В. С. Остапець,
вчитель-методист Щасливського НВК,
Бориспільський район

ВІД ЗАДАЧІ ДО ЗАДАЧІ

(КОМБІНАТОРНА ЗАДАЧА ІЗ ВИКОРИСТАННЯМ БАГАТО-РОЗРЯДНОЇ АРИФМЕТИКИ)

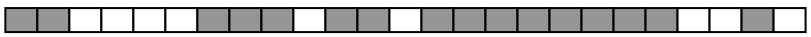
2005 рік

В. С. Остапець,
вчитель-методист Щасливського НВК, Бориспільський район

ВІД ЗАДАЧІ ДО ЗАДАЧІ

Перед тренерами учасників учнівських олімпіад з інформатики, а точніше, з програмування, постійно виникають питання, *якими повинні бути задачі* та *як навчати їх розв'язуванню*. Питання зрозумілі і незрозумілі водночас, якщо враховувати, що програмування – це нетрадиційний для школи предмет, який вимагає вільного володіння математичним апаратом, а задачі важко розділяти за віковими категоріями. Перш за все треба розібратись, які розділи математики та в якому порядку і поєднанні доцільно розглядати у процесі підготовки юного програміста, на яких типових задачах слід починати шліфовку знань, умінь і практичних навичок? На наш погляд, відповідь на це питання дуже вдало дано у [1], де після розгляду арифметики багаторозрядних чисел поставлено комбінаторні алгоритми, а вже потім розглянуто теми перебору і методів його скорочення, алгоритми на графах, обчислювальну геометрію і т.д. Тому і ми тут, зовсім не випадково, зосередимось на розгляді комбінаторних алгоритмів, взявши у якості робочого матеріалу дуже цікаву задачу з [1]. Додатково до цього спонукала ще й необхідність своєрідної післямови до III етапу Всеукраїнської олімпіади-2005 з інформатики у Київській області, де розглянута нижче задача була представлена.

Задача 15. ([1], стор. 75) На смужці клітинкового паперу висотою в одну і довжиною N клітинок деякі клітинки зафарбовані в чорний і білий ко-



льори. Кодом клітинки є послідовність чисел – кількості ідучих підряд чорних клітинок зліва направо. Наприклад, для наведеної смужки кодом буде послідовність 2, 3, 2, 8, 1. При цьому кількість білих клітинок, які розділяють групи чорних ніде не враховуються (головне, щоб дві сусідні групи розділені принаймні однією білою клітинкою). Одному й тому ж коду можуть відповідати кілька смужок, наприклад наведеному коду відповідає й така смужка:

Задача полягає в тому, щоб знайти кількість смужок довжини N, які відповідають заданому коду.

Вхідні дані. У єдиному рядку файлу файла *input.txt* записано число N – довжина смужки ($1 \le N \le 200$), потім число K ($0 \le K \le (N+1)/2$) – кількість чисел у коді і далі K чисел, що визначають код.

Вихідні дані. У вихідний файл *output.txt* записується кількість смужок довжини N, які відповідають заданому коду.

а) ***Дослідження умови та виділення параметрів***. З умови видно, що задача має комбінаторний характер, причому на підрахунок смужок, а не на їх генерацію, отже слід знайти потрібні формули. На користь цього також свідчить можливість надто великого числа смужок, генерація яких може виходити за межі стандартних розрядів та допустимих проміжків часу.

Зробимо позначення: s - сума чисел коду та числа K - 1, $Ns = N - s$ - кількість незафарбованих клітинок, які слід розмістити серед $K1 = K + 1$ місць. Кількість варіантів такого розподілу, яку позначимо KS і буде шуканою відповіддю. З огляду на це, запитання задачі 3 можна сформулювати інакше: *знайти число KS - кількість способів розміщення Ns пропусків на K*1 *місцях, щоб їх сума дорівнювала Ns*. (*)

Зауваження. Слід звернути увагу на те, що у [1] за s помилково взято суму чисел коду та числа K. Там же, у вказівці до розв'язування, помилково стверджується, що залишається підрахувати *число розміщень N – s по відомій кількості місць – K* + 1. Але жодна із формул - $A_N^k = = N^k$ (розміщень з повтореннями) та $A_N^M = N! / (N - M)!$ (розміщень без повторень) у нашому випадку не підходить. Очевидно, задача відноситься до категорії складних (не випадково вона наведена під №15 із 17, включених у якості додаткових до розділу "Комбінаторні алгоритми"), тому, на нашу думку, короткої вказівки (причому з указаними неточностями та обмеженнями, що потребують застосування багаторозрядної арифметики) зовсім не достатньо. Поставимо перед собою завдання: *знайти ефективний алгоритм пошуку числа KS, яке за рахунок умов* $1 \le N \le 200$ *та* $0 \le K \le (N+1)/2$ *може бути багаторозрядним числом*. Забігаючи наперед, зауважимо, що в процесі розв'язання цього завдання доведеться шукати відповіді й на інші, які виникнуть по ходу і з яких випливатимуть важливі наслідки.

