ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

ТЕХНИКИ ЗА КРЕИРАЊЕ АЛГОРИТМИ

**АЛГОРИТМИ И
СТРУКТУРИ НА ПОДАТОЦИ**

- предавања -

А
С
П

Содржина на лекцијата

- Вовед во пресметковни проблеми
- Начини на опишување на проблемите
- Псевдокод нотација
- Техники за дизајн на алгоритми
  - Алгоритми базирани на груба сила
  - Алчни алгоритми
  - **Раздели-и-владеј**
  - **Динамичко програмирање**
  - **Алгоритми со случајни броеви**
  - **Останати алгоритми**

Алчни (greedy) алгоритми

- **Проблем**: пакување на ранец (Knapsack algorithm)
  - Нека бидат дадени *n* пакети со големини
    $s_1, s_2, \ldots, s_n, \quad 0 < s_i \leq 1, \quad i = 1..n$
  - Проблемот на пакување го бара оптималното пакување на пакетите во ранци со големина 1 (Треба да се искористи најмал број на ранци)
  - Често пати овој проблем се нарекува и проблем на крадецот кој имал торби кој можеле да носат X килограми злато и златни предмети со различни тежини

Алчни (greedy) алгоритми

0,2 | 0,5 | 0,4 | 0,7

0,1 | 0,3 | 0,8

Проблем: Колку ранци со големина 1 се потребни за пакување на овие пакети?

0,2 | 0,8

0,3 | 0,7

0,1 | 0,4 | 0,5

Алчни (greedy) алгоритми

- Постојат две верзии на алгоритмот:
  - **Прва везрија**: не се знае следната големина на пакетот во низата се додека тековниот пакет не се смести во некој ранец
  - **Втора верзија**: познати се сите големини на пакети однапред

Алчни (greedy) алгоритми

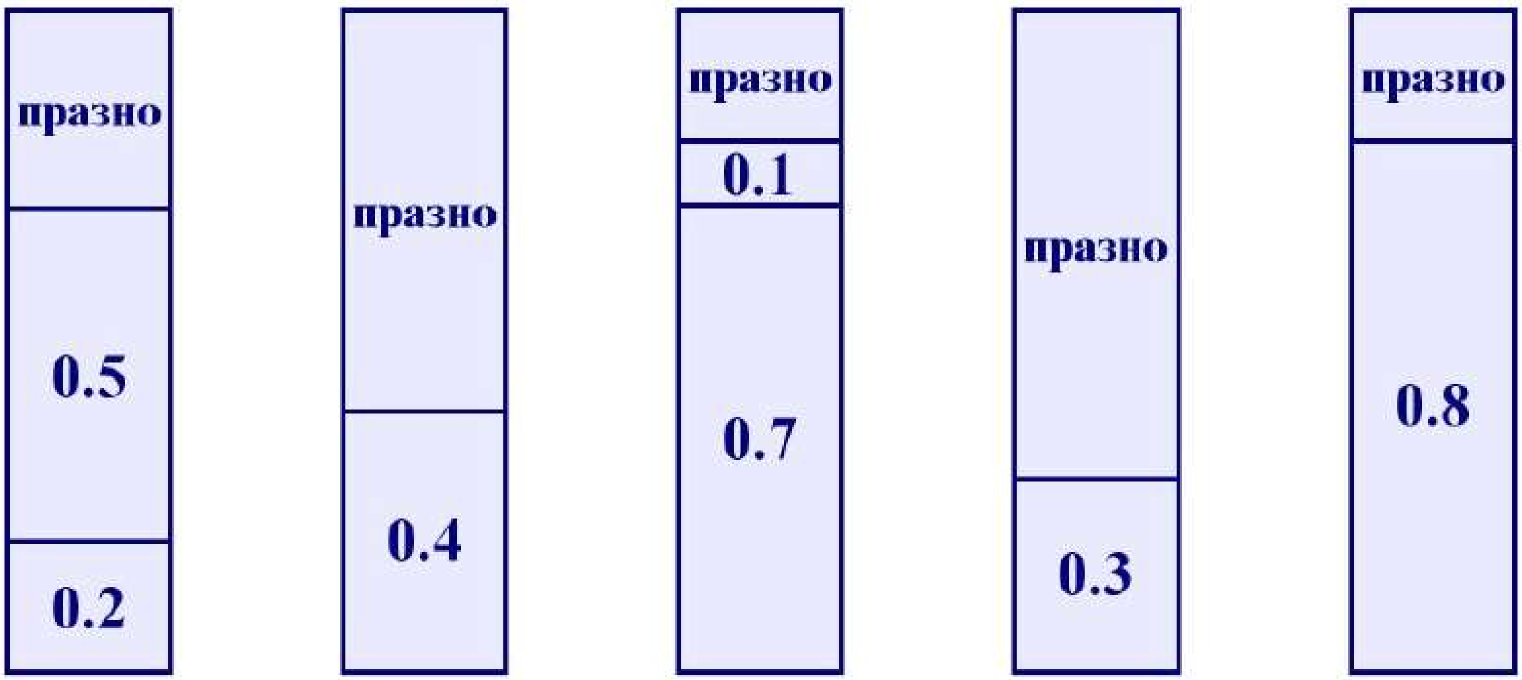
Next fit решение:

