COMPFEST 9

Pembahasan Coder Class

SCPC - Minggu 4

COMPFEST 9

Daftar Soal

A. Baskom Lagi Baskom Lagi
B. Aglio Carbonara
C. Count ST
D. Pusing-Pusing Poligon
E. Bukan Convex Hull
F. Subpohon Ganda
G. Captain Badak
H. Pohon Kelereng

A. Baskom Lagi Baskom Lagi

Tag

ad hoc

Pembahasan

Pertama, untuk mempermudah kita dapat menambahkan nilai M ke $A[1]$ sehingga $A[1]$ yang baru bernilai $A[1] + M$. Selanjutnya, kita dapat mendefinisikan sum(i) sebagai jumlah $A[j]$ untuk $1 \leq j < i$.. Karena Pak Chanek akan mengambil air sebanyak i kali setelah anak ke-i mengisi air maka diperlukan $X * i \geq sum(i)$ untuk membuat bak mandi kosong setelah anak ke-i mengisi air, sehingga $X \geq \frac{sum(i)}{i}$, kita sebut nilai $\frac{sum(i)}{i}$ ini sebagai $f(i)$. Maka kita memerlukan $X = f(k)$ untuk menyelesaikan soal ini, namun jika terdapat $f(i)$ $(1 \leq i < k)$ yang lebih kecil dari $f(k)$ maka Pak Chanek tidak dapat selesai mandi setelah anak ke-K mengisi air karena akan selesai duluan setelah anak ke-i tersebut mengisi bak.

Kompleksitas: $O(N)$.

B. Aglio Carbonara

Tag

Ad hoc

Pembahasan

Untuk mengerjakan soal ini, kita cukup mensimulasikan cara Pak Walikota yaitu dengan menemukan huruf paling awal dari kiri dan dari kanan yang merupakan huruf vokal.

Kompleksitas: $O(N)$

C. Count ST

Tag

Depth-first-search

Pembahasan

Apabila diamati, dalam pembuatan *spanning tree*, untuk setiap *cycle*, kita harus membuang salah satu edge pada *cycle* tersebut. Tentunya, untuk suatu *cycle* berukuran C, maka ada C cara memilih edge yang bisa dibuang. Mengingat pembuangan edge antar *cycle* bersifat independen, banyak cara adalah dengan mengalikan cara antar *cycle*. Misal, *cycle-cycle* yang terbentuk adalah $C_1, C_2, \ldots, C_k$. Maka, banyak caranya adalah:

$$\prod_{i=1}^{k} |C_i|$$

Sekarang, yang menjadi pertanyaan adalah, bagaimana cara mencari ukuran tiap *cycle* dengan efisien?

Kita bisa membuat sebuah *rooted tree* dengan menggunakan *depth-first search*, yang biasa disebut DFS *spanning tree*. Bagaimana cara memanfaatkan *tree* ini? Awalnya, kita ambil vertex sembarang sebagai *root*. Lalu, jalankan DFS, dan catat kedalaman tiap vertex (anggap *root* memiliki kedalaman 0). Observasinya, setiap edge yang terdapat dalam graf awal, namun tidak terdapat dalam *tree*, pasti merupakan *back edge*, yang termasuk dalam tepat satu *cycle*.

Observasi selanjutnya, panjang dari *cycle* tersebut pasti merupakan selisih antara kedalaman 2 vertex yang edge tersebut hubungkan ditambah 1. Maka, kita bisa mendapatkan seluruh ukuran *cycle* dengan cepat.

Kompleksitas: $O(N + M)$

D. Pusing-Pusing Poligon

Tag

Geometri

Pembahasan

Soal ini dapat diselesaikan dengan menggunakan sebuah observasi penting. Yaitu, apabila sebuah poligon reguler diputar dengan periode $N \cdot K$, maka irisan poligon reguler tersebut akan menjadi poligon reguler dengan jumlah sisi sebanyak $N \cdot K$. Ini dapat dibuktikan dengan melihat sebagian rotasi. Apabila poligon tersebut telah berputar selama K detik, maka poligon tersebut akan tepat bertumpukan sempurna dengan poligon awalnya, sehingga setiap sisi poligon akan terbagi menjadi K bagian yang sama panjang dan memiliki sudut yang sama. Karena poligon awal memiliki N sisi, maka jumlah sisi akhir adalah $N \cdot K$ sisi.

