ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Техники за креирање алгоритми

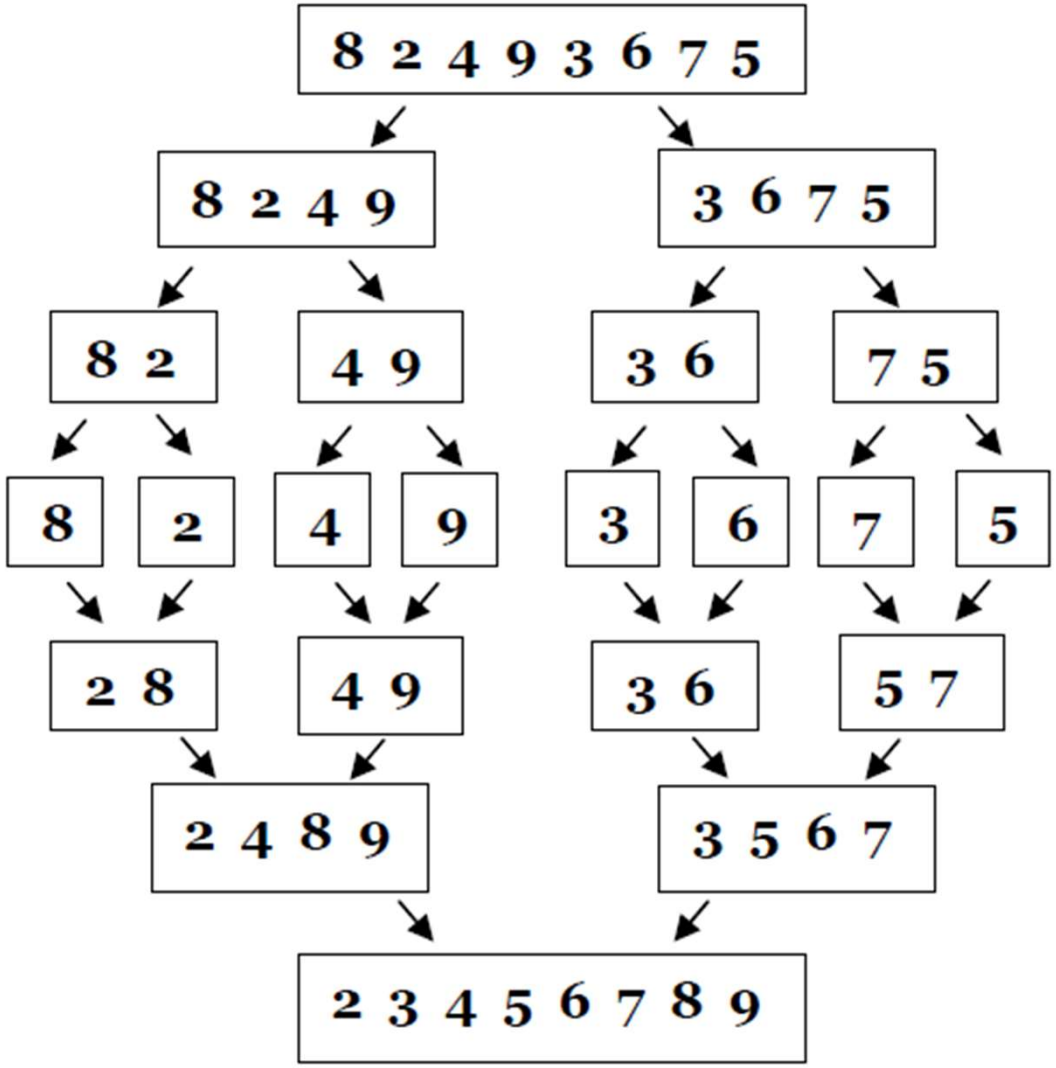
Алгоритми и податочни структури

Аудиториска вежба 4

Сортирање со спојување (merge sort)

