Pembahasan OSN Informatika 2019

Gunawan, Jonathan Irvin
jonathanirvingunawan@gmail.com

Djonatan, Prabowo
prabowo1048576@gmail.com

Ramli, Inigo
igoramli.igo@gmail.com

Sulami, Kezia
keziasulami.pooh@gmail.com

Lie, Maximilianus Maria Kolbe
maxhaibara97@gmail.com

Hendry, Reynaldo Wijaya
wijayareynaldo@gmail.com

July 4, 2019

Di bawah ini adalah solusi resmi yang digunakan oleh Scientific Committee OSN Informatika 2019. Perhatikan bahwa mungkin saja terdapat lebih dari satu solusi untuk beberapa subsoal. Perhatikan juga bahwa subsoal pada diskusi ini mungkin saja tidak terurut untuk kemudahan diskusi.

Karena tujuan utama editorial ini adalah untuk memberikan ide umum untuk menyelesaikan setiap subsoal, kami melewatkan beberapa detil (termasuk detil implementasi) pada diskusi ini untuk latihan para pembaca.

1 1A. Mengumpulkan Peserta

Soal ini ditulis oleh Jonathan Irvin Gunawan dan Lie Maximilianus Maria Kolbe.

Definisi 1.1. *K adalah banyaknya peserta pada ruang lomba.*

1.1 Subsoal 3

Subsoal ini dapat diselesaikan dengan mencoba penempatan peserta pada seluruh kemungkinan persegi panjang. Hanya akan ada tepat satu ukuran persegi panjang yang valid yaitu persegi panjang dengan ukuran $1 \times K$.

Misalkan persegi panjang yang akan dipilih terletak pada petak $(1, c_1)$ hingga $(1, c_2)$. Maka untuk menghitung banyak pemindahan minimal peserta ke dalam persegi panjang yang dipilih adalah banyak peserta dikurang banyak peserta yang sudah berada di dalam persegi panjang tersebut. Untuk menghitung banyak peserta pada persegi panjang yang dipilih, cukup iterasikan semua nilai dari petak $(1, c_1)$ hingga $(1, c_2)$.

Banyak kemungkinan persegi panjang yang dapat dibentuk adalah sebanyak $O(C)$ dan banyak iterasi yang dibutuhkan untuk menghitung banyak peserta pada persegi panjang adalah $O(C)$ untuk setiap kemungkinan persegi panjang.

Kompleksitas dari solusi ini adalah $O(C^2)$.

1.2 Subsoal 4

Menggunakan observasi yang sama dengan subsoal sebelumnya, kita dapat mengoptimasi cara menghitung banyak peserta pada suatu persegi panjang dengan menggunakan *prefix sum*.

Definisi 1.2. *$prefix_i$ adalah banyak peserta dari petak $(1, 1)$ hingga $(1, i)$.*

Untuk menghitung banyak peserta menggunakan *prefix sum* dari petak $(1, c_1)$ hingga petak $(1, c_2)$ adalah dengan menghitung nilai $prefix_{c_1} - prefix_{c_2}$.

Kompleksitas dari solusi ini adalah $O(C)$.

1.3 Subsoal 5

Subsoal ini dapat diselesaikan dengan mencoba semua kemungkinan ukuran persegi panjang serta mencoba seluruh penempatan persegi panjang tersebut. Menggunakan observasi yang sama dengan subsoal 3 kita dapat menghitung pemindahan minimal untuk setiap kemungkinan nilai.

Banyaknya kemungkinan persegi panjang adalah $O(RC)$ sedangkan banyaknya penempatan suatu persegi panjang adalah $O(RC)$ dan iterasi yang dibutuhkan untuk menghitung peserta pada suatu persegi persegi panjang adalah $O(RC)$.

Kompleksitas akhir solusi ini adalah $O((RC)^3)$.

1.4 Subsoal 6

Menggunakan cara yang sama dengan subsoal sebelumnya, kita dapat mengoptimasi cara menghitung banyak peserta pada suatu persegi panjang dengan menggunakan *prefix sum 2D*.

Definisi 1.3. *$prefix2_{i,j}$ adalah banyak peserta dari petak $(1, 1)$ hingga (i, j).*

Untuk menghitung banyak peserta menggunakan *prefix sum 2D* dari petak (r_1, c_1) hingga (r_2, c_2) adalah dengan menghitung nilai $prefix2_{r_2,c_2} - prefix2_{r_2,c_1-1} - prefix2_{r_1-1,c_2} + prefix2_{r_1-1,c_1-1}$.

Kompleksitas akhir solusi ini adalah $O((RC)^2)$.

1.5 Subsoal 7

Menggunakan cara yang sama dengan subsoal sebelumnya, kita dapat mengoptimasi perhitungan banyak ukuran persegi panjang. Persegi panjang yang valid dengan ukuran $A \times B$ harus memenuhi persamaan $A \times B = K$ atau dalam kata lain A habis membagi K. Banyaknya bilangan asli yang habis membagi K tidak akan lebih dari $\sqrt{K}$.

Kompleksitas akhir solusi ini adalah $O((RC)^{1.5})$

2 1B. Pertahanan Manado

Soal ini ditulis oleh Ashar Fuadi.

