Рекурсия

Рекурсивные алгоритмы.

Рекурсивные типы данных.

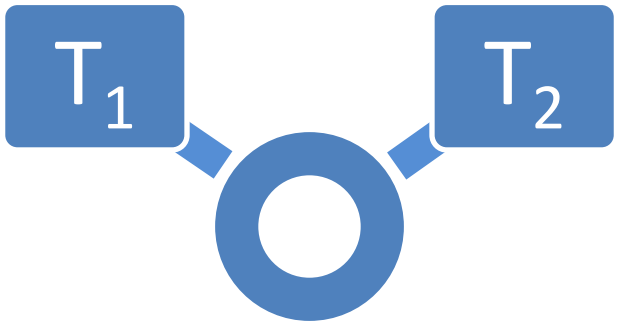
РЕКУРСИВНЫЕ АЛГОРИТМЫ.

Определение и примеры

- Объект называется *рекурсивным*, если он содержит сам себя или определен с помощью самого себя.
- <Имя>::=<Буква>|<Имя><Буква>|<Имя><Цифра>
- Дерево это

a) $\varnothing$ – пустое дерево

b) Если T_1 и T_2 – деревья, то тоже есть дерево.



- Факториал:

$$n! = \begin{cases} 1, & \text{если } n = 0 \\ n(n-1)!, & \text{если } n > 0 \end{cases}$$

Формальное представление

- Пусть:

P – рекурсивная процедура;

S_i – базовые операторы, не содержащие P;

$\mathcal{P}$ – композиция операторов;

- Тогда:

$$P = \mathcal{P}[S_i, P]$$

- Средством создания рекурсивных программ является описание функций (процедур, под-программ), т.к. оно позволяет присвоить оператору имя, с помощью которого он может быть вызван.

Классификация

Прямо рекурсивная

- $P = \mathcal{P}[S_i, P]$

Косвенно рекурсивная

- $P = \mathcal{P}[S_i, Q]$

  $Q = \mathcal{Q}[S_i, P]$

Набор локальных объектов, определенных внутри рекурсивной функции, создается заново при каждом вызове такой функции.

Идентификаторы всегда ссылаются на множество переменных, созданное последним.

N.B.

- Проблема окончания работы.

  $$P = \mathcal{P}[S_i, \underline{\textbf{if}}\ !B\ \underline{\textbf{then}}\ P]$$

  Наиболее надежный способ окончания работы – связать с P параметр *n* и рекурсивно вызывать P с параметром *n-1*:

  $$P(n) = \mathcal{P}[S_i, \underline{\textbf{if}}\ n>0\ \underline{\textbf{then}}\ P(n-1)]$$

- Глубина рекурсии.

  Следует убедиться, что она не только конечна, но и достаточно мала.

Поиск пути в лабиринте

Пример рекурсивной программы

Вид файла данных

```
1.  20 24
2.  ....................#
3.  ................#...#.
4.  ...............#...#..
5.  ..............#...#...
6.  .............#...#....
7.  ............#...#.....
8.  ...........#...#......
9.  ..........#...#.......
10. .........#...#........
11. ........#...#.........
12. .......#...#..........
13. ......#...#...........
14. .....#...#............
15. ....#...#.............
16. ...#...#..............
17. ..#...#...............
18. .#...#................
19. #...#.................
20. ...#.................E
21. #.....................
```

Первая строка:

N – количество сток

M – количество символов в строке

Вторая строка и далее:

N строк, со словами из алфавита {#,.,E}

. – проходимая клетка

- непроходимая клетка

E – конечная клетка

Файлы заголовков и глобальные переменные

```
#include <iostream.h>
#include <stdio.h>
int N,M,F=0,X,Y;
char** A;
int** B;
```

Создание динамических массивов

```
void array(int n, int m) {
   A = new char*[n];    // план лабиринта
   B = new int*[n];      // ранг полей лабиринта
   while (n--) { A[n] = new char[m];
                 B[n] = new int [m]; }
}
```

Рекурсивное построение пути

```
void next(int n,int m,int l) {
   if( n<0 || m<0 || n>=N || m>=M) return;
   if( A[n][m]=='.' && (!B[n][m] || l<B[n][m]) ) {
        B[n][m]=l; next(n-1,m,l+1); next(n,m+1,l+1);
        next(n+1,m,l+1); next(n,m-1,l+1); }
   else if( A[n][m]=='E' && (!B[n][m] || l<B[n][m]) ) {
        B[n][m]=l; F++; X=n; Y=m; }
   return;
}
```

