СЪСТЕЗАТЕЛНИТЕ ЗАДАЧИ ПО ИНФОРМАТИКА – ОСНОВНИ ИЗИСКВАНИЯ

Антон Ив. Моллов,
Бисерка Б. Йовчева, Румяна И. Ганева

COMPETITIVE TASKS INFORMATICS - BASIC REQUIREMENTS

Anton Iv. Mollov,
Biserka B. Yovcheva, Rumyana I. Ganeva

ABSTRACT: *The article presents the main features and requirements for forming problems for competitions and training in informatics – structure, content, formatting, saving files and requirements to test cases to check for correct code – number, content, shaping.*

KEYWORDS: *Education, informatics, programming, competition problems, test cases.*

1. Изисквания към оформянето на състезателните задачи

Решаването на задачи с компютър чрез програмиране на конкретен алгоритмичен език (в момента актуален е C++) е друга основна дейност на учениците в обучението по информатика в училище. Съставянето на програми и непосредствената работа със среди за програмиране реализират специфични дидактически цели като: формиране на знания и практически умения за работа с конкретна среда за програмиране и конкретна операционна система, прилагане на различни алгоритми за решаване на реални практически задачи, творческо прилагане на усвоените знания и умения за ефективно решаване на проблеми с компютър и други [4].

Всяка задача от състезание по информатика или задача за тренировки има определена структура. За да фиксираме

отделните задължителни структурни елементи нека разгледаме една примерна задача от Национален пролетен турнир по информатика 2014 г. за група D с автор Валентина Спасова [7].

Задача D1. Маршрут /route/

Представител на фирма CompSys има маршрут от ***n*** *последователни града, номерирани с числата от* ***1*** *до* ***n.*** *Във всеки от градовете той продава нови компютърни конфигурации или изкупува стари. В даден град за даден ден се извършват само продажби, само изкупуване или нищо. Приходите от продажбите в съответния град се отразяват с положително число, разходите за изкупуване – с отрицателно число, а липсата на продажби и изкупуване – с числото* ***0****. Представителят тръгва по маршрута, ако има поне един град с продажби. При това той минава през всички градове от маршрута.*

За поддържане на статистически данни е необходимо да се знае кой е печелившият подмаршрут и неговият оборот. Печеливш подмаршрут е част от маршрута с максимален оборот. Оборотът за всеки подмаршрут се изчислява като сума от приходите и разходите за градовете в него. Подмаршрутът се задава с номерата на първия и последния град в него и включва всички градове между тях. Ако има няколко печеливши подмаршрута се пази първият (този, с най-малък номер на първия град от подмаршрута).

Напишете програма ***route****, която по дадена информация за сумите, които е получил представителят от всеки град от маршрута, определя печелившия подмаршрут и неговия оборот.*

Вход

От първия ред на стандартния вход се въвежда цяло положително число ***n*** *– броят на градовете в маршрута.*

От втория ред на стандартния вход се въвеждат n цели числа в интервала от ***-10000*** *до* ***100000*** *– сумите, които представителят получава във всеки град от маршрута.*

Изход

На първия ред на стандартния изход програмата трябва да изведе оборота на печелившия подмаршрут и номерата на първия и последния град, които определят този подмаршрут.

Ограничения

$1 \leq n \leq 1000000$

Примери :

Пример 1

Вход

13
25 -21 -12 33 59 -23 -11 9 45 -97 37 35 12
Изход
112 4 9

Ще започна първо с изисквания за оформяне на материалите /задачите/ за едно състезание.

Всяка задача се записва в отделна папка с име – номерът на задачата, последван от името на задачата. Примерно: 1-route. Имената на папките и файловете са на английски. Папката съдържа две папки: author и tests. Папка author съдържа файл с разширение route.cpp, който съдържа авторското решение на задачата. Освен това в тази папка задължително се съдържат още два файла с разширение route_bg.doc и route_en.doc, в които е дадено условието на задачата на български и на английски език, и един файл route-author.doc с анализ от автора на решението на задача маршрут /route/.

Във втората папка tests са всички тестове за задачата. Всеки тест е в отделен файл. Всеки тест се състои от два файла – route.01.in /за вход/ и route.01.sol /за изход /. Броят на тестовете обикновено е от 10+1 /примерния нулев тест/ до 25+1. Резултатите от проверяващата система са върху всички тестове без нулевия. Трите папки със задачи са в една обща папка с име латинска буква, показваща възрастовата група /в конкретния случай – D/. Папка D е препоръчително да съдържа един текстов документ, в който са записани и трите задачи, като всяка задача започва на нова страница.