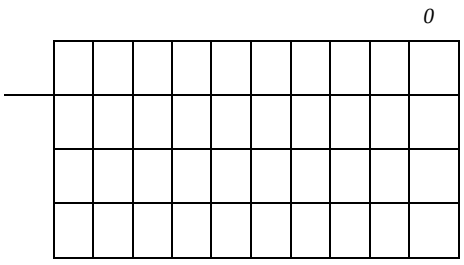
b) ***Пошуки математичної моделі***. Як уже згадувалось вище, формули числа розміщень застосувати до задачі не можна. Але при визначенні кількості варіантів завжди виникає спокуса використати одну з комбінаторних формул. Нескладний аналіз допоможе представити шукане число у вигляді числа комбінацій без повторень, але з огляду на наступні міркування, це буде зроблено значно нижче.

Справа в тому, що математичну модель задачі вибирають здебільшого залежно від кількох обставин: обмежень на вхідні та вихідні дані, віку виконавців або стану їх математичної підготовки. Ми поки що не будемо звертати увагу на формат даних, обмежившись стандартними типами цілих чисел, зокрема типом *longint*[1]. Дві інші обставини проігнорувати ніяк не можна. За браком достатньої математичної підготовки, учні змушені користуватись спрощеною моделлю задачі, що приводить до неповного чи неефективного розв'язку. Виходячи з цього, можна стверджувати, що учні, розв'язуючи подібну задачу, майже обов'язково, підуть не шляхом застосування комбінаторики, яка вивчається, на жаль тільки в 11 класі, причому без достатнього закріплення та застосування.

Отже, прослідкуємо за емпіричним ходом пошуку робочих формул. Зупинимось на прикладі смужки при $N = 8$, $K = 3$, код 1, 2, 1. Як початкову, візьмемо смужку (малюнок 1):

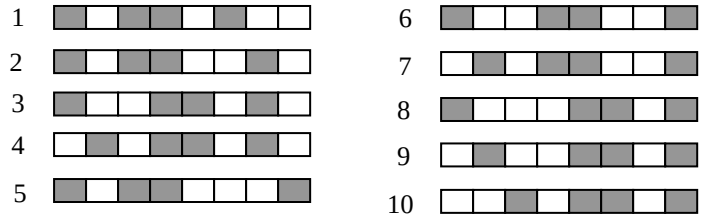
мал. 1

Зробимо підрахунок, попередньо згенерованих варіантів смужок.Як видно з малюнка 2, таких смужок 10. Як уникнути генерування смужок? З (*) випливає, що задача для смужки, зображеної на малюнку 1, зводиться до варіантів, які видно з наступної таблиці:



[1] Маємо на увазі тип *longint* для мови Turbo Pascal.

таблиця 1



мал. 2

Очевидно, що *KS* буде функцією від двох величин – *Ns* та *K*+1, тобто *KS*(*Ns*,*K*+1), у нашому прикладі *KS*(2, 4). Спробуємо визначити формулу для знаходження *KS*(2, 4), а на її основі формулу для визначення *KS*(*i*, *j*). Очевидно, що *KS*(1,*j*) = *j*, *KS*(*i*,1)=1, *KS*(0,*j*)=1. (1)

Отже: *KS*(2, 1) = 1;

$$\left.\begin{aligned} &KS(2,2) = KS(2,1) + KS(1,1) + KS(0,1) = 1+1+1 = 3;\\ &KS(2,3) = KS(2,2) + KS(1,2) + KS(0,2) = 3+2+1 = 6;\\ &KS(2,4) = KS(2,3) + KS(1,3) + KS(0,3) = 6+3+1 = 10 \end{aligned}\right\} \quad (2)$$