Основная программа

```
void main(int argv, char** argn) {
FILE *pole;
int n=0,m;

if ((pole = fopen(argn[1], "r")) == NULL) {
   fprintf(stderr, "Cannot open input file .\n"); return; }

fscanf(pole,"%d %d",&N,&M);
array(N,M);
while (!feof(pole)) fscanf(pole,"%s",A[n++]);
fclose(pole);
if(n!=N) { fprintf(stderr,"Error in data file\n"); return; }
```

Основная программа

```
for(n=0;n<N;n++) for(m=0;m<M;m++) B[n][m]=0;
next(0,0,1);
if(!F) cout << "No way\n ";
    else print_way(X,Y,B[X][Y]);
for(n=0;n<N;n++) cout << A[n] << endl;
}
```

Демонстрация

Печать найденного пути

```
void print_way(int n,int m,int l) {
   cout << n << ' ' << m << endl; A[n][m]='*';
   if(!--l) return;
   if(n && B[n-1][m]==l) n--;
   else if(n+1<N && B[n+1][m]==l) n++;
         else if(m && B[n][m-1]==l) m--;
               else m++;
   print_way(n,m,l);
}
```

Разбиение числа m на слагаемые

6

5 + 1

4 + 2 4 + 1 + 1

3 + 3 3 + 2 + 1 3 + 1 + 1 + 1

2 + 2 + 2 2 + 2 + 1 + 1 2 + 1 + 1 + 1 + 1

1 + 1 + 1 + 1 + 1 + 1

P(6) = 11

Разбиение m на слагаемые $\leq$ n

- Q(m,1) = 1 $\quad 1 + 1 + \ldots + 1 + 1$
- Q(1,n) = 1 $\quad 1 - \forall n$
- Q(m,n) = Q(m,m) $\quad \forall n \geq m$
- Q(m,m) = 1 + Q(m,m-1)
- Q(m,n) = Q(m-n,n) + Q(m,n-1) $\quad \forall n < m$

Разложения содержащие слагаемое равное n

Разложения не содержащие слагаемое равное n

```
#include <iostream>
using namespace std;

int Q(int m, int n) {
    if( m==1 || n==1 ) return 1;
    if (n>=m) return Q(m,m-1)+1;
    return Q(m-n,n)+Q(m,n-1); }
int main() {
    int N;
    cin >> N;
    cout << Q(N,N) << endl;
    return 0; }
```

ЧИСЛА ФИБОНАЧЧИ

Или о том, что рекурсию следует использовать весьма осмотрительно...

Вычисление чисел Фибоначчи

- Итеративный

```
long F(int n) { long a=1,b=1;
do{a+=b;b=a-b;} while(--n>2);
return a; }
```

- Рекурсивный

```
long F(int n) {
return n<3?1:F(n-1)+F(n-2); }
```

```
void main() { int n;
  cout << "Input n="; cin >> n;
  start = clock();
  cout << " F(" << n << ")= " << F(n) << '\t';
  end = clock();
  cout << “Time was: " << (end - start) / CLK_TCK << " s\n"; }
```

