Algoritmos sobre grafos

Víctor Racsó Galván Oyola

19 de mayo de 2021

Índice

Breadth-First Search (BFS)

Concepto

Idea

Propaguemos información por niveles.

Concepto

Idea

Propaguemos información por niveles.

Uso básico

El uso más común de BFS es para hallar el camino más corto desde un nodo s hacia todos los que sean alcanzables por el mismo en un grafo no ponderado (todas las aristas tienen peso 1).

Concepto

Idea

Propaguemos información por niveles.

Uso básico

El uso más común de BFS es para hallar el camino más corto desde un nodo s hacia todos los que sean alcanzables por el mismo en un grafo no ponderado (todas las aristas tienen peso 1).

Otras aplicaciones

- Verificar si un grafo es bipartito.
- Hallar el camino más corto desde un nodo s hacia todos los demás en un grafo cuyas aristas tienen pesos en el conjunto $\{0, x\}$ y $x > 0$.
- Hallar el diámetro de un árbol. El diámetro es el camino más largo entre algún par de nodos del árbol.
- Hallar el *centro* de un árbol. El *centro* de un árbol es algún nodo u tal que el camino más largo desde u hasta algún nodo v se minimice.
- Hallar el orden topológico de un grafo G acíclico.

Intentemos problemas en el SPOJ - I

Ada and Cycle

Se nos da la matriz de adyacencia M de un grafo y se nos pide determinar el ciclo de menor longitud o determinar que este no existe.

Intentemos problemas en el SPOJ - I

Ada and Cycle

Se nos da la matriz de adyacencia M de un grafo y se nos pide determinar el ciclo de menor longitud o determinar que este no existe.

Solución

Podemos probar mandar un BFS desde cada nodo u, considerando que el primer nivel serán los vecinos inmediatos de u con distancia igual a 1 (no asignamos 0 a la distancia de u).
La complejidad será $O(n^3)$ pero en práctica es mucho más rápido.

Intentemos problemas en el SPOJ - I

Ada and Cycle

Se nos da la matriz de adyacencia M de un grafo y se nos pide determinar el ciclo de menor longitud o determinar que este no existe.

Solución

Podemos probar mandar un BFS desde cada nodo u, considerando que el primer nivel serán los vecinos inmediatos de u con distancia igual a 1 (no asignamos 0 a la distancia de u).
La complejidad será $O(n^3)$ pero en práctica es mucho más rápido.

Inversion Sort

Determinar la menor cantidad de movimientos que uno necesita para transformar una permutación p a una permutación p' de las letras de la 'a' a la 'j'. Un movimiento se define como invertir el orden de un subarreglo de la permutación actual. Son múltiples casos.

Intentemos problemas en el SPOJ - II

Solución I

Aplicamos BFS desde la permutación p hasta encontrar a p' por cada caso.
La complejidad será de $O(q \cdot |p|! \cdot |p|^3)$, donde q es la cantidad de casos. Veredicto: TLE.

Intentemos problemas en el SPOJ - II

Solución I

Aplicamos BFS desde la permutación p hasta encontrar a p' por cada caso.
La complejidad será de $O(q \cdot |p|! \cdot |p|^3)$, donde q es la cantidad de casos. Veredicto: TLE.

Solución II

Podemos considerar que la permutación p y p' son nada más que etiquetas de posiciones, así que podemos mapear la permutación p a la permutación identidad con una función σ y nos bastará con hallar la distancia desde la permutación identidad hasta $\sigma(p')$.
Esto nos permite hallar solo una vez las distancias desde la permutación identidad a todas las permutaciones 1 sola vez y luego responder a los casos en $O(|p|)$.
La complejidad será de $O(|p|! \cdot |p|^3 + q \cdot |p|^2)$. Veredicto: AC.

Intentemos problemas en el SPOJ - II

Solución I

Aplicamos BFS desde la permutación p hasta encontrar a p' por cada caso.
La complejidad será de $O(q \cdot |p|! \cdot |p|^3)$, donde q es la cantidad de casos. Veredicto: TLE.

Solución II

Podemos considerar que la permutación p y p' son nada más que etiquetas de posiciones, así que podemos mapear la permutación p a la permutación identidad con una función σ y nos bastará con hallar la distancia desde la permutación identidad hasta $\sigma(p')$.
Esto nos permite hallar solo una vez las distancias desde la permutación identidad a todas las permutaciones 1 sola vez y luego responder a los casos en $O(|p|)$.
La complejidad será de $O(|p|! \cdot |p|^3 + q \cdot |p|^2)$. Veredicto: AC.