Міркуючи подібним чином, визначимо *KS*(3, 4) і одержимо (3), а потім, узагальнивши (1), (2) та (3), запишемо рекурентні співвідношення для визначення *KS*(*i*, *j*), які пропонуємо довести самостійно. Зауважимо, що кількість доданків у сумі дорівнює *i*+1.

$$\left.\begin{aligned} &KS(3,1) = 1;\\ &KS(3,2) = KS(3,1) + KS(2,1) + KS(1,1) + KS(0,1) = 1+1+1+1 = 4;\\ &KS(3,3) = KS(3,2) + KS(2,2) + KS(1,2) + KS(0,2) = 4+3+2+1 = 10;\\ &KS(3,4) = KS(3,3) + KS(2,3) + KS(1,3) + KS(0,3) = 10+6+3+1 = 20 \end{aligned}\right\} \quad (3)$$

Отже, остаточно маємо:

$$KS(1,j) = j,\ KS(i,1) = 1,\ KS(0,j) = 1,$$

$$KS(i,j) = KS(i,j\text{-}1)+KS(i\text{-}1,j\text{-}1)+KS(i\text{-}2,j\text{-}1)+\ldots+KS(1,j\text{-}1)+KS(0,j\text{-}1) \quad (4)$$

Останні рекурентні співвідношення дають змогу скласти алгоритм розв’язування задачі, але для його спрощення проведемо додаткове дослідження з допомогою Microsoft Excel (див. таблицю 2). Проаналізувавши знайдені в ній результати (2) та (3), помічаємо, наприклад, що *H*9 = *H*8 + *G*8 + *F*8 + *E*8 + *D*8 + *C*8 = 252, тобто значення вибраної комірки *X*(*i*, *j*) дорівнює сумі значень

комірок діапазону $X(i - 1, 0)$:$X(i - 1, j)$. Але тоді $H9 = H8 + G9$, тобто $X(i, j) = X(i, j\text{-}1)$: $X(i - 1, j)$.

A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*
	кількість пропусків, які треба розмістити											
	i / *j*	0	1	2	3	4	5	6	7	8	9	10
для розміщення пропусків	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	2	3	4	5	6	7	8	9	10	11
	3	1	3	6	10	15	21	28	36	45	55	66
	4	1	4	10	20	35	56	84	120	165	220	286
	5	1	5	15	35	70	126	210	330	495	715	1001
	6	1	6	21	56	126	252	462	792	1287	2002	3003
	7	1	7	28	84	210	462	924	1716	3003	5005	8008
	8	1	8	36	120	330	792	1716	3432	6435	11440	19448
	9	1	9	45	165	495	1287	3003	6435	12870	24310	43758
	10	1	10	55	220	715	2002	5005	11440	24310	48620	92378
	11	1	11	66	286	1001	3003	8008	19448	43758	92378	184756
	12	1	12	78	364	1365	4368	12376	31824	75582	167960	352716

місця	48	1	48	1176	19600	249900	2598960	22957480	177100560	1217566350	7575968400	43183019880
	49	1	49	1225	20825	270725	2869685	25827165	202927725	1420494075	8996462475	52179482355
	50	1	50	1257	22100	292825	3162510	28989675	231917400	1652411475	10648873950	62828356305
	51	1	51	1326	23426	316251	3478761	32468436	264385836	1916797311	12565671261	75394027566
	52	1	52	1378	24804	341055	3819816	36288252	300674088	2217471399	14783142660	90177170226
	53	1	53	1431	26235	367290	4187106	40475358	341149446	2558620845	17341763505	1,07519E+11

таблиця 2

Виходячи з цього, з (4) одержуємо остаточні робочі формули:

$$KS(1, j) = j,\ KS(i, 1) = 1,\ KS(0, j) = 1,\ KS(i,j) = KS(i,j\text{-}1)+KS(i\text{-}1,j) \qquad (5)$$

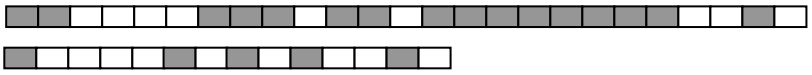
Для перевірки та закріплення формул (5) пропонуємо:

1. Обчислити з її допомогою та перевірити генеруванням кількість смужок, взявши за початкові:

input.txt :	*4 0*	*4 2 2 2*	*5 2 1 2*	*10 4 1 2 1 1*
output.txt:	*1*	*0*	*3*	*15*

мал. 3

2. Обома способами визначити *KS* для смужок, наведених на малюнку 4.