- Првиот пакет се сместува во првиот ранец
- При сместување за секој следен пакет се проверува дали има место во **тековно отворениот ранец**
- Доколку нема место, се зема нов ранец во кој што се сместува пакетот

Овој алгоритам очигледно работи во линеарно време но не секогаш дава оптимални резултати

Алчни (greedy) алгоритми

0.2 0.5 0.4 0.7 0.1 0.3 0.8



Во најлош случај овој алгоритам користи двојно повеќе ранци од оптималното решение

Алчни (greedy) алгоритми

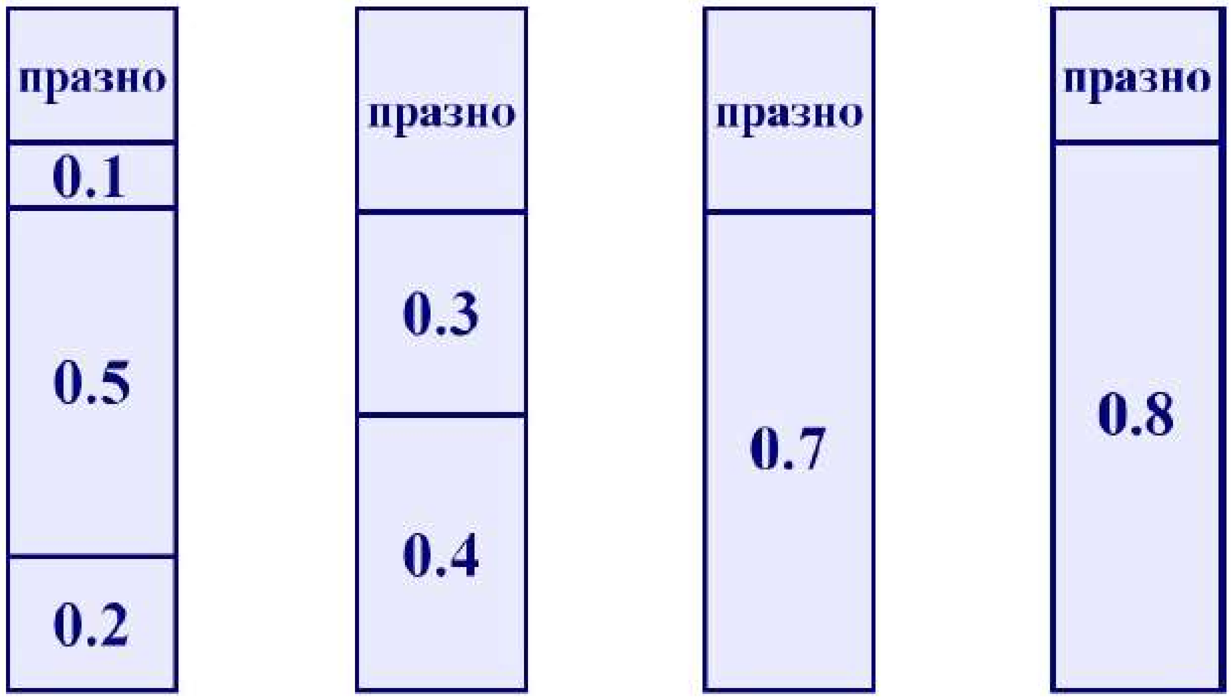
First fit решение:

- Првиот пакет се сместува во првиот ранец
- За секој следен пакет се проверуваат **сите ранци** од почеток, и пакетот се сместува во првиот ранец кој има простор да го собере
- Нов ранец се користи само доколку нема простор во предходно започнатите ранци

Овој алгоритам во најдобар случај може да се реализира со комплексност O(nlogn), а во општ случај O(n²)

Алчни (greedy) алгоритми

0.2 0.5 0.4 0.7 0.1 0.3 0.8



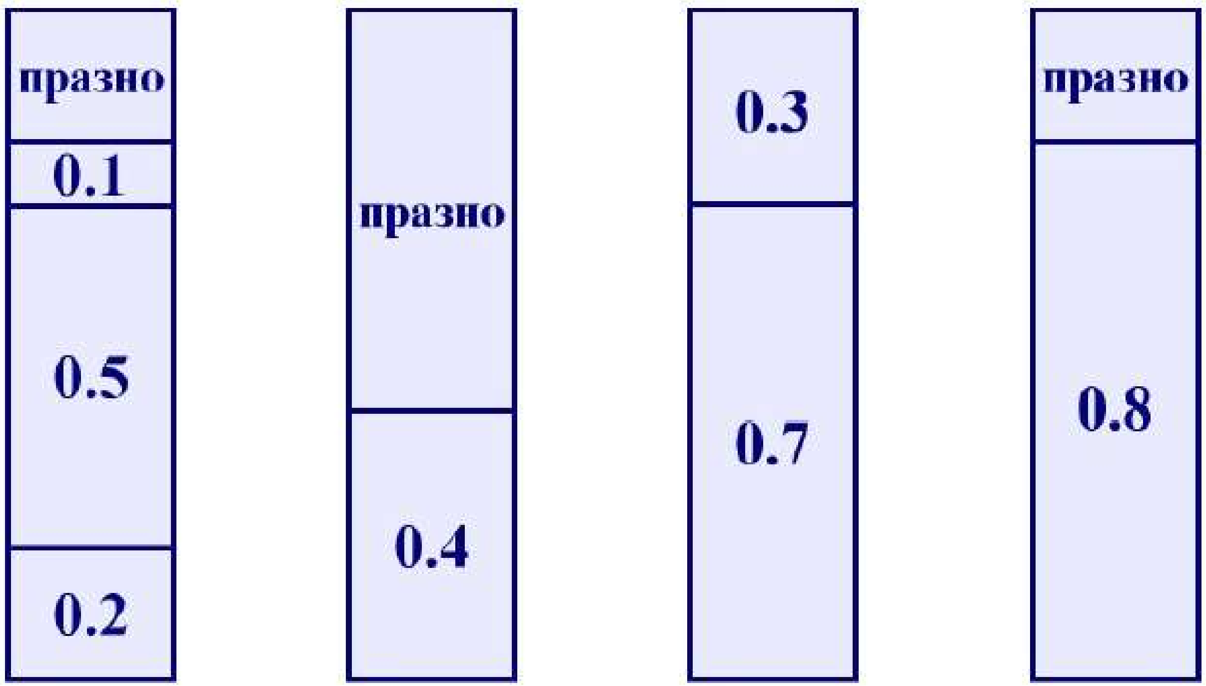
Алчни (greedy) алгоритми

Best fit решение:

- Првиот пакет се сместува во првиот ранец
- За секој следен пакет се проверуваат **сите ранци** од почеток, и пакетот се сместува во ранецот во кој има најмал простор доволен да го собере пакетот со дадена големина
- Нов ранец се користи само доколку нема простор во предходно започнатите ранци