Untuk menghitung luas sebuah poligon reguler dapat menggunakan rumus yang dapat Anda cari sendiri dengan geometri dasar serta trigonometri.

Kompleksitas: $O(1)$

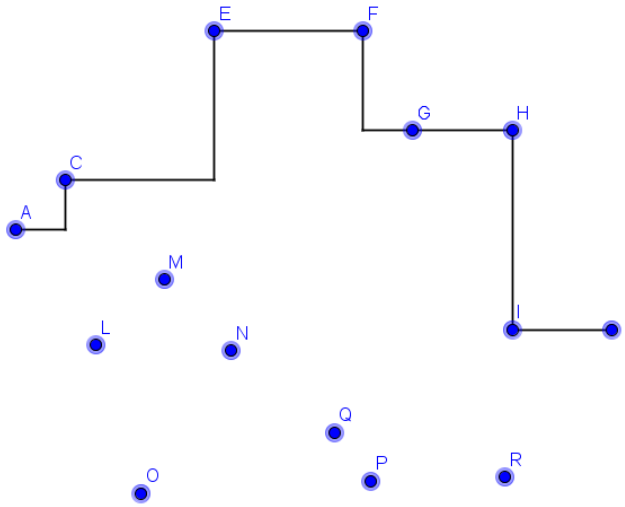
E. Bukan Convex Hull

Tag

Line sweep

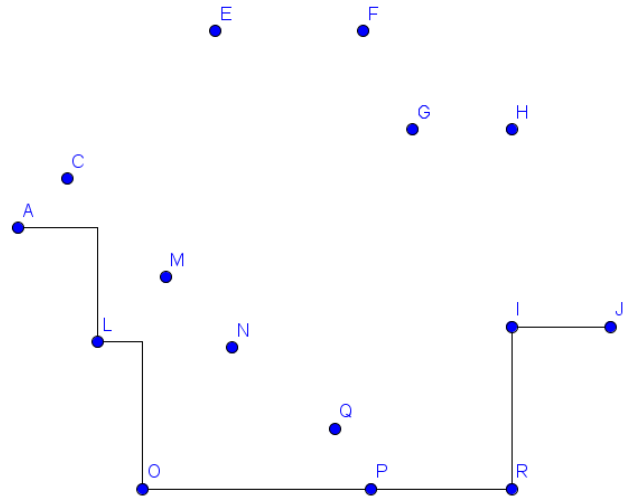
Pembahasan

Pertama, kita memulai dengan membuat *hull* bagian atas. *Hull* ini dimaksudkan untuk membentuk batas atas dari poligon yang akan kita buat. Perhatikan bahwa *hull* bagian atas yang kita buat akan menyerupai gunung (menaik, kemudian turun). *Hull* ini dapat dibuat dengan melakukan *line sweep*. Cara yang digunakan mirip dengan algoritma *Graham's scan* untuk mencari *convex hull*. Mula-mula kita urutkan semua titik berdasarkan nilai x, seandainya seri, berdasarkan y yang lebih kecil. *Hull* disini direpresentasikan sebagai sebuah *stack*. Setiap kali kita mencoba memasukan sebuah titik kedalam *hull* kita, mula-mula kita periksa apakah bila titik ini kita masukan, *hull* kita tetap valid (berbentuk gunung, yaitu naik, kemudian turun). Apabila *hull* kita tetap valid, maka masukkan titik tersebut. Apabila tidak (misalnya mengakibatkan naik, turun, lalu naik), maka kita akan melakukan *pop* pada *hull*, dan lakukan hingga *hull* yang kita miliki valid dan kita dapat memasukan titik tersebut.