2.1 Subsoal 3

Subsoal ini dapat diselesaikan dengan mencoba semua konfigurasi pembelian paket obat yang berbeda. Terdapat 2 kemungkinan untuk setiap paket obat: membeli, atau tidak membeli. Sehingga untuk M jenis paket obat, terdapat 2^M konfigurasi pembelian paket obat yang berbeda. Untuk mengiterasi semua kemungkinan tersebut dapat menggunakan teknik *bitmask*. Pada suatu konfigurasi, obat-obat yang belum dibeli melalui paket obat yang dipilih harus dibeli secara satuan dengan harga S. Keluaran dari subsoal ini dapat diperoleh dengan mencari harga termurah dari semua konfigurasi pembelian paket obat yang ada. Kompleksitas dari solusi ini adalah $O(2^M N)$.

2.2 Subsoal 6

Subsoal ini dapat diselesaikan dengan pendekatan *greedy*.

Definisi 2.1. *$cakupan_i$ adalah bilangan terbesar yang memenuhi properti sebagai berikut: terdapat paket obat ke-j sedemikian sehingga $A[j] = i$ dan $B[j] = cakupan_i$.*

Untuk mempermudah implementasi, kita dapat menganggap satuan obat sebagai sebuah paket obat dengan $A[i] = B[i]$. Kemudian kita dapat melakukan komputasi awal untuk mencari semua nilai $cakupan_i$. Pertama, kita iterasi semua paket obat yang ada. Untuk paket obat ke-j, perbarui nilai $cakupan_{A[j]}$ dengan $max(cakupan_{A[j]}, B[j])$.

Agar solusi kita memperoleh harga termurah, kita harus meminimalkan jumlah pembelian, baik paket obat maupun satuan obat. Dengan kata lain, kita melakukan pembelian obat ke-i jika dan hanya jika $cakupan_j < i$ untuk $1 \leq j < i$. Kompleksitas dari solusi ini adalah $O(N)$.

2.3 Subsoal 4

Definisi 2.2. *$f(x)$ adalah fungsi yang mengembalikan harga termurah untuk membeli setidaknya satu dari obat-obat ke-1 hingga ke-x.*

Subsoal ini dapat diselesaikan dengan menggunakan *dynamic programming* untuk menghitung nilai $f(x)$. Untuk suatu *state* dari fungsi f, terdapat 2 opsi yang harus dipertimbangkan:

- Membeli obat ke-x secara satuan. Dengan kata lain, $f(x) = f(x-1) + S$,
- Membeli obat-obat ke-i hingga ke-x, untuk semua nilai i yang mana terdapat sebuah paket obat ke-j sedemikian sehingga $A[j] \leq i \leq x \leq B[j]$. Apabila terdapat lebih dari satu nilai j yang memenuhi, maka pilih $P[j]$ dengan nilai terkecil. Dengan kata lain, $f(x) = f(i-1) + P[j]$.

Sehingga, nilai dari $f(x)$ dapat diperoleh dengan mengambil nilai terkecil dari kedua opsi tersebut. Keluaran dari subsoal ini dapat diperoleh dengan mencari nilai dari $f(N)$. Kompleksitas dari solusi ini adalah $O(N^2 M)$.

2.4 Subsoal 5

Solusi dari subsoal ini merupakan perbaikan dari solusi subsoal 4. Subsoal ini dapat diselesaikan dengan menggunakan *dynamic programming* dengan ide yang sama seperti yang ada pada subsoal 4. Perbaikan tersebut didasarkan pada (Observasi 2.1).

Observasi 2.1. *Nilai dari fungsi $f(x)$ pasti non-decreasing. Dengan kata lain $f(1) \leq f(2) \leq f(3) \leq \cdots \leq f(N)$.*

Berdasarkan (Observasi 2.1), apabila kita membeli paket obat ke-j, kita tidak perlu memeriksa semua nilai i yang memenuhi $A[j] \leq i \leq x$. Hal ini dikarenakan $f(A[j]-1) \leq f(A[j]) \leq f(A[j]+1) \leq \cdots \leq f(x-1)$. Sehingga nilai i yang akan menghasilkan harga yang termurah adalah $i = A[j]$. Oleh karena itu, opsi kedua yang ada pada suatu *state* dapat diubah menjadi:

- Membeli obat-obat ke-$A[j]$ hingga ke-x, untuk paket obat ke-j yang memenuhi $A[j] \leq x \leq B[j]$. Apabila terdapat lebih dari satu nilai j yang memenuhi, maka pilih $P[j]$ dengan nilai terkecil. Dengan kata lain, $f(x) = f(A[j]-1) + P[j]$.

Definisi 2.3. *$H(x)$ merupakan sebuah himpunan yang berisi nomor-nomor paket obat j yang memenuhi $A[j] \leq x \leq B[j]$. Dengan kata lain $H(x) = \{j | 1 \leq j \leq M, A[j] \leq x \leq B[j]\}$.*

Secara formal, nilai dari $f(x)$ dapat ditentukan dengan rumus sebagai berikut:

$$f(x) = min(f(x-1) + S, \min_{j \in H(x)} (f(A[j]-1) + P[j]))$$

Kompleksitas dari solusi subsoal ini adalah $O(NM)$.

