Технологии разработки алгоритмов решения инженерных задач

лекция №6

Преподаватель: Дреев Александр Николаевич

6. Рекурсивные алгоритмы.

6.1. Задача программного умножения.

6.2. Поиск наибольшего общего делителя.

6.3. Волновой алгоритм на базе рекурсивного.

6.4. Количество вариантов путей «черепахи».

рекуррентные алгоритмы

Что такое "рекурсия"

рекурсивный алгоритм - алгоритм, который упрощает задачу или разбивает ее на более простые и вызывает себя для решения этих упрощенных подзадач. Обязательно присутствие условия прекращения дальнейших вызовов.

пример: Умножения m * n. Можно записать: m * n = (m-1) * n + n.

Алгоритм "множу (m, n)": если m> 1 возвращаем "множу (m-1, n)" + n; иначе возвращаем n.

рекуррентные алгоритмы

Что такое "рекурсия"

Описание работы "множу":

"Множу (3,8)": если 3> 1

возвращаем "Множу (3-1,8)" + 8 ;

иначе возвращаем 8.

"множу (2,8)": если 2> 1

возвращаем "Множу (2-1,8)" + 8 ;

иначе возвращаем 8.

"множу (1,8)": если 1> 1

возвращаем "Множу (1-1,8)" + 8;

иначе возвращаем 8 .

жадные алгоритмы

Что такое "рекурсия"

Схема работы "множу":

