Olimpiada Informática Española
Soluciones de los problemas

2010

¡Caballos! [Dinámica]

Problema: Dado un tablero de $n \times m$ con k caballos de ajedrez, calcular de cuantos modos se puede llegar a cada casilla con en exactamente t movimientos.

Solución: La solución pasa por hacer una dinámica, los estados serían dp[i][j][h] que representaría de cuantas maneras se puede llegar a la posición (i, j) (fila i, columna j) con exactamente h movimientos. La recursión sería:

$$dp[i][j][h] = \sum_{(i',j'):|i-i'|+|j-j'|=3, i' \neq i, j' \neq j} dp[i'][j'][h-1]$$

es decir, la suma de las maneras en que se puede llegar a casillas a distancia de caballo, en 1 movimiento menos. La complejidad del algoritmo será de $O(nmt)$

Dados [Ad-hoc]

Problema: Dadas dos situaciones, decir cual es más probable o si son equiprobables. Cada situación consiste en tirar $k_i \leq 2$ dados y augurar que la suma de los dados será mayor/menor/igual a un cierto número.

Solución: Sabiendo que con un dado la probabilidad de sacar cualquier número del 1 al 6 es 1/6 y con dos dados la probabilidad de sacar un número entre el 2 y el 12 es $\frac{6-|x-7|}{36}$, contemplar los ditintos casos de operaciones.

DFA [Implementación]

Problema: Dado un autómata y un conjunto de palabras decir cuales acepta y cuales no. Un autómata es una serie de instrucciones diseñadas para aceptar palabras de cierto tipo. Se puede entender como un grafo en el que, dependiendo de las letras que recibas, te moverás hacia un nodo u otro. Se acepta la palabra si al acabar de recorrerla te encuentras en el último nodo del grafo.

Solución: Simplemente iteramos por las letras de las palabras, siguiendo las instrucciones del autómata.

Expresiones algebraicas [Recursividad]

Problema: Dado una expresión algebraica con el máximo número de paréntesis (1 por cada operación), calcular su resultado.

Solución: La idea detrás de la solución consiste en tener una función capaz de calcular expresiones entre paréntesis. En cuanto la función encuentre un nuevo paréntesis dentro del suyo se llamará a sí misma para calcular esa nueva expresión.

Sistemas-L [Implementación]

Problema: Dada una palabra inicial y una serie de reglas que transforman una letra en una palabra. Aplicar t veces esas transformaciones sobre la palabra original. Devolviendo una serie de instrucciones, cada una de las cuales está asociada a una letra de la palabra una vez transformada.

Solución: Simplemente hacemos la simulación, es decir, sustituimos las letras de la palabra por sus transformaciones t veces.

Sistema-L (2) [Binary Search]

Problema: El mismo que el anterior, pero esta vez t puede ser mucho mayor, y se pide solo las instrucciones asociadas a las letras de la palabra final entre la posición l y r.

Solución: La idea está en transformar, en cada una de las t palabras que iremos obteniendo, únicamente aquellas partes de la palabra que darán lugar a letras entre la posición l y r. Para ello haremos una búsqueda binaria en cada una de las t palabras, donde necesitaremos una función que nos diga cuantas palabras de la palabra final corresponderán a el trozo de palabra que queda a la izquierda. Para calcular esa cantidad, lo óptimo es tener un contador por cada letra, y realizar las iteraciones que falten actualizándolo. La complejidad sería de $O(t^2 \log t)$.

Muralla china [Binary Search, Greedy]

Problema: Dadas $N+1$ torres situadas en una muralla, y $k < N+1$ batallones a distribuir entre las torres, minimizar la máxima distancia entre un punto de la muralla y la torre con batallón más cercana.

Solución: La solución esperada consiste en realizar una búsqueda binaria sobre la mínima distancia y comprobar si es posible lograr una distribución con esta distancia mediante un algoritmo voraz: iterando por las torres en orden, y colocando un batallón cuando sea estrictamente necesario.

OMG [Grafos]

Problema: Dado un conjunto de personas con las horas a partir de las cuales se despiertan, y para cada una de ellas, su conjunto de amigos, hallar el tiempo que tardará la persona n en recibir un mensaje de la persona 1 si en cuanto alguien recibe un mensaje y está despierto se lo envía a todos sus amigos.