2.5 Subsoal 7

Solusi dari subsoal ini merupakan optimasi dari solusi *dynamic programming* untuk subsoal 5. Perhatikan bahwa pada suatu *state* dalam $f(x)$, kita hanya membutuhkan nilai terkecil dari $f(A[j]-1)+P[j]$ untuk semua nilai $j \in H(x)$. Sehingga kita tidak perlu mengiterasi semua anggota dari $H(x)$ selama kita bisa mendapatkan nilai terkecilnya. Permasalahannya adalah, himpunan $H(x)$ akan berubah mengikuti nilai x. Oleh karena itu, dibutuhkan suatu struktur data yang dapat memperbarui isinya sesuai dengan anggota himpunan $H(x)$ dan memperoleh nilai $f(A[j]-1)+P[j]$ terkecil dari himpunan $H(x)$ dengan cepat. Kita dapat menggunakan struktur data *Min-Heap*.

Definisi 2.4. $masuk(i) = \{f(A[j]-1) + P[j] | 1 \leq j \leq M, A[j] = i\}$.

Definisi 2.5. $keluar(i) = \{f(A[j]-1) + P[j] | 1 \leq j \leq M, B[j] = i\}$.

Pada saat kita ingin mencari nilai dari $f(x)$, kita harus memastikan bahwa isi dari *Min-Heap* yang kita punyai memiliki bijeksi dengan $H(x)$. Implementasi dari $f(x)$ akan lebih mudah apabila dilakukan secara *bottom-up*. Sehingga pada saat kita sedang menghitung nilai dari $f(x)$, pastikan semua anggota himpunan $masuk(x)$ sudah ada di dalam *Min-Heap*, dan semua anggota himpunan $keluar(x-1)$ sudah tidak ada di dalam *Min-Heap*. Apabila nilai minimum dari $f(A[j]-1)+P[j])$ untuk $j \in H(x)$ direpresentasikan dalam variabel r (yang berada pada *root Min-Heap*), maka nilai dari $f(x)$ adalah $min(f(x-1)+S, r)$.

Perlu diingat bahwa operasi *pop*() pada *Min-Heap* berjalan dalam kompleksitas $O(logM)$, akan tetapi kita hanya dapat menghapus *root* dari *Min-Heap* tersebut. Sedangkan pada solusi ini, dibutuhkan *Min-Heap* yang dapat menghapus data yang kita inginkan dengan kompleksitas yang sama. Teknik yang dapat kita gunakan untuk mengatasi masalah ini adalah dengan *pending delete*. Secara kasarnya, kita dapat terlebih dahulu menyimpan nilai yang seharusnya sudah dikeluarkan dari *Min-Heap*. Nilai ini akan sebenarnya dikeluarkan pada saat nilai ini berada pada *root Min-Heap*.

Karena setiap data hanya akan masuk dan keluar ke dalam *Min-Heap* sebanyak tepat 1 kali, maka kompleksitas untuk memperbarui *Min-Heap* tersebut secara keseluruhan adalah $O(NlogM)$. Akses *root* pada Min-Heap dapat dilakukan dalam $O(1)$. Kompleksitas akhir dari solusi ini adalah $O(NlogM)$.

3 1C. Rekreasi OSN

Soal ini ditulis oleh Jonathan Irvin Gunawan dan Lie Maximilianus Maria Kolbe. Terdapat 3 solusi yang dapat digunakan untuk menyelesaikan soal ini, ketiganya dapat meletakkan penghalang sebanyak banyaknya penghalang tambahan minimum pada solusi optimal.

3.1 Solusi 1: $C \leq 16$

Soal ini dapat diselesaikan menggunakan dynamic programming dengan bitmask. Mari kita definisikan urutan petak menggunakan urutan *row-major*:

$$(1,1),(1,2),\cdots,(1,C),(2,1),(2,2),\cdots,(2,C),\cdots,(R,1),(R,2),\cdots,(R,C)$$

.

Definisi 3.1. *Untuk kemudahan diskusi, definisikan* $next^0(i,j)$ *sebagai notasi petak setelah petak* (i,j)*, dan* $next^k(i,j) = next^0(next^{k-1}(i,j))$*. Secara serupa, definisikan* $prev^1(i,j)$ *sebagai notasi petak sebelum petak* (i,j)*, dan* $prev^k(i,j) = prev^0(prev^{k-1}(i,j))$*.*

Sebagai contoh, $next^0(1,C) = (2,1), next^1(1,C) = (2,2)$ dan $prev^0(2,2) = (2,1), prev^1(2,2) = (1,C)$.

Untuk setiap petak (dengan urutan yang sudah didefinisikan), kita akan mencoba kedua kemungkinan antara meletakkan penghalang atau membiarkannya kosong. Definisikan $f((i,j), mask)$ sebagai banyaknya penghalang tambahan minimum pada petak $(i,j),(i,j+1),\cdots,(R,C)$ dan $mask$ merupakan bilangan C-bit: bit ke-k menyimpan apakah $prev^k(i,j)$ dapat diraih dari $(1,1)$.

