COMPFEST 7

Senior Competitive Programming Contest

PEMBAHASAN CODERCLASS

WEEK 02

DAFTAR SOAL

BERTI SANG GALAU

Tag

matematika

Pembahasan

Perhatikan bahwa setiap bit dari N faktor keunikan itu independen terhadap bit lainnya dalam menentukan bit yang bersesuaian pada X. Sehingga, cukup dihitung berapa banyak konfigurasi biner yang jika semuanya di-XOR menghasilkan masing-masing bit pada X. Misalkan X_i adalah bit ke-i pada X. Jika $X_i = 0$ maka banyaknya bit 1 pada posisi ke-i dari ke-N bilangan tersebut haruslah genap. Sebaliknya, jika $X_i = 1$ maka banyaknya bit 1 pada posisi ke-i dari ke-N bilangan tersebut haruslah ganjil.

Pada kasus yang genap, banyaknya konfigurasi bit yang mungkin adalah:

$$\sum_{\substack{i=0 \\ i \text{ is even}}}^{N} \binom{N}{i}$$

Sedangkan pada kasus yang ganjil, banyaknya konfigurasi bit yang mungkin adalah:

$$\sum_{\substack{i=0 \\ i \text{ is odd}}}^{N} \binom{N}{i}$$

Perhatikan bahwa pada koefisien binomial, berlaku untuk seluruh $i, 0 \leq i \leq N$:

$$\binom{N}{i} = \binom{N}{N-i}$$

Sehingga jika N ganjil, berlaku:

$$\sum_{\substack{i=0 \\ i \text{ is even}}}^{N} \binom{N}{i} = \sum_{\substack{j=0 \\ j \text{ is odd}}}^{N} \binom{N}{j}$$

dan:

$$\sum_{\substack{i=0 \\ i \text{ is even}}}^{N} \binom{N}{i} + \sum_{\substack{j=0 \\ j \text{ is odd}}}^{N} \binom{N}{j} = 2^N$$

$$\sum_{\substack{i=0 \\ i \text{ is even}}}^{N} \binom{N}{i} = \sum_{\substack{j=0 \\ j \text{ is odd}}}^{N} \binom{N}{j} = 2^{N-1}$$

Artinya, tidak peduli X_i bernilai 1 atau 0, banyaknya kemungkinan ke-N bit agar hasil XOR-nya adalah X_i adalah 2^{N-1}. Karena terdapat K bit, maka total semua kemungkinan caranya adalah sebanyak $2^{(N-1)K}$.

