PROBLEM ARRANGE

Nama: Andika Rahman Teja

NRP: 5025221022

Mata Kuliah: Pemrograman Berbasis Objek

Kelas: C

Pada problem ini, kita akan mencari kombinasi susunan barisan untuk **m** siswa. Pada **m** siswa tersebut, terdapat sejumlah **n** siswa perempuan yang mana tidak boleh ada 2 siswa perempuan berdampingan. Pada *input* data, akan terdapat variabel *testcase* **t** dengan *constraint* ($1 \leq \boldsymbol{t} \leq 10^5$) dan setiap **t** baris akan terdapat variabel **m** dan **n** dengan *constraint* ($1 < \boldsymbol{n} \leq \boldsymbol{m} \leq 10^6$). Sementara pada *output*, akan melakukan *print* banyak kombinasi susunan untuk **m** siswa dan **n** siswa perempuan pada tiap *testcase* yang telah di-*modulo* $10^9 + 7$.

Karena pada soal diketahui nilai maksimal **n** dan **m** adalah 10^6 dan terdiri dari beberapa *testcase*, saya akhirnya membuat *array* bertipe data *long long* (jika menggunakan *int*, akan terjadi *stack overflow* pada *array*) yang bernilai faktorial dari 0 hingga 10^6 yang juga sudah di-*modulo* dengan $10^9 + 7$. Saya melakukan inisialisasi nilai faktorial pada *array* menggunakan pendekatan *dynamic programming* yaitu *bottom-up* dengan inisialisasi awal pada indeks ke-0 adalah 1. Kemudian saya melakukan deklarasi dan memberikan fungsi *scanf* agar dapat melakukan input pada variabel **t** sebagai jumlah *testcase*.

```
int main()
{
    // vector<ll> fact(1000001);
    ll fact[1000001];
    fact[0] = 1;
    for (int i = 1; i <= 1000000; i++)
    {
        fact[i] = (i * fact[i - 1]) % mod;
    }
    int t;
    scanf("%d", &t);
```

Gambar 1

Selanjutnya, saya membuat iterasi untuk setiap *testcase* dengan menggunakan *while*. Pada tiap iterasi, saya mendeklarasikan **m** dan **n** bertipe data *int* dan memanggil fungsi *scanf* untuk dapat memberi input pada kedua variabel tersebut. Selanjutnya saya memiliki teori jika nilai **m** sekurang-kurangnya adalah $2\boldsymbol{n} - 1$ karena jika tidak, akan ada setidaknya 1 pasang siswa perempuan yang keduanya berdampingan posisinya sehingga otomatis tidak ada kombinasi barisan yang sesuai kriteria. Jika nilai **m** lebih dari sama dengan $2\boldsymbol{n} - 1$ maka akan dilakukan operasi *Modular Combinatorics* seperti di bawah ini.

```
    while (t--)
    {
        int m, n;
        scanf("%d %d", &m, &n);
        if (m < 2 * n - 1)
        {
            printf("0\n");
        }
        else
        {
            ll comb = nCk(m - n + 1, n, mod, fact);
            printf("%lld\n", (((comb * fact[n]) % mod) * fact[m - n]) % mod);
        }
    }
```

Gambar 2

Untuk dapat menemukan kombinasi barisan, saya akan mengalikan hasil faktorial **n** (untuk merepresentasikan banyak kombinasi barisan siswa perempuan) dan hasil faktorial **m-n** (untuk merepresentasikan banyak kombinasi barisan siswa laki-laki) serta kombinasi C_n^{m-n+1}. Rumus C_n^{m-n+1} terjadi karena untuk menyusun barisan laki-laki tiap orangnya dengan barisan perempuan yang diumpamakan satu kesatuan. Karena ada *modulo*, maka sesuai teori *modulo*, setiap selesai mengalikan 2 bilangan akan langsung di *modulo*.