Solución: El problema se puede reformular como: hallar la mínima distancia en un grafo ponderado, donde los nodos son las personas y los pesos de los arcos son el máximo valor entre las horas a las que se despiertan los nodos a los que el arco conecta. Por lo tanto la solución esperado es un dijkstra clásico.

Palabras crecientes [Hashing, Grafos]

Problema: Dada una palabra inicial, una palabra objetivo, y una serie de reglas que permiten transformar trozos de la palabra original en otras palabras, encontrar el mínimo número de pasos para llegar a la palabra objetivo desde la inicial, o decir que no es posible. Se garantiza que, de existir ese mínimo número de pasos, este será menor a 10. Además todas las palabras están formadas por las letras *a* y/o *b*.

Solución: Podemos entender el problema como un grafo, en el que las palabras son nodos y con algunas transformaciones nos podemos mover entre nodos. Por lo que podemos aplicar un BFS sobre las palabras, para hallar la mínima "distancia" entre la palabra inicial y destino. Además, para optimizar las comparaciones entre palabras y las transformaciones es buena idea pasar las palabras a números, entendiendo que estas son números en base 2 (*a* y *b* como 0 y 1).

Partido [Implementación]

Problema: Dado los goles que ha marcado cada partido, indicar el resultado final.

Solución: Simplemente contamos los goles de cada equipo.

Producto vectorial [Implementación]

Problema: Dados dos vectores de 3 coordenadas, calcular su producto vectorial.

Solución: Realizamos los cálculos que se indican en el enunciado.

RoGN1 [Dinámica]

Problema: Dada una cuadrícula con valores $v_{i,j}$ y la posición inicial de un robot que únicamente se puede mover hacia de derecha, arriba y abajo, calcular la máxima suma que puede obtener el robot, si cada vez que pasa por una casilla se queda con su valor. La única restricción es que el robot no puede pasar por casillas con un valor $> k$.

Solución: La solución consiste en realizar una dinámica, donde los estados son dp[x][y] que representa la máxima suma empezando por la posición (x, y). La recursión sería:

$$dp[i][j] = \sum_{i' \in I} v_{i',j} + \max_{i' \in I} dp[i'][j+1]$$

donde I representa las filas de la j-ésima columna accesibles desde (i, j).

Salta-salta [Implementación]

Problema: Se da un vector con n números y empezando en la posición 0, en cada turno se mira el número de la posición en la que te encuentras y avanzas tantas casillas. Se considera que el vector es circular (después de la última posición vuelve a venir la primera). Decir en que posición se acabará tras t turnos.

Solución: Simplemente simulamos el proceso, en cada iteración i nos vamos a la posición $p_{i+1} = (p_i + V[p_i])$ módulo n.

Suffix array [Recursividad, Aleatoriedad]

Problema: Dada una palabra construye su suffix array. Un suffix array es un vector de valores, de modo que v_i representa la posición de la letra por la que empieza el sufijo que estaría en la posición i-ésima si ordenásemos todos los sufijos. Los tests son aleatorios.

Solución: Hay diversas maneras de resolver este problema, pero se nos hace entender que podemos explotar el hecho de que los tests son aleatorios. Una posible solución consiste en tener una función recursiva que dada una lista de posiciones por las que empiezan los sufijos nos devuelve su suffix array. En caso de que para una letra solo haya un sufijo que empiece por ella, está claro que posición ocupará. Para los demás sufijos avanzamos una posición y llamamos a la función. El algoritmo es $O(n^2)$ pero, al ser aleatorio, no habrá muchos sufijos con un mismo prefijo largo. Por ejemplo, la probabilidad de que dos sufijos tengan un prefijo de 20 letras iguales $< 10^{-22}\%$. Por lo tanto, a efectos prácticos será $O(n)$.

Programando el vídeo [Greedy]

Problema: Dado un conjunto de intervalos (i_j, f_j) donde cada uno empieza en i_j y finaliza en f_j, calcular el número máximo de intervalos que se pueden seleccionar de modo que no haya solapamientos.

Solución: Una solución sería realizar un algoritmo voraz. Podríamos ordenar los intervalos por orden de finalización e ir seleccionándolos por orden, siempre y cuando no generen solapamiento con alguno escogido previamente.