Programación Dinámica

Leopoldo Taravilse

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Training Camp 2012

Contenidos

1. Programación Dinámica
   - Recursión
   - Programación Dinámica
   - Principio de Optimalidad
   - Ejemplos
2. Juegos
   - El juego de las piedras
   - Juegos Combinatorios

Contenidos

1. Programación Dinámica
   - Recursión
   - Programación Dinámica
   - Principio de Optimalidad
   - Ejemplos
2. Juegos
   - El juego de las piedras
   - Juegos Combinatorios

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

```
1 | int fibo(int n)
2 | {
3 |     if(n<=1)
4 |         return 1;
5 |     else
6 |         return fibo(n-2)+fibo(n
  |                -1);
7 | }
```

Cálculo Recursivo de Fibonacci

```
1  int fibo(int n)
2  {
3      if(n<=1)
4          return 1;
5      else
6          return fibo(n-2)+fibo(n
               -1);
7  }
```

Notemos que fibo(n) llama a fibo($n-2$), pero después 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 introducir el concepto de programación dinámica

Contenidos

1. Programación Dinámica
   - Recursión
   - Programación Dinámica
   - Principio de Optimalidad
   - Ejemplos
2. Juegos
   - El juego de las piedras
   - Juegos Combinatorios

Programación dinámica

La programación dinámica es una técnica que consiste en:

Programación dinámica

La programación dinámica es una técnica que consiste en:

- Dividir un problema en dos o más subproblemas o reducirlo a una instancia más fácil de calcular del problema.

Programación dinámica

La programación dinámica es una técnica que consiste en:

- Dividir un problema en dos o más subproblemas o reducirlo a una instancia más fácil de calcular del problema.
- Resolver las instancias de cada subproblema de la misma manera (dividiendo en subproblemas o reduciendo a otra instancia) hasta llegar a un caso base.

Programación dinámica

La programación dinámica es una técnica que consiste en:

- Dividir un problema en dos o más subproblemas o reducirlo a una instancia más fácil de calcular del problema.
- Resolver las instancias de cada subproblema de la misma manera (dividiendo en subproblemas o reduciendo a otra instancia) hasta llegar a un caso base.
- Guardar el resultado de cada instancia del problema la primera vez que se calcula, para que cada vez que se vuelva a necesitar el valor de esa instancia ya esté almacenado, y no sea necesario calcularlo nuevamente.

Programación dinámica

La programación dinámica es una técnica que consiste en:

- Dividir un problema en dos o más subproblemas o reducirlo a una instancia más fácil de calcular del problema.
- Resolver las instancias de cada subproblema de la misma manera (dividiendo en subproblemas o reduciendo a otra instancia) hasta llegar a un caso base.
- Guardar el resultado de cada instancia del problema la primera vez que se calcula, para que cada vez que se vuelva a necesitar el valor de esa instancia ya esté almacenado, y no sea necesario calcularlo nuevamente.

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

Cálculo de Fibonacci mediante Programación Dinámica

```
1  int fibo[100];
2  int calcFibo(int n)
3  {
4      if(fibo[n]!=-1)
5          return fibo[n];
6      fibo[n] = calcFibo(n-2)+calcFibo(n-1);
7      return fibo[n];
8  }
9  int main()
10 {
11     for(int i=0;i<100;i++)
12         fibo[i] = -1;
13     fibo[0] = 1;
14     fibo[1] = 1;
15     int fibo50 = calcFibo(50);
16 }
```

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

1. Programación Dinámica
   - Recursión
   - Programación Dinámica
   - Principio de Optimalidad
   - Ejemplos
2. Juegos
   - El juego de las piedras
   - Juegos Combinatorios

Otro tipo de problemas

- Hasta ahora vimos problemas en los que hay que calcular un número de terminado.

Otro tipo de problemas

- Hasta ahora vimos problemas en los que hay que calcular un número de terminado.
- En algunos casos lo que tenemos que calcular es el menor o el mayor número que cumple con cierta propiedad.

Otro tipo de problemas

- Hasta ahora vimos problemas en los que hay que calcular un número de terminado.
- En algunos casos lo que tenemos que calcular es el menor o el mayor número que cumple con cierta propiedad.
- En estos casos para poder usar programación dinámica necesitamos asegurarnos que la solución de los subproblemas es parte de la solución de la instancia original del problema.

Principio de optimalidad

Definición

El principio de optimalidad de Bellman dice que la solución óptima de un subproblema es parte de una solución óptima del problema original.