Di sini, saya akan menjabarkan rumus kombinasi **nCk** sebagai berikut:

- Mengubah rumus kombinasi menjadi bentuk berikut

$$^nC_k = n! \cdot \frac{1}{k!} \cdot \frac{1}{(n-k)!}$$

- Karena *modulo* berlaku aturan $\frac{a}{b} \neq \frac{a \bmod p}{b \bmod p}$, maka digunakan operasi *modular multiplicative inverse*

$$^nC_k \equiv n! \cdot \text{inv}(k!) \cdot \text{inv}((n-k)!) \pmod{p}$$

$$^nC_k \mod p = n! \mod p \cdot \text{inv}(k!) \mod p \cdot \text{inv}((n-k)!) \mod p$$

- Karena $10^9 + 7$ adalah bilangan prima, maka kita bisa menghitung nilai *invers modulo*, digunakan teori *Fermat Little Theorem* [Jika pada kode, akan merujuk pada fungsi **inv**]

$$a^p \equiv a \pmod{p}$$

$$a^{p-1} \equiv 1 \pmod{p}$$

$$a \cdot \text{inv}(a) \equiv a^{p-1} \pmod{p}$$

$$\text{inv}(a) \equiv a^{p-2} \pmod{p}$$

- Karena pada *Fermat Little Theorem* terdapat operasi perpangkatan, maka dapat diterapkan *modular exponentiation* untuk mencegah *integer overflow* [Jika pada kode, akan merujuk pada fungsi **powmod**]

$$a^b \bmod p = (a \bmod p)^b \bmod p$$

- Untuk dapat mempercepat proses perpangkatan, maka jika pangkat bernilai ganjil, nilai **a** akan dikalikan pada variabel **product** (variabel yang menjadi nilai hasil *modular exponentiation*) lalu di-*modulo* dan nilai **b** akan dikurangi 1 seperti berikut

$$a^b = a^{b-1} \cdot a$$

- Sementara jika pangkat bernilai genap, nilai **a** akan langsung dikuadratkan lalu di-*modulo* dan nilai **b** akan langsung dibagi 2 seperti berikut

$$a^b = (a^2)^{\frac{b}{2}}$$

Selain itu, tidak lupa juga untuk setiap melakukan operasi perkalian pada fungsi **nCk** akan langsung di-*modulo* dengan $10^9 + 7$

```cpp
#define ll long long
#define mod 1000000007

ll powmod(ll a, ll b, ll p)
{
    a %= p;
    if (a == 0)
        return 0;
    ll product = 1;
    while (b > 0)
    {
        if (b & 1)
        {
            product *= a;
            product %= p;
            --b;
        }
        a *= a;
        a %= p;
        b /= 2;
    }
    return product;
}

ll inv(ll a, ll p)
{
    return powmod(a, p - 2, p);
}

ll nCk(ll n, ll k, ll p, ll fact[])
{
    ll numerator = fact[n];
    ll denominator = (fact[k] * fact[n - k]) % p;
    return (numerator * inv(denominator, p)) % p;
}

```

Gambar 3

Bukti Submisi Revisi:

14572463	andika_c_022	Oct 6, 2023, 5:06:06 AM	57 ms	7792 KB	C++ 20 (gnu 10.2)

Referensi:

- https://cplusplus.com/forum/general/19903/
- Kenneth H. Rosen. 2012. Discrete Mathematics and Its Applications (7th. ed.). McGraw-Hill Higher Education.

Foto Kode Keseluruhan:

```cpp
#include <stdio.h>
// #include <vector>
// using namespace std;

#define ll long long
#define mod 1000000007

ll powmod(ll a, ll b, ll p)
{
    a %= p;
    if (a == 0)
        return 0;
    ll product = 1;
    while (b > 0)
    {
        if (b & 1)
        {
            product *= a;
            product %= p;
            --b;
        }
        a *= a;
        a %= p;
        b /= 2;
    }
    return product;
}

ll inv(ll a, ll p)
{
    return powmod(a, p - 2, p);
}

ll nCk(ll n, ll k, ll p, ll fact[])
{
    ll numerator = fact[n];
    ll denominator = (fact[k] * fact[n - k]) % p;
    return (numerator * inv(denominator, p)) % p;
}

int main()
{
    // vector<ll> fact(1000001);
    ll fact[1000001];
    fact[0] = 1;
    for (int i = 1; i <= 1000000; i++)
    {
        fact[i] = (i * fact[i - 1]) % mod;
    }
    int t;
    scanf("%d", &t);
    while (t--)
    {
        int m, n;
        scanf("%d %d", &m, &n);
        if (m < 2 * n - 1)
        {
            printf("0\n");
        }
        else
        {
            ll comb = nCk(m - n + 1, n, mod, fact);
            printf("%lld\n", (((comb * fact[n]) % mod) * fact[m - n]) % mod);
        }
    }
    return 0;
}
```

