Schematics
2019

Editorial Penyisihan NPC Senior

NPC

National
Programming
Contest

Problem A – Angka Keren

Tag : Adhoc

Diberitahukan bahwa akan ada N angka. Dari N angka tersebut akan dicari semua kemungkinan pasangan. Jadi misalkan ada 3 angka, maka ada kemungkinan $3C2$ pasangan. Dari pasangan – pasangan yang ada, akan dilakukan operasi XOR, lalu untuk setiap hasil XOR akan di OR. Dari test case, jumlah N terbanyak adalah 10^6. Dengan ini akan ada $1000000C2$ pasangan yang akan terlalu lama jika dikerjakan dengan tanpa ada optimisasi.

Untuk menyelesaikan soal ini, yang harus kita lakukan adalah mencari cara untuk menyederhanakan operasi XOR dan OR-nya. Ini dapat diselesaikan dengan observasi atau menyederhanakan dengan aljabar Boolean biasa.

https://en.wikipedia.org/wiki/Boolean_algebra

Catatan :

- ‘ berarti operator not
- $+$ berarti operator or
- $\times$ berarti operator and (jadi AB berarti A AND B)

Operasi XOR dari A XOR B dapat ditulis lagi dengan $(A'B + AB')$

Misalkan ada 3 angka, yaitu A, B, dan C. jadi yang dicari adalah $(A\ xor\ B) + (A\ xor\ C) + (B\ xor\ C)$

Dapat ditulis juga dengan, $(A'B + AB' + A'C + AC' + B'C + BC')$ yang dapat kita tambahkan dengan AA’, BB’ dan CC’ sesuai dengan sifat operasi OR.

Jadi operasi ini akan menjadi $(A'A + A'B + A'C + B'A + B'B + B'C + C'A + C'B + C'C)$

Sesuai dengan sifat distribusi dapat ditulis lagi dengan

$$(A' + B' + C')\,(A + B + C)$$

Dengan ini, operasi Angka keren dapat dilakukan dengan kompleksitas $O(n)$ dan bukan $O(n^2)$.

Sifat – sifat diatas juga berlaku jika angka melebihi 3, dapat dibuktikan sendiri.

Problem B – Teka-Teki Nih

Tag : Number Theory, Math

Problem ini merupakan problem untuk melakakukan dekripsi pada enkripsi RSA (Rivest-Shamir-Adleman). Pada awalnya dilakukan enkripsi dengan $c = x^e mod\ n$ untuk $x < n$ dan x koprima terhadap $phi(n)$. Untuk mengembalikan nilai x, diperlukan nilai d yang merupakan private key sehingga menjadi

$$x = c^d\ mod\ n$$

Untuk mencari nilai d dapat menggunakan modular multiplicative inverse dari $e(mod\ phi(n))$ dengan menggunakan Extended Euclidean Algorithm. Setelah didapat nilai d, dilakukan modular exponentiation.

Pembuktiannya sebagai berikut.

$$x = c^d\ mod\ n$$

$$x = (x^e\ mod\ n)^d\ mod\ n$$

Menurut Modular Arithmetic Exponentiation Law, untuk $a < m$, maka

$$a^k\ mod\ m = (a\ mod\ m)^k\ mod\ m$$

karena $x^e\ mod\ n$ kurang dari n maka

$$x = x^{ed}\ mod\ n$$

Dimana dari Euler's Theorem, $ed = phi(n) \cdot k + 1$ dan jika a bilangan bulat positif yang koprima terhadap n, maka $a^{phi(m)\cdot k}\ mod\ m = 1$. Sehingga

$$x = x^{phi(n)\cdot k + 1}\ mod\ n$$

$$x = x\left(x^{phi(n)\cdot k}\ mod\ n\right)$$

Problem C – Envisi Property

Tag : Advanced Data Structure, Segment-Tree

Problem ini menitik beratkan *query* pada sebuah data yang statis. Karakteristik *query* yang diberikan berupa *range*. Jika dilihat, batasan jumlah *query* (Q) pada problem adalah $1 \leq Q \leq 10000$, yang artinya apabila diterapkan metode naif (*bruteforce*) tidak akan efisien untuk *query* yang sangat besar. Untuk itu, disini memerlukan struktur data yang menunjang *query* tersebut.

