BACHELOR OF COMPUTER SCIENCE
SCHOOL OF COMPUTER SCIENCE
BINA NUSANTARA UNIVERSITY
JAKARTA

ASSESSMENT FORM

Course: COMP6049001 - Algorithm Design and Analysis

Method of Assessment: Case Study

Semester/Academic Year : 3/2023-2024

Name of Lecturer : Islam Nur Alam, S.Kom., M.Kom.

Date : Senin, 8 Januari 2024

Class : LJ01

Topic : Review II

Group Name :	**bebekijo**
Group Members :	**1. Leonardo Dahendra (2602097076) – Ketua** **2. Tiara Intan Kusuma (2602172220) – Anggota** **3. Fendy Wijaya (2602092150) – Anggota**

Student Outcomes:
(SO 1) Mampu menganalisis masalah komputasi yang kompleks dan mengaplikasikan prinsip komputasi dan keilmuan lain yang sesuain untuk mengidentifikasi solusi
Able to analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions

(SO 2) Mampu merancang, mengimplementasikan, dan mengevaluasi solusi berbasis komputasi untuk memenuhi serangkaian persyaratan komputasi dalam konteks ilmu computer

Able to design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of computer science.

Learning Objectives:
(LObj 1.1) Mampu menganalisis masalah komputasi yang kompleks
Able to analyze a complex computing problem

(LObj 2.3) Mampu mengevaluasi solusi berbasis komputasi untuk memenuhi serangkaian persyaratan komputasi tertentu dalam konteks ilmu komputer
Able to evaluate a computing-based solution to meet a given set of computing requirements in the context of computer science

No	Assessment criteria	Weight	Excellent (85 - 100)	Good (75-84)	Average (65-74)	Poor (0 - 64)	Score	(Score x Weight)
1	Ability to solve problems in INC 2023 during the event.	**50%**	Able to solve > 3 problems in INC 2023 during the event.	Able to solve 3 problems in INC 2023 during the event.	Able to solve 2 problems in INC 2023 during the event.	Able to solve 1 problem in INC 2023 during the event.		
2	Ability to write an analysis of the solutions used in INC 2023.	**30%**	Able to explain the algorithms for solved problems and provide a rigorous proof of the correctness of the algorithm.	Able to explain the algorithms for solved problems and provide the thought process of why the algorithms should solved the problems.	Able to explain the algorithms for solved problems.	Able to provide the source code for the solved problems.		
3	Ability to upsolve the unsolved problems in INC 2023 after the event.	**20%**	Able to explain the correct solution for at least 1 unsolved problem (if exist) and provide the source code. Report on why your group cannot	Able to explain the correct solution for at least 1 unsolved problem (if exist). Report on why your group cannot solve	Explain on why your group cannot solve the problem during event. Explain your attempt during the	Explain on why your group cannot solve the problem during event.		

No	Assessment criteria	Weight	Excellent (85 - 100)	Good (75-84)	Average (65-74)	Poor (0 - 64)	Score	(Score x Weight)
			solve the problem during event.	the problem during event.	event and why it failed.			
	Total Score: ∑(Score x Weight)							

Remarks:

ASSESSMENT METHOD

Instructions

Before INC 2023

1. Students form a group consisting of 3 people.
2. Follow the provided instruction to register to INC 2023.

During INC 2023

1. Follow the schedule and rules of INC 2023.
2. Solve as many problems as your team could solve. The number of solved problems will be used in assessment criteria 1.

After INC 2023

1. Write a problem analysis (editorial) for the problems you solved. If you solve > 3 problems during INC, pick at least 3 of the solved problems. The way you analyze and explain your solution will be used in assessment criteria 2.
2. Take your time to keep practicing by solving the unsolved problems during INC 2023. You are allowed to discuss the problems with other groups, but your group must show that you understand the solution and be able to solve the problems. If you upsolve > 1 problems, pick at least 1 of the upsolved problems. The way you explain the solution will be used in assessment criteria 3.
   Note: if you solved all problems during the event, you don't need to do anything for assessment criteria 3.

Project Output

- Your participation in INC 2023
- Problem analysis (editorial) for the solved problems during INC 2023
- Explanation (and source code) of the upsolved problems after INC 2023

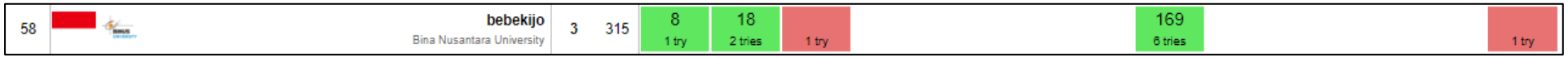
Note for Lecturers:

1. Students submit their project output in the last week of lecture
2. Please verify the project output with the INC 2023 scoreboard

