ЕСЕНЕН ТУРНИР ПО ИНФОРМАТИКА
Шумен, 10–12 ноември 2006 г.
Тема за група А (12 клас)

Задача А1. ИГРА

Даден е неориентиран граф с n върха и m ребра. Върховете са номерирани с числата от 1 до n. Дадени са и n пула, номерирани също с числата от 1 до n, като във всеки връх на графа е поставен по един пул. Разрешено е разместване на пуловете по един единствен начин: пул с номер 1 може да се размени с пул, който се намира в съседен връх на графа. Напишете програма **`move`**, която намира минималния брой ходове, за които пуловете могат да се разместят така, че всеки пул да се намира във връх със същия номер.

Вход: На първия ред на стандартния вход са дадени числата n и m. На следващите m реда са дадени по две числа, представляващи краищата на поредното ребро. На последния ред са дадени n числа, задаващи началното разположение на пуловете: i-тото число представлява номера на пула, намиращ се във връх i.

Изход: На стандартния изход да се изведе едно число – минималния брой ходове, с които може да се стигне до крайната позиция. Ако не е възможно да се достигне желаната позиция, да се изведе `-1`.

Ограничения: $n < 10$.

ПРИМЕР 1	**ПРИМЕР 2**
Вход	**Вход**

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

Изход	**Изход**
`2`	`-1`

Решение:

Да означим графа от условието на задачата с G. На всяко разположение на пуловете във върховете на G, съпоставяме пермутация $p = p_1, p_2, p_3, \ldots, p_n$, където p_i е номерът на пула, който се намира във връх i. Тъй като $n<10$, за представянето на различните пермутации ще използваме низове, например при n=5, низът `"23514"` означава, че във връх 1 се намира пул 2, във връх 2 – пул 3, във връх 3 – пул 5, във връх 4 – пул 1 и във връх 5 – пул 4.

Разглеждаме втори неориентиран граф P. Върховете на P са $n!$, като на всяка пермутация съответства връх от графа. Два върха са съседни, ако от едната пермутация може да се отиде в другата. Графът P е неориентиран, защото ходовете в играта са обратими. Например, ако в G върхове 2 и 4 са съседни, то от пермутацията `"23514"` може да се премине в `"21534"` и обратно.

Вместо да представяме графа P явно в програмата с някакви структури от данни, всеки път, когато трябва да намерим съседите на дадена пермутация-позиция извършваме следните пресмятания:

1) намираме върха u от графа G, в който се намира пул 1;

2) за всеки връх v, който е съсед на u (в графа G), получаваме съседна позиция в графа P, разменяйки пул 1 с пула от връх v.

При извършване на действията от точки 1) и 2), трябва да се съобразяваме с факта, че елементите на низовете са индексирани от 0.

В графа P трябва да намерим най-късия път (по брой ребра) от началната позиция до крайната позиция z="123…n". Ще използваме алгоритъма за търсене в ширина. За целта ни е необходима опашка, в която да бъдат включвани върховете (в нашия случай – позициите), които са достигнати за първи път.

Дефинираме и създаваме празна опашка по следния начин: `queue<string> Q;`

За всяка достигната позиция y, трябва да поддържаме броя ходове, с които може да бъде достигната. За целта ще използваме изображение (`map`), което на низове съпоставя числа, дефинирано така: `map<string,int> dist;`

Инициализираме стойностите на `dist` с `-1`, като за генерирането на всички пермутации използваме функцията от стандартната библиотека `next_permutation`.
Ако от текущата позиция x, може да се премине към съседна позиция y, за която `dist[y]` има стойност `-1`, то трябва да включим y в опашката: `Q.push(y)` и да установим `dist[y]=dist[x]+1`.

Броят на ребрата на графа P е равен на броя на ребрата на графа G. Следователно тялото на вътрешния цикъл в процедурата `bfs` се изпълнява общо m пъти. Едно отделно изпълнение зависи от реализацията на структурата `map`, която според спецификациите на стандартната библиотека осигурява ефективност $\log n!$. Така общо оценката за всички изпълнения на вътрешния цикъл е $O(m \log n!)$.

Всеки връх се посещава точно веднъж, което дава $O(n!)$ за операциите, свързани с опашката. Следователно за сложността на алгоритъма получаваме $O(n! + m \log n!)$ и като вземем пред вид, че $m \leq n(n-1)/2 \leq 36$, окончателно получаваме $O(n!)$ за сложността на алгоритъма по време. Очевидно и сложността по памет е $O(n!)$.