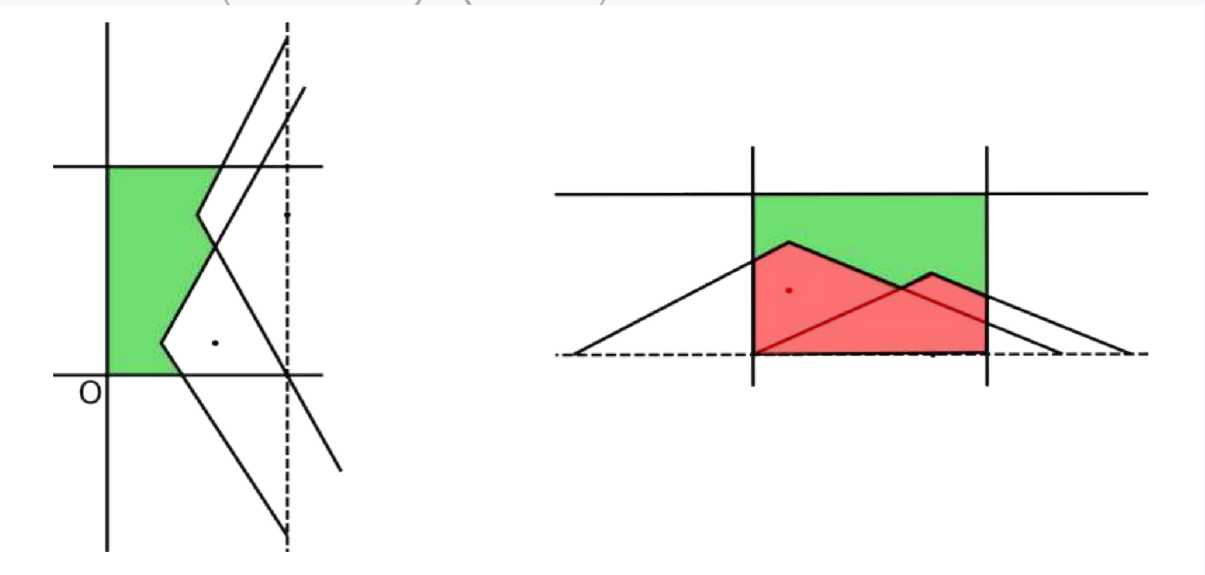
BOCAH PANTAI

Tag

geometri, *line sweep*

Pembahasan

Dari penjelasan contoh kasus, dapat dilihat bahwa inti persoalan ini adalah: Diberikan sejumlah segitiga sama kaki, tentukan luas gabungan daerah yang bukan termasuk daerah dalam segitiga-segitiga tersebut.



Terdapat banyak solusi untuk soal ini, dan solusi yang digunakan juri adalah: Hitung luas persegi panjang (daerah hijau dan daerah merah) dikurangi luas gabungan segitiga (daerah merah). Oleh karena itu, persoalan ini direduksi menjadi persoalan mencari luas gabungan segitiga yang berada pada batas-batas tertentu.

Karena semua segitiga yang terbentuk merupakan segitiga sama kaki, maka terdapat dua kelompok segitiga: yang menyumbang luas dan yang tidak.

Segitiga yang tidak menyumbang luas diberi warna merah. Ciri-ciri dari segitiga-segitiga yang tak menyumbang luas tersebut adalah kedua kaki segitiganya berada dalam kedua kaki suatu (satu buah) segitiga lain.

Solusi sederhananya adalah dengan membuang segitiga-segitiga yang tidak menyumbanh luas dalam $O(N^2)$. Perhitungan luas ini nantinya akan melibatkan pencarian titik potong dua segitiga, dan rumus luas trapesium.

Karena N bisa mencapai 10^5, diperlukan cara yang lebih cepat untuk membuang segitiga-segitiga yang tidak menyumbang luas. Kenyataannya, hal tersebut bisa dilakukan dengan cara mengurutkan segitiga-segitiga itu lalu memprosesnya dari kiri kenan. Dengan cara ini, segitiga-segitiga yang tidak berkontribusi luas dapat dideteksi cukup dengan memeriksa satu segitiga di samping kirinya (belakangnya). Bila segitiga tersebut tidak penting, yaitu bahwa kaki kanannya lebih kiri daripada satu segitiga sebelumnya, buang segitiga tersebut dari kumpulan segitiga yang ada. Dengan cara ini, dapat dijamin bahwa segitiga-segitiga yang tersisa (yang tidak dibuang) merupakan segitiga-segitiga "penting".

Kompleksitas akhir solusi adalah $O(N \log N + N)$ untuk pengurutan dan penghitungan luas, atau sama dengan $O(N \log N)$.

COKELAT BATANGAN

Tag

complete search, *bruteforce,* Dynamic Programming

Pembahasan

Observasi 1:

Perhatikan bahwa jika kita hanya tertarik pada pengecekan apakah jumlah dari sejumlah derajat kemanisan adalah genap atau ganjil. Oleh karena itu dari awal mengolah masukan, kita dapat mengubah setiap derajat kemanisan menjadi 0 jika genap dan 1 jika manis.