$f((i,j), mask)$ dapat dihitung sebagai berikut:

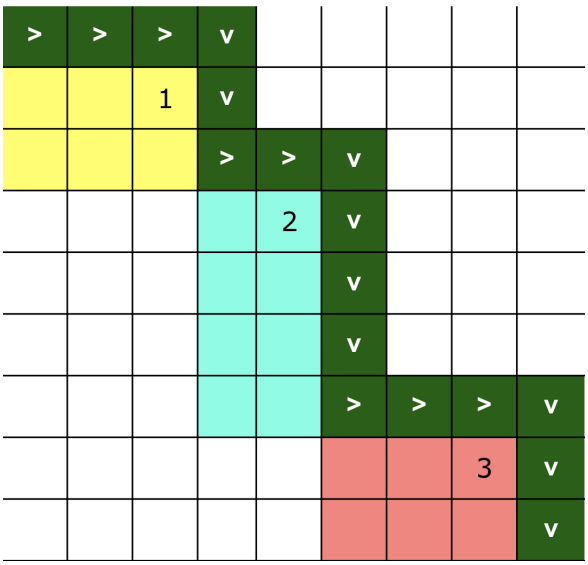
- Anggap $(i',j') = next^0(i,j)$ dan $mask' = (mask << 1)\%2^C$. Dengan kata lain, bit ke-0 pada $mask'$ adalah 0 dan bit ke-k $(1 \leq k < C)$ pada $mask'$ adalah bit ke-$(k+1)$ pada $mask$.
- Jika petak (i,j) dilewati oleh rute yang disiapkan oleh panitia:
  - Jika petak $(i-1,j)$ dan petak $(i,j-1)$ keduanya dapat diraih dari $(1,1)$, maka terdapat lebih dari satu cara untuk meraih petak (i,j). Sehingga, akan terdapat lebih dari satu cari juga untuk meraih petak (R,C). Sehingga, $f((i,j), mask) = \infty$.
  - Jika tidak, maka $f((i,j), mask) = f((i',j'), mask' + 1)$.
- Jika petak (i,j) pada awalnya sudah berisi penghalang, maka $f((i,j), mask) = f((i',j'), mask')$.
- Jika petak (i,j) tidak dilewati oleh rute yang disiapkan oleh panitia dan pada awalnya tidak berisi penghalang, maka $f((i,j), mask)$ adalah nilai minimum dari kedua kemungkinan berikut:
  - Jika kita meletakkan penghalang pada petak (i,j), maka banyaknya penghalang tambahan minimum yang dibutuhkan pada petak $(i',j'),\cdots,(R,C)$ adalah $1 + f((i',j'), mask')$.
  - Jika kita tidak meletakkan penghalang pada petak (i,j), maka banyaknya penghalang tambahan minimum yang dibutuhkan pada petak $(i',j'),\cdots,(R,C)$ adalah $f((i',j'), mask' + k)$, dengan k adalah 0 jika petak $(i-1,j)$ dan petak $(i,j-1)$ keduanya tidak dapat diraih dari $(1,1)$, atau 1 jika tidak.

Banyaknya penghalang yang harus diletakkan adalah $f((1,1),0)$. Untuk mengetahui petak mana saja yang diletakkan penghalang, kita dapat memperhatikan kemungkinan mana yang menghasilkan nilai $f((i,j), mask)$ yang minimum, untuk semua pasang $((i,j), mask)$. Kompleksitas dari solusi ini adalah $O(RC2^C)$.

3.2 Solusi 2: Pada awalnya tidak ada petak yang berisi penghalang

Perhatikan bahwa himpunan petak yang tidak dilewati oleh rute yang disiapkan oleh panitia dapat dipartisi menjadi dua: himpunan petak yang berada di atas rute yang disiapkan oleh panitia, dan himpunan petak yang berada di bawah rute yang disiapkan oleh panitia. Perhatikan bahwa peletakkan penghalang pada kedua himpunan ini bersifat independen.

Untuk kemudahan diskusi, kita akan memperhatikan himpunan petak yang berada di bawah rute yang disiapkan oleh panitia dan akan meletakkan penghalang pada himpunan tersebut. Peletakkan penghalang pada himpunan petak yang berada di atas rute yang disiapkan oleh panitia dapat dilakukan dengan cara serupa. Perhatikan bahwa himpunan petak yang berada di bawah rute yang disiapkan oleh panitia dapat dibagi menjadi beberapa blok setiap kali pembelokan, seperti diilustrasikan pada gambar berikut:



Terdapat tiga blok pada contoh di atas: blok pertama berwarna kuning, blok kedua berwarna biru, dan blok ketiga berwarna merah.

Definisi 3.2. *Definisikan K sebagai banyaknya blok.*

Definisi 3.3. *Untuk blok ke-i, definisikan X_i sebagai sebagai banyaknya penghalang yang dibutuhkan untuk menghubungkan ujung kanan atas blok ke sisi kiri blok, dan Y_i sebagai banyaknya penghalang yang dibutuhkan untuk menghubungkan ujung kanan atas blok ke sisi bawah blok.*

Menggunakan contoh diatas, $K = 3$, $X_1 = 3, X_2 = 2, X_3 = 3$ dan $Y_1 = 2, Y_2 = 4, Y_3 = 2$.