Задача 3 * 8 8

\+

проще задача 2 * 8

Задача 2 * 8 8

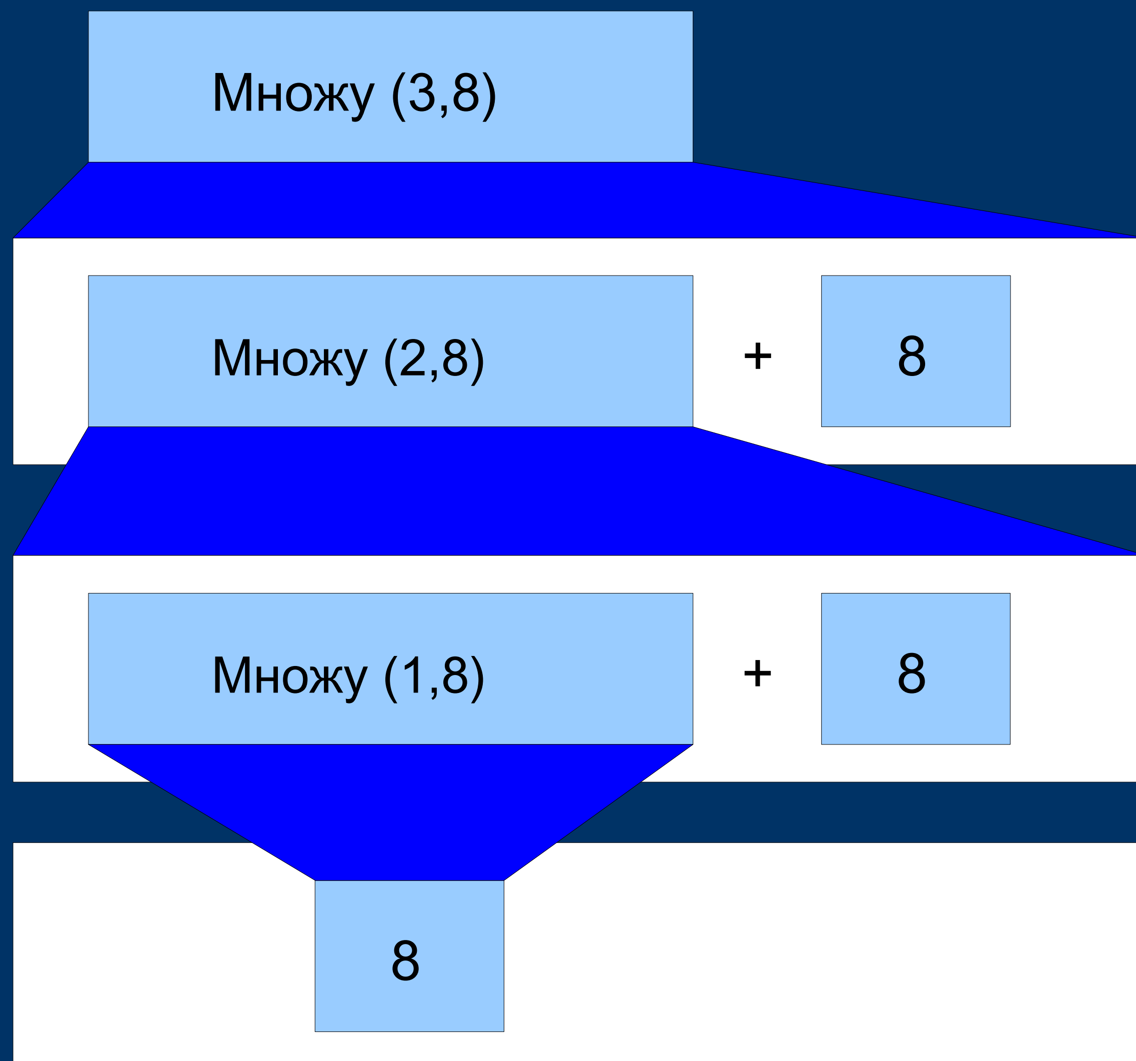
\+

проще задача 1 * 8

Задача 1 * 8 8

рекуррентные алгоритмы

Что такое "рекурсия"



рекуррентные алгоритмы

характеристика рекурсии

Преимущества рекурсии:

1. Значительное упрощение записи алгоритмов определенного класса задач.
2. Открытая логика работы.

недостатки:

1. Ограниченная глубина вызовов.
2. Расходы на вызов функции существенно замедляют работу.
3. Иногда трудно написать вариант программы без рекуррентного вызова.

рекуррентные алгоритмы

Поиск наибольшего общего делителя

Задача: найти K - совместный наибольший делитель чисел M> N.

Упрощенная задача: найти K - совместный наибольший делитель чисел (MN), N. Алгоритм: Большее число называем M, меньше N. Если M = N, тогда возвращаем K = N; иначе К = Алгоритм (MN, N).

рекуррентные алгоритмы

Поиск наибольшего общего делителя

Реализация: int NSD (int M, int

N);

int main () { int M = 345, N =

875;

int K = NSD (M, N); } int NSD

(int M, int N) { if (N == M) return

N;

if (N> M) { int T = M; M = N; N = T; }

return NSD (MN, N); }

рекуррентные алгоритмы

Поиск наибольшего общего делителя

```
Реализация: int NSD ( int M, int
N);
int main () {  int M = 345, N =
875;
    int K = NSD (M, N); } int NSD
( int M, int N) { if (N == M) return
N;
   if (N> M) { int T = M; M = N; N = T; }
   return NSD (MN, N); }
```

рекуррентные алгоритмы

Поиск наибольшего общего делителя

865	345520
	345175
	345170175170
	5

865-345

520-345

345-175

175-170

рекуррентные алгоритмы

Поиск короткого пути, волновой алгоритм

Поиск кратчайшего пути:

0	1	2	3	4	5	6	7	8	9 10	11	
1	1	2	3	4	5	6		8	9 10	11	
2								9	9 10	11	
3	3	4	5	6	7	8					11
4	4	4	5	6	7	8		14 13	12 де	кабря	
					7	8		14 13	13 13		
12 11	10 9			8	8	8		14			
12 11	10 9			9	9	9		15 15	16 17		
11 Де	кабрь							16 16	16 17		
12 12	12 13	14 15	16								17
13 13	13 13	14 15	16 17	18 19	18 18	14 14	14 14	15 16	17 18	19 19	19

рекуррентные алгоритмы

Поиск короткого пути, волновой алгоритм

Анализ задачи, идея:

1. Если я знаю количество шагов для достижения своей ячейки, достигаю следующим шагом соседние клетки.

2. Если в соседней ячейке уже есть число, меньше мое + 1, то туда уже есть более короткий путь.
3. Если в соседней ячейке число больше мое, то нужно его заменить своим, то туда есть более короткий путь.

