СТРОКИ

С-строки. Некоторые интересные алгоритмы на строках. Основы динамического программирования

К. Владимиров, Intel, 2019
mail-to: konstantin.vladimirov@gmail.com

Значения символов

- То, что вы привыкли считать символами, на самом деле кодируется числами

```
int c = 'A';
printf("%d\n", c); // что на экране?
```

- И наоборот достаточно небольшие цифры это коды для символов

```
char x = 65;
printf("%c\n", x); // а сейчас?
```

- В Unix формат символов это UTF-8. Его первые 127 символов совпадают с таблицей ASCII

Упражнение

- Распечатайте символы с `0` по 127 рядами по 16
- Чисто визуально на какие категории вы можете разделить эти символы?

Категории символов

- Специальный заголовочный файл <ctype.h> содержит прототипы функций, категоризирующих символы.
- Например is_alpha(c) проверяет является ли с одним из алфавитных символов (abcdef….), а is_digit(c) проверяет на цифру. Их объединяет is_alphanum(c)
- Иногда в программах которые анализируют символьную информацию важно знать что на вход пришло "что-то вроде буквы" или "что-то вроде пробела"
- На следующей картинке сведены основные определители из ctype

Категории символов

ASCII values			characters	iscntrl iswcntrl	isprint iswprint	isspace iswspace	isblank iswblank	isgraph iswgraph	ispunct iswpunct	isalnum iswalnum	isalpha iswalpha	isupper iswupper	islower iswlower	isdigit iswdigit	isxdigit iswxdigit
decimal	hexadecimal	octal													
0–8	\x0–\x8	\0–\10	control codes (NUL, etc.)	≠0	0	0	0	0	0	0	0	0	0	0	0
9	\x9	\11	tab (\t)	≠0	0	≠0	≠0	0	0	0	0	0	0	0	0
10–13	\xA–\xD	\12–\15	whitespaces (\n, \v, \f, \r)	≠0	0	≠0	0	0	0	0	0	0	0	0	0
14–31	\xE–\x1F	\16–\37	control codes	≠0	0	0	0	0	0	0	0	0	0	0	0
32	\x20	\40	space	0	≠0	≠0	≠0	0	0	0	0	0	0	0	0
33–47	\x21–\x2F	\41–\57	!"#$%&'()*+,-./	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
48–57	\x30–\x39	\60–\71	0123456789	0	≠0	0	0	≠0	0	≠0	0	0	0	≠0	≠0
58–64	\x3A–\x40	\72–\100	:;<=>?@	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
65–70	\x41–\x46	\101–\106	ABCDEF	0	≠0	0	0	≠0	0	≠0	≠0	≠0	0	0	≠0
71–90	\x47–\x5A	\107–\132	GHIJKLMNOP QRSTUVWXYZ	0	≠0	0	0	≠0	0	≠0	≠0	≠0	0	0	0
91–96	\x5B–\x60	\133–\140	[\]^_`	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
97–102	\x61–\x66	\141–\146	abcdef	0	≠0	0	0	≠0	0	≠0	≠0	0	≠0	0	≠0
103–122	\x67–\x7A	\147–\172	ghijklmnop qrstuvwxyz	0	≠0	0	0	≠0	0	≠0	≠0	0	≠0	0	0
123–126	\x7B–\x7E	\172–\176	{|}~	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
127	\x7F	\177	backspace character (DEL)	≠0	0	0	0	0	0	0	0	0	0	0	0

Источник: https://en.cppreference.com/w/c/string/byte/isspace

Disclaimer: мы не нырнём глубоко

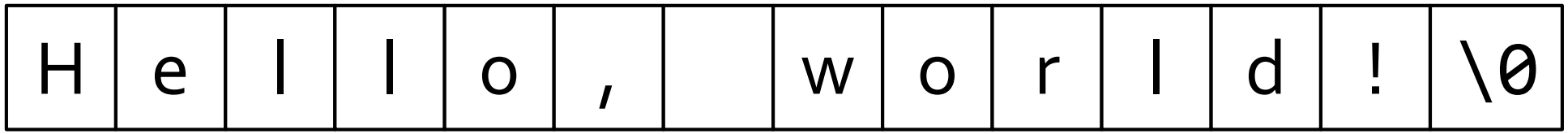
- Далее мы работаем только с ASCII строками то есть со строками, состоящими из однобайтных символов UTF8
- Учёт различных кодировок, русских букв, китайских иероглифов и всего такого просто не входит в программу первого курса
- Приветствуется если любознательная часть аудитории что-нибудь почитает самостоятельно