F(40)= 102334155 Time was: 0 s
F(45)= 1134903170 The time was: 0 s

F(40)= 102334155 Time was: 5.10989 s
F(45)= 1134903170 Time was: 55.714286 s

ФУНКЦИЯ АККЕРМАНА

Или о том, когда рекурсию не следует использовать вообще…

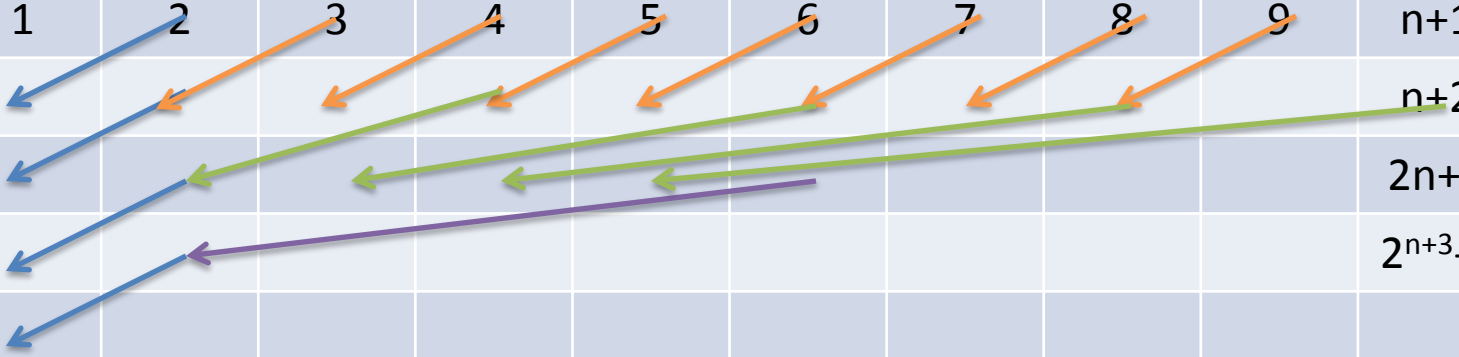
Функция Аккермана

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

```
#include <iostream.h>
int A(int m,int n) { return !m?n+1:
                          !n?A(m-1,1): A(m-1,A(m,n-1)); }
void main() { int n,m;
cout << "Input m="; cin >> m; cout << "Input n="; cin >> n;
cout << "A(" << m << ',' << n << «)= " << A(m,n) << endl; }
```

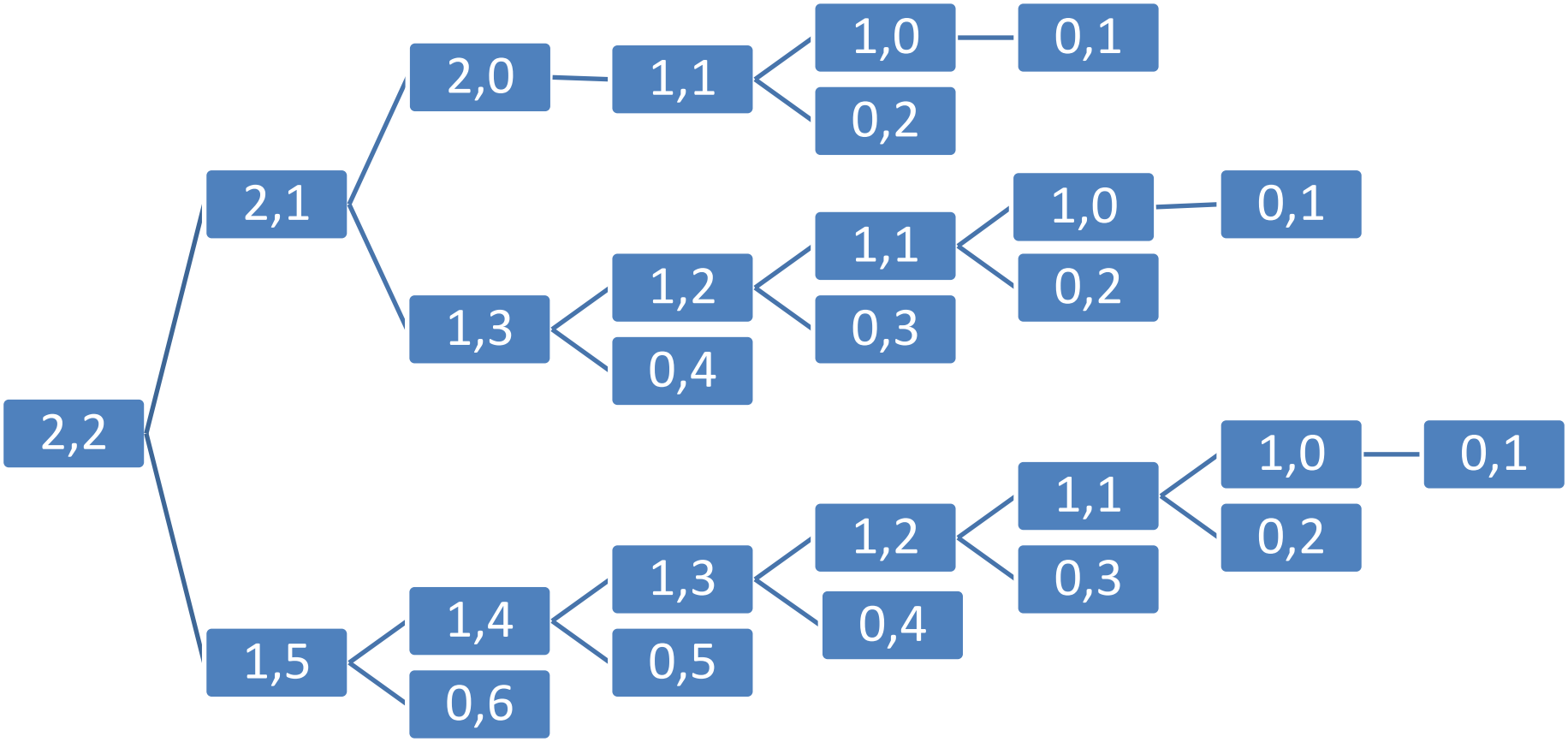
A(2,2) = A(1, A(2,1))