Всяка задача съдържа:

- Условие на задачата – част от текста изяснява какви обекти се разглеждат и какво е известно за тях. Задачата трябва да бъде описана като последователност от задания, които ученикът трябва да програмира. Заданията трябва да са точно формулирани (без двусмислици).
- Вход – описание на входните данни, които ще се въвеждат .

За нашия пример: От първия ред на стандартния вход се въвежда цяло положително число n – броя на градовете в маршрута. От втория ред на стандартния вход се въвеждат n цели

числа в интервала от -10000 до 100000 – сумите, които представителят получава във всеки град от маршрута.

• Изход – описание на изходните данни при правилно решение на задачата.

За нашия пример: На първия ред на стандартния изход програмата трябва да изведе оборота на печелившия подмаршрут и номерата на първия и последния град, които определят този подмаршрут.

Обикновено четенето на данни от нашата програма се нарича вход (като информацията се нарича входни данни), докато връщането на резултат се нарича изход (като върнатата информация се казва изходни данни).

В най-честия случай и входът и изходът се четат и пишат в конзолата. Въпреки, че тя изглежда като едно цяло за потребителя, реално четенето и писането се извършват на различни места – в така наречените стандартен вход и стандартен изход. Но тъй като текстът и от двете се показва на потребителя на едно и също място, се създава илюзията, че те са едно. Примери за състезания, в които се ползва конзолата за вход и изход, са българските ученически състезания, IOI и много други.

• Ограничения – Това е информация за типа данни които е добре да се ползва при решението на задачата. Едно от най-важните неща на всеки компютърен алгоритъм е неговата ефективност – колко време ще отнеме изпълнението му за дадени ограничения на входните данни.

За нашия пример: $1 \leq n \leq 1000000$

• Примери за вход и изход – един или няколко примера за това как трябва да работи решението на задачата /създаденият алгоритъм/. При въвеждане в конзолата на дадения вход програмата трябва да върне дадения изход.

Вход
13
25 -21 -12 33 59 -23 -11 9 45 -97 37 35 12
Изход
112 4 9

- Обяснение на примера – при необходимост, задачата може да съдържа обяснение на дадения пример. В нашият случай, това липсва.

Нека доразгледаме нашия пример. В файл route-author.doc с анализ от автора на решението на задача маршрут / route/, четем:

Ако с вектора a_1, a_2, ..., a_n се представят сумите за всеки град от маршрута, то се търси непрекъснат подвектор, сумата от елементите на който е максимална за всички непрекъснати подвектори.

Стандартен вариант за решаване на тази задача е да се разгледат всички двойки цели числа L и R, удовлетворяващи условието $1 \le L \le R \le n$. За всяка такава двойка да се намери сумата *s* от елементите на подвектора $a[L..R]$ и да се провери дали тази сума е по-голяма от максималната намерена до момента s*max*. Решението в този случай има сложност $O(n^3)$.

Подобрение на този алгоритъм се получава, когато сумата на елементите на подвектора $a[L..R]$ се изчислява за една стъпка вместо необходимите $R-L+1$ стъпки в предходния алгоритъм. Това се постига като се сумира елементът $a[R]$ към изчислената вече сума на подвектора $a[L..R-1]$ – получава се сумата на елементите на подвектора $a[L..R]$.Този алгоритъм има сложност $O(n^2)$.

Алгоритъм с линейна сложност $O(n)$: В началния момент максимума е 0. Предполага се, че задачата е решена за подвектора $a[1..i-1]$. Максималната подсума на първите i елемента е равна или на максималната сума на първите $i-1$ елемента (която означаваме с *smax*) или на максималната подсума на вектора, завършващ в i–тия елемент (която означаваме с *s*). Щом *s* стане отрицателно, се игнорира този резултат, защото той само ще намалява оборота. В този случай *s* се нулира и лявата граница се измества върху следващия елемент. Когато *s* стане по-голяма от *smax*, то се запомня дясната граница за достигнатата текуща максимална сума.

Решението в файл с разширение route.cpp съдържа следната програмата:

```
#include<iostream>
using namespace std;
long long int FindMaxSeq(int*a, int n, int&L, int&R)
{
   long long int smax=0,s=0;
   int l=0;
   for(int i=0;i<n;i++)
   {
      s+=a[i];
      if(s<0)
      {
         s=0;
         l=i+1;
      }
      if(s>smax)
      {
         smax=s;
         L=l;R=i;
      }
   }
   return smax;
}
int main()
{
   int *a,n,L,R,m;
   cin>>n;
   a=new int[n];
   long long int s,max=0;
   int maxL,maxR,maxD;
   for(int i=0;i<n;i++)
         cin>>a[i];
   s=FindMaxSeq(a,n,L,R);
```

```
    cout<<s<<" "<<L+1<<" "<<R+1<<endl;
return 0;
}
```

Тези изисквания са приети от Националната комисия по Информатика към СМБ и са публикувани в сайт за националните състезания по информатика за ученици INFOS [6].

