OSN 2013 Bidang Informatika

Pembahasan Soal

2 - 9 September 2013

Menggelindingkan Kubus

Penulis soal: Derianto Kusuma

Soal ini merupakan masalah *shortest path* pada graf implicit, dengan simpul-simpulnya (*node/vertex*) adalah konfigurasi-konfigurasi dari kubus, dan dua buah simpul A dan B dihubungkan oleh sebuah sisi (*edge*) apabila kubus dengan konfigurasi A dapat digelindingkan dalam sekali langkah menjadi konfigurasi B. Karena setiap kali penggelindingan dihitung sebagai 1 langkah, maka soal ini dapat diselesaikan menggunakan BFS (*Breadth-First Search*).

Subsoal 1 & 2

Subsoal 1 dan 2 pada soal ini merupakan subsoal terbuka (*open subtask*) yang dapat diunduh dan dikerjakan di kertas, sehingga tidak ada pembahasan untuk kedua subsoal ini.

Subsoal 3

Terdapat tepat dua buah sisi kubus yang bernomor 2, sedangkan semua sisi kubus lainnya bernomor 1. Dengan demikian, sebuah *node* dapat direpresentasikan sebagai sebuah pasangan $(arah1, arah2)$, dengan $arah1$ dan $arah2$ menyatakan ke arah mana kedua sisi yang bernomor 2 saat ini sedang menghadap. Jika kita nyatakan setiap arah sebagai bilangan (sebagai contoh atas = 1, bawah = 2, dan seterusnya), maka sebuah *node* dapat direpresentasikan sebagai sebuah pasangan bilangan (x, y).

Dalam implementasinya, pasangan bilangan ini dapat dikonversikan menjadi sebuah bilangan saja untuk kemudahan pemrosesan. Sebagai contoh, (x, y) dapat diubah menjadi $U = x \times 10 + y$. Diberikan U, kita bisa kembali memperoleh x dan y. Yakni, $(x, y) = (U\ div\ 10, U\ mod\ 10)$.

Setelah berhasil merepresentasikan *node*, kita harus memikirkan transisi antar *node*. Sebagai contoh, apabila *node* (utara, bawah) digelindingkan ke arah selatan, apa node baru yang dihasilkan? Kita dapat menyelesaikan ini dengan membangun tabel 2 dimensi $transisi[arah1][g] = arah2$. Ini maksudnya sisi yang berada pada $arah1$ akan berada pada $arah2$ setelah digelindingkan ke arah g. Tabel ini dapat dihitung secara *hardcode* pada solusi.

Subsoal 4

Terdapat tepat sebuah sisi kubus yang bernomor 2 (berbeda dengan subsoal sebelumnya yang memiliki tepat 2 sisi kubus bernomor 2), sedangkan semua sisi kubus lainnya bernomor 1. Dengan demikian, sebuah *node* dapat direpresentasikan sebagai sebuah bilangan saja yang menunjukkan sisi yang bernomor 2 saat ini sedang menghadap ke arah mana. Transisinya pun lebih mudah, yakni cukup menggunakan tabel transisi pada Subsoal 3.

Subsoal 5

Kali ini, nomor-nomor pada sisi-sisi kubus tidak dibatasi. Dengan demikian, untuk merepresentasikan sebuah *node*, tidak ada jalan lain selain menyatakan nomor pada sisi-sisi kubus pada setiap arah. Sehingga, sebuah *node* direpresentasikan sebagai tupel 6 bilangan $(a_1, a_2, a_3, a_4, a_5, a_6)$, dengan a_i adalah nomor pada sisi kubus yang menghadap arah ke-i.

Dalam implementasinya, tupel ini dapat direpresentasikan sebagai *vector* (pada C/C++) maupun *string* yang karakter-karakternya adalah '1' sampai '6'.

Untuk menghitung transisinya, dapat memanfaatkan tabel transisi pada Subsoal 3 sebanyak 6 kali untuk setiap transisi, karena terdapat 6 sisi yang harus diperhitungkan.

Berbaris Sebelum Masuk

Penulis soal: Ashar Fuadi

Subsoal 1 & 2

Subsoal 1 dan 2 pada soal ini merupakan subsoal terbuka (*open subtask*) yang dapat diunduh dan dikerjakan di kertas, sehingga tidak ada pembahasan untuk kedua subsoal ini.

Subsoal 3

Pada subsoal ini, hanya terdapat paling banyak 8 orang siswa. Dengan demikian, kita dapat melakukan *brute force* untuk semua kemungkinan barisan siswa (ada paling banyak $8! = 40.320$ kemungkinan). Untuk setiap kemungkinan barisan, cek apakah barisan tersebut memenuhi syarat. Apabila memenuhi, cetak kemungkinan barisan tersebut.

Kompleksitas solusi ini adalah $O(N!)$.

Subsoal 4

Untuk mempermudah penjelasan, kita definisikan $f(i)$ = banyaknya siswa yang lebih pendek daripada siswa ke-i, dan $g(i)$ = banyaknya siswa di depan siswa ke-i yang lebih pendek daripada siswa ke-i pada barisan. Perlu diingat bahwa semua tinggi badan siswa berbeda-beda. Untuk sembarang siswa ke-i, berapakah nilai terbesar $g(i)$ yang mungkin? Jawabannya tentu saja adalah $f(i)$, yakni jika semua siswa yang lebih pendek darinya berdiri di depannya pada barisan.