PEMBAHASAN

Kami dari kelompok *bebekijo* kelas LJ01, pada penugasan AoL ini akan membahas dan meng-*upsolve* beberapa soal yang terdapat pada INC 2023 sebagai salah satu bentuk output projek AoL mata kuliah COMP6049001 – Algorithm Design and Analysis. Soal yang berhasil kami kerjakan dan tertanda warna hijau adalah soal A, B, dan H. Sementara untuk soal yang kami akan *upsolve* hanya 1 saja, karena kelompok kami tidak terdiskualifikasi. Kami memilih soal E untuk di-*upsolve*. Pertama kami akan menganalisis soal A, B, dan H. Hal yang akan kami bahas adalah analisis soalnya sendiri, analisis solusi yang dapat digunakan dan tidak dapat digunakan, serta analisis kodingan hasil akhir yang benar dan mengapa kodingan ini bisa benar.

Sebelum masuk ke pembahasan soal, berikut ini adalah tangkapan layar bukti bahwa kelompok kami berhasil menyelesaikan 3 soal pada saat pelaksanaan INC 2023 dan tidak terdiskualifikasi.



Berikut ini adalah solusi kode program dan analisis algoritma dalam kode program untuk soal-soal yang berhasil dikerjakan pada saat pelaksanaan INC 2023, yaitu **Problem A, B, dan H**.

PROBLEM A – GOLDEN TICKETS

Analisis Masalah pada Soal:

Pada soal dinyatakan, jika diberikan N kelompok, setiap kelompok S_i dari institusi T_i dengan ranking i, M kelompok teratas akan lolos ke ICPC, kemudian maksimal K kelompok dari institusi berbeda yang tidak lolos, akan diberikan golden ticket yang memungkinkan maksimal 1 kelompok dari institusi itu untuk ikut ICPC. Dan jika terdapat lebih dari 1 kelompok dari institusi tersebut, maka ranking paling tinggi akan dipilih. Soal meminta jika diberikan data tersebut, print jumlah kelompok yang mendapatkan golden ticket dan nama setiap kelompok tersebut.

Dari soal, jika kita perhatikan lebih dalam, inti pertanyaannya adalah, print maksimal K kelompok teratas yang tidak lolos ICPC dan tidak ada kelompok lain dari institusinya yang lolos. Untuk mengetahui metode yang dapat kita pakai, pertama kita akan menganalisis constraint yang diberikan pada soal, diketahui bahkan terdapat maksimal 100 kelompok yang diberikan, 100 kelompok yang lolos ke ICPC, dan 100 kelompok yang

mendapatkan golden ticket. Selain itu, panjang nama kelompok dan institusi maksimal 10 huruf, tapi untuk soal ini, panjangnya tidak terlalu berpengaruh maka dapat kita abaikan. Hal paling penting yang perlu kita perhatikan di sini adalah jumlah kelompok yang diberikan dan jumlah kelompok yang mendapatkan golden ticket. Untuk mengetahui apakah sebuah kelompok bisa mendapatkan golden ticket, maka kita perlu mengecek semua kelompok yang sudah lolos untuk mengetahui apakah institusinya sudah masuk. Jumlah pengecekan maksimalnya adalah M, namun karena $M \leq N$, untuk mempermudah, kita asumsikan saja N, yaitu worst case-nya. Kita juga akan memerlukan looping keseluruhan kelompok yang diberikan, baru kemudian looping untuk mengecek apakah kelompok tersebut bisa mendapatkan golden tiket. Maka dari itu, worst case time-nya adalah $N * N$ atau $100 * 100 = 10.000$.

Dalam competitive programming, time limit biasa di-set 1 detik, dan dalam 1 detik, biasanya kita dapat melakukan loopingan sekitar 10 juta kali. Karena 10 ribu jauh dibawah 10 juta, maka soal ini dapat kita selesaikan dengan mudah menggunakan metode **brute force**. Untuk soal ini, kita tidak perlu memikirkan solusi yang lebih cerdas karena constraintnya kecil.

Kode Program:

Kodingan yang benar untuk menyelesaikan permasalahan ini adalah sebagai berikut. Kodingan menggunakan bahasa pemrograman C++.

```
#include<iostream>
#include<bits/stdc++.h>
#include<vector>
#include<string>
#include<math.h>

#define ll long long

using namespace std;

int main(){
	int n, m, k;
	cin >> n >> m >> k;
	string ins2[n];
	string ans[n];
	int curr = 0, ansCurr = 0;
```

```
	for (int i = 0; i < n; i++){
		string team, ins;
		cin >> team >> ins;
		if (i < m) ins2[curr++] = ins;
		else{
			bool can = 1;
			for (int j = 0; j < curr; j++){
				if (ins == ins2[j]){
					can = 0;
					break;
				}
			}
			if (can && ansCurr < k){
				ans[ansCurr++] = team;
				ins2[curr++] = ins;
			}
		}
	}
	cout << ansCurr << endl;
	for (int i = 0; i < ansCurr; i++){
		cout << ans[i] << endl;
	}
	return 0;
}
```

