Podeli i osvoji
(divide and conquer)

Strukture podataka i algoritmi 2

Algoritamski postupak

- Algoritamski postupak – ideja ili šablon primenljiv na širi spektar algoritamskih problema
- Podeli i osvoji je rekurzivan algoritamski postupak kod kog:
  - Polazni problem se podeli na više manjih nezavisnih podproblema iste vrste
  - Ako je podproblem trivijalan ili “mali” rešava se direktno, inače se rešava rekurzivno (podproblemi se razbijaju na još manje podprobleme)
  - Rešenje polaznog problema dobijamo objedinjavanjem rešenja podproblema
- Najčešće imamo podelu na dva podproblema

Merge sort i quick sort

- Merge sort
  - **Podeli:** podela liste na dve podliste
  - **Osvoji:** spajanje sortiranih podlisti

- Quick sort
  - **Podeli:** podela liste na dve podliste naspram pivota
  - **Osvoji:** kalemljenje sortiranih podlisti

- U oba slučaja trivijalni podproblem je jednoelementna lista ili mala lista koja se sortira elementarnim metodama

Podeli i osvoji

- Opšti postupak zasnovan na ideji “podeli i osvoji” je u pseudo-kodu moguće zapisati na sledeći način

```
Solution solve(Problem p) {
        if (p is an elementary problem)
                solve p directly and return the solution
        else {
                Divide p into independent problems p1 and p2
                Solution s1 = solve(p1);
                Solution s2 = solve(p2);
                merge s1 and s2 into s where s is solution of p;
                return s;
        }
}
```

"Podeli i osvoji" teorema

- Neka se polazni problem veličine n deli na dva podproblema veličine $n/2$ i neka se trivijalni podproblem rešava u konstantnom vremenu.
- Tada se vremenska složenost celog postupka može izraziti sledećom rekurentnom relacijom

  $T(n) = 2T(n/2) + f(n)$,

  gde je $f(n)$ vreme potrebno za podelu problema i objedinjavanje rešenja.

- **Teorema**
  - Ako je f(n) u klasi O(1) tada je T(n) u klasi O(n)
  - Ako je $f(n)$ u klasi $O(n)$ tada je $T(n)$ u klasi $O(n \log n)$.
  - Ako je $f(n)$ u klasi $O(n^2)$ tada je $T(n)$ u klasi $O(n^2)$
- **Dokaz.** Direktna posledica **master teoreme**.

Master teorema

- Neka je data rekurentna relacija

  $T(n) = aT(n/b) + f(n)$, gde je $f(n)$ u $O(n^d)$

  $T(1)$ je u $O(1)$

  gde su a, b i d konstante takve da $a > 0$, $b > 1$ i $d \geq 0$
- Tada je
  - $T(n)$ u $O(n^d)$ ako je $d > \log_b a$
  - $T(n)$ u $O(n^d \log n)$ ako je $d = \log_b a$
  - $T(n)$ u $O(n \wedge \log_b a)$ ako je $d < \log_b a$

- Za $T(n) = 2T(n/2) + f(n)$ imamo $a = b = 2 \rightarrow \log_b a = 1$
  - Ako je $f(n)$ u $O(1)$ tada $d = 0 < \log_b a \rightarrow T(n)$ u $O(n)$
  - Ako je $f(n)$ u $O(n)$ tada $d = 1 = \log_b a \rightarrow T(n)$ u $O(n \log n)$
  - Ako je $f(n)$ u $O(n^2)$ tada $d = 2 > \log_b a \rightarrow T(n)$ u $O(n^2)$

Binarno pretraživanje i stepenovanje uzastopnim kvadriranjem

- Binarno pretraživanje i stepenovanje uzastopnim kvadriranjem su takođe algoritmi iz klase podeli i osvoji
- Kod oba algoritma početni problem veličine *n* redukujemo na **jedan** podproblem veličine *n*/2
  - Rešenje polaznog problema (osvajanje) se trivijalno dobija iz rešenja redukovanog problema
- Za oba algoritma imamo da je