Kita dapat menggunakan struktur data *Segment Tree* untuk menyimpan *state* tiap ruangan. Tiap-tiap node pada *segment tree* menyimpan informasi berupa jumlahan state ruangan (1 untuk lampu hidup dan 0 untuk lampu mati) pada *range* tertentu.

Struktur Penyimpanan State Ruangan

Namun, sebelum membangun *segment tree*, kita lihat terlebih dahulu struktur penyimpanan untuk ruangan. Karena ruangan terpisah menjadi N lantai, yang mana masing-masing lantai terdapat M ruangan, struktur data yang dapat digunakan untuk menyimpan *state* tiap ruangan adalah menggunakan *array* dua dimensi. Misalkan *array* yang digunakan adalah $bit[k][i]$, maka $bit[k][i]$ menandakan *state* lampu pada ruangan i pada lantai ke-k.

Untuk mempermudah proses *query*, kita dapat mengkonversi struktur *array* dua dimensi menjadi *array* satu dimensi saja. Lantai k ruangan ke-i dapat dikonversi menjadi :

$$x = M\,(k-1) + (i-1) \quad \text{..... (1)}$$

	i				
k	1	2	3	4	5
	1	2	3	4	5
	1	2	3	4	5

x														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Membangun *Segment Tree*

Langkah selanjutnya adalah proses untuk membangun *segment tree* dari data yang sudah tersedia. Representasi yang digunakan untuk menyimpan informasi tiap node adalah struktur *array* satu dimensi. Ukuran *array*-nya adalah $node[4 \times N \times M]$. Proses pembangunan *segment tree* menggunakan rekursi.

```
function buildSegmentTree(V, left, right):
      if left == right then
            node[V] = bit[left]
            return

      mid = (left + right)/2
      buildSegmentTree(V*2, left, mid)
      buildSegmentTree(V*2 + 1, mid + 1, right)

      node[V] = node[V*2] + node[V*2 + 1]
```

Dapat dilihat bahwa $node[V]$ mempunyai *child node* pada $node[2V]$ dan $node[2V + 1]$, dan $node[V]$ merupakan jumlah dari $node[2V]$ dan $node[2V + 1]$.

Proses *Query*

- **Query "ON"**

  Untuk menghitung jumlah lampu yang hidup, maka dapat dilihat dari *root segment tree*, atau dengan kata lain nilai dari $node[1]$.

- **Query "OFF"**

  Jumlah lampu yang hidup dapat dihitung dari $total\ lampu - lampu\ hidup$. Total lampunya pasti berjumlah $N * M$, sedangkan lampu yang hidup dapat dihitung dari *root segment tree*. Maka, jumlah lampu yang mati adalah $N * M - node[1]$.

- **Query "TOGGLE $k\ i$"**

  "TOGGLE $k\ i$" sama saja dengan "TOGGLE x", dimana nilai x adalah hasil konversi menggunakan rumus (1). Untuk memperbarui *state* pada $bit[x]$, maka perbaruan juga perlu dilakukan pada nilai *node* yang bersesuaian. Metode yang digunakan adalah menggunakan rekursi.

```
function toggle(x, newValue, V, left, right):
	if x < left or x > right then
		return
	if left == right
		node[V] = newValue
		return

	mid = (left + right)/2
	// menuju left child apabila berada pada range
	if left <= x and x <= right
		toggle(x, newValue, V*2, left, mid)
	// jika tidak, menuju right child
	else
		toggle(x, newValue, V*2 + 1, mid + 1, right)

	node[NodeV] = node[NodeV*2] + node[NodeV*2 + 1]
```

- **Query "COUNT $k\ i\ j$"**

  Misalkan x adalah hasil konversi untuk lantai k ruang i dan y adalah hasil konversi untuk lantai k ruang j, maka *query* "COUNT $k\ i\ j$" dapat disamakan dengan "COUNT $x\ y$", yang artinya menghitung jumlah lampu yang hidup dari x hingga y.

```
function count(x, y, V, left, right):
	if x > right or y < left then
		return 0
	if x <= left and y >= right then
		return node[V]

	mid = (left + right)/2
	return (
		count(x, y, V*2, left, mid) +
		count(x, y, V*2 +1, mid + 1, right)
	)
```