Dapat ditunjukkan bahwa ternyata kita dapat membuat barisan sedemikian sehingga untuk setiap siswa ke-i, $g(i) = x$ untuk sembarang $0 \leq x \leq f(i)$ dengan cara sebagai berikut. Pertama, kita urutkan siswa-siswa tersebut secara terurut menaik berdasarkan tinggi badan. Dengan demikian, untuk setiap siswa ke-i, $g(i) = f(i)$. Lalu, proses siswa-siswa tersebut dari depan. Untuk setiap siswa ke-i yang sedang diproses, tujuan kita adalah mengurangi nilai $g(i)$ menjadi x, dengan x adalah nilai sembarang yang memenuhi $A[i] \leq x \leq B[i]$. Caranya cukup dengan menggeser siswa tersebut sebanyak $f(i) - x$ ke depan. Dapat dibuktikan bahwa penggeseran ini tidak akan mengganggu nilai-nilai $g(i)$ dari siswa-siswa yang telah diproses.

Untuk mempermudah, kita bisa ambil nilai x untuk siswa ke-i adalah $A[i]$.

Kompleksitas solusi ini adalah $O(N^2)$.

Lipat Kertas

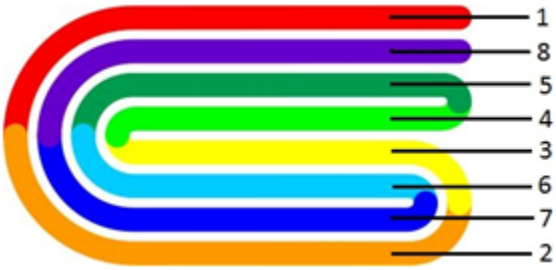
Penulis soal: Ammar Fathin Sabili

Subsoal 1 & 2

Subsoal 1 dan 2 pada soal ini merupakan subsoal terbuka (*open subtask*) yang dapat diunduh dan dikerjakan di kertas, sehingga tidak ada pembahasan untuk kedua subsoal ini.

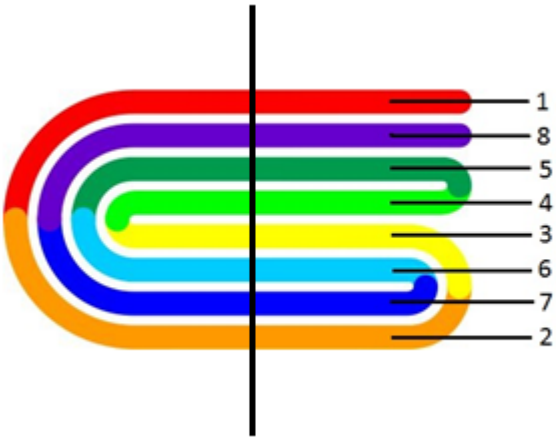
Subsoal 3

Cara menentukan urutan jenis cekungan yaitu dengan menggunakan *pattern finding*. Perhatikan Gambar 1 yang mengilustrasikan lipatan kertas untuk contoh kasus uji.



Gambar 1. Ilustrasi lipatan kertas untuk contoh kasus uji

Pertama, bagi lipatan kertas tersebut menjadi dua bagian (lihat Gambar 2 untuk ilustrasi pembagian yang dilakukan).



Gambar 2. Lipatan kertas yang dibagi 2 oleh garis hitam vertikal

Perhatikan bagian sebelah kiri dari hasil membagi lipatan kertas:

- Sisi warna 1 bergabung dengan sisi warna 2 → Cekungan ke-1.
- Sisi warna 3 bergabung dengan sisi warna 4 → Cekungan ke-3.
- Sisi warna 5 bergabung dengan sisi warna 6 → Cekungan ke-5.
- Sisi warna 7 bergabung dengan sisi warna 8 → Cekungan ke-7.

Setelah itu perhatikan bagian sebelah kanan dari hasil membagi lipatan kertas:

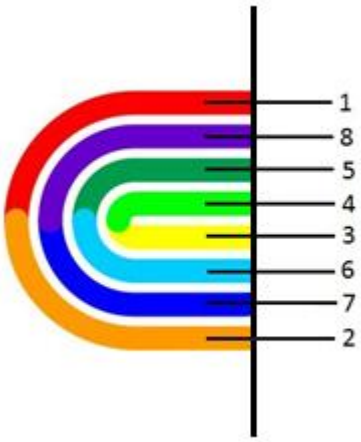
- Sisi warna 2 bergabung dengan sisi warna 3 → Cekungan ke-2.

- Sisi warna 4 bergabung dengan sisi warna 5 → Cekungan ke-4.
- Sisi warna 6 bergabung dengan sisi warna 7 → Cekungan ke-6.

Dari pengamatan di atas dapat disimpulkan hal berikut.

1. **Sisi warna X dan sisi warna $X + 1$ membentuk cekungan ke-X.**
2. **Apabila X merupakan bilangan ganjil, maka cekungan ke-X berada pada bagian kiri.**
3. **Apabila X merupakan bilangan genap, maka cekungan ke-X berada pada bagian kanan.**

Dari sini kita juga dapat menentukan jenis cekungan-cekungan bernomor ganjil dari bagian kiri dan cekungan-cekungan bernomor genap dari bagian kanan. Perhatikan juga bahwa hasil pembukaan lipatan mengakibatkan seluruh nomor berurutan dari kiri ke kanan.