Bonus

¿Cómo podríamos hallar el orden topológico de un grafo G o determinar que no existe usando BFS?

Intenemos un problema en Codeforces - I

Dijkstra?

Dado un grafo ponderado y no dirigido de n nodos, hallar la longitud del camino más corto desde el nodo 1 al n.

Intenemos un problema en Codeforces - I

Dijkstra?

Dado un grafo ponderado y no dirigido de n nodos, hallar la longitud del camino más corto desde el nodo 1 al n.

Problemática

Esta vez ya no funciona aplicar BFS, ¿O si?

Intenemos un problema en Codeforces - I

Dijkstra?

Dado un grafo ponderado y no dirigido de n nodos, hallar la longitud del camino más corto desde el nodo 1 al n.

Problemática

Esta vez ya no funciona aplicar BFS, ¿O si?

Modificación del BFS - Algoritmo de Dijkstra

Consideremos ahora que el nivel i ya no contiene a los nodos a una distancia i, sino a una distancia que es la i-ésima menor distancia de todas las posibles (podemos simular esto con una cola de prioridades), entonces podremos propagar la menor distancia desde el nodo s hasta el nodo v pasando por el nodo u con una complejidad de $O(E \log E)$.

Depth-First Search (DFS)

Concepto

Idea

En vez de ir por niveles, vayamos lo más profundo que podamos antes de no tener más nodos nuevos que visitar y regresar (backtrack).

Concepto

Idea

En vez de ir por niveles, vayamos lo más profundo que podamos antes de no tener más nodos nuevos que visitar y regresar (backtrack).

Uso básico

El uso principal de DFS es para analizar la estructura del grafo, propone una clasificación de aristas que nos permite hallar propiedades interesantes.

Concepto

Idea

En vez de ir por niveles, vayamos lo más profundo que podamos antes de no tener más nodos nuevos que visitar y regresar (backtrack).

Uso básico

El uso principal de DFS es para analizar la estructura del grafo, propone una clasificación de aristas que nos permite hallar propiedades interesantes.

Otras aplicaciones

- Hallar las componentes conexas de un grafo.
- Hallar el Euler tour de un árbol.
- Determinar si un grafo posee un ciclo.
- Hallar los puentes/articulaciones de un árbol.
- Hallar el orden topológico de un grafo G acíclico.
- Hallar las componentes fuertemente conexas de un grafo G.

Clasificación de aristas - I

DFS Tree

Cuando lanzamos un DFS sobre algún grafo, considerando que estamos en un nodo u y v es un vecino directo de u no visitado, podemos agregar la arista (u, v) a un nuevo grafo T. Este nuevo grafo será un árbol llamado *DFS Tree*.

Clasificación de aristas - I

DFS Tree

Cuando lanzamos un DFS sobre algún grafo, considerando que estamos en un nodo u y v es un vecino directo de u no visitado, podemos agregar la arista (u, v) a un nuevo grafo T. Este nuevo grafo será un árbol llamado *DFS Tree*.

Clasificación de aristas

Las aristas se pueden clasificar de la siguiente forma:

- Tree edge: Arista que forma parte del DFS Tree.
- Back edge: Arista que va desde un nodo v hacia un ancestro u que todavía no haya terminado de ser procesado.
- Forward edge: Arista que va desde un nodo u hacia un descendiente v que ya haya sido procesado anteriormente.
- Cross edge: Arista que va desde un nodo u hacia un nodo v que no es descendiente y ya haya sido procesado anteriormente.

