Meet in the middle

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos avanzados
Tecnológico de Monterrey

pperezm@tec.mx

12-2021

Contenido

Meet in the middle

- La técnica de "Encontrarse en el medio" (Meet in the middle) divide el espacio de búsqueda en dos partes de aproximadamente el mismo tamaño, realiza una búsqueda para ambas partes y, finalmente, combina los resultados de las búsquedas.
- Esta técnica nos permite acelerar los resultados de las búsquedas de tiempo $O(2^n)$ para que funcionen en un tiempo $O(2^{\frac{1}{2}})$
- Usando un algoritmo $O(2^n)$, el tamaño máximo que podríamos procesar (por el tiempo de ejecución) es de $n = 20$, mientras que un algoritmo $O(2^{\frac{1}{2}})$ nos permite procesar entradas cercanas a $n = 40$.

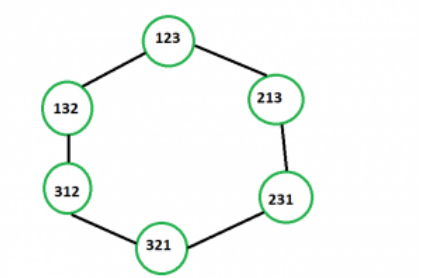
Shortest Path using Meet In The Middle

Data una permutación $P = \{p_1, p_2, \cdots p_n\}$ de los primeros ne números naturales $(1 \leq n \leq 10)$, encontrar el mínimo número de intercambios para pasar de la permutación P a $P = \{p'_1, p'_2, \cdots p'_n\}$. Solo se pueden intercambiar dos elementos consecutivos, p_i y p_{i+1} $(1 \leq i \leq 10)$.

Este problema se puede resolver utilizando el algoritmo Dijkstra. Parece que no hay nada relacionado con un grafo en la declaración. Pero supongamos que una permutación es un vértice, entonces cada intercambio de los elementos de una permutación es un borde que conecta este vértice con otro vértice. Por lo tanto, encontrar el número mínimo de intercambios ahora se convierte en un simple problema de BFS/ruta más corta.

Input: P = "213", P' = "231"

Output: 2



Ahora analicemos la complejidad de tiempo. Tenemos $n!$ vértices, cada vértices tiene $n-1$ vértices adyacentes. Tomando en cuenta esto, la complejidad que tendríamos al aplicar el algoritmo sería $O(n \log n * n!)$. Usando al técnica de "Encontrarse en la mitad" puede hacer que la solución sea más rápida.

Haremos P y P' sean ambos el inicio de nuestra búsqueda. Realizaremos el BFS desde estos puntos, comenzamos y terminamos al mismo tiempo usando solo una fila.

1. Sea *start* el vértice inicial y *finish* el vértice final.
2. $visited[start] = true$ y $visited[finish] = true$.
3. $src[start] = start$ y $src[end] = end$
4. $dist[start] =$ y $dist[start] = 0$
5. Colocamos *start* y *finish* en la fila, marcándolos como visitados.
6. Mientras la fila no está vacía. sacamos el primer elemento. u. Colocamos en al fila todos los elementos adyacentes, v, de u que hayan visitados. Si algún v ya fue visitado, si $src[u] \neq srv[v]$ entonces regresamos $dist[u] + dist[v] + 1$.

Ver código bfs_middle.cpp

El problema de la suma de subconjuntos

Retomemos el problema de la suma de subconjuntos:

- Dado un conjunto S de N números enteros positivos, encontrar los subconjuntos que sumen la cantidad C. Si no se encuentra algún subconjunto que cumple, se debe responder con el conjunto vacío.

La idea es dividir nuestro conjunto en dos conjuntos A y B, de modo que ambos conjuntos contengan aproximadamente la mitad de los números. Realizamos dos búsquedas: la primera búsqueda genera todos los subconjuntos de A y almacena sus sumas en una lista S_A, y la segunda búsqueda crea una lista S_B similar para B. A continuación, basta con verificar si podemos elegir un elemento de S_A y otro de S_B, tal que la suma sea C.

Por ejemplo, veamos cómo se procesa el conjunto $S = \{2, 3, 7, 9\}$. Primero, dividimos el conjunto en conjuntos $A = \{2, 3\}$ y $B = \{7, 9\}$. En seguida, creamos las listas $S_A = [\{\} = 0, \{2\} = 2, \{3\} = 3, \{2, 3\} = 5]$ y $S_B = [\{\} = 0, \{7\} = 7, \{9\} = 9, \{7, 9\} = 16]$. Dada que S_A contiene la suma 2 y S_B contiene la suma 7, concluimos que el conjunto original tiene un subconjunto con suma $2 + 7 = 9$.

Con una buena implementación, podemos crear las listas S_A y S_B en tiempo $O(2^{\frac{n}{2}})$, de la tal forma que las listas estén ordenadas. Después, podemos usar la técnica de dos punteros para verificar en tiempo $O(2^{\frac{n}{2}})$ si la suma C puede ser creada a partir de S_A y S_B. Por tanto, la complejidad del algoritmo es $O(2^{\frac{n}{2}})$.