Gambar 3. Potongan bagian sebelah kiri dari ilustrasi lipatan kertas

Dimulai dari bagian sebelah kiri (ilustrasi di Gambar 3):

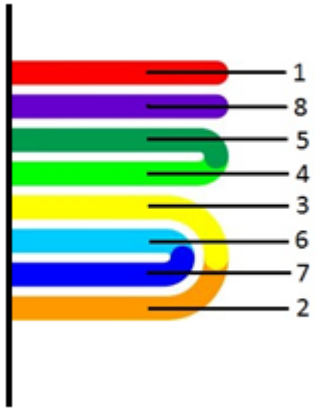
1. Sisi warna 1 berada di atas tumpukan sisi warna 2. Sisi warna 1 dan 2 membentuk tanda ⊂ (cekungan dengan 'mulut' menghadap ke kanan) yang bila diputar 90 derajat berlawanan arah jarum jam akan membentuk tanda ∪ (cekungan dengan 'mulut' menghadap ke atas) dengan sisi warna 1 berada di sebelah kiri sisi warna 2. ∪ berarti cekungan ke bawah (B).
2. Sisi warna 3 berada di bawah tumpukan sisi warna 4. Sisi warna 3 dan 4 membentuk tanda ⊂ yang bila diputar 90 derajat berlawanan arah jarum jam akan membentuk tanda ∪ dengan sisi warna 4 berada di sebelah kiri sisi warna 3. Hal ini tidak dapat dilakukan karena pada hasil pembukaan lipatan, sisi warna 3 harus berada di sebelah kiri sisi warna 4 sehingga harus diputar kembali 180 derajat (dari hasil putaran sebelumnya) yang akan membentuk tanda ∩ (cekungan dengan 'mulut' menghadap ke bawah). ∩ berarti cekungan ke atas (A).
3. Sisi warna 5 berada di atas tumpukan sisi warna 6. Sisi warna 5 dan 6 membentuk tanda ⊂ yang bila diputar 90 derajat berlawanan arah jarum jam akan membentuk tanda ∪ dengan sisi warna 5 berada di sebelah kiri sisi warna 6. ∪ berarti cekungan ke bawah (B).
4. Sisi warna 7 berada di bawah tumpukan sisi warna 8. Sisi warna 7 dan 8 membentuk tanda ⊂ yang bila diputar 90 derajat berlawanan arah jarum jam akan membentuk tanda ∪ dengan sisi warna 8 berada di sebelah kiri sisi warna 7. Hal ini tidak dapat dilakukan karena pada hasil pembukaan lipatan, sisi warna 7 harus berada di sebelah kiri sisi warna 8 sehingga

harus diputar kembali 180 derajat yang akan membentuk tanda ∩. ∩ berarti cekungan ke atas (A).

5. Hasil akhir yang didapat dari sisi lipatan kertas sebelah kiri adalah B*A*B*A.

Dari pengamatan di atas dapat disimpulkan hal-hal berikut:

1. **Apabila sisi warna X berada di atas tumpukan sisi warna $X+1$, maka jenis cekungannya adalah cekungan ke bawah/cekungan dengan 'mulut menghadap ke atas (B).**
2. **Apabila sisi warna X berada di bawah tumpukan sisi warna $X+1$, maka jenis cekungannya adalah cekungan ke atas/cekungan dengan 'mulut' menghadap ke bawah (A).**
3. **Untuk dua aturan di atas hanya berlaku untuk nilai X bilangan ganjil (cekungan ke-X di sebelah kiri lipatan kertas).**



Gambar 4. Potongan bagian sebelah kanan dari ilustrasi lipatan kertas

Sekarang ditinjau dari bagian sebelah kanan potongan lipatan kertas (lihat Gambar 4):

1. Sisi warna 2 berada di bawah tumpukan sisi warna 3. Sisi warna 2 dan 3 membentuk tanda ⊃ (cekungan dengan 'mulut' mengarah ke kiri) yang bila diputar 90 derajat searah jarum jam akan membentuk tanda ∪ dengan sisi warna 2 berada di sebelah kiri sisi warna 3. ∪ berarti cekungan ke bawah (B).
2. Sisi warna 4 berada di bawah tumpukan sisi warna 5. Sisi warna 4 dan 5 membentuk tanda ⊃ yang bila diputar 90 derajat searah jarum jam akan membentuk tanda ∪ dengan sisi warna 4 berada di sebelah kiri sisi warna 5. ∪ berarti cekungan ke bawah (B).
3. Sisi warna 6 berada di atas tumpukan sisi warna 7. Sisi warna 6 dan 7 membentuk tanda ⊃ yang bila diputar 90 derajat searah jarum jam akan membentuk tanda ∪ dengan sisi warna 7 berada di sebelah kiri sisi warna 6. Hal ini tidak dapat dilakukan karena pada hasil pembukaan lipatan, sisi warna 6 harus berada di sebelah kiri sisi warna 7 sehingga harus diputar kembali 180 derajat (dari hasil putaran sebelumnya) yang akan membentuk tanda ∩. ∩ berarti cekungan ke atas (A).
4. Hasil akhir yang didapat dari observasi lipatan kertas sebelah kanan adalah *B*B*A*.