Analisis Solusi dari Kode Program:

Pada soal diberitahukan bahwa pertama akan diberikan 3 integer dalam range 1 sampai 100, maka kita dapat mendeklarasi integer biasa untuk menyimpan ketiga angka ini. Kemudian kita ambil input untuk ketiga integer ini, yaitu n, m, dan k. Kemudian kami membuat dua array string, yaitu `ins2` dengan ukuran n, dan juga `ans` dengan ukuran n. `ins2` ini akan digunakan untuk menampung nama semua institusi di mana terdapat tim dari institusi tersebut yang lolos ke ICPC, baik melalui top ranking maupun melalui golden ticket. Sedangkan `ans` akan digunakan untuk menyimpan jawaban akhir, yaitu tim yang mendapatkan golden ticket. Kedua array ini diberikan ukuran n karena maksimal jumlah institusi

yang lolos pada worst case adalah setara dengan jumlah tim, atau dengan kata lain, setiap tim berasal dari institusi yang berbeda, begitu juga dengan jumlah tim yang mendapatkan golden tiket, di mana setiap tim berasal dari institusi yang berbeda dan jumlah kelompok yang mendapatkan golden tiket setara dengan jumlah kelompok yang ada.

Setelah itu, kami mendeklarasikan 2 integer, yaitu `curr` dan `ansCurr`. `curr` di sini digunakan untuk menyimpan jumlah institusi yang mempunyai minimal 1 kelompok mereka yang lolos ke ICPC, sedangkan `ansCurr` digunakan untuk menyimpan jumlah kelompok yang mendapatkan golden ticket. Kami kemudian akan looping sampai n, karena pada soal diberitahu bahwa setelah inputan 3 angka, akan terdapat n baris input. Di dalam looping pertama, kami akan deklarasi string `team` dan `ins`, yaitu string untuk menampung nama tim dan nama institusi pada baris ke i, kami kemudian akan ambil inputannya. Karena pada soal diberitahu bahwa m kelompok tertinggi akan lolos ke ICPC, maka secara otomatis, semua kelompok input awal di mana i masih kurang dari m akan lolos ke ICPC. Maka, jika itu terjadi, kami memasukkan nama institusi kelompok tersebut ke dalam array. Di sini kami tidak mengecek apakah nama institusi tersebut sudah terdapat dalam array karena constraint yang sangat kecil maka adanya duplikat dalam array tidak akan menghasilkan efek yang signifikan terhadap performa kode kami.

Lalu, jika kelompok input pada baris i sudah tidak kurang dari m, atau dengan kata lain sudah tidak lolos ICPC, kita mengecek apakah kelompok tersebut bisa mendapatkan golden ticket. Caranya adalah dengan melakukan loopingan terhadap semua nama institusi yang sudah terdapat kelompok yang lolos dan mengecek apakah nama institusi sekarang sama dengan nama institusi yang terdapat dalam array. Jika sama, maka kelompok tersebut tidak eligible untuk golden ticket. Namun jika misalnya tidak terdapat nama institusi yang sama, atau dengan kata lain kelompok ini adalah kelompok pertama dari institusi tersebut, maka kelompok tersebut eligible untuk golden ticket. Terakhir, kita hanya perlu mengecek apakah masih terdapat kuota golden ticket yang dilakukan dengan mengecek apakah jumlah `ansCurr` atau jumlah kelompok yang mendapatkan golden ticket masih dibawah kuota golden tiket. Jika kedua kondisi terpenuhi, maka tim tersebut akan dimasukkan dalam array tim yang mendapatkan golden ticket dan nama institusi kelompok tersebut akan dimasukan ke dalam array. Setelah loopingan selesai, maka jawaban sudah didapatkan dan kita hanya perlu mengeluarkan output jawabannya. Pertama, kita akan print jumlah kelompok yang mendapatkan golden ticket, kemudian kita akan looping jumlah kelompok tersebut, dan print nama setiap kelompok.