Gambar 4

PROBLEM LOTTERY

Nama: Andika Rahman Teja

NRP: 5025221022

Mata Kuliah: Pemrograman Berbasis Objek

Kelas: C

Pada problem ini, disuruh mencari angka terkecil ke- **k** pada setiap **c** angka yang diberikan. Pada *input* baris pertama akan diberikan sejumlah *testcase*. Pada setiap *testase*, akan diawali oleh 2 angka yakni **c** dengan *constraint* ($1 \leq c \leq 500000$) dan **k** dengan *constraint* ($1 \leq k \leq \min(c,100000)$). Kemudian, **c** baris berikutnya adalah angka-angka dengan *constraint* dari 1 hingga 1 milyar yang semuanya bersifat *unique* (tidak ada duplikasi angka) dan jika terdapat angka 0, maka *print* angka terkecil ke- **k**.

Semula, saya melakukan deklarasi variabel **t** untuk banyak *testcase*, **c** untuk banyak angka-angka yang akan dicari nilai terkecil ke- **k**, **number** untuk menampung angka-angka yang yang akan dicari nilai terkecil ke- **k**, **res** sebagai angka terkecil ke- **k** setelah dilakukan algoritma yang ada, dan **kth** adalah urutan nilai terkecil (atau nilai **k**). Di sini, saya menggunakan fungsi **getnum** untuk mempercepat proses *input* ke variabel (untuk lebih lengkap, akan saya jelaskan di akhir laporan). Kemudian, saya membuat iterasi sebagai representasi tiap *testcase* untuk melakukan *input* nilai **c** dan **kth**. Selain itu, saya menggunakan *abstract data type* berupa *priority queue* (sebelumnya saya sudah memanggil *library queue*) dengan nama variabel **numlist** yang nantinya digunakan untuk menyimpan nilai dari **number**. *Priority queue* ini tetap dibiarkan untuk melakukan *sorting* secara *descending* (dari nilai besar ke nilai terkecil). Dengan menggunakan *priority queue*, akan sangat menghemat waktu kompilasi jika dibandingkan hanya menggunakan *array* atau *abstract data type* seperti *vector* yang kemudian di-*sorting* setiap melakukan input **number**.

Gambar 5

Selanjutnya, dibuat iterasi sebagai representasi tiap **c**. Pada iterasi tersebut, akan dilakukan *input* variabel **number** dengan fungsi **getnum**. Seperti pada keterangan sebelumnya, jika **number** sama dengan 0, maka akan mencetak nilai terkecil ke- **k** atau nilai *top* dari *priority queue*. Namun perlu diperhatikan jika ternyata nilai terkecil ke- **k** tersebut tidak ada karena mengakses nilai yang melebihi ukuran dari *priority queue* tersebut (misal ukuran *priority queue* adalah 3, namun disuruh mencari nilai terkecil ke-5), maka akan mencetak nilai -1. Hal inilah yang terjadi pada *line* 51-55.