Clasificación de aristas - II

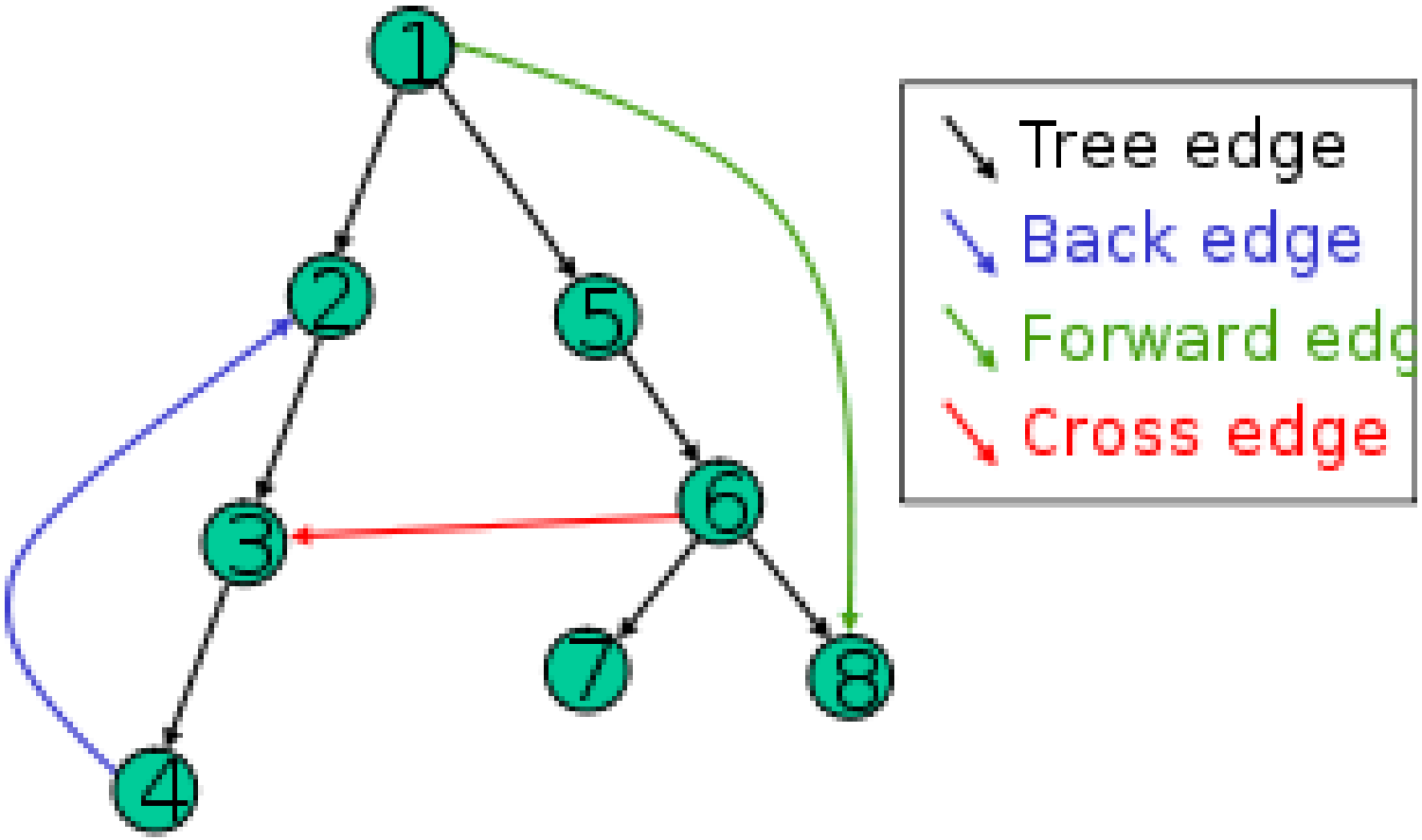

Figura: Ejemplo de clasificación de aristas

Intentemos un problema en SPOJ

ABC Path

Dada una matriz con letras mayúsculas, determinar el camino más largo de letras consecutivas que empiecen en la letra 'A'.

Intentemos un problema en SPOJ

ABC Path

Dada una matriz con letras mayúsculas, determinar el camino más largo de letras consecutivas que empiecen en la letra 'A'.

Solución I

Usamos backtracking para hallar la máxima longitud, al restaurar las celdas a sus estados sin visitar tendremos una complejidad muy grande.
La complejidad será de $O(3^{|\Sigma|})$, donde Σ es el alfabeto.

Intentemos un problema en SPOJ

ABC Path

Dada una matriz con letras mayúsculas, determinar el camino más largo de letras consecutivas que empiecen en la letra 'A'.

Solución I

Usamos backtracking para hallar la máxima longitud, al restaurar las celdas a sus estados sin visitar tendremos una complejidad muy grande.
La complejidad será de $O(3^{|\Sigma|})$, donde Σ es el alfabeto.