Karena kita harus memastikan bahwa seluruh rute yang keluar dari rute yang disarankan oleh panitia melewati himpunan petak yang berada di bawah rute yang disiapkan oleh panitia tidak dapat kembali masuk ke rute yang disarankan oleh panitia, maka banyaknya penghalang yang dibutuhkan adalah

$$\min_{0 \le k \le K} \left(\sum_{1 \le j \le k} X_j + \sum_{k < j \le K} Y_j \right)$$

Kompleksitas dari solusi ini adalah $O(RC)$.

3.3 Solusi 3: Rute yang disiapkan panitia hanya berbelok sekali pada ujung kanan atas

Jika terdapat hanya satu blok pada himpunan petak yang berada di bawah rute yang disiapkan oleh panitia, maka terdapat solusi yang menggunakan *shortest path*. Secara kasarnya, kita ingin menghubungkan ujung kanan atas blok ke ujung kiri (kolom 1) atau ujung bawah (baris R) dengan penghalang, sedemikian sehingga tidak ada rute dari petak $(1, j)$ (untuk setiap $1 \le j < C$)) ke petak (i, C) (untuk setiap $1 < i \le R$) tanpa melewati petak $(1, C)$.

Lebih formalnya, kita dapat menyelesaikan soal ini dengan membentuk graf G dengan himpunan verteks $\{V_{i,j} | 2 \le i \le R, 1 \le j < C\}$. Untuk setiap petak (i, j):

- Untuk setiap petak (i', j') yang bersebelahan secara 8 arah dengan petak (i, j) (dengan kata lain, $|i - i'| \le 1$ dan $|j - j'| \le 1$, jika petak (i', j') awalnya berisi penghalang, maka tambahkan *edge* dengan bobot 0 dari $V_{i,j}$ ke $V_{i',j'}$.
- Tambahkan *edge* dengan bobot 1 dari $V_{i,j}$ ke $V_{i+1,j}$ (jika ada), $V_{i,j-1}$ (jika ada), dan $V_{i+1,j-1}$ (jika ada).
- Tambahkan *edge* dengan bobot 0 dari $V_{i,j}$ ke $V_{i-1,j}$ (jika ada) dan $V_{i,j+1}$ (jika ada).

Secara intuitif, jika kita sudah menelusuri graf G dari $V_{2,C-1}$ dan berada pada verteks $V_{i,j}$, maka kita telah memastikan bahwa untuk seluruh $2 \le i' \le i, j \le j' < C$, tidak ada rute dari $(1, 1)$ ke (i', j') atau tidak ada rute dari (i', j') ke (R, C). Sehingga, dapat dibayangkan bahwa seluruh petak (i', j') (untuk $2 \le i' \le i, j \le j' < C$) sudah diberikan penghalang "bayangan". Karenanya, kita dapat menambahkan *edge* dengan bobot 0 dari $V_{i,j}$ ke $V_{i-1,j}$ (jika ada) dan $V_{i,j+1}$ (jika ada).

Banyaknya penghalang tambahan yang dibutuhkan adalah jarak terpendek pada G dari $V_{2,C-1}$ ke verteks apapun yang berada di sisi kiri atau bawah grid, yaitu himpunan verteks $\{V_{i,j} | j = 1 \vee i = R\}$.

Jika menggunakan algoritme Dijkstra untuk menghitung jarak terpendek, maka kompleksitas solusi ini adalah $O(RC \log(RC))$. Karena seluruh bobot *edge* pada G adalah 0 atau 1, maka jarak terpendek juga dapat dihitung menggunakan BFS dengan *queue* dua ujung. Kompleksitas solusi ini adalah $O(RC)$.

3.4 Menyelesaikan seluruh kasus uji

Dengan menggabungkan seluruh solusi, seluruh kasus uji pada soal ini dapat diselesaikan dalam waktu tidak lebih dari 1 detik.

- Kasus uji 1 sampai 4 memenuhi batasan "$R \leq 100$ dan $C \leq 16$", sehingga solusi 1 dapat digunakan.
- Kasus uji 5 sampai 7 memenuhi batasan "S hanya mengandung karakter `.`", sehingga solusi 2 dapat digunakan.
- Kasus uji 8 sampai 10 memenuhi batasan "$M[i] =$ `>`, untuk $1 \leq i < C$ dan $M[i] =$ `v`, untuk $C \leq i \leq R + C - 2$", sehingga solusi 3 dapat digunakan.

4 2A. Tempat Wisata

Soal ini ditulis oleh Suhendry Effendy.

4.1 Subsoal 3

Untuk setiap tempat wisata ke-i, kita simpan turis mana yang mengunjungi tempat wisata tersebut. Caranya adalah, untuk setiap turis ke-j, iterasikan tempat-tempat mana saja yang ia kunjungi dan tambahkan turis ke-j ini sebagai salah satu pengunjungnya. Kompleksitas untuk mengiterasikan tempat wisata setiap turis adalah $O(M)$.

Setelah komputasi awal di atas, kita mulai menghitung banyaknya turis yang dikenali oleh setiap turis. Untuk setiap turis, dan untuk setiap tempat wisata yang dikunjunginya, lihat turis mana saja yang berada di tempat wisata tersebut menggunakan hasil perhitungan di awal. Kita dapat menyimpan turis mana saja yang sudah ia temui di sebuah *array* berisi N *boolean*. Setelah diiterasikan untuk semua tempat wisata, hitung berapa banyak elemen pada *array* tersebut yang bernilai *true*. Total kompleksitas adalah $O(N^2M)$.