Алчни (greedy) алгоритми

0.2 0.5 0.4 0.7 0.1 0.3 0.8



Раздели и владеј алгоритми

- Во принцип секогаш резултира во коректен алгоритам
- Се состои од два чекора:
  - **Раздели**: Проблемот се дели на помали потпроблеми сé додека не се дојде до елементарни (базични) случаи
  - **Владеј**: Решението на проблемот се добива од решенијата на поедноставните потпроблеми со помош на така наречено „*слевање*" на решението
- Решенијата најчесто се добиваат со користење на **рекурзија**

Раздели и владеј алгоритми

- **Проблем**: Наоѓање на висина на бинарно стебло

Висината на стеблото е за еден поголема од поголемата од висините на левото и десното подстебло на коренот на стеблото

Раздели	Владеј
Подпроблеми: Најди висини на лево и десно подстебло на даден јазел **Елементарен случај**: Висина на лист е еден. Висина на нулта врска е нула	Слевањето на резултатите се состои од наоѓање на поголемата од двете добиени висини и нејзино зголемување за еден.

Раздели и владеј алгоритми

- **Проблем**: Наоѓање на максимален елемент во низа од елементи

Раздели	Владеј
Подпроблеми: Најди најголем елемент во левата и десната половина од низата **Елементарен случај**: Елементот во низа со должина 1 е максимален	Слевањето на резултатите се состои од наоѓање на поголемиот од двата резултати добиени за левата и десната подниза

Раздели и владеј алгоритми

29 14 15 1 6 10 32 12

29 14 15 1   6 10 32 12

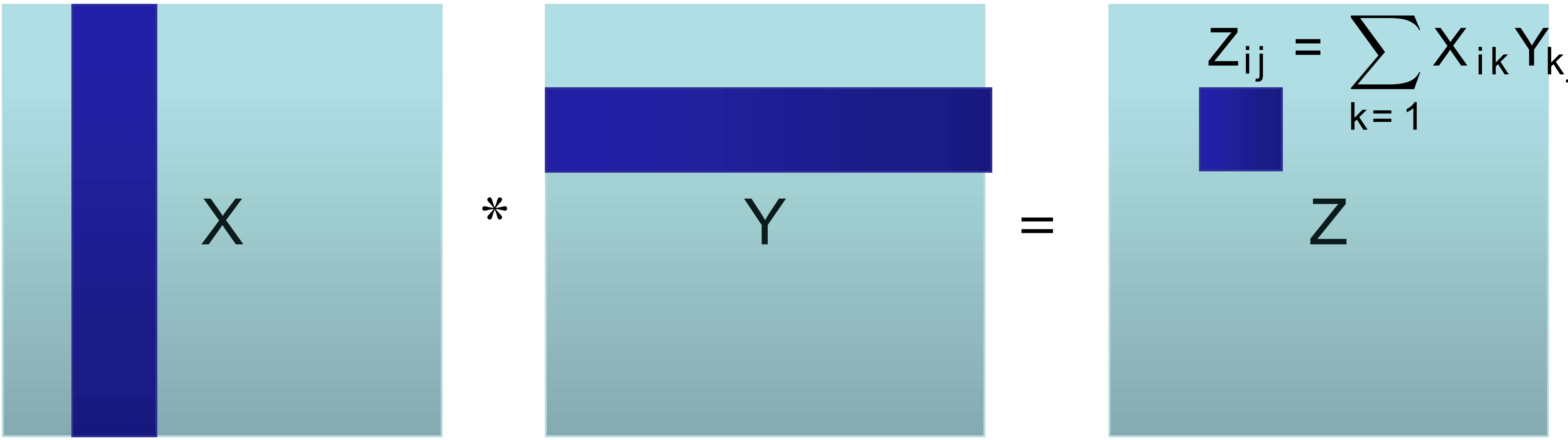
29 14   15 1   6 10   32 12

Раздели и владеј алгоритми

- Класи на проблеми во кои е применлива раздели и владеј парадигмата:
  - **геометриски проблеми** (најблиски точки во рамнина, најмал конвексен полигон кој опфаќа множество точки)
  - **пресметувачки проблеми** (брза фуриева трансформација, множење на матрици)
  - проблеми кои работат со **графови**