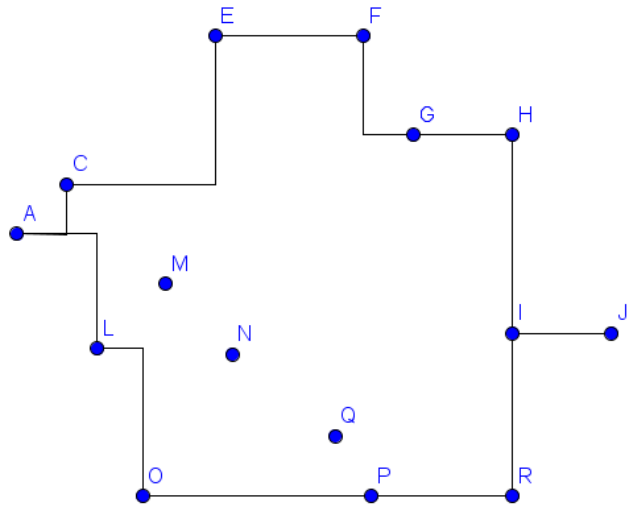
contoh hull bagian atas

Setelah itu, kita akan membuat *hull* bagian bawah. *Hull* ini dimaksudkan untuk membentuk batas bawah dari poligon yang kita buat. Perhatikan bahwa hull bagian bawah yang kita buat akan menyerupai lembah (menurun, kemudian naik). *Hull* ini dapat dibuat dengan cara yang sama dengan *hull* bagian atas, hanya saja dibalik.



contoh hull bagian bawah

Setelah kita mendapatkan *hull* bawah dan *hull* atasnya, kita dapat menggabungkan kedua *hull* tersebut dan membentuk sebuah poligon yang memenuhi syarat dan dengan luas minimal.



Contoh penggabungan hull bagian atas dan bawah

Perhatikan bahwa saat kita membuat hull bagian bawah, kita juga harus memperhatikan garis yang dibuat saat membuat hull bagian atas agar tidak terjadi garis yang saling berpotongan.

Kompleksitas: $O(N \log N)$

F. Subpohon Ganda

Tag

Hashing

Pembahasan

Pertama, kita dapat memodelkan tree ke dalam bentuk string, sebuah subpohon pada tree tersebut akan dibatasi oleh tanda kurung. Maka jika kita mendefinisikan $S(x)$ sebagai representasi String dari subpohon yang berakar di x dan $child(x)$ sebagai vertex yang merupakan keturunan langsung dari x, kita dapat mendefinisikan rumus $S(x)$:

$$S(x) = '(' + S(child(x)1) + S(child(x)2) + \ldots + ')'$$

Selanjutnya kita dapat melakukan hash seolah hash pada String. Misalkan h sebagai fungsi hash tersebut, kita dapat mengurutkan $child(x)$ berdasarkan nilai $h(child(x))$ sebelum memodelkannya menjadi $S(x)$. Kenapa kita harus mengurutkan child berdasarkan $h(child(x))$? Untuk 2 subpohon yang sama namun memiliki urutan child berbeda (perhatikan contoh kasus pada soal).

Kompleksitas: $O(N)$

G. Captain Badak

Tag

Dynamic programming

Pembahasan

Pertama, kita dapat menentukan rumus DP *straightforward* untuk soal ini, nilai maksimum yang dapat kita peroleh pada baris i kolom j yaitu

$$dp(U,X)\begin{cases}0 & i > N \\ x[i][j] + dp(i+1,j) & baris\ i\ tidak\ memiliki\ teleport \\ x[i][j] + \max(dp(i+1,j), \max(dp(i+y,z) 1 \le y \le K, 1 \le z \le N, z \ne j)) & selain\ itu\end{cases}$$