- "Раздели и владеј" алгоритамот во три чекори за mergesort сортирањето на некоја низа a[0 … n] е следниот:
  - **Подели**: низата a[0..n] се дели на половина на две поднизи
  - **Владеј**: секоја половина a[0 .. (n/2)] и a[(n/2) +1 .. n] се сортира рекурзивно
  - **Слеј**: двете сортирани половини се спојуваат во една целина

Сортирање со спојување (merge sort) - илустрација



Сортирање со спојување (merge sort) - Java

```
class DivideAndConquer {
    //spojuvanje na dve sortirani nizi [l, mid], [mid+1, r]
    //rezultatot e nova sortirana niza
    void merge(int a[], int l, int mid, int r) {
        int numel = r - l + 1;
        int temp[] = new int[100]; // nova niza za privremeno cuvanje
                                   // na sortiranite elementi
        int i = l, j = mid+1, k = 0;

        while ((i <= mid) && (j <= r)) {
            if (a[i] < a[j]) {
                temp[k] = a[i];
                i++;
            } else {
                temp[k] = a[j];
                j++;
            }
            k++;
        }
```

Сортирање со спојување (merge sort) - Java

```
        while (i <= mid) {
            temp[k] = a[i];
            i++;
            k++;
        }

        while (j <= r) {
            temp[k] = a[j];
            j++;
            k++;
        }

        for (k = 0; k < numel; k++) {
            a[l + k] = temp[k];
        }
    }
```

Сортирање со спојување (merge sort) - Java

```
    void mergesort(int a[], int l, int r) {
        if (l == r) {
            return;
        }

        int mid = (l + r) / 2;
        mergesort(a, l, mid);
        mergesort(a, mid + 1, r);
        merge(a, l, mid, r);
    }
}
```

Сортирање со спојување (merge sort) - Java

```
public static void main(String[] args) {
    int i;

    DivideAndConquer dac = new DivideAndConquer();

    int a[] = new int[]{9, 2, 4, 6, 0, 8, 7, 3, 1, 5};

    dac.mergesort(a, 0, 9);

    for (i = 0; i < 10; i++) {
        System.out.print(a[i] + " ");
    }
    System.out.println();

}
```

Задача 1

- Да се напише функција која ќе го пресмета n-тиот степен на некој број со примена на техниката "Раздели и владеј".

Задача 1 - Java

```
public static void main(String[] args) {
    int i;

    DivideAndConquer dac = new DivideAndConquer();


    System.out.println("pow: " + dac.pow(2, 10));


}
```

Задача 1 - Java

```
class DivideAndConquer {

   int pow(int x, int n) {
        int r;

        if(n == 0)
            return(1);
        else if(n % 2 == 0) {
            r = pow(x, (n/2));
            return r*r;
        } else {
            r = pow(x, (n/2));
            return x*r*r;
        }
    }
}
```

Задача 1 - анализа

- Анализа за комплексноста на алгоритамот:

$T(n) = T(n/2) + O(1)$

$T(n/2) = (T(n/4) + 1) \rightarrow$

$T(n) = T(n/4) + 2$

$T(n/4) = T(n/8) + 1 \rightarrow$

$T(n) = T(n/8) + 3$

…

$T(n) = T(n/2^k)+k$ и ако замениме $k=\log(n)$ се добива:

$T(n) = T(1)+\log(n) = \log(n)$

Значи комплексноста е $O(\log n)$

Задача 1 - анализа

- Решение 2:

```
int pow(int x, int n) {
    int r;

    if(n == 0)
        return(1);
    else if(n % 2 == 0) {
        return pow(x, (n/2))*pow(x,
(n/2));
    } else {
        return x*pow(x, (n/2))*pow(x,
(n/2));
    }
}
```

- Што може да се заклучи?

Задача 1 (решение 2)- анализа

- Анализа за комплексноста на алгоритамот:

$T(n) = 2T(n/2)+O(1)$

$T(n/2) = 2(T(n/4) + 1) \rightarrow$

$T(n) = 4T(n/4)+2$

$T(n/4) = 8T(n/8) + 1 \rightarrow$

$T(n) = 8T(n/8)+3$

…

$T(n) = 2^kT(n/2^k)+k$ и ако замениме $k=\log(n)$ се добива:

$T(n) = nT(1)+\log(n) = n + \log(n)$

Значи комплексноста е $O(n)$

Задача 2.