  $T(n) = T(n/2) + f(n)$, gde je $f(n)$ u $O(1)$
- Primenom master teoreme dobijamo
  - $a = 1$, $b = 2$ → $\log_b a = 0$
  - $d = 0 = \log_b a$ → $T(n)$ je u $O(n^d \log n)$
  - Drugim rečima $T(n)$ je u $O(\log n)$ jer je $n^0 = 1$

Množenje kvadratnih matrica

- Neka su A i B dve kvadratne matrice dimenzije n. Množenje matrica po definiciji je u $O(n^3)$
- Podeli i osvoji pristup: Podelimo A i B u blokove dimenzije n/2

$$AB = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

- Štrasen je pokazao da se blokovi matrice C mogu dobiti iz blokova matrica A i B uz 7 množenja i 16 sabiranja
- **$T(n) = 7T(n/2) + f(n)$ gde je $f(n) = 16(n/2)^2 \rightarrow f(n)$ u $O(n^2)$**
- Primenom master teoreme dobijamo
  - $a = 7$, $b = 2$, $\log_b a = 2.807$
  - $d = 2 \rightarrow d < \log_b a \rightarrow T(n)$ u $O(n \wedge \log_b a) = O(n^{2.807})$
- Blokove trivijalno dobijamo iz 8 množenja i 4 sabiranja, no tada imamo da je $T(n)$ u $O(n^3)$

Najkraća distanca u skupu 2D tačaka

- Neka je S niz tačaka u ravni
- Jedan od osnovnih problema računarske geometrije: odrediti najkraće rastojanje (distancu) između dve tačke niza S
- Naivan pristup je u $O(n^2)$ – računamo distancu između svake dve tačke i tražimo minimalnu

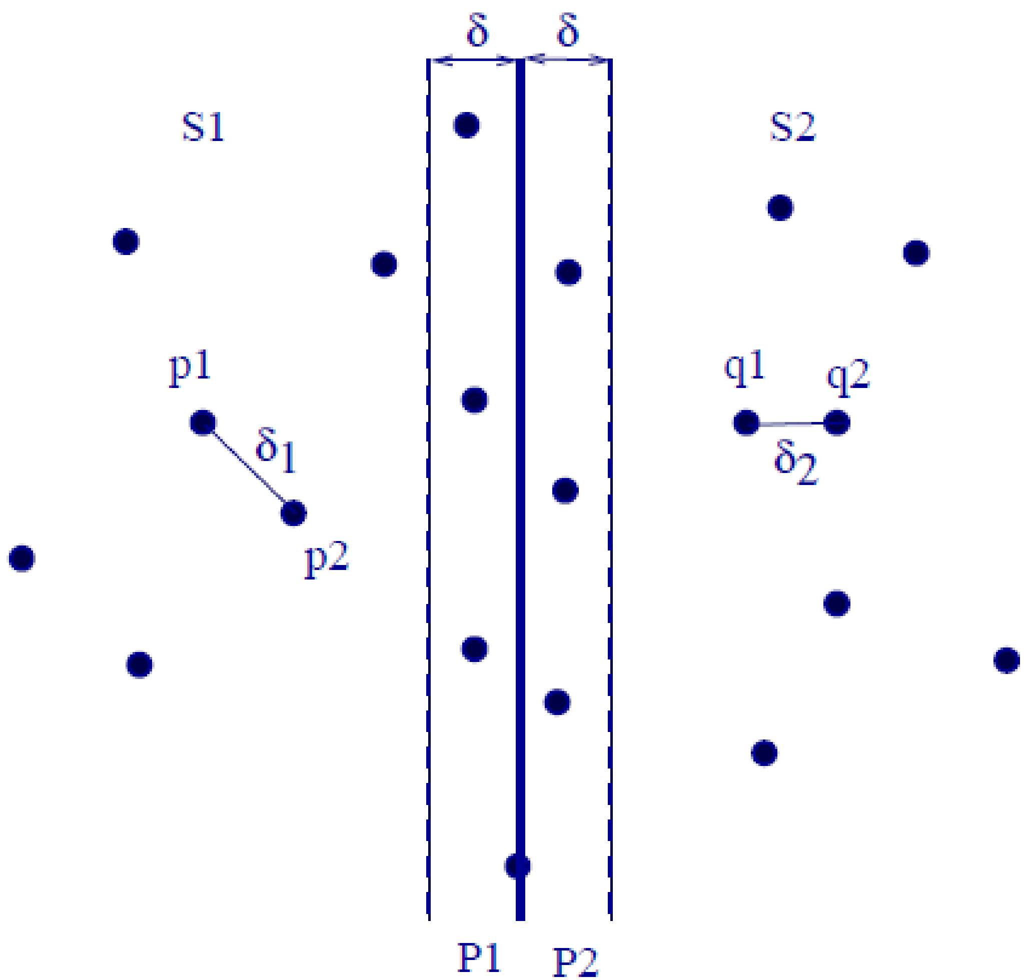
```
double minDistance = Double.MAX_VALUE;
for (int j = 1; j < S.length; j++) {
    for (int i = 0; i < j; i++) {
        double dx2 = (S[i].x – S[j].x) * (S[i].x – S[j].x);
        double dy2 = (S[i].y – s[j].y) * (S[i].y – s[j].y);
        double dist = Math.sqrt(dx2 + dy2);
        if (dist < minDistance)
            minDistance = dist;
    }
}
```