Раздели и владеј алгоритми

- Пример: множење на матрици



$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}.$$

Комплексноста на овој пристап е $O(n^3)$

Раздели и владеј алгоритми

$$\underset{X}{\begin{bmatrix} A & B \\ C & D \end{bmatrix}} * \underset{Y}{\begin{bmatrix} E & F \\ G & H \end{bmatrix}} = \underset{Z}{\begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}}$$

Вкупното време на извршување може да се пресмета преку формулата

$$T(n) = 8T(n/2) + O(n^2)$$

Која е комплексноста на овој пристап?

Раздели и владеј алгоритми

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

$$\begin{aligned} P_1 &= A(F - H) \\ P_2 &= (A + B)H \\ P_3 &= (C + D)E \\ P_4 &= D(G - E) \end{aligned}$$

$$\begin{aligned} P_5 &= (A + D)(E + H) \\ P_6 &= (B - D)(G + H) \\ P_7 &= (A - C)(E + F) \end{aligned}$$

Времето на извршување ќе биде: $T(n) = 7T(n/2) + O(n^2)$

Резултантната комплексност ќе биде: $O(n^{\log 7}) \approx O(n^{2.81})$

Раздели и владеј алгоритми

- Псевдокод за множење матрици

MMult(A, B, n)

1. If $n = 1$ Output $A \times B$
2. Else
3. Compute $A^{11}, B^{11}, \ldots, A^{22}, B^{22}$ % by computing $m = n/2$
4. $X_1 \leftarrow MMult(A^{11}, B^{11}, n/2)$
5. $X_2 \leftarrow MMult(A^{12}, B^{21}, n/2)$
6. $X_3 \leftarrow MMult(A^{11}, B^{12}, n/2)$
7. $X_4 \leftarrow MMult(A^{12}, B^{22}, n/2)$
8. $X_5 \leftarrow MMult(A^{21}, B^{11}, n/2)$
9. $X_6 \leftarrow MMult(A^{22}, B^{21}, n/2)$
10. $X_7 \leftarrow MMult(A^{21}, B^{12}, n/2)$
11. $X_8 \leftarrow MMult(A^{22}, B^{22}, n/2)$
12. $C^{11} \leftarrow X_1 + X_2$
13. $C^{12} \leftarrow X_3 + X_4$
14. $C^{21} \leftarrow X_5 + X_6$
15. $C^{22} \leftarrow X_7 + X_8$
16. Output C
17. End If