рекуррентные алгоритмы

Поиск короткого пути, волновой алгоритм

Алгоритм ШАГ (N):

1. Число шагов N (в начале 0).
2. Если в соседней ячейке уже есть число, меньше N + 1, то туда уже есть более короткий путь.
3. Если в соседней ячейке число больше мое, или там нет числа то ШАГ (N + 1) .

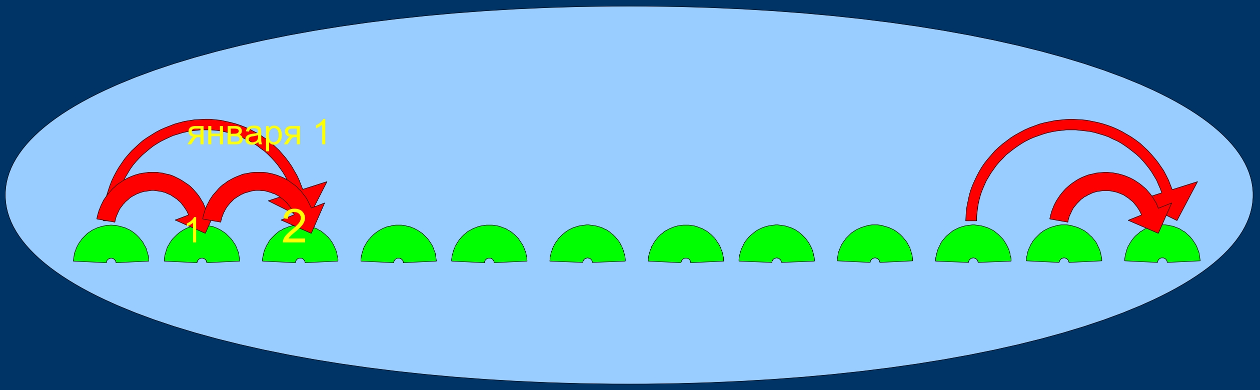
рекуррентные алгоритмы

динамическое программирование

Путь конька:

Задача:

Через болото ведут кочки. Конек может прыгнуть через кочку или на следующую. Найдите количество вариантов пути конька через болото.



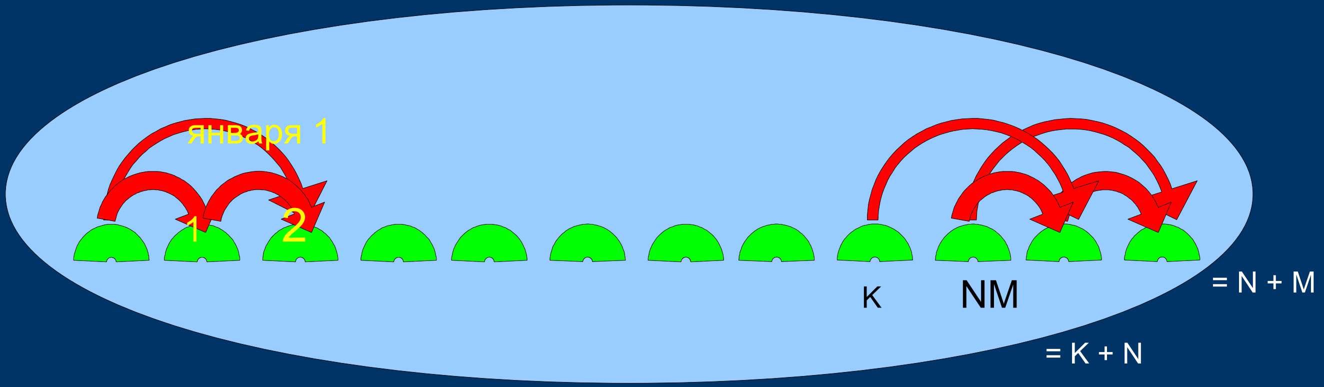
рекуррентные алгоритмы

динамическое программирование

Путь конька:

анализ:

Если известно количество путей с двумя предыдущими кочек, то количество путей будет суммой этих вариантов.