мал. 4

Наголосимо на деяких цікавих особливостях задачі. Врахувавши (*), не вплинемо на результат, якщо перетворимо початкову смужку, наведену в умові задачі і зображену на малюнку 4 (N = 25, K = 5, код 2, 3, 2, 8, 1). Визначивши для обох смужок s_1 = 20, Ns_1 = 5, $K_1 + 1 = 6$ та s_2 = 14, Ns_2 = 5, $K_2 + 1 = 6$, бачимо, що $Ns_1 = Ns_2$ і $K_1 + 1 = K_2 + 1$, отже для цих смужок KS буде однакове. Назвемо смужки з однаковими Ns та $K + 1$ *еквівалентними* і будемо першим кроком алгоритму виконувати заміну початкової смужки на її *мінімальну еквівалентну*, назвавши цю дію *мінімізацією* смужки. Це принесе значний ефект при визначенні KS генеруванням нових смужок. Наведемо алгоритм мінімізації:

1) Початок. п. 2
2) Визначити суму коду (s'). п. 3
3) Одержати новий код (s''), замінити кожне його число на 1. п. 3
4) N := s' - s''. п. 4
5) Кінець.

с) ***Пошуки ефективного алгоритму розв'язування задачі***.[2] Рекурсивний алгоритм, який не потребує доведення і оцінки на складність, легко одержати, використавши (5). Але необхідно дослідити, в яких межах він ефективно працюватиме без використання багаторозрядної арифметики, а також, чи можна його застосувати для багаторозрядних чисел. Серед проблем можуть бути переповнення через надто велике заглиблення в рекурсію та не допустимий час виконання.

```
program Strip _REC; {рекурсивна програма}
  var n,k,s,i,j,b:integer;f:text;
      a:array[1..50]of longint; p:array[1..50,1..50]of longint;
  function KS(i,j:longint):longint;
  begin
    if i=1 then KS:=j
    else if j=1 then KS:=1 else KS:=KS(i,j-1)+KS(i-1,j)
   end;
begin
```

[2] Алгоритми будемо наводити у вигляді програм на мові Turbo Pascal.
[3] Тут і далі обмеження для масивів взято довільно.

```
   assign(f,'n3.dat');reset(f); read(f,n,k);
   for i:=1 to k do begin read(f,a[i]);s:=s+a[i] end;
   close(f);
   i:=n-(s+(k-1));j:=k+1;
   assign(f,'n3.res');rewrite(f); writeln(f,KS(i,j));close(f);
end.
```

Значно ефективнішим є циклічний алгоритм, який передбачає з допомогою вкладених циклів для обчислення *KS*(5, 6) послідовно, рядок за рядком, визначити таблицю 6×6:

```
program Strip_For; {циклічна програма}
    var n,k,s,i,j,b:longint;f:text;
    a:array[1..256]of longint; p:array[1..100,1..100]of longint;
begin
    assign(f,'n3.dat');reset(f);read(f,n,k);
   for i:=1 to k do begin read(f,a[i]);s:=s+a[i] end;
   close(f);
   b:=n-(s+(k-1));
   for i:=1 to b+1 do p[1,i]:=1;  for j:=1 to k+1 do p[j,1]:=1;
   for j:=2 to k+1 do
     for i:=1 to b+1 do  p[i,j]:=p[i-1,j]+p[i,j-1];
   assign(f,'n3.res');rewrite(f); writeln(f,p[b+1,k+1]);close(f);
end.
```

Крім того, в циклічному алгоритмі простіше використати ”довгу арифметику”. Але слід врахувати, що, наприклад, при застосуванні текстового представлення багатоцифрових чисел, коли кожен елемент таблиці матиме тип *string* (256 байт), виникнуть проблеми переповнення пам’яті, тому краще замість одночасного обчислення прямокутної таблиці багаторазово обчислювати лінійну таблицю. Можна знайти інші виходи, наприклад, застосування динамічної пам’яті. Цим багато учнів і закінчать роботу над подібною задачею, хоча одержану програму ще не можна назвати ефективною.