Untuk mengetahui time complexity dari kode ini, kita dapat melakukan analisis line per line kode ini. Namun, karena sebagian besar dari kode ini mempunyai time complexity $O(1)$, maka kami akan membahas bagian penting seperti loopingannya. Dari kodingan ini, bagian penting

pertama adalah loopingan i dari 0 sampai n. Untuk baris kode di atas loopingan ini mempunyai time complexity $O(1)$. Baris-baris ini tidak begitu signifikan, maka dapat diabaikan. Karena loopingan ini melakukan perulangan sampai n, maka tentu akan memerlukan waktu n. Di dalam loopingan kemudian terdapat sebuah inner loop j dari 0 sampai `curr`. Di sini, `curr` pada worst case akan sama dengan n. Oleh karena big O notation merujuk pada worst case, maka kita akan anggap loopingan ini memerlukan waktu n, sehingga didapatkan loopingan luar dan dalam digabung memerlukan waktu $n * n = n^2$. Di bagian paling bawah, terdapat loopingan lagi yang pada worst casenya juga akan sama dengan n, maka total time complexitynya akan menjadi $O(n^2 + n)$. Namun ketika menghitung time complexity, kita hanya akan mengambil yang paling besar dan signifikan saja, maka didapatkan time complexity dari kodingan ini adalah $\boldsymbol{O(n^2)}$.

Kompleksitas Waktu: $\boldsymbol{O(n^2)}$

PROBLEM B – DIET PLAN

Analisis Masalah pada Soal:

Pada soal diketahui bahwa kita mempunyai rencana diet untuk N hari, dan pada hari ke i, kita perlu minum P_i mL susu atau mengonsumsi 1 biskuit. Diketahui juga awalnya kita mempunyai M mL susu dan K biskuit, ditanyakan jumlah hari diet maksimum yang bisa didapatkan jika kita mengonsumsi susu dan biskuit kita secara optimal.

Jika kita lihat secara sekilas, mungkin saja kita mendapatkan sebuah logika penyelesaian. Dari soal diketahui bahwa sebanyak apapun susu yang perlu dikonsumsi, dapat dengan mudah digantikan dengan 1 biskuit, maka mungkin saja kita berpikir bahwa kita dapat mengoptimalkan jawaban kita dengan cara hanya mengonsumsi biskuit untuk hari di mana kita memerlukan paling banyak susu. Pemikiran tersebut tidak salah, namun belum tentu optimal karena kita harus melewati hari dan jumlah susu yang diperlukan secara berurutan. Jadi bisa saja terdapat situasi seperti ini. Kita mempunyai 200 mL susu dan 3 biskuit, kemudian jumlah susu yang diperlukan secara berurutan yaitu 50 80 30 90 100 110 120. Jika menggunakan logika sebelumnya, maka untuk mendapatkan hasil optimal, kita perlu mengonsumsi biskuit untuk 3 hari terakhir di mana jumlah susu yang diperlukan maksimal. Namun jika kita perhatikan dengan lebih dekat, jumlah susu yang diperlukan untuk melewati 4 hari pertama adalah 250 mL, maka dengan logika sebelumnya kita akan kehabisan susu di hari ketiga, dan logika ini bukan merupakan jawaban optimal.

Bagaimana dengan brute force? Salah satu cara yang dapat kita lakukan adalah mengecek setiap kemungkinan. Pada hari pertama, kita dapat memilih untuk meminum susu atau mengonsumsi biskuit. Kemudian hari kedua kita mempunyai pilihan yang sama, diulang terus sampai hari terakhir. Setiap hari pilihan kita akan berlipat ganda. Pada hari pertama akan terdapat 2 pilihan, kemudian pada hari kedua, 2 pilihan tersebut akan mempunyai 2 hasil yang berbeda, dan setiap hasil akan mempunyai 2 pilihan lagi. Dengan kata lain, 2 hasil akan mempunyai 4 pilihan, dan dari 4 pilihan akan menghasilkan 4 hasil. Pada hari ketiga akan terdapat 8 pilihan, dan seterusnya. Terlihat pola jumlah pilihan yang akan dipilih adalah 2^n. Apabila dilihat dari constraint soal ini, yaitu 100 untuk N, M, dan K. Dengan 2^{100}, pada worst case kita akan perlu mengecek kurang lebih 10^{30} pilihan berbeda, dengan asumsi waktu 1 detik di mana umumnya kita dapat mengecek 10 juta kali atau 10^7, jadi dengan solusi brute force ini, pada worst case kita akan memerlukan waktu 10^{23} detik, atau setara dengan 3×10^{15} tahun. Tentu saja solusi ini kurang efisien, maka solusi ini tidak dapat digunakan begitu saja, namun kita dapat meng-improve solusi ini dengan mengimplementasikan **dynamic programming**. Di sini kami menggunakan memoization dan top-down approach. Teknik dynamic programming membantu mengurangi waktunya secara signifikan, untuk penjelasan lebih lanjut mengenai time complexity-nya akan dijelaskan setelah kodingan ditunjukkan.