Строковые литералы

- Следующая строчка может показаться смутно знакомой

```
printf("%s\n", "Hello, world!");
```

- Символы `"Hello, world"` в кавычках это строковый литерал



- Строковый литерал это строка известная во время компиляции
- Любая строка в языке C оканчивается завершающим нулевым символом
- Нулевой символ `'\0'` это не символ `'0'`. Это спецсимвол, имеющий код `0`.

Изменяемые строки

- Самый простой способ создать изменяемую строку это объявить массив символов

```
char hello[20] = "Hello, world!";
```

- Символы после завершающего нуля (он тут 14-й) содержат мусор
- Далее содержимое этой строки можно поменять

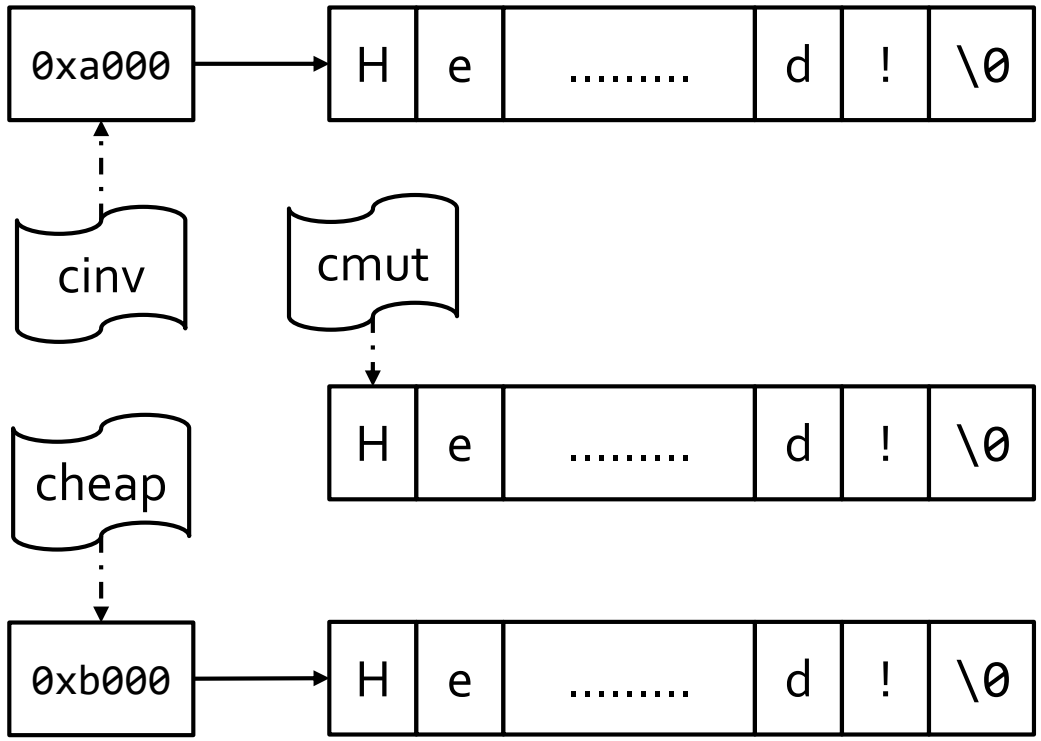
```
hello[1] = 'a';

printf("%s\n", hello); // → Hallo, world!
```

- Изменяемая строка это любой массив символов, завершающийся нулём
- Символы после завершающего нуля ничего не значат

