Тема 22

Държавен изпит

специалност

Приложна математика

Рекурсия. Рекурсивни алгоритми. Рекурсивни структури данни – линеен списък, бинарни и n-арни дървета. Графи

Изготвил : Велико Динков Дончев , 2012

Анотация

Рекурсия. Рекурсивни алгоритми. Рекурсивни структури данни – линеен списък, бинарни и n-арни дървета. Графи

А. Дефиниция на рекурсия.

Пряка и косвена рекурсия – примери. Класически пример за пресмятане на функцията *n!*.

Правила при програмиране на рекурсивни алгоритми.

Примитивно рекурсивни функции (схеми), които зависят от една, две и повече предходни свои стойности. Модели на реализация.

Рекурсивна реализация на функция за намиране на *n*-тото от числата на Фибоначи. Ефективност и неефективност на рекурсията. Линеализация на билинейна рекурсия.

Функция на Акерман.

Задача за Ханойските кули. Намиране на път в квадратна мрежа/лабиринт.

Б. Рекурсивни алгоритми.

Бинарно търсене на елемент в подреден масив/линейна структура.

Метод за бърза сортировка.

В. Рекурсивни структури данни.

1. Линеен списък.

Линеен едносвързан списък – декларация, основни операции – вмъкване, премахване, модификация на елемент в списък. Рекурсивна реализация на операциите. Обхождане на елементите на списък итератор.

Цикличен линеен едносвързан списък.

Линеен двусвързан списък.

Цикличен линеен двусвързан списък.

Търсене на елемент в списък.

Сложност на операциите.

Нареден списък.

2. Дървовидни структури

Бинарно дърво – основни понятия. Бинарно дърво за търсене – основни операции: вмъкване и премахване на елемент в дървото. Сложност на операциите.

Обхождане на елементите в дърво – пост-, пре– и ин– обхождане. Дървовидна сортировка.

Балансирани двоични дървета.

2 – 3 дърво – 2 полета за данни и 3 поддървета.

n-арно дърво. Търсене на елемент в *n*-арно дърво.

3. Графи

Основни понятия – ориентиран, неориентиран граф.

Представяне – матрица на съседство, чрез линейни свързани списъци на съседните върхове, директно представяне.

Основни алгоритми – път в граф между 2 върха, задача за търговския пътник, търсене в дълбочина и широчина.

Примерни задачи. Дадена е квадратна мрежа *n* x *n*, част от квадратчетата на която са проходими, а друга част – непроходими. От едно квадратче може да се премине в съседно през общата им стена. От непроходимо квадратче не може да се премине в съседно. Дадено е едно начално и едно крайно квадратче. Да се напише програма, която определя дължината на минималния път между началното и крайното квадратче (ако има такъв).

Автор : Велико Динков Дончев , 2012

A

Навсякъде в темата примерния код ще е на езика Java, вместо във формата на псевдо-код.

Дефиниция. Казваме, че един обект е рекурсивен, ако се съдържа в себе си, или е дефиниран чрез себе си. Рекусията е математическо понятие за функция/структура(множество), която/което се дефинира чрез себе си – пряко или косвено. В програмирането, рекурсията е програмна техника, при която даден метод извиква сам себе си. Един метод е пряко рекурсивен, ако извиква директно себе си и непряко рекурсивен, ако извиква себе си косвено, чрез извикване на друг/други методи, които в даден момент отново ще го извикат. Разбира се, за да е дефинирана коректно една рекурсивна функция, е нужно да има някаква основа, т.е. да съществув някакъв аргумент от който стойността на функцията е ясна. Множеството от фиксираните аргументите и техните стойности в рекурсивна функция се нарича *база* на рекурсията.