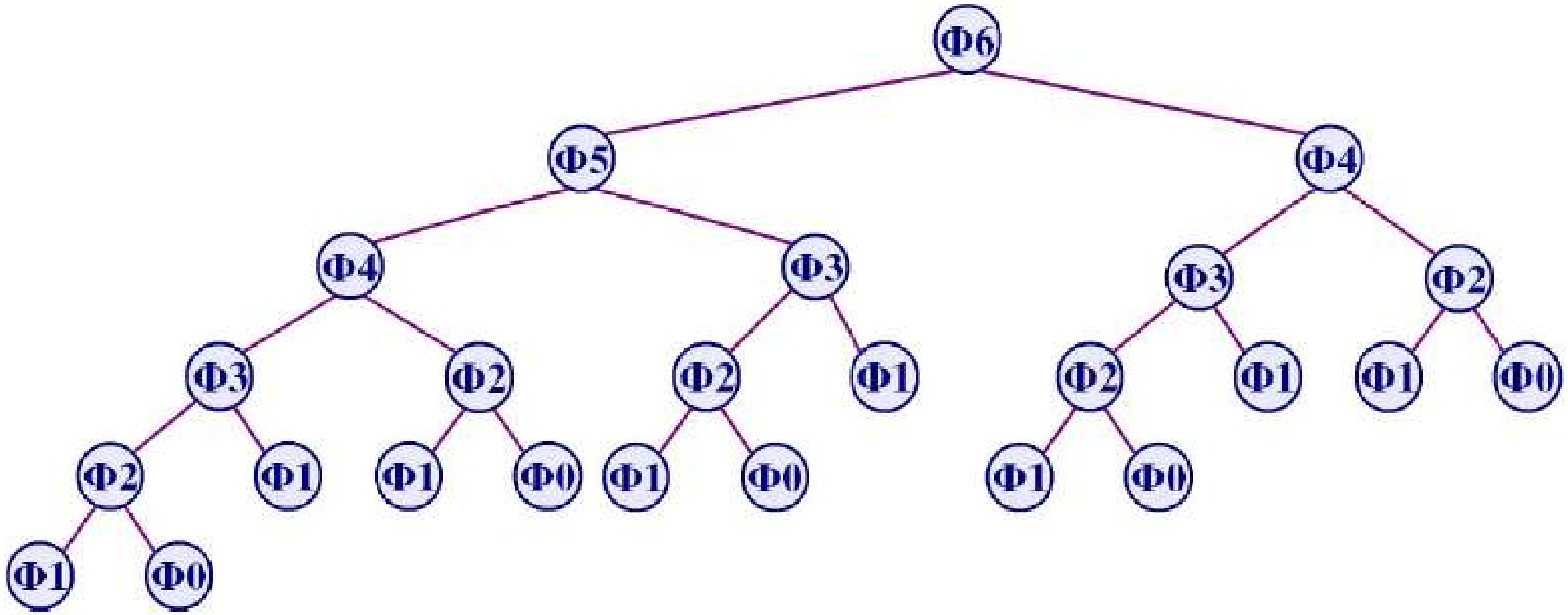
Динамичко програмирање

- Рекурзивното решение на одредени проблеми често може да резултира во неефикасен код, што се должи на повторување на рекурзивните повици

```
FIBONACII(n)
if(n ≤ 1)
    return(1);
else
    return(FIBONACCI(n–1) +FIBONACCI(n–2));
```

Комплексноста на рекурзивниот алгоритам за пресметка на Фибоначиеви броеви е $O(2^n)$

Динамичко програмирање



Очигледно е дека за пресметување на Ф6 дури пет пати се повикува функцијата која го пресметува Ф2

Така, за n=50, $O(n)$ = 1125899906842624

Динамичко програмирање

- Динамичко програмирање се применува кај проблеми (функции) што се **преклопуваат**
- Користи дополнителна меморија за да ги сочувува резултатите од потпроблемите

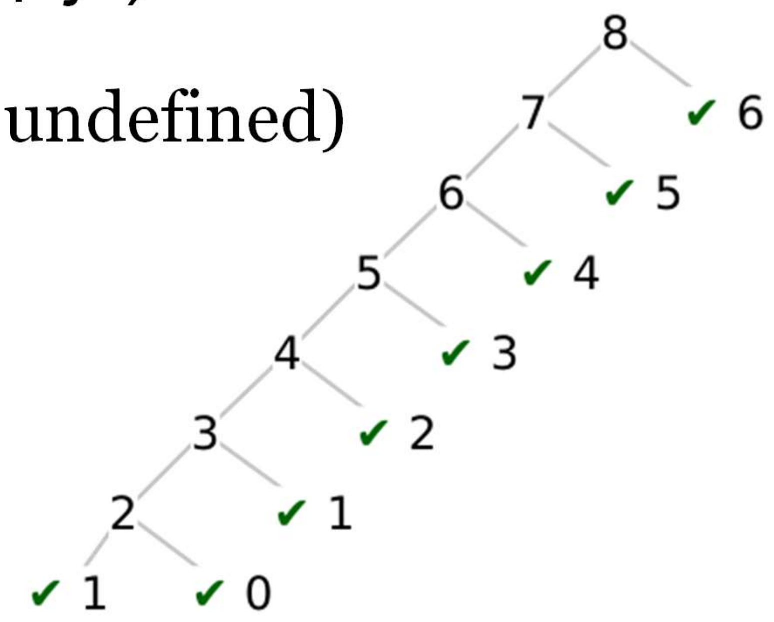
Динамичко програмирање

- Рекурзивен пристап (мемоизација)

$\boldsymbol{F}$ = (1, 1, undefined, undefined, ..., undefined)

...

