Динамическое программирование

Алгоритмы
и алгоритмические языки

goo.gl/c8pyqx

Лекция 9, 07 ноября, 2017

Лектор:

Дмитрий Северов, кафедра информатики 608 КПМ

dseverov@mail.mipt.ru

http://cs.mipt.ru/wp/?page_id=6077

Метод динамического «программирования»

1. **Ричард Беллман 1953 год**
2. **Пользуется свойствами задачи**
    1. Оптимальная подструктура
    2. Аддитивно перекрывающиеся подзадачи
    3. Возможность запоминания решений подзадач
3. **Эффективнее рекурсии и/или перебора**
4. **Примеры задач**
    1. Числа Фибоначчи
    2. О наибольшей подпоследовательности
    3. О редакционном расстоянии
    4. О порядке перемножения матриц
    5. О ранце

Два подхода динамического программирования

Нисходящее ДП

1. Задача разбивается на нужные подзадачи
2. Нужные подзадачи решаются
3. Решения подзадач комбинируются в решение задачи

Восходящее ДП

1. Решаются «все полезные» подзадачи.
2. Результаты подзадач накапливаются и сохраняются.
3. Решения подзадач комбинируются в решение задачи

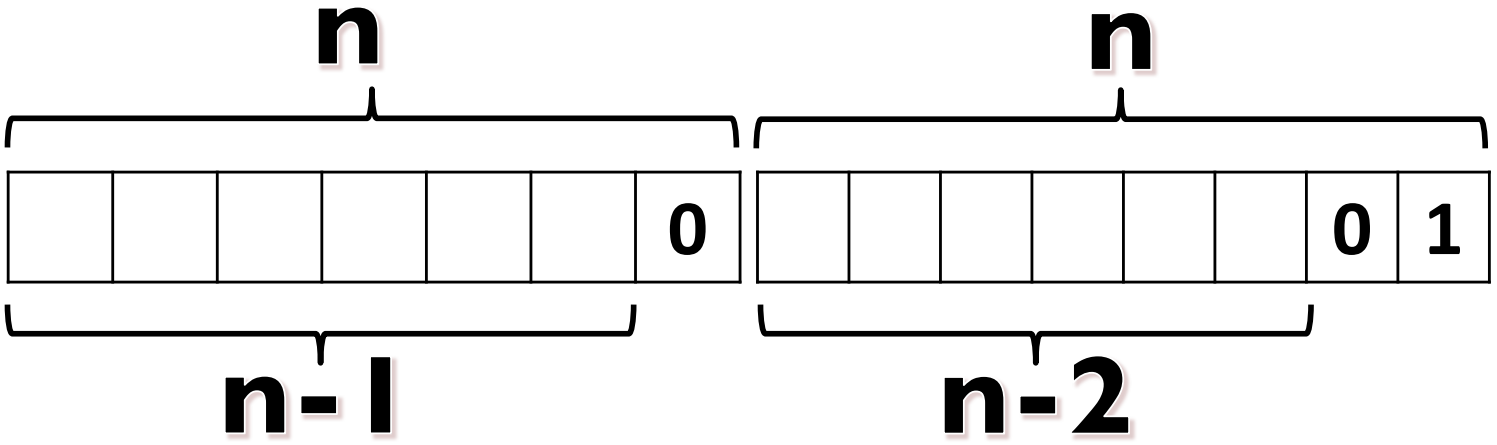
Классический пример: поиск n-ого числа Фибоначчи

```
int Fibo(int n) {
  if(n<2)
     return 1;
  else
     return
      Fibo(n-1) + Fibo(n-2);
}
```

```
int fibo(int a[], int n) {
  if(n<2) return 1;
  if(a[n])
     return a[n];
  else
     return (a[n] =
        fibo(a,n-1) + fibo(a,n-2));
}
```

Сколько слов длины n в алфавите {0,1}, не содержащих две идущие подряд единицы?