C-строки и разные виды памяти

- Обычно используют указатель на первый элемент

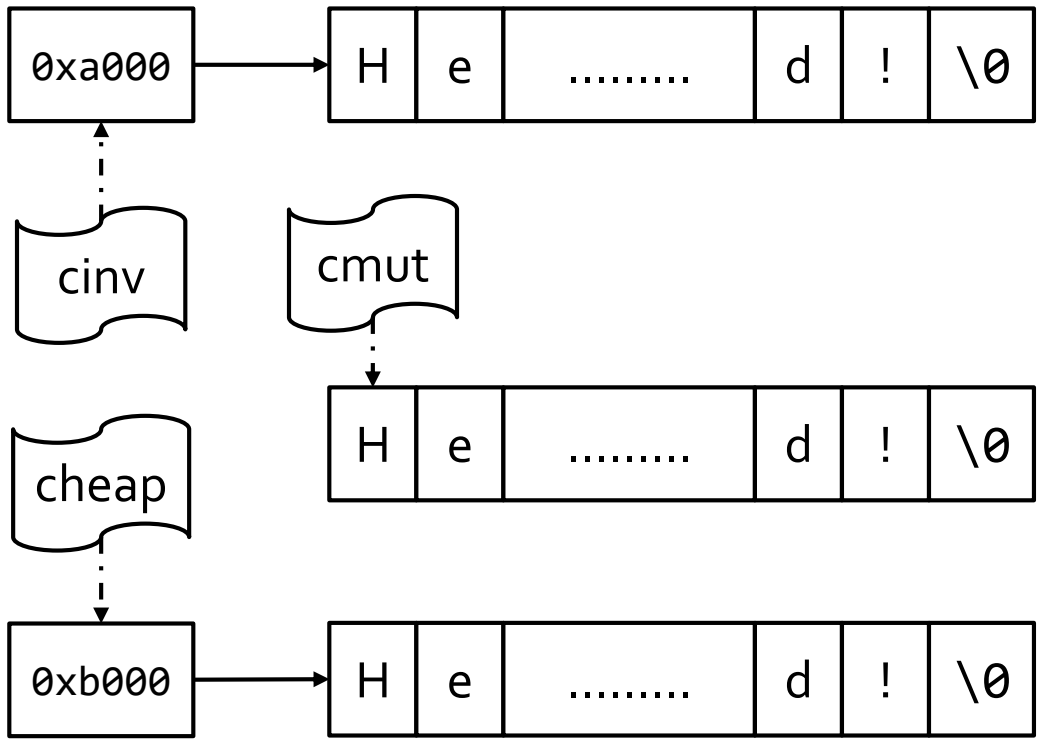
```
char const * cinv = "Hello, world";
char cmut[] = "Hello, world";
char *cheap = (char *) malloc (50);
strcpy (cheap, cinv);
cheap = cinv;

cinv = 0;
cmut = cheap;
```



C-строки и разные виды памяти

- Обычно используют указатель на первый элемент

```
char const * cinv = "Hello, world";
char cmut[] = "Hello, world";
char *cheap = (char *) malloc (50);
strcpy (cheap, cinv);
cheap = cinv; // ошибка компиляции
              + утечка памяти
cinv = 0; // ok
cmut = cheap; // ошибка компиляции
```



Problem SCN – суммирование в строке

- Ваша задача написать программу которая суммирует строчку, воспринимая символы в ней как коды этих символов

H	e	l	l	o	,		w	o	r	l	d	!	\0

#72	#101	#108	#108	#111	#44	#32	#119	#111	#114	#108	#100	#33	#0

- Для `"Hello, world!"` ответом будет `1161`

Копирование строк

- Одна строка может быть скопирована в другую

```
char hello[20] = "Hello, world!";
char duplicate[20];

char const * src = hello;
char * dst = duplicate;

while (*src != '\0') {
  *dst = *src; dst += 1; src += 1;
}

*dst = '\0';
```

- Все ли понимают что происходит в этом коде?