MEMFIBO(n)
1 **if** (n < 2)
2 **return** (n)
3 **else**
4 **if** F_n is undefined
$F_n \leftarrow$ MEMFIBO(n – 1) + MEMFIBO(n – 2)
5 **return** (F_n)



Комплексноста на ваквата реализација е O(n)

Динамичко програмирање

- Итеративен пристап (мемоизација)

INTERFIBO(n)

$F_0 \leftarrow 1$

$F_1 \leftarrow 1$

for $i \leftarrow 2$ **to** n

$\quad F_i \leftarrow F_{i-1} + F_{i-2}$

return (F_n)

1	1	2	3	5	8	13	21

Комплексноста на ваквата реализација е O(n)

Динамичко програмирање

- Клучот за решавање на задачите со динамичко програмирање е во наоѓање на добра ***состојба*** што ќе се запамети
- Состојбата треба да не зазема многу меморија
- Не треба да има преголем број на состојби, со што значително би се заштедиле мемориските побарувања на алгоритамот

Динамичко програмирање

- Модификација на решението: наместо низа од *n* елементи се користат само две мемориски локации (две променливи):

```
INTERFIBO2(n)
  prev ← 1
  curr ← 1
  for i ← 2 to n
      next ← curr + prev
      prev ← curr
      curr ← next
  return (curr)
```

1	1	2
1	2	3
2	3	5
3	5	8
5	8	13
8	13	21

Дали е м...но уште повеќе да се ... ли комплексноста за ...сметување на *n*-тиот Фибоначиев број?

...же да се стигне до ... на комплексност?

Динамичко програмирање

- Динамичкото програмирање вообичаено се употребува кај проблеми каде што се бара некаква **оптимизација**

- Во ваквите проблеми се можни повеќе решенија

- Секое решение си има своја **цена на чинење**, и ние се стремиме да го пронајдеме решението со најпосакуваната цена на чинење

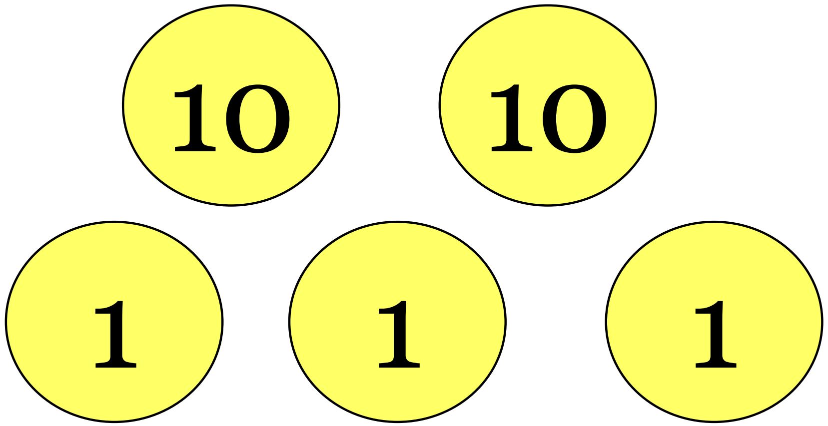
Динамичко програмирање

- **Проблем**: Модифицирана верзија на проблемот за враќање кусур
- Нека ни се дадени следните парички: 1, 5, 8 и 10 денари во неограничена количина
- Да се најде оптималниот број на парички што треба да се вратат во кусурот од 23 денари!

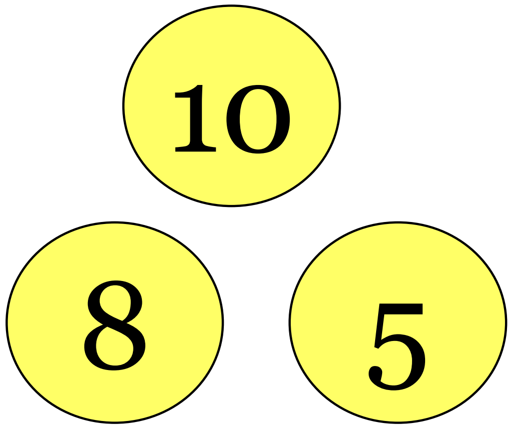
Кое е решението ако се користи алчен пристап?

