OS Theory Exercises pt. 2

(special thanks to maco )

Season Finale

Page Replacement Algorithms

Frame Number = 4

		FIFO		LRU		LFU		
#	Page	Hit / Miss	Frame	Hit / Miss	Frame	Hit / Miss	Frame	Repeats
0	**7**	0	0	0	0	0	?	
1	**0**	0	1	0	1	0	?	
2	**1**	0	2	0	2	0	?	
3	**2**	0	3	0	3	0	?	
4	**0**	1	1	1	1	?	?	1
5	**3**	0	0	0	0	0	?	
6	**0**	1	1	1	1	?	?	2
7	**4**	0	1	0	2	0	?	
8	**2**	1	3	1	3	?	?	3
9	**3**	1	0	1	0	?	?	4
10	**0**	0	2	1	1	?	?	5
11	**3**	1	0	1	0	?	?	6
12	**2**	1	3	1	3	?	?	7
13	**1**	0	3	0	2	?	?	8
14	**2**	0	0	1	3	?	?	9
15	**0**	1	2	1	1	?	?	10
16	**1**	1	3	1	2	?	?	11
17	**7**	0	1	0	0	?	?	12
18	**0**	1	2	1	1	?	?	13
19	**1**	1	3	1	2	?	?	14

Hit / Miss = 0 секаде каде што прв пат се појавува страницата.

FIFO

Брз начин за Hit/Miss:
За секој празен ред, провери дали страницата ја има во предходните 4 (frame number) реда (**игнорирај ги редовите кои имаат 1**). Стави 1 ако ја има – 0 ако ја нема.

Frame: Со помошна табела.

LRU

Брз начин за Hit/Miss:
За секој празен ред, почнуваш од редот еднаков на вредноста **Repeats** и проверуваш дали таа страница се наоѓа во следните 4 (frame number) реда (вклучувајќи го почетниот). Стави 1 ако да – 0 ако не.

Frame: Секој пат кога ќе дојде страница што ја нема во рамките, броиш 4 (frame number) реда назад без дупликати и ја копираш таа рамка.

LFU

Hit/Miss: ...

Frame: ...

Тешко се објаснуваат преку текст ☹️

Еден компјутерски систем има **256 KB** виртуелна меморија. Големината на физичката меморија е **8 KB**, организирани во рамки со големина од **2048 B**. Времето за вчитување на податок при погодок е **300 ns**, а времето потребно за добивање на податок при промашување е **20 μs**.

[1] Форматот на адресите во овој систем е:

Virtual address		Physical address	
Page	Offset	Frame	Offset
7	11	2	11

V-Mem: 262 144 B — 256 * 1024
Ph-Mem: 8192 B — 8 * 1024
Frame Size: 2048 B
Hit Time: 300 ns
Miss Time: 20 000 ns — 20 * 1000

[2] Нека процесот генерира адреси кои припаѓаат на страниците **7d**, **1d**, **8**, **7d**, **63**, **8**, **5**, **3c**, **63**, **8**, **63**, **5** и алгоритмот за замена е **FIFO**.

Страница (Page Hex)	Погодоци (Hits – 0 or 1)	Рамка (Frame)
7d	0	0
1d	0	1
8	0	2
7d	1	0
63	0	3
8	1	2
5	0	0
3c	0	1
63	1	3
8	1	2
63	1	3
5	1	0

[3] Вкупното време на извршување на мемориските барања на оваа програма е: 121 800 ns

[4] Виртуелната адреса **28F9** (hex) се мапира во физичката адреса F9 (hex).

[5] Физичката адреса **588** (hex) се мапира во виртуелната адреса 2D88 (hex).

---

[1] Offset = $2^{???} = 2048 \rightarrow 2^{11} = 2048$

Page = $\frac{V_{mem}}{Frame\ Size} = \frac{262144}{2048} = 128 = 2^7 \rightarrow 7$

Frame = $\frac{Ph_{mem}}{Frame\ Size} = \frac{8192}{2048} = 4 = 2^2 \rightarrow 2$

---

[2] Објаснето горе, само пази на големината на рамката (во случајов е $2^2 = 4$).