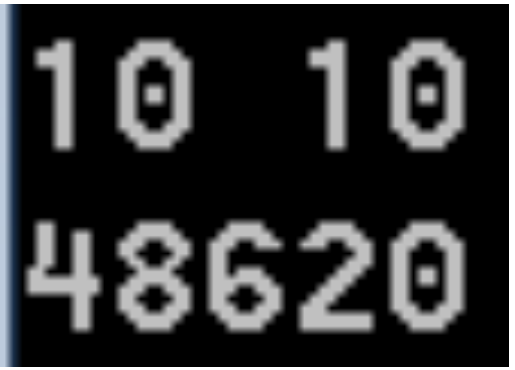
$$K(1) = 2 : 0 \text{ и } 1$$
$$K(2) = 3 : 00, 01 \text{ и } 10$$
$$K(n) = K(n-1) + K(n-2)$$



На прямоугольном поле размером n x m можно двигаться на одно поле вниз или вправо.
Сколькими способами можно попасть из левой верхней клетки в правую нижнюю?

1	1	1	1	1
1				
1				j-1
1				j
1		i-1	i	

```
#include <iostream>
using namespace std;
int main() {
int **a,n,m;
    cin >> n >> m;
    a = new int* [n];
    for(int i=0; i<n; i++) { a[i] = new int [m];
       for(int j=0; j<m; j++) a[i][j]=((!i || !j)?1:0); }
    for(int i=1; i<n; i++) for(int j=1; j<m; j++)
       a[i][j] = a[i-1][j] + a[i][j-1];
    cout << a[n-1][m-1] << endl;
    return 0;}
```



Задача про последовательности

Из последовательности целых чисел выбрать самую длинную строго возрастающую подпоследовательность.

Задача про последовательности

1. Массив A[N] для чисел последовательности
2. Массив L ={l_k} длин строго возрастающих подпоследовательностей, заканчивающих элементом с номером k ≤ N. Наибольший элемент массива L будет длиной искомой подпоследовательности
3. Для предъявления искомой подпоследовательности, ещё один массив B={b_k}, номеров предпоследнего элемента подпоследовательности, длина которой записана в l_k.
4. l_k = max (l_j) + 1 для j от 1 до k-1 таких, что $a_j < a_k$.

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

int *A,*B,*L,N;

void arrays(istream& in,int n=0) { int a;
    if(in >> a) arrays(in,n+1);
    else {  A = new int[N=n]; B = new int[N];
            L = new int[N]; return; }
    A[n] = a;
}

void LEN(int k) { int r=0;
    for(int i=0; i<k; i++)
        if(A[i] < A[k] && r<L[i]) r = L[B[k]=i];
    L[k] = r+1;
}

int main() { int r=0,n = 0;
    arrays(cin);
    for(int i=0;i<N;i++)
        cerr << A[i] << ' '; cerr << "N= " << N << endl;
    B[0] = -1; L[0] = 1;
    for(int i=1;i<N;i++) LEN(i);
    for(int i=0; i<N; i++) if(r < L[i]) r = L[n=i];
    cout << r << endl;
    while(n>=0) { cout << A[n] << ' '; n = B[n]; }
    delete A; delete B; delete L;

    return 0;
}
```

```
2 8 5 9 12 6 ^Z
2 8 5 9 12 6 N= 6
4
12 9 8 2
```

Нахождение палиндрома

Найти длину P наибольшего палиндрома, который может быть составлен из букв заданной строки S.

Нахождение длины палиндрома P(S)

1. **Любая строка S, состоящая из одного символа, - это палиндром длины 1.**
2. **Строка S из двух символов,**
   1. если символы совпадают – это палиндром длины 2,
   2. если различаются – удаляем любой символ и P(S) = 1.
3. **Для строки $S = s_1 \ldots s_L$ длины L:**
   1. если начальный и конечный символы совпадают, то
      $$P(S) = P(s_2 \ldots s_{L-1}) + 2;$$
   2. если различаются, то:
      $$P(S) = \max(P(s_2 \ldots s_L), P(s_1 \ldots s_{L-1}))$$
      длина максимального палиндрома,
      1. либо из строки без первого символа
      2. либо из строки без последнего символа