Kode Program:

Kodingan yang benar untuk menyelesaikan permasalahan ini adalah sebagai berikut. Kodingan menggunakan bahasa pemrograman C++.

```
#include<iostream>
#include<bits/stdc++.h>
#include<vector>
#include<string>
#include<math.h>

#define ll long long

using namespace std;

int dp[101][101][101];

int solve(int arr[], int curr, int m, int k, int ans, int n){
     if ((m <= 0 && k <= 0) || curr >= n) return ans;
     int first = ans;
```

```
      int second = ans;
      if (dp[curr][m][k] != 0) return dp[curr][m][k];
      if (m >= arr[curr]) first = solve(arr, curr + 1, m - arr[curr], k, ans + 1, n);
      if (k) second = solve(arr, curr + 1, m, k - 1, ans + 1, n);
      return dp[curr][m][k] = max(first, second);
}

int main(){
      int n, m, k;
      cin >> n >> m >> k;
      int nums[n];
      for (int i = 0; i < n; i++) cin >> nums[i];
      cout << solve(nums, 0, m, k, 0, n);
      return 0;
}
```

Analisis Solusi dari Kode Program:

Pertama, kami mendeklarasikan sebuah 3D array untuk integer dengan ukuran $101 \times 101 \times 101$. Mengapa 101? Karena pada soal diberitahu bahwa constraint angka yang diberikan adalah 100. Di sini ditambah 1 untuk menyimpan ketika indeksnya 0. Array tiga dimensi ini akan menyimpan jawaban untuk bagian masalah agar dapat digunakan untuk menghitung jawaban akhir. Pada soal terdapat 3 hal yang memengaruhi hasil, yaitu hari ke berapa yang memengaruhi jumlah susu yang diperlukan, jumlah susu yang ada sekarang, dan jumlah biskuit yang ada sekarang, sehingga dibuatlah 3D array untuk menampung semua kemungkinan yang ada. 100 hari berbeda, dengan setiap hari akan terdapat 101 jumlah mL susu kemungkinan yang ada, dan 101 jumlah biskuit yang memungkinkan. Dengan size $101 \times 101 \times 101$, kita mendapatkan ukuran 1.030.301. Oleh karena integer memakan memori 4 byte, maka diperoleh jumlah memori yang diperlukan adalah $1.030.301 * 4 = 4.121.204$ bytes = 4.121 KB, kurang lebih 4 MB. Pada soal competitive programming, umumnya diberikan 128 atau 256 MB space, maka solusi ini masih tergolong aman dari segi space complexity.

Setelah itu, kami membuat sebuah function yang mempunyai return value integer dengan nama solve untuk menyelesaikan masalah ini secara rekursif. Function ini akan mempunyai 5 parameter, yaitu array of integer `arr` untuk menampung jumlah susu yang diperlukan pada setiap

harinya, integer `curr` untuk menampung hari ke berapa saat ini, integer m untuk menampung jumlah susu saat ini, integer k untuk menampung jumlah biskuit saat ini, integer `ans` untuk menampung jawaban hari ini, dan integer n untuk menampung jumlah hari rencana dietnya. Di dalam function ini terdapat base case, yaitu jika $m \leq 0$ dan $k \leq 0$, yang berarti jika jumlah susu dan biskuit kurang dari sama dengan 0, atau `curr` $\geq n$, yang berarti jumlah hari sudah mencapai rencana diet, maka kita akan return `ans` atau hari jawaban akhirnya untuk subproblem tersebut.

Kemudian kita akan set `first` dan `second` ke `ans`, yaitu jumlah hari maksimum yang didapatkan jika kita memilih untuk menggunakan susu atau biskuit. Kita akan mengecek apakah perhitungan spesifik ini sudah pernah dihitung, dengan kata lain apakah pada hari `curr`, dengan jumlah susu m, dan jumlah biskuit k, sudah terdapat hasil maksimum jumlah hari dari perhitungan sebelumnya. Jika sudah ada, maka kita tidak perlu menghitung ulang dan kita dapat langsung menggunakan jawaban tersebut. Kemudian, jika misalnya belum pernah dihitung, maka kita akan mengecek apakah kita masih mempunyai susu. Apabila susu masih cukup, maka kita akan mengecek jumlah hari maksimum yang bisa didapatkan jika kita memilih susu sekarang. Kemudian kita cek apakah kita masih mempunyai biskuit. Bila ada, kita akan cek jumlah hari maksimum yang bisa didapatkan jika kita memilih biskuit sekarang. Kedua jawaban ini akan ditampung di `first` dan `second`, kemudian hari maksimum yang dapat dilalui merupakan nilai maksimum dari `first` dan `second`. Hari maksimum ini akan disimpan kedalam array `dp` dan kemudian digunakan untuk perhitungan berikutnya.

Pada bagian `main`, sesuai dengan inputan soal, kami akan mengambil dan menampung input dari soal, kemudian kami akan meng-output hasil dari perhitungan function `solve`. Mengapa kodingan ini dapat menyelesaikan masalah ini? Karena dengan penggunaan dynamic programming, time complexity-nya berubah secara signifikan menjadi lebih baik. Untuk analisis lebih dalam, kita dapat mengecek mulai dari `main`. Seperti sebelumnya, line kode yang kompleksitasnya konstan akan diabaikan karena tidak memengaruhi time complexity secara signifikan.