---

[3] $Total\ Time = H_a * H_t + M_a * M_t = 6 * 300 + 6 * 20000 = 1800 + 120000 = 121\ 800ns$

a = amount, t = time

---

[4] 28F9 = 0010 1000 1111 1001 — 16 бита
V_{addr} = Page + Offset = 7 + 11 = — 18 бита

2 бита фалат (18 – 16), па додаваме две нули на бинарната адреса: 00 0010 1000 1111 1001

Бинарната адреса ја делиме на два дела (еден со 7 бида и другиот со 11): 0000101 | 00011111001

Вредноста на првиот дел е 5. Во табелата **[2]** од долу нагоре гледаме дека страница 5 е сместена во рамка 0, па првиот дел го менуваме со 0:
0000000 | 00011111001.

Целата адреса ја претвараш во hex:
00011111001 – F9

[5] 588 = 0101 1000 1000 — 12 бита
Ph_{addr} = Frame + Offest = 2 + 11 = — 13 бита

1 бит фали (13 – 12), па додаваме една нула на бинарната адреса: 0 0101 1000 1000

Бинарната адреса ја делиме на два дела (еден со 2 бида и другиот со 11): 00 | 10110001000

Вредноста на првиот дел е 0. Во табела **[2]** од долу нагоре гледаме дека во рамка 0 е сместена страница 5, па првиот дел го менуваме со 5:
0101 | 10110001000

Целата адреса ја претвараш во hex:
010110110001000 – 2D88

Нека компјутерски систем има страници со големина од **512 B**, виртуелен адресен простор од **16 KB** и физичка меморија со големина **4 KB**. Притоа, дадена програма ја има следната табела со страници (прикажани се само валидните во хексадецимален формат).

Page (hex)	Frame (hex)
0	3
1	0
3	2
6	4
7	1

V-Mem:	16 384 B	16 * 1024
Ph-Mem:	4096 B	4 * 1024
Page Size:	512 B	
Време за пристап до страниците 0, 3, 7:	10 ns	
Време за пристап до главна меморија:	200 ns	
Време за носење на страна од диск:	2000 ns	2 * 1000

Системот користи Translation Lookaside Buffer со записи на страните 0, 3 и 7 со време за пристап од **10 ns**, времето на пристап до главната меморија е **200 ns**, а времето за носење на страна од дискот е **2 μs**.

Форматот на адресите во овој систем е:

Virtual address		Physical address	
Page	Offset	Frame	Offset
5	9	3	9

Offset = $2^9 = 512 \rightarrow 9$
Page = $\frac{16384}{512} = 32 = 2^5 \rightarrow 5$
Frame = $\frac{4096}{512} = 8 = 2^3 \rightarrow 3$

Табелата на страници за секој процес има вкупно 32 (2^{Page}) записи, а во физичката меморија има вкупно 8 (2^{Frame}) рамки.

Одговорете на следните прашања доколку имате барање за адреса **635** (hex).

Во која страница припаѓа? 3 (hex)
Дали имаме погодок во TLB табелата? Да (кажано ни е дека TLB има записи за 0, 3 и 7)
Дали имаме погодок во табелата со страници? Да (во табелата страницата 3 има рамка 2)
До TLB табелата се пристапува 1 пат(и), до главната меморија 1 пат(и) и до дискот се пристапува 0 пат(и).
За колку време ќе се добие податокот од адресата **635** (hex)? 210 ns.
Физичката адреса која ќе се побара ќе биде 035 (hex).

635 (hex) → 0110 0011 0101 (binary)
Последните 9 бида се offset, првите 3 ја даваат страницата (011 = 3).

Страницата 3 ја има само еднаш во двете табели, па до дискот нема потреба да се пристапи.
Page Access Time + Mem Access Time = 10 + 200 = 210

Како во предходната задача.

Дополнително, нека оперативниот систем користи **алгоритам за замена на најодамна користена страница со стареење** (LRU – Aging), каде за секоја страница се чуваат пристапите од последните **3 интервали (clock tick)**. Нека се референцирани следните страници во секој од интервалите (интервалот 5 е последен).

Интервал (clock tick)	Пристапени страници (hex)
1	1, 3, 6, 7
2	1, 7
3	1, 6, 7
4	0, 3, 6
5	0, 3, 7

Во овој оперативен систем, за секоја страница ќе се чува 3 (3 интервали) битови за определување на пристапите во минатите интервали.