Jika ternyata **number** tidak bernilai 0, maka akan ada 2 *conditional*, yakni:

- Jika ukuran *priority queue* kurang dari nilai **kth**, maka **number** akan di-*push* ke dalam *priority queue*.
- Jika ukuran *priority queue* sama dengan nilai **kth** dan nilai **number** lebih lebih kecil dibanding nilai terkecil ke- **k** atau nilai *top* dari *priority queue*, maka akan dilakukan *pop* pada *priority queue* terlebih dahulu dan kemudian dilakukan *push* **number**.

```
45          for (int j = 0; j < c; j++)
46          {
47              // scanf("%d", &number);
48              number = getnum();
49              if (number == 0)
50              {
51                  if (kth > numlist.size())
52                      res = -1;
53                  else
54                      res = numlist.top();
55                  printf("%d\n", res);
56              }
57              else
58              {
59                  if (numlist.size() < kth)
60                      numlist.push(number);
61                  else if (numlist.size() == kth && number < numlist.top())
62                  {
63                      numlist.pop();
64                      numlist.push(number);
65                  }
66              }
67          }
68      }
69  }
```

Gambar 6

Berikut ini langkah kerja algoritma pada Gambar 6 jika **c** = 7 dan **kth** = 2:

Iterasi 1 (input = 4711) [kondisi *line* = 59]

Numlist	
4711	

Iterasi 2 (output = -1) [kondisi *line* = 51]

Numlist	
4711	

Iterasi 3 (input = 4) [kondisi *line* = 59]

Numlist	
4711	4

Iterasi 4 (output = 4711) [kondisi *line* = 53]

Numlist	
4711	4

Iterasi 5 (input = 210706) [kondisi *line* = tidak ada, tidak perlu di-*push*]

Numlist	
4711	4

Iterasi 6 (input =3) [kondisi *line* = 61]

Numlist	
4	3

Iterasi 7 (output = 4) [kondisi *line* = 53]

Numlist	
4	3

Sebelum ada kode ini, saya sempat berpikir untuk menggunakan *priority queue* yang di-*sorting* secara *ascending* (dari nilai kecil ke nilai terbesar). Adapun potongan kode yang ingin saya pakai adalah sebagai berikut:

Gambar 7

Namun setelah saya analisis lebih dalam, pada *priority queue* yang *ascending* akan membutuhkan sebuah iterasi tambahan untuk melakukan *pop* hingga kondisi *top* pada *priority queue* adalah nilai terkecil ke- **k**. Selain itu, dengan *priority queue* yang *ascending* tentunya membutuhkan alokasi memori yang lebih banyak untuk menampung **number** (*worst case* bisa saja menampung hingga 500000 angka). Sedangkan pada *priority queue* yang *descending*, tidak perlu memerlukan iterasi tambahan untuk mengakses nilai terkecil ke- **k** dan maksimal kapasitas memorinya *worst case* nya menampung hingga 100000 angka.

```cpp
#include <cstdio>
#include <cstring>
#include <queue>
using namespace std;

int getnum()
{
    int res = 0;
    char c;
    int b = 0;
    while (1)
    {
        c = getchar_unlocked();
        if (c == '-')
            b = 1;
        if (c == ' ' || c == '\n')
            continue;
        else
            break;
    }
    if (c != '-')
        res = c - '0';
    while (1)
    {
        c = getchar_unlocked();
        if (c >= '0' && c <= '9')
            res = 10 * res + c - '0';
        else break;
    }
    if (b == 1)res *= -1;
    return res;
}
```

Gambar 8

Di sini, saya akan menjelaskan fungsi **getnum**. Pada fungsi ini, pertama akan dilakukan:

- Deklarasi dan inisialisasi variabel **res** dan **b** dengan keduanya memiliki tipe data *int* yang bernilai 0
- Deklarasi variabel **c** dengan tipe data *char*

Selanjutnya akan dilakukan iterasi dengan *while true* untuk melakukan *input* pada variabel **c** dengan menggunakan "*getchar_unlocked*". *getchar_unlocked* adalah sebuah cara untuk melakukan *input* untuk membaca karakter (*char*) hingga *end of file* atau EOF. Jika dalam karakter

tersebut terdapat tanda negatif (‘-’), maka variabel **b** akan menjadi 1 dan menunjukkan bahwa selama input, angka tersebut adalah bilangan negatif. Selama iterasi tersebut pula, *whitespace* (‘ ’) atau *newline* (‘/n’), akan dilewati. Iterasi ini berguna untuk memastikan apakah bilangan tersebut positif atau negatif dan perlu *crosscheck* ulang tanda tersebut sesuai *line* 21.