Pada loopingan pertama, untuk scan akan dilakukan loopingan sebanyak n, maka akan memerlukan waktu n. Kemudian kita menganalisis function dari `solve` ini sendiri. Diketahui bahwa dari function ini, jika misalnya nilai pada `dp[curr][m][k]` sudah ada, maka kita tidak perlu menghitung dan melakukan rekursi, melainkan dapat langsung mengambil nilainya. Pada worst case scenario, kita akan mengasumsi bahwa untuk semua perhitungan yang dilakukan, tidak terdapat situasi dimana nilai `curr`, nilai m, dan nilai k sama dengan nilai pada perhitungan sebelumnya, maka kita perlu melakukan perhitungan ulang. Namun karena ukuran arraynya sendiri $n * n * n = n^3$, pada worst case scenario, kita akan perlu melakukan rekursi dan perhitungan maksimal sebanyak n^3 kali. Tidak mungkin bisa lebih tinggi dari itu, karena ketika mencapai n^3, maka nilai

pada setiap index di array akan terisi, yang berarti angka apapun yang diberikan, kita sudah mempunyai hasil perhitungan sebelumnya yang dapat digunakan. Maka didapatkan time complexity dari kodingan ini adalah $O(n^3 + n)$, atau dapat disimpulkan menjadi $\boldsymbol{O(n^3)}$. Dengan ukuran n maksimum 100, maka waktu yang diperlukan adalah maksimum 100^3 = 1 juta, yang masih sedikit di bawah limit pada umumnya, yaitu 10 juta. Maka, kodingan kami dapat menyelesaikan masalah dengan baik dan efisien secara kompleksitas waktu.

Kompleksitas Waktu: $\boldsymbol{O(n^3)}$

PROBLEM H – HORSE CARTS

Analisis Masalah pada Soal:

Pada soal diketahui bahwa terdapat sebuah gua, di mana dalam gua tersebut terdapat N harta karun. Harta karun i mempunyai berat W_i dan nilai V_i. Diketahui juga kita mempunyai M kereta kuda untuk membawa harta karun yang ada. Setiap kereta dapat membawa maksimal 1 harta karun dan kereta j dapat membawa maksimum berat harta karun S_j. Kita diminta untuk menentukan nilai maksimum harta karun yang bisa dibawa.

Dari soal kita tahu bahwa setiap kereta hanya dapat membawa 1 harta karun, maka logikanya setiap kereta harus membawa harta karun dengan nilai tertinggi di mana beratnya masih dapat dibawa oleh kereta. Karena setiap kereta dengan berat maksimum yang lebih tinggi pasti dapat membawa harta karun yang dapat dibawa oleh kereta dengan berat maksimum yang lebih tinggi, untuk mendapatkan hasil yang optimal kita perlu membawa harta karun dari kereta dengan berat maksimum yang paling rendah sampai paling tinggi.

Jadi, algoritma yang dapat digunakan adalah pertama-tama kita sorting harta karunnya berdasarkan berat setiap harta karun dari paling rendah ke paling tinggi. Kemudian sorting kereta kudanya dari berat maksimum paling rendah sampai paling tinggi. Setelah itu, kita akan mencari nilai maksimum yang dapat dibawa oleh setiap kereta mulai dari kereta dengan berat maksimum terendah sampai tertinggi. Solusi ini bisa mendapatkan hasil yang optimal. Namun, jika kita perhatikan constraint yang diberikan soal, jumlah maksimum harta karun adalah 100 ribu, dan jumlah maksimum kereta kuda adalah 100 ribu juga. Jika kita mengambil salah satu kemungkinan input yang ada, yaitu input di mana diberikan 100 ribu harta karun dengan setiap harta karun mempunyai berat 1, lalu 100 ribu kereta kuda di mana setiap kereta kuda mempunyai berat maksimum

≥ 1, maka kita perlu melakukan 100 ribu $\times$ 100 ribu loopingan atau 10^{10} loopingan, yang jauh di atas limit pada umumnya yaitu 10^8. Maka dari itu, solusi ini meskipun benar, akan membutuhkan waktu yang melebihi batas.

Solusi sebelumnya dapat kita improvisasi agar menggunakan waktu yang lebih sedikit. Pada solusi awal kita, kita looping seluruh harta karun di mana beratnya kurang dari sama dengan berat maksimum kereta, dan ambil nilai yang paling tinggi. Namun hal ini kita lakukan dan ulangi untuk setiap kereta, meskipun sebenarnya, nilai pada setiap harta karun yang pernah di-looping sebelumnya tetap sama dan kita tidak perlu melakukan looping ulang. Untuk mendapatkan hasil yang lebih optimal, kita bisa menggunakan **priority queue**, di mana harta karun dengan nilai yang paling tinggi akan berada di paling depan. Dengan cara ini, kita tidak perlu looping setiap harta karun, namun dapat langsung mengambil nilai pertama dalam queue tersebut.