Solución II

Notamos que, si hemos procesado el camino más largo de letras consecutivas desde una celda, no es necesario volver a procesar una celda más de 2 veces, así que no debemos restaurar el estado de las celdas luego de procesarlas con un DFS. Al final tomaremos la respuesta máxima de entre todas las celdas que tengan una 'A'.
La complejidad será de $O(nm)$.

Intentemos un problema en Codeforces - I

Kefa and Park

Dado un árbol con raíz en el nodo 1, tenemos algunos nodos que contienen gatos. Nos piden hallar la cantidad de hojas tales que el camino desde el nodo 1 a ellas tiene a lo mucho m gatos consecutivos.

Intentemos un problema en Codeforces - I

Kefa and Park

Dado un árbol con raíz en el nodo 1, tenemos algunos nodos que contienen gatos. Nos piden hallar la cantidad de hojas tales que el camino desde el nodo 1 a ellas tiene a lo mucho m gatos consecutivos.

Solución

Mandamos un DFS desde el nodo 1 y llevamos cuántos gatos consecutivos se tiene hasta ahora, si en algún momento tenemos más de m consecutivos, cualquier hoja desde el nodo actual no será válida y deberemos devolver 0. En caso contrario, la respuesta es la suma de las respuestas de los hijos directos con el contador modificado.
La complejidad será de $O(n)$.

Intentemos otro problema en Codeforces - I

Propagating Tree

Dado un árbol de n nodos con raíz en el nodo 1, cada nodo i tiene inicialmente un valor de a_i y tenemos q consultas de 2 posibles tipos:

- Actualizar el nodo u agregándole x, luego a sus hijos directos $-x$, luego a los hijos de los hijos x y así hasta que ya no haya nada más.
- Consultar el valor actual del nodo u.

Intentemos otro problema en Codeforces - I

Propagating Tree

Dado un árbol de n nodos con raíz en el nodo 1, cada nodo i tiene inicialmente un valor de a_i y tenemos q consultas de 2 posibles tipos:

- Actualizar el nodo u agregándole x, luego a sus hijos directos $-x$, luego a los hijos de los hijos x y así hasta que ya no haya nada más.
- Consultar el valor actual del nodo u.

Solución I

Hacer una actualización con un DFS por cada consulta.
Complejidad $O(nq)$

Intentemos otro problema en Codeforces - I

Propagating Tree

Dado un árbol de n nodos con raíz en el nodo 1, cada nodo i tiene inicialmente un valor de a_i y tenemos q consultas de 2 posibles tipos:

- Actualizar el nodo u agregándole x, luego a sus hijos directos $-x$, luego a los hijos de los hijos x y así hasta que ya no haya nada más.
- Consultar el valor actual del nodo u.

Solución I

Hacer una actualización con un DFS por cada consulta.
Complejidad $O(nq)$

DFS Timestamps / Euler tour

Si modificamos un poco el DFS para que nos diga en qué momento llegamos a un nodo y en qué momento terminamos de procesarlo en dos arreglos in y out, debemos notar que los nodos en el rango $[in[u], out[u]]$ pertenecen al subárbol de u.

Intentemos otro problema en Codeforces - II

Pista I

¿Y si separamos los nodos por los que tienen altura par e impar?

Intentemos otro problema en Codeforces - II

Pista I

¿Y si separamos los nodos por los que tienen altura par e impar?

Pista II

¿Si usamos una estructura de datos para modificar los valores?

Intentemos otro problema en Codeforces - II

Pista I

¿Y si separamos los nodos por los que tienen altura par e impar?

Pista II

¿Si usamos una estructura de datos para modificar los valores?

Solución II

Podemos notar que a los nodos que tengan una altura con la misma paridad que u se les va a sumar x y a los que no se les restará x. Además, usando los DFS Timestamps podemos modificar solo los nodos del subárbol de u.
Por último, si mantenemos en un Fenwick tree la **variación** del valor de cada nodo, la respuesta por cada uno sería $a_u + query(u)$. Para modificar las posiciones en el rango $[l, r]$ con el mismo valor x y luego consultar su resultado puntual podemos sumar x a la posición l y restar x a la posición $r + 1$ para compensar la variación.
La complejidad será de $O(n + q \log n)$.