```cpp
#include <iostream>
using namespace std;
#include "My_str.h"

int Lpal(My_string a) {
int l=a.size();
    if(l<=1) return l;
    if(a[0]==a[l-1])
        return      Lpal(My_string(&a[1],&a[l-1]))+2;
    else return max( Lpal(My_string(&a[1])),
                     Lpal(My_string(&a[0],&a[l-1])) );
}

int main() {
My_string a;
    cin >> a; cerr << a << ' ' << a.size() << endl;
    cout << Lpal(a) << endl;

    return 0;
}
```

```
А роза упала на лапу Азора.
А роза упала на лапу Азора. 27
19
```

```
арозаупалааналапуазора
арозаупалааналапуазора 21
21
```

Задача о хрустальном шаре.

Пытаемся найти условия разбивания хрустального шара, сброшенного с высоты.

Имеем m одинаковых хрустальных шаров

За какое минимальное число попыток сможем найти этаж N-этажного здания, начиная с которого этажа шары будут разбиваться ?

Задача о хрустальном шаре.

- m =1
- Неизбежно бросание с каждого следующего вверх этажа
- Ответ: N попыток

- m = 2
- Кажется уместно деление пополам. Сбросим первый шар с $\lfloor N/2 \rfloor$ этажа.

1. Если первый шар разбит, то хватит $\lfloor N/2 \rfloor$ попыток ниже.
2. Если первый шар цел, то хватит $\lfloor N/2 \rfloor$ попыток выше.

- Ответ: $\lfloor N/2 \rfloor$?

Задача о хрустальном шаре (подход на основе ДП)

Пусть, для первой попытки оптимален этаж T.
Время эксперимента F(m,N) зависит от результата

Первый шар разбит:
$F(m-1, T-1) + 1$

Первый шар цел:
$F(m, N-T) + 1$

На этаже T:

$$F(m,N) = \max(F(m-1, T-1),F(m, N-T)) + 1$$

В здании:

$$F(m, N) = \min_T (1 + \max(F(m-1, T-1),F(m, N-T)));$$

$$F(1,N)=N; \quad F(m,0)=0; \quad F(m,1)=1; \quad F(m,2)=2$$

Нисходящее ДП

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

int start,end;

int f(int m, int N) {
int p,r = N;
    if(m==1 || N<3)  return N;
    for(int t=2; t<N; t++) {
        p = 1 + max(f(m-1,t-1),f(m,N-t));
        if(p<r) r=p; }
    return r;
}

int main() {
int m,N;
    while(cin >> m >> N) {
        start = clock(); cout << f(m,N) << '\t';
        end = clock();  cout << end-start << endl; }
    return 0;
}
```

```
1 20
20          0
2 20
6           0
5 40
```

Восходящее ДП

```
#include <iostream>
#include <ctime>
using namespace std;
int start,end;

int f(int m, int N) {                       //      C99 !
int r,p;
int a[m][N+1];
    for(int n=0;n<=N;n++) a[0][n] = n;  // m=1
    for(int M=1;M<m;M++)                    // m>1
        for(int n=0;n<=N;n++)
            if(n<3) a[M][n] = n;            //      N=1,2
            else { r = N;                   //      N>2
                   for(int t=2; t<=n; t++) {
                        p = 1 + max(a[M-1][t-1],a[M][n-t]);
                        if(p<r) r=p; }
                   a[M][n] = r; }
    return a[m-1][N];
}

int main() {
int m,N;
    while(cin >> m >> N) {
        start = clock();
        cout << f(m,N) << '\t';
        end = cl  1/ 2  int f (int m, int N)
        cout << end-start << endl; }
    return 0;
}
```

```
1  20
20          0
2  20
6           0
5  40
6           0
10  1000
10          31
```

Задача о минимальной сдаче

Имеется неограниченное количество монет достоинств $V=\{v_1,\ldots,v_K\}$. $\forall i: v_i<v_{i+1}$

Какое минимальное количество монет D(S) потребуется чтобы набрать заданную сумму S ?

$$D(S) = \min_k(D(S - V_k))$$

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

int change(int S, int V[], int K) {
int r, *d = new int[S+1];
    d[0]=0; for(int i=1; i<=S; i++) d[i]=S+1;
    for(int s=1; s<=S; s++)
        for(int k=0; k<K; k++)
            if(V[k]<=s && (r = d[s-V[k]] + 1) < d[s])
                d[s] = r;
    r = d[S]; delete d;
    return r;
}

int main() {
int K,S,*V;
    cin >> K; V = new int[K];
    for(int i=0;i<K;i++) cin >> V[i];
    cin >> S;
    cout << change(S,V,K);
    delete V;
    return 0;
}
```