Пополнете ги **R битовите** на секоја од страниците, по завршувањето на интервалот 5:

Page (hex)	R битови
0	110
1	001
3	110
6	011
7	101

Ако физичката меморија е полна ќе се замени страницата: 1 (least recently used)

Страница 0 е референцирана во интервал 5 и 4, но не во 3 → 110.
Страница 1 не е референцирана во интервал 5 и 4, но во 3 е → 001.
Страница 3 е референцирана во интервал 5 и 4, но не во 3 → 110.
... итн ...

Еден систем користи страничење со инвертирана табела чија содржина е следната:

PID	Page
1	4
2	4
1	3
5	2
1	9
2	5
1	6

Според табелата може да се заклучи дека бројот на виртуелни страници е / (не може да се определи), а бројот на рамки е 7 (7 реда во табелата). Кај овој тип на страничење, големината на табелата на страници зависи од бројот на рамки (колку рамки толку редови). Ако процесот со PID = 1 генерирал виртуелна адреса која припаѓа на страница 3, тогаш, според дадената табела, страницата се наоѓа во рамка 2 (трети ред, ама нула индексирани се, па затоа 2).

Дадена е шема во која **x** означува дека меморискиот блок е веќе пополнет со некој процес, а **–** означува дека меморискиот блок е слободен за алокација: | x | – | – | – | – | x | x | x | – | – | x | x | – | – | – | x |

Доколку се користи битмапата, тогаш нејзината вредност за блокот на шемата ќе биде 0111100011001110 (0 за x и 1 за –), а доколку се користи поврзана листа, таа ќе биде составена од **[1]** 7 јазли и содржината на првите три јазли ќе биде **[2]** P 0 1 H 1 4 P 5 3.

[1] Јазли:
1. | x |
2. | – | – | – | – |
3. | x | x | x |
4. | – | – |
5. | x | x |
6. | – | – | – |
7. | x |

[2] P = process (x) H = hole (–)

Јазол 1: P 0 1 е процес, индекс 0, должина 1
Јазол 2: H 1 4 е дупка, индекс 1, должина 4
Јазол 3: P 5 3 е процес, индекс 5, должина 3

При пребарување на слободни блокови во меморијата, подобри перформанси дава поврзана листа.

Ако е потребно да се резервираат **два блока**, тогаш според **Best Fit** ќе се резервира блокот со индекс 8. Според **First Fit** ќе се резервира блокот со индекс 1. Ако се користи **Next Fit**, под претпоставка дека предходно бил доделен блок со адреса 10, ќе се резервира блокот со индекс 12. Ако се користи **Quick Fit**, тогаш се користит табела, каде секоја ставка покажува кон листа од слободни блокови со иста големина.

BEST FIT
Ако треба да ставиш 2 мачки во кутија што собира точно 2 мачки или во кутија што собира 3 мачки, во која ќе ги ставиш?

На индекс 8 има точно место за 2 блока.

FIRST FIT
Ако те мрзи да бараш низ кутии, ги ставаш мачките во првата што ќе ги собере.

На индекс 1 има место за 4 блока, па таму ќе ги сместиш овие 2.

NEXT FIT
Брат му на First Fit ама се прави важен па почнува од индекс 10.

Почнувајќи од индекс 10, на индекс 12 има доволно место за 2 блока.

Еден компјутерски систем имплементира виртуелна меморија со големина на страница од **16 B**. Капацитетот на виртуелниот простор е **512 B**. Капацитетот на физичката меморија е **128 B**. Содржината на **дел** од табелата на страници е следната:

Индекс	Рамка	Валидност
0	2	1
1	3	1
2	4	0
3	1	1
4	5	1
5	1	0
6	7	1

Големината на виртуелната адреса е 9 бита, од кои 5 бита се за страница, а 4 бита за поместување.

Големината на физичката адреса е 7 бита од кои 3 бита се за рамка, а 4 бита се за поместување.

Целата табела на страници има вкупно 32 ставки.

Виртуелната адреса 001000111 се мапира во физичката адреса 1010111.

Виртуелната адреса на еден систем има 5 бита за првото ниво, 5 бита за второто ниво и 6 бита за офсет.

Првото ниво ќе се состои од 1 (секогаш почнуваме со една) табела/и со по 32 (2^5 за прво ниво) ставки.

Второто ниво ќе се состои од 32 (32 ставки од предходната табела) табела/и со по 32 (2^5 за второ ниво) ставки. Секоја од овие табели не мора да биде во меморија.