- **Query "FIND x"**

```
function find(x, V, left, right):
	if x > node[V] then
		return -1
	if left == right then
		return left

	mid = (left + right)/2
	if node[V*2] >= x
		return find(x, V*2, left, mid)
	else
		return find(x - node[V*2], V*2 + 1, mid + 1, right)
```

Fungsi di atas akan mengembalikan nilai berupa posisi (indeks) pada *array* satu dimensi yang menyimpan *state* lampu. Untuk mengkonversinya menjadi lantai k ruang i, maka :

Misalkan ans adalah hasil dari fungsi di atas :

- $k = ans/M$
- $i = ans - (k * M)$

Problem D – Masyarakat Berkabut

Tag : Graph

Persoalan ini dapat diselesaikan menggunakan minimum spanning tree. Permasalahannya adalah jumlah vertexnya sebanyak 10^5 sehingga edge nya bisa jadi $(10^5)^2$, hal tersebut tidak bisa disimpan dalam array.

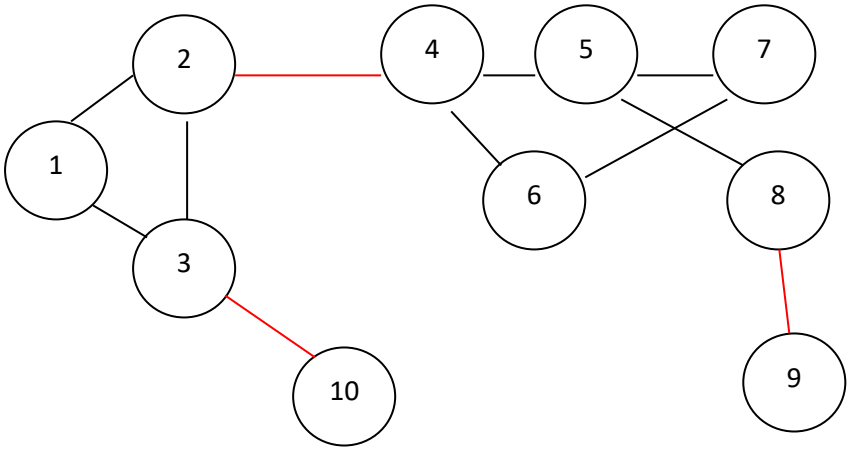
Observasinya adalah dengan membuat path yang dibutuhkan saja. Yaitu minimum terhadap x-axis ataupun y-axis. Sehingga untuk mengkonstruksi grafnya, kita hanya perlu menyimpan semua sumbu x lalu mengurutkannya dan menyimpan sumbu-y lalu mengurutkannya juga. Sehingga graf yang dikonstruksi hanya sebanyak $2N$. untuk edge nya, karena x dan y sudah terurut maka kita hanya perlu menghubungkan vertex yang berurutan pada himpunan x-axis dan y-axis yang telah terurut tadi.