Копирование строк

- Одна строка может быть скопирована в другую

```
char hello[20] = "Hello, world!";
char duplicate[20];

char const * src = hello;
char * dst = duplicate;

while ((*dst++ = *src++) != '\0') {} // немного джигитовки
```

Копирование строк

- Одна строка может быть скопирована в другую

```
char hello[20] = "Hello, world!";
char duplicate[20];

strcpy (duplicate, hello); // используем библиотечную функцию
```

- Вашим выбором по умолчанию должно быть именно использование библиотечных функций, так как они почти всегда корректны и куда лучше оптимизированы
- В языке C довольно много библиотечных функций для работы со строками

API для работы с C-строками

Библиотечная функция	**Что она делает**
`strcpy(dest, src)`	Копирует src в dest
`strcat(dest, src)`	Дописывает src в конец dest
`strlen(src)`	Вычисляет длину src
`strcmp(src1, src2)`	Сравнивает `src1` и `src2`. Возвращает `0`, `1` или `-1`
`strchr(src, c)`	Ищет символ в строке
`strstr(haystack, needle)`	Ищет подстроку в строке
`strspn(src1, src2)`	Ищет максимальный префикс `src1` состоящий только из символов `src2`
`strcspn(src1, src2)`	Тоже только не из символов `src2`
`strpbrk(src1, src2)`	Ищет первое вхождение в `src1` любого символа из `src2`

Обсуждение

- Одна забавная странность в сигнатурах стандартной библиотеки заключается в том, что возвращаемый тип почти всегда `char*`

```
char *strstr(const char *s1, const char *s2);
```

- Казалось бы, если мы принимаем оба аргумента `const char*` почему бы не возвращать `const char*`
- У кого есть идеи почему так сделано?

Problem SR – переворот подстрок

- Напишите программу, которая берёт одно слово, а потом некоторый текст и переворачивает в этом тексте все вхождения этого слова
- Например для слова `"world"` и текста `"Hello, world!"` результатом должно быть `"Hello, dlrow!"`
- Вы можете предполагать что текст состоит из слов, разделённых пробелами и пунктуацией, а слово состоит из алфавитных символов
- Вы также можете использовать любые функции стандартной библиотеки, включая `strstr` для поиска подстроки

Конкатенация строк

- Рассмотрим пристальней функцию strcat

```
char buf[20] = "Hello";
strcat(buf, ", world!"); // → "Hello, world!"
```

- Как она могла бы работать?

Конкатенация строк

- Рассмотрим пристальней функцию strcat

```
char buf[20] = "Hello";
strcat(buf, ", world!"); // → "Hello, world!"
```

- Как она могла бы работать?
- Единственный алгоритм: дойти до конца строки-приёмника, а потом по одному добавлять символы из строки-источника
- При этом мы тратим $O(N + M)$ времени

Проблема Шлемиля-маляра

- Следующий код довольно плох

```
char buf[MAXSZ];
strcpy(buf, "First ");
strcat(buf, "Second ");
strcat(buf, "Third ");
strcat(buf, "Fourth ");
```

- Что здесь не так? Как бы вы переписали этот код?

Обсуждение

- Ещё одной проблемой strcat (кроме обсуждённых ранее проблем с асимптотикой) является возможное переполнение буфера
- Допустим у нас строки живут только в динамической памяти
- Можем ли мы тогда написать strcat которая увеличивает буфер, если его не хватает? Как она могла бы работать?

Реаллокации

- Чтобы немного расширить буфер в динамической памяти, делать `malloc` на новый размер и `free` на старом месте может быть накладно
- Здесь на помощь приходит `realloc`

```
void * realloc(void * ptr, size_t new_size);
```

- Например

```
int * arr = (int *) malloc(100);
arr = (int *) realloc(arr, 1000);
```

- При нехватке памяти, `realloc` возвращает `NULL`
- Внимание: `realloc` не работает на стеке и в глобальной памяти

Problem SA – concat + realloc

- Ваша задача написать функцию

```
char *strcat_r(char *dest, int bufsz, const char *src);
```