```
3
1 5 7
24
4
```

Задача о вариантах сдачи

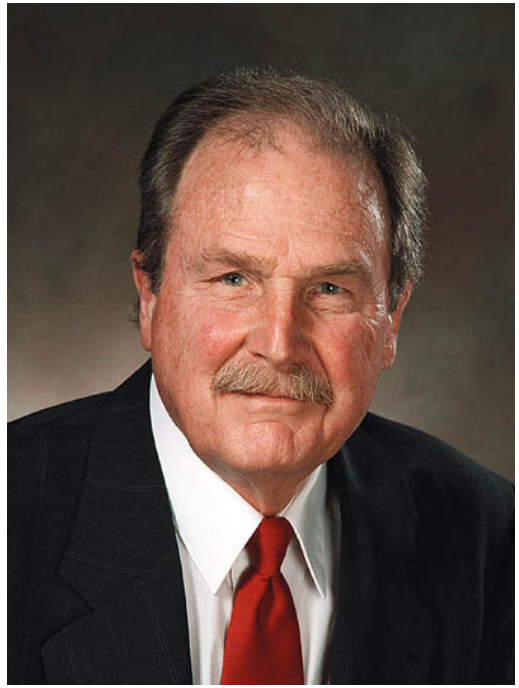
Имеется неограниченное количество монет достоинств $V=\{v_1,\ldots,v_K\}$. $\forall i: v_i<v_{i+1}$

Сколькими способами можно набрать заданную сумму?

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

long int N=0;

void exchange(int sum, int k,int v[],int count[]) {
    if(!sum) N++;
    else {
        if( sum >= v[k] ) { count[k]++;
            exchange(sum - v[k],k,v,count); count[k]--; }
        if(k) exchange(sum,k-1,v,count); }
}

int main() {
int *V, *count, K, S;
    cin >> K;
    V = new int[K]; count = new int[K];
    for(int i=0; i<K; i++) { cin >> V[i]; count[i]=0; }
    cin >> S;
    exchange(S,K-1,V,count);
    cout << N << endl;
    delete V; delete count;
    return 0;
}
```

```
5
1 5 10 25 50
100
292
```

Код Хаффмана



Используется для сжатия данных путем кодирования более частых символов более короткими последовательностями битов.

ПРОГРАММИРОВАТЬ - ЗНАЧИТ ПОНИМАТЬ.

```
^Z
А 4
В 1
Г 1
З 1
И 3
М 3
Н 2
О 3
П 2
Р 3
Т 3
Ч 1
Ь 2
- 1
. 1
```

```cpp
#include <iostream>
using namespace std;
struct  item { char c; int n; item* next;
    item(char C, int N, item* Next=NULL) {
        c=C; n=N; next=Next; }
};
int main() {
    item *p, *T = NULL; char c;
    while( cin >> c )
        if(T) { if(c < T->c ) T = new item(c,1,T);
            else { p=T; for(;;) {
                if( c == p->c) { p->n++; break; }
                if( c < p->c ) {
                    p->next = new item(p->c,p->n,p->next);
                    p->c=c; p->n=1; break; }
                if( p->next ) p = p->next;
                else { p->next = new item(c,1); break; }
                }
            }
        }
        else T = new item(c,1);
    p=T;
    do cout << p->c << ' ' << p->n << endl;
    while(p = p->next);
    return 0;}
```