Selanjutnya, akan dilakukan iterasi dengan *while true* untuk mendapatkan input yang nantinya berupa angka. Sama seperti iterasi sebelumnya, iterasi ini akan melakukan *input* pada variabel **c** dengan menggunakan *getchar_unlocked*. Jika *input* yang diterima adalah angka 0 hingga 9 (namun dalam bentuk *char*), maka bentuk *char* tersebut akan dikonversi ke bentuk *int* yang ditampung pada variabel **res**. Terdapat keunikan disini dimana saat dilakukan konversi ke *int*, urutan angka yang diterima yang semula menempati “satuan” akan menjadi “puluhan” kemudian “ratusan” dan seterusnya dengan mengalikan **res** dengan 10 terlebih dahulu kemudian ditambahkan dengan angka pada **c**. Jika tidak ada angka lagi, maka iterasi selesai dan hanya perlu membuat bilangan tersebut positif jika nilai **b** sama dengan 0 atau bilangan negatif jika nilai **b** sama dengan 1.

Fungsi **getnum** ini menerapkan proses *Mutual Exclusion* (Mutex) yang memfokuskan *thread* atau proses eksekusi program pada proses *input output* pada saat fungsi ini dipanggil. Hal ini bisa sangat mengefisienkan kode program jika diterapkan dengan baik (secara kebetulan dapat diimplementasikan pada kode ini yang semua *input* nya adalah angka). Hal ini terbukti pada submisi pertama saya yang menggunakan *scanf*, waktu kompilasinya adalah 0.15s yang kemudian menjadi 0.06s pada submisi kedua dengan menggunakan **getnum**.

Bukti submisi:

ID		DATE	PROBLEM	RESULT	TIME	MEM	LANG
31966143	☐	2023-10-06 05:29:36	Tickets lottery	**accepted** edit ideone it	0.06	5.3M	CPP14
31966114	☐	2023-10-06 05:27:12	Tickets lottery	compilation error edit ideone it	-	-	CPP14
31966059	☐	2023-10-06 05:19:16	Tickets lottery	**accepted** edit ideone it	0.15	5.4M	CPP14

Referensi:

- https://en.cppreference.com/w/cpp/container/priority_queue
- https://www.geeksforgeeks.org/getchar_unlocked-faster-input-cc-competitive-programming/
- https://en.cppreference.com/w/cpp/thread/mutex

Foto Kode Keseluruhan:

```cpp
#include <cstdio>
#include <cstring>
#include <queue>
using namespace std;

int getnum()
{
    int res = 0;
    char c;
    int b = 0;
    while (1)
    {
        c = getchar_unlocked();
        if (c == '-')
            b = 1;
        if (c == ' ' || c == '\n')
            continue;
        else
            break;
    }
    if (c != '-')
        res = c - '0';
    while (1)
    {
        c = getchar_unlocked();
        if (c >= '0' && c <= '9')
            res = 10 * res + c - '0';
        else break;
    }
    if (b == 1)res *= -1;
    return res;
}

int main()
{
    int t, c, number, res, kth;
    // scanf("%d", &t);
    t = getnum();
    for (int i = 0; i < t; i++)
    {
        priority_queue<int> numlist;
        // scanf("%d %d", &c, &kth);
        c = getnum();
        kth = getnum();
        for (int j = 0; j < c; j++)
        {
            // scanf("%d", &number);
            number = getnum();
            if (number == 0)
            {
                if (kth > numlist.size())
                    res = -1;
                else
                    res = numlist.top();
                printf("%d\n", res);
            }
            else
            {
                if (numlist.size() < kth)
                    numlist.push(number);
                else if (numlist.size() == kth && number < numlist.top())
                {
                    numlist.pop();
                    numlist.push(number);
                }
            }
        }
    }
}
```

Gambar 9