- При нехватке места в старом буфере эта функция должна реаллоцировать память и возвращать новый указатель
- Будьте очень внимательны с нулевым символом

Problem SP – замена в строке

- Реализуйте функцию, осуществляющую замену всех подстрок в строке
- Функция должна вернуть новую строку, аллоцированную в куче

```
char * replace(char *str, char const *from, char const *to);
```

- Например

```
char *s = (char *) malloc(41);
strcpy(s, "Hello, %username, how are you, %username");

char *r = replace(s, "%username", "Eric, the Blood Axe");
```

- Обратите внимание, что строка from может быть и длиннее и короче чем to

Аргументы функции main

- До сих пор мы пользовались `main()` или `main(void)`
- Но есть вторая форма основной функции

```
int main(int argc, char **argv) {
  // тело функции
}
```

- `argc` это количество аргументов
- `argv` это массив `argc+1` строк
- `argv[0]` это имя самой программы

Problem EA – вывести аргументы main

- Напишите программу которая будет печатать аргументы с которыми вызвана, заменяя `'-'` на `'+'`

```
> ./a.out -a t -x --b 123
-a t +x ++b 123
```

- Если среди этих аргументов есть `-r`, программа должна вывести их в обратном порядке

```
> ./a.out -a t -r -x --b 123
123 ++b +x +r t +a
```

- Как вы думаете почему именно `'-'` является стандартным для аргументов командной строки?

Problem SU – поиск CI подстроки

- Напишите простую функцию `strstrci`, которая ищет подстроку в строке, независимо от регистра символов (`ci` означает `case insensitive`)

```
char * strstrci(char const * needle, char const * haystack);
```

- Ниже приведён пример применения

```
char const *needle = "Ab", *src = "abracadaBra";
char *pos1, *pos2, *pos3;

pos1 = strstrci(src, needle);        assert(pos1 != NULL);
pos2 = strstrci(pos1 + 2, needle);   assert(pos2 != NULL);
pos3 = strstrci(pos2 + 2, needle);   assert(pos3 == NULL);
```

- Первая найденная позиция "abracadabra", вторая "abracadaBra"
- Оцените асимптотику наивного алгоритма

HWK – алгоритм КМП (optional)

- Более интересный алгоритм Кнута-Морриса-Пратта делает из строки автомат, вычисляя так называемую failure function `f[i]`

W[i]	A	B	A	C	A	B
S[0:i]	A	AB	ABA	ABAC	ABACA	ABACAB
f[i]	0	0	1	0	1	2

- Далее с использованием этой табличной функции идёт поиск

1	2	3																		
A	B	A	B	A	C	A	A	B	A	C	A	B	C	A	C	B	C	A	A	A
A	B	A	C	A	B															

- Смотрим `f[2] == 1`, сдвигаем на `3 - f[2] == 2`

HWK – алгоритм КМП (optional)

- Более интересный алгоритм Кнута-Морриса-Пратта делает из строки автомат, вычисляя так называемую failure function `f[i]`

W[i]	A	B	A	C	A	B
S[0:i]	A	AB	ABA	ABAC	ABACA	ABACAB
f[i]	0	0	1	0	1	2

- Далее с использованием этой табличной функции идёт поиск

		1	2	3	4															
A	B	A	B	A	C	A	A	B	A	C	A	B	C	A	C	B	C	A	A	A
		A	B	A	C	A	B													

- Смотрим `f[3] == 0`, сдвигаем на `4 - f[3] == 4`

HWK – алгоритм КМП (optional)

- Более интересный алгоритм Кнута-Морриса-Пратта делает из строки автомат, вычисляя так называемую failure function `f[i]`

W[i]	A	B	A	C	A	B
S[0:i]	A	AB	ABA	ABAC	ABACA	ABACAB
f[i]	0	0	1	0	1	2

- Далее с использованием этой табличной функции идёт поиск

						1													
A	B	A	B	A	C	A	A	B	A	C	A	B	C	A	C	B	C	A	A	A
						A	B	A	C	A	B								

- Смотрим `f[0] == 0`, сдвигаем на `1 - f[0] == 1`

HWK – алгоритм КМП (optional)