Пример 1: Функцията n! . Математическата дефиниция на n!=1.2.3…n е следната.

$$f(n)=\begin{cases} nf(n-1), n\geq 1 \\ 1, n=0 \end{cases}$$

Това е пример за пряка рекусия. Примерна реализация на Java:

```
public int fact(int n)
{
if (n==0) return 1;
else return n*fact(n-1);
}
```

Пример 2. (непряка рекурсия)

$$A(n)=\begin{cases} 1, n=1 \\ 2*B(n-1), n>1 \end{cases}$$

$$B(n)=\begin{cases} 1, n=1 \\ 2+A(n-1), n>1 \end{cases}$$

Основното правили при създаването на рекурсивни алгоритми е да избегнем възможни „ зацикляния“, т.е. безкрайно обръщане на функция към себе си, което изчерпва крайните ресурси на компютъра. Това правило се спазва като се внимава каква е базата на рекурсията .

Примери за примитивно рекурсивни схеми, които зависят от повече от една предходни свои стойности са схеми на реализация на рекурентни редици от висок ред. Пример за такава редица е реедицата на Фибоначи: тя зависи от две предходни свои състояния. Математически, редицата на Фибоначи се дефинира по следния начин:

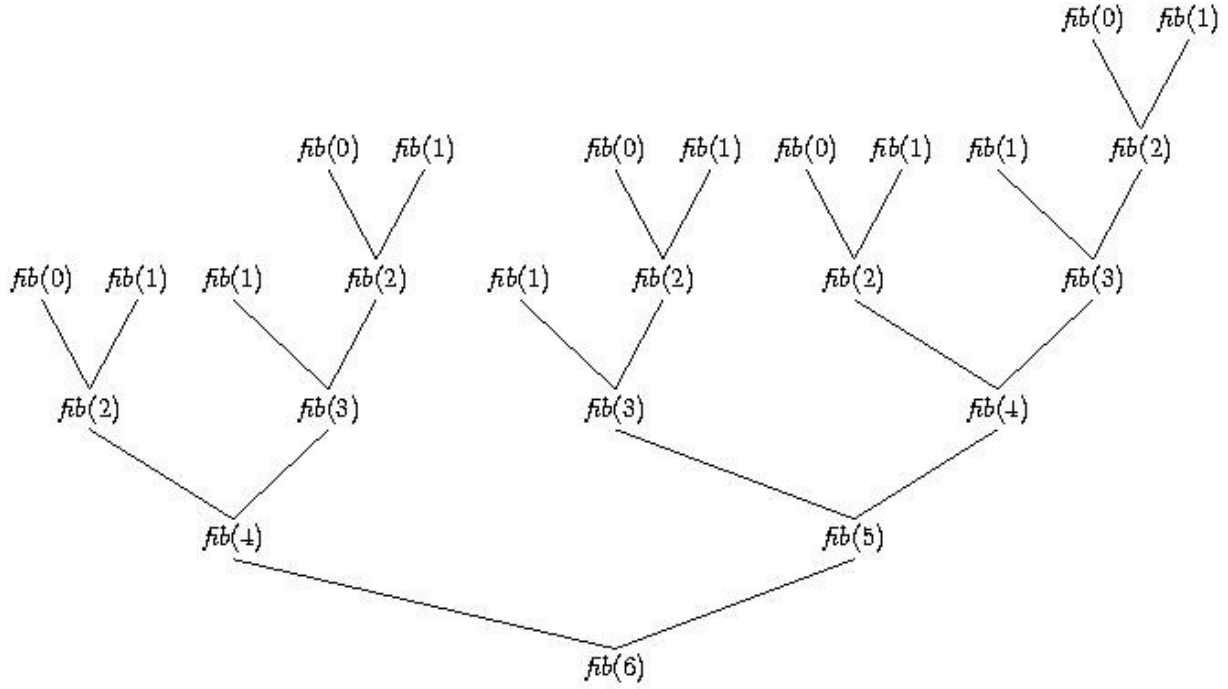
Ф(0)=Ф(1)=1

n>1: Ф(n)=Ф(n-1)+Ф(n-2)

Директната програмна реализация е следната:

```
public int fib(int n)
{
if(n==0 || n==1) return 1;
else return fib(n-1)+fib(n-2);
}
```

Този алгоритъм се оказва доста неефективен ако компилаторът не е опимизиран. Нека си представим пресмятането на редицата на фибоначи дървовидно(фиг. 1)