Ако не се користат повеќенивоски табели, тогаш табелата на страници на овој систем ќе има големина од 1024 (32 * 32) ставки.

Нека еден датотечен систем има **64 000 блокови**. Секој блок има големина од **200 B** и адреса од **2 B**. Доколку се користи евиденција за слободни блокови во листа, тогаш еден блок има капацитет да смести информација за 100 (200 / 2) слободни блокови. Според тоа, ако сите блокови се слободни, целата листа на слободни блокови би зафаќала 640 (64 000 / 100) блокови, а ако се слободни само 50 блокови, тогаш листата ќе зафаќа 1 (секој блок има капацитет за 100 слободни блока, па доволно е само 1 за 50) блокови. Ако се користи евиденција за слободни блокови со бит-мапа, тогаш, во првиот случај, мапата ќе зафаќа 40 (64 000 / [200 * 8]) блокови, а во вториот случај 40 (зафаќа исто во двата случаи) блокови.

Еден датотечен систем користи i-nodes структури кои имаат по 20 директни покажувачи, 1 единечен, 1 двоен и 2 тројни индиректени покажувачи. Претпоставете дека големината на блокот е 500 B, а адресата на секој блок е со големина од 4 B. Според тоа:

Со помош на директните покажувачи може да се пристапи до вкупно 10 000 (20 * 500) податоци.

Со еден единечен индиректен блок може да се адресира вкупно 125 (500 / 4) блокови.

Со еден двоен индиректен блок може да адресира вкупно 15 625 (125^2) блокови.

Со еден троен индиректен блок може да адресира вкупно 1 953 125 (125^3) блокови.

Вкупниот број на блокови кои можат да се адресираат со дадениот i-node изнесува 3 922 020.
20 (директни) + 125 (единечен) + 15 625 (двоен) + 1 953 125 (прв троен) + 1 953 125 (втор троен)

Доколку претпоставиме дека во кешот на имиња (name cache) се наоѓа само бројот на индексниот јазел на root именикот и нема ништо друго и дека во root именикот има многу малку записи, тогаш потребни се вкупно 6 диск читања за да се прочита блокот 100 од датотеката **/bar/foo/os.txt**.
Се чита блок 100 од /bar/foo/os.txt
Дали блокот спаѓа во директните? Не, бидејќи ги има 20 (0 – 19), а треба да се прочита блок 100.
Дали блокот спаѓа во индиректните единечни? Да, бидејќи ги има 125, а треба да се прочита блок 100.

Читања:
- 1 за root (или “ / “)
- 1 за bar
- 1 за foo
- 1 за os.txt
- 1 за индиректен единечен
- 1 за data

Вкупно 6

Колку индексни блокови (блокови со покажувачи) се потребни за датотека со големина од 5 KB (притоа 1 KB = 1000 B), не вклучувајќи го и самиот i-node. Одговор: 0 индексни блокови.
Дадотеката има големина 5000 B (5 KB) и може да се смести во 10 директни блокови затоа што секој од нив има големина од 500 B. Чим се сместува во директни, нема потреба од индексни.

Ако не ја собираше датотеката во директните, тогаш ќе беше сместена во индиректните и одговорот ќе беше бројот на индиректни блокови што ќе зафатеше.

Датотеката **/home/teacher/exam.txt** има i-node со број **9701**.

Процесот со PID = 80 го извршува следниот код:

```
int fd1 = open("/home/teacher/exam.txt", O_RDONLY);
read(fd1, buffer1, 150);
int fd2 = open("/home/teacher/exam.txt", O_RDONLY);
read(fd2, buffer2, 250);
```

Процесот со PID = 81 го извршува следниот код:

```
int fd3 = open("/home/teacher/exam.txt", O_RDONLY);
read(fd3, buffer3, 300);
```

Процесот со PID = 90 го извршува следниот код:

```
// create hard link (ln source_file link)
ln /home/teacher/exam.txt  /home/assistant/os.txt
ln /home/teacher/exam.txt  /home/dean/os.txt
```

PID = 80 го отвара фајлот, чита 150 бајти во buffer1, па го отвара пак и чита 250 бајти во buffer2.
mode: r (read) offset: 150
mode: r (read) offset: 250

PID = 81 го отвара фајлот и чита 300 бајти во buffer3.
mode: r (read) offset: 300

PID = 90 прави 2 hard links.
link count = 1 (од самиот file) + 2 (hard link) = 3

Во System File Table – ref. count е секогаш 1, а i-node ptr го најдовме според I-node Table (9701 – B000)

Во I-node Table – ref. count е 3 бидејќи 3 пати е отворен фајлот.