Dari pengamatan di atas dapat disimpulkan :

1. **Apabila sisi warna X berada di atas tumpukan sisi warna $X+1$, maka jenis cekungannya adalah cekungan ke atas (A).**

2. **Apabila sisi warna X berada di bawah tumpukan sisi warna $X+1$, maka jenis cekungannya adalah cekungan ke bawah (B).**
3. **Dua aturan di atas hanya berlaku untuk nilai X bilangan genap (cekungan ke-X di sebelah kanan).**

Dengan menggabungkan kedua hasil akhir untuk potongan sebelah kiri dan kanan didapat jawaban BBABBAA.

Berikut *pseudocode* dari solusi subsoal 3.

```
for i = 1 to N do begin
   read(x);
   posisi[x] = i;
end;

for i = 1 to (N - 1) do begin
   if (i % 2 = 1) do begin
      if (posisi[i] < posisi[i + 1]) write("A"); else write("B");
   end else begin
      if (posisi[i] < posisi[i + 1]) write("B"); else write("A");
   end;
end;
```

Perhatikan bahwa selembar kertas dapat dibalik sehingga semua cekungan ke atas (A) berubah menjadi cekungan ke bawah (B) dan sebaliknya. Sehingga, selalu terdapat tepat 2 buah solusi untuk pelipatan kertas yang VALID. (Contoh solusi kedua untuk contoh kasus uji adalah AABAABB).

Subsoal 4

Tidak ada algoritma spesifik untuk subsoal ini. Peserta hanya diminta untuk mensimulasikan pelipatan kertas secara langsung untuk semua kasus-kasus yang mungkin (seperti pada subsoal 1 dan 2).

Subsoal 5

Cara menentukan apakah sebuah lipatan VALID atau tidak dapat dilakukan dengan memperhatikan observasi di subsoal 3. Teknik yang digunakan tetaplah *pattern finding*.

Lihat kembali Gambar 2. Perhatikan bahwa di bagian kiri maupun di bagian kanan lipatan kertas, tidak ada 2 lekukan yang saling tumpang tindih.

Sebuah lekukan dapat direpresentasikan sebagai interval dengan ujung-ujungnya adalah posisi kedua sisi warna pada tumpukan dalam lipatan kertas tersebut. Sebagai contoh lekukan ke-1 (sisi warna 1 dan sisi warna 2) memiliki representasi interval [1,8], dan lekukan ke-2 (sisi warna 2 dan sisi warna 3) memiliki rerpresentasi interval [8,5] atau [5,8].

Berikut adalah penjabaran semua representasi interval lekukan di sebelah kiri dan kanan potongan kertas.

- Bagian kiri: [1,8] , [4,5] , [3,6] , [2,7].

- Bagian kanan: [5,8] , [3,4] , [6,7].

Perhatikan bahwa untuk setiap pasang interval pada suatu bagian, salah satu interval berada di dalam interval yang lain (seperti [2,7] dan [4,5]) atau kedua interval saling lepas (seperti [3,4] dan [6,7]). Tidak ada pasangan interval yang hanya beririsan sebagian saja (contohnya [1,5] dan [3,7]).

Dari pengamatan di atas dapat disimpulkan bahwa **suatu lipatan kertas dikatakan VALID apabila setiap pasang notasi interval pada suatu bagian memenuhi salah satu dari kedua syarat berikut**.

1. **Salah satu notasi interval berada di dalam notasi interval yang lain**, atau
2. **Kedua notasi interval saling lepas**.

Untuk memeriksa apakah setiap pasang interval memenuhi syarat tersebut, dapat digunakan metode *brute force* dengan kompleksitas $O(N^2)$.

Subsoal 5

Dengan mengamati lebih seksama lagi, kedua syarat interval sebelumnya merupakan syarat sah bagi permasalahan *bracket matching*. Dengan merepresentasikan ujung-ujung interval sebagai tanda buka kurung dan tutup kurung (dengan catatan bahwa setiap interval merupakan pasangan tanda buka-tutup kurung yang berbeda-beda), dapat ditentukan apakah sebuah bagian merupakan sah atau tidak.

Berikut adalah ilustrasi representasi interval pada contoh kasus uji.

1. Sebelah kiri: [1,8], [4,5], [3,6] , [2,7] menjadi ([{ < > }])
2. Sebelah kanan: [5,8], [3,4], [6,7] menjadi $**$ ( ) [< >] ($*$ menandakan spasi)

Perhatikan bahwa kedua *string* tersebut merupakan konfigurasi *bracket matching* yang sah. Sehingga, kompleksitas solusi dapat direduksi menjadi $O(N)$ dengan bantuan struktur data *stack*.

Tebak Himpunan

Penulis soal: Ashar Fuadi

Seperti yang tertulis di deskripsi soal, solusi untuk subsoal yang lebih tinggi belum tentu dapat menyelesaikan subsoal yang lebih rendah, sehingga harus dilakukan percabangan untuk menyelesaikan subsoal-subsoal dengan solusi berbeda secara terpisah.