Principio de optimalidad

Definición

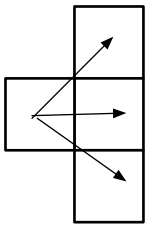
El principio de optimalidad de Bellman dice que la solución óptima de un subproblema es parte de una solución óptima del problema original.

Bellman (quien descubrió la programación dinámica en 1953) afirmó que siempre que se cumple el principio de optimalidad se puede aplicar programación dinámica.

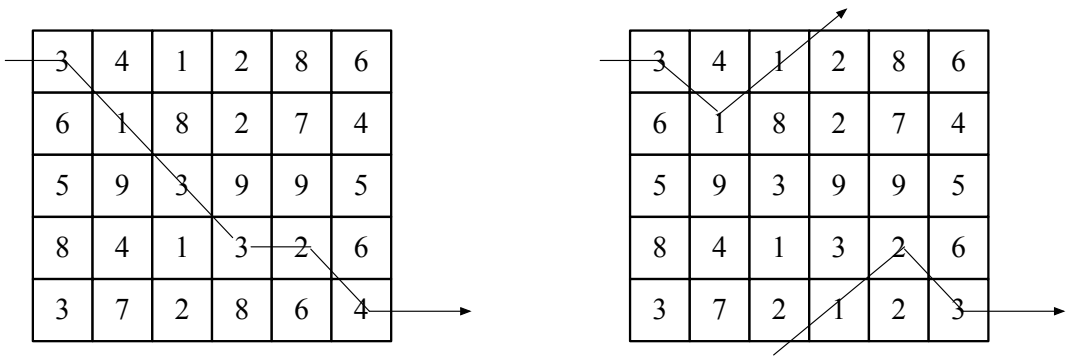
Problema
El camino más corto unidireccional
Archivo: camino (.c, .cpp, .pas)

Problemas que requieren determinar el camino más corto aparecen en diversas áreas de ciencias de la computación. Por ejemplo, una de las restricciones en problema de ruteo en VLSI (*very large scale integration*) es minimizar el largo de los cables. El problema del vendedor viajero, es decir, indicar si un vendedor viajero puede realizar una ruta que visite una sola vez todas las ciudades con un límite de tiempo especificado, es uno de los problemas clásicos de las ciencias de la computación. El problema que se presenta ahora consiste en encontrar un camino mínimo que pase por los puntos de una grilla mientras se viaja sólo de izquierda a derecha.

Dada una m x n matriz de enteros, se le pide escribir un programa que determine el camino con peso mínimo. Un camino comienza en cualquier fila de la primera columna (columna 1), y consiste en una secuencia de pasos terminando en la última columna (columna n). Un paso consiste en pasar de una columna i a una columna i+1 en una fila adyacente (horizontal o diagonal). La primera y última filas (filas 1 y m) de la matriz son consideradas adyacentes, i.e., la matriz se cierra de manera que representa un cilindro horizontal.



El peso de un camino es la suma de enteros en cada una de las n celdas de la matriz que son visitadas. Por ejemplo, una pequeña variación da caminos diferentes como muestra la figura siguiente.



El camino mínimo es ilustrado para cada matriz. Note que el camino para la matriz a la derecha toma ventaja de la propiedad de adyacencia de la primera y última fila.

Entrada

La entrada consiste de una secuencia de especificación de matriz. Cada especificación consiste de las dimensiones de filas y columnas, seguidas por m x n enteros donde m es el número de filas y n es el número de columnas. Los enteros aparecen en la entrada en el orden de mayor orden de filas, es decir, los primeros n enteros son de la primera fila de la matriz, luego los siguientes n enteros son de la segunda fila, y sucesivamente. Los enteros en una línea serán separados de otros enteros por uno o más espacios. Notar que los enteros pueden ser negativos. Puede haber más de una especificación de matriz en un archivo de entrada. La entrada termina con el fin de archivo. Por cada especificación, los número de fila van de 1 a 10 inclusive, los números de columnas van de 1 a 100 inclusivo. Ninguno de los pesos del camino excederá valores de enteros representados usando 30 bits.

Salida

Dos líneas deberían ser la salida por cada especificación de matrices en el archivo de entrada, la primera representa un camino de peso mínimo, y la segunda línea es el costo del camino mínimo. El camino consiste de una secuencia de n enteros (separados por uno o más espacios) representando las filas que constituye el camino mínimo. Si hay más de un camino de peso mínimo, el camino que sea el menor lexicográficamente será el escogido.

