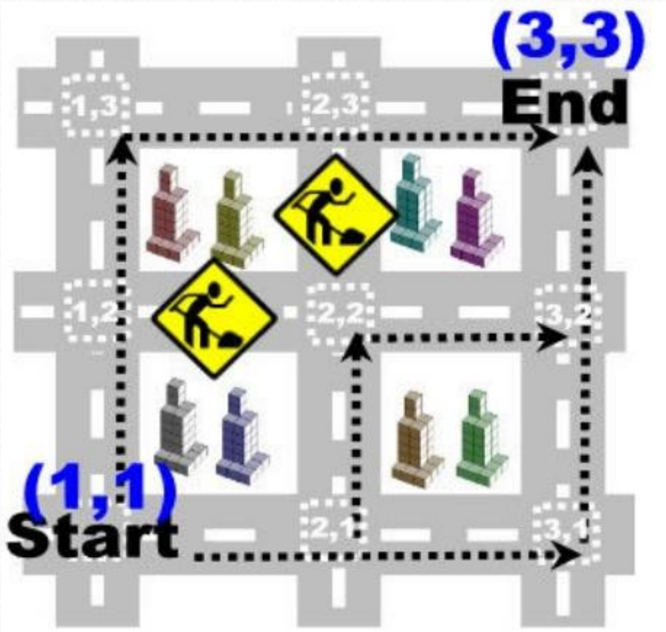
```
Calcular ():
    best[N] = 1
    Para  i: n-1 até 1 fazer
        best[i] = 1
        Para  j: i+1 até N fazer
            Se num[j]>num[i] e 1+best[j]>best[i] então
                best[i] = 1+best[j]
```

i	1	2	3	4	5	6	7	8	9	10
num[i]	7	6	10	3	4	1	8	9	5	2
best[i]	3	3	1	4	3	3	2	1	1	1

Exemplo

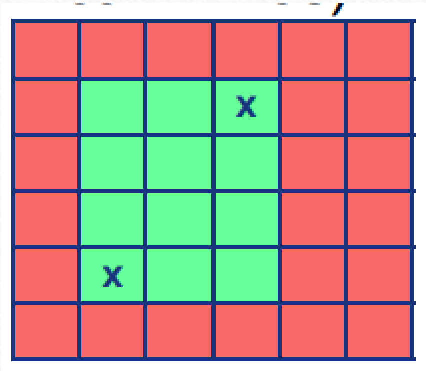
- Cidade com "quadriculado" de ruas (MIUP'2004)
- Algumas estradas têm obras
- Só se pode andar para norte e para leste

De quantas maneiras diferentes se pode ir de (x_1,y_1) para (x_2,y_2) ?



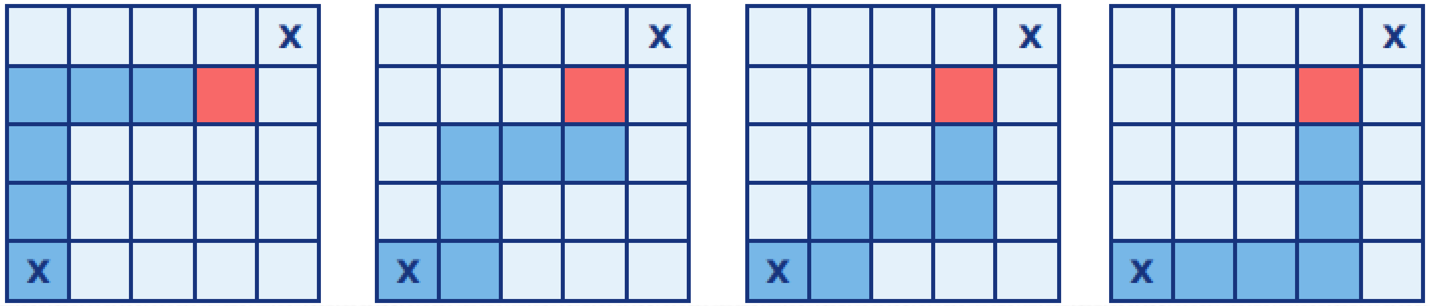
Caracterização do Problema

- Força bruta? (com N=30, existem ~1,2x1017 caminhos)
- Ir de (x1,y1) para (x2,y2): pode ignorar-se tudo o que está “fora “desse retângulo



Caracterização do Problema

- Número de maneiras a partir de uma posição é igual ao número de maneiras desde a posição a norte mais o número de maneiras desde a posição a leste
- Subproblema igual com solução não dependente do problema "maior" (equivalente a princípio da otimalidade)
- Existem muitos subproblemas coincidentes!



Definir solução recursiva

- L – número de linhas
- C – número de colunas
- count(i,j) – número de maneiras a partir de posição (i,j)
- obra(i,j,D) – valor V/F indicando se existe obra a impedir deslocação de (i,j) na direção D (NORTE ou LESTE)

Algoritmo

count(L,C) = 1

count(i,j) = valor_norte(i,j) + valor_este(i,j)

para (i,j)≠(L,C)

onde:

$$\text{valor\_norte(i,j)}: \begin{cases} 0 & \text{se (j=L ou obra(i,j,NORTE))} \\ \text{count(i+1,j) caso contrário} & \end{cases}$$

$$\text{valor\_este(i,j)}: \begin{cases} 0 & \text{se (j=L ou obra(i,j,ESTE))} \\ \text{count(i,j+1) caso contrário} & \end{cases}$$

```
Calcular ():
    Inicializar count[][] com zeros
    count[L][C] = 1
    Para  i: L até 1 fazer
        Para  j: C até 1 fazer
            Se i<L e não(obra(i,j,NORTE)) então
                count[i][j] = count[i][j] + count[i+1][j]
            Se j<C e não(obra(i,j,ESTE)) então
                count[i][j] = count[i][j] + count[i][j+1]
```

Count[][]

1	1	1
1	1	1
3	2	1