Subsoal 1 & 2

Subsoal 1 dan 2 pada soal ini merupakan subsoal terbuka (*open subtask*) yang disediakan dalam format *game* interaktif yang dapat dimainkan (dan Anda bisa mengunduh *source code* yang menjawab dengan benar subsoal yang telah dimainkan dengan benar), sehingga tidak ada pembahasan untuk kedua subsoal ini.

Subsoal 3

Untuk himpunan $\{1, 2, \ldots, N\}$, banyaknya subhimpunan yang mungkin adalah 2^N. Karena batas tebakan yang dapat diajukan juga sebanyak 2^N, maka kita dapat mencoba menebak setiap subhimpunan. Salah satunya pasti adalah subhimpunan yang tepat sebagai jawaban.

Subsoal 4

Sebenarnya, kita tidak perlu menebak semua subhimpunan. Kita cukup mencari tahu apakah setiap bilangan dari 1 sampai dengan N merupakan anggota dari S atau bukan. Caranya adalah dengan melakukan tebakan "`1 x`" untuk setiap $1 \leq x \leq N$. Terakhir, tebak subhimpunan dengan anggota-anggota yang mendapat jawaban "`bisajadi`" pada tebakan-tebakan sebelumnya. Dengan cara ini, akan diperlukan setidaknya $N + 1$ tebakan, yang sesuai dengan batas tebakan pada subsoal ini.

Subsoal 5

Kita ingin menentukan nilai R sedemikan sehingga untuk semua nilai $x \leq R$, x merupakan anggota dari S, dan untuk semua $x > R$, x bukan merupakan anggota S. Dengan demikian, kita bisa menggunakan paradigma *binary search* untuk menentukan R. Yakni, kita ingin mencari bilangan terbesar yang masih merupakan anggota S.

Berikut adalah *pseudocode* yang mengilustrasikan solusi yang dimaksud.

```
low = 1, high = N, R = 0

while (low < high)
   mid = (low + high) / 2
   tebak "1 mid"
   if (jawaban == "bisajadi")
      R = mid, low = mid + 1
   else
      high = mid - 1
   endif
endwhile

tebak "R 1 2 3 ... R"
```

Implementasi *binary search* bisa bermacam-macam sehingga banyaknya tebakan yang dihasilkan pun berbeda-beda. Oleh karena itu, pada soal ini batasan tebakan yang dapat diajukan dibuat longgar, yaitu $2 \times \lceil \log_2 N \rceil + 1$ agar tidak tergantung dari jenis implementasi *binary search* peserta.

Subsoal inilah yang tidak dapat diselesaikan oleh solusi subsoal yang lebih tinggi. Solusi untuk subsoal ini juga tidak bisa digunakan untuk menyelesaikan subsoal yang lebih rendah.

Subsoal 6

Mirip seperti subsoal 4, untuk setiap $1 \leq x \leq N$ kita akan mencari tahu apakah x adalah anggota S atau bukan. Akan tetapi, karena $K = 2$, kita tidak bisa melakukannya dengan menebak "`1 x`".

Didefinisikan himpunan kosong $P = \{\}$. Proses semua bilangan dari 1 sampai dengan N secara berurutan. Pada saat memproses bilangan x, lakukan tebakan "`2 x (x+1)`", "`2 x (x+2)`", ..., dan "`2 x N`". Terdapat dua kemungkinan untuk respons yang diberikan.

1. Terdapat setidaknya satu jawaban "`tidak`". Hal ini menunjukkan bahwa x bukanlah anggota dari S.
2. Semua jawabannya merupakan "`bisajadi`". Terdapat tiga kemungkinan:
   i. $S = P \cup \{x, x+1, x+2, \ldots, N\}$, dengan P merupakan subhimpunan berisi bilangan-bilangan dari 1 sampai $(x-1)$ yang merupakan anggota S. Coba lakukan tebakan untuk subhimpunan ini.
   ii. $S = P \cup \{x+1, x+2, \ldots, N\}$. Coba lakukan tebakan untuk subhimpunan ini.
   iii. Tidak satupun tebakan di atas yang benar. Ini menunjukkan bahwa x adalah anggota dari S. Masukkan x ke dalam P.

Setelah semua pemrosesan dilakukan, jika sebelumnya belum ada tebakan yang berhasil, maka lakukan tebakan untuk P.

Solusi ini dapat dipakai untuk menyelesaikan semua subsoal selain subsoal 4 (karena pada subsoal 4, hanya diperbolehkan menebak paling banyak $N + 1$ kali) dan subsoal 5.

Mengosongkan Matriks

Penulis soal: Mushthofa Fafa

Subsoal 1 & 2

Subsoal 1 dan 2 pada soal ini merupakan subsoal terbuka (*open subtask*) yang dapat diunduh dan dikerjakan di kertas, sehingga tidak ada pembahasan untuk kedua subsoal ini.

Subsoal 3

Pada subsoal ini, matriks hanya memiliki 1 baris. Oleh karena itu untuk mengubah seluruh bilangan matriks menjadi 0, cukup menggunakan operasi 2 (mengurangi kolom).

Kompleksitas solusi ini adalah $O(M)$.

Subsoal 4

Pada subsoal ini, matriks hanya memiliki 1 kolom. Pertama-tama kita harus mengubah kolom tersebut agar seluruh nilai dalam kolom tersebut sama dengan 1.