4.2 Subsoal 4

Berhubung tempat wisata yang dikunjungi oleh setiap turis berada di tempat yang konsekutif, maka tempat wisata yang ia kunjungi dapat direpresentasikan sebagai sebuah interval $[l, r]$. Maka dari itu, dua turis akan berkenalan jika dan hanya jika kedua interval tempat wisata yang dikunjungi saling berpotongan. Salah satu cara untuk mengecek apakah dua buah interval, $[l_1, r_1]$ dan $[l_2, r_2]$, berpotongan adalah dengan cara mengecek bahwa pertidaksamaan $\max(l_1, l_2) \leq \min(r_1, r_2)$ terpenuhi. Kompleksitas totalnya adalah $O(N^2)$.

4.3 Subsoal 5

Sama seperti subsoal 4, kita akan mencoba menghitung banyaknya interval yang berpotongan dengan sebuah interval. Namun, kita akan menyelesaikan masalah ini dengan sudut pandang yang berbeda, yaitu menghitung banyaknya interval yang tidak berpotongan. Dua buah interval $[l_1, r_1]$ dan $[l_2, r_2]$ tidak berpotongan apabila salah satu dari $l_1 > r_2$ atau $r_1 < l_2$ terpenuhi. Maka dari itu, untuk setiap interval i, kita cukup menghitung banyaknya interval j yang memenuhi $l_i > r_j$ atau $r_i < l_j$. Jika kita mengiterasikannya untuk setiap turis, maka kompleksitasnya akan sama saja dengan subsoal 4. Maka dari itu, kita memerlukan cara yang lebih cepat.

Definisi 4.1. *Start point sebuah interval $[l, r]$ adalah ujung kirinya (dengan kata lain, nilai l).*

Definisi 4.2. *End point sebuah interval $[l, r]$ adalah ujung kanannya (dengan kata lain, nilai r).*

Konstruksikan sebuah *prefix sum* dengan elemen ke-i menyimpan banyaknya *end point* yang terletak pada posisi $\leq i$. Konstruksikan pula sebuah *suffix sum* yang mana elemen ke-i menyimpan banyaknya *start point* yang terletak pada posisi $\geq i$. Maka banyaknya elemen yang tidak berpotongan dengan interval $[l, r]$ adalah $prefix_{l-1}$ + $suffix_{r+1}$. Konstruksi $prefix$ dan $suffix$ sum mempunyai kompleksitas $O(M)$, sehingga kompleksitas totalnya menjadi $O(N + M)$.

4.4 Subsoal 6

Lakukan *line sweeping* pada interval-interval yang ada. Jika kita sedang melakukan *sweeping* pada sebuah *start point* dan posisi ini ditempati oleh lebih dari satu interval, maka tambahkan 1 pada jawaban kedua interval tersebut. Total kompleksitas adalah $O(NlogN)$.

4.5 Subsoal 7

Kita lakukan hal yang serupa seperti subsoal 5, tetapi perhitungan banyaknya interval yang tidak berpotongan bukanlah menggunakan bantuan *prefix sum*. Konstruksikan sebuah array s yang berisi *start points* dari semua interval yang terurut berdasarkan posisinya. Konstruksikan pula sebuah array e yang berisi *end points* dari semua interval yang terurut berdasarkan posisinya. Perhitungan banyaknya interval yang tidak berpotongan bisa dengan cara mencari banyaknya elemen yang $< l$ pada array e ditambah banyaknya elemen yang $> r$ pada s menggunakan *binary search*. Kompleksitas totalnya adalah $O(NlogN)$.

5 2B. Mencari Emas

Soal ini ditulis oleh Jonathan Irvin Gunawan.

Definisi 5.1. *Hasil yang diberikan mesin apabila tombol ditekan pada petak (r, c) adalah* $K_{r,c}$.

5.1 Subsoal 1

Tekan tombol mesin pada petak $(i, 1)$ untuk setiap i yang memenuhi $1 \leq i \leq R$. Apabila mesin memberikan hasil terkecil ketika tombol ditekan pada petak $(r, 1)$, maka kedua potongan emas terletak pada petak $(r, K_{r,1} + 1)$. Banyaknya penekanan tombol pada solusi ini adalah R.

5.2 Subsoal 2

Tekan tombol mesin pada petak $(1, 1)$. Petak-petak yang menjadi kandidat jawaban adalah petak-petak (r, c) yang berjarak $K_{1,1}$ dari petak $(1, 1)$, yakni yang memenuhi $r + c = 2 + K_{1,1}$. Tekan tombol mesin pada petak $(R, 1)$. Petak-petak yang menjadi kandidat jawaban adalah petak-petak (r, c) yang berjarak $K_{R,1}$ dari petak $(R, 1)$, yakni yang memenuhi $r - c = R - 1 - K_{R,1}$. Petak jawaban harus memenuhi kedua persamaan tersebut dan dapat ditemukan dengan menyelesaikan sistem persamaan linear dua variabel. Banyaknya penekanan tombol pada solusi ini adalah 2.