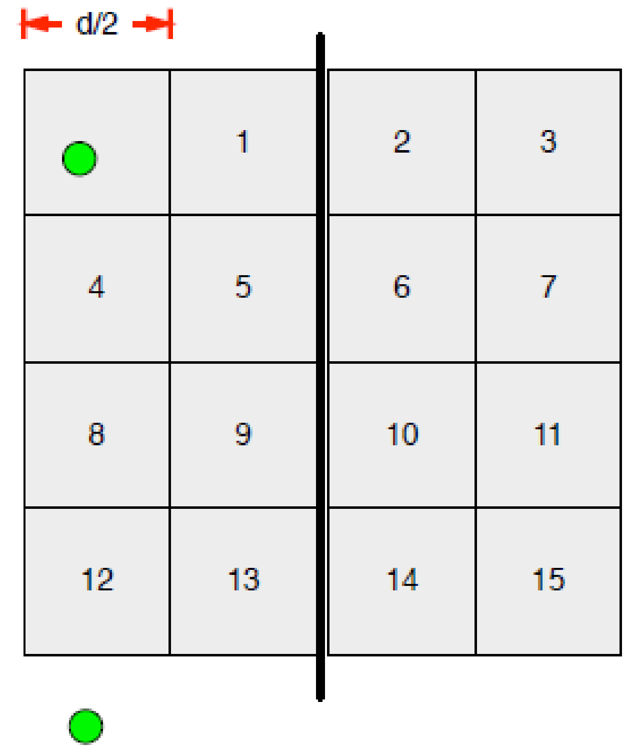
Podeli i osvoji pristup

- “Podelimo” niz S na dva dela
  - $S_1 = S[0 .. n/2]$
  - $S_2 = S[n/2 + 1 .. n – 1]$, $n = S.length$
- m_1 = minimalna distanca u S_1
- m_2 = minimalna distanca u S_2
- m_3 = minimalna distanca između tačaka A i B,
  A je iz S_1, B je iz S2
- Minimalna distanca = $\min\{m_1, m_2, m_3\}$
- **Postavlja se pitanje kako naći m_3?**
- Ako bi smo računali rastojanje između svake dve tačke gde je prva tačka iz A, a druga tačka iz B, tada bi imali
  $T(n) = 2T(n/2) + f(n)$, gde je $f(n)$ iz $O(n^2)$
  → **$T(n)$ je u $O(n^2)$**

Pametniji podeli i osvoji pristup

- Sortiramo S po *x* koordinati pa onda izvršimo podelu na S_1 i S_2





Pametan podeli i osvoji pristup

- Sortiramo S po *x* kordinati pa onda izvršimo podelu na S_1 i S_2
- m_1 = minimalna distanca u S_1
- m_2 = minimalna distanca u S_2
- delta = min(m_1, m_2)
- **Napravimo niz C u kome su sadržane sve tačke koje su od tačke S[n/2] po x koordinati udaljene manje od delta**
- Sortiramo niz C po y koordinati
- Tada za svaku tačku iz C **najviše 15 sledećih tačaka** može biti na rastojanju manje od delta.