- Да се пресметаат биномните коефициенти на полином со степен n со помош на Паскаловиот триаголник како што е прикажано:

$$
\begin{array}{ll}
n = 1 & 1 \\
n = 2 & 1\ 2\ 1 \\
n = 3 & 1\ 3\ 3\ 1 \\
n = 4 & 1\ 4\ 6\ 4\ 1 \\
n = 5 & 1\ 5\ 10\ 10\ 5\ 1
\end{array}
$$

Задача 2. - анализа

- Биномниот коефициент C(n, k) претставува број на начини да се избере подмножество со k елементи од множество со n елементи
- Формула за пресметување на биномниот коефициент: C(n, k) = n! / (k! (n - k)!)
- Меѓурезултатите може да направат overflow: C(100, 15) = 253338471349988640 може да се смести во 64-битен long, но бинарната репрезентација на 100! е долга 525 бита
- Паскалово равенство:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Задача 2. - анализа

- Наивна (лоша) имплементација:
  - Не ја пробувајте :)

```
// DO NOT RUN THIS CODE FOR LARGE INPUTS
public static long binomial(int n, int k) {
    if (k == 0)
        return 1;
    if (n == 0)
        return 0;
    return binomial(n-1, k) + binomial(n-1, k-1);
}
```

- Истите подпроблеми се решаваат повеќе пати - експоненцијално време

Задача 2. - Java

```
public static void main(String[] args) throws Exception {
        int i, j;
        BufferedReader br = new BufferedReader(new
   InputStreamReader(System.in));

        DP12 dp = new DP12();

        System.out.println(dp.binomial_coefficient(5, 2));


    }
```

Задача 2. - Java

```
public class DP12 {
int binomial_coefficient(int n, int m) {
    int i, j;
    int bc[][] = new int[n + 1][n + 1];          // tabela so binomni
                                                 // koeficienti

    for (i = 0; i <= n; i++)
        bc[i][0] = 1;

    for (j = 1; j <= n; j++)
        bc[j][j] = 1;

    for (i = 1; i <= n; i++)
        for (j = 1; j <= i; j++)
            bc[i][j] = bc[i - 1][j - 1] + bc[i - 1][j];

    return bc[n][m];
}
}
```

Задача 3.

- *Проблем на максимален збир*: Робот е испратен на Марс, со цел да собере што е можно поголем број од новооткриениот вид на камења. Површината низ која се движи роботот е претставена како правоаголна шема A со димензии m x n, и во секое квадратче е означен бројот на камења кои ги содржи квадратчето, како на сликата.

1 (старт)	...	8
27	...	1
...	...	...
69	...	10 (крај)

Задача 3.

- Притоа роботот тргнува од горното лево квадратче, и треба да стигне до долното десно. Роботот може да се движи само во десно или надолу. Да се напише програма која го дава максималниот број на камења кои може да ги собере роботот.

Задача 2. - анализа

- Како што може да се претпостави, генерирање на сите можни патишта и зачувување на оној пат кој што има најголем збир, не е добра идеја, бидејќи бројот на можни патишта расте со експоненцијална брзина со порастот на големината на правоаголникот. (Се покажува дека е $\frac{(m+n)!}{m!n!}$)

Задача 3. - анализа

- Проблемот има оптимална потструктура:
  - Нека е даден патот P чии што полиња имаат најголем збир. Тогаш секој дел од оптималниот пат со почеток i и крај j содржи максимален број на камења што може да се соберат меѓу тие две полиња (т.е. ги соединува тие две полиња на оптимален начин, во спротивно патот P не би бил оптимален)

Задача 3. - анализа

- Нека B(i, j) е максималниот збир кој што се постигнува од полето (1, 1) до полето (i, j)
- Се бара збирот B(m, n)
- Разгледуваме дали може B(m, n) да се изрази рекурзивно
- Рекурзивната равенка е:

  $B(m, n) = \max\{ B(m, n-1), B(m-1, n) \} + A(m, n)$
- Решавањето на почетниот проблем се сведува на решавање на два потпроблеми од ист тип и нивно едноставно комбинирање

Задача 3. - анализа

- Тривијалните проблеми се наоѓање на елементи од првиот ред и првата колона на матрицата B.
- Потпроблемите кај овој проблем се преклопуваат. На пример проблемот за полето (m-1, n-1) се појавува и кај потпроблемот (m, n-1) и кај (m-1, n). Затоа, за решавање на овој проблем може да се искористи динамичко програмирање со што нема да се пресметува решение за одреден потпроблем повеќе од еднаш.

Задача 3. - Java

```
public static void main(String[] args) throws Exception {
        int i, j;
        BufferedReader br = new BufferedReader(new
    InputStreamReader(System.in));

        DP12 dp = new DP12();
        System.out.println("Vnesi broj na redici: ");
        int m = Integer.parseInt(br.readLine());
        System.out.println("Vnesi broj na koloni: ");
        int n = Integer.parseInt(br.readLine());

        for (i = 0; i < m; i++) {       // vnesuvanje na broj na kamenja vo
    sekoe pole
            System.out.println("Vnesi ja " +(i+1)+ " redica: ");
            for (j = 0; j < n; j++) {
                dp.a[i][j] = Integer.parseInt(br.readLine());
            }
        }
        dp.maksimalen_zbir(m, n);
        System.out.println("Maksimalniot zbir e " + dp.best[m - 1][n - 1]);
     }
```

Задача 3. - Java

```
public class DP12 {
int a[][] = new int[100][100];
int best[][] = new int[100][100];

void maksimalen_zbir(int m, int n) {
    int i, j;
    // inicijalizacija na trivijalni reshenija
    best[0][0] = a[0][0];

    for (i = 1; i < m; i++)
        best[i][0] = best[i - 1][0] + a[i][0]; // prva kolona
    for (i = 1; i < n; i++)
        best[0][i] = best[0][i - 1] + a[0][i]; // prva redica

    for (i = 1; i < m; i++)
      for (j = 1; j < n; j++)
        best[i][j] = Math.max(best[i-1][j], best[i][j-1])+a[i][j];
}
}
```

Задача 3. - варијанти

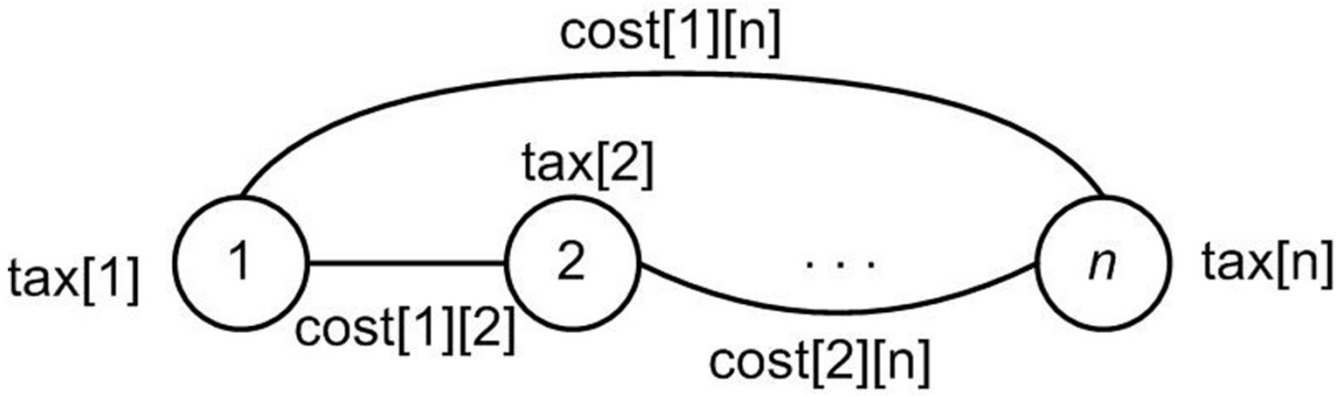
- Варијации на проблемот на максимален збир:
  - Роботот може да се движи и дијагонално десно-долу. Рекурзивната равенка е:

    B(m, n) = max{ B(m, n-1), B(m-1, n), B(m-1, n-1) } + A(m, n)
  - Ако роботот стигнал до квадратче од последниот ред, движејќи се десно-долу може да се телепортира во квадратчето од следната колона и првиот ред (цилиндрична мапа). Така од (m, x) може се телепортира во (1, x+1).

Задача 4.

- *Проблем на најкраток пат*: Еден патник тргнува од градот 1 (најзападно), и треба да стигне до градот *n* (најисточно), патувајќи со авион. Притоа патникот не смее да се враќа назад, т.е. единствена дозволена насока е запад – исток. Постои директен авионски лет од секој град *i* до секој град *j*. Секој град *c* се карактеризира со цена на аеродромската такса која треба да се плати на аеродромот во тој град tax[*c*], а секој лет од *i* до *j* со цената на летот cost[*i*][*j*], како што е прикажано на сликата:

Задача 4.



- Да се напише програма која ќе ја пресмета минималната цена (збир од цените на аеродромските такси и цените на летовите), со која патникот може да стигне од градот 1 до градот *n*.

Задача 4. - анализа

- Испитувањето на сите можни комбинации со кои би летал патникот не е изводливо, бидејќи бројот е експоненцијален (се покажува дека е 2^{n-2}), и многу бргу расте со зголемувањето на n.
- Доколку би се одлучиле на ова решение, програмата ќе дава брзи (за 1 секунда) резултати само доколку n<25).

Задача 4. - анализа

- Нека се познати најевтините вкупни цени од градот 1 до секој од градовите 1, 2, ... , *k*. Дали со користење на овие информации, може да се одреди најевтината цена до градот *k*+1?
- Единствено што треба да испитаме е од кој град $1 \le c \le k$ е најдобро да се патува директно до градот *k*+1 (оптималната цена до *c* е веќе пресметана)
- Исто така познато е и тривијалното решение до градот 1, бидејќи тој е почетната точка, па best[1] = tax[1] (се плаќа само аеродромската такса)
- Рекурзивната формула е:

  $best[k+1] = \min_c\{ best[c]+cost[c][k+1]+tax[k+1] \}, 1 \le c \le k$

Задача 4. - Java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class DP3 {

    static int cost[][];
    static int tax[], best[];
    static int n;
    static int INFINITY = 1000000;

    static int min(int x, int y) {
        if (x < y)
            return x;
        return y;
    }
```

Задача 4. - Java

```
public static void main(String[] args) throws Exception {
        int i, j;
        BufferedReader br = new BufferedReader(new
                                InputStreamReader(System.in));

        System.out.println("Vnesi broj na gradovi:");
        n = Integer.parseInt(br.readLine());

        tax = new int[n];
        best = new int[n];
        cost = new int[n][n];

        for(i = 0; i < n; i++) {
            System.out.println("Vnesi aerodromska taksa za gradot "
                                +(i+1)+ " : ");
            tax[i] = Integer.parseInt(br.readLine());
```

Задача 4. - Java

```
            for(j = i + 1; j < n; j++) {
                System.out.println("Cena na bilet od " +(i+1)+
                                        " do "+(j+1) + " : ");
                cost[i][j] = Integer.parseInt(br.readLine());
            }
        }

        best[0] = tax[0]; // za prviot grad se plakja samo taksa
        for(i = 1; i < n; i++) {
            best[i] = INFINITY; // inicijalizacija

            // go barame opt. grad j, od koj bi patuvale do gradot i
            for(j = 0; j < i; j++)
                best[i] = min(best[i], best[j] + cost[j][i] + tax[i]);
        }