Для пошуку справді ефекивного алгоритму ще раз скористаємось таблицею 2. З неї перш за все можна побачити межі дії алгоритму без використання багаторозрядних чисел. Права та нижня межі таблиці відсутні, що вказує на можливе продовження її в цих напрямках. Пунктирні межі між рядками №13-47 вказують на розрив таблиці. Нижній фрагмент (рядки 48-53) наведено лише для того, щоб указати одне із значень, що виходить за межі типу *longint* - *KS*(10, 53) = 1,07519E+11. У лівому верхньому куті таблиці виділено прямокутну область,

яка ілюструє обчислення *KS*(5, 6) = 252, що дорівнює кількості смужок наведеного в умові задачі зразка (мал. 4). Як виявляється, таблиця відображає відомий числовий трикутник біноміальних коефіцієнтів Паскаля, якщо за його вершину взяти її лівий верхній кут. У цій частині таблиці заштрихованими та білими клітинками показано ”поверхи” трикутника Паскаля. На перший погляд, обчислювати значення таблиці з допомогою формули числа комбінацій $C_n^m = n!/(m!(n-m)!)$ не зрозуміло як, адже, наприклад, $KS(5, 6) = C_{10}^5$. Але, представивши частину таблиці 2, в дещо іншому вигляді, де для вирівнювання ширини стовпчиків всі числа замінено відповідними записами зразка C_j^i і сірими лініями виділено рядки трикутника Паскаля (табл. 3), легко встановити залежність між *i, j* з одного боку та *n*, *m* з іншого.

C	***D***	***E***	***F***	***G***	***H***	***I***	***J***	***K***	***L***	***M***
0	1	2	3	4	5	6	7	8	9	10
C_0^0	C_1^1	C_2^2	C_3^3	C_4^4	C_5^5	C_6^6	C_7^7	C_8^8	C_9^9	C_{10}^{10}
C_1^0	C_2^1	C_3^2	C_4^3	C_5^4	C_6^5	C_7^6	C_8^7	C_9^8	C_{10}^9	C_{10}^{10}
C_2^0	C_3^1	C_4^2	C_5^3	C_6^4	C_7^5	C_8	C_9^7	C_{10}^8	C_{11}^9	C_{10}^{10}
C_3^0	C_4^1	C_5^2	C_6^3	C_7^4	C_8^5	C_9^6	C_{10}^7	C_{11}^8	C_{12}^9	C_{10}^{10}
C_4^0	C_5^1	C_6^2	C_7^3	C_8^4	C_9^5	C_{10}^6	C_{11}^7	C_{12}^8	C_{13}^9	C_{10}^{10}
C_5^0	C_6^1	C_7^2	C_8^3	C_9^4	C_{10}^5	C_{11}^6	C_{12}^7	C_{13}^8	C_{14}^9	C_{10}^{10}
C_6^0	C_7^1	C_8^2	C_9^3	C_{10}^4	C_{11}^5	C_{12}^6	C_{13}^7	C_{14}^8	C_{15}^9	C_{10}^{10}
C_7^0	C_8^1	C_9^2	C_{10}^3	C_{11}^4	C_{12}^5	C_{13}^6	C_{14}^7	C_{15}^8	C_{16}^9	C_{10}^{10}
C_8^0	C_9^1	C_{10}^2	C_{11}^3	C_{12}^4	C_{13}^5	C_{14}^6	C_{15}^7	C_{16}^8	C_{17}^9	C_{10}^{10}
C_9^0	C_{10}^1	C_{11}^2	C_{12}^3	C_{13}^4	C_{14}^5	C_{15}^6	C_{16}^7	C_{17}^8	C_{18}^9	C_{10}^{10}
C_{10}^0	C_{11}^1	C_{12}^2	C_{13}^3	C_{14}^4	C_{15}^5	C_{16}^6	C_{17}^7	C_{18}^8	C_{19}^9	C_{10}^{10}
C_{11}^0	C_{12}^1	C_{13}^2	C_{14}^3	C_{15}^4	C_{16}^5	C_{17}^6	C_{18}^7	C_{19}^8	C_{20}^9	C_{10}^{10}

таблиця 3

Очевидно, що $m = i$, $n = i + j - 1$, тобто $KS(i, j) = C_{i+j-1}^i$, а, врахувавши позначення (*), остаточно одержимо: $KS = C_{i+j-1}^i$ (6)

Без значної і систематичної практики застосування комбінаторики, що характерно для більшості учнів, (6) встановити аналітично не легко, а з допомогою таблиці 3 вона стає очевидною.