Observasi 2:

Kita dapat menggunakan DP *partial sum* dan memanfaatkan prinsip inklusi-eksklusi untuk mendapatkan informasi mengenai apakah jumlah derajat kemanisan suatu potongan cokelat dari suatu pojok kiri atas (r_0, c_0) hingga pojok kanan bawah (r_1, c_1) adalah genap atau ganjil.

Nilai-nilai dari hasil penghitungan DP tersebut pun dapat diubah menjadi 0 atau 1 juga, tergantung genap-ganjil-nya.

Contoh (dp[i][j] menyimpan jumlahan untuk semua petak cokelat di posisi (k, l) untuk $i \leq k \leq R$ dan $j \leq l \leq C$):

Derajat kemanisan awal:

```
1 0 1
0 1 1
```

dapat didapat tabel DP *partial sum*:

```
4 3 2
2 2 1
```

dan tabel DP dapat diubah menjadi:

```
0 1 0
0 0 1
```

Sekarang, bila misalkan soal diubah menjadi subsoal berikut: Diberikan sebuah array a[] yang dapat berisikan bilangan bulat berapapun, berapa banyak *continuous subsegment* a[i], a[i+1], …, a[j] sehingga nilai dari a[i] + a[i+1] + … + a[j] adalah genap?

Solusinya adalah dengan bekerja dari indeks 1 hingga N (sebut saja dengan indeks i), lalu untuk setiap indeksnya, cari pasangan indeks j yang kurang dari i dimana *parity* dari a[1] + a[2] + … + a[j] sama dengan [1] + a[2] + … + a[i] adalah sama.

Mengapa hal ini benar?

Anggap X = a[1] + a[2] + … + a[i] genap. Jika Y = a[1] + a[2] + … + a[j] genap juga, maka X – Y pasti bernilai genap juga. Hal tersebut juga berlaku untuk X dan Y sama-sama ganjil.

Implementasi naif untuk subsoal ini adalah $O(N^2)$,namun bisa di-*improve* menjadi $O(N)$ dengan cara berikut.

Saat bekerja dari indeks 1 hingga N (sebut menggunakan indeks i), kita menyimpan informasi mengenai banyaknya indeks yang telah diproses sehingga *parity*-nya genap atau ganjil. Perhatikan bahwa meskipun tidak terdapat indeks 0, namun kita juga harus memasukannya ke banyaknya indeks dengan *parity* genap, karena jumlah dari 0 elemen adalah 0 (genap).

Pseudocode solusi subsoal:

```
// input a[]
odd_sum <- 0
even_sum <- 1 // indeks 0 memiliki parity genap
ans <- 0

for i <- 1 .. N:
    if sum(1, i) is odd:
        ans <- ans + odd_sum // cari yang sama parity-nya
        odd_sum <- odd_sum + 1
    else if sum(1, i) is even:
        ans <- ans + even_sum
        even_sum <- even_sum + 1

print ans
```

Perhatikan bahwa sum(1, i) dapat dihitung dengan menggunakan DP *partial* sum sehingga kompleksitas dari sum(x, y) adalah $O(1)$ dan kompleksitas pseudocode di atas adalah $O(N)$.

Solusi Soal

Sekarang telah diketahui solusi subsoal dimana dalam permasalahannya, terdapat array satu dimensi, bukan dua dimensi. Bagaimana solusi jika yang diberikan adalah array dua dimensi seperti pada soal?

Kita dapat tetap memanfaatkan algoritme untuk subsoal di atas, namun kali ini kita menghitung untuk seluruh kemungkinan subbaris array tersebut yang berukuran $1 \times C, 2 \times C, \ldots, R \times C$.

Banyaknya seluruh kemungkinan subbaris tersebut adalah $\frac{R(R-1)}{2}$ dan kompleksitas untuk memproses suatu subbaris adalah $O(C)$, sehingga kompleksitas solusi adalah $O(R^2C)$.