5.3 Subsoal 3

Tekan tombol pada setiap petak (i, j) yang memenuhi $1 \leq i \leq R$ dan $1 \leq j \leq C$. Petak-petak jawaban adalah petak-petak (r, c) yang memenuhi $K_{r,c} = 0$. Banyaknya penekanan tombol pada solusi ini adalah $O(RC)$.

5.4 Subsoal 4

Untuk mencari potongan emas yang lebih kiri, tekan tombol pada setiap petak $(i, 1)$ yang memenuhi $1 \leq i \leq R$. Apabila mesin memberikan hasil terkecil ketika tombol ditekan pada petak $(r_1, 1)$, maka potongan emas terletak pada petak $(r_1, K_{r_1,1} + 1)$. Untuk mencari potongan emas yang lebih kanan, tekan tombol pada setiap petak (i, C) yang memenuhi $1 \leq i \leq R$. Apabila mesin memberikan hasil terkecil ketika tombol ditekan pada petak (r_2, C), maka potongan emas terletak pada petak $(r_2, C - K_{r_2,C})$. Apabila kedua potongan emas terletak pada kolom yang sama namun baris yang berbeda, pastikan $r_1 \neq r_2$. Banyaknya penekanan tombol pada solusi ini adalah $2R$.

5.5 Subsoal 5

Tekan tombol mesin pada petak $(1, 1)$. Potongan emas pertama terletak pada petak $(1, K_{1,1}+1)$. Tekan tombol mesin pada petak $(1, C)$. Potongan emas kedua terletak pada petak $(1, C-K_{1,C})$. Banyaknya penekanan tombol pada solusi ini adalah 2.

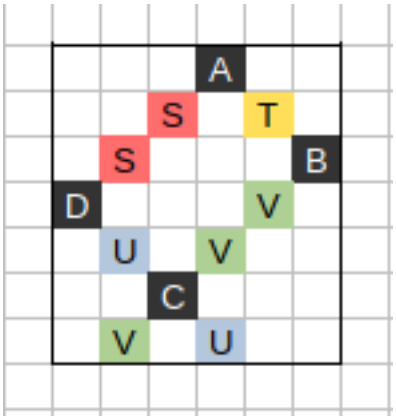
5.6 Subsoal 6

Tekan tombol mesin pada petak-petak $(1, 1)$, $(1, C)$, $(R, 1)$, dan (R, C).

Definisi 5.2. *S adalah himpunan petak-petak yang berjarak $K_{1,1}$ dari petak $(1, 1)$. T adalah himpunan petak-petak yang berjarak $K_{1,C}$ dari petak $(1, C)$. U adalah himpunan petak-petak yang berjarak $K_{R,1}$ dari petak $(R, 1)$. V adalah himpunan petak-petak yang berjarak $K_{R,C}$ dari petak (R, C).*

Definisi 5.3. $A = S \cap T$. $B = T \cap V$. $C = U \cap V$. $D = S \cap U$.

Himpunan A, B, C, dan D dapat ditemukan dengan menyelesaikan sistem persamaan linear dua variabel. Ilustrasi dapat dilihat di bawah ini:



Untuk mempermudah, anggap A, B, C, dan D bukan merupakan himpunan kosong (ukurannya masing-masing 1). Misalkan $a \in A$, $b \in B$, $c \in C$, dan $d \in D$. Anggap pula a, b, c, dan d merupakan petak yang berbeda.

Teorema 5.1. *Potongan emas pasti terletak pada (a dan c) atau (b dan d).*

Bukti. Asumsikan potongan emas berada pada a dan b, maka $U = T$. Sehingga, $a = d$, dan $b = c$. Hal ini adalah sebuah kontradiksi. Hal serupa untuk asumsi emas berada pada (b dan c), (c dan d), atau (a dan d). □

Kita mempunyai dua kandidat jawaban, yaitu (a dan c) atau (b dan d). Untuk mengetahui yang mana yang benar, kita dapat menekan tombol pada salah satu titik (katakan a). Jika $K_a = 0$, maka jawabannya adalah a dan c. Jika tidak, maka jawabannya adalah b dan d. Banyaknya penekanan tombol pada solusi ini adalah 5.

6 2C. Penyebaran Hoaks

Soal ini ditulis oleh Suhendry Effendy. Untuk mempermudah penjabaran solusi, akan digunakan definisi berikut:

Definisi 6.1. *Jumlah waktu online semua pengguna dinyatakan dalam U, yaitu*

$$U = \sum_{1 \le i \le N} T_i$$

6.1 Subsoal 3

Subsoal ini dapat diselesaikan dengan mengsimulasikan penyebaran hoaks pada setiap jam mulai dari awal hari pertama hingga akhir hari ke-N. Catat semua pengguna yang *online* pada setiap jam mulai dari jam pertama hingga jam ke-S. Lalu, untuk setiap jam dalam N hari, tentukan apakah terdapat pengguna yang sedang *online* dan memiliki hoaks. Apabila ada, maka semua pengguna lain yang sedang *online* akan memiliki hoaks. Lakukan simulasi ini sebanyak Q kali untuk menjawab semua skenario. Kompleksitas dari solusi ini adalah $O(QN^2S)$.

