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“)