LORONG BATU

Tag

greedy

Pembahasan

Observasi yang cukup penting adalah, bagaimanapun juga konfigurasi batu di dua kolom yang bersebelahan, batu-batu tersebut pasti bertetangga.

Oleh karena itu, kita dapat memproses pengambilan batu dari kiri ke kanan. Bila ditemukan batu di suatu kolom, ambil sekaligus satu batu yang bertetangga dengannya (bila ada). Bila ada lebih dari satu kemungkinan batu yang dapat diambil sekaligus, utamakan untuk mengambil yang berada di kolom yang sama dengan batu yang sedang diambil pada awalnya.

Kompleksitas solusi adalah $O(N)$.

MAIN BATU LAGI

Tag

sliding window, *sprague-grundy theorem*

Pembahasan

Sebelum membaca pembahasan soal ini, ada baiknya Anda mengerti mengenai materi berikut mengenai Impartial Games dan Sprague-Grundy Theorem:

http://www.gabrielnivasch.org/fun/combinatorial-games

http://www.gabrielnivasch.org/fun/combinatorial-games/sprague-grundy

Perhatikan bahwa permainan Batuman memenuhi syarat suatu game impartial, yaitu untuk posisi manapun, pemain manapun memiliki himpunan langkah-selanjutnya yang sama. Selain itu, permainan ini dapat dimodelkan menjadi suatu graf berarah asiklik (*directed acyclic graph*). Oleh karena itu, teorema Sprague-Grundy dapat diaplikasikan pada soal ini.

Mengapa bisa yakin bahwa langkah-langkah dalam permainan ini bersifat asiklik? Perhatikan bahwa terdapat batasan untuk setiap $i < j$, berlaku $a_i \leq a_j$ dan $b_i \leq b_j$. Hal tersebut berarti jika pada suatu saat suatu bidak berada pada posisi x dan melangkah ke posisi $y, x > y$, tidak mungkin pada suatu saat bidak tersebut pernah berada pada posisi x kembali.

Detail solusi sebagai berikut.

Asosiasikan nilai grundy (*grundy value*) untuk setiap petak permainan, sebut saja $g(pos)$ adalah nilai grundy untuk posisi petak pos. Seperti *impartial games* pada umumnya, berikan nilai $g(pos) = 0$ untuk $pos = 0$ (posisi tidak bisa bergerak lagi/*terminal position*). Perhatikan bahwa untuk mengetahui nilai dari $g(x)$, dibutuhkan informasi mengenai seluruh nilai $g(y)$ untuk $a_x \leq y \leq b_x$.

Pengisian/pencarian nilai $g(pos)$ biasanya dapat dilakukan menggunakan teknik Dynamic Programming dengan rekurens $g(pos) = \text{mex}(\{v \mid g(y) = v, y \text{ visitable from } pos\})$. Namun untuk soal ini, perhatikan bahwa banyaknya posisi y yang dapat dikunjungi dari suatu posisi pos bisa mencapai 10^5 posisi berbeda, sehingga jika

implementasinya menggunakan DP *top-down* biasa, akan mendapatkan putusan TLE (kompleksitas $O(N^2)$). Sehingga, dibutuhkan cara lebih efisien untuk mencari nilai $\text{mex}(\{v\})$.

Salah satu cara untuk mempercepat pencarian nilai mex tersebut adalah dengan menggunakan DP secara *bottom-up* ("mengisi" tabel memo dari kasus dasar) yang dilengkapi dengan teknik *sliding window* (atau yang juga dikenal sebagai *monotone queue*): http://abitofcs.blogspot.com/2014/11/data-structure-sliding-window-minimum.html. Teknik ini bisa digunakan karena terdapat properti $i < j \rightarrow a_i \leq a_j, b_i \leq b_j$ seperti yang telah dituliskan di atas.