Покажемо, як можна одержати цю формулу без наведених вище міркувань. Зрозуміло, що загальна кількість білих клітинок у смужці після усіх вставок дорівнюватиме $j + i$. Якщо прийняти, що вставка кожної з *i* додаткових бі-

лих клітинок можлива тільки між двома іншими білими клітинками, то першу з i білих клітинок можна розмістити j - 1 способами, наступну - j способами і т.д., а останню можна розмістити $i + j$ - 1 способами. Отже кількість таких розміщень дорівнюватиме $A_{i+j-1}^{\,i} = (i + j - 1)! /(i + j - 1 - i)! = (i + j - 1)! /(j - 1)!$. Але тут враховано всі розміщення білих клітинок, а в нашій задачі білі клітинки, які вставляються - рівноправні, тому одержане число слід ще поділити на кількість перестановок з i елементів, тобто на $i!$, у результаті чого отримаємо рівність (6).

Цю формулу важко побачити учневі, який з відомих причин навіть не обов'язково знайомий з комбінаторикою, тому, для нього більш звична математична модель, яка використовує (5). Поза сумнівом, вищеописана схема розв'язування даної задачі, тобто: рекурсивна програма ***Strip_REC*** $\Rightarrow$ циклічна програма ***Strip_FOR*** $\Rightarrow$ програма на базі (6) - це найбільш природний шлях пошуку повного і ефективного розв'язання, тому можна порадити учителям саме такої схеми дотримуватись при роботі з учнями над подібною задачею.

Необхідно також зауважити, що розглянуту задачу слід давати на олімпіаді диференційовано, відповідно до віку учасників. Це можна зробити за рахунок зміни формату вхідних і вихідних даних, а також тестів, для 8-9 класів достатньо обмежитись використанням стандартних числових типів, що дозволить обмежитись рекурсивним чи циклічним алгоритмами, для 10-11 класів можна дати наведені технічні умови, що вимагатиме опрацювання багаторозрядних чисел і може привести до застосування комбінаторики.

Отже, доведення формули (6), хоч і доступне далеко не всім учасникам олімпіади, але радикально спрощує алгоритм та забезпечує одержання повного розв'язку даної задачі, в тому числі на множині багаторозрядних чисел. Наведемо відповідну програму:

program ***Strip***;[4]

[4] Програма враховує застосування "довгої арифметики" з представленням чисел у вигляді рядкових величин, необхідні процедури і функції для реалізації базових алгоритмів опрацювання багаторозрядних чисел, зокрема порівняння чисел, додавання, віднімання, множення багацифрового числа на одноцифрове, множення багатоцифрових чисел, визначення факторіалу та цілочисельне ділення наведено нижче, передбачається, що вони включені у модуль Long_Ar, який оголошений у програмі.

```
 uses Long_Ar;
 var n,k,i,s,j,m:integer;a:array[1..101] of integer;f,f1:text;b, ks:string;
begin{Strip}
 Assign(f,'input.txt');  Reset(f); Assign(f1,'output.txt'); ReWrite(f1);
 Read(f,n); Read(f,k);
 for m:=1 to k do  Read(f,a[m]);
 if (n<1) or (n>200) or (k<0) or (k>(n+1)/2) then
    begin WriteLn(f1,'ERROR'); Close(f); Close(f1) end
 else  for m:=0 to k do s:=s+a[m+1];
 j:=k+1; i:=n-s-k+1;
 MoDiv(Fact(i+j-1), Mult(Fact(i), Fact(j-1)),b, ks);
 WriteLn(f1,ks); Close(f); Close(f1);
end.{end  Strip}
```