Код Хаффмана

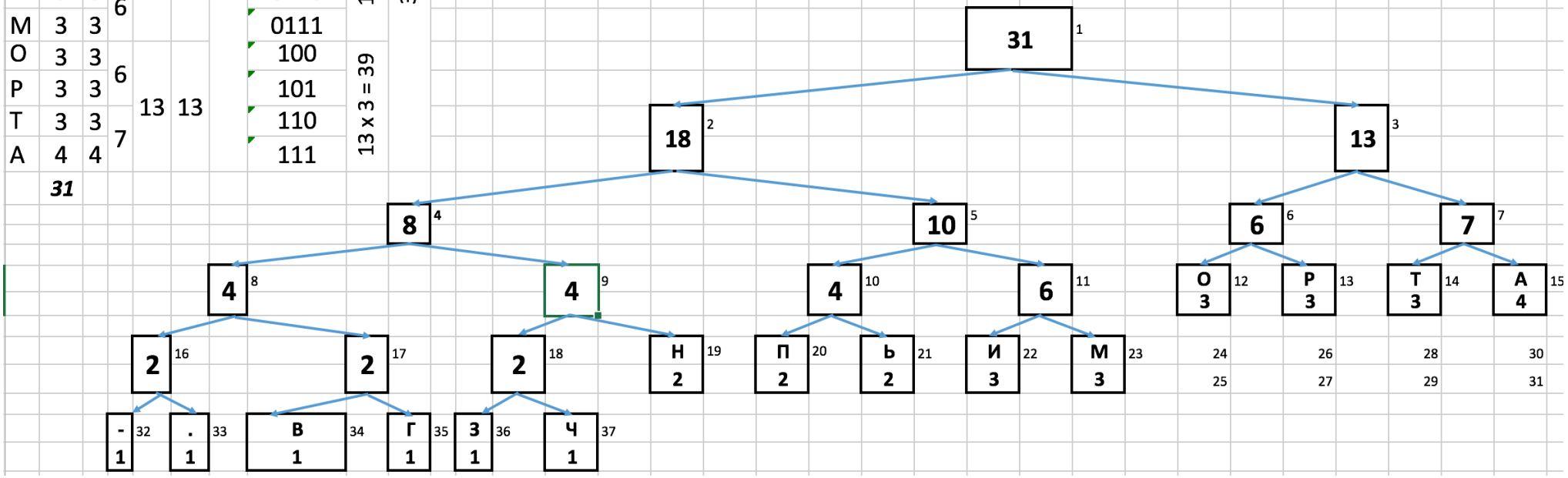
- Более частые символы последовательности должны иметь более короткий код, а более редкие – более длинный
- Чтобы исключить проблема декодирования символов, кодированых двоичными кодами разной длины, есть префиксное кодирование:

Код никакого символа не является началом кода другого.

- Кодирование – бинарное дерево, где листья содержат символы, а набор ребер (левое – 0, правое – 1), входящих в путь от корня дерева к листу, порождает код символа.

-	1	2	4	8	18	31	00000	6 x 5 = 30
.	1						00001	
В	1	2					00010	
Г	1						00011	
З	1	2	4				00100	
Ч	1						00101	
Н	2	2					0011	12 x 4 = 48
П	2	2	4	10			0100	
Ь	2	2					0101	
И	3	3	6				0110	
М	3	3					0111	
О	3	3	6	13	13		100	13 x 3 = 39
Р	3	3					101	
Т	3	3	7				110	
А	4	4					111	
	31							39 + 48 + 30 = 117

0 ⟷ 1



Кодирование

```cpp
#include <iostream>

using namespace std;
struct huff { char symbol; const char* code; } Huff[] = {
{'А',"111"},{'Т',"110"},{'Р',"101"},{'О',"100"},
{'М',"0111"},{'И',"0110"},{'Ь',"0101"},{'П',"0100"},{'Н',"0011"},
{'Ч',"00101"},{'З',"00100"},{'Г',"00011"},
{'В',"00010"},{'.',"00001"},{'-',"00000"}
};

const char *source = "ПРОГРАММИРОВАТЬ-ЗНАЧИТПОНИМАТЬ.";

const char* find(char c) {
    for(int i=0;i<15;i++) if(Huff[i].symbol==c) return Huff[i].code;
    return NULL;
}

int main() {
const char *p=source;
    while(*p) cout << find(*p++);
    cout << endl;
    return 0;
}
```

```
010010110000011101111011101110110101100000101111100101000000001000011111001010110
110010010000110110011111111100101000001
```

Декодирование

```
#include <iostream>

using namespace std;

char t[] = {
    0,0,0,0,0,0,0,0,0,0,0,0,
    'о','р','т','а',
    0,0,0,
    'н','п','ь','и','м',
    0,0,0,0,0,0,0,0,
    '-','.','в','г','з','ч',
    0};

int main() {
int i=1;
char c;
    while(cin>>c) { i*=2; if(c=='1') i++;
        if(t[i]) { cout << t[i]; i=1; } }
    cout << endl;
    return 0;}
```

```
0100101100000111011110111011101101011000001011111001010000000100001111100101011011001001000011011001111111100101000010
программировать-значитпонимать.
```