Gambaran kasar penggunaan teknik *sliding window* adalah sebagai berikut. Anggap kita telah mengetahui nilai-nilai $g(pos)$ untuk $a_i \leq pos \leq b_i$ sehingga nilai $g(i)$ dapat diketahui. Informasi nilai-nilai tersebut bisa dimanfaatkan untuk mencari nilai-nilai $g(pos')$ untuk $a_{i+1} \leq pos' \leq b_{i+1}$ sehingga nilai $g(i+1)$ dapat diketahui. Bagaimana cara memanfaatkannya?

Saat kita menyimpan nilai-nilai $g(pos)$ untuk $a_i \leq pos \leq b_i$. Karena berlaku $a_i \leq a_{i+1}$ dan $b_i \leq b_{i+1}$, untuk mendapatkan nilai-nilai $g(pos')$ untuk $a_{i+1} \leq pos' \leq b_{i+1}$ cukup dengan melakukan *pop*/membuang nilai-nilai $g(x)$ untuk $a_i \leq x < a_{i+1}$ dan *push*/memasukkan nilai-nilai $g(y)$ untuk $b_i < y \leq b_{i+1}$. Setelah konfigurasi nilai-nilai yang disimpan telah sesuai dengan batasan a_{i+1} dan b_{i+1}, pencarian nilai mex dari nilai-nilai tersebut dapat dilakukan. Lakukan hal ini secara berurutan untuk setiap posisi $i \in [1, N]$.

Kompleksitas yang diharapkan untuk solusi soal ini adalah $O(N \cdot \log^2 N)$.

RANDOM GENERATOR

Tag

ad hoc, matematika

Pembahasan

Expected value atau nilai harapan memiliki rumus sebagai berikut:

$$\sum_{x \in S} x \cdot p(x)$$

dimana S adalah himpunan seluruh kejadian, dan $p(x)$ adalah fungsi peluang dari kejadian x. Dengan kata lain, solusi untuk soal ini dapat ditulis sebagai berikut:

$$\sum_{x=0}^{2^N-1} x \cdot p(x)$$

namun, rumus di atas tidak dapat secara polos diimplementasikan dikarenakan nilai N yang dapat mencapai 50 ($2^{50} \sim 10^{15}$).

Rumus di atas dapat dijabarkan sebagai berikut:

$$\sum_{x=0}^{2^N-1} \sum_{i=0}^{N-1} x_i \cdot 2^i \cdot p(x_i)$$

dimana x_i menyatakan bit ke-i dari bilangan bulat x (0 atau 1). Rumus di atas dapat dimanipulasi lebih lanjut:

$$\sum_{i=0}^{N-1} \sum_{x=0}^{2^N-1} x_i \cdot 2^i \cdot p(x_i)$$

mengapa manipulasi di atas penting? Perhatikan bahwa dalam menghitung rumus tersebut, kita dapat mengerjakannya per-bit secara independen, baru menjumlahkan seluruh hasilnya.

Observasi selanjutnya adalah, perhatikan bahwa untuk posisi bit manapun (sebut saja posisi bit ke-i), dari 0 hingga $2^N - 1$, pasti terdapat tepat 2^{N-1} bilangan yang memiliki bit 1

pada posisi tersebut dan terdapat tepat 2^{N-1} bilangan yang memiliki bit 0 pada posisi tersebut. Dengan observasi tersebut, rumus di atas dapat kembali ditingkatkan menjadi:

$$\sum_{i=0}^{N-1} 2^i \cdot 2^{N-1} \cdot \frac{P}{100}$$

$$\equiv 2^{N-1} \cdot \frac{P}{100} \sum_{i=0}^{N-1} 2^i$$

$$\equiv 2^{N-1} \cdot \frac{P}{100} \cdot (2^N - 1)$$

RANTAI CANTIK

Tag

graf

Pembahasan

Perhatikan bahwa kita hanya diminta untuk mencari tahu apakah sebuah graf yang diberikan merupakan rantai cantik atau bukan. Sehingga, perlu dicaritahu apa saja ciri-ciri dari sebuah rantai cantik.