Динамичко програмирање

Алчен алгоритам



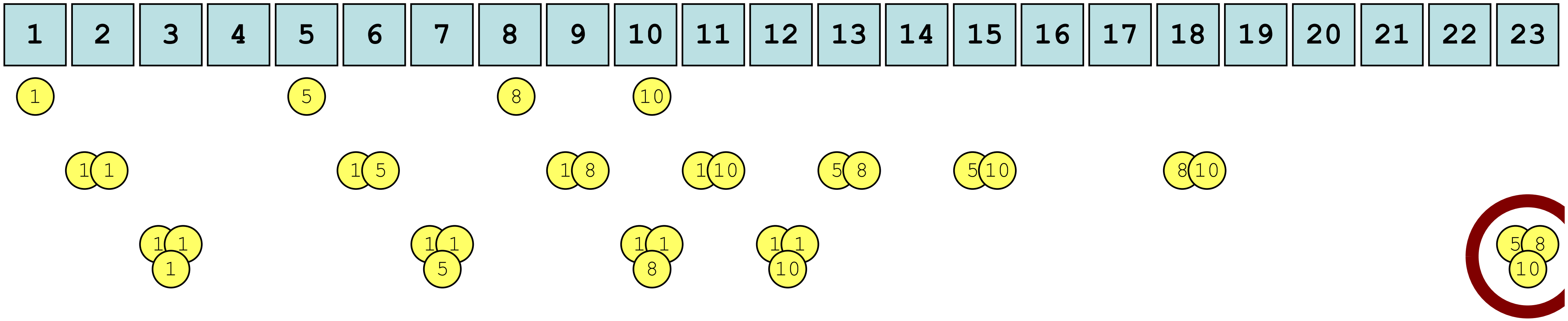
Оптимално решение



Динамичко програмирање

- Која е состојбата што може да се искористи за решавање на проблемот?
- Состојбата е начин да се опише одредена ситуација, која е под-решение за проблемот
- Ако за финална состојба го земеме решението на проблемот, потребно е да можеме да ги најдеме (или генерираме) и сите останати состојби кои би не донеле до финалната состојба

Динамичко програмирање




```
// Initialization
for (i = 0; i < SUMS; ++i)
{    sum[i] = 0;    }

// Starting conditions
for (i = 0; i < COINS; ++i)
{    sum[coins[i]] = 1;    }
```

```
for (i = 0; i < SUMS; ++i)
{
  if (sum[i] == 0)
  {  continue;  }
  for (j = 0; j < COINS; ++j)
  {
    if((sum[i+coins[j]]==0)||(sum[i+coins[j]]>sum[i]+1))
    {
        sum[i+coins[j]] = sum[i] + 1;
    }
  }
}
```

Алгоритми со случајни броеви

- **Проблем**: Проверка на настава преку тестови

  - Професорот може да си дозволи проверка на 50% од материјата преку тестови, без тоа да влијае на времето потребно да се реализира наставата

  - Недостатоци:
    - тестовите се однесуваат само на важните теми
    - најавување на тестовите
    - поставена „шема“ на тестирање

Алгоритми со случајни броеви

- Решение на проблемот:
  - Професорот спрема тестови за секоја тема
  - После секој час, пред студентите фрла паричка. Доколку се падне „глава“, ќе се одржи тестирање
- Дали ќе падне „глава“ или не, е **случаен процес** кој се реализира со генератор на случајни броеви
- Секој програмски јазик има функција за генерирање на случаен број (псевдослучаен број)

Други алгоритми

Алгоритми за враќање наназад од резултатот

- Припаѓаат на типот на алгоритми „груба сила“
- Ги препознаваат патеките кои не водат до резултатот
- На овој начин ги отфрлаат непотребните извршувања и проверки
- Зголемување на ефикасноста на алгоритмот