6.2 Subsoal 4

Mirip seperti subsoal 3, subsoal ini juga dapat diselesaikan dengan mengsimulasikan penyebaran hoaks. Namun, simulasi dilakukan per hari dan tidak per jam. Berikut cara mengsimulasikan penyebaran hoaks setiap harinya selama N hari: Untuk setiap pengguna yang memiliki hoaks, cari semua pengguna yang waktu *online*-nya berpotongan dengan waktu *online* pengguna yang memiliki hoaks. Lalu, tandai pengguna tersebut sebagai memiliki hoaks. Untuk menentukan apakah dua pengguna memiliki waktu *online* yang berpotongan, dapat digunakan definisi:

Definisi 6.2. *Dua interval (L_1, R_1) dan (L_2, R_2) dikatakan berpotongan jika dan hanya jika kondisi berikut terpenuhi:*

$$(L_1 \le L_2 \le R_1) \vee (L_1 \le R_2 \le R_1) \vee (L_2 \le L_1 \le R_2) \vee (L_2 \le R_1 \le R_2)$$

Definisi 6.3. *Dua pengguna A dan B dikatakan memiliki waktu online yang berpotongan jika dan hanya jika terdapat setidaknya satu interval waktu online pada A yang berpotongan dengan setidaknya satu interval waktu online pada B*

Kompleksitas dari solusi ini adalah $O(QNU^2)$.

6.3 Subsoal 5

Perhatikan bahwa waktu *online* setiap pengguna selalu tetap setiap harinya. Maka, pencarian pengguna yang saling berpotongan hanya perlu dilakukan sekali.

Modelkan waktu *online* semua pengguna sebagai graf G dengan N verteks, di mana dua verteks i dan j terhubung secara langsung jika dan hanya jika pengguna ke-i dan pengguna ke-j memiliki waktu *online* yang berpotongan. Anggap sumber hoaks adalah pengguna P. Maka, jumlah pengguna yang memiliki hoaks setelah N hari sama dengan jumlah verteks dalam G yang terhubung ke P dengan jarak kurang dari atau sama dengan N.

Graf G dapat dibangun dengan mengiterasi semua pasangan pengguna dan mengecek apakah waktu *online* mereka berpotongan.

Skenario ke-i dapat dijawab dengan menjalankan algoritme *Breadth First Search* dari verteks P[i] dan menghitung jumlah verteks yang dikunjungi dan berjarak kurang dari N.

Kompleksitas dari solusi ini adalah $O(U^2 + N^2Q)$.

6.4 Subsoal 6

Untuk memudahkan penjabaran solusi, akan didefinisikan notasi berikut:

Definisi 6.4. *Suatu edge tak berarah dari graf dinyatakan dalam (i, j), di mana i dan j adalah pasangan verteks di G yang dihubungkan oleh edge tersebut*

Selain itu, perhatikan observasi berikut:

Observasi 6.1. *Pada graf dengan N verteks, jarak antara sembarang dua verteks di dalam graf pasti tidak lebih dari N.*

Maka, jawaban dari skenario ke-i adalah jumlah verteks di dalam graf G yang terhubung dengan verteks P[i]. Jumlah ini dapat dihitung dengan membuat sebuah *Disjoint Set Union* (DSU) berukuran N elemen. Lalu, untuk setiap *edge* (i, j) yang terdapat di G, gabungkan elemen i dan j di DSU. Jawaban dari skenario ke-i adalah jumlah elemen yang terdapat pada *subset* P[i].

Kompleksitas dari solusi ini adalah $O(U^2 + Q)$.

6.5 Subsoal 7

Solusi dari soal ini adalah solusi subsoal 6 yang dioptimasi dengan teknik *line sweep*. Pertama, siapkan dua variabel bernama D dan cnt, yang mana D menyimpan indeks dari salah satu pengguna yang sedang aktif (atau bernilai -1 jika tidak ada pengguna yang sedang aktif). Variabel cnt berfungsi untuk mencatat jumlah interval yang sedang aktif. Urutkan titik awal dan titik akhir semua interval waktu, dan lakukan *line sweep* dari awal ke akhir secara berurutan. Terdapat dua skenario dalam proses *line sweep*:

1. Ketika memasuki titik awal suatu interval (anggap interval ini adalah waktu *online* pengguna X):
   (a) Apabila $cnt > 0$, gabungkan pengguna X dengan pengguna D di DSU.
   (b) Apabila $cnt = 0$, ubah nilai D menjadi X.

   Terakhir, tambahkan nilai cnt sebanyak 1.
2. Ketika memasuki titik akhir suatu interval, kurangi nilai cnt sebanyak 1. Apabila cnt berubah menjadi 0, maka ubah D menjadi -1.

Perhatikan bahwa apabila terdapat lebih dari satu interval yang telah dilewati titik awalnya dan berpotongan dengan interval milik X, kita hanya perlu menggabungkan X dengan **satu** pengguna saja. Hal ini karena semua interval yang telah dilewati dan memang berpotongan pasti sudah saling tergabung satu sama lain. Jadi, jumlah penggabungan maksimal adalah $O(N)$.

Kompleksitas dari solusi ini adalah $O(NlogN)$.