- Более интересный алгоритм Кнута-Морриса-Пратта делает из строки автомат, вычисляя так называемую failure function `f[i]`

W[i]	A	B	A	C	A	B
S[0:i]	A	AB	ABA	ABAC	ABACA	ABACAB
f[i]	0	0	1	0	1	2

- Далее с использованием этой табличной функции идёт поиск

							m	a	t	c	h	!								
A	B	A	B	A	C	A	A	B	A	C	A	B	C	A	C	B	C	A	A	A
							A	B	A	C	A	B								

- Нашли подстроку. **Дома: реализуйте этот алгоритм на языке C**

Расстояние редактирования

- Как перейти от слова spoon к слову sponge?
- Что можно делать:
  - Вставлять букву в любое место (цена == 1)
  - Удалять любую букву (цена == 1)
  - Изменять любую букву (цена == 2)
- Нужно найти последовательность действий с минимальной ценой

spoon → sponn (+2) → spong (+2) → sponge (+1), cost = 5

spoon → spon (+1) → spong (+1) → sponge (+1), cost = 3

Обсуждение

- Можете ли вы предложить наивное решение для расстояния редактирования?

Обсуждение

- Можете ли вы предложить наивное решение для расстояния редактирования?
- Скорее всего оно очень плохо.
- Для такого рода задач обычно используется дискретное динамическое программирование

Принцип Беллмана

- «An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision» $[Bellman]$
- Изменение состояния: $x_t \to x_{t+1} = T(x_t, a_t)$ with pay-off $F(x_t, a_t)$
- Обозначим $V(x_0) = \max\limits_{a_{0\ldots}a_T} \sum_t F(x_t, a_t)$
- Тогда $V(x_t) = \max\limits_{a_t}\{F(x_t, a_t) + V\big(T(x_t, a_t)\big)\}$
- Простыми словам: любой кусок оптимальной траектории является оптимальной траекторией

Размен монет

- Приложение к размену монет: пусть есть монеты номиналами c_1, c_2, c_3 и необходимо разменять сумму N
- Заведём таблицу $V(x_k)$ от 0 до N и протабулируем оптимальные размены. Пусть есть номиналы $1,3,4$ и надо разменять 10 используя минимальное количество монет

k	1	2	3	4	5	6	7	8	9	10
V	1	2								

Размен монет

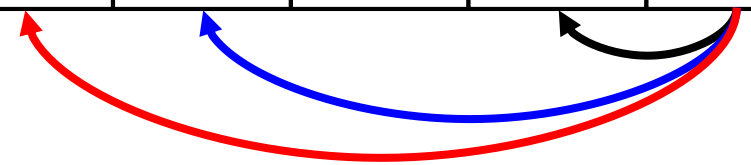
- Приложение к размену монет: пусть есть монеты номиналами c_1, c_2, c_3 и необходимо разменять сумму N
- Заведём таблицу $V(x_k)$ от 0 до N и протабулируем оптимальные размены. Пусть есть номиналы $1,3,4$ и надо разменять 10 используя минимальное количество монет

k	1	2	3	4	5	6	7	8	9	10
V	1	2	1	2						

Размен монет

- Приложение к размену монет: пусть есть монеты номиналами c_1, c_2, c_3 и необходимо разменять сумму N
- Заведём таблицу $V(x_k)$ от 0 до N и протабулируем оптимальные размены. Пусть есть номиналы $1,3,4$ и надо разменять 10 используя минимальное количество монет