Стоян Капралов

Задача А2. СЛЪНЧОГЛЕДИ

Иванчо много обичал да ходи на гости на баба си Кунка. Тя имала къща с много дълъг балкон, на перваза на който били наредени саксии със слънчогледи. Противно на общоприетото схващане, те всъщност не гледали към слънцето. След като внимателно ги разгледал, Иванчо ограничил слънчогледите до четири типа – „северни“ (такива, които гледат на север), „южни“, „западни“ и „източни“. Установил още, че броят на северните слънчогледи е равен на броя на южните, както и, че броят на западните съвпада с броя на източните.

Баба Кунка размествала саксиите със слънчогледи всеки ден. При това не го правела безмозъчно, а с някаква странна логика и Иванчо скоро открил каква е тя: ако разгледаме всички саксии от някоя произволна позиция наляво, то броят на северните слънчогледи винаги е поне колкото на южните, както и броят на западните е поне колкото на източните. Изказано по-формално, горните правила звучат така:

n – брой саксии;
$SUM_N(i)$ – брой северни слънчогледи до саксия номер i ($1 <= i <= n$);
$SUM_S(i)$ – брой южни слънчогледи до саксия номер i ($1 <= i <= n$);
$SUM_W(i)$ – брой западни слънчогледи до саксия номер i ($1 <= i <= n$);
$SUM_E(i)$ – брой източни слънчогледи до саксия номер i ($1 <= i <= n$);

Изпълнено е, че:

$SUM_N(n) = SUM_S(n)$;
$SUM_W(n) = SUM_E(n)$;
$SUM_N(i) >= SUM_S(i)$ за всяко i, $i = 1, 2, ..., n$;
$SUM_W(i) >= SUM_E(i)$ за всяко i, $i = 1, 2, ..., n$.

Един ден Иванчо, гледайки поредната подредба на слънчогледите, се зачудил колко други подредби има, запазващи бабината логика. Той решил да счита, че саксиите с еднакво гледащи слънчогледи са неразличими, т.е. ако размени местата на две саксии с еднакво гледащи слънчогледи, няма да получи нова подредба. Помогнете на Иванчо – напишете програма **`flower`**, която прочита една конфигурация от слънчогледи и определя колко още други конфигурации съществуват.

Входните данни се състоят от два реда. На първия ред се намира четното естествено число n – броя саксии на балкона на баба Кунка. Този брой не надвишава 100. На следващия ред се намират n символа, разделени със по един интервал – това са типовете слънчогледи, разгледани от ляво надясно. С „**N**“ ще означаваме северен слънчоглед, с „**S**“ - южен, с „**W**“ - западен и с „**E**“ - източен. Изходът трябва да се състои от единствен ред с едно число – търсеният брой.

ПРИМЕР 1:	ПРИМЕР 2:
Вход:	**Вход:**
`6`	`8`
`N W E W S E`	`N N W E S S N S`
Изход:	**Изход:**
`29`	`139`

Решение:

Задачата може да се разглежда като просто обобщение на известната „задача за скобите“ (ако са ни дадени **n** двойки скоби, колко е броя на „правилните аритметични изрази“, които може да образуваме с тях). Тази задача има връзка с числата на Каталан, а именно, K_n изразява броя изрази с **n** двойки скоби.

Иначе казано, в дадената задача се пита по колко начина можем да образуваме израз с два различни типа скоби (например, кръгли и квадратни) – ако имаме **a** двойки кръгли и **b** двойки квадратни скоби, така че всеки тип скоби поотделно образуват правилен израз. Тъй като едните скоби не влияят на „правилността“ на другите, то имаме просто умножението на Каталановите им числа: $K_a * K_b$. Трябва, обаче, да вземем предвид как скобите са „размесени“. Тъй като целия израз има 2*$\mathbf{a}$ + 2*$\mathbf{b}$ символа, може просто да приемем, че започваме от 2($\mathbf{a}$+$\mathbf{b}$) „празни“ клетки, в които трябва да попълним 2*$\mathbf{a}$ полета със скоби от първия тип, а останалите полета запълним със другия тип. Това, очевидно, става по C(2($\mathbf{a}$+$\mathbf{b}$), 2$\mathbf{a}$) начина.

Т.е. задачата може да се реши чрез следната формула:

$$A = K_a * K_b * C(2(\mathbf{a}+\mathbf{b}), 2\mathbf{a}) = C(2\mathbf{a}, \mathbf{a}) * C(2\mathbf{b}, \mathbf{b}) * C(2(\mathbf{a}+\mathbf{b}), 2\mathbf{a}) / ((\mathbf{a}+1)*(\mathbf{b}+1))$$

Където **a** е броят „северни“ слънчогледи (спрямо условието), а **b** е броят „западни“.