Пополнета ја содржината на дадените табели по извршувањето на кодот од процесите. Ако некои од полињата не се потребни оставете ги празни (или ставете /).

Process Table	
PID = 80	
Index	**Value**
0	/
1	/
...	
7	2000
8	2040
9	/

PID = 81	
Index	**Value**
0	/
1	/
...	
5	2080
6	/
7	/

PID = 90	
Index	**Value**
0	/
1	/
2	/
3	/
4	/
5	/

System File Table		
Address	**Name**	**Value**
2000	mode	read
	offset	150
	ref.count	1
	i-node ptr.	B000
	...	
2040	mode	read
	offset	250
	ref.count	1
	i-node ptr.	B000
	...	
2080	mode	read
	offset	300
	ref.count	1
	i-node ptr.	B000
	...	
20C0	mode	/
	offset	/
	ref.count	/
	i-node ptr.	/

I-node Table		
Address	**Name**	**Value**
B000	number	9701
	link count	3
	ref.count	3
	...	
B400	number	9639
	link count	/
	ref.count	/
	...	
B800	number	6760
	link count	/
	ref.count	/
	...	
BC00	number	8385
	link count	/
	ref.count	/
	...	

Вредноста на fd1, fd2 и fd3 ќе биде: 7 (индекс на 2000), 8 (индекс на 2040) и 5 (индекс на 2080) соодветно.

Именикот **/home/teacher/** ќе го содржи парот име на датотека exam.txt – број на i-node 9701, а именикот **/home/assistant/** ќе го содржи парот os.txt – 9701 .

Доколку PID = 90 го извршуваше кодот **ln -s /home/teacher/exam.txt /home/dean/os.txt** тогаш link count за соодветниот i-node ќе имаше вредност 2 (1 од file + 1 hard link, soft links не се бројат), а именикот **/home/dean/** ќе го содржеше парот os.txt – /home/teacher/exam.txt .

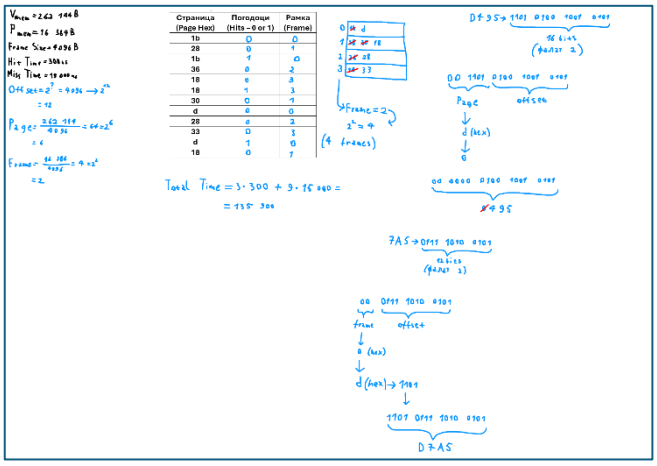
Компјутерски систем има **256 KB** виртуелна меморија. Големината на физичката меморија е **16 KB**, организирана во рамки од **4096 B**. Времето за вчитување на податок при погодок е **300 ns**, а времето потребно за добивање на податок при промашување е **15 μs**.

Форматот на адресите во овој систем е:

Virtual address		Physical address	
Page	Offset	Frame	Offset
6	12	2	12

Нека процесот генерира адреси кои припаѓаат на страниците **1b**, **28**, **1b**, **36**, **18**, **18**, **30**, **d**, **28**, **33**, **d**, **18** и алгоритмот за замена е **Least Recently Used**. Пополнете ја следната табела.

Страница (Page Hex)	**Погодоци (Hits – 0 or 1)**	**Рамка (Frame)**
1b	0	0
28	0	1
1b	1	0
36	0	2
18	0	3
18	1	3
30	0	1
d	0	0
28	0	2
33	0	3
d	1	0
18	0	1



Вкупно време за извршување на мемориските барања на оваа програма е 135 900 ns.

Виртуелната адреса **D495** (hex) се мапира во физичката адреса 495 (hex).

Физичката адреса **7A5** (hex) се мапира во виртуелната адреса D7A5 (hex).