Misalkan bilangan terbesar di kolom tersebut adalah X. Kita akan mengalikan baris ke-i dengan 2^{k_i} terbesar sedemikian sehingga bilangan yang ada di baris ke-i di kolom tersebut tidak melebihi X. Misalkan bilangan terkecil di kolom tersebut setelah hasil operasi sebelumya adalah Y. Lakukan operasi pengurangan pada kolom tersebut dengan bilangan $Y - 1$, sehingga bilangan terbesar di kolom tersebut akan berkurang menjadi setengahnya serta bilangan terkecil di kolom tersebut menjadi 1.

Ulangi langkah yang dijelaskan di paragraf sebelumnya sampai seluruh bilangan di kolom tersebut sama dengan 1. Setelah itu, kita dapat melakukan operasi mengurangi kolom dengan bilangan 1 agar seluruh bilangan dalam kolom tersebut bernilai 0.

Kompleksitas solusi ini adalah $O(N \times \log_2 1.000.000.007) = O(N)$.

Subsoal 5

Subsoal ini merupakan generalisasi dari subsoal 4 dengan matriks yang diberikan berukuran $N \times M$. Masing-masing kolom dapat diproses secara terpisah. Maka kita hanya perlu menjalankan algoritma subsoal 4 pada tiap kolom matriks.

Kompleksitas solusi ini adalah $O(M \times N \times \log_2 1.000.000.007) = O(M \times N)$.

Kontes Menari

Penulis soal: Alham Fikri Aji

Subsoal 1 & 2

Subsoal 1 dan 2 pada soal ini merupakan subsoal terbuka (*open subtask*) yang dapat diunduh dan dikerjakan di kertas, sehingga tidak ada pembahasan untuk kedua subsoal ini.

Subsoal 3

Salah satu solusi termudah untuk soal ini adalah melakukan permutasi gerakan untuk setiap juri yang ada. Kemudian untuk setiap permutasi tarian yang mungkin, periksa apakah total nilainya melebihi nilai batas keindahan juri.

Perhatikan bahwa pada subsoal ini semua gerakan merupakan gerakan biasa, sehingga implementasi permutasi untuk menyelesaikan subsoal ini cukup *straightforward*. Untuk membangkitkan permutasi mengambil R tarian dari total N tarian (${}_NP_R$), bisa dengan memanfaatkan *depth-first search* (DFS) atau fungsi `next_permutation()` (di C++). Kompleksitas untuk solusi ini adalah $O(J \times N!)$.

Khusus untuk subsoal ini ada solusi dengan kompleksitas lebih rendah, yaitu solusi dengan memanfaatkan kombinasi (kompleksitas *upper bound* tetap $O(J \times N!)$, namun dijamin banyaknya operasi aktual untuk membangkitkan subhimpunan gerakan terpilih jauh di bawah $N!$). Untuk setiap kombinasi tarian yang melebihi nilai batas keindahan juri, kalikan dengan $R!$ (yang bisa dihitung terlebih dahulu). Namun karena implementasi kode untuk membangkitkan kombinasi tidak sesederhana implementasi kode permutasi, maka solusi ini tidak dibahas.

Subsoal 4

Sedikit perbedaan dengan subsoal sebelumnya adalah pada subsoal ini jenis gerakan tidak dibatasi pada hanya pada gerakan biasa saja, sehingga solusi memanfaatkan kombinasi yang sedikit dibahas di subsoal 3 tidak dapat digunakan lagi.

Untuk menyelesaikan soal ini dapat memanfaatkan solusi dengan permutasi di subsoal 3. Kompleksitas untuk solusi ini adalah $O(J \times N!)$.

Subsoal 5

Untuk subsoal ini batasan nilai J (banyaknya juri) membesar menjadi maksimal 1.000. Jika kita berusaha menyelesaikan subsoal ini dengan solusi naif di atas (dengan kompleksitas $O(J \times N!)$), pada kasus terburuk $N = 10$ dan $J = 1.000$, total operasi $= 1.000 \times 10! \approx 3{,}6$ milyar operasi. Jumlah operasi yang sangat besar ini tentu saja akan menghasilkan *verdict Time Limit Exceeded* (TLE).

Salah satu solusi yang bisa dimanfaatkan adalah dengan memanfaatkan permutasi + memoisasi. Pertama-tama siapkan sebuah *array* berukuran hingga total maksimal yang mungkin dari suatu permutasi gerakan, misalkan `counter[]`. Kemudian untuk setiap permutasi gerakan yang mungkin, hitung nilai total gerakannya, misalkan T. Terakhir, *increment* nilai `counter[T]` (`counter[T] = counter[T] + 1`). Dengan prosedur ini, pada akhir operasi permutasi, nilai

`counter[K]` menyatakan banyaknya permutasi gerakan tarian berbeda yang menghasilkan total nilai tepat K.

Kemudian, siapkan sebuah *array* lain berukuran sama dengan ukuran *array* `counter[]`, misalkan `sum[]`. Lakukan operasi pengisian nilai dengan isi dari `sum[K] = counter[K] + counter[K+1] + counter[K+2] + ... + counter[MAX-1] + counter[MAX]`, dengan MAX adalah indeks terbesar *array*. Dengan observasi singkat secara ekivalen pengisian nilai di atas dapat ditulis sebagai `sum[K] = counter[K] + sum[K+1]`, $K < MAX$ (`sum[MAX] = counter[MAX]`). Dengan prosedur ini, nilai `sum[K]` akan berisi banyaknya permutasi gerakan dengan total nilai $\geq K$.