Kode Program:

Kodingan yang benar untuk menyelesaikan permasalahan ini adalah sebagai berikut. Kodingan menggunakan bahasa pemrograman C++.

```
#include<iostream>
#include<bits/stdc++.h>
#include<vector>
#include<string>
#include<math.h>
#include<queue>

#define ll long long

using namespace std;

bool sortColumn(vector<ll>& v1, vector<ll>& v2)
{
    return v1[0] < v2[0];
}

int main(){
     ll n, m;
     cin >> n >> m;
```

```
	priority_queue<ll> queue;

	vector<vector<ll> > stuff(n, vector<ll>(2));
	vector<ll> horse(m);
	for (ll i = 0; i < n; i++) {
		ll weight, val;
		cin >> stuff[i][0] >> stuff[i][1];
	}
	sort(stuff.begin(), stuff.end(), sortColumn);

	for (ll i = 0; i < m; i++) cin >> horse[i];
	sort(horse.begin(), horse.end());
	ll ans = 0;

	ll curPos = 0;

	for (ll i = 0; i < m; i++){
		while(curPos < n && stuff[curPos][0] <= horse[i]){
			queue.push(stuff[curPos++][1]);
		}
		if (!queue.empty()){
			ans += queue.top();
			queue.pop();
		}
	}

	cout << ans << endl;

	return 0;
}
```

Analisis Solusi dari Kode Program:

Dalam kodingan ini, pertama kami membuat function dengan nama `sortColumn`. Function ini digunakan untuk melakukan sorting terhadap vector 2D. Function ini akan membantu untuk sorting berdasarkan kolom pertama dari vector 2D tersebut dari kecil ke besar. Di kode `main`, pertama kita akan membuat priority queue untuk menyimpan semua nilai harta karun diurutkan dari nilai paling tinggi ke nilai paling rendah, kemudian kita akan membuat vector 2D untuk menyimpan berat dan nilai dari setiap harta karun yang diinput, dan juga sebuah vector untuk menyimpan berat maksimum dari setiap kereta kuda. Kemudian, kita akan looping dan mengambil input berat dan nilai dari setiap harta karun. Setelah selesai, kita akan sorting datanya, lalu kita akan ambil input berat maksimum setiap kereta kuda dan sorting dari yang paling rendah ke tinggi. Setelah itu, kita akan looping setiap kereta kuda yang ada. Dalam loopingan tersebut, kita kemudian akan memasukkan semua harta karun yang beratnya kurang dari sama dengan berat maksimum kereta tersebut ke dalam priority queuenya, dan jika sudah selesai, kita akan mengambil nilai pertama dalam priority queue tersebut yang merupakan nilai maksimum. Setelah semuanya selesai, maka kita akan memperoleh nilai maksimum yang bisa didapatkan.

Untuk mengetahui time complexity dari kode ini, kita dapat menganalisisnya seperti sebelumnya. Baris kode yang mempunyai time complexity $O(1)$ akan diabaikan. Bagian pertama yang mempunyai time complexity tinggi adalah looping input harta karun sebanyak n dan mempunyai time complexity $O(n)$. Selanjutnya data akan di-sorting, di mana function sort pada C++ mempunyai time complexity $O(n \log n)$. Sementara itu, loopingan untuk mengambil input berat maksimum setiap kereta kuda mempunyai time complexity $O(m)$, dan sorting-nya memiliki kompleksitas waktu $O(m \log m)$. Bagian terakhir yang mempunyai time complexity tinggi adalah loopingan untuk mengambil nilai tertinggi harta karun. Jika dilihat secara sekilas, bisa saja kita mengasumsi time complexitynya adalah $m * n$ karena terdapat nested loop. Namun, jika kita lihat dengan lebih dekat, kita dapat melihat bahwa `curPos` tidak pernah reset kembali setelah setiap loopingan, maka dari itu loopingan di dalamnya akan dijalankan maksimal n kali tanpa memerhatikan berapa kali loopingan luar dijalankan. Maka time complexity dari loopingan ini sebenarnya adalah $O(m + n)$. Diperoleh total time complexity dari kode ini adalah $O(n + n \log n + m + m \log m + m + n)$. Jika disederhanakan dan kita hilangkan fungsi waktu yang lebih rendah, didapatkan time complexity akhir bernilai $\boldsymbol{O(n \log n + m \log m)}$. Dengan nilai n maksimum 100 ribu dan nilai m maksimum 100 ribu, diperoleh jumlah operasi maksimum adalah 1 juta, yang masih di bawah limit umumnya, yaitu 10 juta.