Далі наведено підпрограми модуля Long_Ar.

```
function Comp(s1,s2:string):boolean;{порівняння st1 та st2}
  var j,l1,l2:byte;st:string;
begin l1:=Length(s1);l2:=Length(s2);
  if l1<l2 then for j:=l1-l2 downto 1 do s1:='0'+s1
  else          for j:=l2-l1 downto 1 do s2:='0'+s2;
  if s1>=s2 then Comp:=true else Comp:=false
end; {end порівняння st1 та st2}
function Add(st1,st2:string):string;{Add=st1+st2}
  var i,l1,l2,pm,s1,s2,s:byte;cod:integer;st:string;
begin
 if st1[0]<st2[0] then begin st:=st1;st1:=st2;st2:=st end;
 st1:='0'+st1;l1:=Length(st1);l2:=Length(st2);pm:=0;
 for i:=l1-l2 downto 1 do st2:='0'+st2;
 for i:=l1 downto 1 do                                  begin
  Val(st1[i],s1,cod);Val(st2[i],s2,cod);s:=s1+s2+pm;
  st1[i]:=Chr((s mod 10)+48);pm:=s div 10          end;
 if st1[1]='0' then Delete (st1,1,1);Add:=st1
end;{end Add}
function Sub(st1,st2:string):string;{Sub=st1-st2}
  var i,l1,l2,pm,s1,s2,s:byte;cod:integer;st:string;
begin
  if st1[0]<st2[0] then begin st:=st1;st1:=st2;st2:=st end;
  l1:=Length(st1);l2:=Length(st2);pm:=0;
  if st1=st2 then Sub:='0'
  else                                                 begin
    for i:=l1-l2 downto 1 do st2:='0'+st2;
    for i:=l1 downto 1 do                       begin
      Val(st1[i],s1,cod);Val(st2[i],s2,cod);
      if s1>=s2+pm then begin s:=s1-(s2+pm);pm:=0 end
   else       begin s:=(s1+10)-(s2+pm);pm:=1 end;
```

```
    st1[i]:=Chr(s+48)                              end;
   while st1[1]='0' do Delete (st1,1,1);  Sub:=st1   end
end;{end Sub}
function Mult1(st1,st2:string):string;{Mult1=st1*st2, st2 - одноцифрове}
   var i,l1,l2,pm,s1,s2,cod,s:word;st:string;
begin
  st1:='0'+st1;l1:=Length(st1);l2:=Length(st2);pm:=0;
  for i:=l1 downto 1 do                                  begin
    Val(st1[i],s1,cod);Val(st2,s2,cod);s:=s1*s2+pm;
    st1[i]:=Chr((s mod 10)+48);pm:=s div 10          end;
  if st1[1]='0' then Delete(st1,1,1);Mult1:=st1
end;{end Mult1}
function Mult(st1,st2:string):string;  {Mult = st1*st2}
  var i,j,l:word;st0,st:string;
begin
  st0:='';l:=Length(st2);
  for i:=l downto 1 do          begin
     st:=Mult1(st1,Copy(st2,i,1));
     for j:=1 to l-i do st:=st+'0';
     st0:=Add(st,st0)            end;
  Mult:=st0;
end; {end Mult}
function Fact(n:integer):string; { n!}
   var i:integer; f, f1:string;
begin
function Fact(n:integer):string;
 var i:integer;f,f1:string;
begin
  f:='1';
  if n>1 then for i:=2 to n do            begin
                  str(i,f1);f:=Mult(f,f1) end;  Fact:=f;
end; { end Fact}
procedure MoDiv(st1, st2:string; var stmod, stdiv:string);{mod, div}
  var i,l,k, j:integer;st_1:string; bZero: boolean;
begin
  st_1:='';stdiv:='';i:=1;l:=Length(st1);
  repeat
     k:=0;  if st2 = '' then break;
    if st2 = '0' then begin st2 := 'ERROR';Break end;
    while not Comp(st_1,st2) do                    begin
       st_1:=st_1+st1[i]; Inc(i); stdiv:=stdiv+'0';
       if i>length(st1) then break                 end;
    Delete (stdiv,Length(stdiv),1);
    while st_1[1]='0' do                          begin
       if st_1 = '' then break; Delete (st_1,1,1) end;
```

```
    while Comp(st_1,st2) do              begin
      st_1:=Sub(st_1,st2);  k:=k+1  end;
    stmod:=st_1; if st_1 = '' then stmod := '0'; stdiv:=stdiv+Chr(k+48);
    while Length(stdiv) > Length(st1) do Delete(stdiv, Length(stdiv), 1);
  until i>l;
  while stdiv[1]='0' do                              begin
    if stdiv= '0' then break; Delete (stdiv,1,1) end;
end;  {end MoDiv}
```

Зауважимо, що програма ***Strip*** не у всіх випадках повертає правильне значення *ks*. Проаналізуємо кілька вхідних файлів і результати занесемо до таблиці 4 (неуточнені вихідні файли наведено у додатку).

input1.txt: 8 3 1 2 1
input2.txt: 25 5 2 3 2 8 1
input3.txt:
111 52 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
input4.txt:
200 50 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
input5.txt:
200 60 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
input6.txt:
200 62 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
input7.txt:
200 65 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
input8.txt:
200 75 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
input9.txt:
200 90 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
input10.txt:
200 20 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 11