Решение 1 („Формулата“)

Това е първият начин да се реши задачата – достатъчно е да се намери формулата и да се смята по нея.

От формулата е достатъчно очевидно, че бройката расте много бързо, и дори при малки стойности на **a** и **b**, броят надхвърля обхвата на 32-битов тип. В тестовете има само 3 примера, които се „хващат“ с 32- (че дори и 64-) битова аритметика. Най-голямият отговор, който се получава при **a** = 25 и **b** = 25, е с *55* цифри. Явна е необходимостта от реализиране на „дълги числа“ под някаква форма. Тук сложността откъм писане на задачата много зависи от пътя за реализация, които ще избере състезателя. Формулата горе, макар и проста, изисква тежките операции „умножение“ и „деление“, които са доста обемисти за писане. Лесно се вижда, обаче, че човек може да мине и без тях, защото множителите и делителите навсякъде са числа до 100. Казано на „програмистски жаргон“, необходими са само операции на умножение/деление на „дълги с къси“ числа.

Един по-ефикасен и удобен метод е през цялото време да пазим числата разложени в каноничен вид (т.е. всяко число X представяме във вида $X = a^x b^y c^z ...$). Така, например, за да умножим едно число X, което е вече в каноничен вид, с друго, Y, което не е, просто трябва разложим Y и да добавим простите множители и степените им в разлагането на X.

Пример:
$X = 90 = 2.3^2.5$, $Y = 14 = 2.7$
$X*Y = 2^2.3^2.5.7$

Така пресмятането на формулата горе се свежда до разлагане на множители и добавяне/изваждане в 100-елементов масив. Разбира се, накрая числото трябва да се превърне от каноничен вид в „нормален“. Тук отново се стига до „умножение на дълго с късо“ число, което не вярвам да затрудни състезателите.

Решение 2 (Динамично оптимиране)

Вторият начин за решаване на задачата се основава на техниката „динамично оптимиране“. За човек, който не е отлично запознат с числата на Каталан, биномните коефициенти и т.н. е по-логично да тръгне по този начин. Дефинира се четириаргументната функция $f(N, S, W, E)$, с която означаваме броя начини да „довършим правилно“ израза, ако вече сме разположили N северни, S южни, W западни и E източни слънчогледа. Функцията много лесно се дефинира:

$f(N, S, W, E) :=$

0, ако $N > \mathbf{a}$ или $S > \mathbf{a}$ или $W > \mathbf{b}$ или $E > \mathbf{b}$ или $S > N$ или $E > W$.
1, ако $N = S = \mathbf{a}$ и $W = E = \mathbf{b}$.
$f(N+1, S, W, E) + f(N, S+1, W, E) + f(N, S, W+1, E) + f(N, S, W, E+1)$, иначе.

Последния ред може да обясним със следното разсъждение: Ако вече сме разположили известен брой слънчогледи, то на текущата позиция можем да сложим някой от четирите типа слънчогледи и отговора е сумата от броя начини да довършим правилно израза при вече зафиксиран слънчоглед на текущата позиция.

Отговорът на задачата е стойността на $f(0, 0, 0, 0)$. Директното прилагане на рекурсивната дефиниция по-горе, обаче, е непрактично, поради експоненциалната сложност на изчислението: всеки екземпляр от функцията (в общия случай) поражда 4 извиквания на същата функция. За целта ще използваме таблица dyn[51][51][51][51], в която ще записваме вече пресметнатите стойности на функцията (*memoization*).

Както вече споменахме, задачата изисква „дълги числа“ в някаква форма. Тук, обаче, за разлика от комбинаторното решение, нещата са по-приятни, тъй като се изисква

само операцията „събиране“. Съществен проблем обаче започва да става обема на използваната памет: таблицата dyn е с 6,764,201 елемента и ако за всеки елемент отделим (например) 150 байта за едно дълго число, то заеманата памет ще стане към 1 GB. Има няколко начина да се разреши този проблем, между които:

- Да се определи каква е максималната големина на числата, които ще се съхраняват в таблицата. Това може да стане и експериментално. При една достатъчно пестелива реализация на дълги числа, необходимата памет за таблицата става около 200 MB.
- Лесно се забелязва, че не всички клетки в таблицата ще се използват. Например, при **a** = 25, **b** = 25, таблица dyn[26][26][26][26] би ни свършила работа, а при **a** = 48, **b** = 2, дори по-малка таблица стига: dyn[49][49][3][3]. Това наблюдение може да се приложи по два начина: единият е да се задели толкова голяма таблица, колкото е нужно; вторият е, в таблицата да се съхраняват само указатели към числа, вместо самите числа, и паметта за всяка клетка да се заделя само когато е нужно.