m\n	0	1	2	3	4	5	6	7	8	…
0	1	2	3	4	5	6	7	8	9	n+1
1										n+2
2										2n+3
3										2^{n+3}-3
4										
…										



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

Дерево вычисления функции A(2,2)



A(2,2)=7 Количество вершин дерева - 27

Демонстрация

Биномиальные коэффициенты

$$C_n^k = \frac{n!}{k!\,(n-k)!}$$

$$C_{n+1}^k = C_n^k + C_n^{k-1}$$

1. На основе определения

```
int fact(int n) {
   switch (n) {     case 0:
                    case 1: return 1;
                    default: return n*fact(n-1); }    }
                            …
for(n=0; n<=N; n++) for(k=0; k<=n; k++)
   cout << fact(n)/fact(k)/fact(n-k) << (n==k?'\n':' ');
```

2. На основе рекуррентного соотношения

```
int Cnk(int n,int k) {
    if(!k) return 1;
    if(!n) return 1;
    if(n==k) return 1;
    return Cnk(n-1,k)+Cnk(n-1,k-1);
}
```

3. Треугольник Паскаля

```
void C(int N) {
int *a;
int i,n;
    a = new int[++N];
    a[0]=1; if(N>1) a[1]=1;
    for(n=3; n<=N; n++) {
        for(i=n-2,a[n-1]=1; i>0; i--) a[i]+=a[i-1];
        for(i=0;i<n;i++) cout << a[i] << (i==n-1?'\n':' ');
    }
    delete a;
}
```

ДИНАМИЧЕСКИЕ ИНФОРМАЦИОННЫЕ СТРУКТУРЫ

Линейные наборы данных

Древовидные структуры

Указатели

- Динамическое распределение памяти – выделение памяти для отдельных компонент в тот момент, во время выполнения программы, когда они появляются.
- Компилятор во время трансляции выделяет фиксированный объём памяти для хранения адреса (указатель) динамически размещаемой компоненты.

ЛИНЕЙНЫЕ НАБОРЫ ДАННЫХ

- Каждый элемент содержит только одну ссылку, связывающую его со следующим элементом списка:

template <class T> struct Item { T n; Item* next; };

- Указатель на список:

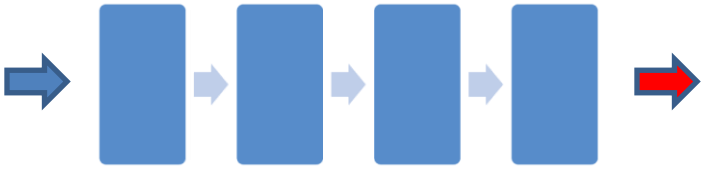
Item<Myclass> *top;

- <ЛНД> ::= Ø | Item | Item ¤ <ЛНД>

СПИСОК, ОЧЕРЕДЬ, СТЕК

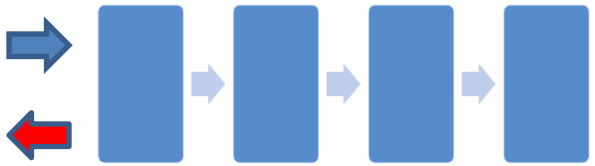
- Очередь

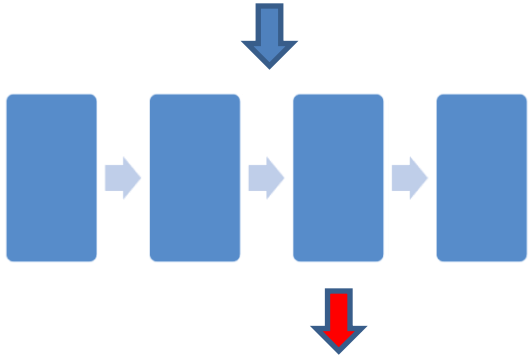
  FIFO – First In First Out



- Стек

  LIFO – Last In First Out



- Список

  Casual In Casual Out