тест	*N*	*K*	*i*	*j*	*i+j-1*
1	8	3	6	4	9
2	25	5	5	6	10
3	111	52	8	53	60
4	200	50	101	51	151
5	200	60	81	61	141
6	200	62	77	63	139
7	200	65	71	66	136
8	200	75	51	76	126
9	200	90	21	91	111
10	200	20	151	21	171

таблиця 4

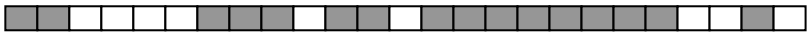
Згідно (6), наприклад, у тесті №10 необхідно обчислити 20!, 151! та 171!, але представлення багаторозрядних чисел у рядковому вигляді ставить під сумнів правильність визначення факторіалів таких великих чисел та значення неповної частки при їх цілочисельному діленні. Для уникнення цієї проблеми слід модернізувати програму ***Strip***, використавши перетворення:

$$C_{171}^{151} = \frac{171!}{151!\cdot 20!} = \frac{152\cdot 153\cdot ...170\cdot 171}{20!}$$

З огляду на елементарність при наявності модуля ***Long_Ar*** пропонуємо виконати це перетворення самостійно та одержати результати до наведених тестів.

На цьому дослідження задачі можна вважати цілком вичерпаним. Проте деякі виявлені факти дозволяють зробити додаткові висновки. Наприклад, ми поки-що не побачили практичної потреби у згаданій вище мінімізації смужки. Але можна запропонувати нову задачу:

Задача 15′. На смужці клітинкового паперу висотою в одну і довжиною *N* клітинок деякі клітинки зафарбовані в чорний і білий кольори. Кодом



клітинки є непуста множина чисел – кількості ідучих підряд чорних клітинок зліва направо. Наприклад, для наведеної смужки кодом буде множина {2, 3, 2, 8, 1}. При цьому кількість білих клітинок, які розділяють групи чорних ніде не враховуються (головне, щоб дві сусідні групи розділені принаймні однією білою клітинкою). Задача полягає в тому, щоб знайти кількість смужок довжини *N*, які відповідають заданому коду.

Вхідні дані. У єдиному рядку файлу файла *input.txt* записано число *N* – довжина смужки ($1 \le N \le 25$), потім число *K* ($0 \le K \le (N+1)/2$) – кількість чисел у коді і далі *K* чисел, що визначають код.

Вихідні дані. У вихідний файл *output.txt* записується кількість смужок довжини *N*, які відповідають заданому коду.

Задача 15′ лише зовнішньо дуже схожа на задачу 15. Насправді ж з умови випливає, що слід враховувати всі смужки з послідовностями чорних клітинок, що є перестановками з повторенням у коді. Звідси легко вивести формулу для визначення *Ks* у задачі 15′. У зв'язку з цим у файлі *input.txt* взято обмеження $1 \le N \le 25$. Очевидно, що перша задача легша від другої, більше того, вона є її частинним випадком, а побачити це допоможе введене нами поняття мінімізації смужки.

Не будемо вказувати на інші виявлені факти. По-перше, це дасть можливість уважним читачам зробити власні маленькі відкриття, по-друге, нема потреби розкривати таємниці ще кількох нових задач, які можуть бути представлені у якості олімпіадних. Зауважимо лише, що ми намагались прослідкувати, як, шукаючи розв'язок однієї задачі можна прийти до інших, можливо не менш цікавих, а то й зовсім несподіваних, тобто продемонструвати шлях *від задачі до задачі*.

Надзвичайно цінною якістю особистості є вміння швидко, за екстремальних обставин, як буває на олімпіадах, шукати розв'язки задач, та не менш важливе вміння без поспіху, але грунтовно, проводити повне дослідження поставленої проблеми, що часто приводить до нових проблем.

ЛІТЕРАТУРА:

1. С. Окулов. Программирование в алгоритмах. Москва. Бином. Лаборатория Знаний. 2002;

додаток

output1.txt: 10
output2.txt: 252
output3.txt: 2558620845
output4.txt:
22294864007509202019319906 27
output5.txt:
39349707693707417946654498477149309918680
output6.txt:
21045998620349722340045216678707617713760
output7.txt:
52164357285294491100042489748515630828 00
output8.txt:
61643832300695848134415168385026326 0
output9.txt:
23225278051098322489290
output10.txt: 0