Kompleksitas Waktu: $\boldsymbol{O(n \log n + m \log m)}$

Selanjutnya, berikut ini adalah solusi kode program dan analisis algoritma dalam kode program yang kami pilih untuk di-*upsolve* pasca-pelaksanaan INC 2023. Soal yang akan kami bahas adalah **Problem E**.

PROBLEM E – REVERSE SEVERER

Analisis Masalah pada Soal:

Pada soal diketahui bahwa akan diberikan sebuah string dengan panjang N huruf. Kemudian, akan diberikan Q baris input, untuk setiap baris input akan diberikan sebuah string dengan panjang N, dan kita perlu menentukan apakah string tersebut dapat dibentuk dari string awal menggunakan suatu algoritma dengan 3 step, yaitu:

1. Split stringnya menjadi satu atau lebih substring
2. Putar balik urutan substring-nya
3. Gabung kembali substring-nya menjadi satu string

Dalam menjawab permasalahan ini, kita tidak bisa menggunakan metode brute force untuk menyelesaikannya, karena jika kita menggunakan metode brute force dengan mencoba semua kemungkinan string yang dapat dibentuk, time complexity dari kodingan ini akan menjadi $O(n^3)$. Dengan panjang string maksimum 10.000, kompleksitas n^3 akan mencapai 10^{12} komputasi yang jauh di atas limit kita.

Untuk menyelesaikan soal ini, kita memerlukan solusi yang lebih efisien lagi. Jika kita perhatikan soalnya, kita bisa menemukan algoritma solusi yang lebih efisien dengan cara mengecek salah satu string dari awal, dan salah satu string dari ujung, kemudian mengambil huruf dari kedua string satu per satu. Jika pada suatu saat, isi dari kedua string baru tersebut sama, kita update titik awal string baru tersebut dan melanjutkan algoritma. Jika ketika algoritma selesai, titik awal string baru sama dengan panjang huruf, maka jawabannya adalah *iya*, namun jika tidak, maka jawabannya adalah *tidak*.

Kode Program:

Kodingan yang benar untuk menyelesaikan permasalahan ini adalah sebagai berikut. Kodingan menggunakan bahasa pemrograman C++.

```
#include<iostream>
#include<bits/stdc++.h>
#include<vector>
```

```
#include<string>
#include<math.h>

#define ll long long

using namespace std;

int main(){
	int n;
	string text;
	cin >> n;
	cin >> text;
	int q;
	cin >> q;
	while(q--){
		string find;
		cin >> find;
		int start = 0;
		bool can = 1;
		for (int i = 0; i < n; i++){
			if (find.substr(start, i-start+1).compare(text.substr(n-i-1, i-start+1)) == 0){
				start = i + 1;
			}
		}
		if (start != n) can = 0;
		can ? cout << "YES" << endl : cout << "NO" << endl;
	}

	return 0;
}
```

Analisis Solusi dari Kode Program:

Pertama, kami akan mengambil dan menampung input dari soal, yaitu panjang string, isi string, dan jumlah query. Setelah itu, kami akan melakukan looping sebanyak jumlah query. Dalam setiap loopingan, kita pertama-tama akan mengambil input stringnya, kemudian di dalamnya kita akan melakukan looping lagi. Loopingan ini akan dilakukan sebanyak panjang string. Di dalam loopingan inilah, kita mengecek apakah string ini dapat dibentuk dari string awal. Di sini kami mempunyai integer `start`, yang akan digunakan untuk menentukan titik awal pengecekan kami. Jika isi string sama, maka titik awal ini akan di-update. Di bagian akhir, jika titik awal tidak sama dengan panjang string, maka string tersebut tidak bisa dibentuk dari string awal. Untuk kodingan ini, bagian signifikan yang memengaruhi time complexity hanya berada di bagian looping saja. Bagian loopingan luar mempunyai time complexity $O(q)$, dan loopingan dalam mempunyai time complexity $O(n)$. Maka, time complexity akhir dari kodingan ini adalah $\boldsymbol{O(q * n)}$, dengan nilai q maksimal 100 dan nilai n maksimal 10.000, sehingga worst case scenarionya akan dilakukan 1 juta kali loopingan, yang masih di bawah limit yang ditentukan.
Kompleksitas Waktu: $\boldsymbol{O(q * n)}$

Keterangan:

Seluruh file .cpp kode program (Problem A, B, E, H) yang kami buat dalam pengerjaan INC 2023 dan AoL Algorithm Design and Analysis ini juga akan kami lampirkan bersamaan dengan file Word ini dalam bentuk file terkompresi .zip.

--- bebekijo ---