Dengan rumus di atas maka permasalahan kita adalah kita perlu iterasi $O(NK)$ untuk baris yang memiliki teleport sehingga total kompleksitas DP kita menjadi $O(N^3K)$. Kita dapat membuatnya lebih optimal dengan beberapa observasi. Pertama, perhatikan bahwa untuk setiap baris yang memiliki teleport dan kemungkinan banyak baris yang dilangkahi $(1 \le y \le K)$ kita tentu ingin mengambil maksimal dari semua kolom j pada $dp(i+y,j)$. Hanya kolom z yang memiliki $dp(i+y,z)$ sebagai nilai maksimal yang tidak dapat mengambil nilai maksimal tersebut maka pada kolom z kita dapat mengambil nilai kedua maksimal. Kita dapat menyimpan nilai maksimal dan nilai kedua maksimal di setiap baris sehingga kita sudah berhasil optimasi iterasi kita menjadi hanya $O(K)$ dan kompleksitas total DP $O(N^2K)$.

Selanjutnya, untuk menghilangkan iterasi $O(K)$, kita dapat melakukannya dengan mengerjakan DP secara bottom-up dan bantuan struktur data *deque*. Kita memerlukan *deque* untuk setiap kolom, deque ini akan menyimpan maksimal K buah nilai berisi nilai maksimal pertama/kedua sesuai baris dan kolom sesuai penjelasan sebelumnya dan baris nilai tersebut berada. Kita misalkan val sebagai nilai optimal tersebut dan pos sebagai baris nilai tersebut berada. Selanjutnya kita juga memisalkan front sebagai isi *deque* yang paling depan, back sebagai isi deque yang paling belakang, dan current sebagai nilai yang akan kita isi. Jika kita mengisi dari depan, maka apabila $current.val \geq front.val$ dan $current.pos < now.pos$ maka kita dapat pop front karena current lebih optimal dalam val maupun pos. Kita akan terus mengisi *deque* dengan cara seperti ini, bagaimana dengan yang akan kita ambil untuk mengisi $dp(i,j)$? Kita hanya perlu mengambil $back.val$ karena isi *deque* paling belakang pasti paling optimal dari nilai val namun kita perlu pop juga back yang nilai posnya lebih dari jarak maksimal K.

Kompleksitas: $O(N^2)$

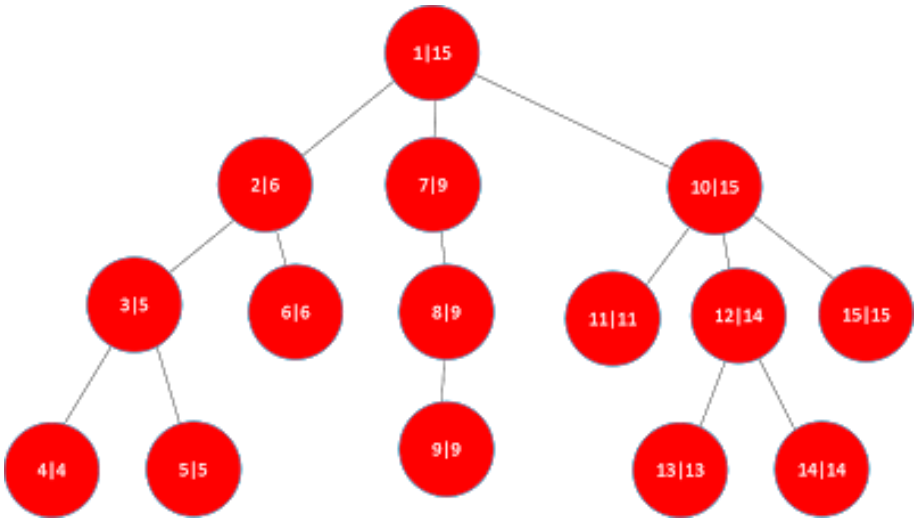
H. Pohon Kelereng

Tag

Data structure, lowest common ancestor

Pembahasan

Pertama, karena operasi kedua adalah query subtree maka kita dapat mengerjakannya dengan pre-order numbering dan query menggunakan struktur data. Kesulitan soal ini terletak pada operasi pertama yang mentoggle verteks dan jika kita ingin operasi nomor 2 kita tetap query subtree biasa maka kita perlu menyesuaikan operasi nomor 1 kita.