Това, което се забелязва е, че компютъра прави прекалено много излишни пресмятания: В следващата таблица са дадено броевете пъти за пресмятане на fib(n), n=1,..,6

Fib(n)	6	5	4	3	2	1	0
Брой пресмятания	1	1	2	3	5	8	5

Този брой расте експоненциално (основа – златното сечение фи)

Линеализацията на билинейната рекурсия за задачата на Фибоначи се състои в това да създадем такава програма, която да имитира пряка линейна рекурсия, каквато е тази за пресмятане на n! Линеаризираща програма (Наков):

```
public int fibLin(int n)
int fn=1,fn_1=1,fn_2=2;
while(n>0) {
fn_2=fn_1;
fn_1=fn;
fn=fn_1+fn_2;
n--;
}
return fn_1;
```

Това, което функцията прави е да смята от 1 към n, а не обратното.

Функцията на Акерман е исторически първата функция, която не е примитивно(обяснението на понятието примитивна рекурсивност е твърде сложно и свързано с други дефиниции) рекурсивна.

$$A(m,n)=\begin{cases} n+1, m=0 \\ A(m-1,1), m>0 \\ A(m-1, A(m,n-1)), m>0 \wedge ¿ \wedge, n>0 \end{cases}$$

Задачата за Ханойските кули е математическа игра, която представлява осем диска, различни по размер един от друг, и три стълба. В началото дисковете са подредени на левия стълб, като най-големият е най-отдолу, а най-малкият - отгоре. Целта е кулата да бъде преместена на десния стълб. Може да се мести само по един диск на ход и не може по-голям диск да бъде поставен върху по-малък. Всеки ход е съставен от взимането на горния диск от даден стълб и поставянето му най-отгоре на друг стълб.Ще обобщим решението, което търсим за n диска. Нека Tn бъде броят ходове, нужни за
решаване на задачата. Един от начините за решаване на задачата с n диска е:

- преместваме n-1 диска от левия към средния стълб: Tn-1 хода
- преместваме най-големия диск от левия на десния стълб: 1 ход
- преместваме дисковете от средния стълб на десния: Tn-1 хода

Следователно Tn = 2Tn-1+1, ∀n>1

Намирането на път в квадратна мрежа/лабиринт също е задача, която може да бъде решена рекурсивно. Тръгваме от началното квадратче и проверяваме всички съседни дали са достъпни. За всяко, което е, търсим път от него до крайното по същия начин. Алгоритъмът приключва, когато достигнем крайното квадратче.

Б.

Два от най- важните рекурсивни алгоритми са бинарното търсене (binary search) и метод на бързата сортировка.

Бинарно търсене.Обикновените методи за търсене в неподредена линейна стуктура са със сложност O(n).Нека дадена линейна структура е сортирана (за определеност във възходящ ред).Метода на бинарното търсене постига сложност log(n). Можем да си мислим, че структурата ни съдържа числа, но може да е всеки абстрактен тип, който е comparable(елементите му се сравняват). Да кажем че търсим числото a.

0.

0.1 Ако списъка, с който работим е празен – край, елементът го няма в списъка.

0.2 иначе, към т. 1.

1. Сравняваме a с елемента в средата на списъка, b.

1.1 ако a=b, край- a е намерен.

1.2 Иначе ако b>a – към стъпка 0 със списък с всички елементи от текущия, които са преди b.

1.3 Иначе – към стъпка 0 със списък с всички елементи от текущия, които са след b.

Метод за бърза сортировка:

1. Избира се “главен” елемент от списъка с елементи, които ще бъдат сортирани
2. Списъкът се пренарежда така, че всички елементи, които са по-малки от “главния” се поставят вляво от него, а всички, които са по-големи – вдясно от него
3. Рекурсивно се повтарят горните стъпки върху списъка с по-малките и списъка с по-големите елементи
4. Получените списъци се сливат и се получава сортираният списък.

B.