Setelah mengisi nilai pada seluruh sel dalam array `sum[]`, selanjutnya dapat dicari jawaban untuk setiap juri dengan lebih mudah. Untuk menjawab banyaknya permutasi gerakan yang melebihi nilai batas keindahan suatu juri (misalkan D), maka jawabannya adalah nilai pada `sum[D+1]` (bukan `sum[D]`, karena untuk membuat juri terkesima haruslah total nilai tarian > nilai batas keindahan, bukan $\geq$ nilai batas keindahan).

Permasalahan baru muncul dari solusi di atas: bagaimana cara menentukan ukuran maksimal *array* `counter[]` dan `sum[]` (dengan kata lain, nilai MAX)? Perhatikan bahwa banyak gerakan maksimal adalah 10, dan nilai maksimal setiap gerakan adalah 1.000. Namun menghitung nilai total tertinggi yang mungkin dicapai tidak semudah menghitung $10 \times 1.000 = 10.000$ saja karena masih ada faktor 'gerakan memukau' (yang menyebabkan tepat 1 gerakan setelah gerakan memukau digandakan nilainya) dan 'gerakan meyakinkan' (yang menyebabkan semua gerakan setelah gerakan menyakinkan ditambah nilainya dengan Y, dengan nilai maksimal Y adalah 100).

Untuk gampangnya, anggap saja semua gerakan mendapat pengaruh dari gerakan meyakinkan dan gerakan memukau, sehingga batas atas nilai tertinggi yang dapat dicapai dari permutasi gerakan adalah $10 \times (1.000 \times 2 + 100) = 21.000$. Tentunya ini hanya nilai perkiraan dan bukan nilai tertinggi sebenarnya yang mampu dicapai, namun hal ini cukup karena kita sebenarnya tidak perlu tahu nilai maksimal sebenarnya (dan karena untuk menghitung nilai tertinggi sebenarnya cukup rumit) asalkan bisa diperkirakan nilai batas atas yang cukup besar (tidak boleh terlalu besar karena akan dijadikan ukuran *array*).

Sayangnya, pada solusi di atas masih terdapat kasus khusus yang belum dapat ditangani. Perhatikan pada deskripsi soal batasan tertinggi nilai batas keindahan seorang juri adalah 100.000, sementara ukuran *array* hanya dibuat sebatas sampai 21.000. Tentunya dapat mengakibatkan *Runtime Error* jika kita berusaha mengakses *array* di luar batasan indeks untuk menjawab pertanyaan (misal jika $D = 95.000$, kita berusaha menjawab dengan mengembalikan nilai `sum[95.000]`).

Untuk itu perlu diperiksa apakah nilai batas keindahan suatu juri masih dalam ukuran *array*. Jika iya, kembalikan nilai dalam `sum[]` untuk menjawab pertanyaan. Jika tidak, kembalikan 0. Hal ini dikarenakan MAX (dalam hal ini $MAX = 21.000$) adalah perkiraan batas atas nilai tertinggi yang mungkin dicapai oleh suatu permutasi gerakan tarian, sehingga semua nilai di atas nilai MAX pastilah tidak mungkin dicapai.

Cuti Liburan

Penulis soal: Ashar Fuadi

Subsoal 1 & 2

Subsoal 1 dan 2 pada soal ini merupakan subsoal terbuka (*open subtask*) yang dapat diunduh dan dikerjakan di kertas, sehingga tidak ada pembahasan untuk kedua subsoal ini.

Subsoal 3

Lakukan *brute force* pada semua kemungkinan subhimpunan baju (ada 2^N subhimpunan). Untuk setiap kemungkinan, cek apakah kemungkinan tersebut memenuhi semua syarat yang ditentukan. Dari semua kemungkinan yang memenuhi syarat, tentukan daya tahan terbesarnya.

Kompleksitas solusi ini adalah $O(2^N)$.

Subsoal 4

Karena $Q = 0$, maka sebenarnya semua subhimpunan baju sudah memenuhi syarat warna. Oleh karena itu, syarat warna dapat diabaikan. Sekarang subsoal ini menjadi *dynamic programming 0-1 knapsack* klasik. Kita definisikan $DP(i, j)$ adalah daya tahan terbesar jika kita mempunyai pilihan baju dari indeks 1 sampai i, dan kapasitas koper sebesar j. Nilainya didefinisikan sebagai berikut.

$$DP(i,j) = \begin{cases} 0 & , i = 0 \ \wedge \ 0 \leq j \leq P \\ -\infty & , 1 \leq i \leq N \ \wedge j < 0 \\ \max(\ DP(i-1,j), DP(i-1,j-W[i]) + \ V[i]) & , \text{selain di atas} \end{cases}$$

Pada formula ketiga di atas, $DP(i-1,j)$ adalah kondisi jika memilih untuk tidak membawa baju ke-i, sementara $DP(i-1,j-W[i]) + V[i]$ adalah kondisi jika memilih untuk membawa baju ke-i.