2. Изисквания към условията на задачата

2.1. Условието на задачата трябва да бъде съобразено с учебния материал предвиден за съответната възрастова група.

2.2. Условието на задачата трябва да бъде съобразено с възрастовите особености на учениците.

- Текста да е забавен и лесен за четене.
- В условието на задачата ясно да личат всички частни случаи и ограничения.
- Спецификацията на вход и изход да бъдат описани точно без двусмислия.
- Задачата да провокира творчески мислене на учениците, да формира и обогатява основни ИТ умения, като например анализ на задачите, проектиране на алгоритми и структури от данни, програмиране и тестване**.**
- Условието на задачата трябва да предвижда няколко варианта за решение, което от своя страна ще даде възможност по-ефективните реализации да бъдат оценени по-високо.

3. Изисквания към тестовете за проверката

Тестването при разработката на софтуер е не по-малко важно от писането на кода. В сериозните софтуерни корпорации на един програмист се пада поне един тестер. Например в Microsoft на един програмист, който пише код (software engineer) се назначават средно по двама души, които тестват кода (software quality assurance engineers). Тези разработчици също са програмисти, но не пишат основния софтуер, а пишат тестващи

програми за него, които позволяват цялостно автоматизирано тестване.

Опитните програмисти знаят, че ако напишат код и той не е тестван, това означава, че той още не е завършен. В повечето софтуерни фирми е недопустимо да се предаде код, който не е тестван. В софтуерната индустрия дори е възприета концепцията за unit testing –автоматизирано тестване на отделните единици от кода (методи, класове и цели модули).

Unit testing означава за всяка програма да пишем и още една програма, която я тества дали работи коректно. В някои фирми първо се пишат тестовете за програмата и най-накрая се пише самата програма. Темата за unit testing е много сериозна и обемна. Засега, нека се фокусираме върху ръчното тестване, което всеки един програмист може да извърши, за да се убеди, че неговата програма работи коректно [5]. Една програма е коректна, ако работи коректно за всеки възможен валиден набор от входни данни. Тестването е процес, който цели да установи наличие на дефекти в програмата, ако има такива. За съжаление всички възможни набори входни данни за една програма обикновено са неизброимо много и не може да се тества всеки от тях.

Затова в практиката на софтуерното тестване се подготвят и изпълняват такива набори от входни данни (тестове), които целят да обхванат максимално пълно всички различни ситуации (случаи на употреба), които възникват при изпълнение на програмата. Този набор има за цел с минимални усилия (т. е. с минимален брой и максимална простота на тестовете) да провери всички основни случаи на употреба.

Трябва да се подберат такива примери, които в пълнота покриват различните случаи, които алгоритъм трябва да преодолее.

3.1. Тестовете за проверката трябва да обхващат всички частни случаи.

Те трябва да са с подходящи примери.

Тестването е добре да започва от един пример, с който обхващаме типичния случай в задача и може да бъде тестван „на ръка“.

- Лесен тест / прости примери / – това са примери, които с помощта на лист хартия и химикал може да се направи тестване „на ръка“. Примерите трябва от една страна да не са лесни за тествания алгоритъм, а от друга да са достатъчно прости, за да се разпишат бързо и лесно. Такива примери се наричат "добри представители на общия случай".

Например ако се реализира алгоритъм за сортиране на масив в нарастващ ред, удачно е да вземе пример с 5–6 числа, сред които има 2 еднакви, а останалите са различни. Числата трябва първоначално да са подредени в случаен ред. Това е добър пример, понеже покрива много голяма част от случаите, в които тествания алгоритъм трябва да работи.

- Сериозен тест на обичайния случай – Тестване на програмата с по-голям и по-сложен пример, за да видим как се държи тя в по-сложни ситуации.

Тестът трябва да е с по-голям обем данни, отколкото ръчния тест, но все пак трябва да може да се провери дали изхода от програмата е коректен. Например за задача: сортиране на числа. Един сериозен тест е да се генерира поредица от 100 или дори 1000 случайни числа и програмата да ги сортира. Проверката за коректност е лесна: трябва числата да са подредени по големина. Друг тест, който е удачен при сортирането на числа е да се въвеждат числата от 1000 до 1 в намаляващ ред. Трябва да се получат същите числа, но сортирани в нарастващ ред. Би могло да се каже, че това е най-трудният възможен тест за тази задача и ако той работи за голям брой числа, значи програмата най-вероятно е коректна.