рекуррентные алгоритмы

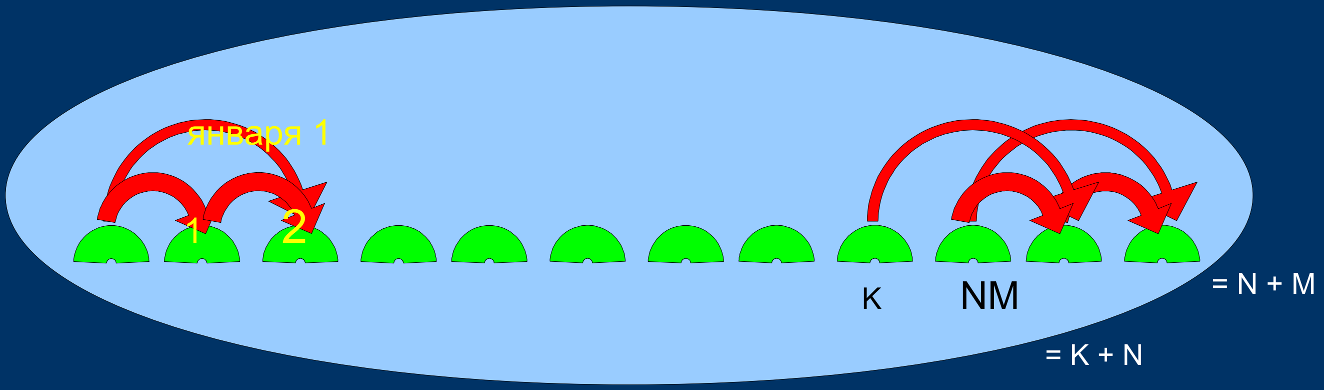
динамическое программирование

Путь конька:

алгоритм:

Варианты (0) = 1; Варианты (1) = 1;

Варианты (N) = варианты (N-1) + варианты (N-2).



рекуррентные алгоритмы

динамическое программирование

Путь конька:

Сложность алгоритма:

Алгоритм проследит все варианты, а при N + 1 количество вариантов почти удваивается - экспоненциальная сложность! Для N = 100 расчеты будут продолжаться 2^{100} микросекунд, или несколько миллиардов лет!

Вывод: в таком виде алгоритм неприменим.

рекуррентные алгоритмы

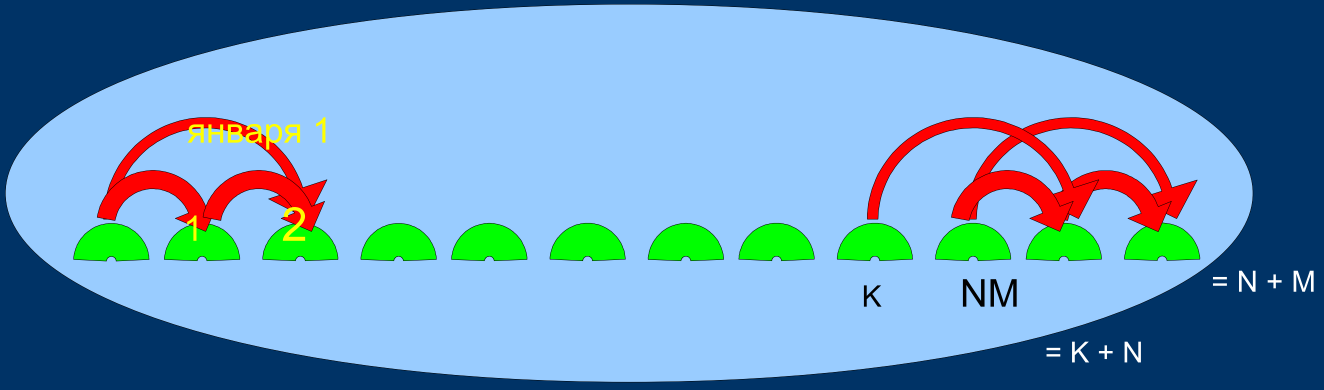
динамическое программирование

Путь конька:

расчет:

Варианты (M + 1) = варианты (M) + Варианты (N) . Варианты (M) = Варианты (N) + Варианты (K).

Вывод: для большинства кочек функция вызывается много раз.