Pertama, perhatikan bahwa semua mata rantai harus saling terhubung (graf harus terdiri tepat dari satu komponen terhubung). Algoritme *flood fill* dapat digunakan untuk mengecek syarat ini.

Kedua, karena rantai utama memiliki minimal dua mata rantai dan pada ujung dari rantai utama terhubung pada dua mata rantai berbeda, maka rantai cantik memiliki minimal enam mata rantai ($N \geq 6$).

Ketiga, perhatikan bahwa rantai tidak memiliki *cycle* (karena rantai utama hanya berbentuk garis lurus dan mata rantai yang terhubung dengan ujung rantai utama tidak terhubung dengan mata rantai yang lain). Karena graf hanya memiliki satu komponen (lihat poin pertama) dan tidak memiliki *cycle*, maka graf ini adalah sebuah pohon (*tree*) sehingga jumlah sambungan haruslah sebanyak $M = N - 1$.

Keempat, perhatikan bahwa hal-hal berikut harus terpenuhi:

- Ada empat mata rantai yang memiliki derajat satu (mata rantai yang terhubung dengan ujung rantai utama)
- Ada dua mata rantai yang memiliki derajat tiga (mata rantai pada ujung rantai utama)
- Ada $N - 6$ mata rantai yang memiliki derajat dua (mata rantai pada rantai utama namun bukan yang berada pada ujung rantai utama).

Kelima, perhatikan bahwa pada mata rantai yang memiliki derajat satu hanya terhubung dengan mata rantai pada ujung rantai utama (yang memiliki derajat tiga).

Jika graf memenuhi kelima persyaratan di atas, maka dapat dipastikan graf adalah sebuah rantai cantik.

TRUEL

Tag

graf

Pembahasan

Perhatikan bahwa relasi mendominasi-didominasi dalam soal ini dapat dipetakan ke dalam sebuah *tournament graph*. Sehingga, permasalahan dalam soal ini dapat direduksi menjadi mencari tahu apakah terdapat setidaknya sebuah *cycle* dengan panjang 3 (untuk selanjutnya akan disebut sebagai *triangle*) pada graf tersebut.

Pencarian sebuah *triangle* pada graf paling cepat berjalan dalam kompleksitas $\frac{N^3}{64}$, dan itu pun masih mendapat putusan TLE jika digunakan untuk menjawab permasalahan ini.

Lemma 1

Jika terdapat sebuah *cycle* dengan panjang lebih dari tiga pada suatu *tournament graph*, maka pada graf tersebut dapat ditemukan pula *cycle* dengan panjang tepat tiga.

Bukti lemma 1

Anggap dalam graf tersebut terdapat *cycle* terpendek C (*cycle* terpendek berarti *cycle* dengan banyak *vertex* tersedikit). Apabila C memiliki panjang tiga, maka telah ditemukan *cycle* dengan panjang tiga. Namun, apabila $|C| > 3$, anggap terdapat tiga buah *vertex* x, y, z pada cycle tersebut sehingga $x \to y \to z$ merupakan salah satu *path* dari *cycle* tersebut.

Karena graf merupakan *tournament graph*, antara x dan z pasti terdapat suatu *edge*/sisi. Apabila sisi tersebut berbentuk $z \to x$, maka C bukan merupakan *cycle* terpendek (karena terdapat *cycle* $x \to y \to z \to x$). Apabila sisi tersebut berbentuk $x \to z$, maka C juga bukan merupakan *cycle* terpendek karena sisi $x \to z$ dan sisi-sisi lainnya dalam C selain yang melewati y membentuk *cycle* lain yang panjangnya tepat lebih pendek satu path dari C sendiri.

