Programación Dinámica

Leopoldo Taravilse[1]

[1] Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Algoritmos y Estructuras de Datos III

Contenidos

Contenidos

Sucesión de Fibonacci

Sucesión de Fibonacci

La sucesión de Fibonacci se define como $f_0 = 1$, $f_1 = 1$ y $f_{n+2} = f_n + f_{n+1}$ para todo $n \geq 0$

Sucesión de Fibonacci

Sucesión de Fibonacci

La sucesión de Fibonacci se define como $f_0 = 1$, $f_1 = 1$ y $f_{n+2} = f_n + f_{n+1}$ para todo $n \geq 0$

¿Cómo podemos computar el término 100 de la sucesión de Fibonacci?

Sucesión de Fibonacci

Sucesión de Fibonacci

La sucesión de Fibonacci se define como $f_0 = 1$, $f_1 = 1$ y $f_{n+2} = f_n + f_{n+1}$ para todo $n \geq 0$

¿Cómo podemos computar el término 100 de la sucesión de Fibonacci?
¿Cómo podemos hacerlo eficientemente?

Algoritmos recursivos

Definición

Un algoritmo se dice recursivo si calcula instancias de un problema en función de otras instancias del mismo problema hasta llegar a un caso base, que suele ser una instancia pequeña del problema, cuya respuesta generalmente está dada en el algoritmo y no es necesario calcularla.

Algoritmos recursivos

Definición

Un algoritmo se dice recursivo si calcula instancias de un problema en función de otras instancias del mismo problema hasta llegar a un caso base, que suele ser una instancia pequeña del problema, cuya respuesta generalmente está dada en el algoritmo y no es necesario calcularla.

Ejemplo

Para calcular el factorial de un número, un posible algoritmo es calcular Factorial(n) como Factorial($n-1$) $\times n$ si $n \geq 1$ o 1 si $n = 0$

Algoritmos recursivos

Definición

Un algoritmo se dice recursivo si calcula instancias de un problema en función de otras instancias del mismo problema hasta llegar a un caso base, que suele ser una instancia pequeña del problema, cuya respuesta generalmente está dada en el algoritmo y no es necesario calcularla.

Ejemplo

Para calcular el factorial de un número, un posible algoritmo es calcular Factorial(n) como Factorial($n-1$) $\times n$ si $n \geq 1$ o 1 si $n = 0$

Veamos como calcular el n-ésimo fibonacci con un algorimto recursivo

Cálculo Recursivo de Fibonacci

```
int fibo(int n)
{
    if(n<=1)
        return 1;
    else
        return fibo(n-2)+
               fibo(n-1);
}
```

Cálculo Recursivo de Fibonacci

```
int fibo(int n)
{
    if(n<=1)
        return 1;
    else
        return fibo(n-2)+
               fibo(n-1);
}
```

Notemos que fibo(n) llama a fibo($n-2$), pero después se vuelve a llamar a fibo($n-2$) para calcular fibo($n-1$), y a medida que va decreciendo el parámetro que toma fibo son más las veces que se llama a la función fibo con ese parámetro.

Problemas de la recursión

- La función que usamos para calcular Fibonacci tiene un problema.

Problemas de la recursión

- La función que usamos para calcular Fibonacci tiene un problema.
- Llamamos muchas veces a la misma función con los mismos parámetros

Problemas de la recursión

- La función que usamos para calcular Fibonacci tiene un problema.
- Llamamos muchas veces a la misma función con los mismos parámetros
- ¿Podemos solucionar esto? ¿Podemos hacer que la función sea llamada pocas veces para cada parámetro?

Problemas de la recursión

- La función que usamos para calcular Fibonacci tiene un problema.
- Llamamos muchas veces a la misma función con los mismos parámetros
- ¿Podemos solucionar esto? ¿Podemos hacer que la función sea llamada pocas veces para cada parámetro?
- Para lograr resolver este problema, vamos a utilizar el concepto de programación dinámica

Contenidos

Programación dinámica

Definición

La programación dinámica es una técnica para resolver problemas, que consiste en dividir una instancia de un problema en subproblemas, resolviendo una sóla vez cada instancia del problema para poder obtener la respuesta de una instancia del mismo eficientemente.

Programación dinámica

Definición

La programación dinámica es una técnica para resolver problemas, que consiste en dividir una instancia de un problema en subproblemas, resolviendo una sóla vez cada instancia del problema para poder obtener la respuesta de una instancia del mismo eficientemente.

¿Cómo hacemos para calcular una sóla vez cada intancia, por ejemplo, en el caso de Fibonacci?