- Тестовете за проверката не трябва да се явяват само от един тип частни случаи. Ако тестовете са еднотипни или не покриват всички частни случай, то тогава е възможно да отчетат грешна програмата като коректна. За примерната задача за сортиране има множество неподходящи примери, с които няма да можете ефективно да проверите дали алгоритъма работи

коректно. Например – пример само с 2 числа. За него алгоритъмът може да работи, но по идея да е грешен. Или пример само с еднакви числа-при него всеки алгоритъм за сортиране ще работи. При пример с числа, които са предварително подредени по големина алгоритъмът може да работи, но да е грешен.

3.2. Тестове, включващи екстремните ситуации – гранични случаи

Граничните ситуации се получават при входни данни на границата на нормалното и допустимото. При тях често пъти програмата се проваля, защото не очаква толкова малки, големи или необичайни данни, но те все пак са допустими по условие.

Възможно е да имаме екстремно малки стойности, екстремно големи стойности или просто странни комбинации от стойности. Ако по условие имаме ограничения, например до 100, стойностите около това число 100 също са гранични и могат да причинят проблеми.

3.3. Тестове, включващи проверка за бързодействие – тестове за производителност

Нека разгледаме нашата примерна задача: сортиране на масив с числа. При нея бързодействието може да се окаже много проблемно. Нека сме направили просто решение, което работи така: намира най-малкото число в масива и го разменя с числото на позиция 0. След това намира сред останалите числа най-малкото и го поставя на позиция 1. Това се повтаря докато се стигне до последното число. Този алгоритъм е известен като "метод на пряката селекция". Когато се опитваме да сортираме 10 000 случайни числа,този алгоритъм работи за под секунда на нормална съвременна машина.

Ако обаче опитаме да сортираме 300 000 случайни числа ще видим, че програмата като че ли блогира /зависва/ или работи прекалено бавно. Това е сериозен проблем с бързодействието. Конкретно за тази задача подходящо е да използваме алгоритъма за сортиране "heap sort" или “merge sort”. За някои задачи няма

нужда от бързина, защото очакваме входните данни да са достатъчно малки и тогава не е необходимо да търсим сложни алгоритми с цел бързодействие. Например задачата за сортиране на оценките на ученици от даден клас може да се реши с произволен алгоритъм за сортиране и при всички случаи ще работи бързо, тъй като броят на учениците се очаква да е достатъчно малък.

3.4. Тестовете трябва да балансират резултатите от различните решения

Ако тестовете за бавни /тъпи/ решения са повече на брой от тестовете, които различават по-добрия алгоритъм /по-бързия, с по-малък обем памет и др./, това ще доведе до оценяване на по-лошото решение с повече точки .

4. Изводи

Изискванията към комплекта, съдържащ състезателната задача за едно състезание по информатика са той да съдържа: текст на задачата, анализ на решението, авторско решение, тестове за проверка и програми за проверка (ако са необходими). Текстът на всяка задача има задължителни структурни елементи – условие на задачата, вход, изход, ограничения, примери за вход и изход.

Изисквания към тестовете за проверката са да се подберат такива примери /тестове/, които в пълнота покриват различните случаи, които алгоритъмът трябва да преодолее, да обхващат всички частни случаи, всички екстремни ситуации и да балансират резултатите от различните решения.

Настоящата разработка е финансирана със средства на фонд Научни изследвания на ШУ "Еп. К. Преславски" договор № РД-08-286/14.03.2014 г.

ЛИТЕРАТУРА

[1]. **Йовчева, Б., И. Иванова.** Първи стъпки в програмирането на C/C++. Издателство КЛМН, София, 2006.

[2]. **Йовчева, Б.** Спираловидно обучение по програмиране на 10-11 годишни деца (на базата на езика С++). Математика и математическо образование, София, 2007.
[3]. **Богданова, В., Г. Момчева.** Структуриране на учебното съдържание в извънкласни форми по информатика. сп. “Математика и информатика”, бр. 1, 2001.
[4]. **Гъров**, **К.** Задачите в обучението по информатика и информационни технологии. // Национална конференция „Образованието в информационното общество”, 2010.
[5]. **Наков, С., В. Колев** и колектив. Въведение в програмирането със C#. Академия на Телерик за софтуерни инженери, София, 2011.
[6]. **Йовчева, Б., И. Иванова, П**. **Петров**. Втори стъпки в програмирането на С/С++. Издателство КЛМН, София, 2014.
[7]. www.math.bas.bg/infos/ Сайт за националните състезания по информатика за ученици INFOS
[8]. http://infoman.musala.com/ Информатически портал ИнфоМан