Puntuación

30 puntos. Se probarán un número de casos con matrices de una dimensión menor a 5 x 5.
30 puntos. Se probarán un número de casos con matrices de una dimensión menor a 10 x 10.
40 puntos. Se probarán un número de casos con matrices de una dimensión menor a 10 x 100.

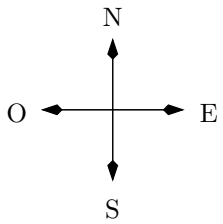
Entrada	Salida para la entrada de ejemplo
5 6 3 4 1 2 8 6 6 1 8 2 7 4 5 9 3 9 9 5 8 4 1 3 2 6 3 7 2 8 6 4 5 6 3 4 1 2 8 6 6 1 8 2 7 4 5 9 3 9 9 5 8 4 1 3 2 6 3 7 2 1 2 3 2 2 9 10 9 10	1 2 3 4 4 5 16 1 2 1 5 4 5 11 1 1 1 9

Problema D
Montaña

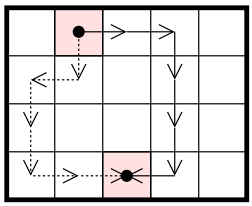
archivo: `montana{.c,.cpp,.pas}`

Ema y Camilo disfrutan de pasear por las montañas, sin embargo, la cordillera tiene tantos cerros distintos que generalmente para moverse de un punto a otro deben subir y bajar repetidamente. Ellos están cansados de esta situación y les gustaría encontrar un camino que los lleve de un punto a otro tal que, o bien sólo suba, o bien sólo baje. También pueden mantenerse por ciertos tramos a la misma altura, ya que esto realmente no les molesta, pero si en su camino alguna vez tuvieron que bajar, quieren nunca tener que volver a subir. Similarmente, si en el camino tuvieron que subir, quieren nunca tener que volver a bajar. Camilo está seguro que siempre podrá encontrar un camino que cumpla sus requisitos, pero Ema no está tan segura. Ellos se han enterado de tu participación en las olimpiadas de informática y les gustaría tener tu ayuda para encontrar tal camino (si es que existe).

Ema y Camilo tienen un mapa que indica la altura del terreno en cada punto. Estos puntos forman una grilla equiespaciada en el terreno. Para evitar perderse Ema y Camilo quisieran que les entregaras instrucciones muy simples. Debes indicarles cómo moverse desde el punto de partida hasta el final. Los posibles movimientos son (N)orte, (S)ur, (E)ste y (O)este (como se indica a la izquierda de la figura). Obviamente, Ema y Camilo quieren recorrer lo menos posible, por lo que tu programa deberá ser capaz de encontrar la ruta con la menor cantidad de movimeintos. En la figura (centro) se muestra un ejemplo de un posible mapa del terreno con indicaciones de alturas. El punto de partida es el destacado en la parte superior y el de llegada el destacado en la parte inferior. El dibujo en la derecha muestra dos rutas posibles con el mínimo número de pasos, 6 en este caso, y son caminos de subida.



7	10	15	15	16
14	12	20	18	17
14	8	11	19	15
14	16	20	19	8



La ruta de flechas punteadas en el dibujo estaría dada por los movimientos SOSSEE, y la ruta de flechas sólidas estaría dada por los movimientos EESSSO.

Entrada

Cada caso de prueba contiene en la primera linea 7 números enteros, W, H, x_{ini}, y_{ini}, x_{fin}, y_{fin}, C, todos separados por un espacio en blanco. W y H corresponden al ancho y alto del mapa, respectivamente, $(x_{\text{ini}}, y_{\text{ini}})$ son las coordenadas del punto de partida, $(x_{\text{fin}}, y_{\text{fin}})$ las coordenadas del punto final, y C el tipo de consulta que se hará. Los valores son tales que $1 \leq W \leq 2000$, $1 \leq H \leq 2000$, $0 \leq x_{\text{ini}} < W$, $0 \leq y_{\text{ini}} < H$, $0 \leq x_{\text{fin}} < W$, $0 \leq y_{\text{fin}} < H$, y C es igual a 0 o 1. Las coordenadas en el mapa crecen en el eje x de izquierda a derecha, y en el eje y de arriba hacia abajo, ambas partiendo desde 0.

Las siguientes H lineas de la entrada contienen cada una W enteros separados por un espacio, correspondientes a la altura en cada punto del mapa.