Cálculo de Fibonacci mediante Programación Dinámica (bottom-up)

```
int fibo[100];
int calcFibo(int n)
{
    fibo[0] = 1;
    fibo[1] = 1;
    for(int i=2;i<=n;i++)
        fibo[n] = fibo[n-2]+fibo[n-1];
    return fibo[n];
}
```

Cálculo de Fibonacci mediante Programación Dinámica (bottom-up)

```
int fibo[100];
int calcFibo(int n)
{
    fibo[0] = 1;
    fibo[1] = 1;
    for(int i=2;i<=n;i++)
        fibo[n] = fibo[n-2]+fibo[n-1];
    return fibo[n];
}
```

Esta forma de calcular los Fibonacci se conoce como bottom-up. Veamos una forma alternativa (top-down) de cómo calcular los Fibonacci.

Cálculo de Fibonacci mediante Programación Dinámica(top-down)

```
int fibo[100];
int calcFibo(int n)
{
    if(fibo[n] != -1)
        return fibo[n];
    if(n <= 1)
        fibo[n] = 1;
    else
        fibo[n] = calcFibo(n-2)+calcFibo(n-1);
    return fibo[n];
}
```

En este caso, inicializamos el arreglo fibo con el valor -1 (un valor arbitrario que nunca puede tomar un número de Fibonacci) para indicar que el número no fue todavía calculado.

Ventajas de la Programación Dinámica

- La función que vimos recién que usa programación dinámica tiene una ventaja con respecto a la versión recursiva que vimos anteriormente.

Ventajas de la Programación Dinámica

- La función que vimos recién que usa programación dinámica tiene una ventaja con respecto a la versión recursiva que vimos anteriormente.
- Llama menos veces a cada función

Ventajas de la Programación Dinámica

- La función que vimos recién que usa programación dinámica tiene una ventaja con respecto a la versión recursiva que vimos anteriormente.
- Llama menos veces a cada función
- Para calcular calcFibo(n-1) necesita calcular calcFibo(n-2), pero ya lo calculamos antes, por lo que no es necesario volver a llamar a calcFibo(n-3) y calcFibo(n-4)

Ventajas de la Programación Dinámica

- La función que vimos recién que usa programación dinámica tiene una ventaja con respecto a la versión recursiva que vimos anteriormente.
- Llama menos veces a cada función
- Para calcular calcFibo(n-1) necesita calcular calcFibo(n-2), pero ya lo calculamos antes, por lo que no es necesario volver a llamar a calcFibo(n-3) y calcFibo(n-4)
- Así podemos calcular calcFibo(50) mucho más rápido ya que este algoritmo es lineal mientras que el anterior era exponencial.

Contenidos

Camino mínimo en una grilla

Un problema muy común en la materia será el de calcular camino mínimo en distintos tipos de estructuras (por ejemplo grafos, árboles, etc). Un caso particular de este problema es el de buscar el camino mínimo en una grilla. Formalmente el problema lo podemos definir de la siguiente manera:

Camino mínimo en una grilla

Un problema muy común en la materia será el de calcular camino mínimo en distintos tipos de estructuras (por ejemplo grafos, árboles, etc). Un caso particular de este problema es el de buscar el camino mínimo en una grilla. Formalmente el problema lo podemos definir de la siguiente manera:

Camino mínimo en una grilla

Dada una matriz de $N \times M$, con enteros en todos sus casilleros, queremos encontrar el camino que vaya desde el casillero superior izquierdo, hasta el casillero inferior derecho, tal que la suma de los casilleros por los que pasa sea mínima, donde los movimientos permitidos son moverse un sólo casillero a la derecha o hacia abajo en cada paso.

Camino mínimo en una grilla - Ejemplo

1	5	4	3	2
2	2	4	5	1
5	1	3	2	8
4	2	2	6	7
1	3	1	2	3

En este caso vemos un ejemplo de una instancia del problmea con su solución marcada en azul.

Camino mínimo en una grilla - Solución

- En este problema, sabemos que para llegar al casillero (N, M) tenemos que venir del casillero $(N-1, M)$ o del casillero $(N, M-1)$

Camino mínimo en una grilla - Solución

- En este problema, sabemos que para llegar al casillero (N, M) tenemos que venir del casillero $(N-1, M)$ o del casillero $(N, M-1)$
- Podríamos resolver las dos instancias recursivamente, pero si vamos a hacer eso, conviene memorizar los resultados, es decir, utilizar programación dinámica.