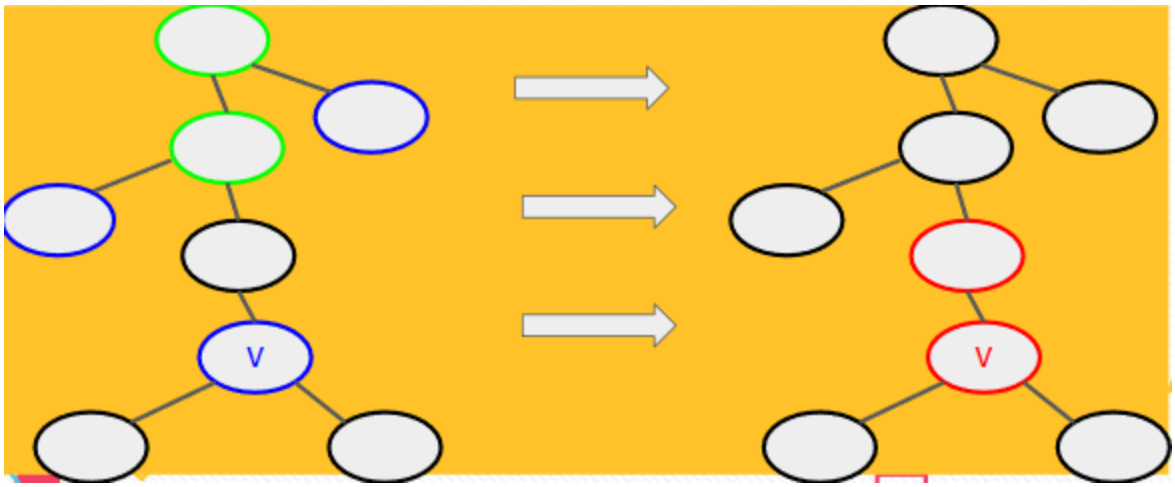


pre-order numbering

Salah satu cara yang dapat digunakan adalah mengubah operasi nomor 1 menjadi path update. Untuk mempermudah, akan dibahas update yang menyalakan lampu terlebih dahulu. Perhatikan bahwa kita dapat mengupdate path dari root ke verteks v jika belum ada verteks lain yang menyala dan memiliki nilai yang sama dengan nilai verteks v. Update path dari root ke verteks v bisa dilakukan dengan *fenwick tree* atau *heavy light decomposition*.

Apabila terdapat satu verteks lain yang sudah menyala dan memiliki nilai yang sama dengan nilai verteks v, maka kita cukup update path dari *lowest common ancestor*(LCA) kedua verteks hingga verteks v yang bisa kita lakukan dengan update path dari root ke v dan update path dari root ke LCA dengan nilai negatif dari nilai update.

Apabila terdapat lebih dari satu verteks lain yang menyala dan memiliki nilai yang sama dengan nilai verteks v, maka kita harus mencari LCA dari masing-masing verteks tersebut dengan verteks v yang paling bawah. Kita dapat melakukannya dengan hanya mencari yang paling bawah dari 2 LCA yaitu LCA 2 buah verteks yang urutan pre-order numberingnya tepat di sebelah kiri dan kanan verteks v. Untuk mendapatkan 2 verteks ini kita dapat menyimpan verteks yang menyala pada set sehingga dapat diperoleh dalam $O(\log N)$. Selanjutnya cukup update path dari LCA terbawah ke-v dengan cara sebelumnya.



verteks biru bernilai sama dengan nilai verteks v, hijau merupakan LCAnya dan merah adalah verteks yang diupdate

Untuk update yang bersifat mematikan lampu pada verteks, cukup lakukan cara yang sama dengan di atas dengan update yang mengurangi nilai dan mengeluarkan verteks dari set verteks menyala. Bacaan tambahan: link .

Kompleksitas: $O(N \log N)$