КУРС „ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ”
летен семестър 2008

Техники за отбелязване на посетените върхове

Всички знаем, че като преминаваме през даден връх в някакъв граф, почти винаги се налага да го отбелязваме като посетен. Как обаче да правим това? Разбира се, зависи от типа на върховете.

Ако той е **string**, тогава в повечето случай може би е най-добре да трупаме посетените върхове в *set* (множество). Set-ът почти винаги се имплементира с някакво балансирано двоично дърво за търсене (най-често червено-черно), което означава, че можем да разчитаме на логаритмична сложност при поставянето, изваждането и изтриването (или по-точно *O(len*log(n))*, където *len* е дължината на стринга, а *n* – броят елементи в множеството). Реализации на *set* има в почти всички съвременни езици за програмиране (включително C++ и Java), което значително улеснява решението на задачата.

Много често има различни възможности за оптимизации. Например, ако в условието ни е дадено, да кажем, че стринговете ще са по-къси от 15 символа и ще се състоят само от буквите A, B и C, тогава можем да си представяме, че всъщност са ни дадени числа в троична бройна система. В този случай стандартната техника с един масив ще ни свърши работа. И по точно: ако сме прочели низа str, и сме го превърнали в десетичното число *code(str)*, то отбелязваме str като посетен така:

visited[code(str)] = true;

Тук имаме сложност само *O(len)*, защото след като получим кода на низа, индексирането в масива visited е константно. Това обаче има своята цена – ще ни трябват 3^{15} байта памет за булев масив, което е почти 15 MB. Добре е да правим подобни сметки преди да започнем да пишем.

Ако пък нямаме възможност за кодиране, може да се напънем малко и да пробваме да хешираме. Добре е предварително да знаем с колко оперативна памет разполагаме, за да можем да направим масива visited колкото се може по-голям, т.е. хеш-функцията колкото се може по-добра. Когато обаче направим проверката:

if(visited[hash(str)])

за да сме сигурни, че не се е получила колизия, е хубаво да видим дали наистина текущият връх е посетен. За тази цел за всеки намерен хеш-код може да пазим един *set* от *string*, който съдържа посетените върхове, отговарящи на дадения хеш-код, или *set* от int. който съдържа техните индекси в някаква си наша структура. Този подход е доста по-трудоемък от първоначално предложения и е по-добър от него, но има смисъл само в наистина огромни графи, тъй че е малко вероятно да ни се наложи да го ползваме в този курс.

Всъщност, когато типът на върховете е string, доста неща зависят от съответната задача и по-точно от представянето на графа. В някои случаи нелоша идея е да образуваме един *map:* връх->число и *масив:* индекс->връх. Така получаваме двупосочна връзка и можем да си работим с числата, а когато ни потрябва – да преминаваме към низовете.

Вторият вариант е типът на върховете да е **число с плаваща запетая**. В повечето случай обаче, тези числа ще идват на входа с определена точност. Примерно може да е казано така – „*числата ще са между 0 и 10 000 и ще са с точност до три знака след десетичната запетая*“. Бързо се досещаме, че можем да ги умножим по 1000 и така всъщност да ги превърнем в цели числа. Отново правим сметката – 10 000 числа, между всеки две има още 1000, това прави 10 милиона числа, за които ще ни трябват 10MB памет – нормално.

Най-често обаче типът на върховете е **целочислен**. Такива ще са и повечето задачи, които ще решаваме в курса. Както вече казахме, в този случай винаги е добре, когато можем, да пазим масив visited и да отбелязваме:

visited[node] = true;

Да разгледаме обаче един конкретен пример, а именно задачата за трите чаши (домашното за седмица 6). Тук имаме доста голям граф, който обаче е силно разреден: 1 000 000 върха с най-много 6 ребра на връх. Върховете се образуват от наредени тройки цели числа, значи масивът *visited* ще е тримерен и ще бележим:

visited[x][y][z] = true;

Интересното тук идва от инициализацията. На всеки пореден тест трябва да занулуваме този масив, което значи да минем през 100*100*100 = 1 000 000B = 1MB памет. Примерните тестове на spoj0 са 1000, което прави 1GB памет. Занулуването на 1GB неравномерно разпределена памет (т.е. 1000 пъти по 1MB)

си е доста бавна операция. За да сме абсолютно точни, ще говорим за конкретната задача, с конкретните тестове и конкретен компилатор – *gcc* (извинявам се на Java-рите, но просто този език не е подходящ за изчисления в текущия пример, тъй като се губи време за зареждане на виртуалната машина и това разваля сметките). Ако си пуснем 3 for-а по следния начин:

```
for(int i = 0; i < MAX; i++)
    for(int j = 0; j < MAX; j++)
        for(int k = 0; k < MAX; k++)
            visited[i][j][k] = false;
```

ще забележим, че решението ни върви за **41 секунди**. Ако включим оптимизациите на компилатора (-O2), ще го намалим на **14 секунди**. По-хитро обаче е да ползваме memset:

```
memset(visited, false, sizeof(visited));
```

Това ще намали времето ни за работа на **8 секунди** (със и без -O2). Ако първо прочетем трите числа от входа и след това инициализираме:

```
for(int i = 0; i < m1; i++)
    for(int j = 0; j < m2; j++)
        for(int k = 0; k < m3; k++)
            visited[i][j][k] = false;
```

ще сведем времето за изпълнение до **4 секунди**. (тестовете са генерирани random – разбира се, ако имахме *worst case* – 1000 теста, във всеки от който чашите са (100, 100, 100), то нямаше да има никаква разлика). Не решаваме обаче проблема с многократното зануляване.

Това може да стане като направим масивът не *bool*, ами примерно *int*. За всеки тест ще си пазим едно уникално число – примерно номера на теста (*test*). Ще отбелязваме като посетен така:

```
visited[x][y][z] = test;
```

и ще проверяваме дали даден връх *не* е посетен по следния начин:

```
if(visited[x][y][z] != test)
```

Вярвате или не вярвате, това свежда времето за работа до **0.11 секунди,** а с -O2 до **0.08 секунди** (между другото, на spoj0 кодът се компилира с -O2). Получихме 100 пъти по-добро решение от варианта с memset-а. Разбира се, разликите са толкова големи, само защото имаме много тестове и много голям граф, но като цяло не е лоша идея да прилагате последната техника.