Camino mínimo en una grilla - Solución

- En este problema, sabemos que para llegar al casillero (N, M) tenemos que venir del casillero $(N-1, M)$ o del casillero $(N, M-1)$
- Podríamos resolver las dos instancias recursivamente, pero si vamos a hacer eso, conviene memorizar los resultados, es decir, utilizar programación dinámica.
- Vamos a ver las dos formas de hacerlo: Bottom Up y Top Down.

Camino mínimo en una grilla - Bottom Up

```
1:  matriz ← int[N][M], mejor ← int[N][M]
2:  function CAMINOMINIMO(n, m)
3:      mejor[0][0] ← matriz[0][0]
4:      for i ← 1..n do
5:          mejor[i][0] ← mejor[i-1][0] + matriz[i][0]
6:      end for
7:      for j ← 1..n do
8:          mejor[0][j] ← mejor[0][j-1] + matriz[0][j]
9:      end for
10:     for i ← 1..n do
11:         for j ← 1..n do
12:             mejor[i][j] ← min(mejor[i-1][j],mejor[i][j-1]) + matriz[i][j]
13:         end for
14:     end for
15:     return mejor[n][m]
16: end function
```

Camino mínimo en una grilla - Top Down

```
1:  matriz ← int[N][M], mejor ← int[N][M]
2:  function CAMINOMINIMO(n, m)
3:      if mejor[n][m] != 1 then
4:          return mejor[n][m]
5:      else if n == 0 and m == 0 then
6:          mejor[n][m] ← matriz[0][0]
7:      else if n == 0 then
8:          mejor[n][m] ← CAMINOMINIMO(n,m-1) + matriz[n][m]
9:      else if m == 0 then
10:         mejor[n][m] ← CAMINOMINIMO(n-1,m) + matriz[n][m]
11:     else
12:         mejor[n][m] ←
    min(CAMINOMINIMO(n-1,m),CAMINOMINIMO(n,m-1)) + matriz[n][m]
13:     end if
14:     return mejor[n][m]
15: end function
```

Camino mínimo en una grilla - Top Down

En este caso nuevamente inicializamos mejor en -1 en todos sus posiciones.

Camino mínimo en una grilla - Top Down

En este caso nuevamente inicializamos mejor en -1 en todos sus posiciones.
OJO! En este caso hay que tener cuidado con el Stack Overflow!

Contenidos

Manos a la obra

Ahora van a tener que resolver un ejercicio usando programación dinámica.

Ejercicio

Hay una varilla de L metros de longitud que queremos hacerle ciertos cortes. Un corte cuesta el largo de la varilla, pero una vez que se cortó la varilla en un punto, cortar los pedazos restantes cuesta el largo de esos pedazos. Por ejemplo, cortar una varilla de 20 centímetros a los 4, 7 y 10 centímetros se puede cortar a los 10 (con costo de 20), luego a los 4 (con costo de 10) y luego a los 7 (con costo de 6) con un costo total de 36. Su tarea es hallar el mínimo costo posible total de hacer todos los cortes.

Manos a la obra

Veamos cómo se resuelve esto con programación dinámica.

Manos a la obra

Veamos cómo se resuelve esto con programación dinámica.

- El algoritmo propuesto tendrá una complejidad de $O(N^3)$ donde N es la cantidad de cortes.

Manos a la obra

Veamos cómo se resuelve esto con programación dinámica.

- El algoritmo propuesto tendrá una complejidad de $O(N^3)$ donde N es la cantidad de cortes.
- Para cada subvarilla posible (con extremos en los extremos de la varilla original o en los cortes) podemos calcular su costo en función de los posibles cortes que le podemos hacer

Manos a la obra

Veamos cómo se resuelve esto con programación dinámica.

- El algoritmo propuesto tendrá una complejidad de $O(N^3)$ donde N es la cantidad de cortes.
- Para cada subvarilla posible (con extremos en los extremos de la varilla original o en los cortes) podemos calcular su costo en función de los posibles cortes que le podemos hacer
- De todos los posibles cortes que le podemos hacer a una varilla nos quedamos con el de costo mínimo

Manos a la obra

Veamos cómo se resuelve esto con programación dinámica.

- El algoritmo propuesto tendrá una complejidad de $O(N^3)$ donde N es la cantidad de cortes.
- Para cada subvarilla posible (con extremos en los extremos de la varilla original o en los cortes) podemos calcular su costo en función de los posibles cortes que le podemos hacer
- De todos los posibles cortes que le podemos hacer a una varilla nos quedamos con el de costo mínimo
- Nuestro caso base es una varilla con extremos en dos cortes consecutivos (considerando a los extremos también como cortes)

Manos a la obra

A trabajar!