Компјутерски систем има **256 KB** виртуелна меморија. Големината на физичката меморија е **2 KB**, организирана во рамки од **512 B**. Времето за вчитување на податок при погодок е **200 ns**, а времето потребно за добивање на податок при промашување е **25 μs**.

Форматот на адресите во овој систем е:

Virtual address		Physical address	
Page	Offset	Frame	Offset
9	9	2	9

Нека процесот генерира адреси кои припаѓаат на страниците **7d**, **f5**, **85**, **1c6**, **7d**, **85**, **1c6**, **47**, **1fc**, **b3**, **1c6**, **1fc** и алгоритмот за замена е **Least Recently Used**. Пополнете ја следната табела.

Страница (Page Hex)	**Погодоци (Hits – 0 or 1)**	**Рамка (Frame)**
7d	0	0
f5	0	1
85	0	2
1c6	0	3
7d	1	0
85	1	2
1c6	1	3
47	0	1
1fc	0	0
b3	0	2
1c6	1	3
1fc	1	0

Вкупно време за извршување на мемориските барања на оваа програма е 176 000 ns.

Виртуелната адреса **3F88B** (hex) се мапира во физичката адреса 8B (hex).

Физичката адреса **20F** (hex) се мапира во виртуелната адреса 8E0F (hex).

Даден е дел од табелата на страници:

Помошна (Page)	Frame	Read	Modified
0	3	1	0
1	2	1	1
2	1	0	1
3	4	1	0
4	5	0	0
	7	0	1
	6	1	0

Редослед за изфрлање:

Read	Modified
0	0
0	1
1	0
1	1

Страница 4 (рамка 5) има вредност 00, па таа прва ќе се исфрли.

Според табелата, ако се користи **NRU**, прва страница која ќе биде исфрлена ќе биде страницата со број 4 со цел да се ослободи рамката со број 5 .

Еден датотечен систем користи i-nodes структури кои имаат по 10 директни блокови, 3 единечни и 1 двоен индиректен блок. Големината на блокот е 300 B, а адресата на секој блок на дискот е со големина од 10 B. Според тоа:

Максималната датотека која може да се алоцира со помош на директните блокови е 3000 B.

Еден единечен индиректен блок може да адресира вкупно 30 податочни блокови.

Еден двоен индиректен блок може да адресира 900 податочни блокови.

Вкупниот број на блокови кои можат да се адресираат со дадениот i-node изнесува 1000 (10 + 3 * 30 + 1 * 900), па според тоа максималната големина на една датотека е 300 000 (1000 * 300) B.

Нека еден податочен систем има 10 000 блокови. Секој блок има големина од 1000 B и адреса од 4 B. Доколку се користи евиденција за слободни блокови во листа, тогаш еден блок има капацитет да смести информација за 250 слободни блокови, па според тоа, целата листа на слободни блокови би зафаќала 40 блокови. Оваа листа ја менува големината во зависност од зафатеноста на блоковите. Ако се користи бит-мапа, тогаш мапата ќе зафаќа 2 (цел број) блокови. Бит-мапата не ја менува големината во зависност од зафатеноста на блоковите.

Еден датотечен систем користи FAT (File Allocation Table) за складирање на податоците за блоковите. Секој блок има големина од 2 KB. Моменталната состојба на FAT табелата е:

Index	Value
0	4
1	2
2	-1
3	6
4	5
5	7
6	8
7	-1
8	15
9	-1
10	-1
11	14
12	-1
13	-1
14	13
15	12

Содржината на даден именик е:

A	0
B	3
C	1
D	9
E	10
F	11

Напишете ги датотеките во растечки редослед според нивните големини (ако имаат иста големина, подредете ги по азбучен редослед): D E C F A B

Напишете ги големините на овие датотеки: 2 2 4 6 8 10

A: 0 → 4 → 5 → 7 → -1 — 2 KB * 4 стрелки = 8
B: 3 → 6 → 8 → 15 → 12 → -1 — 2 KB * 5 стрелки = 10
C: 1 → 2 → -1 — 2 KB * 2 стрелки = 4
D: 9 → -1 — 2 KB * 1 стрелка = 2
E: 10 → -1 — 2 KB * 1 стрелка = 2
F: 11 → 14 → 13 → -1 — 2 KB * 3 стрелки = 6

---

aaaaa

ми се смачи 😫