Oleh karena itu, dapat dideduksi bahwa tidak mungkin terdapat suatu *cycle* terpendek C dalam suatu graf turnamen sehingga $|C| > 3$. Perlu diperhatikan juga bahwa $|C|$ tidak

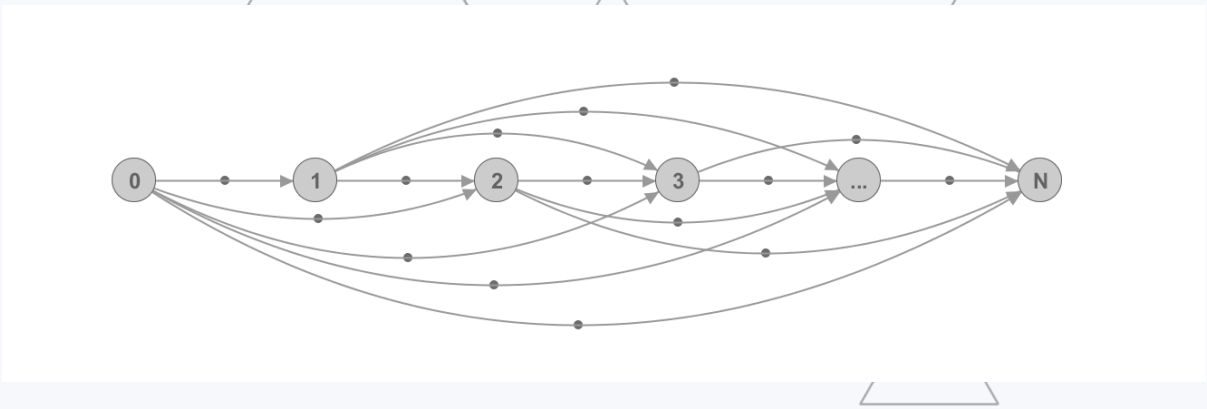
mungkin satu atau dua karena graf turnamen tidak mengizinkan adanya *self-loop* atau *cycle* dalam bentuk $x \rightarrow y \rightarrow x$.

Dari **lemma 1**, dapat ditarik kesimpulan lagi bahwa permasalahan soal ini dapat direduksi kembali menjadi mencari setidaknya satu *cycle* (panjang berapapun) dari graf yang diberikan.

Lalu, bagaimana mencari *cycle*-panjang-berapapun dari graf yang diberikan?

Permasalahan ini jawabannya dapat dicari dengan mencari kebenaran dari komplemennya: Apakah tidak terdapat *cycle* pada graf yang diberikan?

Perhatikan bahwa graf turnamen tidak mengandung *cycle* apabila himpunan nilai-nilai *in-degree* dari *vertex-vertex*-nya ekivalen dengan $\{0, 1, 2, \ldots, |V| - 1\}$ dengan V adalah himpunan *vertex* dan $|V|$ menyatakan banyaknya *vertex*, atau dengan kata lain, graf turnamen tersebut juga merupakan *Directed Acyclic Graph (DAG)*. Hal ini benar karena untuk suatu graf turnamen, hanya ada satu bentuk yang merupakan graf asiklis, yaitu seperti berikut:



Atau dengan kata lain, untuk suatu graf turnamen yang asiklis dengan banyak *vertex* $|V|$:

- Terdapat tepat satu *vertex* v_0 yang memiliki *in-degree* 0: Untuk semua *vertex* $v_i \neq v_0$, terdapat *edge* $v_0 \rightarrow v_i$
- Terdapat tepat satu *vertex* v_1 yang memiliki *in-degree* 1: Untuk semua *vertex* $v_i \neq v_1, v_i \neq v_0$, terdapat *edge* $v_1 \rightarrow v_i$

- Dan seterusnya, hingga *vertex* v_n yang memiliki *in-degree* sebesar $|V| - 1$.

Pengecekean nilai *in-degree* dari setiap *vertex* ini dapat dilakukan dalam waktu $O(E)$ atau dengan kata lain $O(V^2)$.