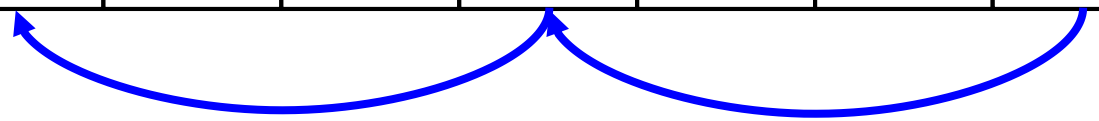
k	1	2	3	4	5	6	7	8	9	10
V	1	2	1	2	2					



Размен монет

- Приложение к размену монет: пусть есть монеты номиналами c_1, c_2, c_3 и необходимо разменять сумму N
- Заведём таблицу $V(x_k)$ от 0 до N и протабулируем оптимальные размены. Пусть есть номиналы $1,3,4$ и надо разменять 10 используя минимальное количество монет

k	1	2	3	4	5	6	7	8	9	10
V	1	2	1	1	2	2	2	2	3	3



Problem MC – размен монет

- Вам необходимо написать программу, которая считывая со стандартного ввода:
- Сумму размена
- Число монет для размена
- Номиналы монет для размена
- Выдавала бы на стандартный вывод минимальное количество монет для размена

Problem BP – задача о рюкзаке

- Вам необходимо написать программу, которая считывая со стандартного ввода:
- Количество вещей
- Вес каждой вещи
- Общий вес входящий в рюкзак
- Выдавало бы наибольшее количество вещей которые можно положить в рюкзак
- Например для: 4, $\{1, 3, 3, 4\}$, 10 ответ будет 3

Расстояние редактирования

- Как перейти от слова spoon к слову sponge?
- Что можно делать:
  - Вставлять букву в любое место (цена == 1)
  - Удалять любую букву (цена == 1)
  - Изменять любую букву (цена == 2)
- Нужно найти последовательность действий с минимальной ценой

spoon → sponn (+2) → spong (+2) → sponge (+1), cost = 5

spoon → spon (+1) → spong (+1) → sponge (+1), cost = 3

- Можем ли мы применить динамическое программирование?

Динамическое программирование

- Любая часть оптимальной траектории является оптимальной траекторией

$$D(i,j) = min\begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1)+K(i,j) \end{cases}$$

$$K(i,j) = \begin{cases} 0, & S_1(i) = S_2(j) \\ 2, & S_1(i) \neq S_2(j) \end{cases}$$

- Задача теперь сводится к тому, чтобы найти $D(5,6)$
- Потренируемся заполнять таблицу

	#	S	P	O	N	G	E
#	0	1	2	3	4	5	6
S	1	$D(1,1)$					
P	2						
O	3						
O	4						
N	5						$D(5,6)$

Динамическое программирование

- Любая часть оптимальной траектории является оптимальной траекторией

$$D(i,j) = min \begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1)+K(i,j) \end{cases}$$

$$K(i,j) = \begin{cases} 0, & S_1(i) = S_2(j) \\ 2, & S_1(i) \neq S_2(j) \end{cases}$$

- Задача теперь сводится к тому, чтобы найти $D(5,6)$
- Потренируемся заполнять таблицу

	#	S	P	O	N	G	E
#	0	1	2	3	4	5	6
S	1	0	1	2	3	4	5
P	2	1	0	1	2	3	4
O	3	2	1	0	1	2	3
O	4	3	2	1	2	3	4
N	5	4	3	2	1	2	3

Problem ED – расстояние редактирования

- Реализуйте программу, которая считывает со стандартного ввода
  - стоимость добавления
  - стоимость удаления
  - стоимость замены
  - длину первой строки
  - первую строку
  - длину второй строки
  - вторую строку
- И выводит на стандартный вывод минимальное расстояние редактирования

Литература

- [C11] ISO/IEC – "Information technology – Programming languages – C", 2011
- [$K\&R$] Brian W. Kernighan, Dennis Ritchie – The C programming language, 1988
- [JSP] Joel Spolsky – Back to Basics, 2001
- [$Linden$] Peter van der Linden – Expert C Programming: Deep C Secrets , 1994
- [$Bellman$] Richard Bellman – Dynamic Programming, 1957
- [$Cormen$] Thomas H. Cormen – Introduction to Algorithms, 2009
- [$TAOCP$] Donald E. Knuth – The Art of Computer Programming, 2011
- [$SALG$] Robert Sedgewick Algorithms, 4th edition, 2011