рекуррентные алгоритмы

динамическое программирование

Путь конька:

Алгоритм:

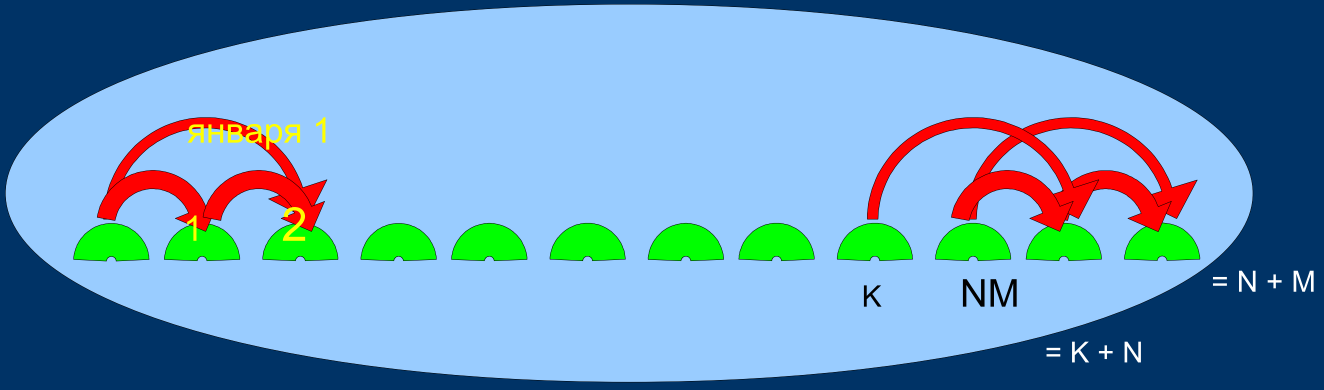
Массив:

1	1	2	…	F (N-2) F	(N-1) F (N)	

Варианты (N) = варианты (N-1) + Варианты (N-2) .

Если в таблице нет Варианты (N-1), тогда

Варианты (N-1) = Варианты (N-2) + Варианты (N-3).



рекуррентные алгоритмы

динамическое программирование

Путь конька:

Алгоритм:

Массив:

1	1	2	…	F (N-2)	F (N-1)	F (N)

Варианты (N) = варианты (N-1) + Варианты (N-2) .

Если в таблице нет Варианты (N-1), тогда

Варианты (N-1) = Варианты (N-2) + Варианты (N-3).

Для каждого числа в таблице расчет делается один раз. Алгоритм линейные.

рекуррентные алгоритмы

динамическое программирование

динамическое программирование - рекурсивный алгоритм в которой некоторые варианты упрощенных задач повторяются, и вместо их повторного расчета используют готовый ответ найденная ранее. преимущества:

Имеет меньшую сложность

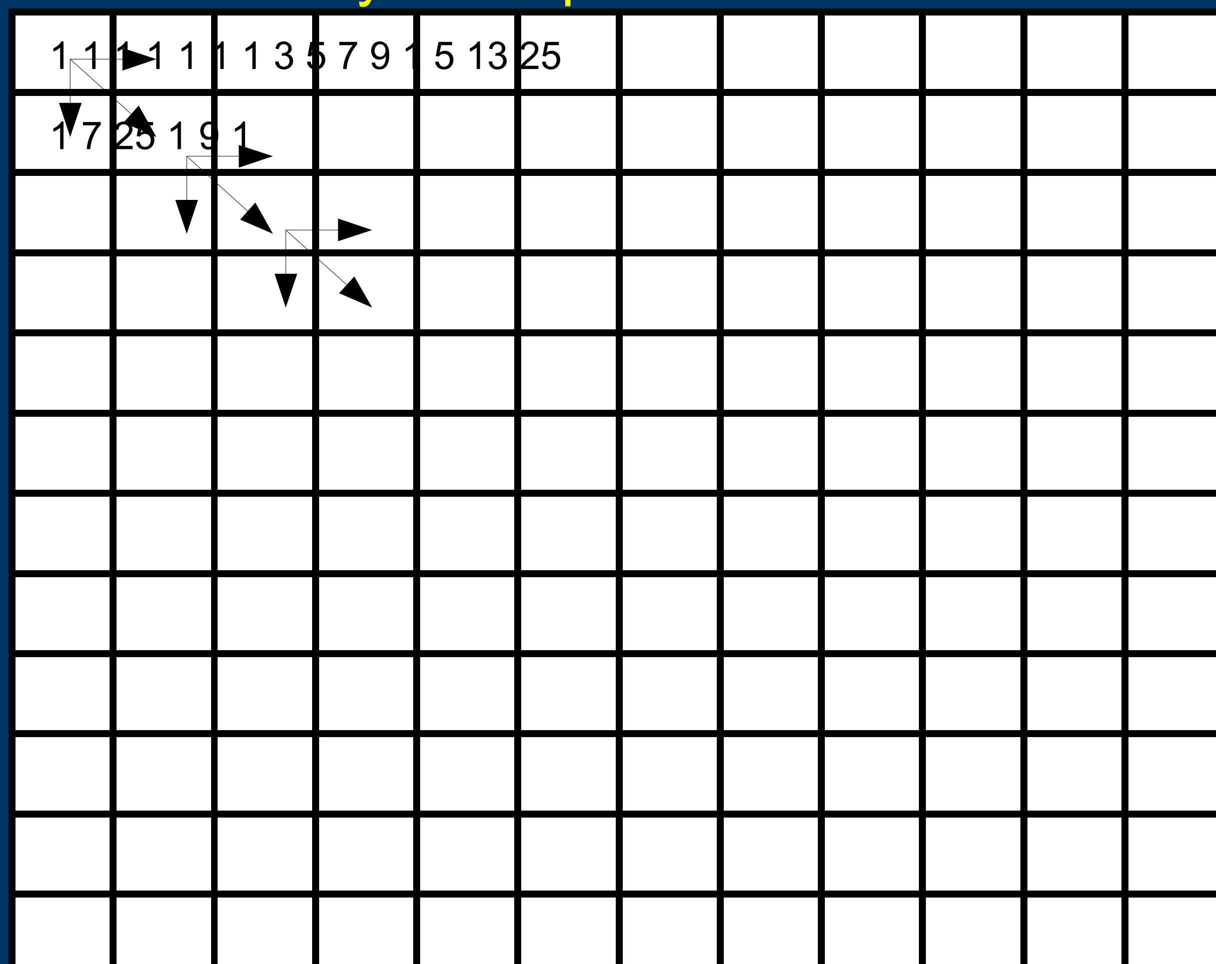
Недостатки:

Требует дополнительную память для промежуточных результатов.

рекуррентные алгоритмы

динамическое программирование

Задача на количество путей черепахи:



рекуррентные алгоритмы

динамическое программирование

Задача на количество путей черепахи.

Рекуррентный алгоритм Для

клетки x, y:

Вариантов (x, y) = вариантов (x-1, в) + вариантов (x, y-1)

+ Вариантов (x-1, в-1)

Вариантов (0,0) = 1.

Количество вызовов для одной клетки?
сложность 3_{nm}

(N, m) - размер поля

рекуррентные алгоритмы

динамическое программирование

Задача на количество путей черепахи.

Динамический алгоритм Для

клетки x, y:

Если (x, y) есть в таблице - возвращаем его. иначе

Вариантов (x, y) = вариантов (x-1, в) + вариантов (x, y-1)

+ Вариантов (x-1, в-1)

Вариантов (0,0) = 1.

Количество вызовов для одной клетки 1. Сложность

n * m (n, m) - размер поля

рекуррентные алгоритмы

динамическое программирование

Задача на количество путей черепахи.

Реализация:

```
int tabl [10] [10];
int variant ( int x, int y)


int main ()
{Printf ( "% d \ n", variant (9,9));

   return 0; }
```

рекуррентные алгоритмы

динамическое программирование

Задача на количество путей черепахи.

Реализация:

```
// int tabl [10] [10];
int variant ( int x, int y) { int n =
0;
    if (Tabl [x] [y]> 0) return tabl [x] [y];
    if (X> 0) n + = variant (x-1, y)
    if (Y> 0) n + = variant (x, y-1);
    if (X> 0 && y> 0) n + = variant (x-1, y-1); tabl [x] [y]
    = n;                     return n;
}
```