        System.out.println("Najmala cena e "+best[n-1]);
    }
}
```

Задача 5.

- *0-1 knapsack проблем*: Дадени се *n* објекти $O=\{o_1, o_2, o_3, \ldots, o_n\}$ и еден ранец. Секој објект o_i има тежина t_i, а ранецот има капацитет *C*. Ако објектот o_i е сместен во ранецот, тогаш се добива профит од p_i. Целта е да се наполни ранецот така што ќе се максимизира заработениот профит. Бидејќи капацитетот на ранецот е *C*, условот при пополнување е тежината на сите избрани објекти да не го надминува капацитетот на ранецот *C*. Објектите не може да се делат, т.е. објектите или се земаат во ранецот или не.

Задача 5. - анализа

- Индексите на елементите вклучени во ранецот формираат вектор. Нека *i* е индексот на последниот објект од векторот на оптималното решение *S* за капацитет *C*
- Тогаш $S' = S - \{o_i\}$ е оптимално решение на потпроблем за ранец со капацитет $C - t_i$, додека пак профитот на решението *S* е p_i + профитот на потпроблемот

Задача 5. - анализа

- Нека D[*i*][*j*] е максималниот профит за множеството на објекти 1, 2, 3, ..., *i* при капацитет на ранецот *j*. Тогаш:

```
if (j > t[i])
    D[i][j] = max(p[i] + D[i-1][j-t[i]], D[i-1][j])
else
    D[i][j] = D[i-1][j]
```

- Ова кажува дека решението за *i* објекти или го вклучува *i*-тиот објект ако така се остварува поголем профит, или не го вклучува *i*-тиот објект, во кој случај тоа е решение на потпроблем за *i* - 1 објекти и ранец со ист капацитет *j*.

Задача 5. - анализа

- Пример:
  - C=50
  - $(p_1, p_2, p_3) = (60, 100, 120)$
  - $(t_1, t_2, t_3) = (10, 20, 30)$
- Матрицата која ја пополнува алгоритамот базиран на динамичко програмирање е:

i j	0	10	20	30	40	50
0	0	0	0	0	0	0
1	0	60	60	60	60	60
2	0	60	100	160	160	160
3	0	60	100	160	180	220

Задача 5. - Java

```
public class DP5 {

    int max(int a, int b) {
        if (a > b) {
            return a;
        }
        return b;
    }

    public static void main(String[] args) throws
  Exception {
        DP5 dp = new DP5();

        int n = 3;
        int C = 50;
        int p[] = new int[]{60, 100, 120};
        int t[] = new int[]{10, 20, 30};

        System.out.println(dp.DPKnapsack(t, p, C));
    }
```

Задача 5. - Java

```
int DPKnapsack(int t[], int p[], int C) {
    int i, j;
    int n = t.length;
    int D[][] = new int[n + 1][C + 1];

    for (j = 0; j <= C; j++) {
        D[0][j] = 0;
    }

    for (i = 1; i <= n; i++) {
        D[i][0] = 0;
    }

    for (i = 1; i <= n; i++)
        for (j = 1; j <= C; j++)
            if (t[i - 1] <= j)
                D[i][j] = max(p[i-1] + D[i-1][j - t[i-1]], D[i-1][j]);
            else
                D[i][j] = D[i - 1][j];