Total Kompleksitas: $O(N\ log\ N\ +\ \alpha(N))$

Problem E – Budak Cinta

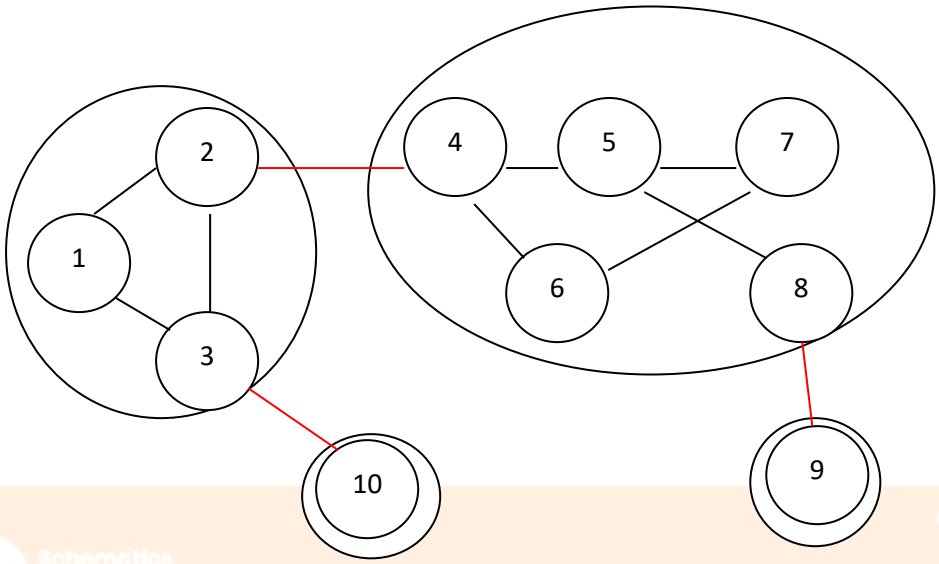
Tag : Graph, Data Structure

Perhatikan bahwa “Jalan Spesial” yang dimaksud di soal adalah *bridge* dari graf yang diberikan. Dengan begitu, kita dapat memodifikasi graf yang diberikan dengan menganggap semua *edge* yang bukan *bridge* memiliki *weight* 0 . Kita dapat menggunakan algoritma Tarjan agar dapat mencari bridge dalam kompleksitas $O(N + M)$. Dengan begitu,permasalahan kita sekarang adalah mencari *shortest path* dari suatu node ke node lainnya. Akan tetapi, dengan melakukan *dijkstra* untuk setiap *query*, akan menyebabkan TLE. Untuk menghasilkan kompleksitas yang lebih effisien, kita membutuhkan observasi tambahan.



Bridge adalah *edge* yang berwarna merah.

Perhatikan bahwa graf yang diberikan sebenarnya merupakan beberapa *Connected Component* yang dihubungkan oleh *bridge*



Vertex-vertex yang dilingkari berada di dalam 1 *Connected Component* yang sama.

Ini artinya bahwa untuk mencari *Shortest Path* dari suatu node X ke node Y, kita hanya perlu mencari *Shortest Path* dari *Connected Component* yang memuat node X ke *Connected Component* yang memuat node Y.

Perhatikan lagi bahwa, *Connected Components* tersebut membentuk sebuah *tree*. Jadi, permasalahan kita sekarang tinggal mencari jarak antara 2 node di dalam sebuah *tree* yang mana bisa dilakukan dengan struktur data *Heavy-light Decomposition* ataupun *Sparse Table* dengan kompleksitas $O(lg\ N)$ per query.

Kompleksitas mencari bridge : $O(N + M)$

Kompleksitas per query : $O(lg\ N)$

Total kompleksitas : $O((N + M) + (Q\ lg\ N))$

Problem F – Berlarian di Penjara

Tag : Dynamic Programming, Graph

Problem pada soal ini adalah kita harus mencari apakah Baney dapat keluar dari penjara dengan waktu yang telah ditentukan. Penyelesaian problem ini dilakukan dengan mencari waktu tempuh minimal dan maksimal dari tiap wilayah yang ada. Ketika Baney berada di suatu wilayah dan Baney bisa berhasil keluar dari semua rute maka Baney pasti lolos ("Lolos gan"). Jika ada beberapa rute saja yang membuat Baney bisa keluar, maka Baney memiliki kemungkinan berhasil. Namun jika semua rute tidak memungkinkan maka, Baney tidak dapat keluar dari penjara.

Untuk mencari waktu minimal dan maksimal, dilakukan dp pada tree. Mula-mula dilakukan dfs dimulai pada salah satu node untuk menyimpan waktu minimum dan maksimum sementara. Kemudian untuk mencari nilai maksimum bisa dilakukan dfs kembali sedangkan waktu minimum diperoleh dengan melakukan bfs dengan melakukan update pada waktu minimum dan maksimum yang telah diperoleh sebelumnya.

Problem G – Luas Tanah

Tag : Computational Geometry

Problem luas tanah ini meminta anda untuk menentukan luas minimum tanah yang mencakup semua tanaman yang mempunyai bidang berbentuk convex, maka langkah pertama yang harus dilakukan adalah dengan menentukan tanaman mana saja yang dipakai untuk membentuk bidang berbentuk convex dengan menggunakan algoritma convex hull, setelah dilakukan convex hull, lakukan shoelace untuk mendapatkan luas tanah tersebut, permasalahan kecil yang lain adalah input tidak hanya berupa integer. Untuk pencarian convex hull kompleksitas waktu nya adalah $O(n \log n)$ dimana n merupakan jumlah semua tanaman dan untuk shoelace kompleksitas waktu nya adalah $O(m)$ dimana m merupakan jumlah tanaman yang terpakai sebagai convex hull maka kompleksitas waktu totalnya adalah $O(n \log n + m)$.