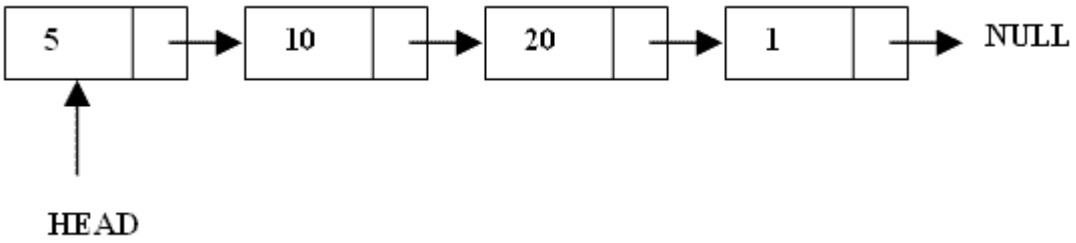
Рекурсивни стуктури от данни са такива, които могат да се дефинират рекурсивно. Линейни структури данни – такива структури са всички , чиито елементи са част от списък или последователност на данни.

1. Линеен списък

Общото на всички структури - линейни списъци е, че независимо от имплементацията си те реализират следната логика:

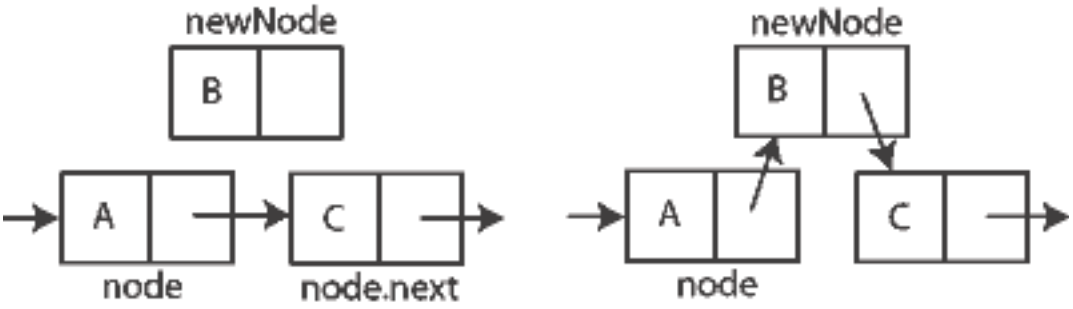
- Има пръв елемент
- Всеки елемент (освен евентуално един- последния) осигурява преместване към следващия (чрез „указател“)

Линейния едносвързан списък е линеен списък, удовлетворяващ само гореизброените характеристики. В ООП се реализира като клас, който притежава един фиксиран елемент(HEAD,може да се съдържа и последния) от тип Node и евентуално цяло число, обозначаващо текущия брой на елементи в списъка. Класът node, пък, се състои от две полета, едното от което е най- общо от тип Object (няма да се впускаме в подробности) и съдържа информация, а другото е указател (друг Node) .



Добавянето на елемент в списъка става като в последния node (който сочи към NULL) указателя се променя да сочи към node, който е създаден съдържащ необходимата нова информация и сочещ към NULL. Ако в класа пазим указател към последния Node, операцията е със сложност O(1), ако не- O(n). Вмъкването на елемент след даден елемент става чрез „разкъсване“ Опрацията вмъкване обикновено има аргумент, който посочва на кое място да се вмъкне. Ако трябва да вмъкнем на последно място, то операцията е еквивалентна на добавяне. Ако трябва да добавим на първо място, то новия node насочваме към HEAD и го правим HEAD. В останалите случаи, когато вмъкването е

междинно, то става както на следващата фигура:



Сложността на вмъкването е O(n). Модификация на елемент става по следния начин: чрез обхождане намираме елемента, който трябва да се модифицира и заменяме информационната клетка. Сложност O(n).

Забележка: Темата е Недовършена!!! Има я развита от Мира

Литература:
[1] Записки по лекциите по СДПООП, спец. ПМ, С и КН, Д. Биров
[2] Програмране=++Алгоритми, Наков
[3] Теория на програмирането, Соскова
[4] wiki

Темата е разработена от Велико Дончев, уч. 2011/2012 г.