Kompleksnost pristupa

- $T(n) = 2T(n / 2) + f(n)$
- $f(n)$ je komponovan od sledećih operacija
  - formiranje niza C – $O(n)$
  - sortiranje niza C po y koordinati – $O(n \log n)$
  - računanje distanci za tačke iz niza C – $O(1)$
    - distance računamo za najviše 15 sledećih tačaka
- Stoga je $f(n)$ u $O(n \log n)$
- $T(n)$ je tada u $O(n \log^2 n)$
- Početno sortiranje po x koordinati je u $O(n \log n)$
- Stoga je vremenska složenost algoritma $O(n \log^2 n)$

Point2D – klasa koja opisuje jednu tačku

```java
public class Point2D {
	public double x, y;

	public Point2D(double x, double y) {
		this.x = x;
		this.y = y;
	}

	public boolean coincident(Point2D p) {
		return x == p.x && y == p.y;
	}

	public double dist(Point2D p) {
		double x2 = (x - p.x) * (x - p.x);
		double y2 = (y - p.y) * (y - p.y);
		return Math.sqrt(x2 + y2);
	}

	public String toString() {
		return "(" + x + ", " + y + ")";
	}
}
```

```java
import java.util.Arrays;
import java.util.Comparator;

public class MinDistance2D {

	private Point2D[] points;

	public MinDistance2D(Point2D[] points) {
		if (points.length < 2) {
			throw new IllegalArgumentException("Bar dve tacke");
		}

		this.points = points;
	}

	private static class XComparator implements Comparator<Point2D> {
		public int compare(Point2D arg0, Point2D arg1) {
			double d = arg0.x - arg1.x;
			if (d < 0) return -1;
			else if (d > 0) return 1;
			else return 0;
		}
	}

	private static class YComparator implements Comparator<Point2D> {
		public int compare(Point2D arg0, Point2D arg1) {
			double d = arg0.y - arg1.y;
			if (d < 0) return -1;
			else if (d > 0) return 1;
			else return 0;
		}
	}
	//... to be continued
}
```

```java
public double findDQ() {
	Arrays.sort(points, new XComparator());

	for (int i = 0; i < points.length - 1; i++) {
		if (points[i].coincident(points[i + 1]))
			return 0.0;
	}

	return findDQ(0, points.length - 1);
}

private double findDQ(int from, int to) {
	if (to == from) {
		return Double.MAX_VALUE;
	}
	if (to - from == 1) {
		return points[from].dist(points[to]);
	}

	int median = (from + to) / 2;
	double dl = findDQ(from, median);
	double dr = findDQ(median + 1, to);
	double minDist = Math.min(dl, dr);
	double minCrossing = findMinCrossing(from, to, median, minDist);
	return Math.min(minDist, minCrossing);
}

//... to be continued
```

```java
private double findMinCrossing(int from, int to, int median, double delta) {
	double medianX = points[median].x;
	Point2D[] box = new Point2D[points.length];
	int boxCnt = 0;
	for (int i = from; i <= to; i++) {
		if (Math.abs(points[i].x - medianX) < delta)
			box[boxCnt++] = points[i];
	}

	Arrays.sort(box, 0, boxCnt, new YComparator());

	double minDist = Double.MAX_VALUE;
	for (int i = 0; i < boxCnt - 1; i++) {
		Point2D current = box[i];

		// ova ce se petlja izvrsiti najvise 15 puta
		for (int j = i + 1; j < boxCnt; j++) {
			if (box[j].y - current.y >= delta)
				break;

			double d = current.dist(box[j]);
			if (d < minDist)
				minDist = d;
		}
	}

	return minDist;
}
```