Jawaban untuk subsoal ini adalah $DP(N,P)$. Jika $DP(N,P) = -\infty$, maka cetak -1.

Kompleksitas solusi ini adalah $O(N \times P)$.

Subsoal 5

Kali ini, nilai Q bisa lebih besar dari 0. Perhatikan bahwa terdapat batasan $W_i = i$; dengan kata lain, semua baju warnanya berbeda-beda. Oleh karena itu, syarat warna dapat diubah menjadi "Pak Dengklek harus membawa setidaknya Q baju dalam koper".

Kita dapat menambah sebuah parameter pada formula yang dijabarkan di subsoal 4. Parameter tambahan tersebut menyimpan informasi 'sudah berapa baju yang dibawa'. Kita definisikan $DP(i,j,k)$ adalah daya tahan terbesar jika kita mempunyai pilihan baju dari indeks 1 sampai i, kapasitas koper sebesar j, dan banyaknya baju yang sudah dibawa sebanyak k. Nilainya didefinisikan sebagai berikut.

$$DP(i,j,k) = \begin{cases} 0 & ,i = 0 \ \wedge \ 0 \leq j \leq P \ \wedge \ k \geq Q \\ -\infty & ,i = 0 \ \wedge \ 0 \leq j \leq P \wedge \ k < Q \\ -\infty & ,1 \leq i \leq N \ \wedge \ j < 0 \ \wedge \ k \leq Q \\ \max(\ DP(i-1,j,k), DP(i-1,j-W[i],k+1) + V[i]) & \text{, selain di atas} \end{cases}$$

Jawaban untuk subsoal ini adalah $DP(N,P,0)$. Jika $DP(N,P,0) = -\infty$, maka cetak -1.

Kompleksitas untuk solusi ini adalah $O(N \times P \times Q)$.

Subsoal 6

Kali ini, $0 \leq Q \leq N$ dan mungkin terdapat lebih dari satu baju dengan warna sama. Kita dapat mulai dengan menggunakan formula DP yang sama seperti pada subsoal 5. Masalahnya adalah, saat kita memilih untuk membawa sebuah baju, apakah kita harus menambahkan k dengan 1? Apabila baju berwarna sama belum dibawa, maka tidak perlu dinaikkan. Akan tetapi, apabila belum ada baju berwarna sama yang dibawa, maka perlu dinaikkan (sehingga, definisi nilai k berubah menjadi menyatakan berapa banyak **warna** berbeda yang sudah dibawa).

Untuk menyelesaikan masalah tersebut, kita dapat mengurutkan baju-baju pada masukan terlebih dahulu menurut warnanya. Sehingga, baju-baju dengan warna yang sama akan terletak bersebelahan. Kemudian, kita tambahkan lagi sebuah parameter pada formula DP tersebut, yang bertipe *boolean* dan menyatakan apakah baju dengan warna yang sama dengan baju ke-i sudah pernah dibawa atau belum. Perubahan nilai parameter ini mudah ditentukan karena warna-warnanya sudah terurut; hanya bergantung pada nilai k sebelumnya, warna baju ke-i, dan warna baju ke-$(i-1)$. Detilnya diserahkan kepada pembaca untuk ditemukan sebagai latihan.

Jawaban untuk subsoal ini adalah $DP(N,P,0,false)$. Jika $DP(N,X,0,false) = -\infty$, cetak -1.

Kompleksitas solusi ini adalah $O(N \log N + N \times P \times Q)$.

Pabrik Kue

Penulis soal: Risan Petrus

Solusi soal ini terdiri dari beberapa bagian, masing-masing dijelaskan di bawah ini. Untuk soal ini disediakan *game* interaktif yang dapat dimainkan untuk membantu menuntun intuisi Anda untuk melakukan langkah-langkah di bawah.

1. Melakukan estimasi nilai K_i

Di beberapa hari awal, sangat sulit menentukan nilai K_i. Akan tetapi, semakin bertambahnya hari, data yang kita miliki semakin banyak sehingga kita semakin mudah untuk melakukan estimasi ini.

Pada suatu hari t, $f(i,k)$ adalah banyaknya hari t' dengan $t' + k \leq t$ dan pelanggan memesan kue jenis i tepat di salah satu hari: $t' + k$ atau t, tetapi tidak di kedua hari itu. K_i diestimasi dengan mencari nilai k sedemikian sehingga nilai $f(i,k)$ sekecil mungkin.

Untuk t cukup besar, estimasi nilai K_i sangat akurat sehingga kalkulasi K_i bisa berhenti dilakukan untuk menghemat *running time*.

2. Melakukan estimasi nilai $p[i,j]$

Berdasarkan estimasi K_i, $p[i,j]$ dihitung (tentunya setiap hari $p[i,j]$ dihitung agar memperoleh probabilitas yang lebih akurat) dengan menghitung banyaknya pesanan kue pada hari-hari yang sisa baginya dengan K_i adalah j, dibagi dengan banyak hari yang sisa baginya dengan K_i adalah j.

3. Menentukan kue-kue mana saja yang perlu diproduksi setiap harinya

Hal ini dilakukan untuk menjawab interaksi dari program juri. Berdasarkan hasil estimasi $p[i,j]$, ambil 10 kue yang estimasi probabilitasnya dipesan paling besar setiap harinya. Pasang mesin-mesin tersebut di pabrik.