Решение 3 (Прекалкулиране)

С използването на кой да е от двата начина горе, но без да се решават всички тънкости, е лесно да се изчислят отговорите за всеки възможен входен пример. Тъй като отговорите са симетрични за **a** и **b**, достатъчно е да се прекалкулират около 600 отговора.

Веселин Георгиев

Задача А3. ХРАНА

Гошо Ламерът е още ученик. Но вместо да ходи редовно на училище, той ходи редовно в близкия игрален клуб, където играе докато му свършат джобните. Един ден, когато това се случи, вече беше следобед и Гошо беше огладнял. Той отиде в близката верига за бързо хранене и застана пред вкусното меню с празни джобове. В заведението имаше много различни храни и напитки както и много промоции, с които ако човек си вземе например пица и кола спестява два лева. Може Гошо да е с празни джобове, но не е глупав. Той забеляза, че системата, която смята сметката първо прибавя цената на всички закупени храни и после изважда промоциите, за които има необходимите продукти. Гошо установи, че системата не проверява дали някой продукт вече е участвал в промоция и само с една кола например може да се хванат няколко промоции наведнъж (пица с кола, бургер с кола и т.н.)

Имайки всички продукти и промоциите за комбинации от тях, Гошо иска да му помогнете (като напишете програма **`food`**) да избере тези храни и напитки, за които не само няма да има нужда да плаща, но и да вземе максимално "ресто", за да се върне в игралния клуб.

Не забравяйте, че Гошо няма никакви пари, така че ако не може да поръча нещо без да плати, той няма да си купи нищо и ще се прибере вкъщи.

Входните данни се четат от стандартния вход. На първия ред има две числа N ($3 \leq N \leq 75$) и M ($1 \leq M \leq 75$), съответстващи на броя различни храни и броя промоции във веригата за бързо хранене. Храните и промоциите са номерирани от 1 до техния съответен брой. Следват N реда, съдържащи по едно цяло число C_i ($1 \leq C_i \leq 1000000$) – цената на съответната храна. Следват M реда започващи с числата P_i ($1 \leq P_i \leq 1000000$) и F_i ($1 \leq F_i \leq 10$) – отстъпката за съответната промоция както и броят храни, които участват в нея. Всеки от M-те реда завършва с F_i различни числа – от 1 до N – храните в съответната промоцията.

Вашата програма трябва да изведе на стандартния изход едно число – максимално ресто, което Гошо може да получи при подходящо подбрана поръчка. Ако няма храни, такива, че закупувайки ги не се налага да се плаща – изведете 0.

ПРИМЕР

ВХОД	**ИЗХОД**
5 4 20 20 20 100 10 10 1 1 60 3 1 2 3 20 2 3 4 10 1 5	10

Пояснение: Поръчвайки храни 1, 2, 3 и 5, Гошо ще хване промоции 1, 2 и 4, и ще е на печалба 10.

Решение:

Решението на задачата се свежда до построяване на претеглен насочен граф и намиране на максимален поток в него. Върховете на графа ще бъдат храните и промоциите, както и два специални върха – източник и цел. Ребрата в графа са както следва:

- Свързваме източника с всеки продукт, като теглото на насоченото ребро е цената на продукта.
- Свързваме всяка промоция с целта, като теглото на насоченото реброто е отстъпката на промоцията.
- Свързваме всеки продукт с промоциите, в които участва като за тегло на насочените ребра слагаме безкрайност.

В така построеният граф намираме максимален поток от върха източник до върха цел. Нека означим с $c[x][y]$ капацитета на ребро от връх x до връх y, а с $f[x][y]$ потокът преминал по реброто от връх x до връх y.

Печалбата, която Гошо може да постигне е равна на сумата на всички промоции минус големината на потока. Което всъщност е еквивалентно на сумата ($c[x][y] - f[x][y]$), за всички ребра излизащи от промоциите към върха цел.

За да достигнем до този извод може да разсъждаваме така – първоначално считаме, че всички промоции ни носят пари. Потокът, които минава през графа са парите които плащаме за храните. Той е с големина цените на храните. Също така, след като мине през някоя храна, потокът минава само през тези промоции, в които участва съответната храна – тоест той отнема от печалбата само на тези промоции. Следователно, щом за реброто на някоя промоция имаме $f[x][y] < c[x][y]$, то всички продукти за нея са изплатени (от нея и от други промоции) и ние ще добием печалба от нея.

Иван Анев