    return D[n][C];
}
}
```

Задача 5. - дискусија

- Како да се добие множеството на објекти сместени во ранецот кога тој има максимален профит?
  - Со враќање наназад треба да се открие од каде произлегуваат оптималните вредности
  - Ако D[*i*][*p*] = D[*i*-1][*j*] објектот *i* не припаѓа на векторот на решението, и продолжуваме да бараме со D[*i*-1][*j*]
  - Во спротивно објектот *i* припаѓа на векторот на решението, и продолжуваме да бараме со D[*i*-1][$C-j_i$]

Задача 5. - дискусија

- Оптималниот избор на објекти за конкретниот пример е:
  - $S = \{o_2, o_3\}$

i j	0	10	20	30	40	50
0	0	0	0	0	0	0
1	0	60	60	60	60	60
2	0	60	100	160	160	160
3	0	60	100	160	180	220

- За 0-1 knapsack алгоритамот потребно е $\theta(nC)$ време

Задача 5. – (дополнување)

- ЗА ДОМА. Да се напише кодот кој ќе враќа низа од индексите на објектите кои влегуваат во оптималното решение за 0-1 knapsack проблемот.

Задача 6. (за дома)

- Напишете алгоритам кој ќе ја пронајде најдолгата "заедничка" подсеквенца на два стринга x[] и y[] и ќе даде која е нејзината должина.
- Пример
  - најдолга "заедничка" подсеквенца на стринговите: **ggca**cc**acg** и ac**ggc**gg**a**t**acg** е **ggcaacg**.

Задача 6. - анализа

- Нека со NZP[i][j] ја означиме должината на најдолгата "заедничка" подсеквенца на подстринговите x[0] … x[i-1] и y[0] … y[j-1]
- Рекурзивната формула со која може да се опише решението NZP[i][j] е:

$$NZP[i][j] = \begin{cases} 0, \text{ако } i = 0 \text{ или } j = 0 \\ NZP[i-1][j-1] + 1, \text{ако } x[i] = y[j] \\ \max(NZP[i, j-1], NZP[i-1][j]), \text{ако } x[i] \neq y[j] \end{cases}$$

Задача 6. - Java

```
class NajgolemaPodniza{

int max(int a, int b) {
    if (a > b)
        return a;
    return b;
}
```

Задача 6. - Java

```
int najdolgaZaednickaPodsekvencaDolzina(String x, String y) {
    int i, j;
    int N = x.length();
    int M = y.length();

    int NZP[][] = new int[N + 1][M + 1];

    for (i = 0; i < N; i++) {
        NZP[i][0] = 0;
    for (j = 0; j < M; j++) {
        NZP[0][j] = 0;

    for (i = 1; i <= N; i++)
        for (j = 1; j <= M; j++)
            if (x.charAt(i - 1) == y.charAt(j - 1))
                NZP[i][j] = NZP[i - 1][j - 1] + 1;
            else
                NZP[i][j] = max(NZP[i - 1][j], NZP[i][j - 1]);

        return NZP[N][M];
    }
```

Задача 6. - Java

```
String najdolgaZaednickaPodsekvencaString(String x, String y) {
    int i, j;
    int N = x.length();
    int M = y.length();

    int NZP[][] = new int[N + 1][M + 1];

    for (i = 0; i < N; i++)
        NZP[i][0] = 0;

    for (j = 0; j < M; j++)
        NZP[0][j] = 0;

    for (i = 1; i <= N; i++)
        for (j = 1; j <= M; j++)
            if (x.charAt(i - 1) == y.charAt(j - 1))
                NZP[i][j] = NZP[i - 1][j - 1] + 1;
            else
                NZP[i][j] = max(NZP[i - 1][j], NZP[i][j - 1]);

    char rez1[] = new char[max(N, M)];
    int L = 0;
```

Задача 6. - Java

```
    i = N;
    j = M;

    while ((i != 0) && (j != 0)) {
        if (x.charAt(i - 1) == y.charAt(j - 1)) {
            rez1[L] = x.charAt(i - 1);
            L++;
            i--;
            j--;
        } else {
            if (NZP[i][j] == NZP[i - 1][j])
                i--;
            else
                j--;
        }
    }

    String rez2 = "";
    for (i = 0; i < L; i++)
        rez2 += rez1[L - 1 - i];

    return rez2;
}
```

Задача 6. - Java

```
public static void main(String[] args) throws Exception {

    DP6 dp = new DP6();

    String x = "ggcaccacg";
    String y = "acggcggatacg";

    System.out.println(dp.najdolgaZaednickaPodsekvencaDolzina(x, y));
    System.out.println(dp.najdolgaZaednickaPodsekvencaString(x, y));

}
```